Blazor & Maui

Juan Carlos Zuluaga

2023, Semestre 1

Indice

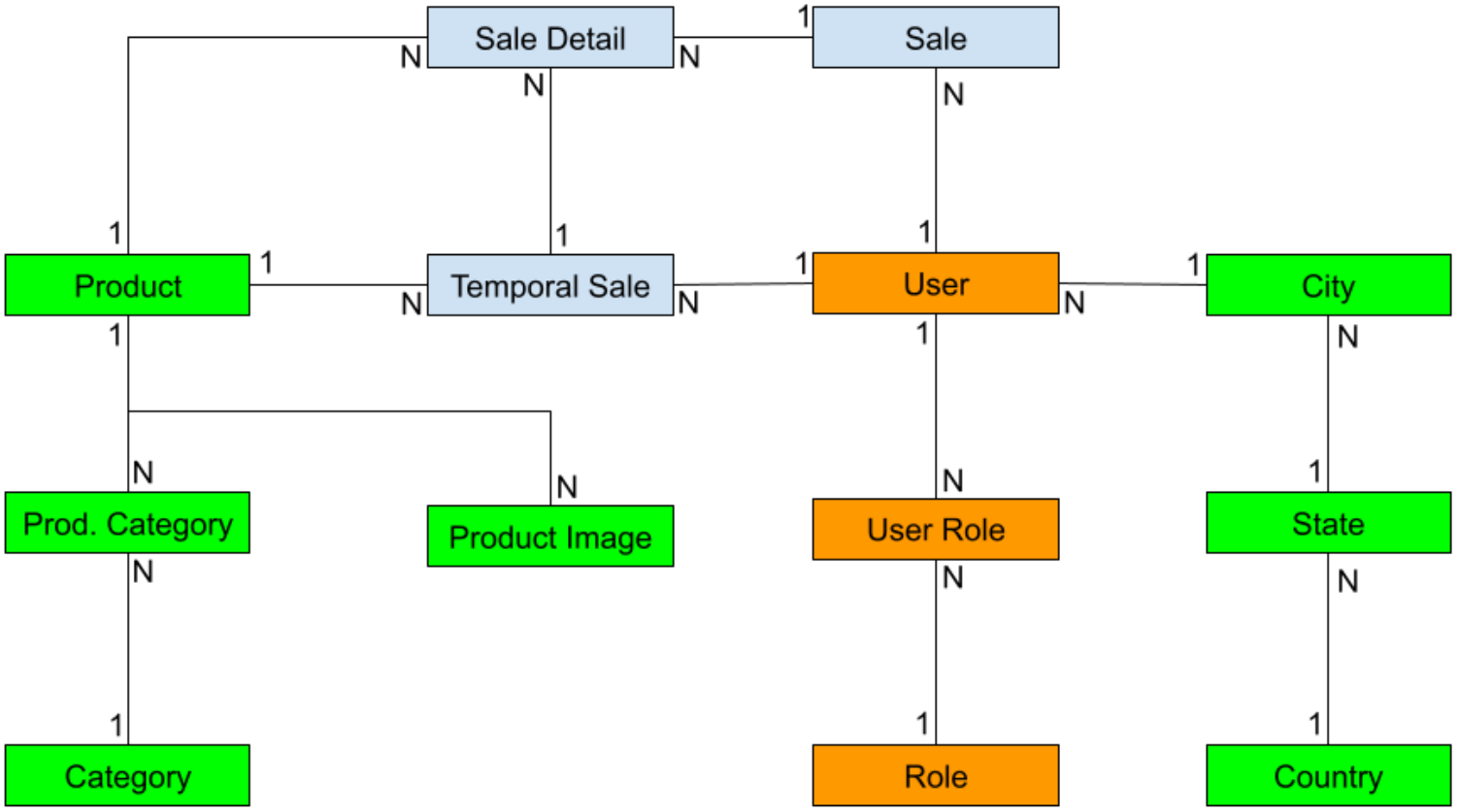
PARTE I - WEB & API

Matriz de funcionalidad

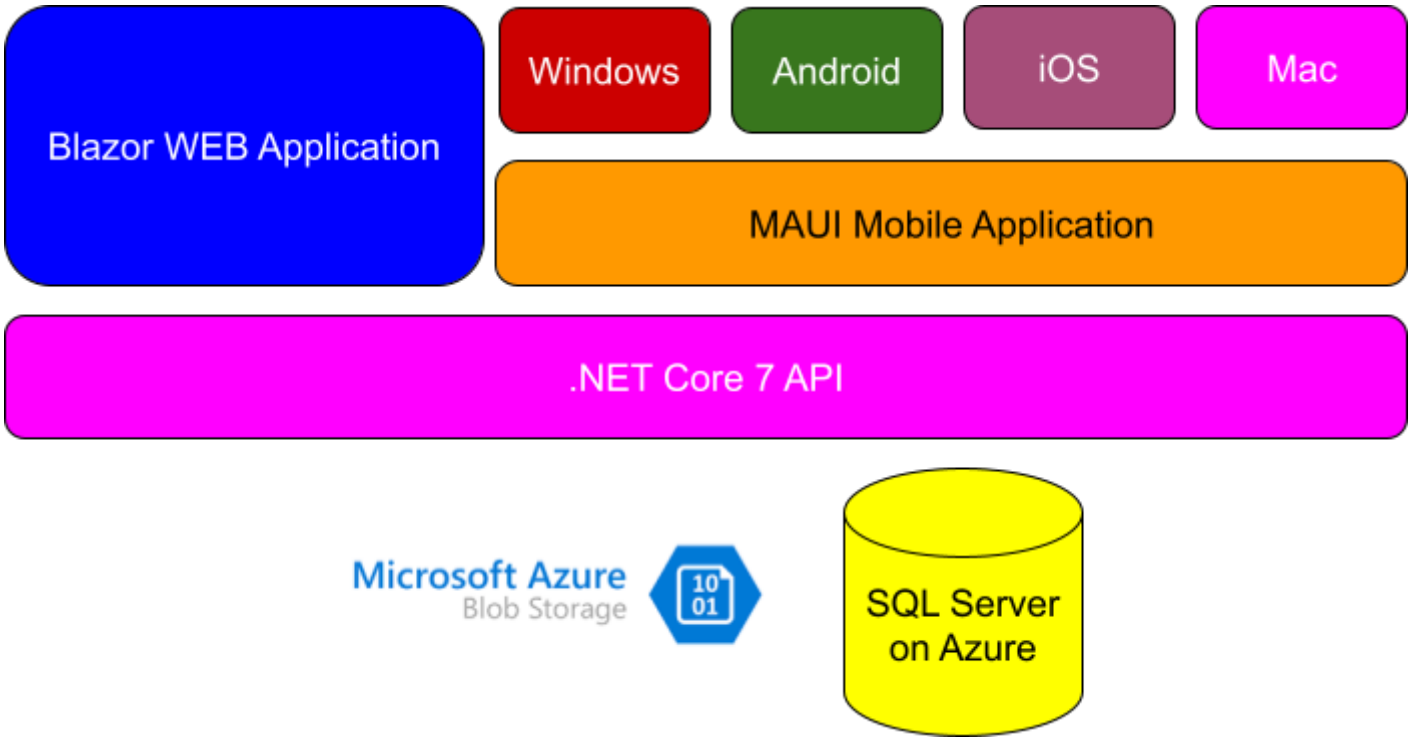
Funcionalidad	Administrador	Usuario	Anónimo	WEB	Mobile
Ingresar al sistema con email y contraseña	X	X		X	X
Editar datos de usuario (incluyendo foto)	X	X		X	X
Cambiar contraseña	X	X		X	X
Recuperar contraseña, si el usuario olvida la contraseña se le enviará un correo con un token para poder recuperar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

Diagrama Entidad Relación

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



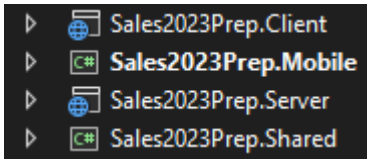
Estructura básica de proyecto



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

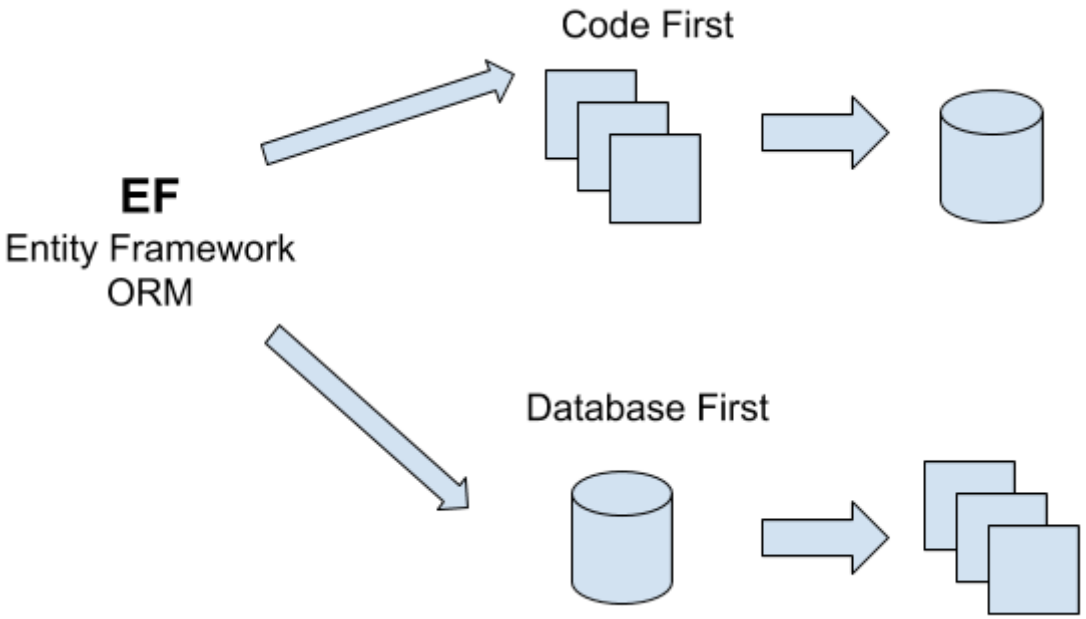
- Una solución en blanco llamada **Sales**.
- A la solución le agregamos un proyecto tipo: **Class Library**, llamado **Sales.Shared**.
- A la solución le agregamos un proyecto tipo: **ASP.NET Core Web API**, llamado **Sales.API**.
- A la solución le agregamos un proyecto tipo: **Blazor WebAssembly App**,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

Debe quedar algo como esto:



Hacemos el primer commit en nuestro repositorio.

Crear la BD con EF



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

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

```
using System.ComponentModel.DataAnnotations;

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

        [Display(Name = "País")]
        [MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]
        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
```

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

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

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

namespace Sales.API.Data
{
    public class DataContext : DbContext
    {
        public DataContext(DbContextOptions<DataContext> options) : base(options)
        {
        }
        public DbSet<Country> Countries { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Country>().HasIndex(c => c.Name).IsUnique();
        }
    }
}
```

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

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

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

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

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

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

```
builder.Services.AddSwaggerGen();
```

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


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

6. Correr los comandos:

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

7. Hacemos nuestro segundo **Commit**.

Creando los primeros métodos en el primer controlador

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

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

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

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

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

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

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

```
app.MapControllers();
```

```
app.UseCors(x => x
    .AllowAnyMethod()
    .AllowAnyHeader()
    .SetIsOriginAllowed(origin => true)
    .AllowCredentials());

app.Run();
```

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

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

5. Hacemos el **commit** de lo que llevamos.

Creando nuestros primeros componentes en Blazor

6. Ahora vamos listar y crear países por la interfaz WEB. Primero configuramos en el proyecto **WEB** la dirección por la cual sale nuestra **API**:

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

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

```
using System.Net;

namespace Web.Repositories
{
    public class HttpResponseWrapper<T>
    {
        public HttpResponseWrapper(T? response, bool error, HttpResponseMessage httpResponseMessage)
        {
            Error = error;
            Response = response;
            HttpResponseMessage = httpResponseMessage;
        }

        public bool Error { get; set; }

        public T? Response { get; set; }

        public HttpResponseMessage HttpResponseMessage { get; set; }

        public async Task<string?> GetErrorMessageAsync()
        {
            if (!Error)
            {
                return null;
            }

            var codigoEstatus = HttpResponseMessage.StatusCode;
            if (codigoEstatus == HttpStatusCode.NotFound)
            {
                return "Recurso no encontrado";
```

```
        }
        else if (codigoEstatus == HttpStatusCode.BadRequest)
        {
            return await HttpResponseMessage.Content.ReadAsStringAsync();
        }
        else if (codigoEstatus == HttpStatusCode.Unauthorized)
        {
            return "Tienes que logearte para hacer esta operación";
        }
        else if (codigoEstatus == HttpStatusCode.Forbidden)
        {
            return "No tienes permisos para hacer esta operación";
        }

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

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

```
namespace Web.Repositories
{
  public interface IRepository
  {
    Task<HttpResponseWrapper<T>> Get<T>(string url);

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

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

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

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

namespace Sales.WEB.Repositories
{
  public class Repository : IRepository
  {
    private readonly HttpClient _httpClient;

    private JsonSerializerOptions _jsonDefaultOptions => new JsonSerializerOptions
    {
      PropertyNameCaseInsensitive = true,
    };

    public Repository(HttpClient httpClient)
    {
      _httpClient = httpClient;
    }
```

```
    public async Task<HttpResponseWrapper<T>> Get<T>(string url)
    {
        var responseHttp = await _httpClient.GetAsync(url);
        if (responseHttp.IsSuccessStatusCode)
        {
            var response = await UnserializeAnswer<T>(responseHttp, _jsonDefaultOptions);
            return new HttpResponseWrapper<T>(response, false, responseHttp);
        }

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

    public async Task<HttpResponseWrapper<object>> Post<T>(string url, T model)
    {
        var mesageJSON = JsonSerializer.Serialize(model);
        var messageContet = new StringContent(mesageJSON, Encoding.UTF8, "application/json");
        var responseHttp = await _httpClient.PostAsync(url, messageContet);
        return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
    }

    public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T model)
    {
        var messageJSON = JsonSerializer.Serialize(model);
        var messageContet = new StringContent(messageJSON, Encoding.UTF8, "application/json");
        var responseHttp = await _httpClient.PostAsync(url, messageContet);
        if (responseHttp.IsSuccessStatusCode)
        {
            var response = await UnserializeAnswer<TResponse>(responseHttp, _jsonDefaultOptions);
            return new HttpResponseWrapper<TResponse>(response, false, responseHttp);
        }
        return new HttpResponseWrapper<TResponse>(default, !responseHttp.IsSuccessStatusCode, responseHttp);
    }

    private async Task<T> UnserializeAnswer<T>(HttpResponseMessage httpResponse, JsonSerializerOptions
jsonSerializerOptions)
    {
        var respuestaString = await httpResponse.Content.ReadAsStringAsync();
        return JsonSerializer.Deserialize<T>(respuestaString, jsonSerializerOptions)!;
    }
  }
}
```


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

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

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

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

```
@typeparam Titem

@if(MyList is null)
```

```
{
    @if(Loading is null)
    {
        <div class="align-items-center">
            <img src="https://upload.wikimedia.org/wikipedia/commons/b/b1/Loading_icon.gif?20151024034921" />
        </div>
    }
    else
    {
        @Loading
    }
}
else if(MyList.Count == 0)
{
    @if(NoRecords is null)
    {
        <p>No hay registros para mostrar...</p>
    }
    else
    {
        @NoRecords
    }
}
else
{
    @Body
}


@code {
    [Parameter]
    public RenderFragment? Loading { get; set; }

    [Parameter]
    public RenderFragment? NoRecords { get; set; }

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

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

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

```
@page "/countries"
@inject IRepository repository

<h3>Paises</h3>

<div class="mb-3">
```

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

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

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

    protected async override Task OnInitializedAsync()
    {
        var responseHppt = await repository.Get<List<Country>>("api/countries");
        Countries = responseHppt.Response!;
    }
}
```

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

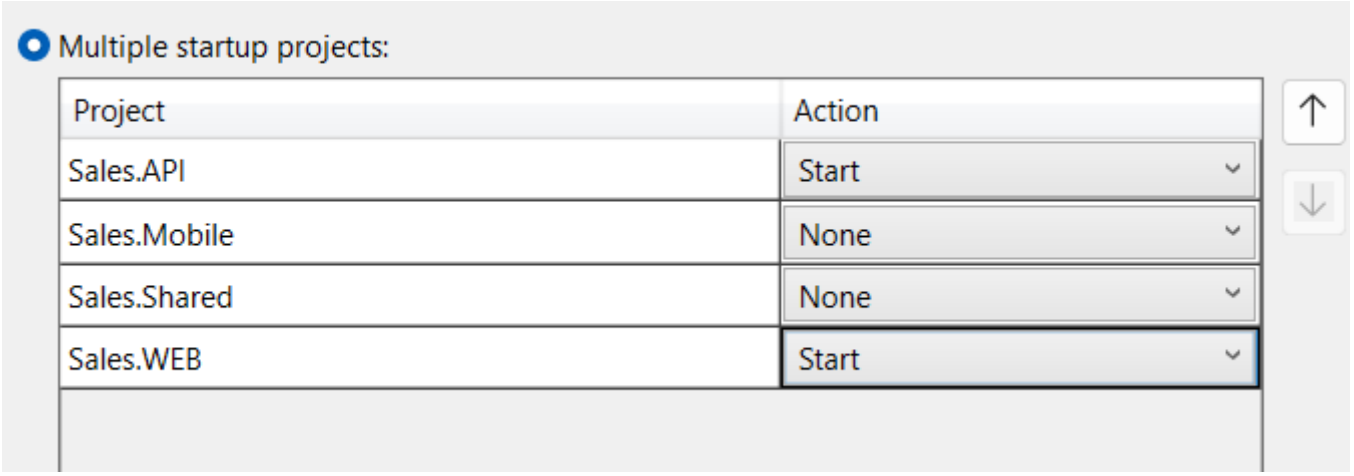
```
@using Sales.WEB.Shared
@using Sales.Shared.Entities
@using Sales.WEB.Repositories
```

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

```
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="counter">
            <span class="oi oi-plus" aria-hidden="true"></span> Counter
        </NavLink>
    </div>
    <div class="nav-item px-3">
```

```
    <NavLink class="nav-link" href="countries">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Países
    </NavLink>
</div>
```

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



16. Probamos y hacemos nuestro commit.

Completando las acciones de crear, editar y borrar países

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

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

    return Ok(country);
}

[HttpPut]
public async Task<ActionResult> Put(Country country)
{
    _context.Update(country);
    await _context.SaveChangesAsync();
    return Ok(country);
}

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

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

    return NoContent();
}
```

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

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

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

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

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

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

```
public async Task<HttpResponseWrapper<object>> Delete(string url)
{
    var responseHTTP = await _httpClient.DeleteAsync(url);
    return new HttpResponseWrapper<object>(null, !responseHTTP.IsSuccessStatusCode, responseHTTP);
}

public async Task<HttpResponseWrapper<object>> Put<T>(string url, T model)
{
    var messageJSON = JsonSerializer.Serialize(model);
    var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");
    var responseHttp = await _httpClient.PutAsync(url, messageContent);
    return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
}

public async Task<HttpResponseWrapper<TResponse>> Put<T, TResponse>(string url, T model)
{
    var messageJSON = JsonSerializer.Serialize(model);
    var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");
    var responseHttp = await _httpClient.PutAsync(url, messageContent);
    if (responseHttp.IsSuccessStatusCode)
    {
        var response = await UnserializeAnswer<TResponse>(responseHttp, _jsonDefaultOptions);
        return new HttpResponseWrapper<TResponse>(response, false, responseHttp);
    }

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

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

22. Vamos a la página de Sweet Alert 2 (Basaingeal/Razor.SweetAlert2: A Razor class library for interacting with SweetAlert2 (github.com) y copiamos el script que debemos de agregar al **index.html** que está en el **wwwroot** de nuestro proyecto **WEB**.

```
    <script src="_framework/blazor.webassembly.js"></script>
    <script src="_content/CurrieTechnologies.Razor.SweetAlert2/sweetAlert2.min.js"></script>
</body>
```

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

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

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

```
<EditForm Model="country" OnValidSubmit="OnSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>País:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Country.Name" />
            <ValidationMessage For="@(() => Country.Name)" />
        </div>
    </div>

    <button class="btn btn-primary" type="submit">Guardar Cambios</button>
    <button class="btn btn-success" @onclick="ReturnAction">Regresar</button>
</EditForm>

@code {
    [EditorRequired]
    [Parameter]
    public Country Country { get; set; } = null!;

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

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

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

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

<h3>Crear País</h3>

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

@code {
    private Country country = new();
```

```
    private async Task Create()
    {
        var responseHttp = await repository.Post("/api/countries", country);
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message);
            return;
        }

        navigationManager.NavigateTo("/countries");
    }

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

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

```
<h3>Países</h3>

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

<GenericList MyList="Countries">
```

27. Probamos la creación de países por interfaz.

28. Mejorermos el formulario previniendo que el usuario salga y deje el formulario incompleto, modificamos nuestro componente **CountryForm**:

```
@inject SweetAlertService sweetAlertService

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

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

    <button class="btn btn-primary" type="submit">Guardar Cambios</button>
    <button class="btn btn-success" @onclick="ReturnAction">Regresar</button>
</EditForm>

@code {
    private EditContext editContext = null!;

    protected override void OnInitialized()
```

```
    {
        editContext = new(Country);
    }

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

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

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

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

    private async Task OnBeforeInternalNavigation(LocationChangingContext context)
    {
        var formWasEdited = editContext.IsModified();

        if (!formWasEdited)
        {
            return;
        }

        if (FormPostedSuccessfully)
        {
            return;
        }

        var result = await 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();
    }
}
```

29. Y hacemos este cambia a **CreateCountry**:

```
@page "/countries/create"
```

```
@inject NavigationManager navigationManager
@inject IRepository repository
@inject SweetAlertService sweetAlertService

<h3>Crear País</h3>

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

@code {
    private Country country = new();
    private CountryForm? countryForm;

    private async Task Create()
    {
        var httpResponse = await repository.Post("api/countries", country);

        if (httpResponse.Error)
        {
            var mensajeError = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            countryForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo("countries");
        }
    }

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

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

31. Ahora creamos el componente **CountryEdit**:

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

<h3>Editar País</h3>

@if (country is null)
{
    <p>Cargando...</p>
}
else
{
    <CountryForm @ref="countryForm" Country="country" OnValidSubmit="Edit" ReturnAction="Return" />
}
```

```
@code {
    private Country? country;
    private CountryForm? countryForm;

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

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

        if (responseHTTP.Error)
        {
            if (responseHTTP.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("countries");
            }
            else
            {
                var messageError = await responseHTTP.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", messageError, SweetAlertIcon.Error);
            }
        }
        else
        {
            country = responseHTTP.Response;
        }
    }

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

        if (responseHTTP.Error)
        {
            var mensajeError = await responseHTTP.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            countryForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo("countries");
        }
    }

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

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

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

<h3>Paises</h3>

<div class="mb-3">
    <a class="btn btn-primary" href="/countries/create">Nuevo País</a>
</div>

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

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

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

    private async Task Load()
    {
        var responseHppt = await repository.Get<List<Country>>("api/countries");
        Countries = responseHppt.Response!;
    }

    private async Task Delete(Country country)
    {
```

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

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

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

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

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

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

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

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

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

```
using Sales.Shared.Entities;

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

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

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

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

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

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

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

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

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

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

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

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

Actividad #1

Con el conocimiento adquirido hasta el momento hacer lo mismo para las categorías. El modelo categoría es muy sencillo, solo tiene el Id y el Name (igual a país). Cree todo lo necesario para que haya un CRUD de categorías, y modifique el alimentador de base de datos para que adicione algunas categorías por defecto.

Relación uno a muchos e índice compuesto

1. Creamos la entidad **State**:

```
using System.ComponentModel.DataAnnotations;

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

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

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

2. Modificamos la entidad **Country**:

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

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

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

3. Creamos la entidad **City**:

```
using System.ComponentModel.DataAnnotations;

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

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

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

4. Modificamos la entidad **State**:

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

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

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

5. Modificamos el **DataContext**:

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

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

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

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

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

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

6. Antes de continuar vamos a modifcar la entidad **State** y **City** para agregar el **CountryId** y **StateId** y esto nos va a facilitar la vida para cuando estemos implementando el CRUD multi-nivel:

   Para **State** agregamos:

```
public int CountryId { get; set; }
```

   Para **City** agregamos:

```
public int StateId { get; set; }
```

7. Luego de esto corremos para agregar migración y la aplicamos.

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

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

9. 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();
}
```

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

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

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

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

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

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

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

```
                    </td>
                    <td>
                        @country.StatesNumber
                    </td>
                </tr>
            }
        </tbody>
    </table>
  </RecordsComplete>
</GenericList>
```

14. Probamos y hacemos el **commit**.

Creando un CRUD multinivel

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

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

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

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

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

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

```
        return Ok(state);
    }

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

            return BadRequest(dbUpdateException.Message);
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }

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

            return BadRequest(dbUpdateException.Message);
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }

    [HttpDelete("{id:int}")]
    public async Task<IActionResult> DeleteAsync(int id)
```

```
        {
            var state = await _context.States.FirstOrDefaultAsync(x => x.Id == id);
            if (state == null)
            {
                return NotFound();
            }

            _context.Remove(state);
            await _context.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

16. Luego creamos el **CitiesController**:

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

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

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

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

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

            return Ok(city);
        }

        [HttpPost]
        public async Task<ActionResult> PostAsync(City city)
```

```
    {
        try
        {
            _context.Add(city);
            await _context.SaveChangesAsync();
            return Ok(city);
        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
            {
                return BadRequest("Ya existe una ciudad con el mismo nombre.");
            }

            return BadRequest(dbUpdateException.Message);
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }

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

            return BadRequest(dbUpdateException.Message);
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }

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

```
            _context.Remove(city);
            await _context.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

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

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

@if(country is null)
{
    <p>Cargando...</p>
} else
{
    <h3>@country.Name</h3>
    <div class="mb-2">
        <a class="btn btn-primary" href="/states/create/@country.Id">Nuevo Estado/Departamento</a>
        <a class="btn btn-success" href="/countries">Regresar</a>
    </div>

    <GenericList MyList="states">
        <Body>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Estado / Departamento</th>
                        <th>Ciudades</th>
                        <th></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var state in states!)
                    {
                        <tr>
                            <td>
                                @state.Name
                            </td>
                            <td>
                                @state.CitiesNumber
                            </td>
                            <td>
                                <a class="btn btn-info" href="/states/details/@state.Id">Detalles</a>
                                <a class="btn btn-warning" href="/states/edit/@state.Id">Editar</a>
                                <button class="btn btn-danger" @onclick=@(() => DeleteAsync(state.Id))>Borrar</button>
                            </td>
                        </tr>
                    }
                </tbody>
```

```
        </table>
    </Body>
</GenericList>
}

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

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

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

    private async Task LoadAsync()
    {
        var responseHttp = await repository.Get<Country>($"/api/countries/{Id}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/countries");
                return;
            }

            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        country = responseHttp.Response;
        states = country!.States!.ToList();
    }

    private async Task DeleteAsync(int id)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }
```

```
        var responseHttp = await repository.Delete($"/api/states/{id}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }
}
```

18. Probamos lo que llevamos hasta el momento.

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

```
@inject SweetAlertService sweetAlertService

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator/>
    <div class="mb-3">
        <label>Estado/Departamenteo:</label>
        <div>
            <InputText class="form-control" @bind-Value="@State.Name"/>
            <ValidationMessage For="@(() => State.Name)" />
        </div>
    </div>
    <button class="btn btn-primary" type="submit">Guardar Cambios</button>
    <button class="btn btn-success" @onclick="ReturnAction">Regresar</button>
</EditForm>

@code {
    private EditContext editContext = null!;

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

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

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

    public bool FormPostedSuccessfully { get; set; }
```

```
    protected override void OnInitialized()
    {
        editContext = new(State);
    }

    private async Task OnBeforeInternalNavigation(LocationChangingContext context)
    {
        var formWasMofied = editContext.IsModified();
        if (!formWasMofied || FormPostedSuccessfully)
        {
            return;
        }

        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Deseas abandonar la página y perder los cambios?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = !string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        context.PreventNavigation();
    }
}
```

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

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

<h3>Crear Estado/Departamento</h3>

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

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

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

    private async Task CreateAsync()
```

```
    {

        state.CountryId = CountryId;
        var httpResponse = await repository.Post("/api/states", state);
        if (httpResponse.Error)
        {
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        Return();
    }

    private void Return()
    {
        stateForm!.FormPostedSuccessfully = true;
        navigationManager.NavigateTo($"/countries/details/{CountryId}");
    }
}
```

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

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

<h3>Editar Estado/Departamento</h3>

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

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

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

    protected override async Task OnInitializedAsync()
    {
        var responseHttp = await repository.Get<State>($"/api/states/{StateId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)
            {
```

```
            navigationManager.NavigateTo("/countries");
            return;
        }

        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    state = responseHttp.Response;
}

private async Task EditAsync()
{
    var responseHttp = await repository.Put("/api/states", state);
    if (responseHttp.Error)
    {
        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    Return();
}

private void Return()
{
    stateForm!.FormPostedSuccessfully = true;
    navigationManager.NavigateTo($"/countries/details/{state!.CountryId}");
}
}
```

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

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

@if (state is null)
{
    <p>Cargando...</p>
}
else
{
    <h3>@state.Name</h3>
    <div class="mb-2">
        <a class="btn btn-primary" href="/cities/create/@state.Id">Nueva Ciudad</a>
        <a class="btn btn-success" href="/countries/details/@state.CountryId">Regresar</a>
    </div>

    <GenericList MyList="cities">
        <Body>
```

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

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

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

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

    private async Task LoadAsync()
    {
        var responseHttp = await repository.Get<State>($"/api/states/{StateId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/countries");
                return;
            }

            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }
```

```
        state = responseHttp.Response;
        cities = state!.Cities!.ToList();
    }

    private async Task DeleteAsync(int cityId)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        var responseHttp = await repository.Delete($"/api/cities/{cityId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }
}
```

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

```
@inject SweetAlertService sweetAlertService

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Cuidad:</label>
        <div>
            <InputText class="form-control" @bind-Value="@City.Name" />
            <ValidationMessage For="@(() => City.Name)" />
        </div>
    </div>
    <button class="btn btn-primary" type="submit">Guardar Cambios</button>
```

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

@code {
    private EditContext editContext = null!;

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

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

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

    public bool FormPostedSuccessfully { get; set; }

    protected override void OnInitialized()
    {
        editContext = new(City);
    }

    private async Task OnBeforeInternalNavigation(LocationChangingContext context)
    {
        var formWasMofied = editContext.IsModified();
        if (!formWasMofied || FormPostedSuccessfully)
        {
            return;
        }

        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Deseas abandonar la página y perder los cambios?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = !string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        context.PreventNavigation();
    }
}
```

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

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

<h3>Crear Ciudad</h3>

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

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

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

    private async Task CreateAsync()
    {
        city.StateId = StateId;
        var httpResponse = await repository.Post("/api/cities", city);
        if (httpResponse.Error)
        {
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        Return();
    }

    private void Return()
    {
        cityForm!.FormPostedSuccessfully = true;
        navigationManager.NavigateTo($"/states/details/{StateId}");
    }
}
```

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

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

<h3>Editar Ciudad</h3>

@if (city is null)
{
    <p>Cargando...</p>
}
```

```
else
{
    <CityForm @ref="cityForm" City="city" OnValidSubmit="EditAsync" ReturnAction="Return" />
}

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

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

    protected override async Task OnInitializedAsync()
    {
        var responseHttp = await repository.Get<City>($"/api/cities/{CityId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/countries");
                return;
            }

            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        city = responseHttp.Response;
    }

    private async Task EditAsync()
    {
        var responseHttp = await repository.Put("/api/cities", city);
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        Return();
    }

    private void Return()
    {
        cityForm!.FormPostedSuccessfully = true;
        navigationManager.NavigateTo($"/states/details/{city!.StateId}");
    }
}
```

26. Probamos y hacemos el **commit**.

Poblar los Países, Estados y Ciudades con un API externa

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

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

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

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

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

        public string? Message { get; set; }

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

30. Luego continuamos con **CountryResponse**:

```
using Newtonsoft.Json;

namespace Sales.Shared.Responses
{
    public class CountryResponse
    {
```

```
        [JsonProperty("id")]
        public long Id { get; set; }

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

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

31. Continuamos con **StateResponse**:

```
using Newtonsoft.Json;

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

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

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

32. Y luego con **CityResponse**:

```
using Newtonsoft.Json;

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

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

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

```
using Sales.Shared.Responses;

namespace Sales.API.Services
{
    public interface IApiService
    {
```

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

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

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

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

    public ApiService(IConfiguration configuration)
    {
      _configuration = configuration;
      _urlBase = _configuration["CoutriesAPI:urlBase"]!;
      _tokenName = _configuration["CoutriesAPI:tokenName"]!;
      _tokenValue = _configuration["CoutriesAPI:tokenValue"]!;
    }

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

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

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

        List<T> list = JsonConvert.DeserializeObject<List<T>>(result)!;
        return new Response
        {
          IsSuccess = true,
          Result = list
```

```
            };
        }
        catch (Exception ex)
        {
            return new Response
            {
                IsSuccess = false,
                Message = ex.Message
            };
        }
    }
  }
}
```

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

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

36. Luego modificamos el **SeedDb**:

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

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

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

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

    private async Task CheckCountriesAsync()
    {
      if (!_context.Countries.Any())
      {
        Response responseCountries = await _apiService.GetListAsync<CountryResponse>("/v1", "/countries");
        if (responseCountries.IsSuccess)
        {
          List<CountryResponse> countries = (List<CountryResponse>)responseCountries.Result!;
          foreach (CountryResponse countryResponse in countries)
```

```
            {
                Country country = await _context.Countries!.FirstOrDefaultAsync(c => c.Name ==
countryResponse.Name!)!;
                if (country == null)
                {
                    country = new() { Name = countryResponse.Name!, States = new List<State>() };
                    Response responseStates = await _apiService.GetListAsync<StateResponse>("/v1",
$"/countries/{countryResponse.Iso2}/states");
                    if (responseStates.IsSuccess)
                    {
                        List<StateResponse> states = (List<StateResponse>)responseStates.Result!;
                        foreach (StateResponse stateResponse in states!)
                        {
                            State state = country.States!.FirstOrDefault(s => s.Name == stateResponse.Name!)!;
                            if (state == null)
                            {
                                state = new() { Name = stateResponse.Name!, Cities = new List<City>() };
                                Response responseCities = await _apiService.GetListAsync<CityResponse>("/v1",
$"/countries/{countryResponse.Iso2}/states/{stateResponse.Iso2}/cities");
                                if (responseCities.IsSuccess)
                                {
                                    List<CityResponse> cities = (List<CityResponse>)responseCities.Result!;
                                    foreach (CityResponse cityResponse in cities)
                                    {
                                        if (cityResponse.Name == "Mosfellsbær" || cityResponse.Name == "Șăulița")
                                        {
                                            continue;
                                        }
                                        City city = state.Cities!.FirstOrDefault(c => c.Name == cityResponse.Name!)!;
                                        if (city == null)
                                        {
                                            state.Cities.Add(new City() { Name = cityResponse.Name! });
                                        }
                                    }
                                }
                                if (state.CitiesNumber > 0)
                                {
                                    country.States.Add(state);
                                }
                            }
                        }
                    }
                    if (country.StatesNumber > 0)
                    {
                        _context.Countries.Add(country);
                        await _context.SaveChangesAsync();
                    }
                }
            }
        }
    }
}
```

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

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

39. Probamos y hacemos el **commit**.

Agregando paginación

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

```
namespace Sales.Shared.DTOs
{
    public class PaginationDTO
    {
        public int Id { get; set; }

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

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

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

```
using Sales.Shared.DTOs;

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

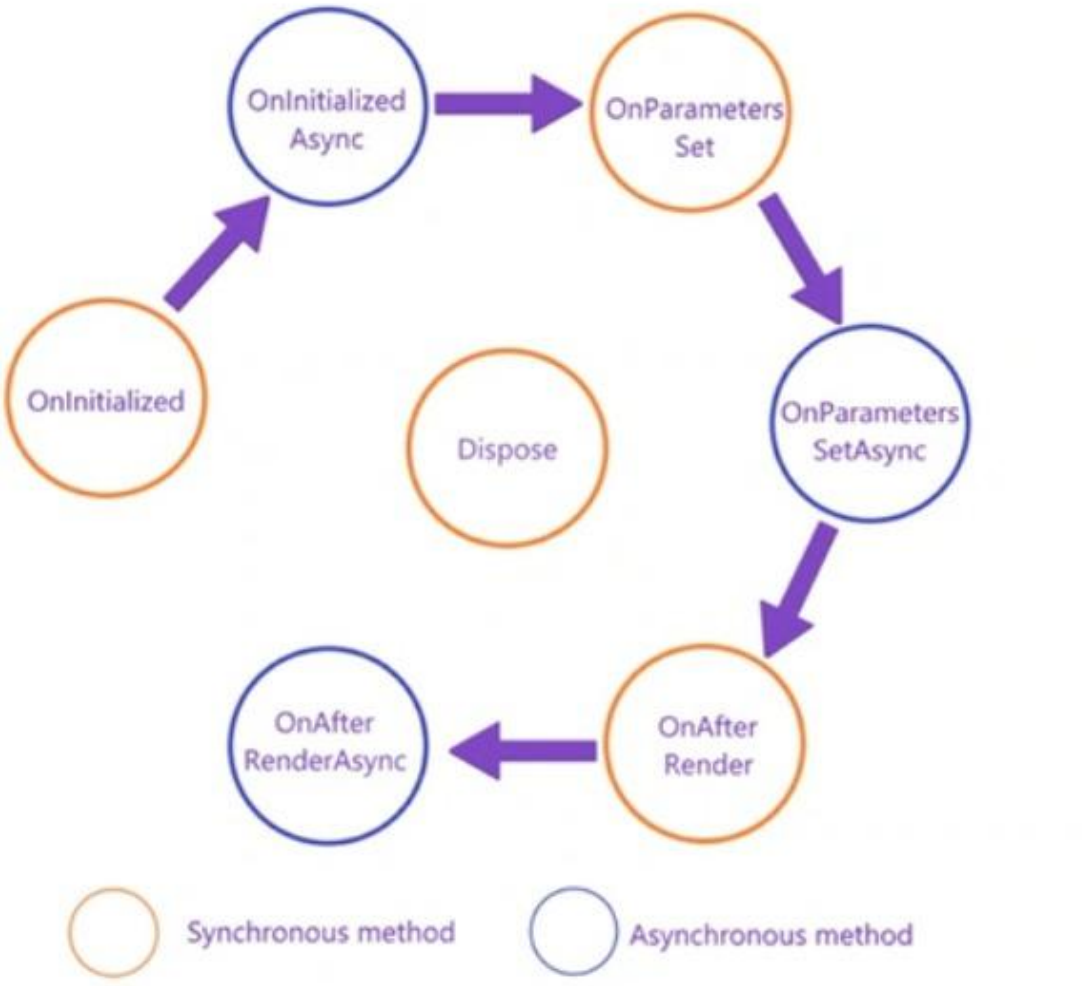
3. Modificamos el **CountriesController** para agregar la paginación en el método **GET** y de paso agregamos el método **GetPages**:

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

    return Ok(await queryable
        .OrderBy(x => x.Name)
```

```
        .Paginate(pagination)
        .ToListAsync());
}

[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Countries.AsQueryable();
    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

4. Probamos la paginación por el Swagger.

5. This is the lifecycle for Blazor App:



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

```
<nav>
    <ul class="pagination">

        @foreach (var link in Links)
```

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


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

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

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

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

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

    }
```

```
    private async Task InternalSelectedPage(PageModel pageModel)
    {
        if (pageModel.Page == CurrentPage || pageModel.Page == 0)
        {
            return;
        }

        await SelectedPage.InvokeAsync(pageModel.Page);
    }

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

7. Modificamos nuestro componente **CountriesIndex**:

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

<h3>Países</h3>

<Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPage" />

<GenericList MyList="Countries">
    <Body>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>País</th>
                    <th style="width:220px">Estados / Departamentos</th>
                    <th style="width:260px"></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var country in Countries!)
                {
                    <tr>
                        <td>
                            @country.Name
                        </td>
                        <td>
                            @country.StatesNumber
                        </td>
                        <td>
                            <a class="btn btn-info" href="/countries/details/@country.Id">Detalles</a>
```

```
                    <a class="btn btn-warning" href="/countries/edit/@country.Id">Editar</a>
                    <button class="btn btn-danger" @onclick=@(() => DeleteAsync(country.Id))>Borrar</button>
                </td>
            </tr>
        }
    </tbody>
</table>
</Body>
</GenericList>

@code {
    public List<Country>? Countries { get; set; }
    private int currentPage = 1;
    private int totalPages;

