# Índice

Contenido

[Índice 1](#_Toc147428576)

[Modelo de datos 2](#_Toc147428577)

[Arquitectura del proyecto 2](#_Toc147428578)

[Crear la base de datos con EF 6](#_Toc147428579)

[Clase DataContext 8](#_Toc147428580)

[Appsettings 9](#_Toc147428581)

[Inyección de dependencias Servicio SQlServer 9](#_Toc147428582)

[Ejecutar proyectos de forma múltiple 10](#_Toc147428583)

[Migración de entidades hacia la base de datos SQL SERVER 10](#_Toc147428584)

[Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT. 11](#_Toc147428585)

[Creando los primeros métodos en el primer controlador 13](#_Toc147428586)

[Creando nuestros primeros componentes en Blazor 15](#_Toc147428587)

[Implementando el patrón repositorio 16](#_Toc147428588)

[Creando componente Generic List 21](#_Toc147428589)

[Creando El Index de la página Owners 22](#_Toc147428590)

[Completando las acciones de crear, editar y borrar Owners 26](#_Toc147428591)

[Creando las tablas de usuarios 42](#_Toc147428592)

[Seguridad desde el backend(API) 48](#_Toc147428593)

[Habilitando tokens en swagger 56](#_Toc147428594)

# Modelo de datos

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

Owner

Pet

Pet Type

History

Service Type

Agenda

1

\*

1

1

1

1

1

\*

\*

\*

\*

\*

User

UserRole

Role

Manager

1

1

1

1

1

1

\*

\*

# Arquitectura del proyecto

SQL Server on Azure

.NET Core 7 API

Blazor WEB Application

Library Class



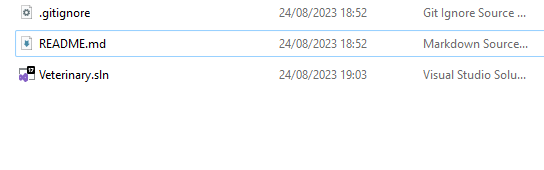
Crear proyecto

Vamos a crear esta estructura en Visual Studio (asegúrese de poner todos los proyectos en el mismo directorio C://Projects

Crear un nuevo repositorio GITHUB, usar gitignore, copiar ruta, repositorio público

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

* Nuevo proyecto Solution Blank llamado **Veterinary**. Dentro de C://Projects al final la .sln queda el ícono dentro de Projects🡪(Veterinary.sln)



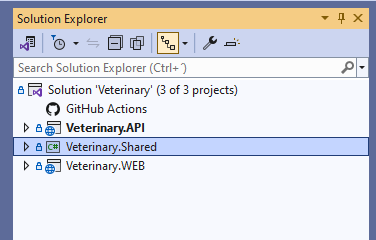
* Sobre el ícono Veterinary.sln del explorador de soluciones oprimimos click derecho y presionamos Open
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Class Library**, llamado **Veterinary.Shared**

Ubicación: C:\Projects/Veterinary (Borramos la class1.cs que se genera)

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

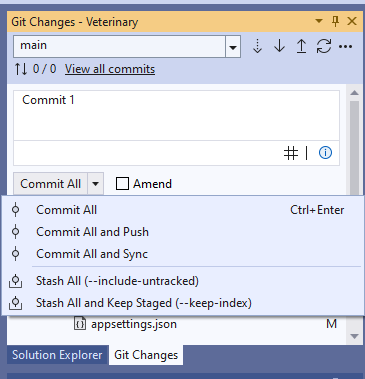
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **.NET MAUI App**, llamado **Veterinary.Mobile**. Ubicación: C:\Projects/Veterinary

Así debe verse al final , la estructura de los proyectos en el Solution Explorer:



Hacemos el primer commit en nuestro repositorio. Pestaña Git Changes

\*(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente) Commit All and Sync



# Crear la base de datos con EF

Diagrama

Descripción generada automáticamente

Code First y Database First. En este curso trabajaremos con EF Code First,

Documentación: <https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db>

1. Empecemos creando en el proyecto **Veterinary.Shared** la carpeta **Entities** y dentro de esta carpeta la entidad **Owner**:

