Producto

IdProducto

Nombre

Precio

Imagen

IdCategoria

Categoria

IdCategoria

Nombre

List<Producto>

public virtual ICollection<Producto> Productos { get; set; } = new List<Producto>();

Pedido

idPedido

Venta

public int IdVenta { get; set; }

NombreUsuario

DireccionUsuario

FormaPago

public decimal? Total { get; set; }

public DateTime? FechaCreacion { get; set; }

public virtual ICollection<DetalleVenta> DetalleVenta { get; set; } = new List<DetalleVenta>();

DetalleVenta

public int IdDetalleVenta { get; set; }

public int? IdVenta { get; set; }

public int? IdProducto { get; set; }

public int? Cantidad { get; set; }

public decimal? Total { get; set; }

BBDD

create database PizzeriaDonRemolo

go

use PizzeriaDonRemolo

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),

Cantidad int,

Imagen varchar(max),

FechaCreacion datetime default getdate()

)

go

go

create table Venta(

IdVenta int primary key identity,

NombreUsuario varchar (50),

Direccion varchar (50),

FormaPago varchar (50),

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'

BBDD Script Posta

create database PizzeriaDonRemolo

use PizzeriaDonRemolo

go

create table Categoria(

IdCategoria int primary key identity,

Nombre varchar(50),

FechaCreacion datetime default getdate()

)

use PizzeriaDonRemolo

create table Producto(

IdProducto int primary key identity,

Nombre varchar(50),

Descripcion varchar(1000),

IdCategoria int references Categoria(IdCategoria),

Precio decimal(10,2),

Cantidad int,

Imagen varchar(max),

FechaCreacion datetime default getdate()

)

use PizzeriaDonRemolo

create table Venta(

IdVenta int primary key identity,

NombreUsuario varchar (50),

Direccion varchar (50),

FormaPago varchar (50),

Total decimal(10,2),

FechaCreacion datetime default getdate()

)

go

use PizzeriaDonRemolo

create table DetalleVenta

(

IdDetalleVenta int primary key identity,

IdVenta int references Venta(IdVenta),

IdProducto int references Producto(IdProducto),

Cantidad int,

Total decimal(10,2)

)

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

Con el Automapper (utilidades) se crea los DTO, al menos aquellos que son iguales o casi, que las clases de model.

Una vez que se crea, ya está, se podría borrar o comentar.

NOOOO. ESTO NO ES ASÍ

Lo que hace es linkear las variables de un par de clases similares. Ahora muestro como utilizo esto en la capa de servicios.

Repositorio

En Repositorio está el contacto con la BBDD:

public async Task<bool> Editar(TModelo modelo)

{

try

{

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

await \_context.SaveChangesAsync();

return true;

}

catch

{

throw;

}

}

Servicios

Accede al repositorio. En este caso, en el método editar, se utilizan 2 métodos de allí, consultar y Editar.

public async Task<bool> Editar(ProductoDTO modelo)

{

try

{

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

var fromDbModelo = await consulta.FirstOrDefaultAsync();

if (fromDbModelo != null)

{

fromDbModelo.Nombre = modelo.Nombre;

fromDbModelo.Descripcion = modelo.Descripcion;

fromDbModelo.IdCategoria = modelo.IdCategoria;

fromDbModelo.Precio = modelo.Precio;

fromDbModelo.PrecioOferta = modelo.PrecioOferta;

fromDbModelo.Cantidad = modelo.Cantidad;

fromDbModelo.Imagen = modelo.Imagen;

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;

}

}

Controladores

Acceden a los servicios. Ya acá no se habla mas de mappers, y menos de BBDD (repositorio).

[HttpPut("Editar")]

public async Task<IActionResult> Editar([FromBody] ProductoDTO modelo)

{

var response = new ResponseDTO<bool>();

try

{

response.EsCorrecto = true;

response.Resultado = await \_productoServicio.Editar(modelo);

}

catch (Exception ex)

{

response.EsCorrecto = false;

response.Mensaje = ex.Message;

}

return Ok(response);

}

Es exactamente en este orden en el que hace las cosas.

WebAssembly

En capa de servicios crea interfaces y sus implementaciones:

public interface ICategoriaServicio

{

Task<ResponseDTO<List<CategoriaDTO>>> Lista(string buscar);

Task<ResponseDTO<CategoriaDTO>> Obtener(int id);

Task<ResponseDTO<CategoriaDTO>> Crear(CategoriaDTO modelo);

Task<ResponseDTO<bool>> Editar(CategoriaDTO modelo);

Task<ResponseDTO<bool>> Eliminar(int id);

}

Siempre se trata con responses.

