# Índice

Contenido

[Índice 1](#_Toc167316664)

[Modelo de datos 3](#_Toc167316665)

[Arquitectura del proyecto 3](#_Toc167316666)

[Crear la base de datos con EF 6](#_Toc167316667)

[Clase DataContext 8](#_Toc167316668)

[Appsettings 9](#_Toc167316669)

[Inyección de dependencias Servicio SQlServer 9](#_Toc167316670)

[Ejecutar proyectos de forma múltiple 10](#_Toc167316671)

[Migración de entidades hacia la base de datos SQL SERVER 10](#_Toc167316672)

[Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT. 11](#_Toc167316673)

[Creando los primeros métodos en el primer controlador 13](#_Toc167316674)

[Creando todas las entidades y sus relaciones 15](#_Toc167316675)

[Creando nuestros primeros componentes en Blazor 21](#_Toc167316676)

[Implementando el patrón repositorio 21](#_Toc167316677)

[Creando componente Generic List 26](#_Toc167316678)

[Creando El Index de la página Owners 28](#_Toc167316679)

[Create, Edit y Delete Owners 34](#_Toc167316680)

[Creando las tablas de usuarios 48](#_Toc167316681)

[Creamos un alimentador de la base de datos SeedDb: 54](#_Toc167316682)

[Creando sistema de seguridad 58](#_Toc167316683)

[Seguridad desde el backend(API) 66](#_Toc167316684)

[Implementando el registro de usuarios, login & logout 74](#_Toc167316685)

[Habilitando tokens en swagger 86](#_Toc167316686)

[Mejorando un poco la interfaz de usuario 89](#_Toc167316687)

[Cómo modificar los íconos de NavMenu: 90](#_Toc167316688)

[´Almacenando la foto del usuario 97](#_Toc167316689)

[Ahora crearemos un Blob Storage en Azure llamado StoreG1 102](#_Toc167316690)

[Editando el usuario 110](#_Toc167316691)

[Cambiando password del usuario 116](#_Toc167316692)

[Confirmar el registro de usuarios 120](#_Toc167316693)

[Reenviar correo de confirmación 134](#_Toc167316694)

[Actualización de la foto del usuario luego de editar usuario 140](#_Toc167316695)

[Recuperación de contraseña 142](#_Toc167316696)

[Implementación de ventanas modales 162](#_Toc167316697)

[Administrar usuarios y crear nuevos administradores 166](#_Toc167316698)

# Modelo de datos

Vamos a crear un sencillo sistema de información para una Veterinaria, 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 8 API

Blazor WebAssembly Application

Library Class



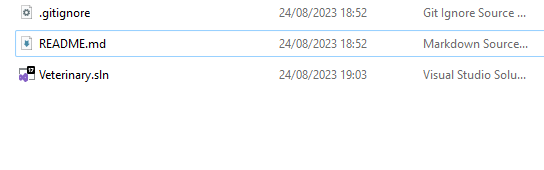
Crear proyecto

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)



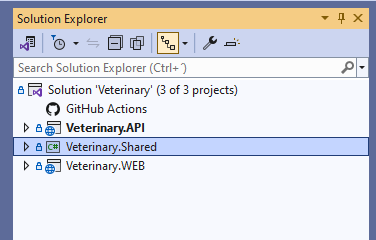
* Sobre el ícono Veterinary.sln del explorador de soluciones oprimimos click derecho y presionamos Open
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Class Library**, llamado **Veterinary.Shared**

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

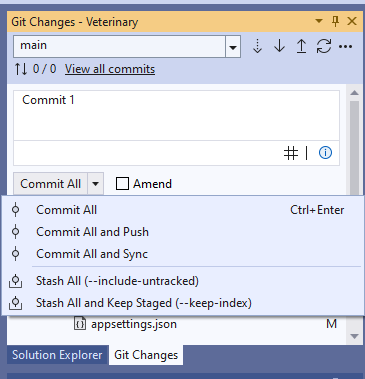
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **.NET MAUI App**, llamado **Veterinary.Mobile**. Ubicación: C:\Projects/Veterinary

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**:

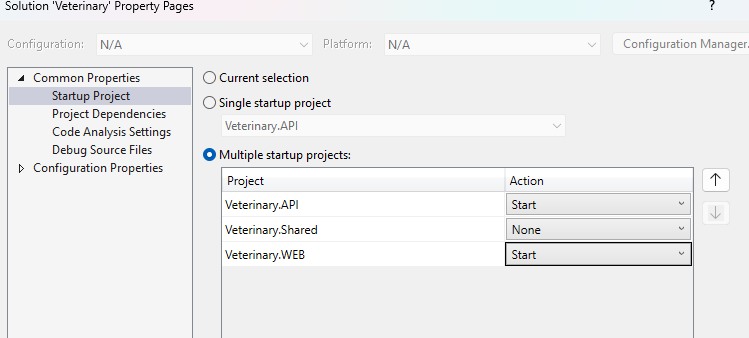
builder.Services.AddSwaggerGen();

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);

}

Creando todas las entidades y sus relaciones

1. Add the entity **PetType**:

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class PetType

{

public int Id { get; set; }

[Display(Name = "Pet Type")]

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

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

public string Name { get; set; }

}

}

1. Add the entity **ServiceType**:

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class ServiceType

{

public int Id { get; set; }

[Display(Name = "Service Type")]

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

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

public string Name { get; set; }

}

}

1. Add the entity **History**:

using System;

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class History

{

public int Id { get; set; }

public ServiceType ServiceType { get; set; }

[Display(Name = "Description\*")]

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

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

public string Description { get; set; }

[Display(Name = "Date\*")]

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

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd HH:mm}", ApplyFormatInEditMode = true)]

public DateTime Date { get; set; }

public string Remarks { get; set; }

public Pet Pet { get; set; }

[Display(Name = "Date\*")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd HH:mm}", ApplyFormatInEditMode = true)]

public DateTime DateLocal => Date.ToLocalTime();

}

}

1. Add the entity **Pet**:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class Pet

{

public int Id { get; set; }

[Display(Name = "Name")]

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

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

public string Name { get; set; }

[Display(Name = "Image")]

public string ImageUrl { get; set; }

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

public string Race { get; set; }

public Owner Owner { get; set; }

public PetType PetType { get; set; }

[Display(Name = "Born")]

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

[DataType(DataType.DateTime)]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd}", ApplyFormatInEditMode = true)]

public DateTime Born { get; set; }

public string Remarks { get; set; }

public ICollection<History> Histories { get; set; }

//TODO: replace the correct URL for the image

public string ImageFullPath => string.IsNullOrEmpty(ImageUrl)

? null

: $"https://TDB.azurewebsites.net{ImageUrl.Substring(1)}";

}

}

1. Add the entity **Agenda**:

using System;

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.Entities

{

public class Agenda

{

public int Id { get; set; }

[Display(Name = "Date")]

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

[DataType(DataType.DateTime)]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd H:mm tt}", ApplyFormatInEditMode = true)]

public DateTime Date { get; set; }

public Owner Owner { get; set; }

public Pet Pet { get; set; }

public string Remarks { get; set; }

[Display(Name = "Is Available?")]

