Con la base de datos: **PruebaEntity:**

A screenshot of a computer

Description automatically generated

1. Crear Aplicación web de ASP.NET Core (Modelo-Vista-Controlador) en C#
2. Instalar Paquetes:

A screenshot of a black screen

Description automatically generated

1. Ejecutar Conexión:

Scaffold-DbContext "Data Source= COLOCAR LO QUE DIGA SQL ;Initial Catalog= NOMBRE DE BASE DE DATOS CREADO;Integrated Security=True; TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

1. Chequear las clases creadas en la carpeta Models
2. Pasar la cadena de conexión a appsettings.json (y comentar el warning en el conexto)

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"CadenaSQL": "Data Source= localhost\\SQLEXPRESS ;Initial Catalog= PruebaEntity;Integrated Security=True; TrustServerCertificate=True"

}

}

*\*cambiar lo que está en amarillo si corresponde*

1. Agregar Controladores con Vistas para Estudiante y Curso (Controlador MVC con vistas que usan Entity Framework)

**No olvidar de agregar la cadena de conexión en Program.cs**

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

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

1. Probar CRUD SI FUNCIONA TODO OK
2. Ver como se relacionan Controllers y Contexto

**UNIDAD DE TRABAJO Y REPOSITORIO**

1. **Crear La carpeta Repositorio**
2. **Crear Interface para repositorio Genérico (**IGenericRepository)

using System.Collections.Generic;

using System.Linq.Expressions;

namespace MVCEF.Repositorio

{

public interface IGenericRepository<TEntity> where TEntity : class

{

IEnumerable<TEntity> GetAll();

TEntity GetById(int id);

void Insert(TEntity entity);

void Update(TEntity entity);

void Delete(int id);

IEnumerable<TEntity> GetAllWithIncludes(params Expression<Func<TEntity, object>>[] includeProperties);

TEntity GetByIdWithIncludes(int id, params Expression<Func<TEntity, object>>[] includeProperties);

}

}

1. **Crear clase Repositorio Genérico**

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using Microsoft.EntityFrameworkCore;

using MVCEF.Models;

namespace MVCEF.Repositorio

{

public class GenericRepository<TEntity> : IGenericRepository<TEntity> where TEntity : class

{

private readonly PruebaEntityContext \_context;

private readonly DbSet<TEntity> \_dbSet;

public GenericRepository(PruebaEntityContext context)

{

\_context = context;

\_dbSet = context.Set<TEntity>();

}

public IEnumerable<TEntity> GetAll()

{

return \_dbSet.ToList();

}

public TEntity GetById(int id)

{

return \_dbSet.Find(id);

}

public void Insert(TEntity entity)

{

\_dbSet.Add(entity);

}

public void Update(TEntity entity)

{

\_dbSet.Attach(entity);

\_context.Entry(entity).State = EntityState.Modified;

}

public void Delete(int id)

{

TEntity entity = \_dbSet.Find(id);

if (entity != null)

{

\_dbSet.Remove(entity);

}

}

public IEnumerable<TEntity> GetAllWithIncludes(params Expression<Func<TEntity, object>>[] includeProperties)

{

IQueryable<TEntity> query = \_dbSet;

foreach (var includeProperty in includeProperties)

{

query = query.Include(includeProperty);

}

return query.ToList();

}

public TEntity GetByIdWithIncludes(int id, params Expression<Func<TEntity, object>>[] includeProperties)

{

IQueryable<TEntity> query = \_dbSet;

foreach (var includeProperty in includeProperties)

{

query = query.Include(includeProperty);

}

return query.FirstOrDefault(e => EF.Property<int>(e, "Id") == id); // Usar EF.Property para obtener la propiedad Id

}

}

}

1. **Crear Interface para Unidad de Trabajo (IUnitOfWork)**

using MVCEF.Models;

using System;

namespace MVCEF.Repositorio

{

public interface IUnitOfWork : IDisposable

{

IGenericRepository<Curso> Cursos { get; }

IGenericRepository<Estudiante> Estudiantes { get; }

void Save();

}

}

1. **Crear Clase de Repositorio**

using MVCEF.Models;

namespace MVCEF.Repositorio

{

public class UnitOfWork : IUnitOfWork

{

private readonly PruebaEntityContext \_context;

private IGenericRepository<Curso> \_cursos;

private IGenericRepository<Estudiante> \_estudiantes;

public UnitOfWork(PruebaEntityContext context)

{

\_context = context;

}

public IGenericRepository<Curso> Cursos

{

get { return \_cursos ??= new GenericRepository<Curso>(\_context); }

}

public IGenericRepository<Estudiante> Estudiantes

{

get { return \_estudiantes ??= new GenericRepository<Estudiante>(\_context); }

}

public void Save()

{

\_context.SaveChanges();

}

public void Dispose()

{

\_context.Dispose();