No se llama al controlador en si mismo, sino directamente se hace una petición httpClient.

Carrito no hace pedidos httpClient, de las mías esa es la única.

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

{

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

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

return result!;

}

Desde las páginas los pedidos se hacen a esta capa de servicios dentro del webAssembly.

@inject ICategoriaServicio categoriaServicio;

@inject IToastService toastService;

@inject SweetAlertService swal;

@inject NavigationManager navServicio;

Hay 3 servicios externos y está ICategoriaServicio que es la que se usa.

EditForm

OnValidSubmit implica que al hacerse click en el botón submit se va a llevar a cabo el método, en este caso “GuardarCambios”.

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

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

<div class="input-group">

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

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

</div>

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

<div class="mt-3">

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

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

</div>

</**EditForm**>

Botoncito

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

@boton es una referencia a esta variable:

private string boton = "Crear";

Como se ve en la Img, no dice crear sino actualizar. Eso es porque al cargarse la página se recibió esto:

[Parameter]

public int id { get; set; }

Categoría es cuando id=0 y la segunda opción es cuando id>0.

@page "/categoria"

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

Lo cual se procesa en el siguiente método:

protected override async Task OnParametersSetAsync()

{

if (id != 0)

{

Titulo = "Editar categoría";

boton = "Actualizar";

var response = await categoriaServicio.Obtener(id);

if (response.EsCorrecto == true)

{

modelo = (CategoriaDTO)response.Resultado!;

}

else

{

toastService.ShowWarning(response.Mensaje);

}

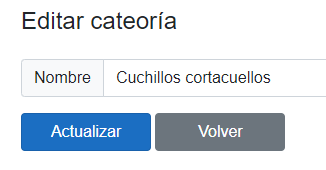
}

}

Esto también es un botón:

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

Por mas que sea un “a”, tiene clas btn





<div class="row justify-content-center mt-3 mb-4">

<div class="col-sm-6 col-md-8 col-xl-9">

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

<input type="text" class="form-control" placeholder="Buscar producto..." @bind-value="buscar">

<button class="btn btn-outline-secondary" type="button" @onclick="(async () => await ObtenerCatalogo())">

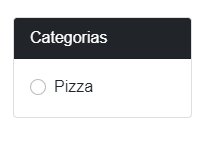
<i class="oi oi-magnifying-glass"></i>

</button>

</div>

</div>

</div>



<div class="col-sm-3">

<div class="card">

<div class="card-header text-white bg-dark">

Categorias

</div>

<div class="card-body">

@if (listaCategoria != null)

{

@foreach (var item in listaCategoria)

{

<div class="form-check">

<input class="form-check-input" type="radio" name="categoria" id="@item.IdCategoria" value=@item.Nombre

@onchange="RadioSeleccion"

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

<label class="form-check-label" for=@item.IdCategoria>

@item.Nombre

</label>

</div>

}

}

else

{

<div class="spinner-border" role="status">

<span class="visually-hidden">Loading...</span>

</div>

}

</div>

</div>

</div>



<div class="col-sm-9">

<div class="row gx-4 row-cols-1 row-cols-md-3 row-cols-xl-4 justify-content-start">

@if (listaProducto != null)

{

foreach (var item in listaProducto)

{

<div class="col mb-5">

<div class="card h-100">

<!-- Sale badge-->

@if (DateTime.Now < item.FechaCreacion.Value.AddDays(30))

{

<div class="badge bg-primary text-white position-absolute" style="top: 0.5rem; right: 0.5rem">Nuevo</div>

}

<div class="badge bg-primary text-white position-absolute" style="top: 0.5rem; right: 0.5rem">Nuevo</div>

<!-- Product image-->

<img class="card-img-top border-dark border-1" src="@item.Imagen" style="width:100%;height:250px">

<!-- Product details-->

<div class="card-body p-4">

<div class="text-center">

<!-- Product name-->

<h5 class="fw-bolder">@item.Nombre</h5>

<!-- Product price-->

<span>S /.@item.Precio</span>

</div>

</div>

<!-- Product actions-->

<div class="card-footer p-3 pt-0 border-top-0 bg-transparent">

<div class="text-center"><a class="btn btn-outline-dark mt-auto" href="/detalle/@item.IdProducto">Ver producto</a></div>

</div>

</div>

</div>

}

}

else

{

<div class="spinner-border" role="status">

<span class="visually-hidden">Loading...</span>

</div>

}

</div>

</div>