public bool IsAvailable { get; set; }

}

}

1. Update the **DataContext**:

using Microsoft.EntityFrameworkCore;

using Veterinary.Shared.Entities;

namespace Veterinary.WEB.Data

{

public class DataContext : DbContext

{

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

{

}

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

public DbSet<PetType> PetTypes { get; set; }

public DbSet<Pet> Pets { get; set; }

public DbSet<ServiceType> ServiceTypes { get; set; }

public DbSet<History> Histories { get; set; }

public DbSet<Agenda> Agendas { get; set; }

}

}

1. Save all and run this commands:

PM> add-migration CompleteDB

PM> update-database

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 Owneres 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>> GetAsync<T>(string url);

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

Task<HttpResponseWrapper<TResponse>> PostAsync<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>> GetAsync<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>> PostAsync<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>> PostAsync<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

@page "/owners"

@using Veterinary.Shared.Entities

@using Veterinary.WEB.Repositories

@using Veterinary.WEB.Shared

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-person" /> Owners

<a class="btn btn-primary btn-sm float-end" href="/owners/create"><i class="bi bi-plus-circle"/>New Owner</a>

</span>

</div>

<div class="card-body">

<GenericList MyList="Owners">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Owners!)

{

<tr>

<td>

<a href="/owners/edit/@item.Id" class="btn btn-sm btn-warning"><i class="bi bi-pencil-square" /> Editar</a>

<button @onclick=@(() => DeleteAsycn(item)) class="btn btn-sm btn-danger"><i class="bi bi-trash-fill"/> Borrar</button>

</td>

<td>

@item.Document

</td>

<td>

@item.FirstName

</td>

<td>

@item.LastName

</td>

<td>

@item.Address

</td>

<td>

@item.FixedPhone

</td>

<td>

@item.CellPhone

</td>

<td>

@item.Address

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

</div>

</div>

@code {

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

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Owners = responseHttp.Response;

}

private async Task DeleteAsycn(Owner owner)

{

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = $"¿Estas seguro de querer borrar el owner: {owner.FirstName}?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

});

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

if (confirm)

{

return;

}

var responseHttp = await repository.DeleteAsync<Owner>($"api/owners/{owner.Id}");

if (responseHttp.Error)

{

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

{

navigationManager.NavigateTo("owners");

}

else

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

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

}

return;

}

await LoadAsync();

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

}

\*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">

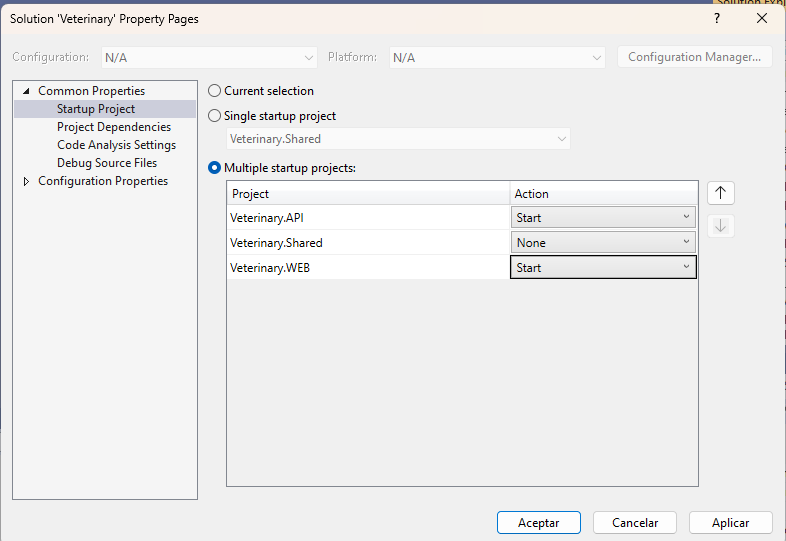
<span class="oi oi-list-rich" aria-hidden="true"></span> Owners

</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.

# Create, Edit y Delete 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>> DeleteAsync(string url);

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

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

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

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

{

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

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

}

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

{

var messageJSON = JsonSerializer.Serialize(model);

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

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

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

}

public async Task<HttpResponseWrapper<TResponse>> PutAsync<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 CurrieTechnologies.Razor.SweetAlert2

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

@inject SweetAlertService swal

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

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>Document:</label>

<div>

<InputText width="10" @bind-Value="Owner.Document"></InputText>

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

</div>

</div>

<div class="mb-3">

<label>Nombres:</label>

<div>

<InputText width="10" @bind-Value="@Owner.FirstName"></InputText>

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

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText width="10" @bind-Value="Owner.LastName"></InputText>

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

</div>

</div>

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width="10" @bind-Value="Owner.Email"></InputText>

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

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText width="10" @bind-Value="Owner.FixedPhone"></InputText>

<ValidationMessage For="@(() => Owner.FixedPhone)" />

</div>

</div>

<div class="mb-3">

<label>Móvil:</label>

<div>

<InputText width="10" @bind-Value="Owner.CellPhone"></InputText>

<ValidationMessage For="@(() => Owner.CellPhone)" />

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText width="10" @bind-Value="Owner.Address"></InputText>

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

</div>

</div>

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

<button class="btn btn-info" type="reset"><i class="bi bi-x-circle"/>Limpiar</button>

<button class="btn btn-success" onclick="ReturnAction"><i class="bi bi-arrow-left" />Regresar</button>

</EditForm>

@code {

private EditContext editContext = null!;

protected override void OnInitialized()

{

editContext = new(Owner);

}

[Parameter]

[EditorRequired]

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

[Parameter]

[EditorRequired]

public EventCallback OnValidSubmit { get; set; }

[EditorRequired]

[Parameter]

public EventCallback ReturnAction { get; set; }

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

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formularioFueEditado = editContext.IsModified();

if (!formularioFueEditado)

{

return;

}

if (FormularioPosteadoConExito)

{

return;

}

var resultado = await swal.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true

});

var confirmado = !string.IsNullOrEmpty(resultado.Value);

if (confirmado)

{

return;

}

context.PreventNavigation();

}

}

}

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

@page "/owners/create"

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService sweetAlertService

<h3>Crear</h3>

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

@code {

private Owner owner = new Owner();

private OwnerForm ownerForm;

private async Task Create()

{

var responseHttp = await repository.PostAsync("/api/owners", owner);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

ownerForm!.FormularioPosteadoConExito = true;

navigationManager.NavigateTo("/owners");

}

}

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

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

@using System.Net

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService sweetAlertService

<h3>Editar</h3>

@if (owner is null)

{

<p>Cargando...</p>

}

else

{

<OwnerForm @ref="ownerForm" Owner="owner" OnValidSubmit="Editar" ReturnAction="Return"/>

}

@code {

[Parameter]

public int Id { get; set; }

private Owner owner;

private OwnerForm ownerForm;

protected override async Task OnParametersSetAsync()

{

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

if (responseHttp.Error)

{

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

{

navigationManager.NavigateTo("/owners");

}

else

{

var messsage = await responseHttp.GetErrorMessageAsync();

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

}

}

else

{

owner = responseHttp.Response;

}

}

private async Task Editar()

{

var responseHttp = await repository.PutAsync("/api/owners", owner);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

ownerForm!.FormularioPosteadoConExito = true;

navigationManager.NavigateTo("/owners");

