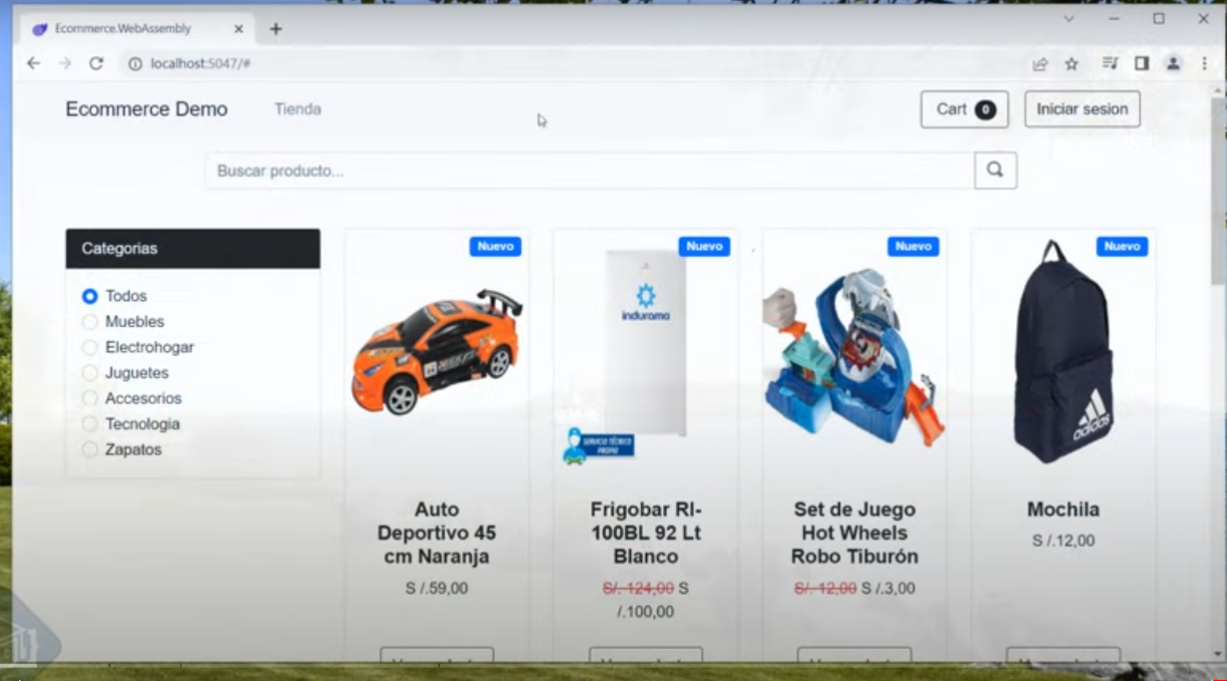
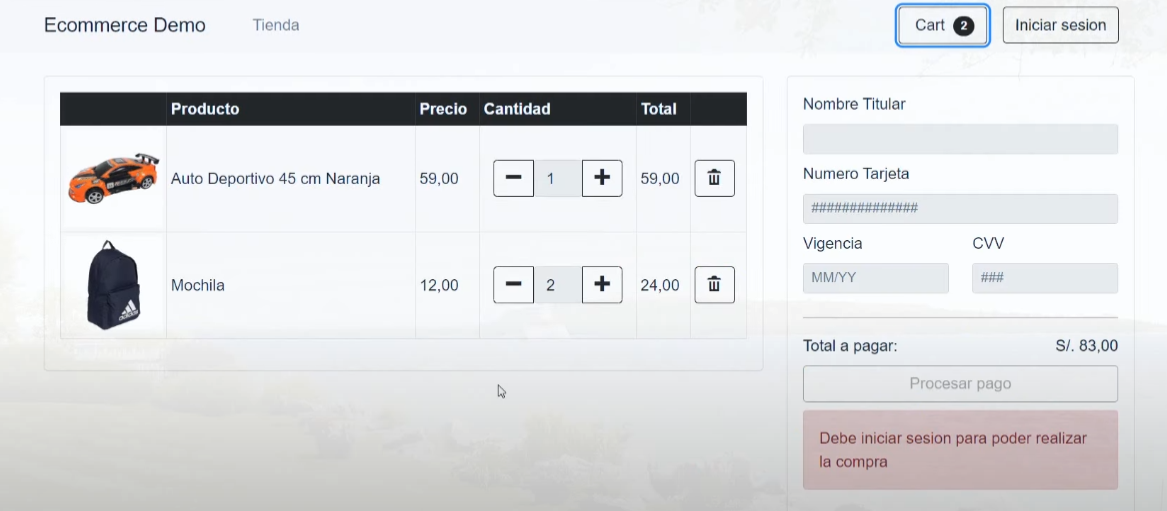
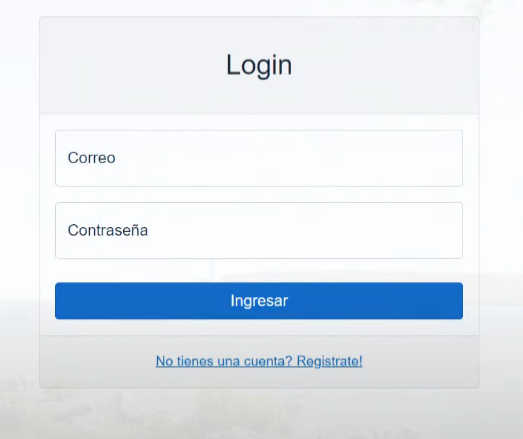
<https://www.youtube.com/watch?v=dB8qx2qJhTQ&list=PLx2nia7-PgoCnF366sDM3yMSgCWExeAf2&index=1>