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

    private async Task SelectedPage(int page)
    {
        currentPage = page;
        await LoadAsync(page);
    }

    private async Task LoadAsync(int page = 1)
    {
        string url1 = $"api/countries?page={page}";
        string url2 = $"api/countries/totalPages";

        var responseHppt = await repository.Get<List<Country>>(url1);
        var responseHppt2 = await repository.Get<int>(url2);
        Countries = responseHppt.Response!;
        totalPages = responseHppt2.Response!;
    }

    private async Task DeleteAsync(int id)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }
```

```
        var responseHttp = await repository.Delete($"/api/countries/{id}");
        if (responseHttp.Error)
        {
            if(responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }
}
```

8. Probamos.

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

```
[HttpGet]
public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.States
        .Include(x => x.Cities)
        .Where(x => x.Country!.Id == pagination.Id)
        .AsQueryable();

    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginate(pagination)
        .ToListAsync());
}

[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.States
        .Where(x => x.Country!.Id == pagination.Id)
        .AsQueryable();

    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

10. Probamos en swagger:

11. Luego modificamos el **CountryDetails**:

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

```
@inject SweetAlertService sweetAlertService

@if(country is null)
{
    <p>Cargando...</p>
} else
{
    <h3>@country.Name</h3>

    <Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPage" />

    <GenericList MyList="sates!">
        <Body>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Estado / Departamento</th>
                        <th style="width:140px">Ciudades</th>
                        <th style="width:260px"></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var state in states!)
                    {
                        <tr>
                            <td>
                                @state.Name
                            </td>
                            <td>
                                @state.CitiesNumber
                            </td>
                            <td>
                                <a class="btn btn-info" href="/states/details/@state.Id">Detalles</a>
                                <a class="btn btn-warning" href="/states/edit/@state.Id">Editar</a>
                                <button class="btn btn-danger" @onclick=@(() => DeleteAsync(state.Id))>Borrar</button>
                            </td>
                        </tr>
                    }
                </tbody>
            </table>
        </Body>
    </GenericList>
}

@code {
    private Country? country;
    private List<State>? states;
    private int currentPage = 1;
    private int totalPages;

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

```
protected override async Task OnInitializedAsync()
{
    await LoadAsync();
}

private async Task SelectedPage(int page)
{
    currentPage = page;
    await LoadAsync(page);
}

private async Task LoadAsync(int page = 1)
{
    string url1 = $"api/states?id={Id}&page={page}";
    string url2 = $"api/states/totalPages?id={Id}";
    var responseHppt = await repository.Get<Country>($"api/countries/{Id}");
    var responseHppt2 = await repository.Get<List<State>>(url1);
    var responseHppt3 = await repository.Get<int>(url2);
    country = responseHppt.Response;
    states = responseHppt2.Response;
    totalPages = responseHppt3.Response;
}

private async Task DeleteAsync(int id)
{
    var result = await sweetAlertService.FireAsync(new SweetAlertOptions
        {
            Title = "Confirmación",
            Text = "¿Realmente deseas eliminar el registro?",
            Icon = SweetAlertIcon.Question,
            ShowCancelButton = true,
            CancelButtonText = "No",
            ConfirmButtonText = "Si"
        });

    var confirm = string.IsNullOrEmpty(result.Value);
    if (confirm)
    {
        return;
    }

    var responseHttp = await repository.Delete($"/api/states/{id}");
    if (responseHttp.Error)
    {
        if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }
    }

    await LoadAsync();
```

```
    }
}
```

12. Probamos.

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

```
[HttpGet]
public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Cities
        .Where(x => x.State!.Id == pagination.Id)
        .AsQueryable();

    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginate(pagination)
        .ToListAsync());
}


[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Cities
        .Where(x => x.State!.Id == pagination.Id)
        .AsQueryable();

    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

14. Probamos en swagger:

15. Luego modificamos el **StateDetail**:

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

@if (state is null)
{
    <p>Cargando...</p>
}
else
{
    <h3>@state.Name</h3>

    <Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPage" />
```

```
    <GenericList MyList="cities!">
        <Body>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Ciudad</th>
                        <th style="width:180px"></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var city in cities!)
                    {
                        <tr>
                            <td>
                                @city.Name
                            </td>
                            <td>
                                <a class="btn btn-warning" href="/cities/edit/@city.Id">Editar</a>
                                <button class="btn btn-danger" @onclick=@(() => DeleteAsync(city.Id))>Borrar</button>
                            </td>
                        </tr>
                    }
                </tbody>
            </table>
        </Body>
    </GenericList>
}

@code {
    private State? state;
    private List<City>? cities;
    private int currentPage = 1;
    private int totalPages;

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

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

    private async Task SelectedPage(int page)
    {
        currentPage = page;
        await LoadAsync(page);
    }

    private async Task LoadAsync(int page = 1)
    {
        string url1 = $"api/cities?id={StateId}&page={page}";
        string url2 = $"api/cities/totalPages?id={StateId}";
        var responseHppt = await repository.Get<State>($"api/states/{StateId}");
```

```
        var responseHppt2 = await repository.Get<List<City>>(url1);
        var responseHppt3 = await repository.Get<int>(url2);
        state = responseHppt.Response;
        cities = responseHppt2.Response;
        totalPages = responseHppt3.Response;
    }

    private async Task DeleteAsync(int CityId)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        var responseHttp = await repository.Delete($"/api/cities/{CityId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }
}
```

16. Probamos y hacemos el **commit**.

Agregando filtros

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

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

```
public string? Filter { get; set; }
```

2. En el projecto **WEB** modificamos los métodos **Get** y **GetPages** del controlador **CountriesController**:

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

    if (!string.IsNullOrWhiteSpace(pagination.Filter))
    {
        queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
    }

    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginate(pagination)
        .ToListAsync());
}

[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Countries.AsQueryable();

    if (!string.IsNullOrWhiteSpace(pagination.Filter))
    {
        queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
    }

    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

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

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

<h3>Países</h3>

<div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
    <div>
        <a class="btn btn-primary" href="/countries/create">Nuevo País</a>
    </div>
    <div class="mx-2">
        <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar país..."
@bind-value="Filter" />
```

```
    </div>
    <div>
        <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync">Filtrar</button>
        <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync">Limpiar</button>
    </div>
</div>

<Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPage" />

<GenericList MyList="Countries">
    <Body>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>País</th>
                    <th style="width:220px">Estados / Departamentos</th>
                    <th style="width:260px"></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var country in Countries!)
                {
                    <tr>
                        <td>
                            @country.Name
                        </td>
                        <td>
                            @country.StatesNumber
                        </td>
                        <td>
                            <a class="btn btn-info" href="/countries/details/@country.Id">Detalles</a>
                            <a class="btn btn-warning" href="/countries/edit/@country.Id">Editar</a>
                            <button class="btn btn-danger" @onclick=@(() => DeleteAsync(country.Id))>Borrar</button>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </Body>
</GenericList>

@code {
    public List<Country>? Countries { get; set; }
    private int currentPage = 1;
    private int totalPages;

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
```

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

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

    private async Task SelectedPage(int page)
    {
        currentPage = page;
        await LoadAsync(page);
    }

    private async Task LoadAsync(int page = 1)
    {
        if (!string.IsNullOrWhiteSpace(Page))
        {
            page = Convert.ToInt32(Page);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/countries?page={page}";
            url2 = $"api/countries/totalPages";
        }
        else
        {
            url1 = $"api/countries?page={page}&filter={Filter}";
            url2 = $"api/countries/totalPages?filter={Filter}";
        }

        var responseHppt = await repository.Get<List<Country>>(url1);
        var responseHppt2 = await repository.Get<int>(url2);
        Countries = responseHppt.Response!;
        totalPages = responseHppt2.Response!;
    }

    private async Task DeleteAsync(int id)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

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

```
        {
            return;
        }

        var responseHttp = await repository.Delete($"/api/countries/{id}");
        if (responseHttp.Error)
        {
            if(responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }

    private async Task ApplyFilterAsync()
    {
        int page = 1;
        await LoadAsync(page);
        await SelectedPage(page);
    }
}
```

4. Probamos y hacemos el **commit**.

5. Replicamos para estados y ciudades, primero modificamos el **StatesController**:

```
[HttpGet]
public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.States
        .Include(x => x.Cities)
        .Where(x => x.Country!.Id == pagination.Id)
        .AsQueryable();

    if (!string.IsNullOrWhiteSpace(pagination.Filter))
    {
        queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
    }

    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginate(pagination)
        .ToListAsync());
}
```

```
[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.States
        .Where(x => x.Country!.Id == pagination.Id)
        .AsQueryable();

    if (!string.IsNullOrWhiteSpace(pagination.Filter))
    {
        queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
    }

    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

6. Luego modificamos el **CitiesController**:

```
[HttpGet]
public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Cities
        .Where(x => x.State!.Id == pagination.Id)
        .AsQueryable();

    if (!string.IsNullOrWhiteSpace(pagination.Filter))
    {
        queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
    }

    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginate(pagination)
        .ToListAsync());
}

[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Cities
        .Where(x => x.State!.Id == pagination.Id)
        .AsQueryable();

    if (!string.IsNullOrWhiteSpace(pagination.Filter))
    {
        queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
    }

    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

7. Modificamos el **CountryDetails**.

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

@if(country is null)
{
    <p>Cargando...</p>
} else
{
    <h3>@country.Name</h3>

    <div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
        <div>
            <a class="btn btn-primary" href="/states/create/@country.Id">Nuevo Estado/Departamento</a>
            <a class="btn btn-success" href="/countries">Regresar</a>
        </div>
        <div class="mx-2">
            <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar
estado/departamento..." @bind-value="Filter" />
        </div>
        <div>
            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync">Filtrar</button>
            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync">Limpiar</button>
        </div>
    </div>

    <Pagination CurrentPage="currentPage"
        TotalPages="totalPages"
        SelectedPage="SelectedPage" />

    <GenericList MyList="states!">
        <Body>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Estado / Departamento</th>
                        <th style="width:140px">Ciudades</th>
                        <th style="width:260px"></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var state in states!)
                    {
                        <tr>
                            <td>
                                @state.Name
                            </td>
                            <td>
                                @state.CitiesNumber
                            </td>
```

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

@code {
    private Country? country;
    private List<State>? states;
    private int currentPage = 1;
    private int totalPages;

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

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

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

    private async Task SelectedPage(int page)
    {
        currentPage = page;
        await LoadAsync(page);
    }

    private async Task LoadAsync(int page = 1)
    {
        if (!string.IsNullOrWhiteSpace(Page))
        {
            page = Convert.ToInt32(Page);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/states?id={Id}&page={page}";
```

```
            url2 = $"api/states/totalPages?id={Id}";
        }
        else
        {
            url1 = $"api/states?id={Id}&page={page}&filter={Filter}";
            url2 = $"api/states/totalPages?id={Id}&filter={Filter}";
        }

        var responseHppt = await repository.Get<Country>($"api/countries/{Id}");
        var responseHppt2 = await repository.Get<List<State>>(url1);
        var responseHppt3 = await repository.Get<int>(url2);
        country = responseHppt.Response;
        states = responseHppt2.Response;
        totalPages = responseHppt3.Response;
    }

    private async Task DeleteAsync(int id)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        var responseHttp = await repository.Delete($"/api/states/{id}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }
```

```
    private async Task ApplyFilterAsync()
    {
        int page = 1;
        await LoadAsync(page);
        await SelectedPage(page);
    }
}
```

8. Modificamos el **StateDetails**.

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

@if (state is null)
{
    <p>Cargando...</p>
}
else
{
    <h3>@state.Name</h3>

    <div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
        <div>
            <a class="btn btn-primary" href="/cities/create/@StateId">Nueva Ciudad</a>
            <a class="btn btn-success" href="/countries/details/@state.CountryId">Regresar</a>
        </div>
        <div class="mx-2">
            <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar ciudad..."
@bind-value="Filter" />
        </div>
        <div>
            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync">Filtrar</button>
            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync">Limpiar</button>
        </div>
    </div>

    <Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPage" />

    <GenericList MyList="cities!">
        <Body>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Ciudad</th>
                        <th style="width:180px"></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var city in cities!)
                    {
```

```
                <tr>
                    <td>
                        @city.Name
                    </td>
                    <td>
                        <a class="btn btn-warning" href="/cities/edit/@city.Id">Editar</a>
                        <button class="btn btn-danger" @onclick=@(() => DeleteAsync(city.Id))>Borrar</button>
                    </td>
                </tr>
            }
        </tbody>
    </table>
</Body>
</GenericList>
}

@code {
    private State? state;
    private List<City>? cities;
    private int currentPage = 1;
    private int totalPages;

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

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

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

    private async Task SelectedPage(int page)
    {
        currentPage = page;
        await LoadAsync(page);
    }

    private async Task LoadAsync(int page = 1)
    {
        if (!string.IsNullOrWhiteSpace(Page))
        {
            page = Convert.ToInt32(Page);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;
```

```
        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/cities?id={StateId}&page={page}";
            url2 = $"api/cities/totalPages?id={StateId}";
        }
        else
        {
            url1 = $"api/cities?id={StateId}&page={page}&filter={Filter}";
            url2 = $"api/cities/totalPages?id={StateId}&filter={Filter}";
        }

        var responseHppt = await repository.Get<State>($"api/states/{StateId}");
        var responseHppt2 = await repository.Get<List<City>>(url1);
        var responseHppt3 = await repository.Get<int>(url2);
        state = responseHppt.Response;
        cities = responseHppt2.Response;
        totalPages = responseHppt3.Response;
    }

    private async Task DeleteAsync(int CityId)
    {
        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Realmente deseas eliminar el registro?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        var responseHttp = await repository.Delete($"/api/cities/{CityId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)
            {
                var message = await responseHttp.GetErrorMessageAsync();
                await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
                return;
            }
        }

        await LoadAsync();
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
```

```
        await ApplyFilterAsync();
    }

    private async Task ApplyFilterAsync()
    {
        int page = 1;
        await LoadAsync(page);
        await SelectedPage(page);
    }
}
```

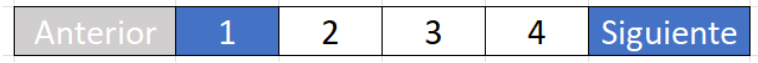
9. Probamos y hacemos el **commit**.

Actividad #2

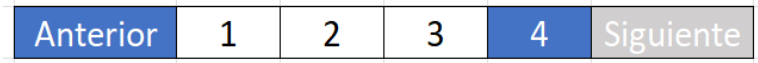
Con el conocimiento adquirido hasta el momento hacer lo mismo para las categorías, es decir, paginación y filtros. Adicionalmente, requiero que modifiquen la funcionalidad del componente genérico Paginantion para que funcione de la siguiente manera:

10. **Si son 10 o menos pagínas muestre, el número de páginas que son, por ejemplo si son 4 páginas deberia mostrar:**

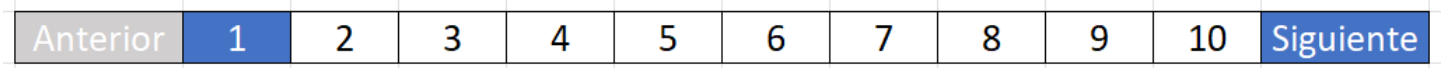
Note que el 2 está de fondo azul porque se supone que es la página activa. Si la página activa fuera la 1, el botón anterior debe estar des-habilitado:



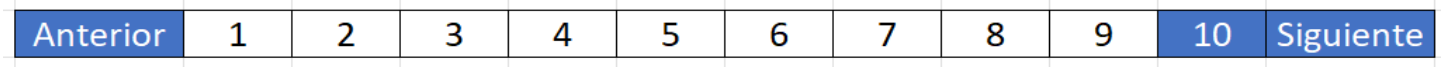
Si la página activa es la última, el botón siguiente debe quedar des-habilitado:



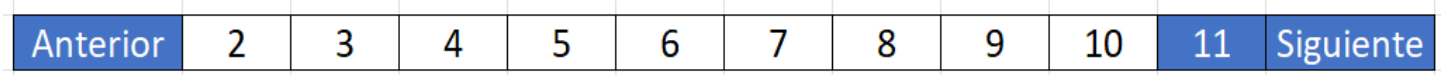
11. **Si son más de 10 pagínas, solo muestre las 10 primeras páginas y el botón siguiente va cambiando los número de página. Por ejemplo si son 20 páginas debe mostrar:**



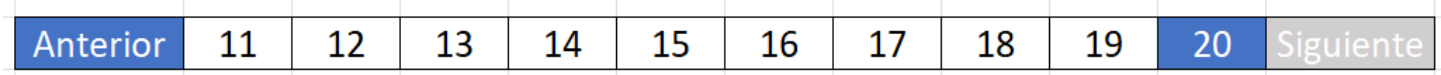
Y cuando el usuario llegue a la página 10 y presione siguiente, debe cambiar los números de las páginas:



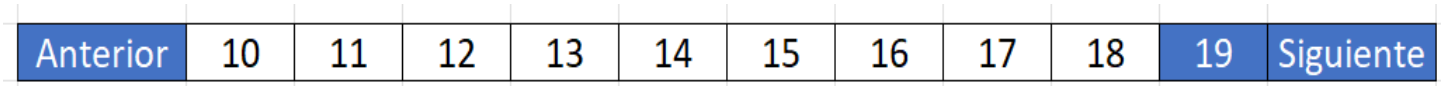
Y clickea en siguiente página:



Y así sucesivamente hasta que llegue al la última página, para nestro ejemplo la 20 en la cual ya se debe deshabilitar el botón siguiente, puesto que ya no hay más paginas:



Y si presiona el botón anterior, debe colocar los números de pagina correctos hasta llegar al 1 y habilitar el boton siguiente, porque ya hay una página siguiente luego de la página 19.



Creando las tablas de usuarios

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

```
namespace Sales.Shared.Enums
{
    public enum UserType
    {
        Admin,
        User
    }
}
```

13. En el proyecto **Shared** el nuget **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.

14. En el proyecto **Shared** en la carpeta **Entities**, crear la entidad **User**:

```
using Microsoft.AspNetCore.Identity;
using Sales.Shared.Enums;
using System.ComponentModel.DataAnnotations;

namespace Sales.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}";
    }
}
```

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

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

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

16. En el proyecto **API** instalar el nugget **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.

17. Modificar el **DataContext**:

```
public class DataContext : IdentityDbContext<User>
```

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

```
using Microsoft.AspNetCore.Identity;
using Sales.Shared.Entities;

namespace Sales.API.Helpers
{
    public interface IUserHelper
    {
        Task<User> GetUserAsync(string email);

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

        Task CheckRoleAsync(string roleName);

        Task AddUserToRoleAsync(User user, string roleName);

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

19. Luego hacemos la implementación de dicha interfaz:

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

namespace Sales.API.Helpers
{
    public class UserHelper : IUserHelper
    {
        private readonly DataContext _context;
        private readonly UserManager<User> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
```

```
        public UserHelper(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole>
roleManager)
        {
            _context = context;
            _userManager = userManager;
            _roleManager = roleManager;
        }

        public async Task<IdentityResult> AddUserAsync(User user, string password)
        {
            return await _userManager.CreateAsync(user, password);
        }

        public async Task AddUserToRoleAsync(User user, string roleName)
        {
            await _userManager.AddToRoleAsync(user, roleName);
        }

        public async Task CheckRoleAsync(string roleName)
        {
            bool roleExists = await _roleManager.RoleExistsAsync(roleName);
            if (!roleExists)
            {
                await _roleManager.CreateAsync(new IdentityRole
                {
                    Name = roleName
                });
            }
        }

        public async Task<User> GetUserAsync(string email)
        {
            return await _context.Users
                .Include(u => u.City)
                .ThenInclude(c => c.State)
                .ThenInclude(s => s.Country)
                .FirstOrDefaultAsync(x => x.Email == email);
        }

        public async Task<bool> IsUserInRoleAsync(User user, string roleName)
        {
            return await _userManager.IsInRoleAsync(user, roleName);
        }
    }
}
```

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

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

builder.Services.AddIdentity<User, IdentityRole>(x =>
{
    x.User.RequireUniqueEmail = true;
    x.Password.RequireDigit = false;
```

```
    x.Password.RequiredUniqueChars = 0;
    x.Password.RequireLowercase = false;
    x.Password.RequireNonAlphanumeric = false;
    x.Password.RequireUppercase = false;
})
    .AddEntityFrameworkStores<DataContext>()
    .AddDefaultTokenProviders();

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

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

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

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

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
```

21. Modificamos el **SeedDb**:

```
public class SeedDb
{
private readonly DataContext _context;
private readonly IApiService _apiService;
private readonly IUserHelper _userHelper;

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

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

```
    await CheckUserAsync("1010", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol",
UserType.Admin);
}

private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string
phone, string address, UserType userType)
{
    var user = await _userHelper.GetUserAsync(email);
    if (user == null)
    {
        user = new User
        {
            FirstName = firstName,
            LastName = lastName,
            Email = email,
            UserName = email,
            PhoneNumber = phone,
            Address = address,
            Document = document,
            City = _context.Cities.FirstOrDefault(),
            UserType = userType,
        };

        await _userHelper.AddUserAsync(user, "123456");
        await _userHelper.AddUserToRoleAsync(user, userType.ToString());
    }

    return user;
}

private async Task CheckRolesAsync()
{
    await _userHelper.CheckRoleAsync(UserType.Admin.ToString());
    await _userHelper.CheckRoleAsync(UserType.User.ToString());
}
```

22. Corremos los siguientes comandos:

```
PM> drop-database
PM> add-migration Users
PM> update-database
```

23. Probamos y hacemos el **commit**.

Creando sistema de seguridad

1. Al proyecto **WEB** agregamos el paquete: **Microsoft.AspNetCore.Components.WebAssembly.Authentication**.

2. Agregamos este using en el **_Imports**:

```
@using Microsoft.AspNetCore.Components.Authorization
```

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

```
using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace Sales.WEB.Auth
{
    public class AuthenticationProviderTest : AuthenticationStateProvider
    {
        public override async Task<AuthenticationState> GetAuthenticationStateAsync()
        {
            var anonimous = new ClaimsIdentity();
            return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonimous)));
        }
    }
}
```

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

```
builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7201/") });
builder.Services.AddScoped<IRepository, Repository>();
builder.Services.AddSweetAlert2();
builder.Services.AddAuthorizationCore();
builder.Services.AddScoped<AuthenticationStateProvider, AuthenticationProviderTest>();
```

5. 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>
```

6. 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)));
}
```

7. Probamos de nuevo y vemos que tarda los 3 segundos haciendo la autorización.

8. Si queremos cambiar el mensaje, modificamos el **App.razor**:

```
<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">
    <Authorizing>
        <p>Autorizando...</p>
    </Authorizing>
</AuthorizeRouteView>
```

9. Probamos de nuevo.

10. Modificacmos el **Index.razor**.

```
@page "/"

<PageTitle>Index</PageTitle>

<AuthorizeView>
    <p>Estas autenticado</p>
</AuthorizeView>

<h1>Hello, world!</h1>

Welcome to your new app.

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

11. Modificamos el **AuthenticationProviderTest**:

```
public override async Task<AuthenticationState> GetAuthenticationStateAsync()
{
    var anonimous = new ClaimsIdentity();
    var zuluUser = new ClaimsIdentity(authenticationType: "test");
    return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(zuluUser)));
}
```

12. Cambiamos el **Index.razor**.

```
<AuthorizeView>
    <Authorized>
        <p>Estas autenticado</p>
    </Authorized>
    <NotAuthorized>
        <p>No estas autorizado</p>
    </NotAuthorized>
</AuthorizeView>
```

13. Y jugamos con el **AuthenticationProviderTest** para ver que pasa con el usuario **anonimous** y con el usuario **zuluUser**.

14. Modificamos nuestro **AuthenticationProviderTest**, para agregar algunos **Claims**:

```
public override async Task<AuthenticationState> GetAuthenticationStateAsync()
{
    var anonimous = new ClaimsIdentity();
    var zuluUser = 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(zuluUser)));
}
```

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

```
<AuthorizeView>
    <Authorized>
        <p>Estas autenticado, @context.User.Identity?.Name</p>
    </Authorized>
    <NotAuthorized>
        <p>No estas autorizado</p>
    </NotAuthorized>
</AuthorizeView>
```

16. 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>
```

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

```
var zuluUser = 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");
```

18. 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="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="oi oi-plus" aria-hidden="true"></span> Counter
```

```
            </NavLink>
        </div>
        <AuthorizeView Roles="Admin">
            <Authorized>
                <div class="nav-item px-3">
                    <NavLink class="nav-link" href="countries">
                        <span class="oi oi-list-rich" aria-hidden="true"></span> Países
                    </NavLink>
                </div>
            </Authorized>
        </AuthorizeView>
    </nav>
</div>
```

19. 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.

20. Para evitar esto le colocamos este atributo a todos los componentes a los que navegamos y queremos proteger:

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

21. 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>
```

22. Antes de continuar aprendamos a identificar si el usuario esta autenticado por código C#, hagamos la prueba en el componente **Counter** y modificamos el **AuthenticationProviderTest** para poder hacer la prueba:

```
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCountAsync">Click me</button>

@code {
    private int currentCount = 0;

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

    private async Task IncrementCountAsync()
    {
        var authenticationState = await authenticationStateTask;
```

```
        var isAuthenticated = authenticationState.User.Identity!.IsAuthenticated;
        if (isAuthenticated)
        {
            currentCount++;
        }
        else
        {
            currentCount--;
        }
    }
}
```

23. Probamos y hacemos el **commit**.

Seguridad desde el backend

24. Antes de empezar corrijamos el Warnig del **GetUserAsync** en el **UserHelper**, esta solución me la pasó: **Rafael Baloyes**. Gracias **Rafa**!

```
public async Task<User> GetUserAsync(string email)
{
    var user = await _context.Users
        .Include(u => u.City!)
        .ThenInclude(c => c.State!)
        .ThenInclude(s => s.Country!)
        .FirstOrDefaultAsync(u => u.Email! == email);
    return user!;
}
```

25. Agregamos al proyecto **API** el paquete **Microsoft.AspNetCore.Authentication.JwtBearer**.

26. Creamos el parámetro **jwtKey** en el appsettings del proyecto **API** (cualquier cosa, entre mas larga mejor):

```
  "AllowedHosts": "*",
  "jwtKey":  "sagdsadgfeSDF674545REFG$%FEfgdslkjfglkjhfgdkljhdR5454545_4TGRGtyo!!kjytkljty"
}
```

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

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

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(x => x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["jwtKey"]!)),
        ClockSkew = TimeSpan.Zero
    });

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

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

```
using Sales.Shared.Entities;
using System.ComponentModel.DataAnnotations;
using System.Xml.Linq;

namespace Sales.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!;
    }
}
```

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

```
using Sales.Shared.Entities;

namespace Sales.Shared.DTOs
{
    public class TokenDTO
    {
        public string Token { get; set; } = null!;

        public DateTime Expiration { get; set; }
    }
}
```

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

```
using System.ComponentModel.DataAnnotations;

namespace Sales.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!;
    }
}
```

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

```
Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();
```

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

```
private readonly DataContext _context;
private readonly UserManager<User> _userManager;
private readonly RoleManager<IdentityRole> _roleManager;
private readonly SignInManager<User> _signInManager;

public UserHelper(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole> roleManager,
SignInManager<User> signInManager)
{
    _context = context;
    _userManager = userManager;
    _roleManager = roleManager;
    _signInManager = signInManager;
}
…
public async Task<SignInResult> LoginAsync(LoginDTO model)
{
    return await _signInManager.PasswordSignInAsync(model.Email, model.Password, false, false);
}

public async Task LogoutAsync()
{
    await _signInManager.SignOutAsync();
}
```

33. Creamos el **AccountsController**:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using Sales.API.Helpers;
using Sales.Shared.DTOs;
using Sales.Shared.Entities;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace Sales.API.Controllers
{
    [ApiController]
    [Route("/api/accounts")]
    public class AccountsController : ControllerBase
    {
        private readonly IUserHelper _userHelper;
```

```
    private readonly IConfiguration _configuration;

    public AccountsController(IUserHelper userHelper, IConfiguration configuration)
    {
        _userHelper = userHelper;
        _configuration = configuration;
    }

    [HttpPost("CreateUser")]
    public async Task<ActionResult> CreateUser([FromBody] UserDTO model)
    {
        User user = model;
        var result = await _userHelper.AddUserAsync(user, model.Password);
        if (result.Succeeded)
        {
            await _userHelper.AddUserToRoleAsync(user, user.UserType.ToString());
            return Ok(BuildToken(user));
        }

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


    [HttpPost("Login")]
    public async Task<ActionResult> Login([FromBody] LoginDTO model)
    {
        var result = await _userHelper.LoginAsync(model);
        if (result.Succeeded)
        {
            var user = await _userHelper.GetUserAsync(model.Email);
            return Ok(BuildToken(user));
        }

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

    private TokenDTO BuildToken(User user)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, user.Email!),
            new Claim(ClaimTypes.Role, user.UserType.ToString()),
            new Claim("Document", user.Document),
            new Claim("FirstName", user.FirstName),
            new Claim("LastName", user.LastName),
            new Claim("Address", user.Address),
            new Claim("Photo", user.Photo ?? string.Empty),
            new Claim("CityId", user.CityId.ToString())
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["jwtKey"]!));
        var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var expiration = DateTime.UtcNow.AddDays(30);
        var token = new JwtSecurityToken(
```

```
            issuer: null,
            audience: null,
            claims: claims,
            expires: expiration,
            signingCredentials: credentials);

        return new TokenDTO
        {
            Token = new JwtSecurityTokenHandler().WriteToken(token),
            Expiration = expiration
        };
    }
  }
}
```

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

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

35. Modificamos el **CountriesIndex**:

```
try
{
  var responseHppt = await repository.Get<List<Country>>(url1);
  var responseHppt2 = await repository.Get<int>(url2);
  Countries = responseHppt.Response!;
  totalPages = responseHppt2.Response!;
}
catch (Exception ex)
{
  await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
}
```

36. Podemos probar por **POSTMAN** como está funcionando nuestro token, y con https://jwt.io/ probamos como está quedando nuestro token.

37. Probamos en la interfaz web, y nos debe salir un error porque aun no le mandamos ningún token a nuestra API. Hacemos el **commit**.

Implementando el registro de usuarios, login & logout

1. En el proyecto **WEB** Instalamos el paquete: **System.IdentityModel.Tokens.Jwt**.

2. En el proyecto **WEB** en la carpeta **Helpers** creamos el **IJSRuntimeExtensionMethods**:

```
using Microsoft.JSInterop;

namespace Sales.WEB.Helpers
{
  public static class IJSRuntimeExtensionMethods
  {
    public static ValueTask<object> SetLocalStorage(this IJSRuntime js, string key, string content)
    {
```

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

    public static ValueTask<object> GetLocalStorage(this IJSRuntime js, string key)
    {
        return js.InvokeAsync<object>("localStorage.getItem", key);
    }

    public static ValueTask<object> RemoveLocalStorage(this IJSRuntime js, string key)
    {
        return js.InvokeAsync<object>("localStorage.removeItem", key);
    }
  }
}
```

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

```
namespace Sales.WEB.Auth
{
  public interface ILoginService
  {
    Task LoginAsync(string token);

    Task LogoutAsync();
  }
}
```

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

```
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.JSInterop;
using Sales.WEB.Helpers;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http.Headers;
using System.Security.Claims;

namespace Sales.WEB.Auth
{
  public class AuthenticationProviderJWT : AuthenticationStateProvider, ILoginService
  {
    private readonly IJSRuntime _jSRuntime;
    private readonly HttpClient _httpClient;
    private readonly String _tokenKey;
    private readonly AuthenticationState _anonimous;

    public AuthenticationProviderJWT(IJSRuntime jSRuntime, HttpClient httpClient)
    {
      _jSRuntime = jSRuntime;
      _httpClient = httpClient;
      _tokenKey = "TOKEN_KEY";
      _anonimous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity()));
    }

    public async override Task<AuthenticationState> GetAuthenticationStateAsync()
```

```
    {
        var token = await _jSRuntime.GetLocalStorage(_tokenKey);
        if (token is null)
        {
            return _anonimous;
        }

        return BuildAuthenticationState(token.ToString()!);
    }

    private AuthenticationState BuildAuthenticationState(string token)
    {
        _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer", token);
        var claims = ParseClaimsFromJWT(token);
        return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));
    }

    private IEnumerable<Claim> ParseClaimsFromJWT(string token)
    {
        var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
        var unserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);
        return unserializedToken.Claims;

    }

    public async Task LoginAsync(string token)
    {
        await _jSRuntime.SetLocalStorage(_tokenKey, token);
        var authState = BuildAuthenticationState(token);
        NotifyAuthenticationStateChanged(Task.FromResult(authState));
    }

    public async Task LogoutAsync()
    {
        await _jSRuntime.RemoveLocalStorage(_tokenKey);
        _httpClient.DefaultRequestHeaders.Authorization = null;
        NotifyAuthenticationStateChanged(Task.FromResult(_anonimous));
    }
  }
}
```

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

```
builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7201/") });
builder.Services.AddScoped<IRepository, Repository>();
builder.Services.AddSweetAlert2();
builder.Services.AddAuthorizationCore();

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

6. Creamos el componente compartido **AuthLinks**:

```
<AuthorizeView>
    <Authorized>
        <span>Hola, @context.User.Identity!.Name</span>
        <a href="Logout" class="nav-link btn btn-link">Cerrar Sesión</a>
    </Authorized>
    <NotAuthorized>
        <a href="Register" class="nav-link btn btn-link">Registro</a>
        <a href="Login" class="nav-link btn btn-link">Iniciar Sesión</a>
    </NotAuthorized>
</AuthorizeView>
```

7. 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>
```

8. Probamos lo que llevamos.

9. Dentro de **Pages** creamos la carpeta **Auth** y dentro de esta el componente **Register**:

```
@page "/Register"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager
@inject ILoginService loginService

<h3>Registrar Nuevo Usuario</h3>

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

    <div class="row">
        <div class="col-6">
            <div class="mb-3">
                <label>Nombres:</label>
                <div>
```

```
                <InputText class="form-control" @bind-Value="@userDTO.FirstName" />
                <ValidationMessage For="@(() => userDTO.FirstName)" />
            </div>
        </div>
        <div class="mb-3">
            <label>Apellidos:</label>
            <div>
                <InputText class="form-control" @bind-Value="@userDTO.LastName" />
                <ValidationMessage For="@(() => userDTO.LastName)" />
            </div>
        </div>
        <div class="mb-3">
            <label>Documento:</label>
            <div>
                <InputText class="form-control" @bind-Value="@userDTO.Document" />
                <ValidationMessage For="@(() => userDTO.Document)" />
            </div>
        </div>
        <div class="mb-3">
            <label>Teléfono:</label>
            <div>
                <InputText class="form-control" @bind-Value="@userDTO.PhoneNumber" />
                <ValidationMessage For="@(() => userDTO.PhoneNumber)" />
            </div>
        </div>
        <div class="mb-3">
            <label>Dirección:</label>
            <div>
                <InputText class="form-control" @bind-Value="@userDTO.Address" />
                <ValidationMessage For="@(() => userDTO.Address)" />
            </div>
        </div>
        <div class="mb-3">
            <label>Email:</label>
            <div>
                <InputText class="form-control" @bind-Value="@userDTO.Email" />
                <ValidationMessage For="@(() => userDTO.Email)" />
            </div>
        </div>
    </div>
    <div class="col-6">
        <div class="mb-3">
            <label>Ciudad:</label>
            <div>
                <InputNumber class="form-control" @bind-Value="@userDTO.CityId" />
                <ValidationMessage For="@(() => userDTO.CityId)" />
            </div>
        </div>
        <div class="mb-3">
            <label>Foto:</label>
            <div>
                <InputText class="form-control" @bind-Value="@userDTO.Photo" />
                <ValidationMessage For="@(() => userDTO.Photo)" />
            </div>
```

```
            </div>
            <div class="mb-3">
                <label>Contraseña:</label>
                <div>
                    <InputText type="password" class="form-control" @bind-Value="@userDTO.Password" />
                    <ValidationMessage For="@(() => userDTO.Password)" />
                </div>
            </div>
            <div class="mb-3">
                <label>Confirmación de contraseña:</label>
                <div>
                    <InputText type="password" class="form-control" @bind-Value="@userDTO.PasswordConfirm" />
                    <ValidationMessage For="@(() => userDTO.PasswordConfirm)" />
                </div>
            </div>
        </div>
    </div>
    <button class="btn btn-primary" type="submit">Registrar</button>
</EditForm>

@code {
    private UserDTO userDTO = new();

    private async Task CreteUserAsync()
    {
        userDTO.UserName = userDTO.Email;
        userDTO.UserType = UserType.User;
        var responseHttp = await repository.Post<UserDTO, TokenDTO>("/api/accounts/CreateUser", userDTO);
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        await loginService.LoginAsync(responseHttp.Response!.Token);
        navigationManager.NavigateTo("/");
    }
}
```

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

```
@page "/Login"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager
@inject ILoginService loginService

<h3>Iniciar Sesión</h3>

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

    <div class="row">
```

```
        <div class="col-4">
            <div class="mb-3">
                <label>Email:</label>
                <div>
                    <InputText class="form-control" @bind-Value="@loginDTO.Email" />
                    <ValidationMessage For="@(() => loginDTO.Email)" />
                </div>
            </div>
            <div class="mb-3">
                <label>Contraseña:</label>
                <div>
                    <InputText type="password" class="form-control" @bind-Value="@loginDTO.Password" />
                    <ValidationMessage For="@(() => loginDTO.Password)" />
                </div>
            </div>
            <button class="btn btn-primary" type="submit">Iniciar Sesión</button>
        </div>
    </div>
</EditForm>

@code {
    private LoginDTO loginDTO = new();

    private async Task LoginAsync()
    {
        var responseHttp = await repository.Post<LoginDTO, TokenDTO>("/api/accounts/Login", loginDTO);
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        await loginService.LoginAsync(responseHttp.Response!.Token);
        navigationManager.NavigateTo("/");
    }
}
```

11. Probemos lo que llevamos.

12. Dentro de **Pages** en la carpeta **Auth** creamos el componente **Logout**:

```
@page "/logout"
@inject ILoginService loginService
@inject NavigationManager navigationManager

<p>Cerrando sesión...</p>

@code {
    protected override async Task OnInitializedAsync()
    {
        await loginService.LogoutAsync();
        navigationManager.NavigateTo("/");
    }
```

```
}
```

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

Habilitando tokens en swagger

Este titulo se lo debemos a **José Rendon** que me explico como se hacia, gracias **Jose**!

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

```
builder.Services.AddSwaggerGen();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "Sales API", Version = "v1" });
    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = @"JWT Authorization header using the Bearer scheme. <br /> <br />
                        Enter 'Bearer' [space] and then your token in the text input below.<br /> <br />
                        Example: 'Bearer 12345abcdef'<br /> <br />",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement()
      {
        {
          new OpenApiSecurityScheme
          {
            Reference = new OpenApiReference
              {
                Type = ReferenceType.SecurityScheme,
                Id = "Bearer"
              },
              Scheme = "oauth2",
              Name = "Bearer",
              In = ParameterLocation.Header,
            },
            new List<string>()
        }
        });
});

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

15. Probamos y hacemos el **commit**.

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

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

```
[AllowAnonymous]
[HttpGet("combo")]
public async Task<ActionResult> GetCombo()
```

```
{
    return Ok(await _context.Countries.ToListAsync());
}
```

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

```
[AllowAnonymous]
[HttpGet("combo/{countryId:int}")]
public async Task<ActionResult> GetCombo(int countryId)
{
    return Ok(await _context.States
        .Where(x => x.CountryId == countryId)
        .ToListAsync());
}
```

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

```
[AllowAnonymous]
[HttpGet("combo/{stateId:int}")]
public async Task<ActionResult> GetCombo(int stateId)
{
    return Ok(await _context.Cities
        .Where(x => x.StateId == stateId)
        .ToListAsync());
}
```

4. Modificamos el **Register.razor**:

```
…
<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>
…
@code {
    private UserDTO userDTO = new();
    private List<Country>? countries;
    private List<State>? states;
    private List<City>? cities;

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

    private async Task CountryChangedAsync(ChangeEventArgs e)
    {
        var selectedCountry = Convert.ToInt32(e.Value!);
        await LoadStatesAsyn(selectedCountry);
    }

    private async Task StateChangedAsync(ChangeEventArgs e)
    {
        var selectedState = Convert.ToInt32(e.Value!);
        await LoadCitiesAsyn(selectedState);
    }

    private async Task LoadCountriesAsync()
    {
        var responseHttp = await repository.Get<List<Country>>("/api/countries/combo");
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
```

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

        countries = responseHttp.Response;
    }

    private async Task LoadStatesAsyn(int countryId)
    {
        var responseHttp = await repository.Get<List<State>>($"/api/states/combo/{countryId}");
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        states = responseHttp.Response;
    }

    private async Task LoadCitiesAsyn(int stateId)
    {
        var responseHttp = await repository.Get<List<City>>($"/api/cities/combo/{stateId}");
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        cities = responseHttp.Response;
    }

    private async Task CreteUserAsync()
…
```

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

Mejorando un poco la interfaz de usuario

Esta sección se la debo a **Jimy Davila** que me hizo las sugerencia y me paso parte del código que acá les comparto, como siempre gracias **Jimy**!

6. Primero vamos a agregar estas líneas a nuestro **app.css**:

```
.spinner {
    border: 16px solid silver;
    border-top: 16px solid #337AB7;
    border-radius: 50%;
    width: 80px;
    height: 80px;
    animation: spin 700ms linear infinite;
    top: 40%;
    left: 55%;
```

```
    position: absolute;
}

@keyframes spin {
    0% {
        transform: rotate(0deg)
    }

    100% {
        transform: rotate(360deg)
    }
}
```

7. Luego modificamos nuestro **CountriesIndex**:

```
…
@if (Countries is null)
{
    <div class="spinner"/>
}
else
{
    <GenericList MyList="Countries">
        <RecordsComplete>
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-globe"></i> Países
                        <a class="btn btn-sm btn-primary float-end" href="/countries/create"><i class="oi oi-plus"></i> Adicionar País</a>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
                        <div>
                            <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar país..." @bind-value="Filter" />
                        </div>
                        <div class="mx-1">
                            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi oi-layers" /> Filtrar</button>
                            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban" /> Limpiar</button>
                        </div>
                    </div>

                    <Pagination CurrentPage="currentPage"
                                TotalPages="totalPages"
                                SelectedPage="SelectedPage" />

                    <table class="table table-striped">
                        <thead>
                            <tr>
                                <th>País</th>
```

```
                        <th style="width:220px">Departamentos / Estados</th>
                        <th style="width:310px"></th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var country in Countries!)
                    {
                        <tr>
                            <td>
                                @country.Name
                            </td>
                            <td>
                                @country.StatesNumber
                            </td>
                            <td>
                                <a href="/countries/details/@country.Id" class="btn btn-info btn-sm"><i class="oi oi-list" />
Detalles</a>
                                <a href="/countries/edit/@country.Id" class="btn btn-warning btn-sm"><i class="oi oi-pencil" />
Editar</a>
                                <button class="btn btn-danger btn-sm" @onclick=@(() => DeleteAsync(country.Id))><i class="oi
oi-trash" /> Borrar</button>
                            </td>
                        </tr>
                    }
                </tbody>
            </table>
        </div>
    </div>
  </RecordsComplete>
</GenericList>
}
…
```

8. Replica el cambio para el resto de la solución. Si quieres una lista de íconos que puedes usar te dejo este link: https://kordamp.org/ikonli/cheat-sheet-openiconic.html

9. Este es un ejemplo de como puede quedar la página de **Register**:

```
<EditForm Model="userDTO" OnValidSubmit="CreteUserAsync">
  <DataAnnotationsValidator />

  <div class="card">
    <div class="card-header">
      <span>
        <i class="oi oi-person" /> Registrar Nuevo  Usuario
        <button class="btn btn-sm btn-primary float-end" type="submit"><i class="oi oi-check" /> Registrar</button>
      </span>
    </div>
    <div class="card-body">
      <div class="row">
        <div class="col-6">
          <div class="mb-3">
            <label>Nombres:</label>
            <div>
```

```
            <InputText class="form-control" @bind-Value="@userDTO.FirstName" />
            <ValidationMessage For="@(() => userDTO.FirstName)" />
        </div>
    </div>
    <div class="mb-3">
        <label>Apellidos:</label>
        <div>
            <InputText class="form-control" @bind-Value="@userDTO.LastName" />
            <ValidationMessage For="@(() => userDTO.LastName)" />
        </div>
    </div>
    <div class="mb-3">
        <label>Documento:</label>
        <div>
            <InputText class="form-control" @bind-Value="@userDTO.Document" />
            <ValidationMessage For="@(() => userDTO.Document)" />
        </div>
    </div>
    <div class="mb-3">
        <label>Teléfono:</label>
        <div>
            <InputText class="form-control" @bind-Value="@userDTO.PhoneNumber" />
            <ValidationMessage For="@(() => userDTO.PhoneNumber)" />
        </div>
    </div>
    <div class="mb-3">
        <label>Dirección:</label>
        <div>
            <InputText class="form-control" @bind-Value="@userDTO.Address" />
            <ValidationMessage For="@(() => userDTO.Address)" />
        </div>
    </div>
    <div class="mb-3">
        <label>Email:</label>
        <div>
            <InputText class="form-control" @bind-Value="@userDTO.Email" />
            <ValidationMessage For="@(() => userDTO.Email)" />
        </div>
    </div>
</div>
<div class="col-6">
    <div class="mb-3">
        <label>País:</label>
        <div>
            <select class="form-select" @onchange="CountryChangedAsync">
                <option value="0">-- Seleccione un país --</option>
                @if (countries is not null)
                {
                    @foreach (var country in countries)
                    {
                        <option value="@country.Id">@country.Name</option>
                    }
                }
            </select>
```

```
    </div>
</div>
<div class="mb-3">
    <label>Estado/Departamento:</label>
    <div>
        <select class="form-select" @onchange="StateChangedAsync">
            <option value="0">-- Seleccione un estado/departamento --</option>
            @if (states is not null)
            {
                @foreach (var state in states)
                {
                    <option value="@state.Id">@state.Name</option>
                }
            }
        </select>
    </div>
</div>
<div class="mb-3">
    <label>Ciudad:</label>
    <div>
        <select class="form-select" @bind="userDTO.CityId">
            <option value="0">-- Seleccione una ciudad --</option>
            @if (cities is not null)
            {
                @foreach (var city in cities)
                {
                    <option value="@city.Id">@city.Name</option>
                }
            }
        </select>
        <ValidationMessage For="@(() => userDTO.CityId)" />
    </div>
</div>
<div class="mb-3">
    <label>Foto:</label>
    <div>
        <InputText class="form-control" @bind-Value="@userDTO.Photo" />
        <ValidationMessage For="@(() => userDTO.Photo)" />
    </div>
</div>
<div class="mb-3">
    <label>Contraseña:</label>
    <div>
        <InputText type="password" class="form-control" @bind-Value="@userDTO.Password" />
        <ValidationMessage For="@(() => userDTO.Password)" />
    </div>
</div>
<div class="mb-3">
    <label>Confirmación de contraseña:</label>
    <div>
        <InputText type="password" class="form-control" @bind-Value="@userDTO.PasswordConfirm" />
        <ValidationMessage For="@(() => userDTO.PasswordConfirm)" />
    </div>
</div>
```

```
            </div>
        </div>
    </div>
  </div>
</EditForm>
```

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

```
@page "/Login"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager
@inject ILoginService loginService

<div class="row">
  <div class="col-md-4  offset-md-4">
    <EditForm Model="loginDTO" OnValidSubmit="LoginAsync">
      <DataAnnotationsValidator />

      <div class="card bg-light">
        <div class="card-header justify-content-center">
          <span>
            <i class="oi oi-account-login" /> Iniciar Sesión
            <button class="btn btn-sm btn-primary float-end" type="submit"><i class="oi oi-check" /> Iniciar
Sesión</button>
          </span>
        </div>
        <div class="card-body">
          <div class="mb-3">
            <label>Email:</label>
            <div>
              <InputText class="form-control" @bind-Value="@loginDTO.Email" />
              <ValidationMessage For="@(() => loginDTO.Email)" />
            </div>
          </div>
          <div class="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>
```

11. También cambiemos todos los ** Cargando… ** por ** **.

12. Hacemos el **commit**.

Almacenando la foto del usuario

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

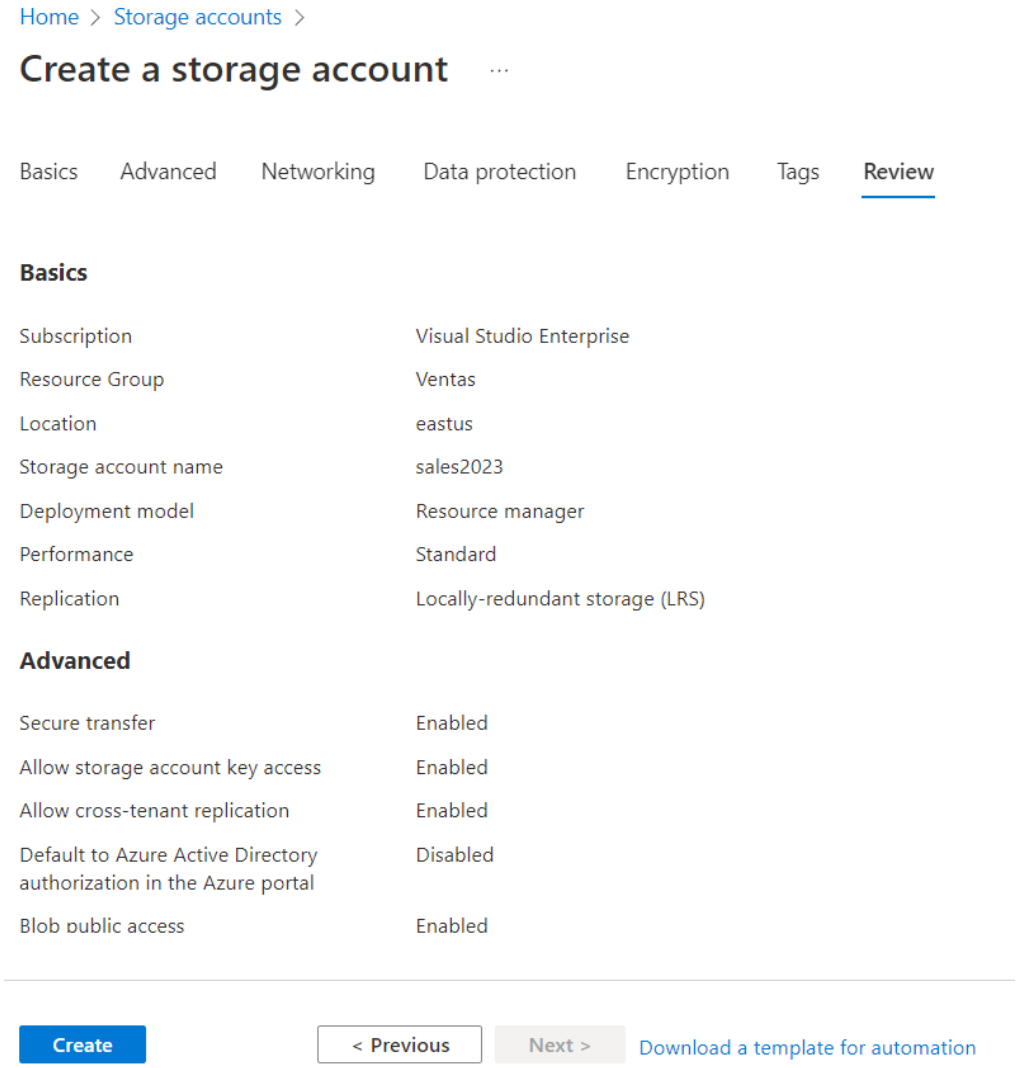
```
<div>
    <label>@Label</label>
    <div>
        <InputFile OnChange="OnChange" accept=".jpg,.jpeg,.png" />
    </div>
</div>

<div>
    @if (imageBase64 is not null)
    {
        <div>
            <div style="margin: 10px">
                <img src=" @imageBase64" style="width:400px" />
            </div>
        </div>
    }

    @if (ImageURL is not null)
    {
        <div>
            <div style="margin: 10px">
                <img src="@ImageURL" style="width:400px" />
            </div>
        </div>
    }
</div>

@code {
    [Parameter] public string Label { get; set; } = "Imagen";
    [Parameter] public string? ImageURL { get; set; }
    [Parameter] public EventCallback<string> ImageSelected { get; set; }
    private string? imageBase64;

    async Task OnChange(InputFileChangeEventArgs e)
    {
        var imagenes = e.GetMultipleFiles();

        foreach (var imagen in imagenes)
        {
            var arrBytes = new byte[imagen.Size];
            await imagen.OpenReadStream().ReadAsync(arrBytes);
            imageBase64 = Convert.ToBase64String(arrBytes);
            ImageURL = null;
            await ImageSelected.InvokeAsync(imageBase64);
            StateHasChanged();
        }
    }
}
```

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

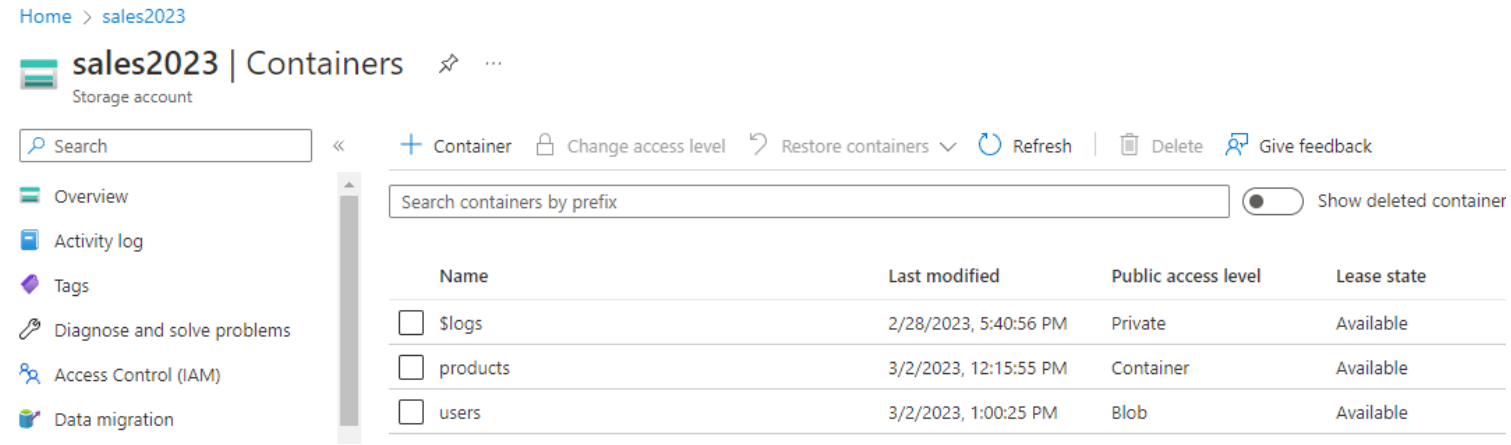
```
…
                <div class="mb-3">
                    <label>Confirmación de contraseña:</label>
                    <div>
                        <InputText type="password" class="form-control" @bind-Value="@userDTO.PasswordConfirm" />
                        <ValidationMessage For="@(() => userDTO.PasswordConfirm)" />
                    </div>
                </div>
                <div class="mb-3">
                    <InputImg Label="Foto" ImageSelected="ImageSelected" ImageURL="@imageUrl" />
                </div>
            </div>
        </div>
    </div>
</EditForm>

@code {
    private UserDTO userDTO = new();
    private List<Country>? countries;
    private List<State>? states;
    private List<City>? cities;
    private bool loading;
    private string? imageUrl;

    protected override async Task OnInitializedAsync()
    {
        await LoadCountriesAsync();

        if (!string.IsNullOrEmpty(userDTO.Photo))
        {
            imageUrl = userDTO.Photo;
            userDTO.Photo = null;
        }
    }

    private void ImageSelected(string imagenBase64)
    {
        userDTO.Photo = imagenBase64;
        imageUrl = null;
    }
…
```

3. Probamos lo que llevamos hasta el momento.

4. Ahora vamos a crear el **blob** en **Azure**:



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



6. Luego que termine copiamos el connection string que necesitamos para acceder a nuestro blob storage, para mi ejemplo es:

DefaultEndpointsProtocol=https;AccountName=sales2023;AccountKey=qC+EUq97TPPgIh8Syt18jgnl4swmJNiaS4fZEWVUwHIzr21H0wVstqJ8t+8t8VHdL3ZvarbAOBiq+AStRAgUtA==;EndpointSuffix=core.windows.net

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

"ConnectionStrings": {

```
  "DockerConnection": "Data Source=.;Initial Catalog=Sales;User ID=sa;Password=Roger1974.;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False",
  "LocalConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=Sales2023;Trusted_Connection=True;MultipleActiveResultSets=true",
  "AzureStorage":
"DefaultEndpointsProtocol=https;AccountName=sales2023;AccountKey=qC+EUq97TPPgIh8Syt18jgnl4swmJNiaS4fZEW
VUwHlzr21H0wVstqJ8t+8t8VHdL3ZvarbAOBiq+AStRAgUtA==;EndpointSuffix=core.windows.net"
 },
```

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

```
namespace Sales.API.Helpers
{
  public interface IFileStorage
  {
    Task<string> SaveFileAsync(byte[] content, string extention, string containerName);

    Task RemoveFileAsync(string path, string nombreContenedor);

    async Task<string> EditFileAsync(byte[] content, string extention, string containerName, string path)
    {
      if (path is not null)
      {
        await RemoveFileAsync(path, containerName);
      }

      return await SaveFileAsync(content, extention, containerName);
    }
  }
}
```

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

```
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;

namespace Sales.API.Helpers
{
  public class FileStorage : IFileStorage
  {
    private readonly string connectionString;
    public FileStorage(IConfiguration configuration)
    {
      connectionString = configuration.GetConnectionString("AzureStorage")!;
    }

    public async Task RemoveFileAsync(string path, string containerName)
    {
      var client = new BlobContainerClient(connectionString, containerName);
      await client.CreateIfNotExistsAsync();
      var fileName = Path.GetFileName(path);
      var blob = client.GetBlobClient(fileName);
      await blob.DeleteIfExistsAsync();
    }
```

```
        public async Task<string> SaveFileAsync(byte[] content, string extention, string containerName)
        {
            var client = new BlobContainerClient(connectionString, containerName);
            await client.CreateIfNotExistsAsync();
            client.SetAccessPolicy(PublicAccessType.Blob);
            var fileName = $"{Guid.NewGuid()}{extention}";
            var blob = client.GetBlobClient(fileName);

            using (var ms = new MemoryStream(content))
            {
                await blob.UploadAsync(ms);
            }

            return blob.Uri.ToString();
        }
    }
}
```

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

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

11. Modificamos el **AccountsController**:

```
[ApiController]
[Route("/api/accounts")]
public class AccountsController : ControllerBase
{
    private readonly IUserHelper _userHelper;
    private readonly IConfiguration _configuration;
    private readonly IFileStorage _fileStorage;
    private readonly string _container;

    public AccountsController(IUserHelper userHelper, IConfiguration configuration, IFileStorage fileStorage)
    {
        _userHelper = userHelper;
        _configuration = configuration;
        _fileStorage = fileStorage;
        _container = "users";
    }

    [HttpPost("CreateUser")]
    public async Task<ActionResult> CreateUser([FromBody] UserDTO model)
    {
        User user = model;
        if(!string.IsNullOrEmpty(model.Photo))
        {
            var photoUser = Convert.FromBase64String(model.Photo);
            model.Photo = await _fileStorage.SaveFileAsync(photoUser, ".jpg", _container);
        }

        var result = await _userHelper.AddUserAsync(user, model.Password);
        if (result.Succeeded)
```

```
        {
            await _userHelper.AddUserToRoleAsync(user, user.UserType.ToString());
            return Ok(BuildToken(user));
        }

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

12. Modificamos el **AuthLinks.razor**:

```
<AuthorizeView>
    <Authorized>
        <span>Hola, @context.User.Identity!.Name</span>
        @if (!string.IsNullOrEmpty(photoUser))
        {
            <div class="mx-2">
                <img src="@photoUser" width="50" height="50" style="border-radius:50%" />
            </div>
        }
        <a href="Logout" class="nav-link btn btn-link">Cerrar Sesión</a>
    </Authorized>
    <NotAuthorized>
        <a href="Register" class="nav-link btn btn-link">Registro</a>
        <a href="Login" class="nav-link btn btn-link">Iniciar Sesión</a>
    </NotAuthorized>
</AuthorizeView>

@code {
    private string? photoUser;

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

    protected async override Task OnParametersSetAsync()
    {
        var authenticationState = await authenticationStateTask;
        var claims = authenticationState.User.Claims.ToList();
        var photoClaim = claims.FirstOrDefault(x => x.Type == "Photo");
        if (photoClaim is not null)
        {
            photoUser = photoClaim.Value;
        }
    }
}
```

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

Editando el usuario

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

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

Task<IdentityResult> UpdateUserAsync(User user);
```

```
Task<User> GetUserAsync(Guid userId);
```

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

```
public async Task<User> GetUserAsync(string email)
{
    var user = await _context.Users
        .Include(u => u.City!)
        .ThenInclude(c => c.State!)
        .ThenInclude(s => s.Country!)
        .FirstOrDefaultAsync(x => x.Email == email);
    return user!;
}

public async Task<User> GetUserAsync(Guid userId)
{
    var user = await _context.Users
        .Include(u => u.City!)
        .ThenInclude(c => c.State!)
        .ThenInclude(s => s.Country!)
        .FirstOrDefaultAsync(x => x.Id == userId.ToString());
    return user!;
}

public async Task<IdentityResult> ChangePasswordAsync(User user, string currentPassword, string newPassword)
{
    return await _userManager.ChangePasswordAsync(user, currentPassword, newPassword);
}

public async Task<IdentityResult> UpdateUserAsync(User user)
{
    return await _userManager.UpdateAsync(user);
}
```

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

```
[HttpPut]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult> Put(User user)
{
    try
    {
        if (!string.IsNullOrEmpty(user.Photo))
        {
            var photoUser = Convert.FromBase64String(user.Photo);
            user.Photo = await _fileStorage.SaveFileAsync(photoUser, ".jpg", _container);
        }

        var currentUser = await _userHelper.GetUserAsync(user.Email!);
        if (currentUser == null)
        {
            return NotFound();
        }
```

```
        currentUser.Document = user.Document;
        currentUser.FirstName = user.FirstName;
        currentUser.LastName = user.LastName;
        currentUser.Address = user.Address;
        currentUser.PhoneNumber = user.PhoneNumber;
        currentUser.Photo = !string.IsNullOrEmpty(user.Photo) && user.Photo != currentUser.Photo ? user.Photo :
currentUser.Photo;
        currentUser.CityId = user.CityId;

        var result = await _userHelper.UpdateUserAsync(currentUser);
        if (result.Succeeded)
        {
            return NoContent();
        }

        return BadRequest(result.Errors.FirstOrDefault());
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}

[HttpGet]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult> Get()
{
    return Ok(await _userHelper.GetUserAsync(User.Identity!.Name!));
}
```

17. Modificamos el **AuthLinks**:

```
<Authorized>
    Hola, <a href="EditUser" class="nav-link btn btn-link">@context.User.Identity!.Name</a>
        @if (!string.IsNullOrEmpty(photoUser))
        {
            <div class="mx-2">
                <img src="@photoUser" width="50" height="50" style="border-radius:50%" />
            </div>
        }
    <a href="Logout" class="nav-link btn btn-link">Cerrar Sesión</a>
</Authorized>
```

18. Creamos el **EditUser.razor**:

```
@page "/EditUser"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager
@inject ILoginService loginService

@if (user is null)
{
```

```
    <div class="spinner" />
}
else
{
    <EditForm Model="user" OnValidSubmit="SaveUserAsync">
        <DataAnnotationsValidator />

        <div class="card">
            <div class="card-header">
                <span>
                    <i class="oi oi-person" /> Editar Usuario
                    <a class="btn btn-sm btn-secondary float-end" href="/changePassword"><i class="oi oi-key" /> Cambiar
Contraseña</a>
                    <button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Guardar
Cambios</button>
                </span>
            </div>
            <div class="card-body">
                <div class="row">
                    <div class="col-6">
                        <div class="mb-3">
                            <label>Nombres:</label>
                            <div>
                                <InputText class="form-control" @bind-Value="@user.FirstName" />
                                <ValidationMessage For="@(() => user.FirstName)" />
                            </div>
                        </div>
                        <div class="mb-3">
                            <label>Apellidos:</label>
                            <div>
                                <InputText class="form-control" @bind-Value="@user.LastName" />
                                <ValidationMessage For="@(() => user.LastName)" />
                            </div>
                        </div>
                        <div class="mb-3">
                            <label>Documento:</label>
                            <div>
                                <InputText class="form-control" @bind-Value="@user.Document" />
                                <ValidationMessage For="@(() => user.Document)" />
                            </div>
                        </div>
                        <div class="mb-3">
                            <label>Teléfono:</label>
                            <div>
                                <InputText class="form-control" @bind-Value="@user.PhoneNumber" />
                                <ValidationMessage For="@(() => user.PhoneNumber)" />
                            </div>
                        </div>
                        <div class="mb-3">
                            <label>Dirección:</label>
                            <div>
                                <InputText class="form-control" @bind-Value="@user.Address" />
                                <ValidationMessage For="@(() => user.Address)" />
                            </div>
```

```
                </div>
            </div>
            <div class="col-6">
                <div class="mb-3">
                    <label>País:</label>
                    <div>
                        <select class="form-select" @onchange="CountryChangedAsync">
                            <option value="0">-- Seleccione un país --</option>
                            @if (countries is not null)
                            {
                                @foreach (var country in countries)
                                {
                                    <option value="@country.Id" selected="@(country.Id ==
user.City!.State!.Country!.Id)">@country.Name</option>
                                }
                            }
                        </select>
                    </div>
                </div>
                <div class="mb-3">
                    <label>Estado/Departamento:</label>
                    <div>
                        <select class="form-select" @onchange="StateChangedAsync">
                            <option value="0">-- Seleccione un estado/departamento --</option>
                            @if (states is not null)
                            {
                                @foreach (var state in states)
                                {
                                    <option value="@state.Id" selected="@(state.Id ==
user.City!.State!.Id)">@state.Name</option>
                                }
                            }
                        </select>
                    </div>
                </div>
                <div class="mb-3">
                    <label>Ciudad:</label>
                    <div>
                        <select class="form-select" @bind="user.CityId">
                            <option value="0">-- Seleccione una ciudad --</option>
                            @if (cities is not null)
                            {
                                @foreach (var city in cities)
                                {
                                    <option value="@city.Id" selected="@(city.Id == user.City!.Id)">@city.Name</option>
                                }
                            }
                        </select>
                        <ValidationMessage For="@(() => user.CityId)" />
                    </div>
                </div>
                <div class="mb-3">
                    <InputImg Label="Foto" ImageSelected="ImageSelected" ImageURL="@imageUrl" />
                </div>
```

```
                </div>
            </div>
        </div>
    </div>
</EditForm>
}

@code {
    private User? user;
    private List<Country>? countries;
    private List<State>? states;
    private List<City>? cities;
    private string? imageUrl;

    protected override async Task OnInitializedAsync()
    {
        await LoadUserAsyc();
        await LoadCountriesAsync();
        await LoadStatesAsyn(user!.City!.State!.Country!.Id);
        await LoadCitiesAsyn(user!.City!.State!.Id);

        if (!string.IsNullOrEmpty(user!.Photo))
        {
            imageUrl = user.Photo;
            user.Photo = null;
        }

    }

    private async Task LoadUserAsyc()
    {
        var responseHTTP = await repository.Get<User>($"/api/accounts");
        if (responseHTTP.Error)
        {
            if (responseHTTP.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/");
                return;
            }
            var messageError = await responseHTTP.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", messageError, SweetAlertIcon.Error);
            return;
        }
        user = responseHTTP.Response;
    }

    private void ImageSelected(string imagenBase64)
    {
        user!.Photo = imagenBase64;
        imageUrl = null;
    }

    private async Task CountryChangedAsync(ChangeEventArgs e)
    {
```

```
        var selectedCountry = Convert.ToInt32(e.Value!);
        await LoadStatesAsyn(selectedCountry);
    }

    private async Task StateChangedAsync(ChangeEventArgs e)
    {
        var selectedState = Convert.ToInt32(e.Value!);
        await LoadCitiesAsyn(selectedState);
    }

    private async Task LoadCountriesAsync()
    {
        var responseHttp = await repository.Get<List<Country>>("/api/countries/combo");
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        countries = responseHttp.Response;
    }

    private async Task LoadStatesAsyn(int countryId)
    {
        var responseHttp = await repository.Get<List<State>>($"/api/states/combo/{countryId}");
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        states = responseHttp.Response;
    }

    private async Task LoadCitiesAsyn(int stateId)
    {
        var responseHttp = await repository.Get<List<City>>($"/api/cities/combo/{stateId}");
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        cities = responseHttp.Response;
    }

    private async Task SaveUserAsync()
    {
        var responseHttp = await repository.Put<User>("/api/accounts", user!);
        if (responseHttp.Error)
        {
```

```
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        navigationManager.NavigateTo("/");
    }
}
```

19. Probamos.

Cambiando password del usuario

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

```
using System.ComponentModel.DataAnnotations;

namespace Sales.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!;
    }
}
```

2. En **Sales.API.Controllers** en el controlador **AccountsController** adicionamos este método:

```
[HttpPost("changePassword")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult> ChangePasswordAsync(ChangePasswordDTO model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

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

```
    if (user == null)
    {
        return NotFound();
    }

    var result = await _userHelper.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);
    if (!result.Succeeded)
    {
        return BadRequest(result.Errors.FirstOrDefault().Description);
    }

    return NoContent();
}
```

3. Dentro de **Sales.WEB.Pages** creamos el **ChangePassword.razor**:

```
@page "/changePassword"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager

@if (loading)
{
    <div class="spinner" />
}
<div class="row">
    <div class="col-6">
        <EditForm Model="changePasswordDTO" OnValidSubmit="ChangePasswordAsync">
            <DataAnnotationsValidator />
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-key" /> Cambiar Contraseña
                        <a class="btn btn-sm btn-success float-end" href="/editUser"><i class="oi oi-arrow-thick-left" />
Regresar</a>
                        <button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Guardar
Cambios</button>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-3">
                        <label>Contraseña actual:</label>
                        <div>
                            <InputText type="password" class="form-control"
@bind-Value="@changePasswordDTO.CurrentPassword" />
                            <ValidationMessage For="@(() => changePasswordDTO.CurrentPassword)" />
                        </div>
                    </div>
                    <div class="mb-3">
                        <label>Nueva contraseña:</label>
                        <div>
                            <InputText type="password" class="form-control"
@bind-Value="@changePasswordDTO.NewPassword" />
                            <ValidationMessage For="@(() => changePasswordDTO.CurrentPassword)" />
```

```
                    </div>
                </div>
                <div class="mb-3">
                    <label>Confirmación de nueva contraseña:</label>
                    <div>
                        <InputText type="password" class="form-control" @bind-Value="@changePasswordDTO.Confirm" />
                        <ValidationMessage For="@(() => changePasswordDTO.Confirm)" />
                    </div>
                </div>
            </div>
        </div>
    </EditForm>
  </div>
</div>

@code {
  private ChangePasswordDTO changePasswordDTO = new();
  private bool loading;

  private async Task ChangePasswordAsync()
  {
      loading = true;
      var responseHttp = await repository.Post("/api/accounts/changePassword", changePasswordDTO);
      if (responseHttp.Error)
      {
          var message = await responseHttp.GetErrorMessageAsync();
          await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
          loading = false;
          return;
      }

      loading = false;
      navigationManager.NavigateTo("/editUser");
      var toast = sweetAlertService.Mixin(new SweetAlertOptions
          {
              Toast = true,
              Position = SweetAlertPosition.TopEnd,
              ShowConfirmButton = true,
              Timer = 5000
          });
      await toast.FireAsync(icon: SweetAlertIcon.Success, message: "Contraseña cambiada con éxito.");
  }
}
```

4. Probamos y hacemos el **commit**.

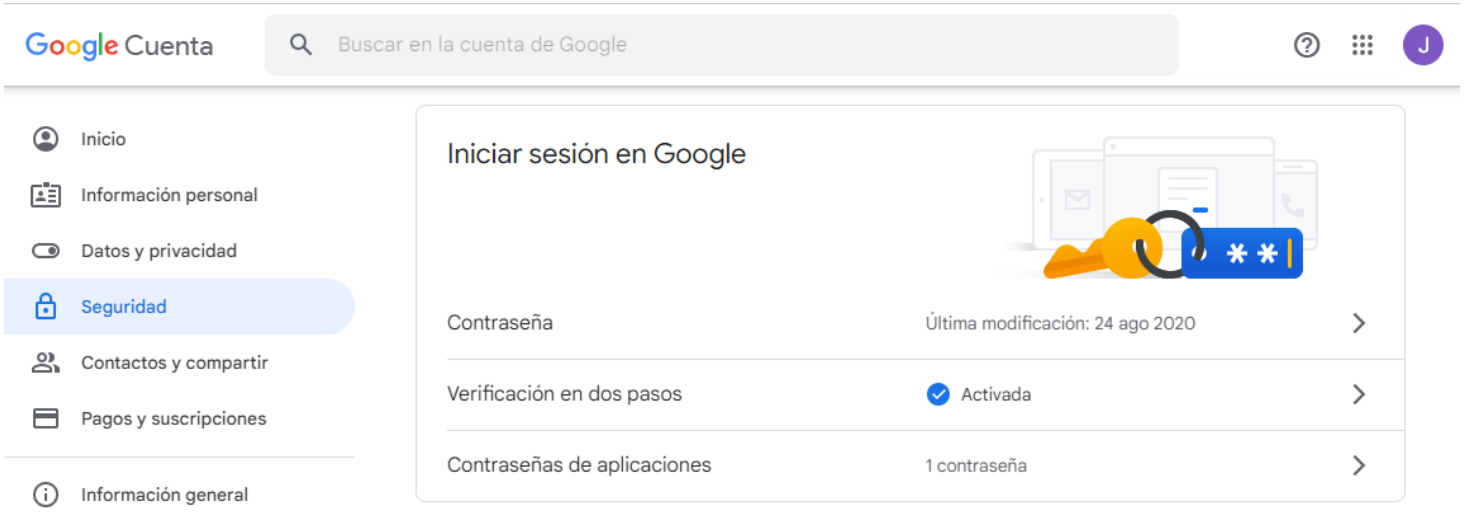
Confirmar el registro de usuarios

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

```
builder.Services.AddIdentity<User, IdentityRole>(x =>
{
  x.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;
  x.SignIn.RequireConfirmedEmail = true;
```

```
    x.User.RequireUniqueEmail = true;
    x.Password.RequireDigit = false;
    x.Password.RequiredUniqueChars = 0;
    x.Password.RequireLowercase = false;
    x.Password.RequireNonAlphanumeric = false;
    x.Password.RequireUppercase = false;
    x.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
    x.Lockout.MaxFailedAccessAttempts = 3;
    x.Lockout.AllowedForNewUsers = true;
})
    .AddEntityFrameworkStores<DataContext>()
    .AddDefaultTokenProviders();
```

2. Verificamos que la cuenta de Gmail con la que vamos a mandar los correos tenga lo siguiente:



3. Adicionamos estos parámetros a la configuración del **API**:

```
  "Mail": {
    "From": "onsalezulu@gmail.com",
    "Name": "Soporte Sales",
    "Smtp": "smtp.gmail.com",
    "Port": 587,
    "Password": "nniufszzppfuzhxe"
  },
  "UrlWEB": "localhost:7175"
```

Nota: reemplazar el 7175 por el puerto donde sale tu App WEB, y reemplazar el password por el generado de tu cuenta.

