Código estudiante

https://www.youtube.com/watch?v=NOJveObsX0Q

BBDD

Create table Departamento(

IdDepartamento int primary key identity (1,1),

Nombre varchar(50) not null )

create table Empleado (

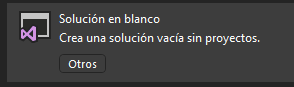
IdEmpleado int primary key identity (1,1),

NombreCompleto varchar (50) not null,

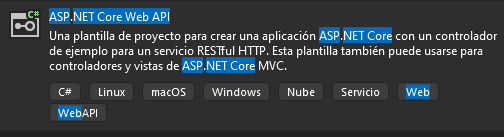
IdDepartamento int references Departamento (IdDepartamento) not null,

Sueldo int not null,

FechaContrato date not null)



Luego, agrega un proyecto de estos:



Deshabilita https.

Paquetes Nugets:Sql y CoreTools

Crea carpeta Models. Ahí guarda toda la configuración de entity Framework.

Consola nuget=>

Scaffold-DbContext “Server=(local); DataBase=CrudBlazor; Trusted\_Connection=True; TrustServerCertificate=True;” Microsoft.EntityFrameworkCore.SqlServer -OutPutDir Models

"ConnectionStrings": {

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

}

Program Cs.

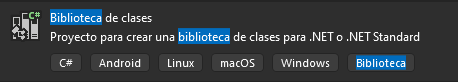
builder.Services.AddDbContext<CrudBlazorContext>(opciones =>

{

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

});

Agrega un nuevo proyecto, en el que almacena la info que se va a compartir con el cliente:



Por eso, por ejemplo, de la tabla Departamento, no copia este campo:

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

EmpleadoDTO

public class EmpleadoDTO

{

public int IdEmpleado { get; set; }

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

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

[Required]

[Range(1, int.MaxValue, ErrorMessage = "El campo {0} es requerido")]

public int IdDepartamento { get; set; }

[Required]

[Range(1, int.MaxValue, ErrorMessage = "El campo {0} es requerido")]

public int Sueldo { get; set; }

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

public DateTime FechaContrato { get; set; }

public DepartamentoDTO? Departamento { get; set; }

}

ResponseApi

Se cambia el formato de como se va a recibir las respuestas del lado del servidor hacia el cliente.

public class ResponseApi<T>

{

public bool EsCorrecto { get; set; }

public T? Valor { get; set; }

public string? Mensaje { get; set; }

}

Clickea en el proyecto principal y le agrega el Shared como dependencia.

HTTPGET

DepartamentoController

Parte desde proyecto Api en blanco.

El contexto la info que transporta y que se inyecta es de la BBDD.

namespace BlazorCrudServer.Controllers

{

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

[ApiController]

public class DepartamentoController : ControllerBase

{

private readonly CrudBlazorContext \_context;

public DepartamentoController(CrudBlazorContext context)

{

\_context = context;

}

[HttpGet]

[Route("Lista")]

public async Task<IActionResult> Lista()

{

var responseApi = new ResponseApi<List<DepartamentoDTO>>();

var listaDepartamentoDTO=new List<DepartamentoDTO>();

try

{

foreach (var item in await \_context.Departamentos.ToListAsync())

{

listaDepartamentoDTO.Add(new DepartamentoDTO

{

IdDepartamento = item.IdDepartamento,

Nombre=item.Nombre,

});

}

responseApi.EsCorrecto = true;

responseApi.Valor = listaDepartamentoDTO;

}

catch (Exception ex)

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = ex.Message;

}

return Ok(responseApi);

}

}

}

ResponseApi como enseñó Hugo Mansilla es un objeto complejo y flexible. Es lo único que se devuelve, pero su llenado depende de si el try es positivo o no. Puede devolver un msj de error o los datos del departamento.