}

}

## 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 8.0.5)
2. En el proyecto **Shared** en la carpeta **Entities**, crear la entidad **User**:

using Microsoft.AspNetCore.Identity;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Veterinary.Shared.Enums;

namespace Veterinary.Shared.Entities

{

public class User:IdentityUser

{

[MaxLength(20, ErrorMessage = "No se permiten más de 20 dígitos")]

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

public string Document { get; set; }

[MaxLength(50, ErrorMessage = "No se permiten más de 50 Caracteres")]

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

public string FirstName { get; set; }

[MaxLength(50, ErrorMessage = "No se permiten más de 50 Caracteres")]

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

public string LastName { get; set; }

public string Address { get; set; }

public UserType UserType { get; set; }

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

}

}

1. En el proyecto **API** instalar el nugget **Microsoft.AspNetCore.Identity.EntityFrameworkCore** la última versión(hoy es la 8.0.5)
2. Modificar el **DataContext**:

public class DataContext : IdentityDbContext<User>

1. Creamos la carpeta Helpers en el proyecto API y dentro creamos la interfaz **IUserHelper**

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. Creamos la clase userHelper y luego hacemos la implementación de dicha interfaz:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using System.Net.Http.Headers;

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)

{

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

if (!roleExists)

{

await \_roleManager.CreateAsync(new IdentityRole

{

Name = roleName

});

}

}

public async Task<User> GetUserAsync(string email)

{

var user = await \_context.Users

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

return user!;

}

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 = 6;

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();

## Creamos un alimentador de la base de datos SeedDb:

using Veterinary.Shared.Entities;

using Veterinary.API.Helpers;

using Veterinary.Shared.Enums;

using Azure.Identity;

using Azure;

using System.Diagnostics.Metrics;

using Microsoft.EntityFrameworkCore;

namespace Veterinary.API.Data

{

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 CheckPetTypesAsync();

await CheckRolesAsync();

await CheckUserAsync("1010", "OAP", "Admin", "oap@yopmail.com","305232456", "Lo que sea", UserType.Admin);

}

private async Task CheckRolesAsync()

{

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

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

}

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

{

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

if (user == null)

{

user = new User

{

Document = document,

FirstName = firstName,

LastName = lastName,

Email = email,

PhoneNumber = phone,

UserName = email,

Address = address,

UserType = userType,

};

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

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

}

return user;

}

private async Task CheckPetTypesAsync()

{

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

{

\_context.PetTypes.Add(new PetType { Name = "Dog" });

\_context.PetTypes.Add(new PetType { Name = "Cat" });

\_context.PetTypes.Add(new PetType { Name = "Hamster" });

\_context.PetTypes.Add(new PetType { Name = "Rabbit" });

}

await \_context.SaveChangesAsync();

}

}

}

1. Corremos los siguientes comandos:

PM> drop-database

PM> add-migration Users

PM> update-database

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

## Creando sistema de seguridad

1. Al proyecto **WEB** agregamos el paquete: **Microsoft.AspNetCore.Components.WebAssembly.Authentication** versión 8.0.5
2. Y agregamos el paquete **System.IdentityModel.Token** 7.5.2
3. Agregamos este using en el **\_Imports,** para que sea global en el proyecto Web:

@using Microsoft.AspNetCore.Components.Authorization

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

using Microsoft.AspNetCore.Components.Authorization;

using System.Security.Claims;

namespace VeterinaryWEB.Auth

{

public class AuthenticationProviderTest : AuthenticationStateProvider

{

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

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

}

}

}

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

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

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

builder.Services.AddSweetAlert2();

builder.Services.AddAuthorizationCore();

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

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

@using Microsoft.AspNetCore.Components.Authorization

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

<Found Context="routeData">

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

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

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>No encontrado</PageTitle>

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

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

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

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

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

await Task.Delay(3000);

var anonimous = new ClaimsIdentity();

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

}

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

@using Microsoft.AspNetCore.Components.Authorization

<Found Context="routeData">

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

<Authorizing>

<p>Autorizando...</p>

</Authorizing>

</AuthorizeRouteView>

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

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>No encontrado</PageTitle>

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

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

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

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

@page "/"

@using Microsoft.AspNetCore.Components.Authorization

<PageTitle>Index</PageTitle>

<AuthorizeView>

<p>Estas autenticado</p>

</AuthorizeView>

<h1>Hello, world!</h1>

Welcome to your new app.

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

1. Modificamos el **AuthenticationProviderTest**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

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

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

}

1. Cambiamos el **Home.razor**.

<AuthorizeView>

<Authorized>

<p>Estas autenticado</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

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

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

var oapUser = new ClaimsIdentity(new List<Claim>

{

new Claim("FirstName", "Orlando"),

new Claim("LastName", "A"),

new Claim(ClaimTypes.Name, "onurisitm@gmail.com")

},

authenticationType: "test");

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

}

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

<AuthorizeView>

<Authorized>

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

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

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

<@page "/"

<PageTitle>Home</PageTitle>

<AuthorizeView Roles="Admin">

<Authorized>

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

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

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

var oapUser = new ClaimsIdentity(new List<Claim>

{

new Claim("FirstName", "Orlando"),

new Claim("LastName", "A),

new Claim(ClaimTypes.Name, "onurisitm@gmail.com"),

new Claim(ClaimTypes.Role, "Admin")

},

authenticationType: "test");

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

@using Microsoft.AspNetCore.Components.Authorization

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

<nav class="flex-column">

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

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

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

</NavLink>

</div>

<AuthorizeView Roles="Admin">

<Authorized>

<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>

</Authorized>

</AuthorizeView>

</nav>

</div>

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

@using Microsoft.AspNetCore.Authorization;

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

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

@using Microsoft.AspNetCore.Components.Authorization

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

<Found Context="routeData">

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

<Authorizing>

<p>Autorizando...</p>

</Authorizing>

<NotAuthorized>

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

</NotAuthorized>

</AuthorizeRouteView>

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

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>No encontrado</PageTitle>

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

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

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

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

## Seguridad desde el backend(API)

1. Agregamos el paquete **Microsoft.AspNetCore.Authentication.JwtBearer** al proyecto **API**, versión 8.0.5
2. Creamos el parámetro **jwtKey** en el appsettings del proyecto **API** :

"jwtKey": "darlkjerlpiouEWRKJERWHERKJerlkdjDFFDOIKFDPOfncs!!je%rowiuj()refdlkdfsjpoe5riwuer907549863472894576rejfhdio84723732e1w098743520342589",

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"runtimeOptions": {

"configProperties": {

"System.Globalization.Invariant": false

}

}

}

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

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

builder.Services.AddTransient<SeedDb>();

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** creamos la carpeta **DTOs** y dentro 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

};

}

}

}

1. Luego le colocamos autorización a todos los controladores

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

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

## Implementando el registro de usuarios, login & logout

1. En el proyecto **WEB** creamos la carpeta **Helpers** y dentro creamos el **IJSRuntimeExtensionMethods**:

using Microsoft.JSInterop;

namespace Veterinary.WEB.Helpers

{

public static class IJSRuntimeExtensionMethods

{

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

{

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

}

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

{

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

}

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

{

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

}

}

}

1. En el proyecto **WEB** creamos la carpeta **Services** y dentro creamos la interface **ILoginService**:

namespace Veterinary.WEB.Auth

{

public interface ILoginService

{

Task LoginAsync(string token);