Clase I

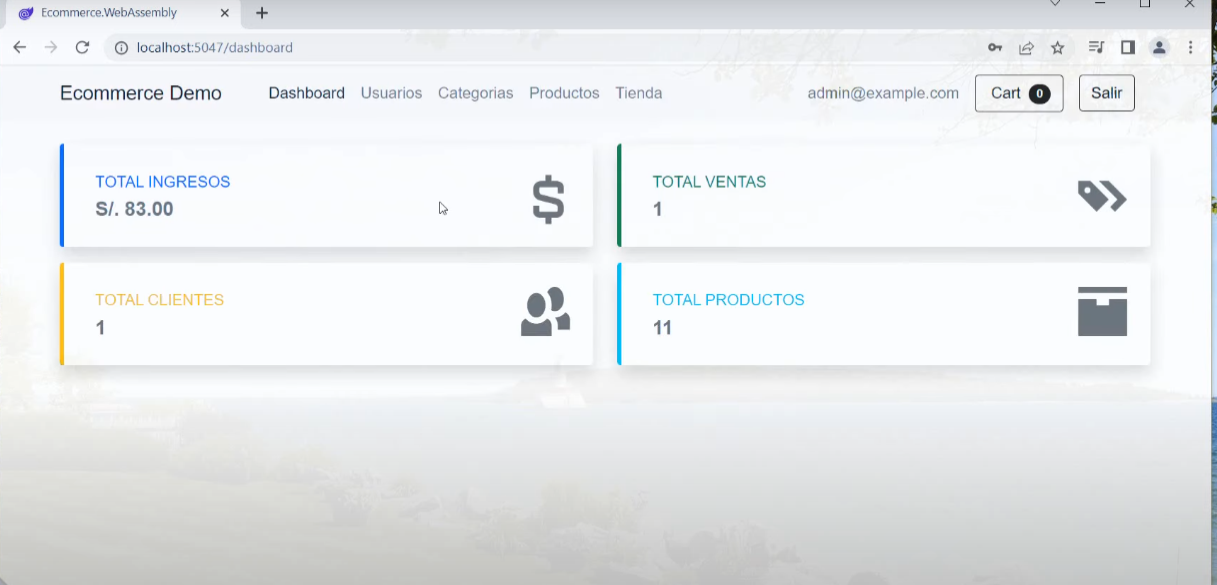


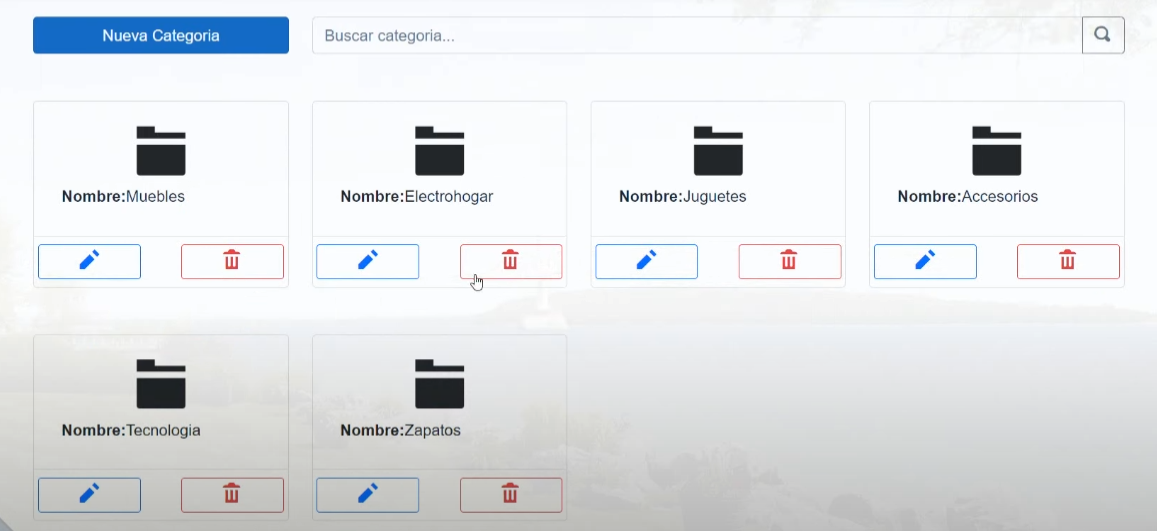






Iniciando sesión como administrador:





BBDD

create database DBEcommerce

go

use DBEcommerce

go

create table Categoria(

IdCategoria int primary key identity,

Nombre varchar(50),

FechaCreacion datetime default getdate()

)

go

create table Producto(

IdProducto int primary key identity,

Nombre varchar(50),

Descripcion varchar(1000),

IdCategoria int references Categoria(IdCategoria),

Precio decimal(10,2),

PrecioOferta decimal(10,2),

Cantidad int,

Imagen varchar(max),

FechaCreacion datetime default getdate()

)

go

create table Usuario(

IdUsuario int primary key identity,

NombreCompleto varchar(50),

Correo varchar(50),

Clave varchar(50),

Rol varchar(50),

FechaCreacion datetime default getdate()

)

go

create table Venta(

IdVenta int primary key identity,

IdUsuario int references Usuario(IdUsuario),

Total decimal(10,2),

FechaCreacion datetime default getdate()

)

go

create table DetalleVenta

(

IdDetalleVenta int primary key identity,

IdVenta int references Venta(IdVenta),

IdProducto int references Producto(IdProducto),

Cantidad int,

Total decimal(10,2)

)

--insertamos un usuario para poder iniciar sesion

insert into Usuario(NombreCompleto,Correo,Clave,Rol) values

('administrador','admin@example.com'

-Modelo pone todas las clases que representan a las tablas.

-Repositorio interactúa con la base de datos a través de los modelos. Accede al modelo

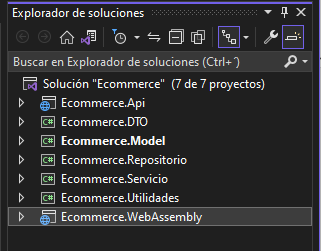
-Servicio interactúa con el repositorio y con la interfaz, que será una Api. Depende de Modelo, DTO, Utilidades y Repositorio.

-DTO contiene clases similares a los modelos. Es como la versión de los modelos que se muestra al usuario.

-Utilidades. Encriptar contraseñas. Automappers (pasar modelos en DTOs). Recursos externos. Depende de DTO y con Modelo.

-API. Depende de Servicio.

-WebAssembly. Depende solo de DTO.



El proyecto WebAssembly se va a conectar con la API. La API interactuará con todas las otras capas.

Modelo

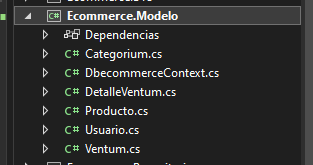
Nuget: entity sql y tools.

Estableció este como proyecto de inicio para hacer esto (no se bien qué ganó con ello), luego lo revirtió.

SE PUEDE USAR COMO INICIO LA API, PERO HAY QUE INSTALAR EL CORE DESIGN.

Comando:

Scaffold-DbContext "Server=(local); DataBase= Inmobiliaria; Trusted\_Connection=True; TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer

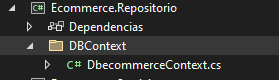


Se generan algunos nombres raros. Corregirlos!

DbecommerceContext, hace la comunicación con la BBDD con los modelos.

*Separar contexto del modelo*

Repositorio=> NuevaCarpeta DbContext



Modificar namespace a dbcommerceContext.

Agregar los using. Referencias.

