**TALLER RESUELTO SOBRE ARQUITECTURA LIMPIA BASICA**

Desarrollado por: Ing. Aníbal Martínez Arcia

**Descripción.**

Este taller tiene como objetivo el diseño e implementación de un sistema académico modular llamado **CLAcademy**, estructurado por capas, respetando los principios de la **arquitectura limpia.** El sistema se desarrollará con C# en .NET 9 y usará **Entity Framework Core** como ORM sobre SQL Server, con una base de datos previamente configurada: AcademyDB.

Se implementará la separación lógica de los siguientes componentes:

* Dominio: Entidades, Enumeraciones.
* Aplicación: Casos de uso y lógica de negocio.
* Infraestructura: Acceso a datos (EF Core + SQL Server).
* Servicios: Lógica de servicios que consumen los repositorios.
* Presentación (API): Exposición de controladores.

**Objetivos.**

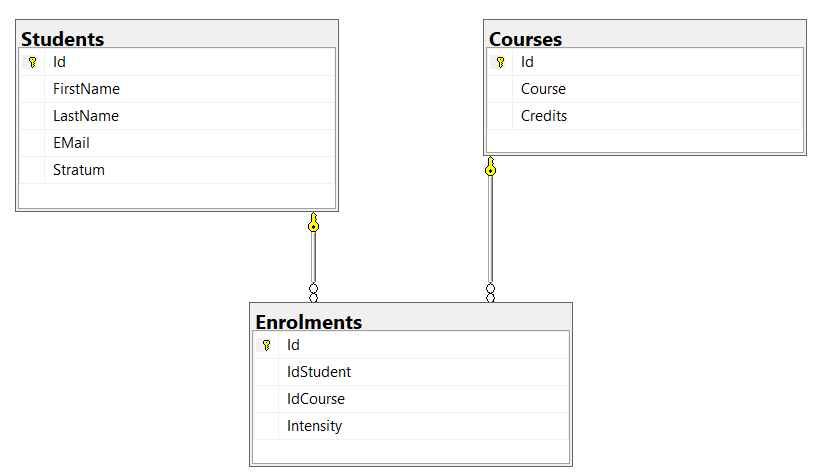
**Objetivo general.**

Diseñar e implementar una aplicación modular con arquitectura limpia, utilizando C# y .NET 9, que permita gestionar estudiantes, cursos y matrículas académicas, respetando los principios de diseño por capas, separación de responsabilidades y persistencia desacoplada mediante EF Core y SQL Server.

**Objetivos específicos.**

* Diseñar una biblioteca de clases de dominio que defina entidades, enumeraciones y objetos de valor relacionados con el modelo académico.
* Crear un proyecto de lógica de aplicación donde se implementen los casos de uso (registro, consulta y gestión de estudiantes y cursos).
* Implementar un proyecto de infraestructura con EF Core y SQL Server que se encargue de la persistencia de datos.
* Construir servicios de aplicación que consuman los repositorios definidos en la infraestructura.
* Desarrollar una capa de presentación (API RESTful) que exponga los controladores y permita el consumo de las funcionalidades del sistema.
* Aplicar buenas prácticas de programación, incluyendo inyección de dependencias, principio de inversión de dependencias y uso correcto del patrón repositorio.

Figura 1.

*Estructura de la base de datos AcademyDB*

**Scipt para creación de la Base de datos.**

CREATE DATABASE AcademyDB

GO

USE AcademyDB

GO

CREATE TABLE Students

(

Id INT PRIMARY KEY IDENTITY(1,1)

,FirstName VARCHAR(50) NOT NULL

,LastName VARCHAR(50) NOT NULL

,EMail VARCHAR(150)

,Stratum TINYINT CHECK(Stratum IN(1,2, 3))

)

GO

CREATE TABLE Courses

(

Id INT PRIMARY KEY IDENTITY(1,1)

,CourseName VARCHAR(100) NOT NULL

,Credits INT DEFAULT 2 CHECK(Credits IN(2,3,4))

)

GO

CREATE TABLE Enrollements

(

Id INT PRIMARY KEY IDENTITY(1,1)

,IdStudent INT FOREIGN KEY REFERENCES Students(Id)

,IdCourse INT FOREIGN KEY REFERENCES Courses(Id)

,Intensity INT DEFAULT 10

)

GO

**Estructura del proyecto.**

CLAcademy.Domain

CLAcademy.Application

CLAcademy.Infrastructure

CLAcademy.Services

CLAcademy.Presentation

Y opcionalmente CLAcademy.Tests, son proyectos de biblioteca de clases.