Task LogoutAsync();

}

}

1. En el proyecto **WEB** dentro dela carpeta **AuthenticationProviders** creamos la clase **AuthenticationProviderJWT**:

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.JSInterop;

using Veterinary.WEB.Helpers;

using System.IdentityModel.Tokens.Jwt;

using System.Net.Http.Headers;

using System.Security.Claims;

namespace Veterinary.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, "jwtkey")));

}

private IEnumerable<Claim> ParseClaimsFromJWT(string token)

{

var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();

var unserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

return unserializedToken.Claims;

}

public async Task LoginAsync(string token)

{

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

var authState = BuildAuthenticationState(token);

NotifyAuthenticationStateChanged(Task.FromResult(authState));

}

public async Task LogoutAsync()

{

await \_jSRuntime.RemoveLocalStorage(\_tokenKey);

\_httpClient.DefaultRequestHeaders.Authorization = null;

NotifyAuthenticationStateChanged(Task.FromResult(\_anonimous));

}

}

}

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

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

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

builder.Services.AddSweetAlert2();

builder.Services.AddAuthorizationCore();

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

builder.Services.AddScoped<AuthenticationProviderJWT>();

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

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

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

<AuthorizeView>

<Authorized>

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

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

</Authorized>

<NotAuthorized>

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

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

</NotAuthorized>

</AuthorizeView>

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

@inherits LayoutComponentBase

<div class="page">

<div class="sidebar">

<NavMenu />

</div>

<main>

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

<AuthLinks/>

<a href="https://learn.microsoft.com/aspnet/core/" target="\_blank">About</a>

</div>

<article class="content px-4">

@Body

</article>

</main>

</div>

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

@page "/Register"

@using Veterinary.Shared.DTOs;

@using Veterinary.Shared.Enums;

@using Veterinary.WEB.AuthenticationProviders;

@using Veterinary.WEB.Services

@using Veterinary.Shared

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@inject ILoginService loginService

<h3>Registrar Nuevo Usuario</h3>

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

<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="bi bi-person-add" /> Registrar</button>

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombres:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.FirstName" />

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

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.LastName" />

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

</div>

</div>

<div class="mb-3">

<label>Documento:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.Document" />

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

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.PhoneNumber" />

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

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.Address" />

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

</div>

</div>

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.Email" />

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

</div>

</div>

<div class="mb-3">

<label>Foto:</label>

<div>

<InputText width="10" @bind-Value="@userDTO.Photo" />

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

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" width="10" @bind-Value="@userDTO.Password" />

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

</div>

</div>

<div class="mb-3">

<label>Confirmación de contraseña:</label>

<div>

<InputText type="password" width="10" @bind-Value="@userDTO.PasswordConfirm" />

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

</div>

</div>

</div>

</div>

</div>

</div>

</EditForm>

@code {

private UserDTO userDTO = new();

private async Task CreateUserAsync()

{

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

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

navigationManager.NavigateTo("/");

}

}

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

@page "/Login"

@using Veterinary.Shared.DTOs;

@using Veterinary.WEB.AuthenticationProviders;

@using Veterinary.WEB.Services

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@inject ILoginService loginService

<h3>Iniciar Sesión</h3>

<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="bi bi-box-arrow-in-right" /> Iniciar Sesión</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width="10" @bind-Value="@loginDTO.Email" />

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

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" width="10" @bind-Value="@loginDTO.Password" />

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

</div>

</div>

</div>

</div>

</EditForm>

@code {

private LoginDTO loginDTO = new();

private async Task LoginAsync()

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

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

navigationManager.NavigateTo("/");

}

}

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

@page "/logout"

@using Veterinary.WEB.Auth;

@inject ILoginService loginService

@inject NavigationManager navigationManager

<p>Cerrando sesión...</p>

@code {

protected override async Task OnInitializedAsync()

{

await loginService.LogoutAsync();

navigationManager.NavigateTo("/");

}

}

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

## Habilitando tokens en swagger

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

builder.Services.AddSwaggerGen();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "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**.

## Mejorando un poco la interfaz de usuario

1. **Primero vamos a agregar estas líneas al final de nuestro Estilos.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)

}

}

## Cómo modificar los íconos de NavMenu:

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

<link href="css/Estilos.css" rel="stylesheet" />

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.1/font/bootstrap-icons.css" integrity="sha384-4LISF5TTJX/fLmGSxO53rV4miRxdg84mZsxmO8Rx5jGtp/LbrixFETvWa5a6sESd" crossorigin="anonymous">

</head>

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

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

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

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

</NavLink>

</div>

1. Esta es la forma de usar cualquier ícono de esta librería, pero al probar, visualmente no se ve bien, porque se ve el ícono más pequeño que el de inicio y se ve desalineado (no olvides presionar el Ctrl + F5, luego de correr para que te refresque el uso del CDN):
2. Esto es porque esta página tiene sus propios estilos **NavMenu.razor.css**, y debemos de incluir un nuevo estilo con la definición SVG del ícono. Para esto vamos tomar la definición SVG del ícono tomada de la misma página de definición del ícono <https://icons.getbootstrap.com/icons/globe-americas/> y lo agregamos a la hoja de estilos, adicionalmente aprovechemos y separemos el código C# del menú, para seguir las buenas prácticas:

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

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

}

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

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

}

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

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

}

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

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

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

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

</NavLink>

</div>

1. Probamos de nuevo y ahora vemos que visualmente se ve mucho mejor:
2. Esto lo debemos hacer para cualquier ícono que tengas en el menú. Ahora ustedes deben escoger los íconos adecuados para el menú de sus proyectos.
3. Este es un ejemplo de cómo puede quedar la página de **Register**:

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

<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 width=“10”@bind-Value="@userDTO.FirstName" />

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

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText width=“10”@bind-Value="@userDTO.LastName" />

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

</div>

</div>

<div class="mb-3">

<label>Documento:</label>

<div>

<InputText width=“10”@bind-Value="@userDTO.Document" />

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

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText width=“10”@bind-Value="@userDTO.PhoneNumber" />

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

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText width=“10”@bind-Value="@userDTO.Address" />

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

</div>

</div>

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width=“10”@bind-Value="@userDTO.Email" />

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

</div>

</div>

</div>

<div class="col-6">

<div class="mb-3">

<label>Owner:</label>

<div>

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

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

@if (Owners is not null)

{

@foreach (var Owner in Owners)

{

<option value="@Owner.Id">@Owner.FirstName</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Estado/Departamento:</label>

<div class="mb-3">

<label>Foto:</label>

<div>

<InputText width=“10”@bind-Value="@userDTO.Photo" />

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

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" width=“10”@bind-Value="@userDTO.Password" />

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

</div>

</div>

<div class="mb-3">

<label>Confirmación de contraseña:</label>

<div>

<InputText type="password" width=“10”@bind-Value="@userDTO.PasswordConfirm" />

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

</div>

</div>

</div>

</div>

</div>

</div>

</EditForm>

1. Y este es un ejemplo de cómo 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 width=“10”@bind-Value="@loginDTO.Email" />