AppSettings.Json(Proyecto Api)

"ConnectionStrings": {

"CadenaSql": "Server=(local); DataBase=DBEcommerce; Trusted\_Connection=True; TrustServerCertificate=True;"

}

Program.Cs

builder.Services.AddDbContext<DbecommerceContext>(options =>

{

options.UseSqlServer(builder.Configuration.GetConnectionString("CadenaSql"));

});

Repositorio:Contrato=>Interface

Crea una sola interfaz, aunque podría hacer una por cada modelo.

public interface IGenericoRepositorio<TModelo> where TModelo : class

Así, con el where se le dice que TModelo es una clase. Se puede usar cualquier clase del modelo.

namespace Ecommerce.Repositorio.Contrato

{

public interface IGenericoRepositorio<TModelo> where TModelo : class

{

IQueryable<TModelo> Consultar(Expression<Func<TModelo,bool>>?filtro=null);

//Func es una función genérica. Recibe un TModelo y devuelve un booleano.

//Por defecto será nulo.

//El filtro se puede poner si se desea o no.

Task <TModelo> Crear(TModelo modelo);

//Que se entienda, sería como poner: Task<Usuario> CrearUsuario(Usuario usuario);

//Para cada modelo.

Task<bool> Editar(TModelo modelo);

Task<bool> Eliminar(TModelo modelo);

}

}

El IQueryable es la consulta SQL. Dice que el filtro es parecido a hacer la parte del where.



La segunda interfaz implementa la primera, pero no es genérica, trabaja solo con el modelo de Venta.

public interface IVentaRepositorio:IGenericoRepositorio<Venta>

{

Task <Venta> Registrar(Venta modelo);

}

No creo que se llame implementar esta relación de interfaz a interfaz.

Repositorio.Implementación

public class GenericoRepositorio<TModelo> : IGenericoRepositorio<TModelo> where TModelo : class

{

private readonly DbecommerceContext \_context;

public GenericoRepositorio(DbecommerceContext context)

{

\_context = context;

}

public IQueryable<TModelo> Consultar(Expression<Func<TModelo, bool>>? filtro = null)

{

//Si filtro es igual a nulo muestra todo. Select \* from Modelo

//Si no, hace Select \* from Modelo where talCosa??filtro

IQueryable<TModelo> consulta = (filtro == null) ? \_context.Set<TModelo>() : \_context.Set<TModelo>().Where(filtro);

return consulta;

}

public async Task<TModelo> Crear(TModelo modelo)

{

try

{

\_context.Set<TModelo>().Add(modelo);

await \_context.SaveChangesAsync();

return modelo;

}

catch

{

throw;

}

}

public async Task<bool> Editar(TModelo modelo)

{

try

{

\_context.Set<TModelo>().Update(modelo);

await \_context.SaveChangesAsync();

return true;

}

catch

{

throw;

}

}

public async Task<bool> Eliminar(TModelo modelo)

{

try

{

\_context.Set<TModelo>().Remove(modelo);

await \_context.SaveChangesAsync();

return true;

}

catch

{

throw;

}

}

}

Hereda no la interfaz sino la implementación. Hace el retorno a la clase padre con :base.

Transacción permite reestablecer todos los pasos generados, en el caso de que haya ocurrido algún error.

*Brutal!!*

public class VentaRepositorio : GenericoRepositorio<Venta>, IVentaRepositorio

{

private readonly DbecommerceContext \_context;

public VentaRepositorio(DbecommerceContext context): base(context)

{

\_context = context;

}

public async Task<Venta> Registrar(Venta venta)

{

Venta ventaGenerada = new Venta();

using (var transaction=\_context.Database.BeginTransaction())

{

try

{

//\*\*\* Resta a la BBDD los productos comprados \*\*\*

//DetalleVenta es una lista dentro de la clase venta

foreach (DetalleVenta item in venta.DetalleVenta)

{

Producto productoEncontrado = \_context.Productos.Where(p => p.IdProducto == item.IdProducto).First();

productoEncontrado.Cantidad -= item.Cantidad;

\_context.Productos.Update(productoEncontrado);

await \_context.SaveChangesAsync();

}

//\*\*\*Crear info en tabla de ventas

await \_context.Venta.AddAsync(venta);

await \_context.SaveChangesAsync();

//Llena de contenido a ventaGenerada

ventaGenerada = venta;

transaction.Commit();

}

catch

{

//Esto es para reestablecer si hubo algún error.

transaction.Rollback();

throw;

}

return ventaGenerada;

}

}

}

Donde se dice que todo esto tiene que ser implementado todo lo hecho:

Program.Cs

Acá se inyecta todo.

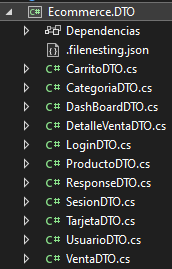
//Esto se hace así porque no se sabe con qué modelo se va a trabajar

builder.Services.AddTransient(typeof(IGenericoRepositorio<>),typeof(GenericoRepositorio<>));

//Acá si se sabe con qué modelo se va a trabajar.

builder.Services.AddScoped<IVentaRepositorio, VentaRepositorio>();

DTO



Como se ve, además de las clases que “copian” a las del modelo, hay otras, que toman de las de modelo, pero hacen su mix o un recorte notable. También toman de otras clases DTO.

ResponseDTO va a ser la respuesta a todas las solicitudes que hagamos desde nuestra API. La hace genérica.

public class ResponseDTO<K>

{

public K? Resultado { get; set; }

public bool? esCorrecto { get; set; }

public string? Mensaje { get; set; }

}

Utilidades

Acá se van a hacer las conversiones de modelo a DTO, así como de DTO a modelo.

*Automapper*

*HDeLeon*

Asignar info de un objeto a otro, que son parecidos (o sea, son distintos).

Cuando los nombres de las variables son iguales, el mapeo se hace automáticamente, con solo poner:

CreateMap<Categoria, CategoriaDTO>();

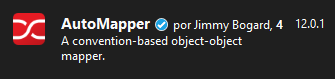
Acá se tiene una variable llamada Nombre y se la linkea con una variable llamada Name:

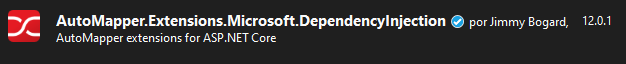


Se toca el program Cs para inyectarlo. HDeLeon no tiene capa de servicios, así que lo usa en controller. Yo, que si tengo, lo uso en servicios y en controller llamo a servicios.

private readonly IMapper \_mapper;

Y se le da valor vía el constructor





Crea una clase para el automapper. Con sus referencias:

using AutoMapper;

using Ecommerce.DTO;

using Ecommerce.Modelo;

Aclara que el mapeo se hace automáticamente, ya que las clases que se mapean tienen las mismas variables, si se les cambiase el nombre ya no sería así. Pero luego aclara, que se puede especificar con “.forMember”.