EmpleadoController

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

[ApiController]

public class EmpleadoController : ControllerBase

{

private readonly CrudBlazorContext \_context;

public EmpleadoController(CrudBlazorContext context)

{

\_context = context;

}

[HttpGet]

[Route("Lista")]

public async Task<IActionResult> Lista()

{

var responseApi = new ResponseApi<List<EmpleadoDTO>>();

var listaEmpleadoDTO = new List<EmpleadoDTO>();

try

{

foreach (var item in await \_context.Empleados.Include(d=>d.IdDepartamentoNavigation).ToListAsync())

{

listaEmpleadoDTO.Add(new EmpleadoDTO

{

IdEmpleado=item.IdEmpleado,

NombreCompleto=item.NombreCompleto,

IdDepartamento=item.IdDepartamento,

Sueldo=item.Sueldo,

FechaContrato=item.FechaContrato,

Departamento=new DepartamentoDTO

{

IdDepartamento=item.IdDepartamentoNavigation.IdDepartamento,

Nombre=item.IdDepartamentoNavigation.Nombre

}

});

}

responseApi.EsCorrecto = true;

responseApi.Valor = listaEmpleadoDTO;

}

catch (Exception ex)

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = ex.Message;

}

return Ok(responseApi);

}

}

HTTPGETporID

[HttpGet]

[Route("Buscar/{id}")]

public async Task<IActionResult> Lista(int id)

{

var responseApi = new ResponseApi<EmpleadoDTO>();

var empleadoDTO = new EmpleadoDTO();

try

{

var dbEmpleado = await \_context.Empleados.FirstOrDefaultAsync(x => x.IdEmpleado == id);

if (dbEmpleado != null)

{

empleadoDTO.IdEmpleado = dbEmpleado.IdEmpleado;

empleadoDTO.NombreCompleto = dbEmpleado.NombreCompleto;

empleadoDTO.IdDepartamento = dbEmpleado.IdDepartamento;

empleadoDTO.Sueldo = dbEmpleado.Sueldo;

empleadoDTO.FechaContrato = dbEmpleado.FechaContrato;

responseApi.EsCorrecto = true;

responseApi.Valor = empleadoDTO;

}

else

{

responseApi.Mensaje = "Empleado no encontrado";

}

}

catch (Exception ex)

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = ex.Message;

}

return Ok(responseApi);

}

HTTPPost

[HttpPost ]

[Route("Guardar")]

public async Task<IActionResult> Guardar(EmpleadoDTO empleado)

{

var responseApi = new ResponseApi<int>();

try

{

//Acá se convierte de tipo Empleado a tipo Empleado DTO

var dbEmpleado = new Empleado

{

NombreCompleto = empleado.NombreCompleto,

IdDepartamento = empleado.IdDepartamento,

Sueldo = empleado.Sueldo,

FechaContrato = empleado.FechaContrato,

};

\_context.Empleados.Add(dbEmpleado);

await \_context.SaveChangesAsync();

if (dbEmpleado.IdEmpleado != 0)

{

responseApi.EsCorrecto = true;

responseApi.Valor = dbEmpleado.IdEmpleado;

}

else

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = "No guardado";

}

}

catch (Exception ex)

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = ex.Message;

}

return Ok(responseApi);

}

HTTPPut

[HttpPut]

[Route("Editar/{id}")]

public async Task<IActionResult> Editar(EmpleadoDTO empleado, int id)

{

var responseApi = new ResponseApi<int>();

try

{

var dbEmpleado = await \_context.Empleados.FirstOrDefaultAsync(emp => emp.IdEmpleado == id);

if (dbEmpleado!= null)

{

dbEmpleado.Sueldo=empleado.Sueldo;

dbEmpleado.NombreCompleto=empleado.NombreCompleto;

dbEmpleado.IdDepartamento = empleado.IdDepartamento;

dbEmpleado.FechaContrato=empleado.FechaContrato;

\_context.Empleados.Update(dbEmpleado);

await \_context.SaveChangesAsync();

responseApi.EsCorrecto = true;

responseApi.Valor = dbEmpleado.IdEmpleado;

}

else

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = "Empleado no encontrado";

}

}

catch (Exception ex)

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = ex.Message;

}

return Ok(responseApi);

}

HTTPDelete

[HttpDelete]

[Route("Eliminar/{id}")]

public async Task<IActionResult> Eliminar(int id)