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

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

```
public interface IMailHelper
{
    Response SendMail(string toName, string toEmail, string subject, string body);
}
```

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

```
using MailKit.Net.Smtp;
using MimeKit;
using Sales.Shared.Responses;

namespace Sales.API.Helpers
{
    public class MailHelper : IMailHelper
    {
        private readonly IConfiguration _configuration;

        public MailHelper(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public Response SendMail(string toName, string toEmail, string subject, string body)
        {
            try
            {
                var from = _configuration["Mail:From"];
                var name = _configuration["Mail:Name"];
                var smtp = _configuration["Mail:Smtp"];
                var port = _configuration["Mail:Port"];
                var password = _configuration["Mail:Password"];

                var message = new MimeMessage();
                message.From.Add(new MailboxAddress(name, from));
                message.To.Add(new MailboxAddress(toName, toEmail));
                message.Subject = subject;
                BodyBuilder bodyBuilder = new BodyBuilder
                {
                    HtmlBody = body
                };
                message.Body = bodyBuilder.ToMessageBody();

                using (var client = new SmtpClient())
                {
                    client.Connect(smtp, int.Parse(port!), false);
                    client.Authenticate(from, password);
                    client.Send(message);
                    client.Disconnect(true);
                }

                return new Response { IsSuccess = true };

            }
            catch (Exception ex)
            {
                return new Response
                {
                    IsSuccess = false,
                    Message = ex.Message,
```

```
                Result = ex
            };
        }
    }
}
```

7. Configuramos la inyección del servicio:

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

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

```
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);
}
```

9. Modificamos el método **CreateUser** del controlador **AccountsController** (primero inyectamos el **IMailHelper**):

```
[HttpPost("CreateUser")]
public async Task<ActionResult> CreateUser([FromBody] UserDTO model)
{
    User user = model;
    if (!string.IsNullOrEmpty(model.Photo))
    {
        var photoUser = Convert.FromBase64String(model.Photo);
        model.Photo = await _fileStorage.SaveFileAsync(photoUser, ".jpg", _container);
    }

    var result = await _userHelper.AddUserAsync(user, model.Password);
    if (result.Succeeded)
    {
        await _userHelper.AddUserToRoleAsync(user, user.UserType.ToString());

        var myToken = await _userHelper.GenerateEmailConfirmationTokenAsync(user);
        var tokenLink = Url.Action("ConfirmEmail", "accounts", new
        {
            userid = user.Id,
            token = myToken
        }, HttpContext.Request.Scheme, _configuration["UrlWEB"]);

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

```
            $"Saless- Confirmación de cuenta",
            $"<h1>Sales - Confirmación de cuenta</h1>" +
            $"<p>Para habilitar el usuario, por favor hacer clic 'Confirmar Email':</p>" +
            $"<b><a href ={tokenLink}>Confirmar Email</a></b>");

        if (response.IsSuccess)
        {
            return NoContent();
        }

        return BadRequest(response.Message);
    }

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

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

```
[HttpGet("ConfirmEmail")]
public async Task<ActionResult> ConfirmEmailAsync(string userId, string token)
{
    token = token.Replace(" ", "+");
    var user = await _userHelper.GetUserAsync(new Guid(userId));
    if (user == null)
    {
        return NotFound();
    }

    var result = await _userHelper.ConfirmEmailAsync(user, token);
    if (!result.Succeeded)
    {
        return BadRequest(result.Errors.FirstOrDefault());
    }

    return NoContent();
}
```

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

```
[HttpPost("Login")]
public async Task<ActionResult> Login([FromBody] LoginDTO model)
{
    var result = await _userHelper.LoginAsync(model);
    if (result.Succeeded)
    {
        var user = await _userHelper.GetUserAsync(model.Email);
        return Ok(BuildToken(user));
    }

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

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

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

12. Agregamos este método al **IRepository**:

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

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

```
public async Task<HttpResponseWrapper<object>> Get(string url)
{
    var responseHTTP = await _httpClient.GetAsync(url);
    return new HttpResponseWrapper<object>(null, !responseHTTP.IsSuccessStatusCode, responseHTTP);
}
```

14. Dentro de **Pages/Auth** creamos la página **ConfirmEmail.razor**:

```
@page "/api/accounts/ConfirmEmail"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager

<h3>Confirmación de email</h3>

<p>Presione el botón para confirmar su cuenta</p>
<button class="btn btn-primary" @onclick="ConfirmAccountAsync">Confirmar Cuenta</button>

@code {
    private string? message;

    [Parameter]
    [SupplyParameterFromQuery]
    public string UserId { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Token { get; set; } = "";

    protected async Task ConfirmAccountAsync()
    {
        var responseHttp = await repository.Get($"/api/accounts/ConfirmEmail/?userId={UserId}&token={Token}");
        if (responseHttp.Error)
        {
            message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            navigationManager.NavigateTo("/");
        }
```

```
        else
        {
            await sweetAlertService.FireAsync("Confirmación", "Gracias por confirmar su email, ahora puedes ingresar al sistema.", SweetAlertIcon.Info);
            navigationManager.NavigateTo("/Login");
        }
    }
}
```

15. Borramos los usuarios de la base de datos.

16. Modificamos el alimentador de la base de datos:

```
private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string phone, string address, UserType userType)
{
    var user = await _userHelper.GetUserAsync(email);

    if (user == null)
    {
        var city = await _context.Cities.FirstOrDefaultAsync(x => x.Name == "Medellín");
        if (city == null)
        {
            city = await _context.Cities.FirstOrDefaultAsync();
        }

        user = new User
        {
            FirstName = firstName,
            LastName = lastName,
            Email = email,
            UserName = email,
            PhoneNumber = phone,
            Address = address,
            Document = document,
            City = city,
            UserType = userType,
        };

        await _userHelper.AddUserAsync(user, "123456");
        await _userHelper.AddUserToRoleAsync(user, userType.ToString());

        var token = await _userHelper.GenerateEmailConfirmationTokenAsync(user);
        await _userHelper.ConfirmEmailAsync(user, token);
    }

    return user;
}
```

17. Modificamos el **Register.razor**:

```
private async Task CreteUserAsync()
{
    loading = true;
```

```
    userDTO.UserName = userDTO.Email;
    userDTO.UserType = UserType.User;
    var responseHttp = await repository.Post<UserDTO>("/api/accounts/CreateUser", userDTO);
    if (responseHttp.Error)
    {
        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        loading = false;
        return;
    }

    loading = false;
    await sweetAlertService.FireAsync("Confirmación", "Su cuenta ha sido creada con éxito. Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario.", SweetAlertIcon.Info);
    navigationManager.NavigateTo("/");
}
```

18. Probamos y hacemos el **commit**.

Reenviar correo de confirmación

1. En **Sales.Shared.DTOs** creamos la clase **EmailDTO**:

```
using System.ComponentModel.DataAnnotations;

namespace Sales.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!;
    }
}
```

2. En el **API** creamos este método en el **AccountsController**:

```
[HttpPost("ResedToken")]
public async Task<ActionResult> ResedToken([FromBody] EmailDTO model)
{
    User user = await _userHelper.GetUserAsync(model.Email);
    if (user == null)
    {
        return NotFound();
    }

    var myToken = await _userHelper.GenerateEmailConfirmationTokenAsync(user);
    var tokenLink = Url.Action("ConfirmEmail", "accounts", new
    {
        userid = user.Id,
        token = myToken
    }, HttpContext.Request.Scheme, _configuration["UrlWEB"]);
```

```
    var response = _mailHelper.SendMail(user.FullName, user.Email!,
        $"Saless- Confirmación de cuenta",
        $"<h1>Sales - Confirmación de cuenta</h1>" +
        $"<p>Para habilitar el usuario, por favor hacer clic 'Confirmar Email':</p>" +
        $"<b><a href ={tokenLink}>Confirmar Email</a></b>");

    if (response.IsSuccess)
    {
        return NoContent();
    }

    return BadRequest(response.Message);
}
```

3. Modificamos nuestro **Login.razor**:

```
<div class="row">
    <div class="col-md-4  offset-md-4">
        <EditForm Model="loginDTO" OnValidSubmit="LoginAsync">
            <DataAnnotationsValidator />

            <div class="card bg-light">
                <div class="card-header justify-content-center">
                    <span>
                        <i class="oi oi-account-login" /> Iniciar Sesión
                        <button class="btn btn-sm btn-primary float-end" type="submit"><i class="oi oi-check" /> Iniciar
Sesión</button>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-3">
                        <label>Email:</label>
                        <div>
                            <InputText class="form-control" @bind-Value="@loginDTO.Email" />
                            <ValidationMessage For="@(() => loginDTO.Email)" />
                        </div>
                    </div>
                    <div class="mb-3">
                        <label>Contraseña:</label>
                        <div>
                            <InputText type="password" class="form-control" @bind-Value="@loginDTO.Password" />
                            <ValidationMessage For="@(() => loginDTO.Password)" />
                        </div>
                    </div>
                </div>
                <div class="card-footer">
                    <a class="bbtn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a>
                </div>
            </div>
        </EditForm>
    </div>
</div>
```

4. Dentro de **Pages/Auth** creamos el **ResendConfirmationEmailToken.razor**:

```
@page "/ResendToken"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager

@if (loading)
{
    <div class="spinner" />
}
<div class="row">
    <div class="col-6">
        <EditForm Model="emailDTO" OnValidSubmit="ResendConfirmationEmailTokenAsync">
            <DataAnnotationsValidator />
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-key" /> Reenviar correo de confirmación de contraseña
                        <button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-loop-square" />
Reenviar</button>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-3">
                        <label>Email:</label>
                        <div>
                            <InputText class="form-control" @bind-Value="@emailDTO.Email" />
                            <ValidationMessage For="@(() => emailDTO.Email)" />
                        </div>
                    </div>
                </div>
            </div>
        </EditForm>
    </div>
</div>

@code {
    private EmailDTO emailDTO = new();
    private bool loading;

    private async Task ResendConfirmationEmailTokenAsync()
    {
        loading = true;
        var responseHttp = await repository.Post("/api/accounts/ResedToken", emailDTO);
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            loading = false;
            return;
        }

        loading = false;
```

```
        await sweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario.", SweetAlertIcon.Info);
        navigationManager.NavigateTo("/");
    }
}
```

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

Actualización de la foto del usuario luego de editar usuario

Este título se lo debemos a **Lou González** que me compartió el código. Gracias **Lou**!

1. Modificamos el **PUT** del **AccountsController**:

```
var result = await _userHelper.UpdateUserAsync(currentUser);
if (result.Succeeded)
{
    return Ok(BuildToken(currentUser));
}
```

2. Agregamos este método al **IRepository**:

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

3. Y su implementación en el **Repository**:

```
public async Task<HttpResponseWrapper<TResponse>> Put<T, TResponse>(string url, T model)
{
    var messageJSON = JsonSerializer.Serialize(model);
    var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");
    var responseHttp = await _httpClient.PutAsync(url, messageContent);
    if (responseHttp.IsSuccessStatusCode)
    {
        var response = await UnserializeAnswer<TResponse>(responseHttp, _jsonDefaultOptions);
        return new HttpResponseWrapper<TResponse>(response, false, responseHttp);
    }
    return new HttpResponseWrapper<TResponse>(default, !responseHttp.IsSuccessStatusCode, responseHttp);
}
```

4. Modificamos el **EditUser**:

```
private async Task SaveUserAsync()
{
    var responseHttp = await repository.Put<User, TokenDTO>("/api/accounts", user!);
    if (responseHttp.Error)
    {
        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    await loginService.LoginAsync(responseHttp.Response!.Token);
    navigationManager.NavigateTo("/");
}
```

5. Probamos y hacemos el **Commit**.

Recuperación de contraseña

1. Modificamos el **Login.razor**:

```
<div class="card-footer">
    <p><a class="bbtn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a></p>
    <p><a class="bbtn btn-link" href="/RecoverPassword">¿Has olvidado tu contraseña?</a></p>
</div>
```

2. Adicionamos en **Sales.Shared.DTOs** la clase **ResetPasswordDTO**:

```
using System.ComponentModel.DataAnnotations;

namespace Sales.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!;
    }
}
```

3. Adicionamos estos métodos al **IUserHelper**:

```
Task<string> GeneratePasswordResetTokenAsync(User user);

Task<IdentityResult> ResetPasswordAsync(User user, string token, string password);
```

Y la implementación:

```
public async Task<string> GeneratePasswordResetTokenAsync(User user)
{
    return await _userManager.GeneratePasswordResetTokenAsync(user);
}
```

```
public async Task<IdentityResult> ResetPasswordAsync(User user, string token, string password)
{
    return await _userManager.ResetPasswordAsync(user, token, password);
}
```

4. Adicionamos estos métodos al **AccountController**:

```
[HttpPost("RecoverPassword")]
public async Task<ActionResult> RecoverPassword([FromBody] EmailDTO model)
{
    User user = await _userHelper.GetUserAsync(model.Email);
    if (user == null)
    {
        return NotFound();
    }

    var myToken = await _userHelper.GeneratePasswordResetTokenAsync(user);
    var tokenLink = Url.Action("ResetPassword", "accounts", new
    {
        userid = user.Id,
        token = myToken
    }, HttpContext.Request.Scheme, _configuration["UrlWEB"]);

    var response = _mailHelper.SendMail(user.FullName, user.Email!,
        $"Sales - Recuperación de contraseña",
        $"<h1>Sales - Recuperación de contraseña</h1>" +
        $"<p>Para recuperar su contraseña, por favor hacer clic 'Recuperar Contraseña':</p>" +
        $"<b><a href ={tokenLink}>Recuperar Contraseña</a></b>");

    if (response.IsSuccess)
    {
        return NoContent();
    }

    return BadRequest(response.Message);
}

[HttpPost("ResetPassword")]
public async Task<ActionResult> ResetPassword([FromBody] ResetPasswordDTO model)
{
    User user = await _userHelper.GetUserAsync(model.Email);
    if (user == null)
    {
        return NotFound();
    }

    var result = await _userHelper.ResetPasswordAsync(user, model.Token, model.Password);
    if (result.Succeeded)
    {
        return NoContent();
    }

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

```
}
```

5. Dentro de **Pages/Auth** creamos el **RecoverPassword.razor**:

```
@page "/RecoverPassword"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager

@if (loading)
{
    <div class="spinner" />
}
<div class="row">
  <div class="col-6">
    <EditForm Model="emailDTO" OnValidSubmit="SendRecoverPasswordEmailTokenAsync">
      <DataAnnotationsValidator />
      <div class="card">
        <div class="card-header">
          <span>
            <i class="oi oi-key" /> Enviar email para recuperación de contraseña
            <button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-loop-square" />
Enviar</button>
          </span>
        </div>
        <div class="card-body">
          <div class="mb-3">
            <label>Email:</label>
            <div>
              <InputText class="form-control" @bind-Value="@emailDTO.Email" />
              <ValidationMessage For="@(() => emailDTO.Email)" />
            </div>
          </div>
        </div>
      </div>
    </EditForm>
  </div>
</div>

@code {
  private EmailDTO emailDTO = new();
  private bool loading;

  private async Task SendRecoverPasswordEmailTokenAsync()
  {
    loading = true;
    var responseHttp = await repository.Post("/api/accounts/RecoverPassword", emailDTO);
    if (responseHttp.Error)
    {
      var message = await responseHttp.GetErrorMessageAsync();
      await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
      loading = false;
      return;
    }
```

```
        loading = false;
        await sweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para recuperar su contraseña.", SweetAlertIcon.Info);
        navigationManager.NavigateTo("/");
    }
}
```

6. Dentro de **Pages/Auth** creamos el **ResetPassword.razor**:

```
@page "/api/accounts/ResetPassword"
@inject IRepository repository
@inject SweetAlertService sweetAlertService
@inject NavigationManager navigationManager

@if (loading)
{
    <div class="spinner" />
}
<div class="row">
    <div class="col-6">
        <EditForm Model="resetPasswordDTO" OnValidSubmit="ChangePasswordAsync">
            <DataAnnotationsValidator />
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-key" /> Cambiar Contraseña
                        <button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Cambiar Contrasña</button>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-3">
                        <label>Email:</label>
                        <div>
                            <InputText class="form-control" @bind-Value="@resetPasswordDTO.Email" />
                            <ValidationMessage For="@(() => resetPasswordDTO.Email)" />
                        </div>
                    </div>
                    <div class="mb-3">
                        <label>Nueva contraseña:</label>
                        <div>
                            <InputText type="password" class="form-control" @bind-Value="@resetPasswordDTO.Password" />
                            <ValidationMessage For="@(() => resetPasswordDTO.Password)" />
                        </div>
                    </div>
                    <div class="mb-3">
                        <label>Confirmar contraseña:</label>
                        <div>
                            <InputText type="password" class="form-control"
@bind-Value="@resetPasswordDTO.ConfirmPassword" />
                            <ValidationMessage For="@(() => resetPasswordDTO.ConfirmPassword)" />
                        </div>
                    </div>
```

```
            </div>
        </div>
    </EditForm>
</div>
</div>

@code {
    private ResetPasswordDTO resetPasswordDTO = new();
    private bool loading;

    [Parameter]
    [SupplyParameterFromQuery]
    public string Token { get; set; } = "";

    private async Task ChangePasswordAsync()
    {
        loading = true;
        resetPasswordDTO.Token = Token;
        var responseHttp = await repository.Post("/api/accounts/ResetPassword", resetPasswordDTO);
        if (responseHttp.Error)
        {
            var message = await responseHttp.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            loading = false;
            return;
        }

        loading = false;
        await sweetAlertService.FireAsync("Confirmación", "Contraseña cambiada con éxito, ahora puede ingresar con su
nueva contraseña.", SweetAlertIcon.Info);
        navigationManager.NavigateTo("/Login");
    }
}
```

7. Probamos y hacemos el **commit**.

Solución del problema de la paginación

1. Modificamos el componente de **Pagination**:

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

@code {
    [Parameter] public int CurrentPage { get; set; } = 1;
```

```
    [Parameter] public int TotalPages { get; set; }
    [Parameter] public int Radio { get; set; } = 10;
    [Parameter] public EventCallback<int> SelectedPage { get; set; }
    List<PageModel> Links = new();

    private async Task InternalSelectedPage(PageModel pageModel)
    {
        if (pageModel.Page == CurrentPage || pageModel.Page == 0)
        {
            return;
        }

        await SelectedPage.InvokeAsync(pageModel.Page);
    }

    protected override void OnParametersSet()
    {
        Links = new List<PageModel>();
        var previousLinkEnable = CurrentPage != 1;
        var previousLinkPage = CurrentPage - 1;

        Links.Add(new PageModel
        {
            Text = "Anterior",
            Page = previousLinkPage,
            Enable = previousLinkEnable
        });

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

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

            if (CurrentPage > Radio && i > CurrentPage - Radio && i <= CurrentPage)
            {
                Links.Add(new PageModel
                {
```

```
                Page = i,
                Enable = CurrentPage == i,
                Text = $"{i}"
            });
        }
    }

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

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

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

CRUD de Categorías

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

```
using System.ComponentModel.DataAnnotations;

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

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

3. Modificamos el **DataContext**:

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

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

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

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

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

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

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

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

```
PM> add-migration AddCategories
PM> update-database
```

5. En **Sales.API.Controllers** adicionamos la controlador **CategoriesController**:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Sales.API.Data;
using Sales.API.Helpers;
using Sales.Shared.DTOs;
using Sales.Shared.Entities;

namespace Sales.API.Controllers
{
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("/api/categories")]
    public class CategoiresController : ControllerBase
    {
        private readonly DataContext _context;

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


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

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

        if (!string.IsNullOrWhiteSpace(pagination.Filter))
        {
            queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
        }

        return Ok(await queryable
            .OrderBy(x => x.Name)
            .Paginate(pagination)
            .ToListAsync());
    }


    [HttpGet("totalPages")]
    public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
    {
        var queryable = _context.Categories
            .AsQueryable();

        if (!string.IsNullOrWhiteSpace(pagination.Filter))
        {
            queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));
        }

        double count = await queryable.CountAsync();
        double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
        return Ok(totalPages);
    }

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

        return Ok(category);
    }


    [HttpPost]
    public async Task<ActionResult> Post(Category category)
    {
        _context.Add(category);
        try
        {
            await _context.SaveChangesAsync();
            return Ok(category);
        }
```

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

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

    [HttpDelete("{id:int}")]
    public async Task<IActionResult> DeleteAsync(int id)
    {
        var category = await _context.Categories.FirstOrDefaultAsync(x => x.Id == id);
        if (category == null)
        {
            return NotFound();
        }

        _context.Remove(category);
        await _context.SaveChangesAsync();
```

```
            return NoContent();
        }
    }
}
```

6. Modificamos el **SeedDb**:

```
public async Task SeedAsync()
{
    await _context.Database.EnsureCreatedAsync();
    await CheckCountriesAsync();
    await CheckCategoriesAsync();
    await CheckRolesAsync();
    await CheckUserAsync("1010", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol",
UserType.Admin);
}

private async Task CheckCategoriesAsync()
{
    if (!_context.Categories.Any())
    {
        _context.Categories.Add(new Category { Name = "Deportes" });
        _context.Categories.Add(new Category { Name = "Calzado" });
        _context.Categories.Add(new Category { Name = "Tecnología " });
        _context.Categories.Add(new Category { Name = "Lenceria" });
        _context.Categories.Add(new Category { Name = "Erótica" });
        _context.Categories.Add(new Category { Name = "Comida" });
        _context.Categories.Add(new Category { Name = "Ropa" });
        _context.Categories.Add(new Category { Name = "Jugetes" });
        _context.Categories.Add(new Category { Name = "Mascotas" });
        _context.Categories.Add(new Category { Name = "Autos" });
        _context.Categories.Add(new Category { Name = "Cosmeticos" });
        _context.Categories.Add(new Category { Name = "Hogar" });
        _context.Categories.Add(new Category { Name = "Jardín" });
        _context.Categories.Add(new Category { Name = "Ferreteria" });
        _context.Categories.Add(new Category { Name = "Video Juegos" });
        await _context.SaveChangesAsync();
    }
}
```

7. En **Pages** creamos la carpeta **Categories** y dentro de esta agregamos el **CategoriesIndex.razor**:

```
@page "/categories"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (categories is null)
{
    <div class="spinner" />
}
else
{
```

```
    <GenericList MyList="categories">
        <RecordsComplete>
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-list"></i> Castegorías
                        <a class="btn btn-sm btn-primary float-end" href="/categories/create"><i class="oi oi-plus"></i> Adicionar Categoría</a>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
                        <div>
                            <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar categoría..." @bind-value="Filter" />
                        </div>
                        <div class="mx-1">
                            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi oi-layers" /> Filtrar</button>
                            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban" /> Limpiar</button>
                        </div>
                    </div>

                    <Pagination CurrentPage="currentPage"
                        TotalPages="totalPages"
                        SelectedPage="SelectedPageAsync" />

                    <table class="table table-striped">
                        <thead>
                            <tr>
                                <th>Categoría</th>
                                <th style="width:200px"></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"><i class="oi oi-pencil" /> Editar</a>
                                        <button class="btn btn-danger" @onclick=@(() => Delete(category.Id))><i class="oi oi-trash" /> Borrar</button>
                                    </td>
                                </tr>
                            }
                        </tbody>
                    </table>
                </div>
            </div>
```

```
        </RecordsComplete>
    </GenericList>
}

@code {
    public List<Category>? categories { get; set; }
    private int currentPage = 1;
    private int totalPages;

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

    protected async override 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);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/categories?page={page}";
            url2 = $"api/categories/totalPages";
        }
        else
        {
            url1 = $"api/categories?page={page}&filter={Filter}";
            url2 = $"api/categories/totalPages?filter={Filter}";
        }

        try
        {
            var responseHppt = await repository.Get<List<Category>>(url1);
            var responseHppt2 = await repository.Get<int>(url2);
            categories = responseHppt.Response!;
```

```
            totalPages = responseHppt2.Response!;
        }
        catch (Exception ex)
        {
            await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
        }
    }

    private async Task Delete(int categoryId)
    {
        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.Delete($"api/categories/{categoryId}");

        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);
            }
        }
        else
        {
            await LoadAsync();
        }
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }

    private async Task ApplyFilterAsync()
    {
        int page = 1;
```

```
        await LoadAsync(page);
        await SelectedPageAsync(page);
    }
}
```

8. Modificamos el **NavMenu.razor**:

```
<AuthorizeView Roles="Admin">
    <Authorized>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="categories">
                <span class="oi oi-list" aria-hidden="true"></span> Categorías
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="countries">
                <span class="oi oi-globe" aria-hidden="true"></span> Países
            </NavLink>
        </div>
    </Authorized>
</AuthorizeView>
```

9. Probamos lo que llevamos hasta el momento.

10. Creamos el **CategoryForm**:

```
@inject SweetAlertService sweetAlertService

<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>

@code {
    private EditContext editContext = null!;

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

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

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

    public bool FormPostedSuccessfully { get; set; }

    protected override void OnInitialized()
    {
        editContext = new(Category);
    }

    private async Task OnBeforeInternalNavigation(LocationChangingContext context)
    {
        var formWasMofied = editContext.IsModified();
        if (!formWasMofied || FormPostedSuccessfully)
        {
            return;
        }

        var result = await sweetAlertService.FireAsync(new SweetAlertOptions
            {
                Title = "Confirmación",
                Text = "¿Deseas abandonar la página y perder los cambios?",
                Icon = SweetAlertIcon.Question,
                ShowCancelButton = true,
                CancelButtonText = "No",
                ConfirmButtonText = "Si"
            });

        var confirm = !string.IsNullOrEmpty(result.Value);
        if (confirm)
        {
            return;
        }

        context.PreventNavigation();
    }
}
```

11. Creamos el **CategoryCreate**:

```
@page "/categories/create"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService

<h3>Crear categoría</h3>

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

@code {
    private Category category = new();
    private CategoryForm? categoryForm;
```

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

    private async Task CreateAsync()
    {
        var httpResponse = await repository.Post("/api/categories", category);
        if (httpResponse.Error)
        {
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        Return();
    }

    private void Return()
    {
        categoryForm!.FormPostedSuccessfully = true;
        navigationManager.NavigateTo($"/categories");
    }
}
```

12. Creamos el **CategoryEdit**:

```
@page "/categories/edit/{CategoryId:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService

<h3>Editar categoría</h3>

@if (category is null)
{
    <div class="spinner" />
}
else
{
    <CategoryForm @ref="categoryForm" Category="category" OnValidSubmit="EditAsync" ReturnAction="Return" />
}

@code {
    private Category? category;
    private CategoryForm? categoryForm;

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

    protected override async Task OnInitializedAsync()
    {
        var responseHttp = await repository.Get<Category>($"/api/categories/{CategoryId}");
        if (responseHttp.Error)
        {
            if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)
```

```
        {
            navigationManager.NavigateTo("/categories");
            return;
        }

        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    category = responseHttp.Response;
}

private async Task EditAsync()
{
    var responseHttp = await repository.Put("/api/categories", category);
    if (responseHttp.Error)
    {
        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    Return();
}

private void Return()
{
    categoryForm!.FormPostedSuccessfully = true;
    navigationManager.NavigateTo($"/categories");
}
}
```

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

Implementación de ventanas modales

Este capitulo es gracias **Lou Gonzalez** que me compartió el código, gracias **Lou**, gran aporte! Documentación oficial en: https://blazored.github.io/Modal/

1. Instalar el paquete **Blazored.Modal**.

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

```
builder.Services.AddBlazoredModal();
```

3. Modificamos el **_Imports.razor**:

```
@using Blazored.Modal
@using Blazored.Modal.Services
```

4. 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>
```

5. Voy hacer el ejemplo con categorías, modificamos el **Categories.index**:

```
…
<a class="btn btn-sm btn-primary float-end" @onclick=@(() => ShowModal())><i class="oi oi-plus"></i> Adicionar
Categoría</a>
…
<a @onclick=@(() => ShowModal(category.Id, true)) class="btn btn-warning"><i class="oi oi-pencil" /> Editar</a>
…
[CascadingParameter]
IModalService Modal { get; set; } = default!;
…
private async Task ShowModal(int id = 0, bool isEdit = false)
{
    IModalReference modalReference;

    if (isEdit)
    {
        modalReference = Modal.Show<CategoryEdit>(string.Empty, new ModalParameters().Add("CategoryId", id));
    }
    else
    {
        modalReference = Modal.Show<CategoryCreate>();
    }

    var result = await modalReference.Result;
    if (result.Confirmed)
    {
        await LoadAsync();
    }
```

```
}
…
```

6. Modificamos el **CategoriesEdit**:

```
…
[CascadingParameter]
BlazoredModalInstance BlazoredModal { get; set; } = default!;
…
private async Task EditAsync()
{
    var responseHttp = await repository.Put("/api/categories", category);
    if (responseHttp.Error)
    {
        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    await BlazoredModal.CloseAsync(ModalResult.Ok());
    Return();
}
…
```

7. Modificamos el **CategoriesCreate**:

```
…
[CascadingParameter]
BlazoredModalInstance BlazoredModal { get; set; } = default!;
…
private async Task CreateAsync()
{
    var httpResponse = await repository.Post("/api/categories", category);
    if (httpResponse.Error)
    {
        var message = await httpResponse.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    await BlazoredModal.CloseAsync(ModalResult.Ok());
    Return();
}
…
```

8. Probamos (Corremos la App con Ctrl + F5) y hacemos el **commit**.

Actividad #4

Deben estar al día en la aplicación, aplicar ventanas modales al resto de la aplicación, es decir: Crear/Editar de países, estados y ciudades y aplicar ventana modal al cambio de contraseña.

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 Sales.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; }
    }
}
```

2. Creamos la entidad **ProductImage**:

```
using System.ComponentModel.DataAnnotations;

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

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

        public int ProductId { get; set; }

        [Display(Name = "Imagen")]
        public string Image { get; set; } = null!;
    }
```

```
}
```

3. Creamos la entidad **ProductCategory**:

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

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

        public int ProductId { get; set; }

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

        public int CategoryId { get; set; }
    }
}
```

4. Modificamos la entidad **Category**:

```
public class Category
{
    public int Id { get; set; }

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

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

    [Display(Name = "Productos")]
    public int ProductCategoriesNumber => ProductCategories == null ? 0 : ProductCategories.Count;
}
```

5. Modificamos la entidad **Product**:

```
public class Product
{
    public int Id { get; set; }

    [Display(Name = "Nombre")]
    [MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]
    [Required(ErrorMessage = "El campo {0} es obligatorio.")]
    public string Name { get; set; } = null!;

    [DataType(DataType.MultilineText)]
    [Display(Name = "Descripción")]
    [MaxLength(500, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]
    public string Description { get; set; } = null!;

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

```
    [DisplayFormat(DataFormatString = "{0:C2}")]
    [Display(Name = "Precio")]
    [Required(ErrorMessage = "El campo {0} es obligatorio.")]
    public decimal Price { get; set; }

    [DisplayFormat(DataFormatString = "{0:N2}")]
    [Display(Name = "Inventario")]
    [Required(ErrorMessage = "El campo {0} es obligatorio.")]
    public float Stock { get; set; }

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

    [Display(Name = "Categorías")]
    public int ProductCategoriesNumber => ProductCategories == null ? 0 : ProductCategories.Count;

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

    [Display(Name = "Imágenes")]
    public int ProductImagesNumber => ProductImages == null ? 0 : ProductImages.Count;

    [Display(Name = "Imagén")]
    public string MainImage => ProductImages == null ? string.Empty : ProductImages.FirstOrDefault()!.Image;
}
```

6. 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();
    }
}
```

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

```
PM> add-migration AddProductsTables
PM> update-database
```

8. Dentro del proyecto **API** copiamos el folder **Images** el cual puedes obtener de mi repositorio.

9. Borramos de la base de datos las **categorías** y **usuarios** que tengamos.

10. Modificamos el **SeedDb** para agregar registros a las nuevas tablas y de paso aprovechamos y creamos los usuarios con foto:

```
public class SeedDb
{
private readonly DataContext _context;
private readonly IApiService _apiService;
private readonly IUserHelper _userHelper;
private readonly IFileStorage _fileStorage;

public SeedDb(DataContext context, IApiService apiService, IUserHelper userHelper, IFileStorage fileStorage)
{
    _context = context;
    _apiService = apiService;
    _userHelper = userHelper;
    _fileStorage = fileStorage;
}

public async Task SeedAsync()
{
    await _context.Database.EnsureCreatedAsync();
    await CheckCountriesAsync();
    await CheckCategoriesAsync();
    await CheckRolesAsync();
    await CheckUserAsync("1010", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol",
"JuanZuluaga.jpeg", UserType.Admin);
    await CheckUserAsync("2020", "Ledys", "Bedoya", "ledys@yopmail.com", "322 311 4620", "Calle Luna Calle Sol",
"LedysBedoya.jpeg", UserType.User);
    await CheckUserAsync("3030", "Brad", "Pitt", "brad@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "Brad.jpg",
UserType.User);
    await CheckUserAsync("4040", "Angelina", "Jolie", "angelina@yopmail.com", "322 311 4620", "Calle Luna Calle Sol",
"Angelina.jpg", UserType.User);
    await CheckUserAsync("5050", "Bob", "Marley", "bob@yopmail.com", "322 311 4620", "Calle Luna Calle Sol",
"bob.jpg", UserType.User);
    await CheckProductsAsync();
}

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("AirPods", 1300000M, 12F, new List<string>() { "Tecnología", "Apple" }, new List<string>() { "airpos.png", "airpos2.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("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("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("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 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();
```

```
    }
}

private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string
phone, string address, string image, UserType userType)
{
    var user = await _userHelper.GetUserAsync(email);

    if (user == null)
    {
        var city = await _context.Cities.FirstOrDefaultAsync(x => x.Name == "Medellín");
        if (city == null)
        {
            city = await _context.Cities.FirstOrDefaultAsync();
        }

        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 _userHelper.AddUserAsync(user, "123456");
        await _userHelper.AddUserToRoleAsync(user, userType.ToString());

        var token = await _userHelper.GenerateEmailConfirmationTokenAsync(user);
        await _userHelper.ConfirmEmailAsync(user, token);
    }

    return user;
}
…
```

11. Probamos lo que llevamos.

12. Creamos el **ProductDTO**:

```
using Microsoft.EntityFrameworkCore.Metadata.Internal;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Sales.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; }
    }
}
```

13. Creamos el **ProductsController**:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Sales.API.Data;
using Sales.API.Helpers;
using Sales.Shared.DTOs;
using Sales.Shared.Entities;

namespace Sales.API.Controllers
{
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("/api/products")]
    public class ProductsController :  ControllerBase
    {
        private readonly DataContext _context;
        private readonly IFileStorage _fileStorage;

        public ProductsController(DataContext context, IFileStorage fileStorage)
```

```
    {
        _context = context;
        _fileStorage = fileStorage;
    }

    [HttpGet]
    public async Task<ActionResult> Get([FromQuery] 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 Ok(await queryable
            .OrderBy(x => x.Name)
            .Paginate(pagination)
            .ToListAsync());
    }


    [HttpGet("totalPages")]
    public async Task<ActionResult> GetPages([FromQuery] 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();
        double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
        return Ok(totalPages);
    }

    [HttpGet("{id:int}")]
    public async Task<IActionResult> 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 NotFound();
        }
```

```
        return Ok(product);
    }

    [HttpPost]
    public async Task<ActionResult> PostAsync(ProductDTO productDTO)
    {
        try
        {
            Product newProduct = new()
            {
                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!)
            {
                newProduct.ProductCategories.Add(new ProductCategory { Category = await
_context.Categories.FirstOrDefaultAsync(x => x.Id == productCategoryId) });
            }

            _context.Add(newProduct);
            await _context.SaveChangesAsync();
            return Ok(productDTO);
        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
            {
                return BadRequest("Ya existe una ciudad con el mismo nombre.");
            }

            return BadRequest(dbUpdateException.Message);
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }

    [HttpPut]
    public async Task<ActionResult> PutAsync(Product product)
    {
        try
```

```
            {
                _context.Update(product);
                await _context.SaveChangesAsync();
                return Ok(product);
            }
            catch (DbUpdateException dbUpdateException)
            {
                if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
                {
                    return BadRequest("Ya existe un producto con el mismo nombre.");
                }

                return BadRequest(dbUpdateException.Message);
            }
            catch (Exception exception)
            {
                return BadRequest(exception.Message);
            }
        }

        [HttpDelete("{id:int}")]
        public async Task<IActionResult> DeleteAsync(int id)
        {
            var product = await _context.Products.FirstOrDefaultAsync(x => x.Id == id);
            if (product == null)
            {
                return NotFound();
            }

            _context.Remove(product);
            await _context.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

14. Dentro de **Pages** creamos la carpeta **Products** y dentro de esta creamos el **ProductsIndex**:

```
@page "/products"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (Products is null)
{
    <div class="spinner" />
}
else
{
    <GenericList MyList="Products">
        <Body>
            <div class="card">
                <div class="card-header">
```

```
                <span>
                    <i class="oi oi-star"/> Productos
                    <a class="btn btn-sm btn-primary float-end" href="/products/create"><i class="oi oi-plus"/> Nuevo
Producto</a>
                </span>
            </div>
            <div class="card-body">
                <div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
                    <div>
                        <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar producto..."
@bind-value="Filter" />
                    </div>
                    <div class="mx-1">
                        <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi
oi-layers" /> Filtrar</button>
                        <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban"
/> Limpiar</button>
                    </div>
                </div>

                <Pagination CurrentPage="currentPage"
                        TotalPages="totalPages"
                        SelectedPage="SelectedPageAsync" />

                <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:200px"></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"><i class="oi oi-pencil" />
Editar</a>
                        <button class="btn btn-danger" @onclick=@(() => Delete(product.Id))><i class="oi oi-trash" />
Borrar</button>
                    </td>
                </tr>
            }
        </tbody>
    </table>
</div>
</div>
</Body>
</GenericList>
}

@code {
    private int currentPage = 1;
    private int totalPages;

    public List<Product>? Products { get; set; }

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

    protected async override 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);
```

```
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/products?page={page}";
            url2 = $"api/products/totalPages";
        }
        else
        {
            url1 = $"api/products?page={page}&filter={Filter}";
            url2 = $"api/products/totalPages?filter={Filter}";
        }

        try
        {
            var responseHppt = await repository.Get<List<Product>>(url1);
            var responseHppt2 = await repository.Get<int>(url2);
            Products = responseHppt.Response!;
            totalPages = responseHppt2.Response!;
        }
        catch (Exception ex)
        {
            await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
        }
    }

    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.Delete($"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 CleanFilterAsync()
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }

    private async Task ApplyFilterAsync()
    {
        int page = 1;
        await LoadAsync(page);
        await SelectedPageAsync(page);
    }
}
```

15. Modificamos el **NavMenu.razor**:

```
<AuthorizeView Roles="Admin">
    <Authorized>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="categories">
                <span class="oi oi-list" aria-hidden="true"></span> Categorías
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="countries">
                <span class="oi oi-globe" aria-hidden="true"></span> Países
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="products">
                <span class="oi oi-star" aria-hidden="true"></span> Productos
            </NavLink>
        </div>
    </Authorized>
</AuthorizeView>
```

16. Probamos y hacemos el **commit** de lo que llevamos.

Creando nuevos productos

17. Creamos el componente genérico para poder seleccionar varitas categorías. Primero creamos en **Sales.WEB.Helpers** la clase **MultipleSelectorModel**:

```
namespace Sales.WEB.Helpers
```

```
{
    public class MultipleSelectorModel
    {
        public MultipleSelectorModel(string key, string value)
        {
            Key = key;
            Value = value;
        }

        public string Key { get; set; }

        public string Value { get; set; }
    }
}
```

18. 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;
    }
```

19. Creamos en **Shared** nuestro **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>

@code {
    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();
    }
}
```

20. Dentro de **Pages/Products** creamos el **ProductForm.razor**:

```
@inject SweetAlertService sweetAlertService

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

<EditForm EditContext="editContext" OnValidSubmit="OnDataAnnotationsValidatedAsync">
    <DataAnnotationsValidator />

    <div class="card">
        <div class="card-header">
            <span>
                <i class="oi oi-star" /> Crear Nuevo Producto
                <a class="btn btn-sm btn-success float-end" href="/products"><i class="oi oi-arrow-thick-left" /> Regresar</a>
                <button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Guardar
Cambios</button>
            </span>
        </div>
        <div class="card-body">
            <div class="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="oi oi-plus"
/> Agregar Imagenes</button>
                        <button type="button" class="btn btn-outline-danger" @onclick="RemoveImageAction"><i class="oi
oi-trash" /> Eliminar Última Imagén</button>
                    </div>
                }
            </div>
        </div>
    </div>
</EditForm>

@*@if (IsEdit && ProductDTO.ProductImages is not null)
{
    <CarouselView Images="ProductDTO.ProductImages" />
}*@

@code {
    private EditContext editContext = null!;
    private List<MultipleSelectorModel> selected { get; set; } = new();
    private List<MultipleSelectorModel> nonSelected { get; set; } = new();
    private string? imageUrl;

    [Parameter]
    public bool IsEdit { get; set; } = false;

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

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

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

    [Parameter]
    public EventCallback AddImageAction { get; set; }
```

```
    [Parameter]
    public EventCallback RemoveImageAction { get; set; }

    [Parameter]
    public List<Category> SelectedCategories { get; set; } = new();

    [Parameter]
    [EditorRequired]
    public List<Category> NonSelectedCategories { 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();
  }
}
```

21. Dentro de **Pages/Products** creamos el **ProductCreate.razor**:

```
@page "/products/create"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (loading)
{
  <div class="spinner" />
}
else
{
  <ProductForm @ref="productForm" ProductDTO="productDTO" NonSelectedCategories="nonSelectedCategories"
OnValidSubmit="CreateAsync" ReturnAction="Return" />
}

@code {
  private ProductDTO productDTO = new ProductDTO
    {
        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;

  protected async override Task OnInitializedAsync()
  {
    var httpResponse = await repository.Get<List<Category>>("/api/categories");
    loading = false;

    if (httpResponse.Error)
    {
```

```
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        nonSelectedCategories = httpResponse.Response!;
    }

    private async Task CreateAsync()
    {
        var httpResponse = await repository.Post("/api/products", productDTO);
        if (httpResponse.Error)
        {
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        Return();
    }

    private void Return()
    {
        productForm!.FormPostedSuccessfully = true;
        navigationManager.NavigateTo($"/products");
    }
}
```

22. Podemos probar la creación de productos.

23. Modificamos el **Post** del **ProductsController**:

```
[HttpPost]
public async Task<ActionResult> PostAsync(ProductDTO productDTO)
{
    try
    {
        Product newProduct = new()
        {
            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!)
        {
            newProduct.ProductCategories.Add(new ProductCategory { Category = await
_context.Categories.FirstOrDefaultAsync(x => x.Id == productCategoryId) });
        }

        _context.Add(newProduct);
        await _context.SaveChangesAsync();
        return Ok(productDTO);
    }
    catch (DbUpdateException dbUpdateException)
    {
        if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
        {
            return BadRequest("Ya existe una ciudad con el mismo nombre.");
        }

        return BadRequest(dbUpdateException.Message);
    }
    catch (Exception exception)
    {
        return BadRequest(exception.Message);
    }
}
```

24. 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**.

3. En el **_Imports.razor** agregamos la línea:

```
@using MudBlazor
```

4. 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, maximum-scale=1.0, user-scalable=no" />
    <title>Sales.WEB</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
    <link rel="icon" type="image/png" href="favicon.png" />
    <link href="Sales.WEB.styles.css" rel="stylesheet" />
    <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>
```

5. Injectamos en el **Program** del proyecto **WEB**:

```
builder.Services.AddMudServices();
```

6. Creamos el componente compartido **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>

@code {
    private bool arrows = true;
    private bool bullets = true;
    private bool enableSwipeGesture = true;
    private bool autocycle = true;
    private Transition transition = Transition.Slide;

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

```
}
```

7. Modificamos el **ProductForm**:

```
…
</EditForm>

@if (IsEdit && Images is not null)
{
    <CarouselView Images="Images" />
}
…
```

8. Creamos el **ProductEdit**:

```
@page "/products/edit/{ProductId:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (loading)
{
    <div class="spinner" />
}
else
{
    <ProductForm @ref="productForm" ProductDTO="productDTO" SelectedCategories="selectedCategories"
NonSelectedCategories="nonSelectedCategories" OnValidSubmit="SaveChangesAsync" ReturnAction="Return"
IsEdit=true AddImageAction="AddImageAsync" RemoveImageAction="RemoveImageAsyc"/>
}

@code {
    private ProductDTO productDTO = new ProductDTO
        {
            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; }

    protected async override Task OnInitializedAsync()
    {
        await LoadProductAsync();
        await LoadCategoriesAsync();
    }
```

```
    private async Task AddImageAsync()
    {
    }

    private async Task RemoveImageAsyc()
    {
    }

    private async Task LoadProductAsync()
    {
        loading = true;
        var httpResponse = await repository.Get<Product>($"/api/products/{ProductId}");

        if (httpResponse.Error)
        {
            loading = false;
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        product = httpResponse.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 httpResponse = await repository.Get<List<Category>>("/api/categories");

        if (httpResponse.Error)
        {
            loading = false;
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        var categories = httpResponse.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 httpResponse = await repository.Put("/api/products", productDTO);
        if (httpResponse.Error)
        {
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        Return();
    }

    private void Return()
    {
        productForm!.FormPostedSuccessfully = true;
        navigationManager.NavigateTo($"/products");
    }
}
```

9. Actualizamos el **PUT** en el **ProductsController**:

```
[HttpPut]
public async Task<ActionResult> PutAsync(ProductDTO productDTO)
{
    try
    {
        var product = await _context.Products
            .Include(x => x.ProductCategories)
            .FirstOrDefaultAsync(x => x.Id == productDTO.Id);
        if (product == null)
        {
            return NotFound();
        }

        product.Name = productDTO.Name;
        product.Description = productDTO.Description;
        product.Price = productDTO.Price;
        product.Stock = productDTO.Stock;
```

```
        product.ProductCategories = productDTO.ProductCategoryIds!.Select(x => new ProductCategory { CategoryId = x
}).ToList();

        _context.Update(product);
        await _context.SaveChangesAsync();
        return Ok(productDTO);
    }
    catch (DbUpdateException dbUpdateException)
    {
        if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
        {
            return BadRequest("Ya existe una ciudad con el mismo nombre.");
        }

        return BadRequest(dbUpdateException.Message);
    }
    catch (Exception exception)
    {
        return BadRequest(exception.Message);
    }
}
```

10. 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

11. Dento de **Sales.Shared.DTOs** creamos el **ImageDTO**.

```
using System.ComponentModel.DataAnnotations;

namespace Sales.Shared.DTOs
{
    public class ImageDTO
    {
        [Required]
        public int ProductId { get; set; }

        [Required]
        public List<string> Images { get; set; } = null!;
    }
}
```

12. Modificamos el **ProductsController**.

```
[HttpPost("addImages")]
public async Task<ActionResult> PostAddImagesAsync(ImageDTO imageDTO)
{
    var product = await _context.Products
        .Include(x => x.ProductImages)
        .FirstOrDefaultAsync(x => x.Id == imageDTO.ProductId);
    if (product == null)
```

```
    {
        return NotFound();
    }

    if (product.ProductImages is null)
    {
        product.ProductImages = new List<ProductImage>();
    }

    for (int i = 0; i < imageDTO.Images.Count; i++)
    {
        if (!imageDTO.Images[i].StartsWith("https://sales2023.blob.core.windows.net/products/"))
        {
            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 Ok(imageDTO);
}

[HttpPost("removeLastImage")]
public async Task<ActionResult> PostRemoveLastImageAsync(ImageDTO imageDTO)
{
    var product = await _context.Products
        .Include(x => x.ProductImages)
        .FirstOrDefaultAsync(x => x.Id == imageDTO.ProductId);
    if (product == null)
    {
        return NotFound();
    }

    if (product.ProductImages is null || product.ProductImages.Count == 0)
    {
        return Ok();
    }

    var lastImage = product.ProductImages.LastOrDefault();
    await _fileStorage.RemoveFileAsync(lastImage!.Image, "products");
    product.ProductImages.Remove(lastImage);
    _context.Update(product);
    await _context.SaveChangesAsync();
    imageDTO.Images = product.ProductImages.Select(x => x.Image).ToList();
    return Ok(imageDTO);
}
```

13. 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://sales2023.blob.core.windows.net/products/"))
            {
                <MudCarouselItem Transition="transition" Color="@Color.Primary">
                    <div class="d-flex" style="height:100%; justify-content:center">
                        <img src="@image" />
                    </div>
                </MudCarouselItem>
            }
        }
    </MudCarousel>
</div>
```

14. Modificamos el **ProductEdit.razor**.