//Profile viene de automapper

public class AutoMapperProfile : Profile

{

public AutoMapperProfile()

{

CreateMap<Usuario, UsuarioDTO>();

CreateMap<Usuario, SesionDTO>();

CreateMap<UsuarioDTO, Usuario>();

CreateMap<Categoria, CategoriaDTO>();

CreateMap<CategoriaDTO, Categoria>();

CreateMap<Producto, ProductoDTO>();

//Acá le dice que ignore navigation ya que es distinta la variable.

CreateMap<ProductoDTO, Producto>().ForMember(destino =>

destino.IdCategoriaNavigation, opt=>opt.Ignore()

);

CreateMap<DetalleVenta, DetalleVentaDTO>();

CreateMap<DetalleVentaDTO, DetalleVenta>();

CreateMap<Venta, VentaDTO>();

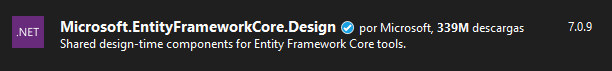
CreateMap<VentaDTO, Venta>();

}

}

Program.Cs

Nuget: Core Design



builder.Services.AddAutoMapper(typeof(AutoMapperProfile));

Servicio

Carpeta contrato y carpeta implementación

public interface IUsuarioServicio1

{

Task<List<UsuarioDTO>> Lista(string rol, string buscar);

Task<UsuarioDTO> Obtener(int id);

Task<SesionDTO> Autorizacion(LoginDTO modelo);

Task<UsuarioDTO> Crear(UsuarioDTO modelo);

Task<bool> Editar(UsuarioDTO modelo);

Task<bool> Eliminar(int id);

}

Clase potentona:

public class UsuarioServicio : IUsuarioServicio

{

private readonly IGenericoRepositorio<Usuario> \_modeloRepositorio;

private readonly IMapper \_mapper;

public UsuarioServicio(IGenericoRepositorio<Usuario> modeloRepositorio, IMapper mapper)

{

\_modeloRepositorio = modeloRepositorio;

\_mapper = mapper;

}

public async Task<SesionDTO> Autorizacion(LoginDTO modelo)

{

try

{

var consulta = \_modeloRepositorio.Consultar(p => p.Correo==modelo.Correo && p.Clave == modelo.Clave);

var fromDbModelo = await consulta.FirstOrDefaultAsync();

if (fromDbModelo != null) {

//Si hay un usuario con ese correo y clave, retorna como respuesta la sesión

//SesiónDTO lo llena con fromDbModelo

return \_mapper.Map<SesionDTO>(fromDbModelo);

}

else

{

throw new TaskCanceledException("No se encontraron coincidencias");

}

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<UsuarioDTO> Crear(UsuarioDTO modelo)

{

try

{

//Llega por parámetro un usuarioDTO y se lo convierte a Usuario.

var dbModelo=\_mapper.Map<Usuario>(modelo);

var rspModelo = await \_modeloRepositorio.Crear(dbModelo);

if (rspModelo.IdUsuario != 0)

{

return \_mapper.Map<UsuarioDTO>(rspModelo);

}

else

{

throw new TaskCanceledException("No se puede crear");

}

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<bool> Editar(UsuarioDTO modelo)

{

try

{

var consulta = \_modeloRepositorio.Consultar(p => p.IdUsuario == modelo.IdUsuario);

var fromDbModelo = await consulta.FirstOrDefaultAsync();

if (fromDbModelo != null)

{

fromDbModelo.NombreCompleto = modelo.NombreCompleto;

fromDbModelo.Correo = modelo.Correo;

fromDbModelo.Clave = modelo.Clave;

var respuesta = await \_modeloRepositorio.Editar(fromDbModelo);

if (!respuesta)

{

throw new TaskCanceledException("No se pudo editar");

}

return respuesta;

}

else

{

throw new TaskCanceledException("No se encontraron resultados");

}

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<bool> Eliminar(int id)

{

try

{

var consulta = \_modeloRepositorio.Consultar(p => p.IdUsuario == id);

var fromDbModelo = await consulta.FirstOrDefaultAsync();

if (fromDbModelo != null)

{

var respuesta = await \_modeloRepositorio.Eliminar(fromDbModelo);

// acá solo entra en caso de ser falso

if (!respuesta)

{

throw new TaskCanceledException("No se pudo eliminar");

}

return respuesta;

}

else

{

throw new TaskCanceledException("No se encontraron resultados");

}

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<List<UsuarioDTO>> Lista(string rol, string buscar)

{

try

{

var consulta = \_modeloRepositorio.Consultar(p =>

p.Rol == rol &&

string.Concat(p.NombreCompleto.ToLower(), p.Correo.ToLower()).Contains(buscar.ToLower()));

//Junta nombre y mail, lo pasa a minuscula y ahí busca si está el texto que

//se le pasa por parámetro.

List<UsuarioDTO > lista=\_mapper.Map<List<UsuarioDTO>>(await consulta.ToListAsync());

return lista;

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<UsuarioDTO> Obtener(int id)

{

try

{

var consulta = \_modeloRepositorio.Consultar(p => p.IdUsuario == id);

var fromDbModelo = await consulta.FirstOrDefaultAsync();

if (fromDbModelo != null)

{

return \_mapper.Map<UsuarioDTO>(fromDbModelo);

}

else

{

throw new TaskCanceledException("No se encontraron coincidencias");

}

}

catch (Exception ex)

{

throw ex;

}

}

}

Luego, hay muuuuucho de copypaste, modificando el tipo de datos allí donde correspondiere.

Finalmente agrega los addscoped por cada servicio.

Controladores

Program.cs

builder.Services.AddScoped<IUsuarioServicio, UsuarioServicio>();

builder.Services.AddScoped<ICategoriaServicio, CategoriaServicio>();

builder.Services.AddScoped<IVentaServicio, VentaServicio>();

builder.Services.AddScoped<IProductoServicio, ProductoServicio>();

builder.Services.AddScoped<IDashboardServicio, DashboardServicio>();

Conflicto de cors:

La api se ejecuta en una dirección, en tanto que webAssembly en otra URL. Cors es establecer la política que permita la conexión entre ambas.

Program.Cs.

Luego de los controladores:

builder.Services.AddCors(options =>

{

options.AddPolicy("NuevaPolitica", app =>

{

app.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader();

}) ;

});

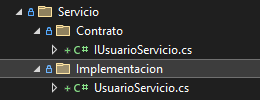
Antes de UseAuthorization

app.UseCors("NuevaPolitica");

Program de WebAssembly

Se le cambia la URL base, para que coincida con la de la API.