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

</div>

</div>

<div class="card-body">

<label>Contraseña:</label>

<div>

<InputText type="password" width=“10”@bind-Value="@loginDTO.Password" />

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

</div>

</div>

</div>

</div>

</EditForm>

</div>

</div>

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

## ´Almacenando la foto del usuario

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

<div>

<label>@Label</label>

<div>

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

</div>

</div>

<div>

@if (imageBase64 is not null)

{

<div>

<div style="margin: 10px">

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

</div>

</div>

}

@if (ImageURL is not null)

{

<div>

<div style="margin: 10px">

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

</div>

</div>

}

</div>

@code {

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

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

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

private string? imageBase64;

async Task OnChange(InputFileChangeEventArgs e)

{

var imagenes = e.GetMultipleFiles();

foreach (var imagen in imagenes)

{

var arrBytes = new byte[imagen.Size];

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

imageBase64 = Convert.ToBase64String(arrBytes);

ImageURL = null;

await ImageSelected.InvokeAsync(imageBase64);

StateHasChanged();

}

}

}

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

@using Veterinary.WEB.Shared;

…

<div class="mb-3">

<label>Confirmación de contraseña:</label>

<div>

<InputText type="password" width=“10”@bind-Value="@userDTO.PasswordConfirm" />

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

</div>

</div>

<div class="mb-3">

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

</div>

</div>

</div>

</div>

</div>

</EditForm>

@code {

private UserDTO userDTO = new();

private string imageUrl;

protected override async Task OnInitializedAsync()

{

await CreateUserAsync();

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

{

imageUrl = userDTO.Photo;

userDTO.Photo = null;

}

}

private async Task CreateUserAsync()

{

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

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

navigationManager.NavigateTo("/");

}

private void ImageSelected(string imagenBase64)

{

userDTO.Photo = imagenBase64;

imageUrl = null;

}

}

1. Probamos lo que llevamos hasta el momento.

Ahora crearemos un Blob Storage en Azure llamado StoreG1

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Y luego creamos el contenedor para **users**(los nombres de contenedores debe ser en minúsculas)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

DefaultEndpointsProtocol=https;AccountName=storesg1;AccountKey=mYA2TL+11JwjI6CqTFUM76SWfUrIqlqpKqn4thNeeQJ6lN5gVawTEBl0uQx3g5RTpSVKsT5I3J4V+AStDVAljg==;EndpointSuffix=core.windows.net

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

"ConnectionStrings": {

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

"AzureStorage": "DefaultEndpointsProtocol=https;AccountName=storesg1;AccountKey=mYA2TL+11JwjI6CqTFUM76SWfUrIqlqpKqn4thNeeQJ6lN5gVawTEBl0uQx3g5RTpSVKsT5I3J4V+AStDVAljg==;EndpointSuffix=core.windows.net"

},

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

namespace Veterinary.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);

}

}

}

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

1. Luego la misma carpeta Helpers del API creamos la clase que la implementa **FileStorage.cs**:

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace Veterinary.API.Helpers

{

public class FileStorage : IFileStorage

{

private readonly string connectionString;

public FileStorage(IConfiguration configuration)

{

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

}

public async Task RemoveFileAsync(string path, string containerName)

{

var client = new BlobContainerClient(connectionString, containerName);

await client.CreateIfNotExistsAsync();

var fileName = Path.GetFileName(path);

var blob = client.GetBlobClient(fileName);

await blob.DeleteIfExistsAsync();

}

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

{

var client = new BlobContainerClient(connectionString, containerName);

await client.CreateIfNotExistsAsync();

client.SetAccessPolicy(PublicAccessType.Blob);

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

var blob = client.GetBlobClient(fileName);

using (var ms = new MemoryStream(content))

{

await blob.UploadAsync(ms);

}

return blob.Uri.ToString();

}

}

}

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

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

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

1. Modificamos el **AccountsController**:

[ApiController]

[Route("/api/accounts")]

public class AccountsController : ControllerBase

{

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

private readonly IFileStorage \_fileStorage;

private readonly string \_container;

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

{

\_userHelper = userHelper;

\_configuration = configuration;

\_fileStorage = fileStorage;

\_container = "users";

}

[HttpPost("CreateUser")]

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

{

User user = model;

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

{

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

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

}

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

if (result.Succeeded)

{

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

return Ok(BuildToken(user));

}

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

}

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

<AuthorizeView>

<Authorized>

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

@if (!string.IsNullOrEmpty(photoUser))

{

<div class="mx-2">

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

</div>

}

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

</Authorized>

<NotAuthorized>

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

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

</NotAuthorized>

</AuthorizeView>

@code {

private string? photoUser;

[CascadingParameter]

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

protected async override Task OnParametersSetAsync()

{

var authenticationState = await authenticationStateTask;

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

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

if (photoClaim is not null)

{

photoUser = photoClaim.Value;

}

}

}

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

## Editando el usuario

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

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

Task<IdentityResult> UpdateUserAsync(User user);

Task<User> GetUserAsync(Guid userId);

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

public async Task<User> GetUserAsync(string email)

{

var user = await \_context.Users

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

return user!;

}

public async Task<User> GetUserAsync(Guid userId)

{

var user = await \_context.Users

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

return user!;

}

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

{

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

}

public async Task<IdentityResult> UpdateUserAsync(User user)

{

return await \_userManager.UpdateAsync(user);

}

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

[HttpPut]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> Put(User user)

{

try

{

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

{

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

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

}

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

if (currentUser == null)

{

return NotFound();

}

currentUser.Document = user.Document;

currentUser.FirstName = user.FirstName;

currentUser.LastName = user.LastName;

currentUser.Address = user.Address;

currentUser.PhoneNumber = user.PhoneNumber;

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

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

if (result.Succeeded)

{

return NoContent();

}

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

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[HttpGet]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> Get()

{

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

}

1. Modificamos el **AuthLinks**:

<Authorized>

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

@if (!string.IsNullOrEmpty(photoUser))

{

<div class="mx-2">

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

</div>

}

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

</Authorized>

1. Creamos el **EditUser.razor** en el proyecto WEB en la cartpeta **Pages** 🡪**Auth**:

@page "/EditUser"

@using CurrieTechnologies.Razor.SweetAlert2;

@using Veterinary.Shared.Entities;

@using Veterinary.WEB.Pages.Auth;

@using Veterinary.WEB.Repositories;

@using Veterinary.WEB.Services

@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 width="10" @bind-Value="@user.FirstName" />

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

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText width="10" @bind-Value="@user.LastName" />

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

</div>

</div>

<div class="mb-3">

<label>Documento:</label>

<div>

<InputText width="10" @bind-Value="@user.Document" />

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

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText width="10" @bind-Value="@user.PhoneNumber" />

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

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText width="10" @bind-Value="@user.Address" />

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

</div>

</div>

</div>

<div class="col-6">

<div class="mb-3">

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

</div>

</div>

</div>

</div>

</div>

</EditForm>

}

@code {

private User user;

private string imageUrl;

protected override async Task OnInitializedAsync()

{

await LoadUserAsyc();

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

{

imageUrl = user.Photo;

user.Photo = null;

}

}

private async Task LoadUserAsyc()