```
private async Task AddImageAsync()
{
    if (productDTO.ProductImages is null || productDTO.ProductImages.Count == 0)
    {
        return;
    }

    var imageDTO = new ImageDTO
    {
        ProductId = ProductId,
        Images = productDTO.ProductImages!
    };

    var httpResponse = await repository.Post<ImageDTO, ImageDTO>("/api/products/addImages", imageDTO);
    if (httpResponse.Error)
    {
        var message = await httpResponse.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    productDTO.ProductImages = httpResponse.Response!.Images;
    var toast = sweetAlertService.Mixin(new SweetAlertOptions
        {
            Toast = true,
            Position = SweetAlertPosition.TopEnd,
            ShowConfirmButton = false,
            Timer = 5000
        });
    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 httpResponse = await repository.Post<ImageDTO, ImageDTO>("/api/products/removeLastImage", imageDTO);
    if (httpResponse.Error)
    {
        var message = await httpResponse.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    productDTO.ProductImages = httpResponse.Response!.Images;
    var toast = sweetAlertService.Mixin(new SweetAlertOptions
        {
            Toast = true,
            Position = SweetAlertPosition.TopEnd,
            ShowConfirmButton = false,
            Timer = 5000
        });
    await toast.FireAsync(icon: SweetAlertIcon.Success, message: "Imagén eliminada con éxito.");
}
```

15. 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.

Creando el “Home” de nuestra aplicación

1. Modificamos el **ProductsController** y le colocamos el **[AllowAnonymous]** a todos los **GET** de este controlador.

2. Modificamos el **Index.razor**.

```
@page "/"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService

<style type="text/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;
    }
</style>

@if (Products is null)
```

```
{
    <div class="spinner" />
}
else
{
    <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 producto..."
@bind-value="Filter" />
        </div>
        <div class="mx-1">
            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi oi-layers" />
Filtrar</button>
            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban" />
Limpiar</button>
        </div>
    </div>

    <Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPageAsync" />

    <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="oi oi-info" />
Detalles</a>
                        <button class="btn btn-sm btn-primary" @onclick=@(() => AddToCartAsync(product.Id))><i class="oi
oi-plus" /> Agregar al Carro</button>
                    </div>
                </div>
            </div>
        }
    </div>
}

@code {
    private int currentPage = 1;
    private int totalPages;

    public List<Product>? Products { get; set; }
```

```
    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

    protected async override 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);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/products?page={page}&RecordsNumber=8";
            url2 = $"api/products/totalPages/?RecordsNumber=8";
        }
        else
        {
            url1 = $"api/products?page={page}&filter={Filter}&RecordsNumber=8";
            url2 = $"api/products/totalPages?filter={Filter}&RecordsNumber=8";
        }

        try
        {
            var responseHppt = await repository.Get<List<Product>>(url1);
            var responseHppt2 = await repository.Get<int>(url2);
            Products = responseHppt.Response!;
            totalPages = responseHppt2.Response!;
        }
        catch (Exception ex)
        {
            await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
        }
    }

    private async Task CleanFilterAsync()
```

```
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }

    private async Task ApplyFilterAsync()
    {
        int page = 1;
        await LoadAsync(page);
        await SelectedPageAsync(page);
    }

    private void AddToCartAsync(int productId)
    {

    }
}
```

3. Probamos y hacemos el **commit**.

Agregando productos al carro de compras

1. Creamos la entidad **TemporalSale**:

```
using System.ComponentModel.DataAnnotations;

namespace Sales.Shared.Entities
{
    public class TemporalSale
    {
        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;
    }
}
```

2. Modificmos la entidad **Product** agregando esta propiedad:

```
public ICollection<TemporalSale>? TemporalSales { get; set; }
```

3. Modificmos la entidad **User** agregando esta propiedad:

```
public ICollection<TemporalSale>? TemporalSales { get; set; }
```

4. La adicionamos en el **DataContext**:

```
public DbSet<TemporalSale> TemporalSales { get; set; }
```

5. Creamos la migración y actualizamos la base de datos.

6. En **Sales.Shared.DTOs** creamos el **TemporalSaleDTO**.

```
namespace Sales.Shared.DTOs
{
    public class TemporalSaleDTO
    {
        public int ProductId { get; set; }

        public float Quantity { get; set; } = 1;

        public string Remarks { get; set; } = string.Empty;
    }
}
```

7. Creamos el **TemporalSalesController**:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Sales.API.Data;
using Sales.Shared.DTOs;
using Sales.Shared.Entities;

namespace Sales.API.Controllers
{
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("/api/temporalSales")]
    public class TemporalSalesController : ControllerBase
    {
        private readonly DataContext _context;

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

        [HttpPost]
        public async Task<ActionResult> Post(TemporalSaleDTO temporalSaleDTO)
        {
```

```
            var product = await _context.Products.FirstOrDefaultAsync(x => x.Id == temporalSaleDTO.ProductId);
            if (product == null)
            {
                return NotFound();
            }

            var user = await _context.Users.FirstOrDefaultAsync(x => x.Email == User.Identity!.Name);
            if (user == null)
            {
                return NotFound();
            }

            var temporalSale = new TemporalSale
            {
                Product= product,
                Quantity= temporalSaleDTO.Quantity,
                Remarks = temporalSaleDTO.Remarks,
                User = user
            };

            try
            {
                _context.Add(temporalSale);
                await _context.SaveChangesAsync();
                return Ok(temporalSaleDTO);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        public async Task<ActionResult> Get()
        {
            return Ok(await _context.TemporalSales
                .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 == User.Identity!.Name)
                .ToListAsync());
        }

        [HttpGet("count")]
        public async Task<ActionResult> GetCount()
        {
            return Ok(await _context.TemporalSales
                .Where(x => x.User!.Email == User.Identity!.Name)
                .SumAsync(x => x.Quantity));
        }
    }
```

```
}
```

8. Modificamos el **Index.razor**.

```
@page "/"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService

<style type="text/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;
    }
</style>

@if (Products is null)
{
    <div class="spinner" />
}
else
{
    <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 producto..."
@bind-value="Filter" />
        </div>
        <div class="mx-1">
            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi oi-layers" />
Filtrar</button>
            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban" />
Limpiar</button>
        </div>
        <AuthorizeView>
            <Authorized>
                @if (counter > 0)
                {
                    <a href="/Orders/ShowCart" class="btn btn-primary">Ver Carro de Compras (@counter)</a>
                }
            </Authorized>
        </AuthorizeView>
    </div>

    <Pagination CurrentPage="currentPage"
            TotalPages="totalPages"
            SelectedPage="SelectedPageAsync" />

    <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="oi oi-info" />
Detalles</a>
                    <button class="btn btn-sm btn-primary" @onclick=@(() => AddToCartAsync(product.Id))><i class="oi
oi-plus" /> Agregar al Carro</button>
                </div>
            </div>
        </div>
    }
</div>
}

@code {
    private int currentPage = 1;
    private int totalPages;
    private int counter = 0;
    private bool isAuthenticated;

    public List<Product>? Products { get; set; }

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

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

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

    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.Get<int>("/api/temporalSales/count");
        if (responseHttp.Error)
        {
            return;
        }
        counter = responseHttp.Response;
    }


    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);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/products?page={page}&RecordsNumber=8";
            url2 = $"api/products/totalPages/?RecordsNumber=8";
        }
        else
        {
            url1 = $"api/products?page={page}&filter={Filter}&RecordsNumber=8";
            url2 = $"api/products/totalPages?filter={Filter}&RecordsNumber=8";
        }

        try
        {
            var responseHppt = await repository.Get<List<Product>>(url1);
            var responseHppt2 = await repository.Get<int>(url2);
```

```
            Products = responseHppt.Response!;
            totalPages = responseHppt2.Response!;
        }
        catch (Exception ex)
        {
            await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
        }
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
        await ApplyFilterAsync();
    }

    private async Task ApplyFilterAsync()
    {
        int page = 1;
        await LoadAsync(page);
        await SelectedPageAsync(page);
    }

    private async Task AddToCartAsync(int productId)
    {
        if (!isAuthenticated)
        {
            navigationManager.NavigateTo("/Login");
            var toast1 = sweetAlertService.Mixin(new SweetAlertOptions
                {
                    Toast = true,
                    Position = SweetAlertPosition.TopEnd,
                    ShowConfirmButton = false,
                    Timer = 5000
                });
            await toast1.FireAsync(icon: SweetAlertIcon.Error, message: "Debes haber iniciado sesión para poder agregar
productos al carro de compras.");
            return;
        }

        var temporalSaleDTO = new TemporalSaleDTO
        {
            ProductId = productId
        };

        var httpResponse = await repository.Post("/api/temporalSales", temporalSaleDTO);
        if (httpResponse.Error)
        {
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        await LoadCounterAsync();
```

```
        var toast2 = sweetAlertService.Mixin(new SweetAlertOptions
        {
            Toast = true,
            Position = SweetAlertPosition.TopEnd,
            ShowConfirmButton = false,
            Timer = 5000
        });
        await toast2.FireAsync(icon: SweetAlertIcon.Success, message: "Producto agregado al carro de compras.");
    }
}
```

4. Dentro de **Pages** creamos la carpeta **Orders** y dentro de esta creamos el **ShowCart.razor** temporal.

```
@page "/Orders/ShowCart"

<h3>ShowCart</h3>

@code {

}
```

5. Probamos lo que llevamos hasta el momento.

6. 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**.

```
@page "/orders/details/{ProductId:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService

@if (loading)
{
    <div class="spinner" />
}
else
{
    <div class="card">
        <div class="card-header">
            <span>
                <i class="oi oi-star" /> @product!.Name
                <a class="btn btn-sm btn-success float-end" href="/"><i class="oi oi-arrow-thick-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="TemporalSaleDTO" OnValidSubmit="AddToCartAsync">
                <DataAnnotationsValidator />
                <div class="mb-3">
                    <label>Cantidad:</label>
                    <div>
                        <InputNumber class="form-control" @bind-Value="@TemporalSaleDTO.Quantity" />
                        <ValidationMessage For="@(() => TemporalSaleDTO.Quantity)" />
                    </div>
                    <label>Comentarios:</label>
                    <div>
                        <InputText class="form-control" @bind-Value="@TemporalSaleDTO.Remarks" />
                        <ValidationMessage For="@(() => TemporalSaleDTO.Remarks)" />
                    </div>
                </div>
                <button class="btn btn-primary" type="submit"><i class="oi oi-plus" /> Agregar Al Carro de
Compras</button>
            </EditForm>

        </div>
    </div>
    <CarouselView Images="images" />
  </div>
</div>
}
```

```
@code {
    private List<string>? categories;
    private List<string>? images;
    private bool loading = true;
    private Product? product;
    private bool isAuthenticated;

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

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

    public TemporalSaleDTO TemporalSaleDTO { get; set; } = new();

    protected async override Task OnParametersSetAsync()
    {
        await CheckIsAuthenticatedAsync();
    }

    private async Task CheckIsAuthenticatedAsync()
    {
        var authenticationState = await authenticationStateTask;
        isAuthenticated = authenticationState.User.Identity!.IsAuthenticated;
    }

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

    private async Task LoadProductAsync()
    {
        loading = true;
        var httpResponse = await repository.Get<Product>($"/api/products/{ProductId}");

        if (httpResponse.Error)
        {
            loading = false;
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        product = httpResponse.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.TopEnd,
                ShowConfirmButton = false,
                Timer = 5000
            });
            await toast1.FireAsync(icon: SweetAlertIcon.Error, message: "Debes haber iniciado sesión para poder agregar productos al carro de compras.");
            return;
        }

        TemporalSaleDTO.ProductId = ProductId;

        var httpResponse = await repository.Post("/api/temporalSales", TemporalSaleDTO);
        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.TopEnd,
            ShowConfirmButton = false,
            Timer = 5000
        });
        await toast2.FireAsync(icon: SweetAlertIcon.Success, message: "Producto agregado al carro de compras.");
        navigationManager.NavigateTo("/");
    }
}
```

7. Probamos y hacemos el **commit**.

Mostrando y modificando el carro de compras

1. Agregamos este campo al **TemporalSaleDTO**:

```
public int Id { get; set; }
```

2. Agregamos la enumeración **OrderStatus**:

```
namespace Sales.Shared.Enums
{
    public enum OrderStatus
    {
        Nuevo,
        Despachado,
        Enviado,
```

```
        Confirmado,
        Cancelado
    }
}
```

3. Agregamos el **SaleDTO**:

```
using Sales.Shared.Enums;

namespace Sales.Shared.DTOs
{
    public class SaleDTO
    {
        public int Id { get; set; }

        public OrderStatus OrderStatus { get; set; }

        public string Remarks { get; set; } = string.Empty;
    }
}
```

4. Agregamos estos métodos al **TemporalSalesController**:

```
[HttpGet("{id:int}")]
public async Task<ActionResult> Get(int id)
{
    return Ok(await _context.TemporalSales
        .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));
}

[HttpPut]
public async Task<ActionResult> Put(TemporalSaleDTO temporalSaleDTO)
{
    var currentTemporalSale = await _context.TemporalSales.FirstOrDefaultAsync(x => x.Id == temporalSaleDTO.Id);
    if (currentTemporalSale == null)
    {
        return NotFound();
    }

    currentTemporalSale!.Remarks = temporalSaleDTO.Remarks;
    currentTemporalSale.Quantity = temporalSaleDTO.Quantity;

    _context.Update(currentTemporalSale);
    await _context.SaveChangesAsync();
    return Ok(temporalSaleDTO);
}

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

```
public async Task<IActionResult> DeleteAsync(int id)
{
    var temporalSale = await _context.TemporalSales.FirstOrDefaultAsync(x => x.Id == id);
    if (temporalSale == null)
    {
        return NotFound();
    }

    _context.Remove(temporalSale);
    await _context.SaveChangesAsync();
    return NoContent();
}
```

5. Modificamos nuestro **ShowCart.razor**.

```
@page "/Orders/ShowCart"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin, User")]

@if (temporalSales is null)
{
    <div class="spinner" />
}
else
{
    <GenericList MyList="temporalSales">
        <Body>
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-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="SaleDTO" OnValidSubmit="ConfirmOrderAsync">
                        <DataAnnotationsValidator />
                        <div class="mb-3">
                            <label>Comentarios:</label>
                            <div>
                                <InputText class="form-control" @bind-Value="@SaleDTO.Remarks" />
                                <ValidationMessage For="@(() => SaleDTO.Remarks)" />
                            </div>
                        </div>
                        <button class="btn btn-primary mb-3" type="submit"><i class="oi oi-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:200px"></th>
                        </tr>
                    </thead>
                    <tbody>
                        @foreach (var temporalSale in temporalSales)
                        {
                            <tr>
                                <td>
                                    @temporalSale.Product!.Name
                                </td>
                                <td>
                                    @temporalSale.Product!.Description
                                </td>
                                <td>
                                    @($"{temporalSale.Quantity:N2}")
                                </td>
                                <td>
                                    @($"{temporalSale.Product!.Price:C2}")
                                </td>
                                <td>
                                    @($"{temporalSale.Value:C2}")
                                </td>
                                <td>
                                    @temporalSale.Remarks
                                </td>
                                <td>
                                    <img src="@temporalSale.Product!.MainImage" style="width:100px;" />
                                </td>
                                <td>
                                    <a href="/Orders/ModifyTemporalSale/@temporalSale.Id" class="btn btn-warning"><i class="oi
oi-pencil" /> Editar</a>
                                    <button class="btn btn-danger" @onclick=@(() => Delete(temporalSale.Id))><i class="oi
oi-trash" /> Borrar</button>
                                </td>
                            </tr>
                        }
                    </tbody>
                </table>
            </div>
        </div>
    </Body>
</GenericList>
}
```

```
@code {
    public List<TemporalSale>? temporalSales { get; set; }
    private float sumQuantity;
    private decimal sumValue;

    public SaleDTO SaleDTO { get; set; } = new();

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

    private async Task LoadAsync()
    {
        try
        {
            var responseHppt = await repository.Get<List<TemporalSale>>("api/temporalSales");
            temporalSales = responseHppt.Response!;
            sumQuantity = temporalSales.Sum(x => x.Quantity);
            sumValue = temporalSales.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 temporalSaleId)
    {
        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.Delete($"api/temporalSales/{temporalSaleId}");

        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.TopEnd,
                ShowConfirmButton = false,
                Timer = 5000
            });
        await toast.FireAsync(icon: SweetAlertIcon.Success, message: "Producto eliminado del carro de compras.");
    }
}
```

6. Probamos lo que llevamos hasta el momento.

7. Dentro de **Pages/Orders** creamos el **ModifyTemporalSale**.

```
@page "/Orders/ModifyTemporalSale/{TemporalSaleId:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService

@if (loading)
{
    <div class="spinner" />
}
else
{
    <div class="card">
        <div class="card-header">
            <span>
                <i class="oi oi-star" /> @product!.Name
                <a class="btn btn-sm btn-success float-end" href="/"><i class="oi oi-arrow-thick-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="temporalSaleDTO" OnValidSubmit="UpdateCartAsync">
                    <DataAnnotationsValidator />
                    <div class="mb-3">
                        <label>Cantidad:</label>
                        <div>
                            <InputNumber class="form-control" @bind-Value="@temporalSaleDTO!.Quantity" />
                            <ValidationMessage For="@(() => temporalSaleDTO.Quantity)" />
                        </div>
                        <label>Comentarios:</label>
                        <div>
                            <InputText class="form-control" @bind-Value="@temporalSaleDTO.Remarks" />
                            <ValidationMessage For="@(() => temporalSaleDTO.Remarks)" />
                        </div>
                    </div>
                    <button class="btn btn-primary" type="submit"><i class="oi oi-check" /> Actualizar Carro de
Compras</button>
                </EditForm>
            </div>
        </div>
        <CarouselView Images="images" />
    </div>
</div>
}
```

```
@code {
    private List<string>? categories;
    private List<string>? images;
    private bool loading = true;
    private Product? product;
    private bool isAuthenticated;
    private TemporalSaleDTO? temporalSaleDTO;

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

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

    private async Task LoadTemporalSaleAsync()
    {
        loading = true;
        var httpResponse = await repository.Get<TemporalSale>($"/api/temporalSales/{TemporalSaleId}");

        if (httpResponse.Error)
        {
            loading = false;
            var message = await httpResponse.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
            return;
        }

        var temporalSale = httpResponse.Response!;
        temporalSaleDTO = new TemporalSaleDTO
        {
            Id = temporalSale.Id,
            ProductId = temporalSale.ProductId,
            Remarks = temporalSale.Remarks!,
            Quantity = temporalSale.Quantity
        };
        product = temporalSale.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.Put("/api/temporalSales", temporalSaleDTO);
        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.TopEnd,
            ShowConfirmButton = false,
            Timer = 5000
        });
        await toast2.FireAsync(icon: SweetAlertIcon.Success, message: "Producto modificado en el de compras.");
        navigationManager.NavigateTo("/");
    }
}
```

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

Procesando el pedido

1. Agregamos la entidad **Sale**:

```
using Sales.Shared.Enums;
using System.ComponentModel.DataAnnotations;

namespace Sales.Shared.Entities
{
    public class Sale
    {
        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<SaleDetail> SaleDetails { get; set; }

        [DisplayFormat(DataFormatString = "{0:N0}")]
        [Display(Name = "Líneas")]
        public int Lines => SaleDetails == null ? 0 : SaleDetails.Count;

        [DisplayFormat(DataFormatString = "{0:N2}")]
        [Display(Name = "Cantidad")]
        public float Quantity => SaleDetails == null ? 0 : SaleDetails.Sum(sd => sd.Quantity);

        [DisplayFormat(DataFormatString = "{0:C2}")]
        [Display(Name = "Valor")]
```

```
        public decimal Value => SaleDetails == null ? 0 : SaleDetails.Sum(sd => sd.Value);
    }
}
```

2. Agregamos la entidad **SaleDetail**:

```
using System.ComponentModel.DataAnnotations;

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

        public Sale? Sale { get; set; }

        public int SaleId { 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;
    }
}
```

3. Modificamos la entidad **Product**:

```
public ICollection<SaleDetail>? SaleDetails { get; set; }
```

4. Modificamos la entidad **User**:

```
public ICollection<Sale>? Sales { get; set; }
```

5. Agregamos las nuevas entidades al **DataContext**:

```
public DbSet<Sale> Sales { get; set; }

public DbSet<SaleDetail> SaleDetails { get; set; }
```

6. Agregamos la migración y actualizamos la base de datos.

7. En **API/Helpers** creamos el **IOrdersHelper**:

```
using Sales.Shared.Responses;

namespace Sales.API.Helpers
{
    public interface IOrdersHelper
    {
        Task<Response> ProcessOrderAsync(string email, string remarks);
    }
}
```

8. Luego hacemos la implementación en el **OrdersHelper**:

```
using Microsoft.EntityFrameworkCore;
using Sales.API.Data;
using Sales.Shared.Entities;
using Sales.Shared.Enums;
using Sales.Shared.Responses;

namespace Sales.API.Helpers
{
    public class OrdersHelper : IOrdersHelper
    {
        private readonly DataContext _context;

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

        public async Task<Response> ProcessOrderAsync(string email, string remarks)
        {
            var user = await _context.Users.FirstOrDefaultAsync(x => x.Email == email);
            if (user == null)
            {
                return new Response
                {
                    IsSuccess = false,
                    Message = "Usuario no válido"
                };
            }

            var temporalSales = await _context.TemporalSales
                .Include(x => x.Product)
                .Where(x => x.User!.Email == email)
                .ToListAsync();
            Response response = await CheckInventoryAsync(temporalSales);
            if (!response.IsSuccess)
            {
                return response;
            }

            Sale sale = new()
```

```
            {
                Date = DateTime.UtcNow,
                User = user,
                Remarks =remarks,
                SaleDetails = new List<SaleDetail>(),
                OrderStatus = OrderStatus.Nuevo
            };

            foreach (var temporalSale in temporalSales)
            {
                sale.SaleDetails.Add(new SaleDetail
                {
                    Product = temporalSale.Product,
                    Quantity = temporalSale.Quantity,
                    Remarks = temporalSale.Remarks,
                });

                Product? product = await _context.Products.FindAsync(temporalSale.Product!.Id);
                if (product != null)
                {
                    product.Stock -= temporalSale.Quantity;
                    _context.Products.Update(product);
                }

                _context.TemporalSales.Remove(temporalSale);
            }

            _context.Sales.Add(sale);
            await _context.SaveChangesAsync();
            return response;
        }

        private async Task<Response> CheckInventoryAsync(List<TemporalSale> temporalSales)
        {
            Response response = new() { IsSuccess = true };
            foreach (var temporalSale in temporalSales)
            {
                Product? product = await _context.Products.FirstOrDefaultAsync(x => x.Id == temporalSale.Product!.Id);
                if (product == null)
                {
                    response.IsSuccess = false;
                    response.Message = $"El producto {temporalSale.Product!.Name}, ya no está disponible";
                    return response;
                }
                if (product.Stock < temporalSale.Quantity)
                {
                    response.IsSuccess = false;
                    response.Message = $"Lo sentimos no tenemos existencias suficientes del producto
{temporalSale.Product!.Name}, para tomar su pedido. Por favor disminuir la cantidad o sustituirlo por otro.";
                    return response;
                }
            }
            return response;
        }
```

```
    }
}
```

9. Lo inyectamos en el **Program** del **API**:

```
builder.Services.AddScoped<IOrdersHelper, OrdersHelper>();
```

10. Creamos el **SalesController**:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Sales.API.Helpers;
using Sales.Shared.DTOs;

namespace Sales.API.Controllers
{
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("/api/sales")]
    public class SalesController : ControllerBase
    {
        private readonly IOrdersHelper _ordersHelper;

        public SalesController(IOrdersHelper ordersHelper)
        {
            _ordersHelper = ordersHelper;
        }

        [HttpPost]
        public async Task<ActionResult> Post(SaleDTO saleDTO)
        {
            var response = await _ordersHelper.ProcessOrderAsync(User.Identity!.Name!, saleDTO.Remarks);
            if (response.IsSuccess)
            {
                return NoContent();
            }

            return BadRequest(response.Message);
        }
    }
}
```

11. Copiamos las imagenes en el **WWWRoot**.

12. Creamos la página de confirmación de pedido **Pages/Orders/SaleConfirmed**:

```
@page "/Orders/SaleConfirmed"

<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">Volver al inicio</a>
```

```
</center>
```

13. Modificamos **ConfirmOrderAsync** del **ShowCart**:

```
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 httpResponse = await repository.Post("/api/sales", SaleDTO);
    if (httpResponse.Error)
    {
        var message = await httpResponse.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        return;
    }

    navigationManager.NavigateTo("/Orders/SaleConfirmed");
}
```

14. Probamos y hacemos el **commit**.

Administrar pedidos

1. Agregamos estos métodos al **SalesController**, primero inyectamos el **DataContext** y el **IUserHelper**:

```
[HttpGet]
public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)
{
    var user = await _context.Users.FirstOrDefaultAsync(x => x.Email == User.Identity!.Name);
    if (user == null)
    {
        return BadRequest("User not valid.");
    }

    var queryable = _context.Sales
        .Include(s => s.User)
        .Include(s => s.SaleDetails)
        .ThenInclude(sd => sd.Product)
        .AsQueryable();

    var isAdmin = await _userHelper.IsUserInRoleAsync(user, UserType.Admin.ToString());
    if (!isAdmin)
```

```
    {
        queryable = queryable.Where(s => s.User!.Email == User.Identity!.Name);
    }

    return Ok(await queryable
        .OrderByDescending(x => x.Date)
        .Paginate(pagination)
        .ToListAsync());
}

[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)
{
    var user = await _context.Users.FirstOrDefaultAsync(x => x.Email == User.Identity!.Name);
    if (user == null)
    {
        return BadRequest("User not valid.");
    }

    var queryable = _context.Sales
        .AsQueryable();

    var isAdmin = await _userHelper.IsUserInRoleAsync(user, UserType.Admin.ToString());
    if (!isAdmin)
    {
        queryable = queryable.Where(s => s.User!.Email == User.Identity!.Name);
    }

    double count = await queryable.CountAsync();
    double totalPages = Math.Ceiling(count / pagination.RecordsNumber);
    return Ok(totalPages);
}
```

2. Creamos en **Pages/Orders** el **SaleIndex**:

```
@page "/sales"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (Sales is null)
{
    <div class="spinner" />
}
else
{
    <GenericList MyList="Sales">
        <Body>
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-dollar" /> Pedidos
                    </span>
```

```
            </div>
            <div class="card-body">
                <Pagination CurrentPage="currentPage"
                            TotalPages="totalPages"
                            SelectedPage="SelectedPageAsync" />

                <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 Sales)
                        {
                            <tr>
                                <td>
                                    @($"{sale.Date:yyyy/MM/dd hh:mm tt}")
                                </td>
                                <td>
                                    @sale.User!.FullName
                                </td>
                                <td>
                                    @sale.Remarks
                                </td>
                                <td>
                                    @sale.OrderStatus
                                </td>
                                <td>
                                    @sale.Lines
                                </td>
                                <td>
                                    @($"{sale.Quantity:N2}")
                                </td>
                                <td>
                                    @($"{sale.Value:C2}")
                                </td>
                                <td>
                                    <a href="/saleDetails/@sale.Id" class="btn btn-info"><i class="oi oi-info" /> Detalles</a>
                                </td>
                            </tr>
                        }
                    </tbody>
                </table>
            </div>
        </div>
    </Body>
```

```
    </GenericList>
}

@code {
    private int currentPage = 1;
    private int totalPages;

    public List<Sale>? Sales { get; set; }

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    protected async override 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);
        }

        string url1 = $"api/sales?page={page}";
        string url2 = $"api/sales/totalPages";

        try
        {
            var responseHppt = await repository.Get<List<Sale>>(url1);
            var responseHppt2 = await repository.Get<int>(url2);
            Sales = responseHppt.Response!;
            totalPages = responseHppt2.Response!;
        }
        catch (Exception ex)
        {
            await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
        }
    }
}
```

3. Modificamos el **NavMenu.razor**:

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="countries">
        <span class="oi oi-globe" aria-hidden="true"></span> Países
    </NavLink>
```

```
</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="sales">
        <span class="oi oi-dollar" aria-hidden="true"></span> Pedidos
    </NavLink>
</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="products">
        <span class="oi oi-star" aria-hidden="true"></span> Productos
    </NavLink>
</div>
```

4. Probamos lo que llevamos hasta el momento.

5. Adicioanamos este método al **SalesController**:

```
[HttpGet("{id:int}")]
public async Task<ActionResult> Get(int id)
{
    var sale = await _context.Sales
        .Include(s => s.User!)
        .ThenInclude(u => u.City!)
        .ThenInclude(c => c.State!)
        .ThenInclude(s => s.Country)
        .Include(s => s.SaleDetails!)
        .ThenInclude(sd => sd.Product)
        .ThenInclude(p => p.ProductImages)
        .FirstOrDefaultAsync(s => s.Id == id);

    if (sale== null)
    {
        return NotFound();
    }

    return Ok(sale);
}
```

6. Creamos el **SaleDetails**:

```
@page "/orders/saleDetails/{SaleId:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (sale is null)
{
    <div class="spinner" />
}
else
{
    <GenericList MyList="sale.SaleDetails!.ToList()">
        <Body>
            <div class="card">
```

```
            <div class="card-header">
                <span>
                    <i class="oi oi-dollar"></i> @sale.User!.FullName
                    @if (sale.OrderStatus == OrderStatus.Nuevo)
                    {
                        <button class="btn btn-sm btn-danger float-end mx-2" @onclick=@(() => CancelSaleAsync())><i class="oi oi-trash" /> Cancelar</button>
                        <button class="btn btn-sm btn-primary float-end mx-2" @onclick=@(() => DispatchSaleAsync())><i class="oi oi-external-link" /> Despachar</button>
                    }
                    else if (sale.OrderStatus == OrderStatus.Despachado)
                    {
                        <button class="btn btn-sm btn-warning float-end mx-2" @onclick=@(() => SendSaleAsync())><i class="oi oi-location" /> Enviar</button>
                    }
                    else if (sale.OrderStatus == OrderStatus.Enviado)
                    {
                        <button class="btn btn-sm btn-dark float-end mx-2" @onclick=@(() => ConfirmSaleAsync())><i class="oi oi-thumb-up" /> Confirmar</button>
                    }
                    <a class="btn btn-sm btn-success float-end" href="/sales"><i class="oi oi-arrow-thick-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>@sale.User.FullName</strong></p>
                    <p><strong>@sale.User.Document</strong></p>
                    <p><strong>@sale.User.PhoneNumber</strong></p>
                    <p><strong>@sale.User.UserName</strong></p>
                    <p><strong>@sale.User.Address, @sale.User.City!.Name, @sale.User.City.State!.Name, @sale.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>@sale.OrderStatus</strong></p>
                    <p><strong>@($"{sale.Date.ToLocalTime():yyyy/MM/dd hh:mm tt}")</strong></p>
                    <p><strong>@(string.IsNullOrEmpty(sale.Remarks) ? "NA" : sale.Remarks)</strong></p>
                    <p><strong>@sale.Lines</strong></p>
                    <p><strong>@($"{sale.Quantity:N2}")</strong></p>
                    <p><strong>@($"{sale.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 sale.SaleDetails!)
                        {
                            <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>
}

@code {
    private Sale? sale;

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

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

    private async Task LoadAsync()
    {
        var responseHppt = await repository.Get<Sale>($"api/sales/{SaleId}");
        if (responseHppt.Error)
        {
            if (responseHppt.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/sales");
```

```
                return;
            }
            var messageError = await responseHppt.GetErrorMessageAsync();
            await sweetAlertService.FireAsync("Error", messageError, SweetAlertIcon.Error);
            return;
        }
        sale = responseHppt.Response;
    }

    private void CancelSaleAsync()
    {

    }

    private void DispatchSaleAsync()
    {

    }

    private void SendSaleAsync()
    {

    }

    private void ConfirmSaleAsync()
    {

    }
}
```

7. Agregamos estos métodos al **SalesController**:

```
[HttpPut]
public async Task<ActionResult> Put(SaleDTO saleDTO)
{
    var user = await _userHelper.GetUserAsync(User.Identity!.Name!);
    if (user == null)
    {
        return NotFound();
    }

    var isAdmin = await _userHelper.IsUserInRoleAsync(user, UserType.Admin.ToString());
    if (!isAdmin)
    {
        return BadRequest("Solo permitido para administradores.");
    }

    var sale = await _context.Sales
        .Include(s => s.SaleDetails)
        .FirstOrDefaultAsync(s => s.Id == saleDTO.Id);
    if (sale == null)
    {
        return NotFound();
    }
```

```
    if (saleDTO.OrderStatus == OrderStatus.Cancelado)
    {
        await ReturnStockAsync(sale);
    }

    sale.OrderStatus = saleDTO.OrderStatus;
    _context.Update(sale);
    await _context.SaveChangesAsync();
    return Ok(sale);
}

private async Task ReturnStockAsync(Sale sale)
{
    foreach (var saleDetail in sale.SaleDetails!)
    {
        var product = await _context.Products.FirstOrDefaultAsync(p => p.Id == saleDetail.ProductId);
        if (product != null)
        {
            product.Stock += saleDetail.Quantity;
        }
    }
    await _context.SaveChangesAsync();
}
```

8. Modificamos estos métodos al **SalesDetails.razor**:

```
private async Task CancelSaleAsync()
{
    await ModifyTemporalSale("cancelar", OrderStatus.Cancelado);
}

private async Task DispatchSaleAsync()
{
    await ModifyTemporalSale("despachar", OrderStatus.Despachado);
}

private async Task SendSaleAsync()
{
    await ModifyTemporalSale("enviar", OrderStatus.Enviado);
}

private async Task ConfirmSaleAsync()
{
    await ModifyTemporalSale("confirmar", OrderStatus.Confirmado);
}

private async Task ModifyTemporalSale(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 saleDTO = new SaleDTO
        {
            Id = SaleId,
            OrderStatus = status
        };

    var responseHTTP = await repository.Put("api/sales", saleDTO);
    if (responseHTTP.Error)
    {
        var mensajeError = await responseHTTP.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        return;
    }

    navigationManager.NavigateTo("/sales");
}
```

9. Probamos y hacemos el **commit**.

Ver estado de mis pedidos

10. Modificamos el método **Put** en el **SalesController**:

```
var isAdmin = await _userHelper.IsUserInRoleAsync(user, UserType.Admin.ToString());
if (!isAdmin && saleDTO.OrderStatus != OrderStatus.Cancelado)
{
    return BadRequest("Solo permitido para administradores.");
}
```

11. Agregamos estas líneas al **NavMenu.razor**:

```
        </AuthorizeView>
        <AuthorizeView Roles="User">
            <Authorized>
                <div class="nav-item px-3">
                    <NavLink class="nav-link" href="sales">
                        <span class="oi oi-dollar" aria-hidden="true"></span> Ver Mis Pedidos
                    </NavLink>
                </div>
            </Authorized>
        </AuthorizeView>
    </nav>
</div>
```

12. Modificamos el **SaleIndex**:

```
@attribute [Authorize(Roles = "Admin, User")]
```

13. Modificamos el **SaleDetails**:

```
…
@attribute [Authorize(Roles = "Admin, User")]
…
<div class="card-header">
   <span>
      <i class="oi oi-dollar"></i> @sale.User!.FullName
      @if (sale.OrderStatus == OrderStatus.Nuevo)
      {
         <button class="btn btn-sm btn-danger float-end mx-2" @onclick=@(() => CancelSaleAsync())><i class="oi
oi-trash" /> Cancelar</button>
         <AuthorizeView Roles="Admin">
            <Authorized>
               <button class="btn btn-sm btn-primary float-end mx-2" @onclick=@(() => DispatchSaleAsync())><i class="oi
oi-external-link" /> Despachar</button>
            </Authorized>
         </AuthorizeView>
      }
      <AuthorizeView Roles="Admin">
         <Authorized>
            @if (sale.OrderStatus == OrderStatus.Despachado)
            {
               <button class="btn btn-sm btn-warning float-end mx-2" @onclick=@(() => SendSaleAsync())><i class="oi
oi-location" /> Enviar</button>
            }
            @if (sale.OrderStatus == OrderStatus.Enviado)
            {
               <button class="btn btn-sm btn-dark float-end mx-2" @onclick=@(() => ConfirmSaleAsync())><i class="oi
oi-thumb-up" /> Confirmar</button>
            }
         </Authorized>
      </AuthorizeView>
      <a class="btn btn-sm btn-success float-end" href="/sales"><i class="oi oi-arrow-thick-left" /> Regresar</a>
   </span>
</div>
```

14. Probamos y hacemos el **commit**.

Administrar usuarios y crear nuevos administradores

15. Adicionamos estos métodos al **AccountController** (primero inyectamos el **DataContext**):

```
[HttpGet("all")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult> GetAll([FromQuery] PaginationDTO pagination)
{
   var queryable = _context.Users
      .Include(u => u.City)
      .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 Ok(await queryable
        .OrderBy(x => x.FirstName)
        .ThenBy(x => x.LastName)
        .Paginate(pagination)
        .ToListAsync());
}

[HttpGet("totalPages")]
public async Task<ActionResult> GetPages([FromQuery] 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 Ok(totalPages);
}
```

16. Adicionamos estas línea al **NavMenu**:

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="products">
        <span class="oi oi-star" aria-hidden="true"></span> Productos
    </NavLink>
</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="users">
        <span class="oi oi-people" aria-hidden="true"></span> Usuarios
    </NavLink>
</div>
```

17. Creamos el **UserIndex** dentro de **Pages/Auth**:

```
@page "/users"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService sweetAlertService
@attribute [Authorize(Roles = "Admin")]

@if (Users is null)
{
    <div class="spinner" />
}
```

```
else
{
    <GenericList MyList="Users">
        <Body>
            <div class="card">
                <div class="card-header">
                    <span>
                        <i class="oi oi-people"></i> Usuarios
                        <a class="btn btn-sm btn-primary float-end" href="/register/?IsAdmin=true"><i class="oi oi-plus"></i>
Adicionar Administrador</a>
                    </span>
                </div>
                <div class="card-body">
                    <div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">
                        <div>
                            <input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar usuario..."
@bind-value="Filter" />
                        </div>
                        <div class="mx-1">
                            <button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi
oi-layers" /> Filtrar</button>
                            <button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban"
/> Limpiar</button>
                        </div>
                    </div>

                    <Pagination CurrentPage="currentPage"
                        TotalPages="totalPages"
                        SelectedPage="SelectedPage" />

                    <table class="table table-striped">
                        <thead>
                            <tr>
                                <th>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</td>
                                    <td>@user.EmailConfirmed</td>
```

```
                            <td>@user.UserType</td>
                        </tr>
                    }
                </tbody>
            </table>
        </div>
    </div>
</Body>
</GenericList>
}

@code {
    public List<User>? Users { get; set; }
    private int currentPage = 1;
    private int totalPages;

    [Parameter]
    [SupplyParameterFromQuery]
    public string Page { get; set; } = "";

    [Parameter]
    [SupplyParameterFromQuery]
    public string Filter { get; set; } = "";

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

    private async Task SelectedPage(int page)
    {
        currentPage = page;
        await LoadAsync(page);
    }

    private async Task LoadAsync(int page = 1)
    {
        if (!string.IsNullOrWhiteSpace(Page))
        {
            page = Convert.ToInt32(Page);
        }

        string url1 = string.Empty;
        string url2 = string.Empty;

        if (string.IsNullOrEmpty(Filter))
        {
            url1 = $"api/accounts/all?page={page}";
            url2 = $"api/accounts/totalPages";
        }
        else
        {
            url1 = $"api/accounts/all?page={page}&filter={Filter}";
            url2 = $"api/accounts/totalPages?filter={Filter}";
```

```
        }

        try
        {
            var responseHppt = await repository.Get<List<User>>(url1);
            var responseHppt2 = await repository.Get<int>(url2);
            Users = responseHppt.Response!;
            totalPages = responseHppt2.Response!;
        }
        catch (Exception ex)
        {
            await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);
        }
    }

    private async Task ApplyFilterAsync()
    {
        await LoadAsync();
    }

    private async Task CleanFilterAsync()
    {
        Filter = string.Empty;
        await LoadAsync();
    }
}
```