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("http://localhost:5049/api/") });



Crea interfaz IUsuarioServicio y luego clase que implementa, UsuarioServicio. Ahora hace referencia a Json.

public class UsuarioServicio:IUsuarioServicio

{

private readonly HttpClient \_httpClient;

public UsuarioServicio(HttpClient httpClient)

{

\_httpClient = httpClient;

}

public async Task<ResponseDTO<SesionDTO>> Autorizacion(LoginDTO modelo)

{

//Se espera la respuesta de haber ejecutado la api.

//Se ejecuta un método post, y se le envía un Json.

//Usuario es el controlador, Autorizacion el método

//modelo es lo escrito en el Body (Html)

var response = await \_httpClient.PostAsJsonAsync("Usuario/Autorizacion", modelo);

var result = await response.Content.ReadFromJsonAsync<ResponseDTO<SesionDTO>>();

return result!;

}

public async Task<ResponseDTO<UsuarioDTO>> Crear(UsuarioDTO modelo)

{

var response = await \_httpClient.PostAsJsonAsync("Usuario/Crear", modelo);

var result = await response.Content.ReadFromJsonAsync<ResponseDTO<UsuarioDTO>>();

return result!;

}

public async Task<ResponseDTO<bool>> Editar(UsuarioDTO modelo)

{

var response = await \_httpClient.PutAsJsonAsync("Usuario/Editar", modelo);

var result = await response.Content.ReadFromJsonAsync<ResponseDTO<bool>>();

return result!;

}

public async Task<ResponseDTO<bool>> Eliminar(int id)

{

//se manda como parámetro el id, con todo ese entrecomillado.

return await \_httpClient.DeleteFromJsonAsync<ResponseDTO<bool>>($"Usuario/Eliminar/{id}");

}

public async Task<ResponseDTO<List<UsuarioDTO>>> Lista(string rol, string buscar)

{

return await \_httpClient.GetFromJsonAsync<ResponseDTO<List<UsuarioDTO>>>($"Usuario/Lista/{rol}/{buscar}");

}

public async Task<ResponseDTO<UsuarioDTO>> Obtener(int id)

{

return await \_httpClient.GetFromJsonAsync<ResponseDTO<UsuarioDTO>>($"Usuario/Lista/{id}");

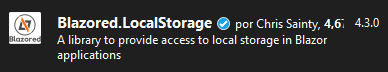
}

}

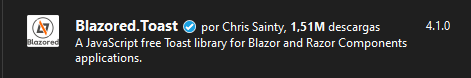
Agrega también un servicio carrito. Que posibilita agregar los productos al carrito.

Carrito

Paquete NuGet para almacenar en LocalStorage



Paquete para que al agregar al carrito se despliegue sexy notificación:



Program.cs

using Blazored.LocalStorage;

builder.Services.AddBlazoredLocalStorage();

using Blazored.Toast;

builder.Services.AddBlazoredToast();

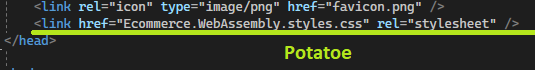
Imports.Razor

@using Blazored.LocalStorage;

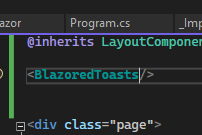
@using Blazored.Toast;

@using Blazored.Toast.Services;

Se puede chequear que se haya hecho todo bien yendo a wwwroot => index



Shared=>MainLayout.Razor



Interfaz y Clase

public interface ICarritoServicio

{

event Action MostrarItems;

int CantidadProductos();

Task AgregarCarrito(CarritoDTO modelo);

Task EliminarCarrito(int idProducto);

Task<List<CarritoDTO>> DevolverCarrito();

Task LimpiarCarrito();

}

public class CarritoServicio : ICarritoServicio

{

private ILocalStorageService \_localStorageService;

private ISyncLocalStorageService \_syncLocalStorageService;

private IToastService \_toastService;

public CarritoServicio(

ILocalStorageService localStorageService,

ISyncLocalStorageService syncLocalStorageService,

IToastService toastService

)

{

\_localStorageService = localStorageService;

\_syncLocalStorageService = syncLocalStorageService;

\_toastService = toastService;

}

public event Action MostrarItems;

public async Task AgregarCarrito(CarritoDTO modelo)

{

try

{

//Acá se intenta obtener toda la lista del carrito que se haya guardado.

var carrito = await \_localStorageService.GetItemAsync<List<CarritoDTO>>("carrito");

if (carrito == null)

{

carrito = new List<CarritoDTO>();

}

//encontrado busca si ya hay algún producto igual al que se acaba de seleccionar, a través de su id.

var encontrado = carrito.FirstOrDefault(c => c.Producto.IdProducto == modelo.Producto.IdProducto);

if (encontrado != null)

{

//se elimina, así no hay duplicidad de productos dentro del carrito.

carrito.Remove(encontrado);

}

//acá se agrega, si no, solo se borraría de haber existido.

carrito.Add(modelo);

await \_localStorageService.SetItemAsync("carrito", carrito);

if (encontrado != null)

{

\_toastService.ShowSuccess("El producto fue actualizado");

}

else

{

\_toastService.ShowSuccess("El producto fue agregado al carrito");

}

//actualizar la vista:

MostrarItems.Invoke();

}

catch (Exception ex)

{

\_toastService.ShowError("No se pudo agregar al carrito");

}

}

public int CantidadProductos()

{

var carrito = \_syncLocalStorageService.GetItem<List<CarritoDTO>>("carrito");

//Si el carrito es null se devuelve 0, si no, se devuelve el .Count();

return carrito==null? 0 : carrito.Count();

}

public async Task<List<CarritoDTO>> DevolverCarrito()

{

var carrito = await \_localStorageService.GetItemAsync<List<CarritoDTO>>("carrito");

if (carrito == null)

{

//Si es nulo se crea un carrito. No se para qué corno sirve esto.

//Se crea, pero queda vacío.

carrito = new List<CarritoDTO>();

}

return carrito;

}

public async Task EliminarCarrito(int idProducto)

{

try

{

var carrito = await \_localStorageService.GetItemAsync<List<CarritoDTO>>("carrito");

if (carrito != null)

{

var elemento = carrito.FirstOrDefault(c => c.Producto.IdProducto == idProducto);

if (elemento != null)

{

carrito.Remove(elemento);

await \_localStorageService.SetItemAsync("carrito", carrito);

MostrarItems.Invoke();

}

}

}

catch (Exception ex)

{ }

}

public async Task LimpiarCarrito()

{

await \_localStorageService.RemoveItemAsync("carrito");

MostrarItems.Invoke();

}

}

Program.Cs

Agrega Servicios:

using Ecommerce.WebAssembly.Servicio.Contrato;

using Ecommerce.WebAssembly.Servicio.Implementacion;

builder.Services.AddScoped<IUsuarioServicio, UsuarioServicio>();

builder.Services.AddScoped<ICategoriaServicio, CategoriaServicio>();

builder.Services.AddScoped<IProductoServicio, ProductoServicio>();

builder.Services.AddScoped<ICarritoServicio, CarritoServicio>();

builder.Services.AddScoped<IVentaServicio, VentaServicio>();

builder.Services.AddScoped<IDashBoardServicio, DashboardServicio>();

Clase 7

NavMenu.Razor

Borra todo el html y se queda solo con el código.