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class Owner

{

public int Id { get; set; }

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[Display(Name = "Fixed Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string FixedPhone { get; set; }

[Display(Name = "Cell Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string CellPhone { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

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

public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

# Clase DataContext

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

using Microsoft.EntityFrameworkCore;

using Veterinary.Shared.Entities;

namespace Veterinary.API.Data

{

public class DataContext : DbContext

{

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

{

}

public DbSet<Owner> Owners { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

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

}

}

}

# Appsettings

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

{

"ConnectionStrings": {

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

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

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

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

# Inyección de dependencias Servicio SQlServer

1. Configurar la inyección del DataContext en la clase **Program** del proyecto **API**:

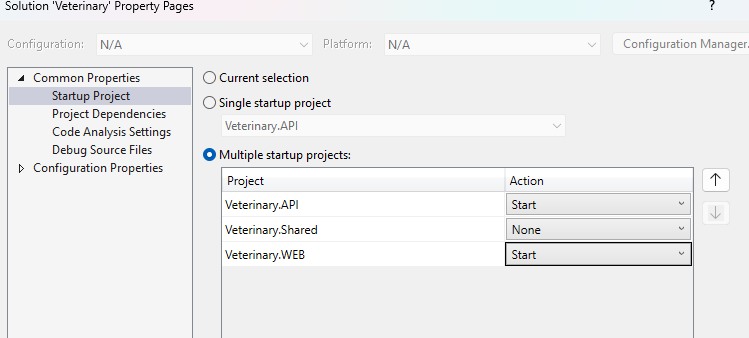
builder.Services.AddSwaggerGen();

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

var app = builder.Build();

# Ejecutar proyectos de forma múltiple

1. En el desplegable Startup Projects seleccionar Veterinary.API como proyecto de inicio, abrir Package Manager Console(Tool) , e igualmente elegir Veterinary.API ,como Default Project



# Migración de entidades hacia la base de datos SQL SERVER

1. Correr los siguientes comandos en Package Manager Console:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

add-migration InitialDb

update-database

# Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Hacemos nuestro segundo **Commit**. All and Sync y luego el Push en una nuevo branch(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente).

# Creando los primeros métodos en el primer controlador

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

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Veterinary.API.Data;

using Veterinary.Shared.Entities;

namespace Veterinary.API.Controllers

{

[ApiController]

[Route("/api/Owners")]

public class OwnersController : ControllerBase

{

private readonly DataContext \_context;

public OwnersController(DataContext context)

{

\_context = context;

}

//Get con lista

//Select \* From owners

[HttpGet]

public async Task<ActionResult> Get()

{

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

}

// Get por parámetro

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

public async Task<ActionResult> Get(int id)

{

//200 Ok

var owner = await \_context.Owners.FirstOrDefaultAsync(x => x.Id == id);

if(owner == null)

{

return NotFound();

}

return Ok(owner);

}

[HttpPost]

public async Task<ActionResult> Post(Owner Owner)

{

\_context.Add(Owner);

await \_context.SaveChangesAsync();

return Ok(Owner);

}

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

app.MapControllers();

app.UseCors(x => x

.AllowAnyMethod()

.AllowAnyHeader()

.SetIsOriginAllowed(origin => true)

.AllowCredentials());

app.Run();

1. Borramos las clases de **WeatherForecast**.
2. Probamos la creación y listado de Owners por el **swagger** y por **Postman**.
3. Hacemos el **commit en Branch** de lo que llevamos.

# Creando nuestros primeros componentes en Blazor

1. Ahora vamos listar y crear países por la interfaz WEB.

Primero configuramos en el proyecto **WEB** la dirección por la cual sale nuestra **API**.

Verificar en cada proyecto el puerto por el cual se despliega, pues el puerto cambia en cada solución.