18. Modificamos el **Register.razor**:

```
…
[Parameter]
[SupplyParameterFromQuery]
public bool IsAdmin { get; set; }
…
private async Task CreteUserAsync()
{
    userDTO.UserName = userDTO.Email;

    if (IsAdmin)
    {
        userDTO.UserType = UserType.Admin;
    }
    else
    {
        userDTO.UserType = UserType.User;
    }

    var responseHttp = await repository.Post<UserDTO>("/api/accounts/CreateUser", userDTO);
    if (responseHttp.Error)
    {
        var message = await responseHttp.GetErrorMessageAsync();
        await sweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);
        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("/");
}
```

19. Probamos y hacemos el **commit**.

Corrección para que corrar el App en Mac

20. 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");
…
```

21. Probamos y hacemos el **commit**.

PARTE II - App Móvil

Páginas

Vamos hacer unas pruebas para irnos familiarizando con MAUI.

1. En el proyecto **Mobile** creamos la carpeta **PagesDemo** y dentro de esta creamos el **ContentPageDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.PagesDemo.ContentPageDemo"
             Title="ContentPageDemo">
    <VerticalStackLayout>
        <Label
            Text="Welcome to .NET MAUI!"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
    </VerticalStackLayout>
</ContentPage>
```

2. Modificamos el **App.xaml.cs** para indicar que nuestra página inicial es **MainPage**:

```
namespace Sales.Mobile
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new ContentPageDemo();
        }
    }
}
```

3. Probamos.

4. Cambiamos el método **OnCounterClicked** de la **MainPage** para que hagamos nuestra primer navegación:

```
private void OnCounterClicked(object sender, EventArgs e)
{
    Navigation.PushAsync(new ContentPageDemo());
}
```

5. Volvemos a cambiar que nuestra página de inicio sea **MainPage**, la cual devemos llamar dentro de un **NavigationPage**:

```
public App()
{
    InitializeComponent();

    MainPage = new NavigationPage(new MainPage());
}
```

6. Probamos.

7. Ahora juguemos con algunas propiedades de la barra de navegación:

```
using Sales.Mobile.PagesDemo;

namespace Sales.Mobile
{
   public partial class App : Application
   {
      public App()
      {
         InitializeComponent();

         var navPage = new NavigationPage(new MainPage());
         navPage.BarBackgroundColor = Colors.Chocolate;
         navPage.BarTextColor = Colors.White;
         MainPage = navPage;
      }
   }
}
```

8. Probamos.

9. Modificacmos nuestra **ContentPageDemo.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.PagesDemo.ContentPageDemo"
             Title="ContentPageDemo">
   <VerticalStackLayout Padding="10">
      <Label
         Text="Welcome to .NET MAUI!"
         VerticalOptions="Center"
         HorizontalOptions="Center" />
      <Button
         Text="Regresar"
         Clicked="Button_Clicked"/>
   </VerticalStackLayout>
</ContentPage>
```

10. Modificacmos nuestra **ContentPageDemo.xaml.cs**:

```
void Button_Clicked(System.Object sender, System.EventArgs e)
{
         Navigation.PopAsync();
}
```

11. Probamos.

12. Creamos dentro de **PagesDemo** el **FlyoutPageDemo.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<FlyoutPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="Sales.Mobile.PagesDemo.FlyoutPageDemo"
            Title="FlyoutPageDemo">
    <FlyoutPage.Flyout>
        <ContentPage Title="My App">
        <Label
            Text="Hello from flyout"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
        </ContentPage>
    </FlyoutPage.Flyout>
    <FlyoutPage.Detail>
        <ContentPage>
        <Label
            Text="Hello from detail"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
        </ContentPage>
    </FlyoutPage.Detail>
</FlyoutPage>
```

13. Modificamos el **FlyoutPageDemo.xaml.cs**:

```
namespace Sales.Mobile.PagesDemo;

public partial class FlyoutPageDemo : FlyoutPage
{
        public FlyoutPageDemo()
        {
                InitializeComponent();
        }
}
```

14. Cambiamos la página de inicio en el **App.xaml.cs**:

```
public App()
{
    InitializeComponent();

    var navPage = new NavigationPage(new MainPage());
    navPage.BarBackgroundColor = Colors.Chocolate;
    navPage.BarTextColor = Colors.White;

    MainPage = new FlyoutPageDemo();
}
```

15. Probamos.

16. Cambiamos algunas propiedades en el **FlyoutPageDemo.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<FlyoutPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="Sales.Mobile.PagesDemo.FlyoutPageDemo"
            Title="FlyoutPageDemo"
            FlyoutLayoutBehavior="Split">
    <FlyoutPage.Flyout>
        <ContentPage
            BackgroundColor="Aqua"
            Title="My App">
        <Label
            Text="Hello from flyout"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
        </ContentPage>
    </FlyoutPage.Flyout>
    <FlyoutPage.Detail>
        <ContentPage
            BackgroundColor="Coral">
        <Label
            Text="Hello from detail"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
        </ContentPage>
    </FlyoutPage.Detail>
</FlyoutPage>
```

17. Probamos.
18. Ahora agregamos a la carpeta **PagesDemo** la **TabbedPageDemo.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="Sales.Mobile.PagesDemo.TabbedPageDemo"
            Title="TabbedPageDemo">
    <ContentPage Title="Page 1"></ContentPage>
    <ContentPage Title="Page 2"></ContentPage>
    <ContentPage Title="Page 3"></ContentPage>
</TabbedPage>
```

19. Modificamos la **TabbedPageDemo.xaml.cs**:

```
namespace Sales.Mobile.PagesDemo;

public partial class TabbedPageDemo : TabbedPage
{
        public TabbedPageDemo()
        {
                InitializeComponent();
        }
}
```

20. Cambiamos la página de inicio en el **App.xaml.cs**:

```
public App()
{
```

```
    InitializeComponent();

    var navPage = new NavigationPage(new MainPage());
    navPage.BarBackgroundColor = Colors.Chocolate;
    navPage.BarTextColor = Colors.White;

    MainPage = new TabbedPageDemo();
}
```

21. Modificamos el **TabbedPageDemo.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="Sales.Mobile.PagesDemo.TabbedPageDemo"
            Title="TabbedPageDemo"
            BackgroundColor="DarkBlue"
            BarTextColor="LightSalmon"
            SelectedTabColor="DarkRed"
            UnselectedTabColor="DarkKhaki">
    <ContentPage
        BackgroundColor="Aqua"
        IconImageSource="dotnet_bot.svg"
        Title="Page 1">
        <Label
            Text="Page 1"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
    </ContentPage>
    <ContentPage
        BackgroundColor="Beige"
        IconImageSource="dotnet_bot.svg"
        Title="Page 2">
        <Label
            Text="Page 2"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
    </ContentPage>
    <ContentPage
        BackgroundColor="Coral"
        IconImageSource="dotnet_bot.svg"
        Title="Page 3">
        <Label
            Text="Page 3"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
    </ContentPage>
</TabbedPage>
```

22. Probamos y hacemos el **Commit**.

23. Creamos una nueva carpeta en el proyecto **Mobile** llamada **ControlsDemo** y dentro de esta creamos la página **PresentationControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.PresentationControlsDemo"
             Title="PresentationControlsDemo">
    <VerticalStackLayout>
        <BoxView
            BackgroundColor="Navy"
            HeightRequest="200"/>
    </VerticalStackLayout>
</ContentPage>
```

24. Establecemos como página de inicio nuestro **PresentationControlsDemo**:

```
MainPage = new PresentationControlsDemo();
```

25. Probamos.

26. Modificamos la página **PresentationControlsDemo**:

```
<Label
    Text="Este es un label"
    TextColor="Black"
    FontAttributes="Bold"
    FontSize="Large"
    HorizontalTextAlignment="Center"/>
```

27. Probamos.

28. Modificamos la página **PresentationControlsDemo**:

```
<Ellipse
    Fill="DarkRed"
    Stroke="DarkGreen"
    StrokeThickness="10"
    HeightRequest="200"
    HorizontalOptions="Center"
    WidthRequest="200"/>
```

29. Probamos.

30. Modificamos la página **PresentationControlsDemo**:

```
<Line
    Stroke="Purple"
    X1="0"
    Y1="0"
    X2="100"
    Y2="50"
```

```
    StrokeThickness="10"/>
```

31. Probamos.

32. Modificamos la página **PresentationControlsDemo**:

```
<Rectangle
    Fill="Aqua"
    Stroke="Black"
    StrokeThickness="5"
    HeightRequest="50"
    HorizontalOptions="End"
    WidthRequest="150"
    RadiusX="10"
    RadiusY="10"/>
```

33. Probamos.

34. Modificamos la página **PresentationControlsDemo**:

```
<Polygon
    Fill="LightBlue"
    Points="40,10 70,80 10,50"
    Stroke="DarkBlue"
    StrokeThickness="5"/>
<Polygon
    Fill="Yellow"
    Points="40,10 70,80 10,50"
    Stroke="Green"
    StrokeDashArray="1,1"
    StrokeDashOffset="6"
    StrokeThickness="5"/>
```

35. Probamos.

36. Modificamos la página **PresentationControlsDemo**:

```
<Polyline
    Points="0,0 10,30 15,0 18,60 23,30 35,30 40,0 43,60 48,30 100,30"
    Stroke="Red"/>
<Polyline
    Points="0 48, 0 144, 96 150, 100 0, 192 0, 192 96, 50 96, 48 192, 150 200 144 48"
    Fill="Blue"
    Stroke="Red"
    StrokeThickness="3" />
```

37. Probamos.

38. Modificamos la página **PresentationControlsDemo**:

```
<Path
    Aspect="Uniform"
    Data="M 10,100 L 100,100 100,50Z"
    HorizontalOptions="Center"
```

```
    Stroke="Black"/>
```

39. Probamos.

40. Modificamos la página **PresentationControlsDemo**:

```
<Border
    Stroke="#C49B33"
    StrokeThickness="4"
    Background="#2B0B98"
    Padding="16,8"
    HorizontalOptions="Center">
    <Border.StrokeShape>
        <RoundRectangle CornerRadius="40,0,0,40" />
    </Border.StrokeShape>
    <Label
        Text="Welcome to .NET MAUI!"
        VerticalOptions="Center"
        HorizontalOptions="Center"
        TextColor="White"/>
</Border>
```

41. Probamos.

42. Modificamos la página **PresentationControlsDemo**:

```
<Frame
    Margin="5"
    BackgroundColor="Azure"
    Padding="10">
    <Image Source="dotnet_bot.svg"/>
</Frame>
```

43. Probamos.

44. Modificamos la página **PresentationControlsDemo**:

```
<WebView
    HeightRequest="500"
    Source="https://www.google.com/"/>
```

45. Probamos y hacemos el **Commit**.

Controles que inician comandos

1. Creamos la página **CommandsControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.CommandsControlsDemo"
             Title="CommandsControlsDemo">
    <VerticalStackLayout
        Padding="10">
```

```
        <Button
            x:Name="btnTest"
            Text="Click Me!"
            Clicked="btnTest_Clicked"/>
    </VerticalStackLayout>
</ContentPage>
```

2. Modificamos la página **CommandsControlsDemo.xaml.cs**:

```
void btnTest_Clicked(System.Object sender, System.EventArgs e)
{
    DisplayAlert("Test", "This is a demo", "Ok");
}
```

3. Establecemos como página de inicio nuestro **CommandsControlsDemo**:

```
<ImageButton
    Source="dotnet_bot.svg"
    Clicked="btnTest_Clicked"/>
```

4. Probamos.

5. Modificamos la página **CommandsControlsDemo**:

```
<RadioButton
    CheckedChanged="RadioButton_CheckedChanged"
    Content="Option 1"
    GroupName="group1"/>
<RadioButton
    CheckedChanged="RadioButton_CheckedChanged"
    Content="Option 2"
    GroupName="group1"/>
<RadioButton
    CheckedChanged="RadioButton_CheckedChanged"
    Content="Option 3"
    GroupName="group2"/>
<RadioButton
    CheckedChanged="RadioButton_CheckedChanged"
    Content="Option 4"
    GroupName="group2"/>
```

6. Modificamos la página **CommandsControlsDemo.xaml.cs**:

```
void RadioButton_CheckedChanged(object sender, CheckedChangedEventArgs e)
{
    DisplayAlert("RadioButton", $"Changed: {e.Value}", "Ok");
}
```

7. Probamos.

8. Modificamos la página **CommandsControlsDemo**:

```
<SwipeView>
    <SwipeView.LeftItems>
```

```
        <SwipeItems>
            <SwipeItem
                BackgroundColor="LightGreen"
                IconImageSource="dotnet_bot.svg"
                Invoked="SwipeItem_Invoked"
                Text="Favorite"/>
            <SwipeItem
                BackgroundColor="LightPink"
                IconImageSource="dotnet_bot.svg"
                Invoked="SwipeItem_Invoked"
                Text="Delete"/>
        </SwipeItems>
    </SwipeView.LeftItems>
    <Grid
        BackgroundColor="LightGray"
        HeightRequest="60"
        WidthRequest="300">
        <Label
            HorizontalOptions="Center"
            Text="Swipe Right"
            VerticalOptions="Center"/>
    </Grid>
</SwipeView>
```

9. Modificamos la página **CommandsControlsDemo.xaml.cs**:

```
void SwipeItem_Invoked(object sender, EventArgs e)
{
    DisplayAlert("SwipeView", $"Element Tapped", "Ok");
}
```

10. Probamos y hacemos el **Commit**.

Controles para establecer valores

1. Dentro de **ControlsDemo** creamos el **InputControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.InputControlsDemo"
             Title="InputControlsDemo">
    <VerticalStackLayout>
        <CheckBox
            IsChecked="True" />
    </VerticalStackLayout>
</ContentPage>
```

2. Establecemos la página **InputControlsDemo** como página de inicio:

```
MainPage = new InputControlsDemo();
```

3. Probamos.

4. Modificamos **InputControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.InputControlsDemo"
             Title="InputControlsDemo">
    <VerticalStackLayout>
        <CheckBox
            IsChecked="True" />

        <Slider
            x:Name="slider"
            Minimum="0"
            Maximum="10"
            MinimumTrackColor="Yellow"
            MaximumTrackColor="Green"
            ThumbColor="DarkRed"
            ValueChanged="slider_ValueChanged"/>
        <Label
            x:Name="lblSlider"/>
    </VerticalStackLayout>
</ContentPage>
```

5. Modificamos **InputControlsDemo.xaml.cs**:

```
void slider_ValueChanged(System.Object sender, Microsoft.Maui.Controls.ValueChangedEventArgs e)
{
    lblSlider.Text = slider.Value.ToString();
}
```

6. Probamos.

7. Modificamos **InputControlsDemo**:

```
<Stepper
    x:Name="stepper"
    ValueChanged="stepper_ValueChanged"
    Maximum="10"
    Minimum="2"
    Increment="2"/>
```

8. Modificamos **InputControlsDemo.xaml.cs**:

```
void stepper_ValueChanged(System.Object sender, Microsoft.Maui.Controls.ValueChangedEventArgs e)
{
    if (stepper != null)
    {
        lblSlider.Text = stepper.Value.ToString();
    }
}
```

9. Probamos.

10. Modificamos **InputControlsDemo**:

```
<Switch
    IsToggled="True"/>
```

11. Probamos.

```
<DatePicker />
<TimePicker/>
```

12. Probamos y hacemos el **Commit**.

Controles de edición de texto

1. Dentro de **ControlsDemo** creamos el **TextControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.TextControlsDemo"
             Title="TextControlsDemo">
    <VerticalStackLayout>
        <Entry
            Placeholder="Name"
            x:Name="txtName"
            PlaceholderColor="DarkGreen"
            IsPassword="True"
            Keyboard="Telephone"
            TextChanged="Entry_TextChanged"
            Completed="Entry_Completed"/>

        <Editor
            AutoSize="TextChanges"/>
    </VerticalStackLayout>
</ContentPage>
```

2. Modificamos el **TextControlsDemo.xaml.cs**:

```
using System.Diagnostics;

namespace Sales.Mobile.ControlsDemo;

public partial class TextControlsDemo : ContentPage
{
        public TextControlsDemo()
        {
                InitializeComponent();
        }

    void Entry_TextChanged(object sender, TextChangedEventArgs e)
    {

    }
```

```
    void Entry_Completed(object sender, EventArgs e)
    {
        Debug.WriteLine(txtName.Text);
    }
}
```

3. Establecemos la página **TextControlsDemo** como página de inicio:

```
MainPage = new TextControlsDemo();
```

4. Probamos y hacemos el **Commit**.

Controles para indicar actividad

1. Dentro de **ControlsDemo** creamos el **ActivityControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.ActivityControlsDemo"
             Title="ActivityControlsDemo">
    <VerticalStackLayout>
        <ActivityIndicator
            IsRunning="True"
            Color="Blue"/>
        <ProgressBar
            Progress=".5"/>
    </VerticalStackLayout>
</ContentPage>
```

2. Colocamos el **ActivityControlsDemo** como página de inicio:

```
MainPage = new ActivityControlsDemo();
```

3. Probamos y hacemos el **Commit**.

Controles para desplegar colecciones

1. Dentro de **ControlsDemos** creamos el **CollectionsControlsDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.ControlsDemo.CollectionsControlsDemo"
             Title="CollectionsControlsDemo">
    <VerticalStackLayout>
        <CarouselView
            IndicatorView="indicatorView">
            <CarouselView.ItemsSource>
                <x:Array Type="{x:Type x:String}">
                    <x:String>mono</x:String>
                    <x:String>monodroid</x:String>
                    <x:String>monotouch</x:String>
                    <x:String>monorail</x:String>
```

```
                <x:String>monodevelop</x:String>
                <x:String>monotone</x:String>
                <x:String>monopoly</x:String>
                <x:String>monomodal</x:String>
                <x:String>mononucleosis</x:String>
            </x:Array>
        </CarouselView.ItemsSource>
        <CarouselView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                    <Frame
                        Margin="20"
                        BorderColor="DarkGray"
                        CornerRadius="5"
                        HasShadow="True"
                        HeightRequest="100"
                        HorizontalOptions="Center"
                        VerticalOptions="CenterAndExpand">
                        <Label
                            Text="{Binding .}"/>
                    </Frame>
                </StackLayout>
            </DataTemplate>
        </CarouselView.ItemTemplate>
    </CarouselView>

    <IndicatorView
        x:Name="indicatorView"
        HorizontalOptions="Center"
        IndicatorColor="LightGray"
        SelectedIndicatorColor="DarkGray"/>
  </VerticalStackLayout>
</ContentPage>
```

2. Establecemos el **CollectionsControlsDemo** como página de inicio:

```
MainPage = new CollectionsControlsDemo();
```

3. Modificamos el **CollectionsControlsDemo** comentaremos la parte del carrusel (incluyendo el VerticalStackLayout) y adicionamos este código:

```
<ListView
  HasUnevenRows="True">
  <ListView.ItemsSource>
    <x:Array Type="{x:Type x:String}">
        <x:String>mono</x:String>
        <x:String>monodroid</x:String>
        <x:String>monotouch</x:String>
        <x:String>monorail</x:String>
        <x:String>monodevelop</x:String>
        <x:String>monotone</x:String>
        <x:String>monopoly</x:String>
        <x:String>monomodal</x:String>
        <x:String>mononucleosis</x:String>
```

```
        </x:Array>
    </ListView.ItemsSource>
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <StackLayout>
                    <Frame
                        Margin="20"
                        BorderColor="DarkGray"
                        CornerRadius="5"
                        HasShadow="True"
                        HeightRequest="100"
                        HorizontalOptions="Center"
                        VerticalOptions="CenterAndExpand">
                        <Label Text="{Binding .}" />
                    </Frame>
                </StackLayout>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

4. Probamos.

5. Modificamos el **CollectionsControlsDemo** comentaremos la parte del **ListView** y adicionamos este código:

```
<CollectionView SelectionMode="Multiple">
    <CollectionView.ItemsSource>
        <x:Array Type="{x:Type x:String}">
            <x:String>mono</x:String>
            <x:String>monodroid</x:String>
            <x:String>monotouch</x:String>
            <x:String>monorail</x:String>
            <x:String>monodevelop</x:String>
            <x:String>monotone</x:String>
            <x:String>monopoly</x:String>
            <x:String>monomodal</x:String>
            <x:String>mononucleosis</x:String>
        </x:Array>
    </CollectionView.ItemsSource>
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <StackLayout>
                <Frame
                    Margin="20"
                    BorderColor="DarkGray"
                    CornerRadius="5"
                    HasShadow="True"
                    HeightRequest="100"
                    HorizontalOptions="Center"
                    VerticalOptions="CenterAndExpand">
                    <Label Text="{Binding .}" />
                </Frame>
            </StackLayout>
```

```
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

6. Probamos.

7. Modificamos el **CollectionsControlsDemo** comentaremos la parte del **CollectionView** y adicionamos este código:

```
<StackLayout>
    <Picker VerticalOptions="Center">
        <Picker.ItemsSource>
            <x:Array Type="{x:Type x:String}">
                <x:String>mono</x:String>
                <x:String>monodroid</x:String>
                <x:String>monotouch</x:String>
                <x:String>monorail</x:String>
                <x:String>monodevelop</x:String>
                <x:String>monotone</x:String>
                <x:String>monopoly</x:String>
                <x:String>monomodal</x:String>
                <x:String>mononucleosis</x:String>
            </x:Array>
        </Picker.ItemsSource>
    </Picker>
</StackLayout>
```

8. Probamos.

9. Modificamos el **CollectionsControlsDemo** comentaremos la parte del **CollectionView** y adicionamos este código:

```
<TableView Intent="Settings">
    <TableRoot>
        <TableSection Title="First Section">
            <TextCell Detail="TextCell Detail" Text="TextCell" />
            <EntryCell Label="Entry Label" Text="EntryCell Text" />
            <SwitchCell Text="SwitchCell Text" />
            <ImageCell
                Detail="ImageCell Detail"
                ImageSource="dotnet_bot.svg"
                Text="ImageCell Text" />
        </TableSection>
        <TableSection Title="Second Section">
            <TextCell Detail="TextCell Detail" Text="TextCell" />
            <EntryCell Label="Entry Label" Text="EntryCell Text" />
            <SwitchCell Text="SwitchCell Text" />
            <ImageCell
                Detail="ImageCell Detail"
                ImageSource="dotnet_bot.svg"
                Text="ImageCell Text" />
        </TableSection>
    </TableRoot>
```

```
</TableView>
```

10. Probamos y hacemos el **Commit**.

DataBinding

1. En el proyecto **Mobile** vamos a crear una carpeta llamada **BindingDemo** y dentro de esta creamos la clase **Person**:

```
using System;
namespace Sales.Mobile.BindingDemo
{
	public class Person
	{
		public string Name { get; set; }

		public string Phone { get; set; }

		public string Address { get; set; }
	}
}
```

2. En la misma carpeta creamos la página **BindigPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.BindingDemo.BindigPage"
             Title="BindigPage">
    <VerticalStackLayout
        Padding="10"
        Spacing="25"
        VerticalOptions="Center">
        <Label
            x:Name="lblName"
            FontSize="50"
            HorizontalOptions="Center"
            Text="No binding yet"
            VerticalOptions="Center"/>

        <Button
            x:Name="btnOk"
            Text="Bind"
            Clicked="btnOk_Clicked"/>
    </VerticalStackLayout>
</ContentPage>
```

3. Modificamos el **BindigPage.xaml.cs**:

```
namespace Sales.Mobile.BindingDemo;

public partial class BindigPage : ContentPage
{
	public BindigPage()
```

```
        {
                InitializeComponent();
        }

    void btnOk_Clicked(System.Object sender, System.EventArgs e)
    {
                var person = new Person
                {
                        Address = "Calle Luna Calle Sol",
                        Name = "Juan Zuluaga",
                        Phone = "322 311 4620"
                };

                var personBinding = new Binding();
                personBinding.Source = person;
                personBinding.Path = "Name";

                lblName.SetBinding(Label.TextProperty, personBinding);
    }
}
```

4. Cambiamos la página de inicio del proyecto **Mobile**:

```
MainPage = new BindigPage();
```

5. Probamos.

6. Ahora vamos a probar el binding desde el XAML, para eso modificamos el **BindingDemo**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:Models="clr-namespace:Sales.Mobile.BindingDemo"
             x:Class="Sales.Mobile.BindingDemo.BindigPage"
             Title="BindigPage">

    <ContentPage.Resources>
        <Models:Person
            x:Key="person"
            Name="Juan Zuluaga"
            Address="Calle Luna Calle Sol"
            Phone="+57 322 311 4620"/>
    </ContentPage.Resources>

    <VerticalStackLayout
        Padding="10"
        Spacing="25"
        VerticalOptions="Center">
        <Label
            x:Name="lblName"
            FontSize="50"
            HorizontalOptions="Center"
            Text="{Binding Name, Source={StaticResource person}}"
            VerticalOptions="Center"/>
```

```
      <Button
         x:Name="btnOk"
         Text="Bind"
         Clicked="btnOk_Clicked"/>
   </VerticalStackLayout>
</ContentPage>
```

7. Luego modificamos el **BindingDemo.xaml.cs**:

```
void btnOk_Clicked(System.Object sender, System.EventArgs e)
{
        var person = new Person
        {
                Address = "Calle Luna Calle Sol",
                Name = "Juan Zuluaga",
                Phone = "322 311 4620"
        };

        //var personBinding = new Binding();
        //personBinding.Source = person;
        //personBinding.Path = "Name";
        //lblName.SetBinding(Label.TextProperty, personBinding);
}
```

8. Probamos

.

9. Modificamos nuevamente el **BindigPage**:

```
<!--<Label
   x:Name="lblName"
   FontSize="50"
   HorizontalOptions="Center"
   Text="{Binding Name, Source={StaticResource person}}"
   VerticalOptions="Center"/>-->

<Label
   x:Name="lblName"
   FontSize="50"
   HorizontalOptions="Center"
   VerticalOptions="Center"/>
```

10. Y modificamos el **BindigPage.xaml.cs**:

```
void btnOk_Clicked(System.Object sender, System.EventArgs e)
{
        var person = new Person
        {
                Address = "Calle Luna Calle Sol",
                Name = "Juan Zuluaga",
                Phone = "322 311 4620"
        };

        lblName.BindingContext = person;
```

```
        lblName.SetBinding(Label.TextProperty, "Name");

        //var personBinding = new Binding();
        //personBinding.Source = person;
        //personBinding.Path = "Name";
        //lblName.SetBinding(Label.TextProperty, personBinding);
}
```

11. Probamos.

12. Modificamos el **BindigPage.xaml.cs**:

```
void btnOk_Clicked(System.Object sender, System.EventArgs e)
{
        var person = new Person
        {
                Address = "Calle Luna Calle Sol",
                Name = "Juan Zuluaga",
                Phone = "322 311 4620"
        };

        BindingContext = person;

        //lblName.BindingContext = person;
        //lblName.SetBinding(Label.TextProperty, "Name");

        //var personBinding = new Binding();
        //personBinding.Source = person;
        //personBinding.Path = "Name";
        //lblName.SetBinding(Label.TextProperty, personBinding);
}
```

13. Modificamos el **BindigPage.xaml**:

```
<Label
    FontSize="50"
    HorizontalOptions="Center"
    Text="{Binding Name}"
    VerticalOptions="Center"/>

<Label
    FontSize="50"
    HorizontalOptions="Center"
    Text="{Binding Phone}"
    VerticalOptions="Center"/>

<Label
    FontSize="50"
    HorizontalOptions="Center"
    Text="{Binding Address}"
    VerticalOptions="Center"/>
```

14. Probamos.

15. Ahora vamos a probar el binding entre controles, creamos una nueva página llamada **SliderPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.BindingDemo.SliderPage"
             Title="SliderPage">
    <VerticalStackLayout
             VerticalOptions="Center"
             HorizontalOptions="Center">
        <Label
             Text="Welcome to .NET MAUI!"
             Rotation="{Binding Source={x:Reference slider}, Path=Value}"
             FontSize="50"/>
        <Slider
             x:Name="slider"
             Minimum="0"
             Maximum="360"/>
    </VerticalStackLayout>
</ContentPage>
```

16. Establecemos el **SliderPage** como página de inicio:

```
MainPage = new SliderPage();
```

17. Probamos.

18. Creamos la página **BindingModes**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.BindingDemo.BindingModes"
             Title="BindingModes">
    <VerticalStackLayout
        VerticalOptions="Center"
        HorizontalOptions="Center">
        <Entry
             x:Name="source"
             FontSize="30"
             Text="Initial"
             Placeholder="Source..."/>
        <Entry
             x:Name="target"
             FontSize="30"
             Placeholder="Target..."
             Text="{Binding Source={x:Reference source}, Path=Text, Mode=OneTime}"/>
    </VerticalStackLayout>
</ContentPage>
```

19. La establecemos como página de inicio:

```
MainPage = new BindingModes();
```

20. Probamos y jugamos con los valores del **Mode** para ver las diferencias.

21. Coloquemos nuevamente como página de inicio la página **BindigPage**:

```
MainPage = new BindigPage();
```

22. Cambiamos en la **BindigPage** los Label por Entry:

```
<Entry
   FontSize="50"
   HorizontalOptions="Center"
   Text="{Binding Name}"
   VerticalOptions="Center"/>

<Entry
   FontSize="50"
   HorizontalOptions="Center"
   Text="{Binding Phone}"
   VerticalOptions="Center"/>

<Entry
   FontSize="50"
   HorizontalOptions="Center"
   Text="{Binding Address}"
   VerticalOptions="Center"/>
```

23. Probamos.

24. Ahora vamos hacer este cambio en la clase **BindigPage.xaml.cs**:

```
namespace Sales.Mobile.BindingDemo;

public partial class BindigPage : ContentPage
{
        Person _person = new();

        public BindigPage()
        {
                InitializeComponent();

      _person = new Person
      {
         Address = "Calle Luna Calle Sol",
         Name = "Juan Zuluaga",
         Phone = "322 311 4620"
      };

      BindingContext = _person;
   }

   void btnOk_Clicked(System.Object sender, System.EventArgs e)
        {
      _person.Name = "Ledys";
      _person.Phone = "322 300 1232";
```

```
        _person.Address = "Avenida Siempre Viva";
        }
}
```

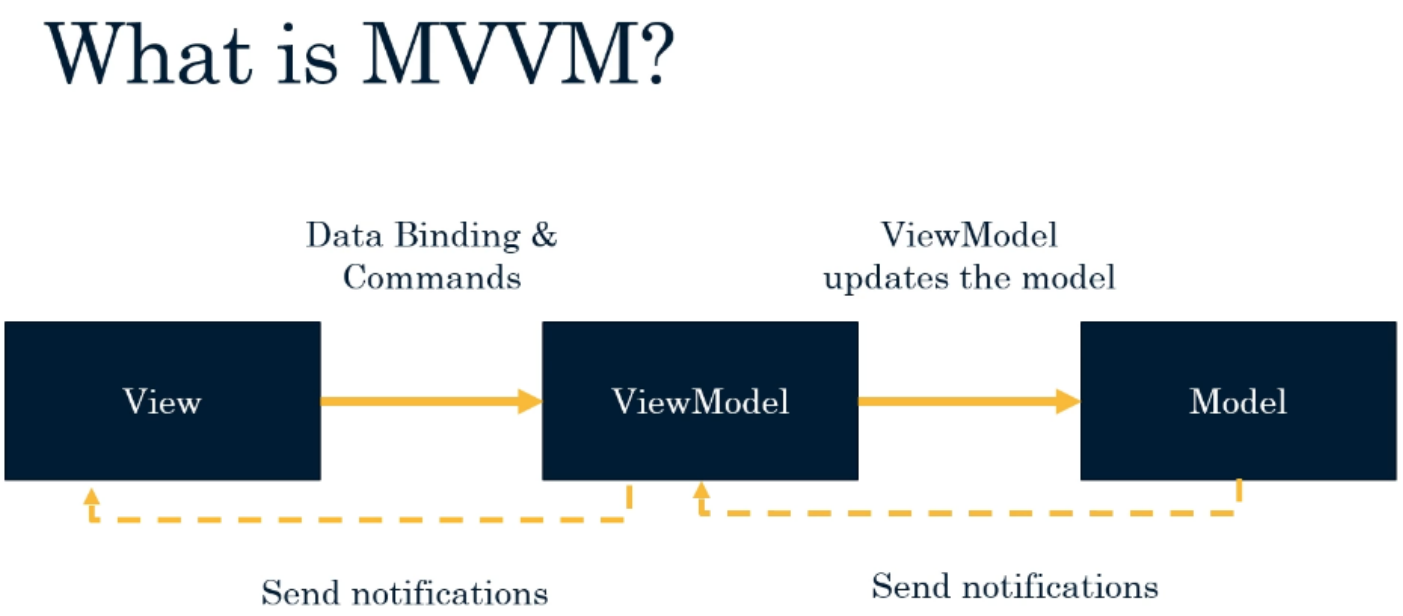
25. Probamos y vemos que las cosas no funcionan como lo esperabamos, para que funcione tenemos que implementar el **INotifyPropertyChanged** en la clase **Person**:

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace Sales.Mobile.BindingDemo
{
   public class Person : INotifyPropertyChanged
   {
       private string _name;
       private string _phone;
       private string _address;

       public string Name
       {
           get => _name;
           set
           {
               _name = value;
               OnPropertyChanged();
           }
       }

       public string Phone
       {
           get => _phone;
           set
           {
               _phone = value;
               OnPropertyChanged();
           }
       }

       public string Address
       {
           get => _address;
           set
           {
               _address = value;
               OnPropertyChanged();
           }
       }

       public event PropertyChangedEventHandler PropertyChanged;

       protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
       {
           PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
```

```
    }
  }
}
```

26. Probamos de nuevo y hacemos el **Commit**.

El patrón MVVM



1. Creamos dentro del proyecto **Mobile** la carpeta **MVVM** y dentro de esta las carpetas **Models**, **Views** y **ViewModels**.

2. Dentro de la carpeta **Models** creamos la clase **Person**:

```
using System;
namespace Sales.Mobile.MVVM.Models
{
	public class Person
	{
		public string Name { get; set; }

		public int Age { get; set; }
	}
}
```

3. Dentro de la carpeta **Views** creamos la vista **PersonView** y modificamos el **PersonView.xaml.cs**:

```
using Sales.Mobile.MVVM.Models;

namespace Sales.Mobile.MVVM.Views;

public partial class PersonView : ContentPage
{
	public PersonView()
	{
		InitializeComponent();
```

```
                var person = new Person
                {
                        Name = "Juan",
                        Age = 48
                };
                BindingContext = person;
        }
}
```

4. Modificamos la vista **PersonView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.PersonView"
             Title="PersonView">
    <VerticalStackLayout
        VerticalOptions="Center"
        HorizontalOptions="Center">
        <Label
            Text="{Binding Name}"
            FontSize="30"/>
    </VerticalStackLayout>
</ContentPage>
```

5. Luego colcamos nuestra vista **PersonView** como página inicial:

```
MainPage = new PersonView();
```

6. Probamos.

7. Pero no debemos permitir que la vista acceda directamente el modelo, por eso creamos en la carpeta de los **ViewModels** nuestro **PesonViewModel**:

8. Probamos.

```
using System;
using Sales.Mobile.MVVM.Models;

namespace Sales.Mobile.MVVM.ViewModels
{
        public class PersonViewModel
        {
                public PersonViewModel()
                {
        Person = new Person
        {
            Name = "Juan",
            Age = 48
        };
    }

    public Person Person { get; set; }
  }
```

```
}
```

9. Modificamos el **PersonView.xaml.cs**:

```
using Sales.Mobile.MVVM.ViewModels;

namespace Sales.Mobile.MVVM.Views;

public partial class PersonView : ContentPage
{
	public PersonView()
	{
		InitializeComponent();
		BindingContext = new PersonViewModel();
	}
}
```

10. Modificamos el **PersonView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
	xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
	x:Class="Sales.Mobile.MVVM.Views.PersonView"
	Title="PersonView">
   <VerticalStackLayout
      VerticalOptions="Center"
      HorizontalOptions="Center">
      <Label
         Text="{Binding Person.Name}"
         FontSize="30"/>
   </VerticalStackLayout>
</ContentPage>
```

11. Probamos.

12. Cambiamos el modelo **Person**:

```
using System;
namespace Sales.Mobile.MVVM.Models
{
	public class Person
	{
		public string Name { get; set; }

		public int Age { get; set; }

		public bool Married { get; set; }

		public DateTime BirthDate { get; set; }

		public int Weight { get; set; }

		public TimeSpan LunchTime { get; set; }
```

```
        }
}
```

13. Cambiamos el **PersonViewModel**:

```
public class PersonViewModel
{
        public PersonViewModel()
        {
      Person = new Person
      {
        Name = "Juan",
        Age = 48,
        Married = true,
        BirthDate = new DateTime(1974, 9, 23),
        Weight = 89,
        LunchTime = new TimeSpan(10, 0, 0)
      };
   }

   public Person Person { get; set; }
}
```

14. Cambiamos el **PersonView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.PersonView"
             Title="PersonView">
   <VerticalStackLayout
      VerticalOptions="Center"
      HorizontalOptions="Center">
      <Label
         Text="{Binding Person.Name}"
         FontSize="30"/>
      <Slider
         Maximum="100"
         Minimum="0"
         Value="{Binding Person.Age}"/>
      <Switch
         IsToggled="{Binding Person.Married}"/>
      <DatePicker
         Date="{Binding Person.BirthDate}"/>
      <Entry
         Text="{Binding Person.Weight}"/>
      <TimePicker
         Time="{Binding Person.LunchTime}"/>
   </VerticalStackLayout>
</ContentPage>
```

15. Probamos.

16. Dentro de **MVVM/Views** creamos una nueva página de contenido llamada **PeopleView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.PeopleView"
             Title="PeopleView">
    <CollectionView SelectionMode="Multiple">
        <CollectionView.ItemsSource>
            <x:Array Type="{x:Type x:String}">
                <x:String>mono</x:String>
                <x:String>monodroid</x:String>
                <x:String>monotouch</x:String>
                <x:String>monorail</x:String>
                <x:String>monodevelop</x:String>
                <x:String>monotone</x:String>
                <x:String>monopoly</x:String>
                <x:String>monomodal</x:String>
                <x:String>mononucleosis</x:String>
            </x:Array>
        </CollectionView.ItemsSource>
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                    <Frame
                        Margin="20"
                        BorderColor="DarkGray"
                        CornerRadius="5"
                        HasShadow="True"
                        HeightRequest="100"
                        HorizontalOptions="Center"
                        VerticalOptions="CenterAndExpand">
                        <Label Text="{Binding .}" />
                    </Frame>
                </StackLayout>
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
</ContentPage>
```

17. Colocamos esta nueva página como página de inicio:

```
MainPage = new PeopleView();
```

18. Probamos

19. Pero ahora vamos a meterle una buena arquitectura a esto. Creamos la **PeopleViewModel**:

```
using System;
namespace Sales.Mobile.MVVM.ViewModels
{
    public class PeopleViewModel
    {
        public PeopleViewModel()
        {
```

```
                People = new List<string>()
                {
                    "Juan",
                    "Ledys",
                    "Valery",
                    "Ronal",
                    "Geralin",
                    "Benedict",
                    "Isis",
                    "Gaia",
                    "Toño"
                };
            }

            public List<string> People { get; set; }
        }
}
```

20. Modificamos el **PeopleView.xaml.cs**:

```
namespace Sales.Mobile.MVVM.Views;
using Sales.Mobile.MVVM.ViewModels;

public partial class PeopleView : ContentPage
{
        public PeopleView()
        {
                InitializeComponent();
                BindingContext = new PeopleViewModel();
        }
}
```

21. Modificamos el **PeopleView.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.PeopleView"
             Title="PeopleView">
    <CollectionView
        ItemsSource="{Binding People}"
        SelectionMode="Multiple">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                    <Frame
                        Margin="20"
                        BorderColor="DarkGray"
                        CornerRadius="5"
                        HasShadow="True"
                        HeightRequest="100"
                        HorizontalOptions="Center"
                        VerticalOptions="CenterAndExpand">
                        <Label Text="{Binding .}" />
```

```
            </Frame>
          </StackLayout>
        </DataTemplate>
      </CollectionView.ItemTemplate>
   </CollectionView>
</ContentPage>
```