Del archivo NavMenu que importé, lo inspecciono y copio todo su contenido.

Navmenu.razor tiene su css. Elimino ese archivo.

Proyecto de inicios múltiples

Click en la solución=>Propiedades

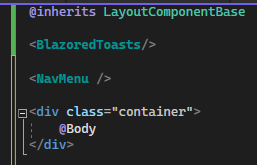
Proyecto de inicios múltiples. Primero API, segundo WebAssembly. Se les pone iniciar.

Lo que está en mainLayout (básicamente, la barra de navegación) no debe verse siempre. Por otro lado, allí se le borra su hoja de estilos.

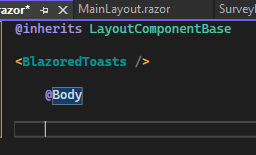
Agrega en razón un nuevo componente de Razor: LoginLayout.

Copia allí todo el contenido de mainLayout.

Así queda el mainmenu



Así LoginLayout:



Crea en pages, 3 carpetas: admin, autorización y tienda. En autorización crea componente de razor: login.

Si lleva esto que sigue es que se puede acceder, con esa dirección.

@page "/login"

Ahí se le dice con que layout va a estar conectado.

@layout LoginLayout

Eso significa que se puede acceder de 2 maneras. La primera para crear, y la segunda para editar un usuario, allí se ingresa con el Id.

@page "/usuario"

@page "/usuario/{id:int}"

Catálogo lo codea así. “/” significa que es la página de inicio.

@page "/"

@page "/catalogo"

Instalar paquetes para hacer alertas personalizadas.



Le enchufa su using a program Cs.

using CurrieTechnologies.Razor.SweetAlert2;

builder.Services.AddSweetAlert2();

en imports razor también agrega.

@using CurrieTechnologies.Razor.SweetAlert2;

Index.html

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

*Poner sobre el final, si no, no anda!!*

Clase 8

Page usuarios

@using Ecommerce.DTO;

@using Ecommerce.WebAssembly.Servicio.Contrato;

@inject IUsuarioServicio usuarioServicio;

@inject IToastService toastService;

@inject SweetAlertService swal;

Acá, al inicializarse la página, llama al método que despliega la lista de usuarios en la página.

protected override async Task OnInitializedAsync()

{

await ObtenerUsuarios();

}

\*\*\*AQUÍ VA USUARIOS.RAZOR\*\*\*

Usuario.Razor

@inject NavigationManager navServicio;

Eso dice que es para navegar entre las páginas.

Los ValidationMessage se vincula con los dataAnotattions presentes en los DTOs. Requieren que se ponga:

<**DataAnnotationsValidator**></**DataAnnotationsValidator**>

Tanto @Titulo como @Boton se verán modificados según se esté editanto o creando un usuario.

Producto.Razor

*Eventos, edición de imagen y mas!!*

Es un evento que se dispara cuando algún evento sucede:

void CambiarCategoria(ChangeEventArgs e)

{

modelo.IdCategoria = Convert.ToInt32(e.Value.ToString());

}

@page "/producto"

@page "/producto/{Id:int}"

@inject ICategoriaServicio categoriaServicio;

@inject IProductoServicio productoServicio;

@inject IToastService toastService;

@inject NavigationManager navServicio;

<h3>Producto</h3>

<div class="row justify-content-center mt-3">

<p class="col-sm-12 col-md-8 col-xl-10 h4 mt-2">@Titulo</p>

</div>

<div class="row justify-content-center mt-3">

<div class="col-sm-12 col-md-8 col-xl-10">

<**EditForm** **Model**="modelo" **OnValidSubmit**="GuardarCambios">

<**DataAnnotationsValidator**></**DataAnnotationsValidator**>

<div class="input-group">

<span class="input-group-text bg-light" id="basic-addon3">Nombre</span>

<input type="text" class="form-control" @bind-value="modelo.Nombre">

</div>

<**ValidationMessage** **For**="@(()=>modelo.Nombre)" />

<div class="input-group mt-3">

<span class="input-group-text bg-light">Descripcion</span>

<textarea class="form-control" @bind="modelo.Descripcion"></textarea>

</div>

<**ValidationMessage** **For**="@(()=>modelo.Descripcion)" />

<div class="input-group mt-3">

<label class="input-group-text bg-light" for="inputGroupSelect01">Categoria</label>

<select class="form-select" value="@modelo.IdCategoria" @onchange="CambiarCategoria">

@foreach (var item in listaCategoria)

{

<option value="@item.IdCategoria">@item.Nombre</option>

}

</select>

</div>

<div class="input-group mt-3">

<span class="input-group-text bg-light" id="basic-addon3">Precio</span>

<input type="number" class="form-control" @bind-value="modelo.Precio">

</div>

<**ValidationMessage** **For**="@(()=>modelo.Precio)" />

<div class="input-group mt-3">

<span class="input-group-text bg-light" id="basic-addon3">Precio Oferta</span>

<input type="number" class="form-control" @bind-value="modelo.PrecioOferta">

</div>

<**ValidationMessage** **For**="@(()=>modelo.PrecioOferta)" />

<div class="input-group mt-3">

<span class="input-group-text bg-light" id="basic-addon3">Cantidad</span>

<input type="number" class="form-control" @bind-value="modelo.Cantidad">

</div>

<**ValidationMessage** **For**="@(()=>modelo.Cantidad)" />

<div class="input-group mt-3">

<label class="input-group-text bg-light">Imagen</label>

<**InputFile** type="file" class="form-control" accept="image/png, image/gif, image/jpeg" **OnChange**="OnFileChange"></**InputFile**>

</div>

<div class="mt-3">

<button type="submit" class="btn btn-primary" style="width:130px">@boton</button>

<a href="/productos" class="btn btn-secondary" style="width:130px">Volver</a>

</div>

</**EditForm**>

</div>

</div>

@code {

[Parameter]

public int Id { get; set; }

private string Titulo = "Nuevo producto";

private string boton = "Crear";

private List<CategoriaDTO> listaCategoria = new List<CategoriaDTO>();

private ProductoDTO modelo = new ProductoDTO();

protected override async Task OnInitializedAsync()