{

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

if (responseHTTP.Error)

{

if (responseHTTP.HttpResponseMessage.StatusCode == 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 SaveUserAsync()

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

navigationManager.NavigateTo("/");

}

}

1. Probamos.

## Cambiando password del usuario

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

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.DTOs

{

public class ChangePasswordDTO

{

[DataType(DataType.Password)]

[Display(Name = "Contraseña actual")]

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

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

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

[DataType(DataType.Password)]

[Display(Name = "Nueva contraseña")]

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

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

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

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

[DataType(DataType.Password)]

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

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

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

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

}

}

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

[HttpPost("changePassword")]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> ChangePasswordAsync(ChangePasswordDTO model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

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

if (user == null)

{

return NotFound();

}

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

if (!result.Succeeded)

{

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

}

return NoContent();

}

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

@page "/changePassword"

@using Veterinary.Shared.DTOs;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

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

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

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

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Contraseña actual:</label>

<div>

<InputText type="password" width="10" @bind-Value="@changePasswordDTO.CurrentPassword" />

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

</div>

</div>

<div class="mb-3">

<label>Nueva contraseña:</label>

<div>

<InputText type="password" width="10" @bind-Value="@changePasswordDTO.NewPassword" />

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

</div>

</div>

<div class="mb-3">

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

<div>

<InputText type="password" width="10" @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.PostAsync("/api/accounts/changePassword", changePasswordDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

navigationManager.NavigateTo("/editUser");

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.TopEnd,

ShowConfirmButton = true,

Timer = 5000

});

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

}

}

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

## Confirmar el registro de usuarios

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

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

{

x.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;

x.SignIn.RequireConfirmedEmail = true;

x.User.RequireUniqueEmail = true;

x.Password.RequireDigit = false;

x.Password.RequiredUniqueChars = 0;

x.Password.RequireLowercase = false;

x.Password.RequireNonAlphanumeric = false;

x.Password.RequireUppercase = false;

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

x.Lockout.MaxFailedAccessAttempts = 3;

x.Lockout.AllowedForNewUsers = true;

})

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

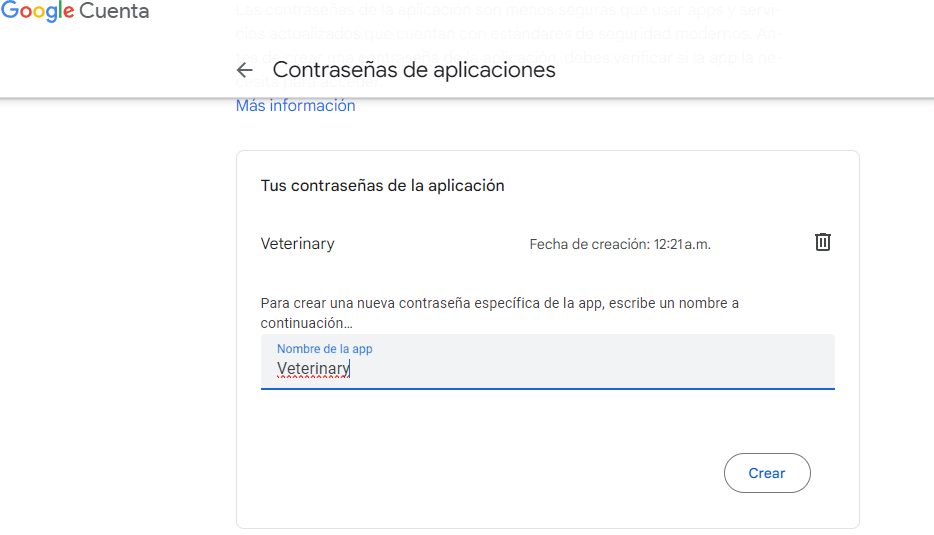
Descripción generada automáticamente

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

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ingresamos el nombre de la aplicación nuestra para generar una contraseña



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

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

"Mail": {

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

"Name": "Veterinary",

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

"Port": 587,

"Password": "nniufszzppfuzhxe"

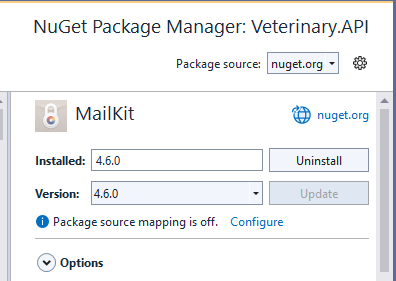
},

"UrlWEB": "localhost:8000"

}

**Nota**: Reemplazar el password entre comillas por la contraseña de aplicación generada en el paso anterior y quitar los espacios al password.

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



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

using Veterinary.Shared.Responses;

public interface IMailHelper

{

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

}

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

using MailKit.Net.Smtp;

using MimeKit;

using Veterinary.Shared.Responses;

namespace Veterinary.API.Helpers

{

public class MailHelper : IMailHelper

{

private readonly IConfiguration \_configuration;

public MailHelper(IConfiguration configuration)

{

\_configuration = configuration;

}

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

{

try

{

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

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

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

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

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

var message = new MimeMessage();

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

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

message.Subject = subject;

BodyBuilder bodyBuilder = new BodyBuilder

{

HtmlBody = body

};

message.Body = bodyBuilder.ToMessageBody();

using (var client = new SmtpClient())

{

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

client.Authenticate(from, password);

client.Send(message);

client.Disconnect(true);

}

return new Response { IsSuccess = true };

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

Result = ex

};

}

}

}

}

1. Configuramos la inyección del servicio:

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

1. Agregamos los métodos a la interface **IUserHelper**:

Task<string> GenerateEmailConfirmationTokenAsync(User user);

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

Y la implementación en la clase UserHelper:

public async Task<string> GenerateEmailConfirmationTokenAsync(User user)

{

return await \_userManager.GenerateEmailConfirmationTokenAsync(user);

}

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

{

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

}

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

(primero inyectamos el **IMailHelper**):

private readonly IMailHelper \_mailHelper;

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

{

\_userHelper = userHelper;

\_configuration = configuration;

\_fileStorage = fileStorage;

\_container = "users";

\_mailHelper = mailHelper;

}

[HttpPost("CreateUser")]

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

{

User user = model;

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

{

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

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

}

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

if (result.Succeeded)

{

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

eliminamos las siguientes líneas que están el final del método **CreateUser**:

return Ok(BuildToken(user));

}

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

}

y las reemplazamos por el siguiente código

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

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

{

userid = user.Id,

token = myToken

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

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

$"Veterinarys- Confirmación de cuenta",

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

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

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

if (response.IsSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

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

}

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

[HttpGet("ConfirmEmail")]

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

{

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

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

if (user == null)

{

return NotFound();

}

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

if (!result.Succeeded)

{

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

}

return NoContent();

}

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

[HttpPost("Login")]

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

{

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

if (result.Succeeded)

{

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

return Ok(BuildToken(user));

}

if (result.IsLockedOut)

{

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

}

if (result.IsNotAllowed)

{

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

}

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

}

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

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

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

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

{

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

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

}

1. Dentro de **Pages/Auth** creamos la página **ConfirmEmail.razor**:

@page "/api/accounts/ConfirmEmail"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