{

var responseApi = new ResponseApi<int>();

try

{

var dbEmpleado = await \_context.Empleados.FirstOrDefaultAsync(emp => emp.IdEmpleado == id);

if (dbEmpleado != null)

{

\_context.Empleados.Remove(dbEmpleado);

await \_context.SaveChangesAsync();

responseApi.EsCorrecto = true;

}

else

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = "Empleado no encontrado";

}

}

catch (Exception ex)

{

responseApi.EsCorrecto = false;

responseApi.Mensaje = ex.Message;

}

return Ok(responseApi);

}

Program.Cs

Dice tener que activar los Cors ya que el proyecto de servidor y el proyecto del cliente van a ser ejecutados en URLs distintas.

Así se la crea:

builder.Services.AddCors(opciones =>

{

opciones.AddPolicy("Nueva política", app =>

{

app.AllowAnyOrigin();

app.AllowAnyHeader();

app.AllowAnyMethod();

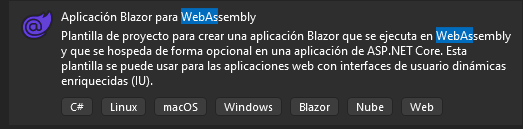
});

});

Así se la activa:

app.UseCors("Nueva política");

Crear proyecto Blazor WebAssembly



Deshabilita https.

Agrega la dependencia en este ultimo proyecto. Solamente se selecciona Shared.

Program.Cs

Del proyecto blazor.

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });

Eso en rojo es lo que viene por defecto. Lo borra e inserta la url de la Api: ("http://localhost:5029/")

Agrega carpeta services. Allí una interfaz IDepartamentoService

IDepartamentoService + DepartamentoService

using BlazorCrudShared;

public interface IDepartamentoService

{

Task<List<DepartamentoDTO>> Lista();

}

namespace BlazorCrud.Cliente.Service

{

public class DepartamentoService:IDepartamentoService

{

private readonly HttpClient \_http;

public DepartamentoService(HttpClient http)

{

\_http = http;

}

public async Task<List<DepartamentoDTO>> Lista()

{

var result = await \_http.GetFromJsonAsync<ResponseApi<List<DepartamentoDTO>>>("api/Departamento/Lista");

if (result!.EsCorrecto) return result.Valor!;

else throw new Exception(result.Mensaje);

}

}

}

IEmpleadoService + EmpleadoService

public interface IEmpleadoService

{

Task<List<EmpleadoDTO>> Lista();

Task<EmpleadoDTO> Buscar(int id);

Task<int> Guardar(EmpleadoDTO empleado);

Task<int> Editar (EmpleadoDTO empleado);

Task<bool> Eliminar(int id);

}

using BlazorCrudShared;

using System.Net.Http.Json;

namespace BlazorCrud.Cliente.Service

{

public class EmpleadoService:IEmpleadoService

{

private readonly HttpClient \_http;

public EmpleadoService(HttpClient http)

{

\_http = http;

}

public async Task<List<EmpleadoDTO>> Lista()

{

var result = await \_http.GetFromJsonAsync<ResponseApi<List<EmpleadoDTO>>>("api/Empleado/Lista");

if (result!.EsCorrecto) return result.Valor!;

else throw new Exception(result.Mensaje);

}

public async Task<EmpleadoDTO> Buscar(int id)

{

var result = await \_http.GetFromJsonAsync<ResponseApi<EmpleadoDTO>>($"api/Empleado/Buscar/{id}");

if (result!.EsCorrecto) return result.Valor!;

else throw new Exception(result.Mensaje);

}

public async Task<int> Guardar(EmpleadoDTO empleado)

{

//Tanto con Post como con Put, en este result se convierte el empleado a JSON.

var result = await \_http.PostAsJsonAsync("api/Empleado/Guardar",empleado);

var response = await result.Content.ReadFromJsonAsync<ResponseApi<int>>();

if (response!.EsCorrecto) return response.Valor!;

else throw new Exception(response.Mensaje);

}

public async Task<int> Editar(EmpleadoDTO empleado)