{

var response = await categoriaServicio.Lista("");

if (response.EsCorrecto == true)

{

listaCategoria = (List<CategoriaDTO>)response.Resultado!;

if (listaCategoria.Any() && Id == 0)

{

modelo.IdCategoria = listaCategoria.First().IdCategoria;

}

}

}

protected override async Task OnParametersSetAsync()

{

if (Id != 0)

{

Titulo = "Editar producto";

boton = "Actualizar";

var response = await productoServicio.Obtener(Id);

if (response.EsCorrecto == true)

{

modelo = (ProductoDTO)response.Resultado!;

}

else

{

toastService.ShowWarning(response.Mensaje);

}

}

}

//lo que recibe por parámetro acá es IDCategoria.

void CambiarCategoria(ChangeEventArgs e)

{

modelo.IdCategoria = Convert.ToInt32(e.Value.ToString());

}

//Se ejecuta cada vez que se cambiamos la selección de algún archivo

async Task OnFileChange(InputFileChangeEventArgs e)

{

FileInfo fileInfo = new FileInfo(e.File.Name);

if (fileInfo.Extension.ToLower().Contains(".jpg") || fileInfo.Extension.ToLower().Contains(".png"))

{

//Replace reemplaza el punto por un vacío. O sea, elimina el punto

var format = $"image/{fileInfo.Extension.Replace(".", "")}";

var resizeImage = await e.File.RequestImageFileAsync(format, 450, 300);

var buffer = new byte[resizeImage.Size];

//Dentro del buffer vamos a poner toda la imagen.

await resizeImage.OpenReadStream().ReadAsync(buffer);

var imageData = $"data:{format};base64,{Convert.ToBase64String(buffer)}";

modelo.Imagen = imageData;

}

}

private async Task GuardarCambios()

{

bool respuesta = true;

string mensaje = string.Empty;

if (Id != 0)

{

var response = await productoServicio.Editar(modelo);

if (response.EsCorrecto == true)

{

mensaje = "El producto fue modificado";

}

else

{

respuesta = false;

mensaje = "No se pudo editar";

}

}

else

{

var response = await productoServicio.Crear(modelo);

if (response.EsCorrecto == true)

{

mensaje = "El producto fue creado";

}

else

{

respuesta = false;

mensaje = "No se pudo crear";

}

}

if (respuesta == true)

{

toastService.ShowSuccess(mensaje);

navServicio.NavigateTo("/productos");

}

else

{

toastService.ShowWarning(mensaje);

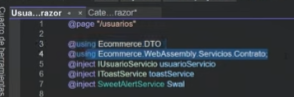
}

}

}

Mantenedor de categorías - Clase 09

CopiPasteo los html que descargué por GitHub. Cambio sus extensione a .razor dentro de VS.



Así se ve usuarios.razor

Saca los primeros 2 using y los manda a otro lugar, donde se hace una referencia global, para no andar repitiéndolos en todos los archivos razor.

Los pone en imports.razor.

Catalogo.razor (de tienda)

Eso es para que ignore si es mayúscula o minúscula:

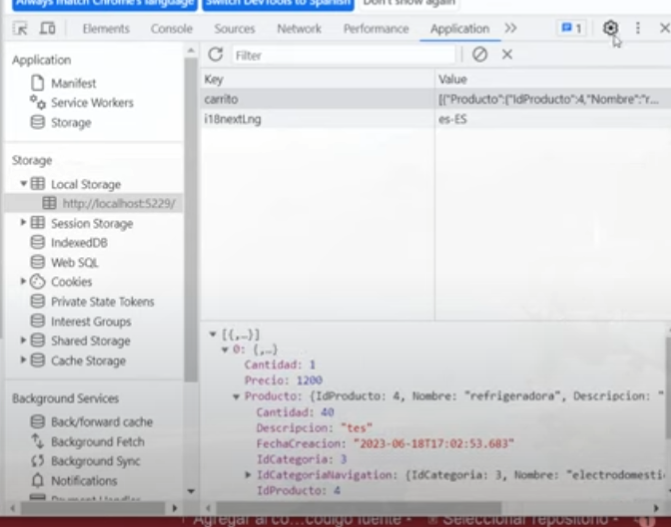
checked=@(categoriaSeleccionada.Equals(item.Nombre, StringComparison.OrdinalIgnoreCase))>

NavMenu.razor

Esto se hace para que se libere recursos, cuando se está en una página en la que no se verá el menú:

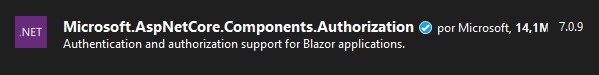
@implements IDisposable

Printeo tras agregar producto al carrito para ver cómo se llena el localStorage.



Login.Razor

Nuget:



Crea carpeta extensiones y allí clase autenticacionExtension

using Blazored.LocalStorage;

using Ecommerce.DTO;

using Microsoft.AspNetCore.Components.Authorization;

using System.Security.Claims;

Hereda de : AuthenticationStateProvider

using Blazored.LocalStorage;

using Ecommerce.DTO;

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.VisualBasic;

using System.Security.Claims;

namespace Ecommerce.WebAssembly.Extensiones

{

public class AutenticacionExtension:AuthenticationStateProvider

{

private readonly ILocalStorageService \_localStorage;

private ClaimsPrincipal \_sinInformacion=new ClaimsPrincipal(new ClaimsIdentity());

public AutenticacionExtension(ILocalStorageService localStorage)

{

\_localStorage = localStorage;

}

public async Task ActualizarEstadoAutenticacion(SesionDTO? sesionUsuario)

{

//claimsPrincipal es el usuario que ha iniciado sesión.

ClaimsPrincipal claimsPrincipal;

if (sesionUsuario!=null)

{

claimsPrincipal = new ClaimsPrincipal(new ClaimsIdentity(new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, sesionUsuario.IdUsuario.ToString()),

new Claim(ClaimTypes.Name, sesionUsuario.NombreCompleto),

new Claim(ClaimTypes.Email, sesionUsuario.Correo),

new Claim(ClaimTypes.NameIdentifier, sesionUsuario.Rol),

},"JwtAuth"));

await \_localStorage.SetItemAsync("sesionUsuario", sesionUsuario);

}

else

{

claimsPrincipal = \_sinInformacion;

await \_localStorage.RemoveItemAsync("sesionUsuario");

}

NotifyAuthenticationStateChanged(Task.FromResult(new AuthenticationState(claimsPrincipal)));

}

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var sesionUsuario = await \_localStorage.GetItemAsync<SesionDTO>("sesionUsuario");

if (sesionUsuario == null)

{

return await Task.FromResult(new AuthenticationState(\_sinInformacion));

}

var claimsPrincipal= new ClaimsPrincipal(new ClaimsIdentity(new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, sesionUsuario.IdUsuario.ToString()),

new Claim(ClaimTypes.Name, sesionUsuario.NombreCompleto),

new Claim(ClaimTypes.Email, sesionUsuario.Correo),

new Claim(ClaimTypes.NameIdentifier, sesionUsuario.Rol),

}, "JwtAuth"));

return await Task.FromResult(new AuthenticationState(\_sinInformacion));

}

}

Program.Cs

using Microsoft.AspNetCore.Components.Authorization;

using Ecommerce.WebAssembly.Extensiones;

builder.Services.AddAuthorizationCore();

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

App.razor

Cascading permite acceder a la info (autenticación) que contenga desde cualquier página.