<h3>Confirmación de email</h3>

<p>Presione el botón para confirmar su cuenta</p>

<button class="btn btn-primary" @onclick="ConfirmAccountAsync">Confirmar Cuenta</button>

@code {

private string? message;

[Parameter]

[SupplyParameterFromQuery]

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

[Parameter]

[SupplyParameterFromQuery]

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

protected async Task ConfirmAccountAsync()

{

var responseHttp = await repository.Get($"/api/accounts/ConfirmEmail/?userId={UserId}&token={Token}");

if (responseHttp.Error)

{

message = await responseHttp.GetErrorMessageAsync();

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

navigationManager.NavigateTo("/");

}

else

{

await sweetAlertService.FireAsync("Confirmación", "Gracias por confirmar su email, ahora puedes ingresar al sistema.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/Login");

}

}

}

1. Borramos los usuarios de la base de datos.
2. Modificamos el alimentador de la base de datos:

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

{

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

if (user == null)

{

user = new User

{

Document = document,

FirstName = firstName,

LastName = lastName,

Email = email,

PhoneNumber = phone,

UserName = email,

Address = address,

UserType = userType,

};

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

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

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

await \_userHelper.ConfirmEmailAsync(user, token);

}

return user;

}

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

@code {

private UserDTO userDTO = new();

private string imageUrl;

[Parameter]

[SupplyParameterFromQuery]

public bool IsAdmin {get; set; }

private async Task CreateUserAsync()

{

userDTO.UserName = userDTO.Email;

if (IsAdmin)

{

userDTO.UserType = UserType.Admin;

}

else

{

userDTO.UserType = UserType.User;

}

var responseHttp = await repository.PostAsync<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. Le hemos enviado un correo electrónico con las instrucciones para activar su usuario.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/");

}

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

## Reenviar correo de confirmación

1. En **Veterinary.Shared.DTOs** creamos la clase **EmailDTO**:

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.DTOs

{

public class EmailDTO

{

[Display(Name = "Email")]

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

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

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

}

}

1. En el **API** creamos este método en el **AccountsController**:

[HttpPost("ResedToken")]

public async Task<ActionResult> ResedToken([FromBody] EmailDTO model)

{

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

if (user == null)

{

return NotFound();

}

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

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

{

userid = user.Id,

token = myToken

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

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

$"Veterinarys- Confirmación de cuenta",

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

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

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

if (response.IsSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

1. Modificamos nuestro **Login.razor**:

<div class="row">

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

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

<DataAnnotationsValidator />

<div class="card bg-light">

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

<span>

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

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

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width=“10”@bind-Value="@loginDTO.Email" />

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

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" width=“10”@bind-Value="@loginDTO.Password" />

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

</div>

</div>

</div>

<div class="card-footer">

<a class="bbtn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a>

</div>

</div>

</EditForm>

</div>

</div>

1. Dentro de **Pages/Auth** creamos el componente **ResendConfirmationEmailToken.razor**:

@page "/ResendToken"

@using Veterinary.Shared.DTOs;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

<EditForm Model="emailDTO" OnValidSubmit="ResendConfirmationEmailTokenAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-key" /> Reenviar correo de confirmación de contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-loop-square" /> Reenviar</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width="10"@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.PostAsync("/api/accounts/ResedToken", emailDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/");

}

}

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

## Actualización de la foto del usuario luego de editar usuario

1. Modificamos el método **PUT** del **AccountsController**:

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

if (result.Succeeded)

{

return Ok(BuildToken(currentUser));

}

1. Agregamos este método al **IRepository**, si ya está creado, no lo tocamos:

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

1. Y su implementación en el **Repository,** si ya está creado, no lo tocamos::

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

{

var messageJSON = JsonSerializer.Serialize(model);

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

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

if (responseHttp.IsSuccessStatusCode)

{

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

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

}

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

}

1. Modificamos el **EditUser**:

@using Veterinary.Shared.DTOs;

private async Task SaveUserAsync()

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

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

navigationManager.NavigateTo("/");

}

1. Probamos y hacemos el **Commit**.

## Recuperación de contraseña

1. Modificamos el **Login.razor**:

<div class="card-footer">

<p><a class="bbtn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a></p>

<p><a class="bbtn btn-link" href="/RecoverPassword">¿Has olvidado tu contraseña?</a></p>

</div>

1. Adicionamos en **Veterinary.Shared.DTOs** la clase **ResetPasswordDTO**:

using System.ComponentModel.DataAnnotations;

namespace Veterinary.Shared.DTOs

{

public class ResetPasswordDTO

{

[Display(Name = "Email")]

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

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

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

[DataType(DataType.Password)]

[Display(Name = "Contraseña")]

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

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

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

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

[DataType(DataType.Password)]

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

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

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

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

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

}

}

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

Task<string> GeneratePasswordResetTokenAsync(User user);

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

Y la implementación:

public async Task<string> GeneratePasswordResetTokenAsync(User user)

{

return await \_userManager.GeneratePasswordResetTokenAsync(user);

}

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

{

return await \_userManager.ResetPasswordAsync(user, token, password);

}

1. Adicionamos estos métodos al **AccountController**:

[HttpPost("RecoverPassword")]

public async Task<ActionResult> RecoverPassword([FromBody] EmailDTO model)

{

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

if (user == null)

{

return NotFound();

}

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

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

{

userid = user.Id,

token = myToken

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

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

$"Veterinary - Recuperación de contraseña",

$"<h1>Veterinary - Recuperación de contraseña</h1>" +

$"<p>Para recuperar su contraseña, por favor hacer clic 'Recuperar Contraseña':</p>" +

$"<b><a href ={tokenLink}>Recuperar Contraseña</a></b>");

if (response.IsSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

[HttpPost("ResetPassword")]

public async Task<ActionResult> ResetPassword([FromBody] ResetPasswordDTO model)

{

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

if (user == null)

{

return NotFound();

}

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

if (result.Succeeded)

{

return NoContent();

}

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

}

1. Dentro de **Pages/Auth** creamos el **RecoverPassword.razor**:

@using Veterinary.Shared.DTOs;

@page "/RecoverPassword"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

<EditForm Model="emailDTO" OnValidSubmit="SendRecoverPasswordEmailTokenAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-key" /> Enviar email para recuperación de contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-loop-square" /> Enviar</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width="10"@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.PostAsync("/api/accounts/RecoverPassword", emailDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para recuperar su contraseña.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/");

}

}

1. Dentro de **Pages/Auth** creamos el **ResetPassword.razor**:

@using Veterinary.Shared.DTOs;

@page "/api/accounts/ResetPassword"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

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

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

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText width=“10”@bind-Value="@resetPasswordDTO.Email" />

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

</div>

</div>

<div class="mb-3">

<label>Nueva contraseña:</label>

<div>

<InputText type="password" width=“10”@bind-Value="@resetPasswordDTO.Password" />

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

</div>

</div>

<div class="mb-3">

<label>Confirmar contraseña:</label>

<div>

<InputText type="password" width=“10”@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.PostAsync("/api/accounts/ResetPassword", resetPasswordDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Contraseña cambiada con éxito, ahora puede ingresar con su nueva contraseña.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/Login");