CLAcademy.sln

│

├── CLAcademy.Domain → Entidades, Enums, Interfaces, Etc.

│

├── CLAcademy.Application → Casos de uso y lógica de negocio

│

├── CLAcademy.Infrastructure → Implementación de Repositorios con EF

│

├── CLAcademy.Services → Servicios de aplicación que consumen los

│

├── CLAcademy.Presentation → API ASP.NET Core

│ ├── Controllers

│ └── Program.cs & Startup.cs (o builder en .NET 9)

│

└── CLAcademy.Tests → Proyecto opcional para pruebas unitarias de integración

**PASO 1: Crear una Solución Vacía**

1. Abre Visual Studio 2022.
2. Ve a Archivo > Nuevo > Proyecto...
3. En el cuadro de búsqueda, escribe: solución vacía o selecciona el tipo de proyecto “Solución vacía” (en inglés: *Blank Solution*).
4. Haz clic en Siguiente.

**Configuración del proyecto:**

* Nombre de la solución: CLAcademy
* Ubicación: Elige una carpeta de tu preferencia.
* Nombre de la solución: (dejamos igual) CLAcademy
* Marca la casilla "Colocar la solución y el proyecto en el mismo directorio" si deseas que todo esté en una sola carpeta raíz (recomendado).
* Haz clic en Crear.

Ahora tendrás una solución completamente vacía llamada CLAcademy, sin ningún proyecto todavía. Desde aquí comenzaremos a agregar los proyectos según tu estructura por capas.

Vamos a crear el primer proyecto: CLAcademy.Domain, donde definiremos el corazón del sistema: las entidades, enums, interfaces, etc.

**PASO 2: Crear el proyecto de Dominio (CLAcademy.Domain)**

**En Visual Studio 2022:**

* En el Explorador de soluciones, haz clic derecho sobre la solución CLAcademy.
* Selecciona:  
  Agregar > Nuevo proyecto…
* En el buscador de plantillas, escribe:  
  biblioteca de clases o selecciona "Biblioteca de clases (.NET Standard o .NET)".
* Haz clic en Siguiente.

**Configuración del proyecto:**

* Nombre del proyecto: CLAcademy.Domain
* Ubicación: (usa la misma carpeta que la solución)
* Framework: Selecciona .NET 9.0
* Haz clic en Crear

Después de esto ya tendrás tu primer proyecto CLAcademy.Domain creado. Este proyecto será totalmente independiente de cualquier otra capa. Aquí no se hace referencia a EF Core ni a infraestructura, solo definiciones puras del dominio.

**Organización de carpetas dentro de CLAcademy.Domain**

Dentro de este proyecto, comienza creando las siguientes carpetas:

CLAcademy.Domain

│

├── Entities → Entidades como Student, Course, Enrollement

├── Enums → StratumType

└── Interfaces

Nota: Borra la clase Class1 que se crea al crear el proyecto de biblioteca de clases.

**PASO 3: Creación de enum para representar el Estrato, clase Student, Course y Enrollement**

**Primero crea el enum StratumType**

* Carpeta: Enums
* Nombre del archivo: StratumType.cs

namespace CLAcademy.Domain.Enums

{

public enum StratumType : byte

{

ALTO = 1,

MEDIO,

BAJO

}

}

**Luego crea la clase Student**

* Carpeta: Entities
* Nombre del archivo: Student.cs

using CLAcademy.Domain.Enums;

namespace CLAcademy.Domain.Entities

{

public class Student

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public StratumType Stratum { get; set; }

}

}

**Resultado:**

Esta clase es:

* POCO: "Plain Old CLR Object"
* Sin lógica de negocio
* Ideal para que EF Core la mapee sin problemas
* Usando enum como StratumType

**PASO 4: Clase Course.cs**

* Carpeta: Entities
* Nombre del archivo: Course.cs

namespace CLAcademy.Domain.Entities

{

public class Course

{

public int Id { get; set; }

public string CourseName { get; set; }

public int Credits { get; set; }

}

}

**Notas:**

* Credits es un int, lo cual permite que el chequeo de valores (2, 3, 4) sea manejado por la base de datos o por validaciones en otras capas (como la capa de aplicación o presentación).

¡Perfecto! Vamos ahora con la entidad **Enrollement**, también como POCO plano, manteniendo la coherencia con las tablas SQL que ya creaste en la base de datos.

**PASO 5: Crear la clase Enrollement.cs**

**Estructura de la tabla SQL:**

CREATE TABLE Enrollements

(

Id INT PRIMARY KEY IDENTITY(1,1)

,IdStudent INT FOREIGN KEY REFERENCES Students(Id)

,IdCourse INT FOREIGN KEY REFERENCES Courses(Id)

,Intensity INT DEFAULT 10

)

**Clase Enrollement.cs**

* Carpeta: Entities
* Nombre del archivo: Enrollement.cs

namespace CLAcademy.Domain.Entities

{

public class Enrollement

{

public int Id { get; set; }

public int IdStudent { get; set; }

public int IdCourse { get; set; }

public int Intensity { get; set; }

}

}

**Notas:**

* Enrollement representa una asociación entre un estudiante y un curso.
* Podrías más adelante agregar propiedades de navegación (Student, Course) si EF Core las necesita, pero por ahora las omitimos porque estás trabajando en un modelo plano.
* El valor por defecto 10 para Intensity lo maneja directamente la base de datos, por lo tanto no necesitas especificarlo en la clase.

Con esto, ya tenemos las tres entidades base listas en el dominio:

* Student
* Course
* Enrollement

**Paso 6: Crear el proyecto CLAcademy.Application**

1. Clic derecho sobre la solución en el Solution Explorer > Add > New Project.
2. Elige Class Library (.NET Standard) o Class Library (.NET 9) (elige el mismo framework que usarás en los demás proyectos).
3. Nómbralo: CLAcademy.Application
4. Clic en Crear.
5. Borra el archivo Class1.cs que se crea por defecto.

**Paso 7: Crea carpetas internas para organizar**

Clic derecho sobre el proyecto CLAcademy.Application, y ve agregando estas carpetas:

* Interfaces
* DTOs
* Services (lo usaremos después)

Deberías ver algo así en el explorador de soluciones:

CLAcademy.Application

│

├── DTOs

├── Interfaces

└── Services

**Paso 8: Crear los DTOs**

En la carpeta DTOs crea las siguientes clases:

namespace CLAcademy.Application.DTOs

{

public class StudentDTO

{

public string FullName { get; set; } = string.Empty;

public string EMail { get; set; } = string.Empty;

public StratumType Stratum { get; set; }

}

}

**Nota**: Este DTO solo se usará en los GET, más no en POST, ni PUT, mucho menos en DELETE. Además, note que no mostrará nombre separado de apellidos, sino el FullName, nombre completo que será la concatenación de nombres y apellidos. Recuerde que la entidad Student es:

using CLAcademy.Domain.Enums;