22. Probamos.

23. Modificamos la **PeopleViewModel**:

```
using System;
using Sales.Mobile.MVVM.Models;

namespace Sales.Mobile.MVVM.ViewModels
{
        public class PeopleViewModel
        {
                public PeopleViewModel()
                {
                        People = new List<Person>()
                        {
            new Person { Name = "Juan", Age = 48, BirthDate = new DateTime(1974, 9, 23), LunchTime = new
TimeSpan(12, 0,0), Married = true, Weight = 89 },
            new Person { Name = "Ledys", Age = 42, BirthDate = new DateTime(1981, 1, 11), LunchTime = new
TimeSpan(13, 0,0), Married = true, Weight = 56 },
            new Person { Name = "Valery", Age = 12, BirthDate = new DateTime(2010, 2, 27), LunchTime = new
TimeSpan(12, 30,0), Married = false, Weight = 38 },
            new Person { Name = "Ronal", Age = 23, BirthDate = new DateTime(2000, 1, 20), LunchTime = new
TimeSpan(14, 0,0), Married = false, Weight = 47 },
                        };
                }

                public List<Person> People { get; set; }
        }
}
```

24. Modificamos la **PeopleView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.PeopleView"
             Title="PeopleView">
   <CollectionView
      ItemsSource="{Binding People}"
      SelectionMode="Multiple">
      <CollectionView.ItemTemplate>
         <DataTemplate>
            <StackLayout>
               <Frame
                  Margin="20"
                  BorderColor="DarkGray"
                  CornerRadius="5"
```

```
                HasShadow="True"
                HeightRequest="100"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand">
                <VerticalStackLayout>
                    <Label
                        Text="{Binding Name}"
                        FontAttributes="Bold"
                        FontSize="Large" />
                    <Label
                        Text="{Binding BirthDate, StringFormat='{0:yyy/MM/dd}'}" />
                    <Label
                        Text="{Binding Married, StringFormat='Casado: {0}'}" />
                </VerticalStackLayout>
            </Frame>
        </StackLayout>
    </DataTemplate>
  </CollectionView.ItemTemplate>
</CollectionView>
</ContentPage>
```

25. Probamos.

26. Dentro de **MVVM** creamos la carpeta **Converters** y dentro de esta la clase **BoolConverter**:

```
using System;
using System.Globalization;

namespace Sales.Mobile.MVVM.Converters
{
    public class BoolConverter : IValueConverter
    {
        public BoolConverter()
        {
        }

    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var boolValue = (bool)value;
        if (boolValue)
        {
            return "Sí";
        }
        return "No";
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var stringValue = value.ToString();
        if (stringValue == "Sí")
        {
            return true;
        }
        return false;
```

```
        }
    }
}
```

27. Modificamos la **PeopleView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:converters="clr-namespace:Sales.Mobile.MVVM.Converters"
             x:Class="Sales.Mobile.MVVM.Views.PeopleView"
             Title="PeopleView">
    <ContentPage.Resources>
        <converters:BoolConverter x:Key="boolConverter"/>
    </ContentPage.Resources>

    <CollectionView
        ItemsSource="{Binding People}"
        SelectionMode="Multiple">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                    <Frame
                        Margin="20"
                        BorderColor="DarkGray"
                        CornerRadius="5"
                        HasShadow="True"
                        HeightRequest="110"
                        HorizontalOptions="Center"
                        VerticalOptions="CenterAndExpand">
                        <VerticalStackLayout>
                            <Label
                                Text="{Binding Name}"
                                FontAttributes="Bold"
                                FontSize="Large" />
                            <Label
                                Text="{Binding BirthDate, StringFormat='{0:yyy/MM/dd}'}" />
                            <Label
                                Text="{Binding Married, Converter={StaticResource boolConverter}, StringFormat='Casado: {0}'}" />
                        </VerticalStackLayout>
                    </Frame>
                </StackLayout>
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
</ContentPage>
```

28. Probamos y hacemos el **Commit**.

El uso de comandos

1. Creamos el **CommandsViewModel**:

```
using System;
```

```
using System.Windows.Input;

namespace Sales.Mobile.MVVM.ViewModels
{
	public class CommandsViewModel
	{
		public CommandsViewModel()
		{
		}

		public ICommand ClickCommand => new Command(() => App.Current.MainPage.DisplayAlert("Title",
"Message", "Ok"));
	}
}
```

2. Creamos el **CommandsView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
	xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
	x:Class="Sales.Mobile.MVVM.Views.CommandsView"
	Title="CommandsView">
   <VerticalStackLayout
      VerticalOptions="Center"
      HorizontalOptions="Center" >
      <Button
	Text="Click Me!"
	Command="{Binding ClickCommand}"/>
   </VerticalStackLayout>
</ContentPage>
```

3. Modificamos el **CommandsView.xaml.cs**:

```
namespace Sales.Mobile.MVVM.Views;
using Sales.Mobile.MVVM.ViewModels;

public partial class CommandsView : ContentPage
{
	public CommandsView()
	{
		InitializeComponent();
		BindingContext = new CommandsViewModel();
   }
}
```

4. Colocamos esta nueva página como la inicial:

```
MainPage = new CommandsView();
```

5. Modificamos el **CommandsView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Sales.Mobile.MVVM.Views.CommandsView"
        Title="CommandsView">
  <VerticalStackLayout
    VerticalOptions="Center"
    HorizontalOptions="Center" >
    <Button
      Text="Click Me!"
      Command="{Binding ClickCommand}"/>
    <SearchBar
      Text="{Binding SearchTerm}"
      SearchCommand="{Binding SearchCommand}"/>
  </VerticalStackLayout>
</ContentPage>
```

6. Modificamos el **CommandsViewModel**:

```
using System;
using System.Windows.Input;

namespace Sales.Mobile.MVVM.ViewModels
{
        public class CommandsViewModel
        {
                public CommandsViewModel()
                {
                }

                public string SearchTerm { get; set; }

                public ICommand ClickCommand => new Command(() => App.Current.MainPage.DisplayAlert("Title",
"Message", "Ok"));

                public ICommand SearchCommand => new Command(() =>
App.Current.MainPage.DisplayAlert("Busqueda", $"Buscaste: {SearchTerm}", "Ok"));
  }
}
```

7. Probamos y hacemos el **Commit**.

Implementando el INotifyPropertyChanged automáticamente

1. Creamos el **DemoAutoPropertyChangedViewModel**:

```
using System;
using System.Windows.Input;

namespace Sales.Mobile.MVVM.ViewModels
{
        public class DemoAutoPropertyChangedViewModel
        {
    public int Number1 { get; set; }

    public int Number2 { get; set; }
```

```
    public int Result { get; set; }

    public ICommand AddCommand => new Command(() => Result = Number1 + Number2);
  }
}
```

2. Creamos el **DemoAutoPropertyChangedView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.DemoAutoPropertyChangedView"
             Title="DemoAutoPropertyChangedView">
  <VerticalStackLayout
        VerticalOptions="Center"
        HorizontalOptions="Center">
    <Entry
        Placeholder="Number 1..."
        Keyboard="Numeric"
        Text="{Binding Number1}"/>
    <Entry
        Placeholder="Number 1..."
        Keyboard="Numeric"
        Text="{Binding Number2}"/>
    <Entry
        IsEnabled="False"
        Placeholder="Result..."
        Text="{Binding Result}"/>
    <Button
        Command="{Binding AddCommand}"
        Text="Calculate"/>
  </VerticalStackLayout>
</ContentPage>
```

3. Modificamos el **DemoAutoPropertyChangedView.xaml.cs**:

```
namespace Sales.Mobile.MVVM.Views;
using Sales.Mobile.MVVM.ViewModels;

public partial class DemoAutoPropertyChangedView : ContentPage
{
        public DemoAutoPropertyChangedView()
        {
                InitializeComponent();
                BindingContext = new DemoAutoPropertyChangedViewModel();
  }
}
```

4. Cambiamos la página de inicio:

```
MainPage = new DemoAutoPropertyChangedView();
```

5. Probamos y nos damos cuenta que no es el resultado esperado.

6. Podemos hacer la implementación clásica del **INotifyPropertyChanged** pero esto nos incrementaría consiferablemente el número de líneas en la **ViewModel**, hay otra mejor forma de implementar esto, pero primero vamos a proceder instalando el paquete **PropertyChanged.Fody**.

7. Luego modificamos el **DemoAutoPropertyChangedViewModel**:

```
using System;
using System.Windows.Input;
using PropertyChanged;

namespace Sales.Mobile.MVVM.ViewModels
{
   [AddINotifyPropertyChangedInterface]
          public class DemoAutoPropertyChangedViewModel
          {
       public int Number1 { get; set; }

       public int Number2 { get; set; }

       public int Result { get; set; }

       public ICommand AddCommand => new Command(() => Result = Number1 + Number2);
   }
}
```

8. Probamos y hacemos el **Commit**.

ACA VAMOS

Estilos en .NET MAUI

1. Dentro de **MVVM** en la carpeta **Views** creamos el **StyleDemoView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.StyleDemoView"
             Title="StyleDemoView">
    <VerticalStackLayout
        VerticalOptions="Center"
        HorizontalOptions="Center">
        <Button
            BackgroundColor="#323031"
            FontAttributes="Bold"
            FontSize="Large"
            Text="Login"
            TextColor="#ffc857"/>
        <Button
            BackgroundColor="#323031"
            FontAttributes="Bold"
            FontSize="Large"
            Text="Visit WebSite"
            TextColor="#ffc857"/>
```

```
    </VerticalStackLayout>
</ContentPage>
```

2. Ponemos esta página como página de inicio:

```
MainPage = new NavigationPage(new StyleDemoView());
```

3. Probamos.

4. Modificamos el **App.xmal** y quitamos el diccionario de recursos:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Sales.Mobile"
             x:Class="Sales.Mobile.App">
    <Application.Resources>
    </Application.Resources>
</Application>
```

5. Probamos.

6. Vamos cortar los colores y estilos que pusimos en el **StyleDemoView.xaml** y los vamos a pasar al **App.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Sales.Mobile.MVVM.Views.StyleDemoView"
             Title="StyleDemoView">

    <ContentPage.Resources>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Button
            Text="Login"
            Style="{StaticResource primaryButton}"/>
        <Button
            Text="Visit WebSite"
            Style="{StaticResource secondaryButton}"/>
    </VerticalStackLayout>
</ContentPage>
```

Y los pasamos al **App.xaml**:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Sales.Mobile"
             x:Class="Sales.Mobile.App">
    <Application.Resources>
        <Color x:Key="bgColor">#323031</Color>
        <Color x:Key="bg2Color">#45f03a</Color>
        <Color x:Key="textColor">#ffc857</Color>
```

```
        <Style TargetType="VerticalStackLayout">
            <Setter Property="VerticalOptions" Value="Center"/>
            <Setter Property="Spacing" Value="5"/>
            <Setter Property="Padding" Value="10"/>
        </Style>
        <Style TargetType="Button" x:Key="primaryButton">
            <Setter Property="BackgroundColor" Value="{StaticResource bgColor}"/>
            <Setter Property="FontAttributes" Value="Bold"/>
            <Setter Property="FontSize" Value="Large"/>
            <Setter Property="TextColor" Value="{StaticResource textColor}"/>
        </Style>
        <Style TargetType="Button" x:Key="secondaryButton">
            <Setter Property="BackgroundColor" Value="{StaticResource bg2Color}"/>
            <Setter Property="FontAttributes" Value="Bold"/>
            <Setter Property="FontSize" Value="Large"/>
            <Setter Property="TextColor" Value="Black"/>
        </Style>
    </Application.Resources>
</Application>
```

7. Probamos.

8. Podemos también heredar de estilos para evitar la duplicidad de código:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Sales.Mobile"
             x:Class="Sales.Mobile.App">
    <Application.Resources>
        <Color x:Key="bgColor">#323031</Color>
        <Color x:Key="bg2Color">#45f03a</Color>
        <Color x:Key="textColor">#ffc857</Color>
        <Style TargetType="VerticalStackLayout">
            <Setter Property="VerticalOptions" Value="Center"/>
            <Setter Property="Spacing" Value="5"/>
            <Setter Property="Padding" Value="10"/>
        </Style>
        <Style TargetType="Button" x:Key="baseButton">
            <Setter Property="BackgroundColor" Value="{StaticResource bgColor}"/>
            <Setter Property="FontAttributes" Value="Bold"/>
            <Setter Property="FontSize" Value="Large"/>
            <Setter Property="TextColor" Value="{StaticResource textColor}"/>
        </Style>
        <Style
            TargetType="Button"
            x:Key="primaryButton"
            BasedOn="{StaticResource baseButton}">
            <Setter Property="BackgroundColor" Value="{StaticResource bgColor}"/>
        </Style>
        <Style
            TargetType="Button"
            x:Key="secondaryButton"
```

```
        BasedOn="{StaticResource baseButton}">
        <Setter Property="BackgroundColor" Value="{StaticResource bg2Color}"/>
        <Setter Property="TextColor" Value="Black"/>
    </Style>
  </Application.Resources>
</Application>
```

9. Probamos.

10. Vamos a mover nuestros estilos personalizados a **Styles.xaml**:

```
  <Style TargetType="VerticalStackLayout">
    <Setter Property="VerticalOptions" Value="Center"/>
    <Setter Property="Spacing" Value="5"/>
    <Setter Property="Padding" Value="10"/>
  </Style>

  <Style TargetType="Button" x:Key="baseButton">
    <Setter Property="BackgroundColor" Value="{StaticResource bgColor}"/>
    <Setter Property="FontAttributes" Value="Bold"/>
    <Setter Property="FontSize" Value="Large"/>
    <Setter Property="TextColor" Value="{StaticResource textColor}"/>
  </Style>

  <Style
    TargetType="Button"
    x:Key="primaryButton"
    BasedOn="{StaticResource baseButton}">
    <Setter Property="BackgroundColor" Value="{StaticResource bgColor}"/>
  </Style>

  <Style
    TargetType="Button"
    x:Key="secondaryButton"
    BasedOn="{StaticResource baseButton}">
    <Setter Property="BackgroundColor" Value="{StaticResource bg2Color}"/>
    <Setter Property="TextColor" Value="Black"/>
  </Style>

</ResourceDictionary>
```

11. Vamos a mover nuestros colores personalizados a **Colors.xaml**:

```
  <Color x:Key="bgColor">#323031</Color>
  <Color x:Key="bg2Color">#45f03a</Color>
  <Color x:Key="textColor">#ffc857</Color>

</ResourceDictionary>
```

12. Dejamos nuestro **App.xmal.cs** como lo teniamos originalmente:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
        xmlns:local="clr-namespace:Sales.Mobile"
        x:Class="Sales.Mobile.App">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
                <ResourceDictionary Source="Resources/Styles/Styles.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

13. Probamos.

14. Para no perdernos entre tantos colores podemos usar la página: https://color.adobe.com/es/create/color-wheel

CollectionView

CollectionView vs ListView

CollectionView

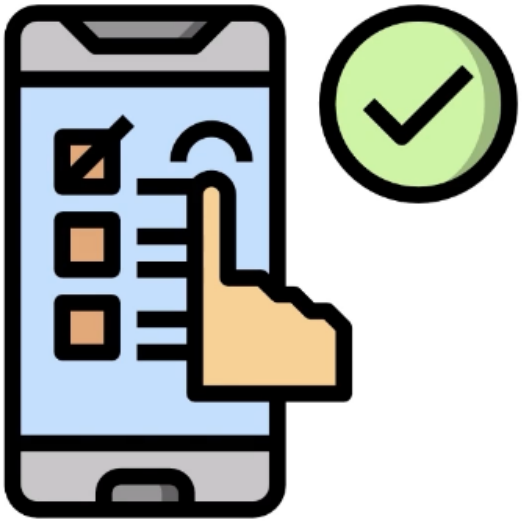
ListView

CollectionView vs ListView

CollectionView vs ListView

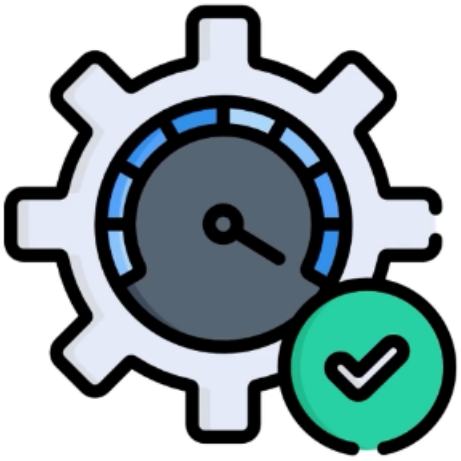
CollectionView

CollectionView vs ListView



Individual and Multiple Selection

CollectionView vs ListView



Flexible and performant alternative to ListView

CollectionView vs ListView

```
<CollectionView>
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                ...
            </ViewCell>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

No concept of Cells

1. Vamos a crear un nuevo proyecto llamado **CollectionViewDemo**.

2. Creamos la carpeta **MVVM** y dentro de esta creamos la carpeta **Views** y dentro de escra creamos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">
    <VerticalStackLayout>
        <Label
```

```
        Text="Welcome to .NET MAUI!"
        VerticalOptions="Center"
        HorizontalOptions="Center" />
    </VerticalStackLayout>
</ContentPage>
```

3. Dentro la carpeta **MVVM** y dentro de esta creamos la carpeta **ViewModels** y dentro de escra creamos el **DataViewModel**:

```
using System;
namespace CollectionViewDemo.MVVM.ViewModels
{
        public class DataViewModel
        {
        }
}
```

4. Modificamos el **DataView.xaml.cs**:

```
using CollectionViewDemo.MVVM.ViewModels;

namespace CollectionViewDemo.MVVM.Views;

public partial class DataView : ContentPage
{
        public DataView()
        {
                InitializeComponent();
                BindingContext = new DataViewModel();
        }
}
```

5. Colocamos la nueva página como la inicial:

```
MainPage = new DataView();
```

6. Probamos.

7. Dentro de **MVVM** creamos la carpeta **Models** y dentro de esta la clase **Product**:

```
using System;
namespace CollectionViewDemo.MVVM.Models
{
        public class Product
        {
                public string Name { get; set; }

                public decimal Price { get; set; }

                public string Image { get; set; }

                public int Stock { get; set; }

                public bool HasOffer { get; set; }
```

```
        public decimal OfferPrice { get; set; }
    }
}
```

8. Modificamos el **DataViewModel**:

```
using System;
using System.Collections.ObjectModel;
using CollectionViewDemo.MVVM.Models;

namespace CollectionViewDemo.MVVM.ViewModels
{
    public class DataViewModel
    {
        public DataViewModel()
        {
            Products = new()
            {
                new Product
                {
                    Name = "Yogurt",
                    Price = 60.0m,
                    Image = "yogurt.png",
                    HasOffer = false,
                    Stock = 28
                },
                new Product
                {
                    Name = "Watermelon",
                    Price = 30.0m,
                    Image = "watermelon.png",
                    HasOffer = false,
                    Stock = 87
                },
                new Product
                {
                    Name = "Water Bottle",
                    Price = 80.0m,
                    Image = "water_bottle.png",
                    HasOffer = true,
                    OfferPrice = 69.99m,
                    Stock = 33
                },
                new Product
                {
                    Name = "Tomato",
                    Price = 120.0m,
                    Image = "tomato.png",
                    HasOffer = false,
                    Stock = 0
                },
                new Product
                {
```

```
            Name = "Tea",
            Price = 65.0m,
            Image = "tea_bag.png",
            HasOffer = false,
            Stock = 82
        },
        new Product
        {
            Name = "Sparkling Drink",
            Price = 35.0m,
            Image = "sparkling_drink.png",
            HasOffer = false,
            Stock = 728
        },
        new Product
        {
            Name = "Spaguetti",
            Price = 15.0m,
            Image = "spaguetti.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Cream",
            Price = 48.0m,
            Image = "cream.png",
            HasOffer = false,
            Stock = 22
        },
        new Product
        {
            Name = "Snack",
            Price = 25.0m,
            Image = "009_snack.png",
            HasOffer = false,
            Stock = 2
        },
        new Product
        {
            Name = "Shrimp",
            Price = 300.0m,
            Image = "shrimp.png",
            HasOffer = true,
            OfferPrice = 250.0m,
            Stock = 58
        },
        new Product
        {
            Name = "Seasoning",
            Price = 185.0m,
            Image = "seasoning.png",
            HasOffer = false,
            Stock = 99
```

```
            },
            new Product
            {
                Name = "Sauce",
                Price = 220.0m,
                Image = "sauce.png",
                HasOffer = false,
                Stock = 72
            },
            new Product
            {
                Name = "Rice",
                Price = 48.0m,
                Image = "rice.png",
                HasOffer = false,
                Stock = 143
            },
            new Product
            {
                Name = "Peas",
                Price = 114.0m,
                Image = "peas.png",
                HasOffer = false,
                Stock = 0
            },
            new Product
            {
                Name = "Ham",
                Price = 215.0m,
                Image = "ham_1.png",
                HasOffer = true,
                OfferPrice = 189.0m,
                Stock = 732
            },
            new Product
            {
                Name = "Chicken Leg",
                Price = 142.0m,
                Image = "chicken_leg.png",
                HasOffer = true,
                OfferPrice = 125.0m,
                Stock = 20
            },
            new Product
            {
                Name = "Pizza",
                Price = 321.0m,
                Image = "pizza.png",
                HasOffer = false,
                Stock = 559
            },
            new Product
            {
                Name = "Pineapple",
```

```
            Price = 49.0m,
            Image = "pineapple.png",
            HasOffer = false,
            Stock = 41
        },
        new Product
        {
            Name = "Pepper",
            Price = 60.0m,
            Image = "pepper.png",
            HasOffer = true,
            OfferPrice = 30.0m,
            Stock = 64
        },
        new Product
        {
            Name = "Pasta",
            Price = 52.0m,
            Image = "pasta.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Oil Bottle",
            Price = 152.0m,
            Image = "oil_bottle",
            HasOffer = false,
            Stock = 87
        },
        new Product
        {
            Name = "Mushroom",
            Price = 28.0m,
            Image = "mushroom.png",
            HasOffer = false,
            Stock = 17
        },
        new Product
        {
            Name = "Milk Bottle",
            Price = 85.0m,
            Image = "milk_bottle.png",
            HasOffer = false,
            Stock = 39
        },
        new Product
        {
            Name = "Meat",
            Price = 450.0m,
            Image = "meat.png",
            HasOffer = false,
            Stock = 28
        },
```

```
            new Product
            {
                Name = "Lemon",
                Price = 20.0m,
                Image = "lemon.png",
                HasOffer = false,
                Stock = 87
            },
            new Product
            {
                Name = "Tomato Sauce",
                Price = 15.0m,
                Image = "tomato_sauce.png",
                HasOffer = false,
                Stock = 26
            },
            new Product
            {
                Name = "Juice",
                Price = 60.0m,
                Image = "juice.png",
                HasOffer = false,
                Stock = 31
            },
            new Product
            {
                Name = "Ice Cream",
                Price = 251.0m,
                Image = "ice_cream.png",
                HasOffer = true,
                OfferPrice = 200.0m,
                Stock = 88
            },
            new Product
            {
                Name = "Ham",
                Price = 290.0m,
                Image = "ham.png",
                HasOffer = false,
                Stock = 0
            },
            new Product
            {
                Name = "Ice",
                Price = 125.0m,
                Image = "ice.png",
                HasOffer = false,
                Stock = 22
            },
            new Product
            {
                Name = "Flour",
                Price = 86.0m,
                Image = "flour.png",
```

```
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Fish",
            Price = 440.0m,
            Image = "fish_1.png",
            HasOffer = false,
            Stock = 80
        },
        new Product
        {
            Name = "Fish 2",
            Price = 425.0m,
            Image = "fish.png",
            HasOffer = false,
            Stock = 24
        },
        new Product
        {
            Name = "Eggs",
            Price = 150.0m,
            Image = "eggs.png",
            HasOffer = false,
            Stock = 47
        },
        new Product
        {
            Name = "Cucumber",
            Price = 35.0m,
            Image = "cucumber.png",
            HasOffer = false,
            Stock = 74
        },
        new Product
        {
            Name = "Croissant",
            Price = 68.0m,
            Image = "croissant.png",
            HasOffer = true,
            OfferPrice = 50.0m,
            Stock = 27
        },
        new Product
        {
            Name = "Cookies",
            Price = 95.0m,
            Image = "cookie.png",
            HasOffer = false,
            Stock = 56
        },
        new Product
        {
```

```
            Name = "Coffee",
            Price = 154.0m,
            Image = "toffee.png",
            HasOffer = false,
            Stock = 83
        },
        new Product
        {
            Name = "Chocolate Bar",
            Price = 32.0m,
            Image = "chocolate_bar.png",
            HasOffer = false,
            Stock = 21
        },
        new Product
        {
            Name = "Cheese",
            Price = 36.0m,
            Image = "cheese.png",
            HasOffer = true,
            OfferPrice = 25.0m,
            Stock =73
        },
        new Product
        {
            Name = "Carrot",
            Price = 15.0m,
            Image = "carrot.png",
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Canned Food",
            Price = 89.0m,
            Image = "canned_food.png",
            HasOffer = false,
            Stock = 773
        },
        new Product
        {
            Name = "Soda",
            Price = 45.0m,
            Image = "can.png",
            HasOffer = false,
            Stock = 843
        },
        new Product
        {
            Name = "Candies",
            Price = 55.0m,
            Image = "candy.png",
            HasOffer = false,
            Stock = 71
```

```
        },
        new Product
        {
            Name = "Cake",
            Price = 250.0m,
            Image = "cake.png",
            HasOffer = true,
            OfferPrice = 200.0m,
            Stock = 0
        },
        new Product
        {
            Name = "Bread",
            Price = 100.0m,
            Image = "bread_1.png",
            HasOffer = false,
            Stock =134
        },
        new Product
        {
            Name = "Bread",
            Price = 85.0m,
            Image = "bread.png",
            HasOffer = false,
            Stock = 8
        },
        new Product
        {
            Name = "Banana",
            Price = 15.0m,
            Image = "banana.png",
            HasOffer = true,
            OfferPrice = 10.0m,
            Stock = 72
        },
        new Product
        {
            Name = "Apple",
            Price = 40.0m,
            Image = "apple.png",
            HasOffer = false,
            Stock = 737
        },
        new Product
        {
            Name = "Alcohol",
            Price = 370.0m,
            Image = "alcohol.png",
            HasOffer = false,
            Stock = 9
        },
    };
            }
```

```
        public ObservableCollection<Product> Products { get; set; }

    }
}
```

9. Cambiemos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">
    <CollectionView ItemsSource="{Binding Products}"/>
</ContentPage>
```

10. Probamos.

11. Modificamos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">
    <CollectionView ItemsSource="{Binding Products}">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <Label Text="{Binding Name}"/>
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
</ContentPage>
```

12. Probamos.

13. Modificamos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">
    <CollectionView ItemsSource="{Binding Products}">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <Grid
                    Margin="15,10,15,0"
                    ColumnDefinitions=".1*,.2*,.7*"
                    RowDefinitions="*,*">
                    <Frame
                        Grid.RowSpan="2"
                        Grid.Column="1"
                        Grid.ColumnSpan="2"
                        BorderColor="White">
```

```
                <Frame.Background>
                    <LinearGradientBrush EndPoint="1,0">
                        <GradientStop Offset="0" Color="#f8f9fa"/>
                        <GradientStop Offset="1" Color="#dee2e6"/>
                    </LinearGradientBrush>
                </Frame.Background>
            </Frame>
            <Image
                Grid.RowSpan="2"
                Grid.ColumnSpan="2"
                HeightRequest="100"
                Source="{Binding Image}"/>
        </Grid>
    </DataTemplate>
  </CollectionView.ItemTemplate>
</CollectionView>
</ContentPage>
```

14. Modificamos el **DataView**:

```
<Label
    Grid.Column="2"
    FontSize="Large"
    Text="{Binding Name}"
    VerticalOptions="Center"/>
<Label
    Grid.Column="2"
    Grid.Row="1"
    FontSize="Large"
    Text="{Binding Price, StringFormat='{0:C}'}"
    VerticalOptions="Center"/>
```

15. Probamos.

16. Creamos el dicionario **CollectionViewDemo** en **Resources/Styles** y borramos el **CollectionViewDemo.xaml.cs**:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xaml-comp compile="true" ?>
<ResourceDictionary xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">
    <DataTemplate x:Key="ProductStyle">
        <Grid
            Margin="15,10,15,0"
            ColumnDefinitions=".1*,.2*,.7*"
            RowDefinitions="*,*">
            <Frame
                Grid.RowSpan="2"
                Grid.Column="1"
                Grid.ColumnSpan="2"
                BorderColor="White">
                <Frame.Background>
                    <LinearGradientBrush EndPoint="1,0">
                        <GradientStop Offset="0" Color="#f8f9fa"/>
```

```
                    <GradientStop Offset="1" Color="#dee2e6"/>
                </LinearGradientBrush>
            </Frame.Background>
        </Frame>
        <Image
            Grid.RowSpan="2"
            Grid.ColumnSpan="2"
            HeightRequest="100"
            Source="{Binding Image}"/>
        <Label
            Grid.Column="2"
            FontSize="Large"
            Text="{Binding Name}"
            VerticalOptions="Center"/>
        <Label
            Grid.Column="2"
            Grid.Row="1"
            FontSize="Large"
            Text="{Binding Price, StringFormat='{0:C}'}"
            VerticalOptions="Center"/>
    </Grid>
  </DataTemplate>
</ResourceDictionary>
```

17. Modificamos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">
  <CollectionView
    ItemsSource="{Binding Products}"
    ItemTemplate="{StaticResource ProductStyle}">
  </CollectionView>
</ContentPage>
```

18. Cambiamos la definición de **App.xmal**:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:CollectionViewDemo"
             x:Class="CollectionViewDemo.App">
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
        <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
        <ResourceDictionary Source="Resources/Styles/CollectionViewDictionary.xaml" />
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

19. Probamos.

20. Ahora vamos a mostrar un diseño diferente para los productos que se encuentran en oferta. Para eso vamos utilizar un **Data Template Selector**.

21. Creamos la carpeta **Selectors** y dentro de esta la clase **ProductDataTemplateSelector**:

```
using System;
using CollectionViewDemo.MVVM.Models;
namespace CollectionViewDemo.Selectors
{
        public class ProductDataTemplateSelector : DataTemplateSelector
        {
      protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
      {
        var product = item as Product;
        if (!product.HasOffer)
        {
            Application.Current.Resources.TryGetValue("ProductStyle", out var productStyle);
            return productStyle as DataTemplate;
        }
        return new DataTemplate();
      }
   }
}
```

22. Modificamos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:DataTemplates="clr-namespace:CollectionViewDemo.Selectors"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">
   <ContentPage.Resources>
      <DataTemplates:ProductDataTemplateSelector x:Key="ProductTemplates"/>
   </ContentPage.Resources>
   <CollectionView
      ItemsSource="{Binding Products}"
      ItemTemplate="{StaticResource ProductTemplates}">
   </CollectionView>
</ContentPage>
```

23. Probamos.

24. Ahora adicionemos el estilo para los productos en oferta, modificando el **CollectionViewDictionary**:

```
…
   </DataTemplate>
   <DataTemplate x:Key="OfferStyle">
      <Grid
```

```
        Margin="15,10,15,0"
        HeightRequest="200"
        ColumnDefinitions=".3*,.7*"
        RowDefinitions="*,*">
        <Frame
            Grid.RowSpan="2"
            Grid.ColumnSpan="2"
            BorderColor="White">
            <Frame.Background>
                <LinearGradientBrush EndPoint="1,0">
                    <GradientStop Offset="0" Color="Yellow"/>
                    <GradientStop Offset="1" Color="#eeb54c"/>
                </LinearGradientBrush>
            </Frame.Background>
        </Frame>
        <Image
            Grid.RowSpan="2"
            HeightRequest="100"
            Source="{Binding Image}"/>
        <Label
            Grid.Column="2"
            FontSize="Title"
            FontAttributes="Bold"
            TextColor="White"
            Text="{Binding Name, StringFormat='OFFER: {0}'}"
            VerticalOptions="Center"/>
        <Label
            Grid.Column="2"
            Grid.Row="1"
            FontSize="Title"
            FontAttributes="Bold"
            TextColor="White"
            Text="{Binding Price, StringFormat='{0:C}'}"
            VerticalOptions="Center">
            <Label.FormattedText>
                <FormattedString>
                    <Span
                        Text="{Binding Price, StringFormat='{0:C}'}"
                        TextDecorations="Strikethrough"
                        TextColor="DarkRed"/>
                    <Span
                        Text="{Binding OfferPrice, StringFormat=' => {0:C}'}"/>
                </FormattedString>
            </Label.FormattedText>
        </Label>
    </Grid>
  </DataTemplate>
</ResourceDictionary>
```

25. Modificamos el **ProductDataTemplateSelector**:

```
using System;
using CollectionViewDemo.MVVM.Models;
namespace CollectionViewDemo.Selectors
```

```
{
        public class ProductDataTemplateSelector : DataTemplateSelector
        {
    protected override DataTemplate OnSelectTemplate(object item, BindableObject container)
    {
        var product = item as Product;
        if (!product.HasOffer)
        {
            Application.Current.Resources.TryGetValue("ProductStyle", out var productStyle);
            return productStyle as DataTemplate;
        }

        Application.Current.Resources.TryGetValue("OfferStyle", out var offerStyle);
        return offerStyle as DataTemplate;
    }
  }
}
```

26. Probamos.

27. Ahora vamos a implementar el Pull to Refresh.

28. Modificamos el **DataView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:DataTemplates="clr-namespace:CollectionViewDemo.Selectors"
             x:Class="CollectionViewDemo.MVVM.Views.DataView"
             Title="DataView">

   <ContentPage.Resources>
      <DataTemplates:ProductDataTemplateSelector x:Key="ProductTemplates"/>
   </ContentPage.Resources>

   <RefreshView
      Command="{Binding RefreshCommand}"
      IsRefreshing="{Binding IsRefreshing}">
      <CollectionView
         ItemsSource="{Binding Products}"
         ItemTemplate="{StaticResource ProductTemplates}">
      </CollectionView>
   </RefreshView>
</ContentPage>
```

29. Instalamos el paquete **PropertyChanged.Fody**.

30. Modificamos el **DataViewModel**:

```
using System;
using System.Collections.ObjectModel;
using System.Windows.Input;
using CollectionViewDemo.MVVM.Models;
```

```
namespace CollectionViewDemo.MVVM.ViewModels
{
        [AddINotifyPropertyChangedInterface]
        public class DataViewModel
        {
                public DataViewModel()
                {
        RefreshItems();
                }

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

    public bool IsRefreshing { get; set; }

    public ICommand RefreshCommand => new Command(async () => {
        IsRefreshing = true;
        await Task.Delay(3000);
        RefreshItems();
        IsRefreshing = false;
    });

    private void RefreshItems()
    {
        Products = new()
        {
            new Product
            {
                Name = "Yogurt",
                Price = 60.0m,
                Image = "yogurt.png",
                HasOffer = false,
                Stock = 28
            },
            new Product
            {
                Name = "Watermelon",
                Price = 30.0m,
                Image = "watermelon.png",
                HasOffer = false,
                Stock = 87
            },
            new Product
            {
                Name = "Water Bottle",
                Price = 80.0m,
                Image = "water_bottle.png",
                HasOffer = true,
                OfferPrice = 69.99m,
                Stock = 33
            },
            new Product
            {
                Name = "Tomato",
                Price = 120.0m,
```

```
            Image = "tomato.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Tea",
            Price = 65.0m,
            Image = "tea_bag.png",
            HasOffer = false,
            Stock = 82
        },
        new Product
        {
            Name = "Sparkling Drink",
            Price = 35.0m,
            Image = "sparkling_drink.png",
            HasOffer = false,
            Stock = 728
        },
        new Product
        {
            Name = "Spaguetti",
            Price = 15.0m,
            Image = "spaguetti.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Cream",
            Price = 48.0m,
            Image = "cream.png",
            HasOffer = false,
            Stock = 22
        },
        new Product
        {
            Name = "Snack",
            Price = 25.0m,
            Image = "009_snack.png",
            HasOffer = false,
            Stock = 2
        },
        new Product
        {
            Name = "Shrimp",
            Price = 300.0m,
            Image = "shrimp.png",
            HasOffer = true,
            OfferPrice = 250.0m,
            Stock = 58
        },
        new Product
```

```
            {
                Name = "Seasoning",
                Price = 185.0m,
                Image = "seasoning.png",
                HasOffer = false,
                Stock = 99
            },
            new Product
            {
                Name = "Sauce",
                Price = 220.0m,
                Image = "sauce.png",
                HasOffer = false,
                Stock = 72
            },
            new Product
            {
                Name = "Rice",
                Price = 48.0m,
                Image = "rice.png",
                HasOffer = false,
                Stock = 143
            },
            new Product
            {
                Name = "Peas",
                Price = 114.0m,
                Image = "peas.png",
                HasOffer = false,
                Stock = 0
            },
            new Product
            {
                Name = "Ham",
                Price = 215.0m,
                Image = "ham_1.png",
                HasOffer = true,
                OfferPrice = 189.0m,
                Stock = 732
            },
            new Product
            {
                Name = "Chicken Leg",
                Price = 142.0m,
                Image = "chicken_leg.png",
                HasOffer = true,
                OfferPrice = 125.0m,
                Stock = 20
            },
            new Product
            {
                Name = "Pizza",
                Price = 321.0m,
                Image = "pizza.png",
```

```
            HasOffer = false,
            Stock = 559
        },
        new Product
        {
            Name = "Pineapple",
            Price = 49.0m,
            Image = "pineapple.png",
            HasOffer = false,
            Stock = 41
        },
        new Product
        {
            Name = "Pepper",
            Price = 60.0m,
            Image = "pepper.png",
            HasOffer = true,
            OfferPrice = 30.0m,
            Stock = 64
        },
        new Product
        {
            Name = "Pasta",
            Price = 52.0m,
            Image = "pasta.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Oil Bottle",
            Price = 152.0m,
            Image = "oil_bottle",
            HasOffer = false,
            Stock = 87
        },
        new Product
        {
            Name = "Mushroom",
            Price = 28.0m,
            Image = "mushroom.png",
            HasOffer = false,
            Stock = 17
        },
        new Product
        {
            Name = "Milk Bottle",
            Price = 85.0m,
            Image = "milk_bottle.png",
            HasOffer = false,
            Stock = 39
        },
        new Product
        {
```

```
            Name = "Meat",
            Price = 450.0m,
            Image = "meat.png",
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Lemon",
            Price = 20.0m,
            Image = "lemon.png",
            HasOffer = false,
            Stock = 87
        },
        new Product
        {
            Name = "Tomato Sauce",
            Price = 15.0m,
            Image = "tomato_sauce.png",
            HasOffer = false,
            Stock = 26
        },
        new Product
        {
            Name = "Juice",
            Price = 60.0m,
            Image = "juice.png",
            HasOffer = false,
            Stock = 31
        },
        new Product
        {
            Name = "Ice Cream",
            Price = 251.0m,
            Image = "ice_cream.png",
            HasOffer = true,
            OfferPrice = 200.0m,
            Stock = 88
        },
        new Product
        {
            Name = "Ham",
            Price = 290.0m,
            Image = "ham.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Ice",
            Price = 125.0m,
            Image = "ice.png",
            HasOffer = false,
            Stock = 22
```

```
            },
            new Product
            {
                Name = "Flour",
                Price = 86.0m,
                Image = "flour.png",
                HasOffer = false,
                Stock = 28
            },
            new Product
            {
                Name = "Fish",
                Price = 440.0m,
                Image = "fish_1.png",
                HasOffer = false,
                Stock = 80
            },
            new Product
            {
                Name = "Fish 2",
                Price = 425.0m,
                Image = "fish.png",
                HasOffer = false,
                Stock = 24
            },
            new Product
            {
                Name = "Eggs",
                Price = 150.0m,
                Image = "eggs.png",
                HasOffer = false,
                Stock = 47
            },
            new Product
            {
                Name = "Cucumber",
                Price = 35.0m,
                Image = "cucumber.png",
                HasOffer = false,
                Stock = 74
            },
            new Product
            {
                Name = "Croissant",
                Price = 68.0m,
                Image = "croissant.png",
                HasOffer = true,
                OfferPrice = 50.0m,
                Stock = 27
            },
            new Product
            {
                Name = "Cookies",
                Price = 95.0m,
```

```
            Image = "cookie.png",
            HasOffer = false,
            Stock = 56
        },
        new Product
        {
            Name = "Coffee",
            Price = 154.0m,
            Image = "toffee.png",
            HasOffer = false,
            Stock = 83
        },
        new Product
        {
            Name = "Chocolate Bar",
            Price = 32.0m,
            Image = "chocolate_bar.png",
            HasOffer = false,
            Stock = 21
        },
        new Product
        {
            Name = "Cheese",
            Price = 36.0m,
            Image = "cheese.png",
            HasOffer = true,
            OfferPrice = 25.0m,
            Stock =73
        },
        new Product
        {
            Name = "Carrot",
            Price = 15.0m,
            Image = "carrot.png",
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Canned Food",
            Price = 89.0m,
            Image = "canned_food.png",
            HasOffer = false,
            Stock = 773
        },
        new Product
        {
            Name = "Soda",
            Price = 45.0m,
            Image = "can.png",
            HasOffer = false,
            Stock = 843
        },
        new Product
```