}

}

1. Probamos y hacemos el **commit**.
2. Modificamos el **SeedDb**:

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckOwnersAsync();

await CheckPetTypesAsync();

await CheckRolesAsync();

await CheckUserAsync("1010", "Orlando", "Alarcon", "oap@yopmail.com", "3001234568", "Calle 78 4424", UserType.Admin);

}

private async Task CheckPetTypesAsync()

{

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

{

\_context.PetTypes.Add(new PetType { Name = "Perro" });

\_context.PetTypes.Add(new PetType { Name = "Gato" });

\_context.PetTypes.Add(new PetType { Name = "Pájaro " });

\_context.PetTypes.Add(new PetType { Name = "Conejo" });

\_context.PetTypes.Add(new PetType { Name = "Hamster" });

\_context.PetTypes.Add(new PetType { Name = "Tortuga" });

await \_context.SaveChangesAsync();

}

}

1. En **Pages** creamos la carpeta **PetTypes** y dentro de esta agregamos el **PetTypesIndex.razor**:

@page "/PetTypes"

@using Microsoft.AspNetCore.Authorization;

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

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

@if (PetTypes is null)

{

<div class="spinner" />

}

else

{

<GenericList MyList="PetTypes">

<Body>

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-list"></i> PetTypes

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

</span>

</div>

<div class="card-body">

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

<table class="table table-striped">

<thead>

<tr>

<th>PetType</th>

<th style="width:200px"></th>

</tr>

</thead>

<tbody>

@foreach (var PetType in PetTypes)

{

<tr>

<td>

@PetType.Name

</td>

<td>

<a href="/PetTypes/edit/@PetType.Id" class="btn btn-warning"><i class="oi oi-pencil" /> Editar</a>

<button class="btn btn-danger" @onclick=@(() => Delete(PetType.Id))><i class="oi oi-trash" /> Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</Body>

</GenericList>

}

@code {

public List<PetType>? PetTypes { get; set; }

protected async override Task OnInitializedAsync()

{

await LoadAsync();

}

("Error", ex.Message, SweetAlertIcon.Error);

}

}

private async Task Delete(int PetTypeId)

{

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/PetTypes/{PetTypeId}");

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;

}

}

1. Modificamos el **NavMenu.razor**:

<AuthorizeView Roles="Admin">

<Authorized>

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

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

<span class="oi oi-list" aria-hidden="true"></span> Pet Type

</NavLink>

</div>

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

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

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

</NavLink>

</div>

</Authorized>

</AuthorizeView>

1. Probamos lo que llevamos hasta el momento.
2. Creamos el **PetTypeForm**:

@inject SweetAlertService sweetAlertService

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>Pet Type:</label>

<div>

<InputText width=“10”@bind-Value="@PetType.Name" />

<ValidationMessage For="@(() => PetType.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 PetType PetType { 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(PetType);

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasMofied = editContext.IsModified();

if (!formWasMofied || FormPostedSuccessfully)

{

return;

}

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Creamos el **PetTypeCreate**:

@page "/PetTypes/create"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Crear Pet Type</h3>

<PetTypeForm @ref="PetTypeForm" PetType="PetType" OnValidSubmit="CreateAsync" ReturnAction="Return" />

@code {

private PetType PetType = new();

private PetTypeForm? PetTypeForm;

[Parameter]

public int StateId { get; set; }

private async Task CreateAsync()

{

var httpResponse = await repository.PostAsync("/api/PetTypes", PetType);

if (httpResponse.Error)

{

var message = await httpResponse.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

PetTypeForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo($"/PetTypes");

}

}

1. Creamos el **PetTypeEdit**:

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

@using System.Net;

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Editar categoría</h3>

@if (PetType is null)

{

<div class="spinner" />

}

else

{

<PetTypeForm @ref="PetTypeForm" PetType="PetType" OnValidSubmit="EditAsync" ReturnAction="Return" />

}

@code {

private PetType? PetType;

private PetTypeForm? PetTypeForm;

[Parameter]

public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

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

if (responseHttp.Error)

{

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

{

navigationManager.NavigateTo("/PetTypes");

return;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

PetType = responseHttp.Response;

}

private async Task EditAsync()

{

var responseHttp = await repository.PutAsync("/api/PetTypes", PetType);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

PetTypeForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo($"/PetTypes");

}

}

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

## Implementación de ventanas modales

Documentación oficial en:<https://blazored.github.io/Modal/>

1. Instalar el Nugget **Blazored.Modal** En Veterinary.WEB
2. Lo inyectamos en el **Program** del proyecto **WEB**:

builder.Services.AddBlazoredModal();

1. Modificamos el **\_Imports.razor**:

@using Blazored.Modal

@using Blazored.Modal.Services

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

<CascadingBlazoredModal Position="ModalPosition.Middle" Size="ModalSize.Large" HideHeader="true" DisableBackgroundCancel="true" AnimationType="ModalAnimationType.FadeInOut">

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

<Found Context="routeData">

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

<Authorizing>

<p>Autorizando...</p>

</Authorizing>

<NotAuthorized>

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

</NotAuthorized>

</AuthorizeRouteView>

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

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>No encontrado</PageTitle>

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

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

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

</CascadingBlazoredModal>

1. Ejemplo con categorías, modificamos el **PetTypes.index**:

…

<a class="btn btn-sm btn-primary float-end" @onclick=@(() => ShowModal())><i class="oi oi-plus"></i> Adicionar Categoría</a>

…

<a @onclick=@(() => ShowModal(PetType.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<PetTypeEdit>(string.Empty, new ModalParameters().Add("PetTypeId", id));

}

else

{

modalReference = Modal.Show<PetTypeCreate>();

}

var result = await modalReference.Result;

if (result.Confirmed)

{

await LoadAsync();

}

}

…

1. Modificamos el **PetTypesEdit**:

…

[CascadingParameter]

BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

private async Task EditAsync()

{

var responseHttp = await repository.Put("/api/PetTypes", PetType);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

}

…

1. Modificamos el **PetTypesCreate**:

…

[CascadingParameter]

BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

private async Task CreateAsync()

{

var httpResponse = await repository.Post("/api/PetTypes", PetType);

if (httpResponse.Error)

{

var message = await httpResponse.GetErrorMessageAsync();

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

return;

}

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

}

…

1. Probamos (Corremos la App con Ctrl + F5) y hacemos el **commit**.

## Administrar usuarios y crear nuevos administradores

1. Adicionamos estos métodos al **AccountController** (primero inyectamos el **DataContext**):

private readonly DataContext \_context;

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

{

\_userHelper = userHelper;

\_configuration = configuration;

\_fileStorage = fileStorage;

\_container = "users";

\_context = context;

this.\_mailHelper = mailHelper;

}

[HttpGet(“all”)]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> GetAll([FromQuery] )

{

var queryable = \_context.Users

.AsQueryable();

return Ok(await queryable

.OrderBy(x => x.FirstName)

.ThenBy(x => x.LastName)

.ToListAsync());

}

1. Adicionamos estas líneas al **NavMenu**:

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

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

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

</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>

1. Creamos el **UserIndex** dentro de **Pages/Auth**:

@page "/users"

@using Microsoft.AspNetCore.Authorization;

@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>

<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; }

protected async override Task OnInitializedAsync()

{

await LoadAsync();

}

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