En mi caso la uri es: https://localhost:7000

**Veterinary.WEB🡪 Program**

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7000/") });

# Implementando el patrón repositorio

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

using System.Net;

namespace Veterinary.WEB.Repositories {

public class HttpResponseWrapper<T>

{

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

{

Error = error;

Response = response;

HttpResponseMessage = httpResponseMessage;

}

public bool Error { get; set; }

public T? Response { get; set; }

public HttpResponseMessage HttpResponseMessage { get; set; }

public async Task<string?> GetErrorMessage()

{

if (!Error)

{

return null;

}

var codigoEstatus = HttpResponseMessage.StatusCode;

if (codigoEstatus == HttpStatusCode.NotFound)

{

return "Recurso no encontrado";

}

else if (codigoEstatus == HttpStatusCode.BadRequest)

{

return await HttpResponseMessage.Content.ReadAsStringAsync();

}

else if (codigoEstatus == HttpStatusCode.Unauthorized)

{

return " Debes loguearte para realizar esta acción";

}

else if (codigoEstatus == HttpStatusCode.Forbidden)

{

return " No tienes permisos para ejecutar esta acción";

}

return "Ha ocurrido un error inesperado";

}

}

}

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

namespace Veterinary.WEB.Repositories{

public interface IRepository

{

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

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

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

}

}

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

using System.Text;

using System.Text.Json;

namespace Veterinary.WEB.Repositories

{

public class Repository : IRepository

{

private readonly HttpClient \_httpClient;

private JsonSerializerOptions \_jsonDefaultOptions => new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

};

public Repository(HttpClient httpClient)

{

\_httpClient = httpClient;

}

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

{

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

if (responseHttp.IsSuccessStatusCode)

{

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

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

}

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

}

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

{

var mesageJSON = JsonSerializer.Serialize(model);

var messageContet = new StringContent(mesageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PostAsync(url, messageContet);

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

}

public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContet = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PostAsync(url, messageContet);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<TResponse>(responseHttp, \_jsonDefaultOptions);

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

}

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

}

private async Task<T> UnserializeAnswer<T>(HttpResponseMessage httpResponse, JsonSerializerOptions jsonSerializerOptions)

{

var respuestaString = await httpResponse.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<T>(respuestaString, jsonSerializerOptions)!;

}

}

}

8

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

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7000/") });

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

await builder.Build().RunAsync();

# Creando componente Generic List

1. En la carpeta **Shared del proyecto WEB** creamos el componente razor **GenericList**:

@typeparam Titem

@if(MyList is null)

{

@if(Loading is null)

{

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

<img src=" https://i.gifer.com/ZZ5H.gif" />

</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!;

}

# Creando El Index de la página Owners

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

@page "/owners"

@inject IRepository repository

<h3>Owners</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/owners/create">New Owner</a>

</div>

<**GenericList** **MyList**="Owners">

<**Body**>

<table class="table table-striped">

<thead>

<tr>

<th>Owner</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var owner in Owners!)

{

<tr>

<td>

@owner.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</**Body**>

</**GenericList**>

@code {

public List<Owner>? Owners { get; set; }

protected async override Task OnInitializedAsync()

{

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

Owners = responseHppt.Response!;

}

}

\*Importante:

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

Veterinary.Shares

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

@using Veterinary.WEB.Shared

@using Veterinary.Shared.Entities

@using Veterinary.WEB.Repositories

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

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

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

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

</NavLink>

</div>

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

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

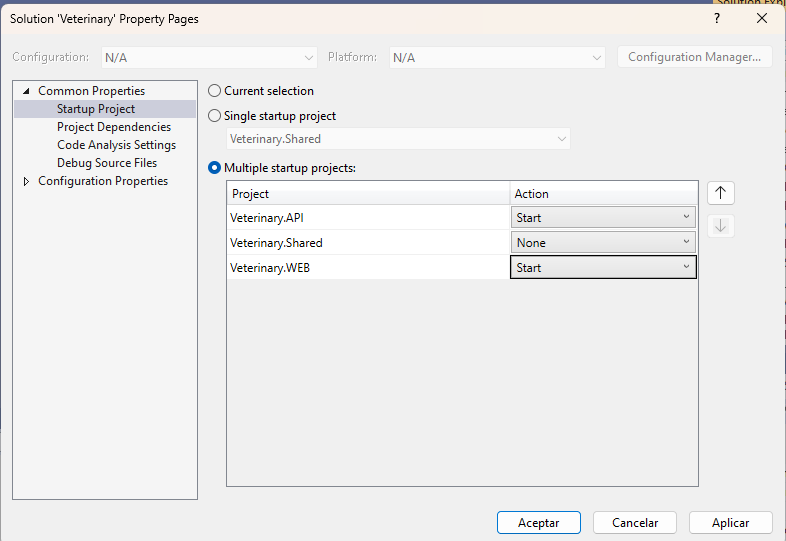
<span class="oi oi-list-rich" aria-hidden="true"></span> Owners