{

var result = await \_http.PutAsJsonAsync($"api/Empleado/Editar/{empleado.IdEmpleado}", empleado);

var response = await result.Content.ReadFromJsonAsync<ResponseApi<int>>();

if (response!.EsCorrecto) return response.Valor!;

else throw new Exception(response.Mensaje);

}

public async Task<bool> Eliminar(int id)

{

var result = await \_http.DeleteAsync($"api/Empleado/Eliminar/{id}");

var response = await result.Content.ReadFromJsonAsync<ResponseApi<int>>();

if (response!.EsCorrecto) return response.EsCorrecto!;

else throw new Exception(response.Mensaje);

}

Program.Cs

Para agregar los servicios recién creados.

using BlazorCrud.Cliente.Service;

//Aquí, tras el addScoped, primero agrega el tipo, y luego, la clase que la implementa.

builder.Services.AddScoped<IDepartamentoService, DepartamentoService>();

builder.Services.AddScoped<IEmpleadoService, EmpleadoService>();



using CurrieTechnologies.Razor.SweetAlert2;

builder.Services.AddSweetAlert2();

wwwroot=>index.html

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

</body>

Agrega ese Script justo antes del fin del body.

Crea componente de razor Empleados en la carpeta Pages.

Empleados.Razor

Para que el componente se comporte como página hay que poner al ppio @page. Luego se pone cómo se accede a la misma.

@page "/Empleados"

@using BlazorCrudShared;

@using BlazorCrud.Cliente.Service;

@using CurrieTechnologies.Razor.SweetAlert2;

Ahí se está inyectando, y creando una variable.

@inject SweetAlertService Swal;

@inject IEmpleadoService empleadoService;

<h3>Empleados</h3>

Este evento se ejecuta cuando la página está empezando a renderizarse.

OnInitializedAsync()

NavMenu

En la page NavMenú, dentro de shared, agrega un enlace a Empleados.

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

<**NavLink** class="nav-link" href="Empleados">

<span class="oi oi-person" aria-hidden="true"></span> Empleados

</**NavLink**>

</div>

Ese es el empleado potatoe.

Empleados.Razor

@page "/Empleados"

@using BlazorCrudShared;

@using BlazorCrud.Cliente.Service;

@using CurrieTechnologies.Razor.SweetAlert2;

@inject SweetAlertService Swal;

@inject IEmpleadoService empleadoService;

<h3>Empleados</h3>

<!-- Este botón te dirige a la page Empleado (en singular)-->

<a class="btn btn-success btn-sm mb-3" href="Empleado">Nuevo Empleado</a>

<table class="table">

<thead>

<tr>

<th>Nombre Completo</th>

<th>Departamento</th>

<th>Sueldo</th>

<th>Fecha de contrato</th>

<th></th>

</tr>

</thead>

<tbody>

@if (listaEmpleados == null)

{

<tr>

<td colspan="5" align="center">

<img style="width:50px" src="https://media.tenor.com/On7kvXhzml4AAAAj/loading-gif.gif">

</td>

</tr>

}

else

{

@foreach (var item in listaEmpleados)

{

<tr>

<td>@item.NombreCompleto </td>

<td>@item.Departamento!.Nombre </td>

<td>@item.Sueldo </td>

<td>@item.FechaContrato.ToString("dd/MM/yyyy") </td>

<td>

<a class="btn btn-primary btn-sm" href="Empleado/@item.IdEmpleado">

<i class="oi oi-pencil"></i>

</a>

<!-- Son 2 botones, pero solo el 2° es button. Porque el primero te redirige a otra page -->

<button type="button" class="btn btn-danger btn-sm ms-2" @onclick="(() => Eliminar(item.IdEmpleado))">

<i class="oi oi-trash"></i>

</button>

</td>

</tr>

}

}

</tbody>

</table>

@code {

List<EmpleadoDTO>? listaEmpleados = null;

protected override async Task OnInitializedAsync()

{

listaEmpleados = await empleadoService.Lista();

}

private async Task Eliminar(int id)