namespace CLAcademy.Domain.Entities

{

public class Student

{

public int Id { get; set; }

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

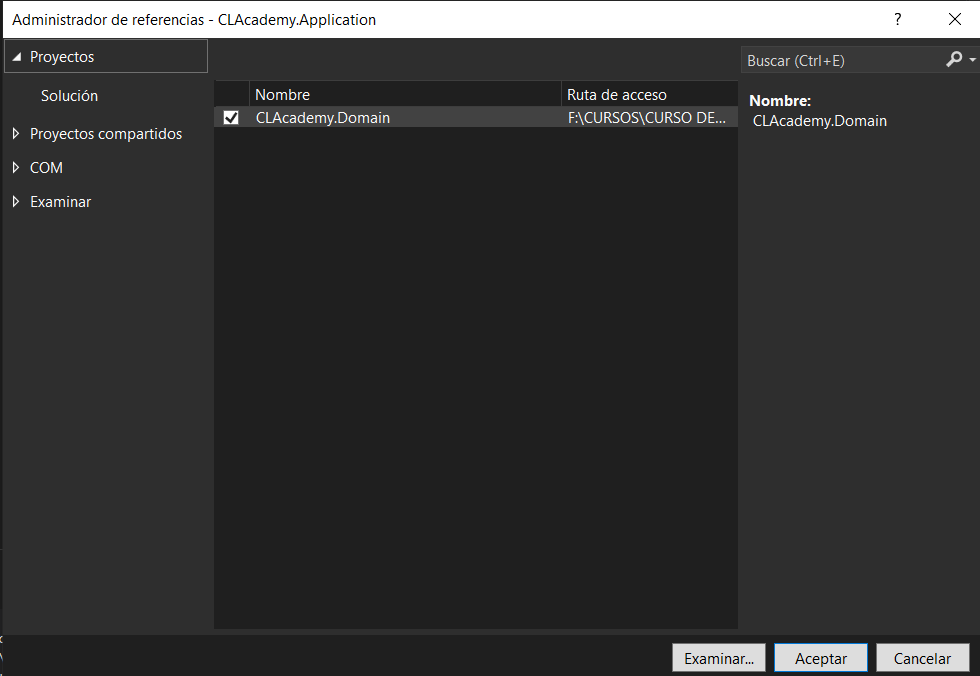
public string EMail { get; set; } = string.Empty;

public StratumType Stratum { get; set; }

}

}

En este momento tiene que agregar una referencia en el proyecto CLAcademy.Application, esta referencia apunta al proyecto CLAcademy.Domain. Para conseguirlo:

* Haga clic derecho sobre Dependencias del proyecto CLAcademy.Application
* Elija Agregar referencia del proyecto…
* Marque la casilla de verificación:
* Clic en aceptar.

Luego la clase CourseDTO:

namespace CLAcademy.Application.DTOs

{

public class CourseDTO

{

public string CourseName { get; set; } = string.Empty;

public int Credits { get; set; }

}

}

Esta clase también se usará solo para los GET’s.

**Paso 9: Crear el proyecto Infrastructure**

Agrega un nuevo proyecto de biblioteca de clases a la solución:

CLAcademy.Infrastructure

**Nota**: Elimina la clase Class1 que se creó con el proyecto.

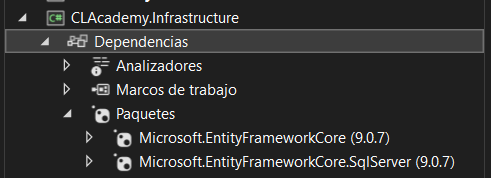
Luego dentro del proyecto CLAcademy.Infrastructure, agrega una carpeta llamada Persistence:

CLAcademy.Infrastructure/Persistence

**Paso 10: Agregar una referencia a CLAcademy.Domain**

Haga clic derecho sobre Dependencias del Proyecto CLAcademy.Infrastructure, Agregar referencia a Proyecto…

**Paso 11: Agregar los paquetes Microsoft.EntityFrameworkCore y Microsoft.EntityFrameworkCore.SqlServer**

**Nota**: Debes tener mucho cuidado con las versiones que instales de estos paquetes. En este caso yo he usado las versiones 9.0.7 para ambos paquetes.