```
            {
                Name = "Candies",
                Price = 55.0m,
                Image = "candy.png",
                HasOffer = false,
                Stock = 71
            },
            new Product
            {
                Name = "Cake",
                Price = 250.0m,
                Image = "cake.png",
                HasOffer = true,
                OfferPrice = 200.0m,
                Stock = 0
            },
            new Product
            {
                Name = "Bread",
                Price = 100.0m,
                Image = "bread_1.png",
                HasOffer = false,
                Stock =134
            },
            new Product
            {
                Name = "Bread",
                Price = 85.0m,
                Image = "bread.png",
                HasOffer = false,
                Stock = 8
            },
            new Product
            {
                Name = "Banana",
                Price = 15.0m,
                Image = "banana.png",
                HasOffer = true,
                OfferPrice = 10.0m,
                Stock = 72
            },
            new Product
            {
                Name = "Apple",
                Price = 40.0m,
                Image = "apple.png",
                HasOffer = false,
                Stock = 737
            },
            new Product
            {
                Name = "Alcohol",
                Price = 370.0m,
                Image = "alcohol.png",
```

```
                HasOffer = false,
                Stock = 9
            },
        };
    }
  }
}
```

31. Probamos.

32. Ahora vamos a cargar los datos con paginación.

33. Modificamos la **DataViewModel**:

```
using System;
using System.Collections.ObjectModel;
using System.Windows.Input;
using CollectionViewDemo.MVVM.Models;
using PropertyChanged;

namespace CollectionViewDemo.MVVM.ViewModels
{
   [AddINotifyPropertyChangedInterface]
        public class DataViewModel
        {
                public DataViewModel()
                {
        RefreshItems();
                }

    public ObservableCollection<Product> Products { get; set; } = new();

    public bool IsRefreshing { get; set; }

    public ICommand RefreshCommand => new Command(async () => {
        IsRefreshing = true;
        await Task.Delay(3000);
        RefreshItems();
        IsRefreshing = false;
    });

    public ICommand ThresholdReachedCommand => new Command(async() =>
    {
        IsRefreshing = true;
        await Task.Delay(1000);
        RefreshItems(Products.Count);
        IsRefreshing = false;
    });

    private void RefreshItems(int lastIndex = 0)
    {
        int numberOfItemsPerPage = 10;
        var items = new ObservableCollection<Product>()
        {
```

```
            new Product
            {
                Name = "Yogurt",
                Price = 60.0m,
                Image = "yogurt.png",
                HasOffer = false,
                Stock = 28
            },
            new Product
            {
                Name = "Watermelon",
                Price = 30.0m,
                Image = "watermelon.png",
                HasOffer = false,
                Stock = 87
            },
            new Product
            {
                Name = "Water Bottle",
                Price = 80.0m,
                Image = "water_bottle.png",
                HasOffer = true,
                OfferPrice = 69.99m,
                Stock = 33
            },
            new Product
            {
                Name = "Tomato",
                Price = 120.0m,
                Image = "tomato.png",
                HasOffer = false,
                Stock = 0
            },
            new Product
            {
                Name = "Tea",
                Price = 65.0m,
                Image = "tea_bag.png",
                HasOffer = false,
                Stock = 82
            },
            new Product
            {
                Name = "Sparkling Drink",
                Price = 35.0m,
                Image = "sparkling_drink.png",
                HasOffer = false,
                Stock = 728
            },
            new Product
            {
                Name = "Spaguetti",
                Price = 15.0m,
                Image = "spaguetti.png",
```

```
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Cream",
            Price = 48.0m,
            Image = "cream.png",
            HasOffer = false,
            Stock = 22
        },
        new Product
        {
            Name = "Snack",
            Price = 25.0m,
            Image = "009_snack.png",
            HasOffer = false,
            Stock = 2
        },
        new Product
        {
            Name = "Shrimp",
            Price = 300.0m,
            Image = "shrimp.png",
            HasOffer = true,
            OfferPrice = 250.0m,
            Stock = 58
        },
        new Product
        {
            Name = "Seasoning",
            Price = 185.0m,
            Image = "seasoning.png",
            HasOffer = false,
            Stock = 99
        },
        new Product
        {
            Name = "Sauce",
            Price = 220.0m,
            Image = "sauce.png",
            HasOffer = false,
            Stock = 72
        },
        new Product
        {
            Name = "Rice",
            Price = 48.0m,
            Image = "rice.png",
            HasOffer = false,
            Stock = 143
        },
        new Product
        {
```

```
            Name = "Peas",
            Price = 114.0m,
            Image = "peas.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Ham",
            Price = 215.0m,
            Image = "ham_1.png",
            HasOffer = true,
            OfferPrice = 189.0m,
            Stock = 732
        },
        new Product
        {
            Name = "Chicken Leg",
            Price = 142.0m,
            Image = "chicken_leg.png",
            HasOffer = true,
            OfferPrice = 125.0m,
            Stock = 20
        },
        new Product
        {
            Name = "Pizza",
            Price = 321.0m,
            Image = "pizza.png",
            HasOffer = false,
            Stock = 559
        },
        new Product
        {
            Name = "Pineapple",
            Price = 49.0m,
            Image = "pineapple.png",
            HasOffer = false,
            Stock = 41
        },
        new Product
        {
            Name = "Pepper",
            Price = 60.0m,
            Image = "pepper.png",
            HasOffer = true,
            OfferPrice = 30.0m,
            Stock = 64
        },
        new Product
        {
            Name = "Pasta",
            Price = 52.0m,
            Image = "pasta.png",
```

```
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Oil Bottle",
            Price = 152.0m,
            Image = "oil_bottle",
            HasOffer = false,
            Stock = 87
        },
        new Product
        {
            Name = "Mushroom",
            Price = 28.0m,
            Image = "mushroom.png",
            HasOffer = false,
            Stock = 17
        },
        new Product
        {
            Name = "Milk Bottle",
            Price = 85.0m,
            Image = "milk_bottle.png",
            HasOffer = false,
            Stock = 39
        },
        new Product
        {
            Name = "Meat",
            Price = 450.0m,
            Image = "meat.png",
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Lemon",
            Price = 20.0m,
            Image = "lemon.png",
            HasOffer = false,
            Stock = 87
        },
        new Product
        {
            Name = "Tomato Sauce",
            Price = 15.0m,
            Image = "tomato_sauce.png",
            HasOffer = false,
            Stock = 26
        },
        new Product
        {
            Name = "Juice",
```

```
            Price = 60.0m,
            Image = "juice.png",
            HasOffer = false,
            Stock = 31
        },
        new Product
        {
            Name = "Ice Cream",
            Price = 251.0m,
            Image = "ice_cream.png",
            HasOffer = true,
            OfferPrice = 200.0m,
            Stock = 88
        },
        new Product
        {
            Name = "Ham",
            Price = 290.0m,
            Image = "ham.png",
            HasOffer = false,
            Stock = 0
        },
        new Product
        {
            Name = "Ice",
            Price = 125.0m,
            Image = "ice.png",
            HasOffer = false,
            Stock = 22
        },
        new Product
        {
            Name = "Flour",
            Price = 86.0m,
            Image = "flour.png",
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Fish",
            Price = 440.0m,
            Image = "fish_1.png",
            HasOffer = false,
            Stock = 80
        },
        new Product
        {
            Name = "Fish 2",
            Price = 425.0m,
            Image = "fish.png",
            HasOffer = false,
            Stock = 24
        },
```

```
            new Product
            {
                Name = "Eggs",
                Price = 150.0m,
                Image = "eggs.png",
                HasOffer = false,
                Stock = 47
            },
            new Product
            {
                Name = "Cucumber",
                Price = 35.0m,
                Image = "cucumber.png",
                HasOffer = false,
                Stock = 74
            },
            new Product
            {
                Name = "Croissant",
                Price = 68.0m,
                Image = "croissant.png",
                HasOffer = true,
                OfferPrice = 50.0m,
                Stock = 27
            },
            new Product
            {
                Name = "Cookies",
                Price = 95.0m,
                Image = "cookie.png",
                HasOffer = false,
                Stock = 56
            },
            new Product
            {
                Name = "Coffee",
                Price = 154.0m,
                Image = "toffee.png",
                HasOffer = false,
                Stock = 83
            },
            new Product
            {
                Name = "Chocolate Bar",
                Price = 32.0m,
                Image = "chocolate_bar.png",
                HasOffer = false,
                Stock = 21
            },
            new Product
            {
                Name = "Cheese",
                Price = 36.0m,
                Image = "cheese.png",
```

```
            HasOffer = true,
            OfferPrice = 25.0m,
            Stock =73
        },
        new Product
        {
            Name = "Carrot",
            Price = 15.0m,
            Image = "carrot.png",
            HasOffer = false,
            Stock = 28
        },
        new Product
        {
            Name = "Canned Food",
            Price = 89.0m,
            Image = "canned_food.png",
            HasOffer = false,
            Stock = 773
        },
        new Product
        {
            Name = "Soda",
            Price = 45.0m,
            Image = "can.png",
            HasOffer = false,
            Stock = 843
        },
        new Product
        {
            Name = "Candies",
            Price = 55.0m,
            Image = "candy.png",
            HasOffer = false,
            Stock = 71
        },
        new Product
        {
            Name = "Cake",
            Price = 250.0m,
            Image = "cake.png",
            HasOffer = true,
            OfferPrice = 200.0m,
            Stock = 0
        },
        new Product
        {
            Name = "Bread",
            Price = 100.0m,
            Image = "bread_1.png",
            HasOffer = false,
            Stock =134
        },
        new Product
```

```
            {
                Name = "Bread",
                Price = 85.0m,
                Image = "bread.png",
                HasOffer = false,
                Stock = 8
            },
            new Product
            {
                Name = "Banana",
                Price = 15.0m,
                Image = "banana.png",
                HasOffer = true,
                OfferPrice = 10.0m,
                Stock = 72
            },
            new Product
            {
                Name = "Apple",
                Price = 40.0m,
                Image = "apple.png",
                HasOffer = false,
                Stock = 737
            },
            new Product
            {
                Name = "Alcohol",
                Price = 370.0m,
                Image = "alcohol.png",
                HasOffer = false,
                Stock = 9
            },
        };
        var pageItems = items.Skip(lastIndex).Take(numberOfItemsPerPage);
        foreach (var item in pageItems)
        {
            Products.Add(item);
        }
    }
  }
}
```

34. Modificamos el **DataView**:

```
<RefreshView
  Command="{Binding RefreshCommand}"
  IsRefreshing="{Binding IsRefreshing}">
  <CollectionView
    ItemsSource="{Binding Products}"
    ItemTemplate="{StaticResource ProductTemplates}"
    RemainingItemsThreshold="1"
    RemainingItemsThresholdReachedCommand="{Binding ThresholdReachedCommand}">
  </CollectionView>
</RefreshView>
```

35. Probamos.

36. Ahora vamos a agregar menús desplegables.

37. Modificamos el **CollectionViewDictionary.xaml**:

```
…
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:ViewModels="clr-namespace:CollectionViewDemo.MVVM.ViewModels">
    <DataTemplate x:Key="ProductStyle">
        <SwipeView>
            <SwipeView.LeftItems>
                <SwipeItems>
                    <SwipeItem
                        BackgroundColor="DarkRed"
                        Command="{Binding Source={RelativeSource AncestorType={x:Type ViewModels:DataViewModel}},
Path=DeleteCommand}"
                        CommandParameter="{Binding}"
                        IconImageSource="trash.png"/>
                </SwipeItems>
            </SwipeView.LeftItems>
            <Grid
…
```

38. Modificamos el **DataViewModel**:

```
public ICommand DeleteCommand => new Command((p) => {
    Products.Remove((Product)p);
});
```

39. Ahora vamos a ver diferentes tipos de vistas que podemos usar con nuestro CollectionView.

40. En **Views** creamos el **LayoutsPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="CollectionViewDemo.MVVM.Views.LayoutsPage"
    Title="LayoutsPage">

    <CollectionView ItemsSource="{Binding Products}">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <Frame
                    Margin="15"
                    WidthRequest="200"
                    HeightRequest="250">
                    <VerticalStackLayout>
                        <Image Source="{Binding Image}"/>
```

```
                <Label
                    HorizontalTextAlignment="Center"
                    Text="{Binding Name}"/>
            </VerticalStackLayout>
        </Frame>
    </DataTemplate>
  </CollectionView.ItemTemplate>
 </CollectionView>
</ContentPage>
```

41. Modificamos el **LayoutsPage.xaml.cs**:

```
using CollectionViewDemo.MVVM.ViewModels;

namespace CollectionViewDemo.MVVM.Views;

public partial class LayoutsPage : ContentPage
{
        public LayoutsPage()
        {
                InitializeComponent();
                BindingContext= new DataViewModel();
        }
}
```

42. Ponemos esta como página de inicio:

```
MainPage = new LayoutsPage();
```

43. Probamos.

44. Podemos cambiar la orientación con una simple propiedad:

```
 <CollectionView
    ItemsSource="{Binding Products}"
    ItemsLayout="HorizontalList">
    <CollectionView.ItemTemplate>
```

45. Probamos.

46. O si lo prefieres de esta otra forma:

```
 <CollectionView
    ItemsSource="{Binding Products}">
    <CollectionView.ItemsLayout>
        <LinearItemsLayout
            ItemSpacing="50"
            Orientation="Horizontal"/>
    </CollectionView.ItemsLayout>
    <CollectionView.ItemTemplate>
```

47. Probamos.

48. Ahora vamos a probar los formatos grilla.

49. Cambiamos por:

```
<CollectionView
    ItemsSource="{Binding Products}">
    <CollectionView.ItemsLayout>
        <GridItemsLayout
            Span="2"
            Orientation="Horizontal"/>
    </CollectionView.ItemsLayout>
```

50. Probamos y jugamos con las propiedades de Orientation y quitarle los tamaños al Frame.

51. Ahora vamos a agregar Header y Footer.

52. Modificamos el **DataView**:

```
…
<CollectionView
    ItemsSource="{Binding Products}"
    Header="Products"
    Footer="End of list">
    <CollectionView.ItemsLayout>
        <LinearItemsLayout Orientation="Vertical"/>
    </CollectionView.ItemsLayout>
…
```

53. Probamos.

54. Ahora personalicemos nuestro Header y Footer. Modificamos el **DataView**:

```
…
<CollectionView ItemsSource="{Binding Products}">
    <CollectionView.Header>
        <Frame BackgroundColor="{StaticResource Primary}">
            <Label
                FontAttributes="Bold"
                Text="Products"
                TextColor="White"/>
        </Frame>
    </CollectionView.Header>
    <CollectionView.Footer>
        <HorizontalStackLayout>
            <Label FontSize="Title">
                <Label.FormattedText>
                    <FormattedString>
                        <Span
                            Text="Powered by: "
                            TextColor="{StaticResource Tertiary}"/>
                        <Span
                            Text=".NET MAUI"
                            TextColor="{StaticResource Primary}"/>
                    </FormattedString>
                </Label.FormattedText>
```

```
            </Label>
        </HorizontalStackLayout>
    </CollectionView.Footer>
    <CollectionView.ItemsLayout>
…
```

55. Probamos.

56. Ahora vamos a ver como seleccionamos elementos de la lista. Modificamos el **LayoutsPage**:

```
…
<CollectionView
  ItemsSource="{Binding Products}"
  SelectionMode="Single"
  SelectedItem="{Binding SelectedProduct}"
  SelectionChangedCommand="{Binding ProductChangedCommand}">
…
```

57. Modificamos el **DataViewModel**:

```
…
public Product SelectedProduct { get; set; }

public ICommand ProductChangedCommand => new Command(() =>
{
  var selectedProduct = SelectedProduct;
});
…
```

58. Probamos.

59. Ahora vamos a mostrar un mensaje sencillo si la lista está vacía.

60. Modificamos el **LayoutsPage**:

```
<CollectionView.EmptyView>
  <VerticalStackLayout
    HorizontalOptions="Center"
    VerticalOptions="Center"
    Spacing="20">
    <Image
      HeightRequest="150"
      Source="notfound.png"/>
    <Label
      FontAttributes="Bold"
      FontSize="Title"
      Text="No data found."/>
  </VerticalStackLayout>
</CollectionView.EmptyView>
```

61. Modificamos el **DataViewModel**:

```
public DataViewModel()
```

```
{
    //RefreshItems();
}
```

62. Probamos.

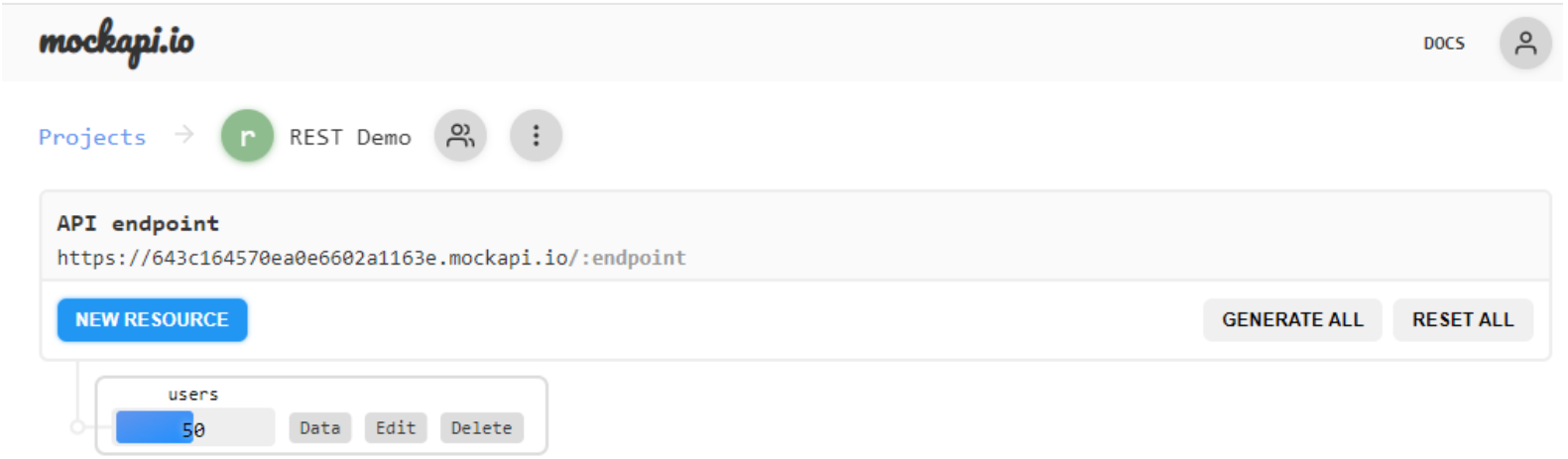
63. Volvemos a dejart el **DataViewModel** como lo teníamos:

```
public DataViewModel()
{
    RefreshItems();
}
```

64. Probamos.

Consumir APIs

1. Para esta demostración vamos a usar: https://mockapi.io/



2. Vamos a crear un proyecto llamado **RESTDemo**.

3. Creamos las carpetas **MVVM/Models**, **MVVM/Models** y **MVVM/ViewModels**.

4. Creamos el **MainViewModel**:

```
namespace RESTDemo.MVVM.ViewModels
{
    public class MainViewModel
    {
    }
}
```

5. Creamos el **MainView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="RESTDemo.MVVM.Views.MainView"
             Title="MainView">
    <VerticalStackLayout>
```

```
        <Label
            Text="Welcome to .NET MAUI!"
            VerticalOptions="Center"
            HorizontalOptions="Center" />
    </VerticalStackLayout>
</ContentPage>
```

6. Ligamos la **View** con la **ViewModel**:

```
public MainView()
{
        InitializeComponent();
        BindingContext = new MainViewModel();
}
```

7. Modificamos el **MainView**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="RESTDemo.MVVM.Views.MainView"
             Title="MainView">
    <VerticalStackLayout
        VerticalOptions="Center"
        HorizontalOptions="Center">
        <Button
            Command="{Binding AddUserCommand}"
            Text="Add User"/>
    </VerticalStackLayout>
</ContentPage>
```

8. Modificamos el **MainViewModel**:

```
using System.Text.Json;
using System.Windows.Input;

namespace RESTDemo.MVVM.ViewModels
{
    public class MainViewModel
    {
        private readonly HttpClient _client;
        private readonly JsonSerializerOptions _serializerOptions;
        private readonly string _baseUrl;

        public MainViewModel()
        {
            _client = new HttpClient();
            _baseUrl = "https://643c164570ea0e6602a1163e.mockapi.io";
            _serializerOptions = new JsonSerializerOptions
            {
                WriteIndented= true,
                PropertyNameCaseInsensitive = true,
            };
        }
```

```
    public ICommand AddUserCommand => new Command(async () =>
    {
        var response = await _client.GetStringAsync($"{_baseUrl}/users");
    });
  }
}
```

9. Colocamos la vista como página inicial:

```
public App()
{
        InitializeComponent();
        MainPage = new NavigationPage(new MainView());
}
```

10. Probamos.

11. Generamos un error y vemos que no es la mejor manera de consumir una API, cambiamos nuestro código por:

```
public ICommand AddUserCommand => new Command(async () =>
{
  var response = await _client.GetAsync($"{_baseUrl}/users");
  if (response.IsSuccessStatusCode)
  {
    var content = await response.Content.ReadAsStringAsync();
  }
});
```

12. Probamos.

13. Pero para mejorar aun más la cosa, vamos a crear el modelo **User**:

```
namespace RESTDemo.MVVM.Modes
{
  public class User
  {
    public DateTime CreatedAt { get; set; }

    public string Name { get; set; }

    public string Avatar { get; set; }

    public string Id { get; set; }
  }
}
```

14. Modificamos nuevamente nuestro **MainViewModel**:

```
public ICommand AddUserCommand => new Command(async () =>
{
  var response = await _client.GetAsync($"{_baseUrl}/users");
  if (response.IsSuccessStatusCode)
  {
```

```
        using(var responseStream = await response.Content.ReadAsStreamAsync())
        {
            var data = await JsonSerializer.DeserializeAsync<List<User>>(responseStream, _serializerOptions);
        }
    }
});
```

15. Probamos.

16. Modificamos el **MainView**:

```
<Button
    Command="{Binding GetAllUsersCommand}"
    Text="Get All Users"/>
<Button
    Command="{Binding GetSingleUserCommand}"
    Margin="0,5,0,0"
    Text="Get Gingle User"/>
```

17. Modificamos el **MainViewModel**:

```
public ICommand GetAllUsersCommand => new Command(async () =>
{
    var response = await _client.GetAsync($"{_baseUrl}/users");
    if (response.IsSuccessStatusCode)
    {
        using (var responseStream = await response.Content.ReadAsStreamAsync())
        {
            var data = await JsonSerializer.DeserializeAsync<List<User>>(responseStream, _serializerOptions);
        }
    }
});

public ICommand GetSingleUserCommand => new Command(async () =>
{
    var response = await _client.GetAsync($"{_baseUrl}/users/10");
    if (response.IsSuccessStatusCode)
    {
        using (var responseStream = await response.Content.ReadAsStreamAsync())
        {
            var data = await JsonSerializer.DeserializeAsync<User>(responseStream, _serializerOptions);
        }
    }
});
```

18. Probamos.

19. Ahora vamos a adicionar un registro.

20. Modificamos el **MainView**:

```
<Button
    Command="{Binding AddUserCommand}"
    Margin="0,5,0,0"
```

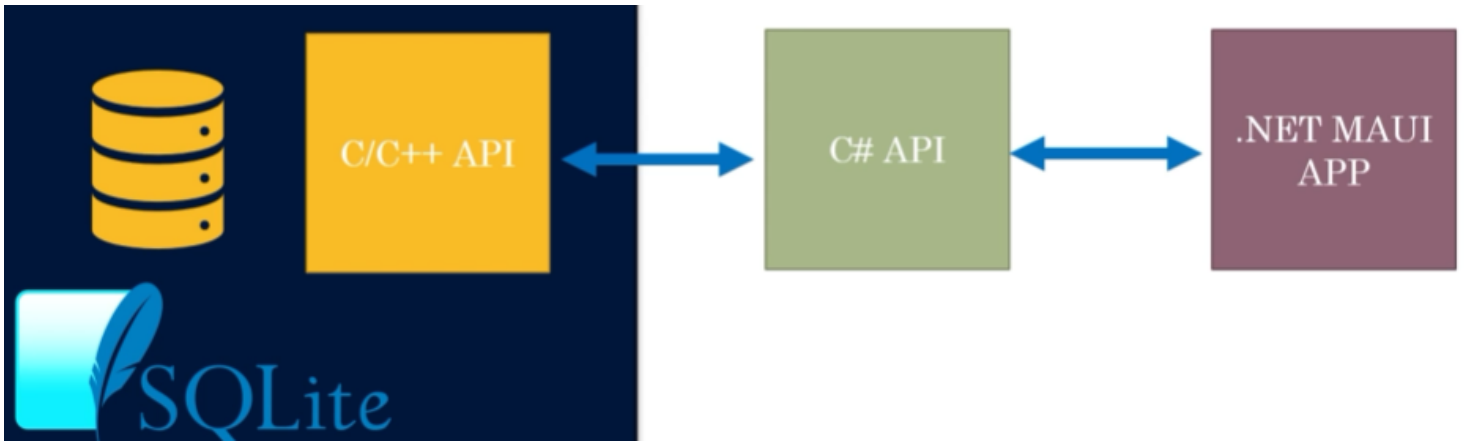
```
    Text="Add User"/>
```

21. Modificamos el **MainViewModel**:

```
public ICommand AddUserCommand => new Command(async () =>
{
    var user = new User
    {
        CreatedAt = DateTime.UtcNow,
        Name = "Zulu",
        Avatar = "https://fakeimg.pl/350x200/?text=MAUI"
    };
    var json = JsonSerializer.Serialize(user, _serializerOptions);
    var content = new StringContent(json, Encoding.UTF8, "application/json");
    var response = await _client.PostAsync($"{_baseUrl}/users", content);
});
```

22. Probamos.

SQLite



Atributo	Descripción
[Table(name)]	Specifies the name of the table
[Column(name)]	Specifies the column name
[PrimaryKey]	Specifies whether a property is the primary key
[AutoIncrement]	Specifies whether the property will be incremented automatically
[Indexed]	Specifies whether the column is indexed
[MaxLength(value)]	Specifies a maximum length for a property
[Unique]	Specifies whether the property value will be unique in a column.
[NotNull]	Specifies that the field value must not be empty.
[Ignore]	Specifies that a property will not be part of the table.
[Collation]	Specifies how text values are compared to determine order and equality

1. Creamos un nuevo proyecto llamado **SQLiteDemo**.

2. Luego agregamos el paquete **sqlite-net-pcl**.

3. Creamos la clase **Constants**:

```
using SQLite;

namespace SQLiteDemo
{
    public static class Constants
    {
        private const string dbFileName = "SQLite.db3";

        public const SQLiteOpenFlags flags = SQLiteOpenFlags.ReadWrite |
                                             SQLiteOpenFlags.Create |
                                             SQLiteOpenFlags.SharedCache;

        public static string DatabasePath => Path.Combine(FileSystem.AppDataDirectory, dbFileName);

    }
}
```

4. Creamos las carpetas **MVVM/Models**, **MVVM/Models** y **MVVM/ViewModels**.

5. Creamos el modelo **Customer**:

```
using SQLite;

namespace SQLiteDemo.MVVM.Models
{
    [Table("Customers")]
    public class Customer
```

```
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }

        [Indexed, MaxLength(100), NotNull]
        public string Name { get; set; }

        [MaxLength(20), Unique]
        public string Phone { get; set; }

        public int Age { get; set; }

        [MaxLength(100)]
        public string Address { get; set; }
    }
}
```

6. Creamos la carpeta **Repository** y dentro de esta creamos el **CustomerRepository**:

```
using SQLite;
using SQLiteDemo.MVVM.Models;

namespace SQLiteDemo.Repository
{
    public class CustomerRepository
    {
        private readonly SQLiteConnection _connection;

        public CustomerRepository()
        {
            _connection = new SQLiteConnection(Constants.DatabasePath, Constants.Flags);
            _connection.CreateTable<Customer>();
        }

        public string StatusMessage { get; set; }

        public void AddOrUpdate(Customer customer)
        {
            try
            {
                int result = 0;
                if (customer.Id == 0)
                {
                    result = _connection.Insert(customer);
                    StatusMessage = $"{result} row(s) added.";
                }
                else
                {
                    result = _connection.Update(customer);
                    StatusMessage = $"{result} row(s) updated.";
                }
            }
            catch (Exception ex)
            {
```

```
            StatusMessage = $"Error {ex.Message}.";
        }
    }

    public List<Customer> GetAll()
    {
        try
        {
            return _connection.Table<Customer>().ToList();
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public List<Customer> GetAll2()
    {
        try
        {
            return _connection.Query<Customer>("SELECT * FROM Customers").ToList();
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public Customer Get(int id)
    {
        try
        {
            return _connection.Table<Customer>().FirstOrDefault(c => c.Id == id);
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public void Delete(int id)
    {
        try
        {
            var customer = Get(id);
            _connection.Delete(customer);
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
        }
```

```
    }
  }
}
```

7. Modificamos el **App.xaml.cs**:

```
using SQLiteDemo.Repository;

namespace SQLiteDemo;

public partial class App : Application
{
        public static CustomerRepository CustomerRepo { get; private set; }

        public App(CustomerRepository repo)
        {
                InitializeComponent();
                CustomerRepo = repo;
                MainPage = new AppShell();
        }
}
```

8. Modificamos el **MauiProgram**:

```
builder.Services.AddSingleton<CustomerRepository>();
return builder.Build();
```

9. Borramos el **MainPage** y el **AppShell**.

10. Dentro de la carpeta Views creamos el nuevo **MainPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SQLiteDemo.MVVM.Views.MainPage"
             Title="MainPage">
    <Grid
        RowDefinitions=".2*,.8*"
        Padding="5">
        <VerticalStackLayout>
            <Entry Placeholder="Name..."/>
            <Entry Placeholder="Address..."/>
            <Button Text="Add or Update"/>
        </VerticalStackLayout>
        <CollectionView Grid.Row="1">

        </CollectionView>
    </Grid>
</ContentPage>
```

11. Cambiamos la página de inicio:

```
MainPage = new NavigationPage(new MainPage());
```

12. Y al tratar de correr, obenermos un error. Debemos instalar el paquete: **SQLitePCLRaw.provider.dynamic_cdecl**:

13. Probamos.

14. Ahora vamos a crear el ViewModel de esta página. Adicionamos el **MainPageViewModel**:

```
using SQLiteDemo.MVVM.Models;

namespace SQLiteDemo.MVVM.ViewModels
{
    public class MainPageViewModel
    {
        public List<Customer> Customers { get; set; }

        public Customer CurrentCustomer { get; set; }
    }
}
```

15. Modificamos la **MainPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SQLiteDemo.MVVM.Views.MainPage"
             Title="MainPage">
    <Grid
        RowDefinitions=".2*,.8*"
        Padding="5">
        <VerticalStackLayout>
            <Entry Placeholder="Name..."/>
            <Entry Placeholder="Address..."/>
            <Button
                Command="{Binding AddOrUpdateCommand}"
                Text="Add or Update"/>
        </VerticalStackLayout>
        <CollectionView
            ItemsSource="{Binding Customers}"
            Grid.Row="1">

        </CollectionView>
    </Grid>
</ContentPage>
```

16. Ligamos la View con la ViewModel:

```
BindingContext = new MainPageViewModel();
```

17. Probamos.

18. Instalamos el paquete **Bogus**.

19. Instalamos el paquete **PropertyChanged.Fody**.

20. Modificamos el **MainPageViewModel**:

```
using Bogus;
using PropertyChanged;
using SQLiteDemo.MVVM.Models;
using System.Windows.Input;

namespace SQLiteDemo.MVVM.ViewModels
{
    [AddINotifyPropertyChangedInterface]
    public class MainPageViewModel
    {
        public MainPageViewModel()
        {
            GenerateNewCustomer();
        }

        public List<Customer> Customers { get; set; }

        public Customer CurrentCustomer { get; set; }

        public ICommand AddOrUpdateCommand => new Command(() =>
        {
            App.CustomerRepo.AddOrUpdate(CurrentCustomer);
            Console.WriteLine(App.CustomerRepo.StatusMessage);
            GenerateNewCustomer();
        });

        private void GenerateNewCustomer()
        {
            CurrentCustomer = new Faker<Customer>()
                .RuleFor(x => x.Name, f => f.Person.FullName)
                .RuleFor(x => x.Address, f => f.Person.Address.Street)
                .Generate();
        }
    }
}
```

21. Modificamos la **MainPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SQLiteDemo.MVVM.Views.MainPage"
             Title="MainPage">
    <Grid
        RowDefinitions=".2*,.8*"
        Padding="5">
        <VerticalStackLayout>
            <Entry
                Text="{Binding CurrentCustomer.Name}"
                Placeholder="Name..."/>
            <Entry
                Text="{Binding CurrentCustomer.Address}"
```

```
            Placeholder="Address..."/>
        <Button
            Command="{Binding AddOrUpdateCommand}"
            Text="Add or Update"/>
    </VerticalStackLayout>
    <CollectionView
        ItemsSource="{Binding Customers}"
        Grid.Row="1">

    </CollectionView>
  </Grid>
</ContentPage>
```

22. Probamos.

23. Modificamos nuestra **MainPageViewModel**:

```
public MainPageViewModel()
{
  Refresh();
  GenerateNewCustomer();
}

public List<Customer> Customers { get; set; }

public Customer CurrentCustomer { get; set; }

public ICommand AddOrUpdateCommand => new Command(() =>
{
  App.CustomerRepo.AddOrUpdate(CurrentCustomer);
  Console.WriteLine(App.CustomerRepo.StatusMessage);
  GenerateNewCustomer();
  Refresh();
});

private void GenerateNewCustomer()
{
  CurrentCustomer = new Faker<Customer>()
    .RuleFor(x => x.Name, f => f.Person.FullName)
    .RuleFor(x => x.Address, f => f.Person.Address.Street)
    .Generate();
}

private void Refresh()
{
  Customers = App.CustomerRepo.GetAll();
}
```

24. Probamos.

25. Modificamos nuestra **MainPage**:

```
<CollectionView
  ItemsSource="{Binding Customers}"
```

```
    Grid.Row="1">
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <Grid ColumnDefinitions="*,*">
                <Label
                    Text="{Binding Name}"/>
                <Label
                    Grid.Column="1"
                    Text="{Binding Address}"/>
            </Grid>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

26. Probamos.

27. Ahora vamos a actualizar los registros de una forma muy simple, actualizamos el **MainPage**:

```
…
<CollectionView
    ItemsSource="{Binding Customers}"
    SelectionMode="Single"
    SelectedItem="{Binding CurrentCustomer}"
    Grid.Row="1">
…
```

28. Probamos.

29. Ahora posibilitemos el borrado de registros. Modificamos el **MainPage**:

```
…
            xmlns:local="clr-namespace:SQLiteDemo.MVVM.ViewModels"
…
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <SwipeView>
                        <SwipeView.LeftItems>
                            <SwipeItems>
                                <SwipeItem
                                    Command="{Binding Source={RelativeSource AncestorType={x:Type local:MainPageViewModel}},
Path=DeleteCommand}"
                                    Text="Delete"
                                    BackgroundColor="Purple"/>
                            </SwipeItems>
                        </SwipeView.LeftItems>
                        <VerticalStackLayout
                            HeightRequest="30"
                            Margin="0,0,0,5">
                            <Grid ColumnDefinitions="*,*">
                                <Label
                                    Text="{Binding Name}"/>
                                <Label
                                    Grid.Column="1"
                                    Text="{Binding Address}"/>
```

```
                    </Grid>
                </VerticalStackLayout>
            </SwipeView>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

30. Modificamos el **MainPageViewModel**:

```
public ICommand DeleteCommand => new Command(() =>
{
    App.CustomerRepo.Delete(CurrentCustomer.Id);
    Refresh();
});
```

31. Probamos.

32. Ahora vamos a sobre cargar este método en el **CustomerRepository**:

```
public List<Customer> GetAll(Expression<Func<Customer, bool>> predicate)
{
    try
    {
        return _connection.Table<Customer>()
            .Where(predicate)
            .ToList();
    }
    catch (Exception ex)
    {
        StatusMessage = $"Error {ex.Message}.";
        return null;
    }
}
```

33. Modificamos el **MainPegeViewModel**:

```
private void Refresh()
{
    //Customers = App.CustomerRepo.GetAll();
    Customers = App.CustomerRepo.GetAll(x => x.Name.StartsWith("A"));
}
```

34. Probamos y luego volvemos a dejar el método **Refresh** como estaba.

Definiendo un repositorio genérico

1. Creamos el **TableData** en la carpeta **Abstractions**:

```
using SQLite;

namespace SQLiteDemo.Abstractions
{
    public class TableData
    {
```

```
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; }
    }
}
```

2. Modificamos la definición de **Customers**:

```
using SQLite;
using SQLiteDemo.Abstractions;

namespace SQLiteDemo.MVVM.Models
{
    [Table("Customers")]
    public class Customer : TableData
    {
        [Indexed, MaxLength(100), NotNull]
        public string Name { get; set; }

        [MaxLength(20), Unique]
        public string Phone { get; set; }

        public int Age { get; set; }

        [MaxLength(100)]
        public string Address { get; set; }
    }
}
```

3. Cremos la tabla **Orders** para efectos de prueba:

```
using SQLiteDemo.Abstractions;

namespace SQLiteDemo.MVVM.Models
{
    public class Order : TableData
    {
        public int CustomerId { get; set; }

        public DateTime OrderDate { get; set; }
    }
}
```

4. Creamos la interfaz **IBaseReposityory**:

```
using System.Linq.Expressions;

namespace SQLiteDemo.Abstractions
{
    public interface IBaseRepository<T> : IDisposable where T : TableData, new()
    {
        void SaveItem(T item);

        T GetItem(int id);
```

```
        T GetItem(Expression<Func<T, bool>> predicate);

        List<T> GetItems();

        List<T> GetItems(Expression<Func<T, bool>> predicate);

        void DeleteItem(T item);
    }
}
```

5. Creamos el **BaseRepository**:

```
using SQLite;
using SQLiteDemo.Abstractions;
using System.Linq.Expressions;

namespace SQLiteDemo.Repository
{
    public class BaseRepository<T> : IBaseRepository<T> where T : TableData, new()
    {
        private readonly SQLiteConnection _connection;

        public BaseRepository()
        {
            _connection = new SQLiteConnection(Constants.DatabasePath, Constants.Flags);
            _connection.CreateTable<T>();
        }

        public string StatusMessage { get; set; }

        public void DeleteItem(T item)
        {
            try
            {
                _connection.Delete(item);
            }
            catch (Exception ex)
            {
                StatusMessage = $"Error {ex.Message}.";
            }
        }

        public void Dispose()
        {
            _connection.Close();
        }

        public T GetItem(int id)
        {
            try
            {
                return _connection.Table<T>().FirstOrDefault(c => c.Id == id);
            }
            catch (Exception ex)
```

```
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public T GetItem(Expression<Func<T, bool>> predicate)
    {
        try
        {
            return _connection.Table<T>()
                .Where(predicate)
                .FirstOrDefault();
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public List<T> GetItems()
    {
        try
        {
            return _connection.Table<T>().ToList();
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public List<T> GetItems(Expression<Func<T, bool>> predicate)
    {
        try
        {
            return _connection.Table<T>()
                .Where(predicate)
                .ToList();
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
            return null;
        }
    }

    public void SaveItem(T item)
    {
        try
        {
            int result = 0;
```

```
            if (item.Id == 0)
            {
                result = _connection.Insert(item);
                StatusMessage = $"{result} row(s) added.";
            }
            else
            {
                result = _connection.Update(item);
                StatusMessage = $"{result} row(s) updated.";
            }
        }
        catch (Exception ex)
        {
            StatusMessage = $"Error {ex.Message}.";
        }
    }
}
```

6. Modificamos el **App**:

```
using SQLiteDemo.MVVM.Models;
using SQLiteDemo.MVVM.Views;
using SQLiteDemo.Repository;

namespace SQLiteDemo;

public partial class App : Application
{
    public static BaseRepository<Customer> CustomerRepo { get; private set; }

    public static BaseRepository<Order> OrderRepo { get; private set; }

    public App(BaseRepository<Customer> customerRepo, BaseRepository<Order> orderRepo)
        {
                InitializeComponent();
        CustomerRepo = customerRepo;
        OrderRepo = orderRepo;
        MainPage = new NavigationPage(new MainPage());
        }
}
```

7. Modificamos el **MauiProgram**:

```
builder.Services.AddSingleton<BaseRepository<Customer>>();
builder.Services.AddSingleton<BaseRepository<Order>>();
return builder.Build();
```

8. Borramos el **CustomerRepository**.

9. Probamos.

HASTA ACÁ HE PREPARADO