</NavLink>

</div>

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

Vamos a las Solution Veterinary, click derecho properties:



1. Probamos y hacemos nuestro commit.

# Completando las acciones de crear, editar y borrar Owners

1. En el proyecto **API** vamos a adicionar estos métodos al **OwnersController**:

[HttpPut]

public async Task<ActionResult> Put(Owner owner)

{

\_context.Update(owner);

await \_context.SaveChangesAsync();

return Ok(owner);

}

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

public async Task<ActionResult> Delete(int id)

{

var afectedRows = await \_context.Owners

.Where(x => x.Id == id)

.ExecuteDeleteAsync();

if (afectedRows == 0)

{

return NotFound();

}

return NoContent();

}

1. Probamos estos métodos por **Swagger** o por **Postman**.
2. Agregamos estos métodos a la interfaz **IRepository**.

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

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

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

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

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

{

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

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

}

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

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PutAsync(url, messageContent);

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

}

public async Task<HttpResponseWrapper<TResponse>> Put<T, TResponse>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PutAsync(url, messageContent);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<TResponse>(responseHttp, \_jsonDefaultOptions);

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

}

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

}

1. Vamos agregarle al proyecto **WEB** el nugget **CurrieTechnologies.Razor.SweetAlert2**, que nos va a servir para mostrar alertas muy bonitas.
2. Vamos a la página de Sweet Alert 2 ([Basaingeal/Razor.SweetAlert2: A Razor class library for interacting with SweetAlert2 (github.com)](https://github.com/Basaingeal/Razor.SweetAlert2) y copiamos el script que debemos de agregar al **index.html** que está en el **wwwroot** de nuestro proyecto **WEB**.

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

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

</body>

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

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

builder.Services.AddSweetAlert2();

Insertar lo siguiente en el archivo imports del proyecto Web

@using Currie.Technologies.Razor.sweetAlert2

1. En la carpeta **Owners** agregar el componente **OwnerForm**:

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>País:</label>

<div>

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

<ValidationMessage For="@(() => Owner.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 Owner Owner { get; set; } = null!;

[EditorRequired]

[Parameter]

public EventCallback OnValidSubmit { get; set; }

[EditorRequired]

[Parameter]

public EventCallback ReturnAction { get; set; }

}

1. En la carpeta **Owners** agregar el componente **OwnerCreate**:

@page "/Owners/create"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Crear País</h3>

<OwnerForm Owner="Owner" OnSubmit="Create" ReturnAction="Return"/>

@code {

private Owner Owner = new();

private async Task Create()

{

var responseHttp = await repository.Post("/api/Owners", Owner);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

navigationManager.NavigateTo("/Owners");

}

private void Return()

{

navigationManager.NavigateTo("/Owners");

}

}

1. Agregamos el boton de crear país en **OwnersIndex**:

<h3>Países</h3>

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

<GenericList MyList="Owners">

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

@inject SweetAlertService swal

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

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>País:</label>

<div>

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

<ValidationMessage For="@(() => Owner.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(Owner);

}

[EditorRequired]

[Parameter]

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

[EditorRequired]

[Parameter]

public EventCallback OnValidSubmit { get; set; }

[EditorRequired]

[Parameter]

public EventCallback ReturnAction { get; set; }

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

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited)

{

return;

}

if (FormPostedSuccessfully)

{

return;

}

var result = await swal.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true

});

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

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

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

@page "/Owners/create"