{

var resultado = await Swal.FireAsync(new SweetAlertOptions

{

Title = "Eliminar empleado",

Text = "¿Deseas eliminar el empleado seleccionado?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

if (resultado.IsConfirmed)

{

var eliminado = await empleadoService.Eliminar(id);

if (eliminado)

{

//Acá omite llamar al servicio, mostrando simplemente la lista, sin el elemento recientemente eliminado.

listaEmpleados = listaEmpleados!.FindAll(e => e.IdEmpleado != id);

}

}

}

}

Empleado.Razor

@page "/Empleado"

@page "/Empleado/{idEmpleadoEditar:int}"

@using BlazorCrudShared;

@using BlazorCrud.Cliente.Service;

@using Microsoft.AspNetCore.Components.Forms

@inject IEmpleadoService empleadoService;

@inject IDepartamentoService departamentoService;

@inject NavigationManager navegacion;

<h3>@Titulo</h3>

<**EditForm** **Model**="empleado" **OnValidSubmit**="OnValidSubmit">

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

<div class="mb-3">

<label class="form-label">Nombre Completo</label>

<**InputText** class="form-control" @bind-Value="empleado.NombreCompleto"></**InputText**>

<**ValidationMessage** **For**="@(()=>empleado.NombreCompleto)"></**ValidationMessage**>

</div>

<div class="mb-3">

<label class="form-label">Departamento</label>

<**InputSelect** class="form-select" @bind-Value="empleado.IdDepartamento">

<option value="0">-- Seleccionar --</option>

@foreach (var item in listaDepartamento)

{

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

}

</**InputSelect**>

<**ValidationMessage** **For**="@(()=>empleado.IdDepartamento)"></**ValidationMessage**>

</div>

<div class="mb-3">

<label class="form-label">Sueldo</label>

<**InputNumber** class="form-control" @bind-Value="empleado.Sueldo"></**InputNumber**>

<**ValidationMessage** **For**="@(()=>empleado.Sueldo)"></**ValidationMessage**>

</div>

<div class="mb-3">

<label class="form-label">Fecha de contrato</label>

<**InputDate** class="form-control" @bind-Value="empleado.FechaContrato"></**InputDate**>

<**ValidationMessage** **For**="@(()=>empleado.FechaContrato)"></**ValidationMessage**>

</div>

<button class="btn btn-primary" type="submit">

@btnTexto

</button>

<a class="btn btn-warning" href="Empleados">Volver</a>

</**EditForm**>

@code {

[Parameter] public int idEmpleadoEditar { get; set; } = 0;

string Titulo = string.Empty;

string btnTexto = string.Empty;

EmpleadoDTO empleado = new EmpleadoDTO();

List<DepartamentoDTO> listaDepartamento = new List<DepartamentoDTO>();

protected override async Task OnInitializedAsync()

{

if (idEmpleadoEditar != 0)

{

empleado = await empleadoService.Buscar(idEmpleadoEditar);

btnTexto = "Actualizar empleado";

Titulo = "Editar empleado";

}

else

{

empleado.FechaContrato = DateTime.Today;

btnTexto = "Guardar empleado";

Titulo = "Nuevo empleado";

}

listaDepartamento = await departamentoService.Lista();

}

private async Task OnValidSubmit()

{

int idDevuelto = 0;

if (idEmpleadoEditar == 0)

{

idDevuelto = await empleadoService.Guardar(empleado);

}

else

{

idDevuelto = await empleadoService.Editar(empleado);

}

if (idDevuelto != 0)

{

navegacion.NavigateTo("/empleados");

}

}

}

https://www.youtube.com/watch?v=dNq3A-EtOdk

Cosas nuevas:

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

De esa forma, de no ponerse nada, tiraría ese msg.

Este otro campo es numérico, por eso hace así. Solo acepta números entre el 1 y el mayor número de tipo int.

[Required]

[Range(1, int.MaxValue, ErrorMessage = "El campo {0} es requerido")]

Http=>Http

Me tiraba un error: Failed to load resource: net::ERR\_SSL\_PROTOCOL\_ERROR

Lo arreglé cambiando los Http por Https