}

}

}

1. **Actualizar controlador de Cursos**

private readonly IUnitOfWork \_unitOfWork;

public CursosController(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

// GET: Cursos

public async Task<IActionResult> Index()

{

return View(await Task.FromResult(\_unitOfWork.Cursos.GetAll()));

}

// GET: Cursos/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var curso = await Task.FromResult(\_unitOfWork.Cursos.GetById(id.Value));

if (curso == null)

{

return NotFound();

}

return View(curso);

}

// GET: Cursos/Create

public IActionResult Create()

{

return View();

}

// POST: Cursos/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,NombreCurso")] Curso curso)

{

if (ModelState.IsValid)

{

\_unitOfWork.Cursos.Insert(curso);

\_unitOfWork.Save();

return RedirectToAction(nameof(Index));

}

return View(curso);

}

// GET: Cursos/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var curso = await Task.FromResult(\_unitOfWork.Cursos.GetById(id.Value));

if (curso == null)

{

return NotFound();

}

return View(curso);

}

// POST: Cursos/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,NombreCurso")] Curso curso)

{

if (id != curso.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_unitOfWork.Cursos.Update(curso);

\_unitOfWork.Save();

}

catch (DbUpdateConcurrencyException)

{

if (!CursoExists(curso.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(curso);

}

// GET: Cursos/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var curso = await Task.FromResult(\_unitOfWork.Cursos.GetById(id.Value));

if (curso == null)

{

return NotFound();

}

return View(curso);

}

// POST: Cursos/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var curso = await Task.FromResult(\_unitOfWork.Cursos.GetById(id));

if (curso != null)

{

\_unitOfWork.Cursos.Delete(id);

\_unitOfWork.Save();

}

return RedirectToAction(nameof(Index));

}

private bool CursoExists(int id)

{

return \_unitOfWork.Cursos.GetAll().Any(e => e.Id == id);

}

}

1. **Actualizar controlador de Estudiantes**

private readonly IUnitOfWork \_unitOfWork;

public EstudiantesController(IUnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

// GET: Estudiantes

public async Task<IActionResult> Index()

{

var estudiantes = \_unitOfWork.Estudiantes.GetAllWithIncludes(e => e.IdCursoNavigation);

return View(await Task.FromResult(estudiantes));

}

// GET: Estudiantes/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var estudiante = \_unitOfWork.Estudiantes.GetByIdWithIncludes(id.Value, e => e.IdCursoNavigation);

if (estudiante == null)

{

return NotFound();

}

return View(await Task.FromResult(estudiante));

}

// GET: Estudiantes/Create

public IActionResult Create()

{

ViewData["IdCurso"] = new SelectList(\_unitOfWork.Cursos.GetAll(), "Id", "Id");

return View();

}

// POST: Estudiantes/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Nombre,Apellido,Edad,IdCurso")] Estudiante estudiante)

{

if (ModelState.IsValid)

{

\_unitOfWork.Estudiantes.Insert(estudiante);

\_unitOfWork.Save();

return RedirectToAction(nameof(Index));

}

ViewData["IdCurso"] = new SelectList(\_unitOfWork.Cursos.GetAll(), "Id", "Id", estudiante.IdCurso);

return View(estudiante);

}

// GET: Estudiantes/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var estudiante = \_unitOfWork.Estudiantes.GetById(id.Value);

if (estudiante == null)

{

return NotFound();

}

ViewData["IdCurso"] = new SelectList(\_unitOfWork.Cursos.GetAll(), "Id", "Id", estudiante.IdCurso);

return View(estudiante);

}

// POST: Estudiantes/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Nombre,Apellido,Edad,IdCurso")] Estudiante estudiante)

{

if (id != estudiante.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_unitOfWork.Estudiantes.Update(estudiante);

\_unitOfWork.Save();

}

catch (DbUpdateConcurrencyException)

{

if (!EstudianteExists(estudiante.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["IdCurso"] = new SelectList(\_unitOfWork.Cursos.GetAll(), "Id", "Id", estudiante.IdCurso);

return View(estudiante);

}

// GET: Estudiantes/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var estudiante = \_unitOfWork.Estudiantes.GetByIdWithIncludes(id.Value, e => e.IdCursoNavigation);

if (estudiante == null)

{

return NotFound();

}

return View(await Task.FromResult(estudiante));

}

// POST: Estudiantes/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var estudiante = \_unitOfWork.Estudiantes.GetById(id);

if (estudiante != null)

{

\_unitOfWork.Estudiantes.Delete(id);

\_unitOfWork.Save();

}

return RedirectToAction(nameof(Index));

}

private bool EstudianteExists(int id)

{

return \_unitOfWork.Estudiantes.GetAll().Any(e => e.Id == id);

}

}

1. **Actualizar Program.cs**

**Agregar servicio debajo de cadena de conexión**

builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));

builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();