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService swal

<h3>Crear País</h3>

<OwnerForm @ref="OwnerForm" Owner="Owner" OnValidSubmit="Create" ReturnAction="Return" />

@code {

private Owner Owner = new();

private OwnerForm? OwnerForm;

private async Task Create()

{

var httpResponse = await repository.Post("api/Owners", Owner);

if (httpResponse.Error)

{

var mensajeError = await httpResponse.GetErrorMessageAsync();

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

}

else

{

OwnerForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("Owners");

}

}

private void Return()

{

navigationManager.NavigateTo("Owners");

}

}

1. Probamos la creación de países por interfaz y luego hacemos nuestro **commit**. **Asegúrate de presionar Ctrl + F5, para que te tome los cambios**.
2. Ahora creamos el componente **OwnerEdit**:

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

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService swal

<h3>Editar País</h3>

@if (Owner is null)

{

<p>Cargando...</p>

}

else

{

<OwnerForm @ref="OwnerForm" Owner="Owner" OnValidSubmit="Edit" ReturnAction="Return" />

}

@code {

private Owner? Owner;

private OwnerForm? OwnerForm;

[Parameter]

public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

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

if (responseHTTP.Error)

{

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

{

navigationManager.NavigateTo("Owners");

}

else

{

var messageError = await responseHTTP.GetErrorMessage();

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

}

}

else

{

Owner = responseHTTP.Response;

}

}

private async Task Edit()

{

var responseHTTP = await repository.Put("api/Owners", Owner);

if (responseHTTP.Error)

{

var mensajeError = await responseHTTP.GetErrorMessage();

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

}

else

{

OwnerForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("Owners");

}

}

private void Return()

{

navigationManager.NavigateTo("Owners");

}

}

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

@page "/Owners"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService swal

<h3>Owners</h3>

<div class="mb-3">

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

</div>

<GenericList MyList="Owners">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var Owner in Owners!)

{

<tr>

<td>

@Owner.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

@code {

public List<Owner>? Owners { get; set; }

protected async override Task OnInitializedAsync()

{

await Load();

}

private async Task Load()

{

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

Owners = responseHppt.Response!;

}

private async Task Delete(Owner Owner)

{

var result = await swal.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

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

if (confirm)

{

return;

}

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

if (responseHTTP.Error)

{

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

{

navigationManager.NavigateTo("/");

}

else

{

var mensajeError = await responseHTTP.GetErrorMessageAsync();

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

}

}

else

{

await Load();

}

}

}

## Creando las tablas de usuarios

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

namespace Veterinary.Shared.Enums

{

public enum UserType

{

Admin,

User

}

}

1. En el proyecto **Shared** instalar el nuget **Microsoft.AspNetCore.Identity.EntityFrameworkCore** última versión (hoy es 7.0.11)
2. En el proyecto **Shared** en la carpeta **Entities**, crear la entidad **User**:

using Microsoft.AspNetCore.Identity;

using Veterinary.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class User : IdentityUser

{

[Display(Name = "Documento")]

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

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

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

[Display(Name = "Nombres")]

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

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

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

[Display(Name = "Apellidos")]

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

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

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

[Display(Name = "Dirección")]

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

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

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

[Display(Name = "Foto")]

public string? Photo { get; set; }

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

public UserType UserType { get; set; }

public City? City { get; set; }

[Display(Name = "Ciudad")]

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

public int CityId { get; set; }

[Display(Name = "Usuario")]

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

}

}

1. En el proyecto **API** instalar el nugget **Microsoft.AspNetCore.Identity.EntityFrameworkCore** la última versión(hoy es la 7.0.11)
2. Modificar el **DataContext**:

public class DataContext : IdentityDbContext<User>

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

using Microsoft.AspNetCore.Identity;

using Veterinary.Shared.Entities;

namespace Veterinary.API.Helpers

{

public interface IUserHelper

{

Task<User> GetUserAsync(string email);

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

Task CheckRoleAsync(string roleName);

Task AddUserToRoleAsync(User user, string roleName);