@using Microsoft.AspNetCore.Components.Authorization;

@inject NavigationManager navManager;

<**CascadingAuthenticationState**>

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

<**Found** **Context**="routeData">

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

<**NotAuthorized**>

@{

if (context.User.Identity?.IsAuthenticated! != true)

{

navManager.NavigateTo("/login");

}

// Esto que sigue es en caso de que esté autenticado pero que por su rol no tenga acceso.

else

{

<p class="alert alert-danger">No está autorizado para ver esta página</p>

}

}

</**NotAuthorized**>

</**AuthorizeRouteView**>

</**Found**>

<**NotFound**>

<**PageTitle**>Not found</**PageTitle**>

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

<p role="alert">Sorry, there's nothing at this address.</p>

</**LayoutView**>

</**NotFound**>

</**Router**>

</**CascadingAuthenticationState**>

NavMenu

@inject ICarritoServicio carritoServicio

@implements IDisposable

@inject SweetAlertService swal;

@inject NavigationManager navigation;

@using Ecommerce.WebAssembly.Extensiones;

@using Microsoft.AspNetCore.Components.Authorization;

@using System.Security.Claims;

@inject AuthenticationStateProvider autenticacionProvider;

<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-light">

<div class="container">

<a class="navbar-brand" runat="server" href="#">Ecommerce Demo</a>

<button type="button" class="navbar-toggler" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" title="Alternar navegación" aria-controls="navbarSupportedContent"

aria-expanded="false" aria-label="Toggle navigation" @onclick="ToggleNavMenu">

<span class="navbar-toggler-icon"></span>

</button>

<div class="@NavMenuCssClass navbar-collapse d-sm-inline-flex justify-content-between">

<ul class="navbar-nav me-auto mb-2 mb-lg-0 ms-lg-4">

<**AuthorizeView** role="Administrador" >

<li class="nav-item" b-test><**NavLink** class="nav-link" href="dashboard"> Dashboard</**NavLink**></li>

<li class="nav-item"><**NavLink** class="nav-link" href="usuarios"> Usuarios</**NavLink**></li>

<li class="nav-item"><**NavLink** class="nav-link" href="categorias"> Categorias</**NavLink**></li>

<li class="nav-item"><**NavLink** class="nav-link" href="productos"> Productos</**NavLink**></li>

</**AuthorizeView**>

@\* Alguien no logueado solo podrá ver lo que viene del navmenu \*@

<li class="nav-item"><**NavLink** class="nav-link" href="catalogo"> Tienda</**NavLink**></li>

</ul>

<**AuthorizeView**>

<**Authorized**>

<span class="navbar-text me-3">

@context.User.Claims.Where(x=>x.Type==ClaimTypes.Email).Select(x=>x.Value).SingleOrDefault())

</span>

</**Authorized**>

</**AuthorizeView**>

<a class="btn btn-outline-dark ms-auto ms-md-0 me-3" href="/cart">

<i class="bi-cart-fill me-1"></i>

Cart

<span class="badge bg-dark text-white ms-1 rounded-pill">@MostrarCantidadCarrito()</span>

</a>

<**AuthorizeView**>

<**Authorized**>

<button class="btn btn-outline-dark ms-auto ms-md-0 me-3" type="button" @onclick="CerrarSesion">Salir</button>

</**Authorized**>

<**NotAuthorized**>

<a class="btn btn-outline-dark ms-auto ms-md-0 me-3" href="/login">Iniciar sesion</a>

</**NotAuthorized**>

</**AuthorizeView**>

</div>

</div>

</nav>

@code {

private bool collapseNavMenu = true;

private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

private void ToggleNavMenu()

{

collapseNavMenu = !collapseNavMenu;

}

protected override void OnInitialized()

{

//Ese evento lo que hace es recargar al componente. Se van a ir sumando los ítems

//Son 2 métodos que se convierten en uno solo.

carritoServicio.MostrarItems+=StateHasChanged;

}

public void Dispose()

{

//Acá se resta, para que ya no estén relacionados

carritoServicio.MostrarItems -= StateHasChanged;

}

private int MostrarCantidadCarrito()

{

return carritoServicio.CantidadProductos();

}

private async Task CerrarSesion()

{

SweetAlertResult result = await swal.FireAsync(new SweetAlertOptions

{

Title = "¿Desea salir?",

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

ConfirmButtonText = "Si, eliminar",

CancelButtonText = "No, volver"

});

if (result.IsConfirmed)

{

var autenticacionExterno = (AutenticacionExtension)autenticacionProvider;

await autenticacionExterno.ActualizarEstadoAutenticacion(null);

navigation.NavigateTo("/", true);

}

}

Componentes varios

Eso lo pega en varios .razor. Aquellos que solo puede ver el administrador.

@attribute [Authorize(Roles="administrador")];

Este lo lleva a imports:

@using Microsoft.AspNetCore.Authorization;

Hola Emiliano ¿cómo estás?

Te cuento que finalmente no conseguí quien pudiese seguir con Luciano por privado. Por eso te molesto para preguntarte si querías que siga su tratamiento por API.

Hola Alicia ¿cómo estás? Recordando que el martes que viene sería el último encuentro, te quería preguntar si habían decidido seguir tratamiento psicológico por API o habían tomado la decisión de discontinuarlo.

<p>@message</p>

<div class="container container-md">

<div class="mt-3">

<button type="submit" class="btn btn-primary" style="width:130px">Ir al veo veo con button</button>

<a href="/veoveo" class="btn btn-secondary" style="width:130px">Ir al veo veo con a</a>

</div>

</div>

@code

{

CategoriaDTO modelo = new CategoriaDTO();

string message;

protected override async Task OnInitializedAsync()

{

var response = await categoriaServicio.Obtener(1);

if (response.EsCorrecto == true)

{

modelo = (CategoriaDTO)response.Resultado!;

message = modelo.Nombre;

}

else

{

//toastService.ShowWarning(response.Mensaje);

}

}

}