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

}

}

1. Luego hacemos la implementación de dicha interfaz:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Veterinary.API.Data;

using Veterinary.Shared.Entities;

namespace Veterinary.API.Helpers

{

public class UserHelper : IUserHelper

{

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

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

{

\_context = context;

\_userManager = userManager;

\_roleManager = roleManager;

}

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

{

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

}

public async Task AddUserToRoleAsync(User user, string roleName)

{

await \_userManager.AddToRoleAsync(user, roleName);

}

public async Task CheckRoleAsync(string roleName)

{

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

if (!roleExists)

{

await \_roleManager.CreateAsync(new IdentityRole

{

Name = roleName

});

}

}

public async Task<User> GetUserAsync(string email)

{

return await \_context.Users

.Include(u => u.City)

.ThenInclude(c => c.State)

.ThenInclude(s => s.Country)

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

}

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

{

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

}

}

}

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

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

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

{

x.User.RequireUniqueEmail = true;

x.Password.RequireDigit = false;

x.Password.RequiredUniqueChars = 0;

x.Password.RequireLowercase = false;

x.Password.RequireNonAlphanumeric = false;

x.Password.RequireUppercase = false;

})

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

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

var app = builder.Build();

SeedData(app);

void SeedData(WebApplication app)

{

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

using (IServiceScope? scope = scopedFactory!.CreateScope())

{

SeedDb? service = scope.ServiceProvider.GetService<SeedDb>();

service!.SeedAsync().Wait();

}

}

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

## Seguridad desde el backend(API)

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

public async Task<User> GetUserAsync(string email)

{

var user = await \_context.Users

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

return user!;

}

1. Agregamos el paquete **Microsoft.AspNetCore.Authentication.JwtBearer** al proyecto **API**, versión 7.0.11.
2. Creamos el parámetro **jwtKey** en el appsettings del proyecto **API** (cualquier cosa, entre más larga mejor):

"AllowedHosts": "\*",

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

}

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

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

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

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

{

ValidateIssuer = false,

ValidateAudience = false,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

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

ClockSkew = TimeSpan.Zero

});

var app = builder.Build();

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

using Veterinary.Shared.Entities;

using System.ComponentModel.DataAnnotations;

using System.Xml.Linq;

namespace Veterinary.Shared.DTOs

{

public class UserDTO : User

{

[DataType(DataType.Password)]

[Display(Name = "Contraseña")]

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

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

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

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

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

[DataType(DataType.Password)]

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

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

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

}

}

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

using Veterinary.Shared.Entities;

namespace Veterinary.Shared.DTOs

{

public class TokenDTO

{

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

public DateTime Expiration { get; set; }

}

}

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

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.DTOs

{

public class LoginDTO

{

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

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

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

[Display(Name = "Contraseña")]

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

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

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

}

}

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

Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();

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

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly SignInManager<User> \_signInManager;

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

{

\_context = context;

\_userManager = userManager;

\_roleManager = roleManager;

\_signInManager = signInManager;

}

…

public async Task<SignInResult> LoginAsync(LoginDTO model)

{

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

}

public async Task LogoutAsync()

{

await \_signInManager.SignOutAsync();

}

1. Creamos el **AccountsController**:

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using Veterinary.API.Helpers;

using Veterinary.Shared.DTOs;

using Veterinary.Shared.Entities;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace Veterinary.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)

};

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

};

}

}

}

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

## Habilitando tokens en swagger

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

builder.Services.AddSwaggerGen();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "Veterinary 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=DefaultConnection"));

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

3. Creamos el **SeedDb** en la carpeta Data del proyecto API

public class SeedDb

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

public SeedDb(DataContext context, IUserHelper userHelper)

{

\_context = context;

\_userHelper = userHelper;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckRolesAsync();

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

}

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

{

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

if (user == null)

{

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

UserType = userType,

};

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

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

}

return user;

}

private async Task CheckRolesAsync()

{

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

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

}

1. Inyectamos el servicio del SeedDb en la clase Program del API

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();

}

}

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

1. Corremos los siguientes comandos:

PM> drop-database

PM> add-migration Users

PM> update-database

1. Luego le colocamos autorización a **OwnersController** y probamos si funciona el token

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]