Dentro de la carpeta CLAcademy.Infrastructure/Persistence, crea una clase AcademyDbContext.cs. Sabes que puedes cambiar este nombre por el que mejor te parezca.

using Microsoft.EntityFrameworkCore;

using CLAcademy.Domain.Entities;

namespace CLAcademy.Infrastructure.Persistence

{

public class AcademyDbContext : DbContext

{

public AcademyDbContext(DbContextOptions<AcademyDbContext> options)

: base(options)

{

}

public DbSet<Student> Students { get; set; }

public DbSet<Course> Courses { get; set; }

}

}

**Paso 12: Crear la interfaz IGenericRepository**

Esta interfaz se debe ubicar en: CLAcademy.Application/Interfaces/IGenericRepository.cs

namespace CLAcademy.Application.Interfaces

{

public interface IGenericRepository<TEntity> where TEntity : class

{

Task<List<TEntity>> GetAll();

Task<TEntity?> GetById(int id);

Task<TEntity> Insert(TEntity entity);

Task<TEntity> Update(TEntity entity);

Task DeleteById(int id);

}

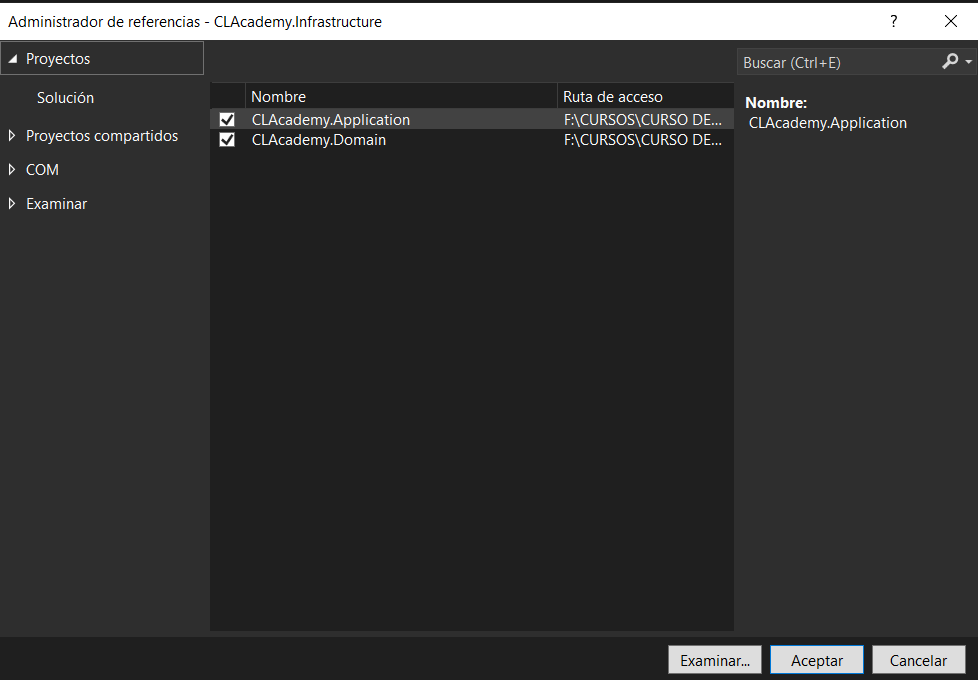
}

**Nota**: El uso de genéricos (where TEntity : class), permite que la interfaz funcione con cualquier entidad.

**Paso 13: Implementar IGenericRepository mediante GenericRepository<TEntity>**

Comienza creando la carpeta Repositories en CLAcademy.Infrastructure, además debes referenciar al proyecto CLAcademy.Application de CLAcademy.Infrastructure, recuerda que la interfaz IGenericRepository está en CLAcademy.Application.

Haga clic derecho sobre CLAcademy.Infrastructure/Dependencias y elija Agregar referencia del proyecto. Marca la casilla CLAcademy.Application.



Ahora ya puedes crear la clase GenericRepository en CLAcademy.Infrastructure/Repositories.

using CLAcademy.Application.Interfaces;

using CLAcademy.Infrastructure.Persistence;

using Microsoft.EntityFrameworkCore;

namespace CLAcademy.Infrastructure.Repositories

{

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

{

private readonly AcademyDbContext \_context;

private readonly DbSet<TEntity> \_dbSet;

public GenericRepository(AcademyDbContext context)

{

\_context = context;

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

}

public async Task<List<TEntity>> GetAll()

{

return await \_dbSet.ToListAsync();

}

public async Task<TEntity?> GetById(int id)

{

return await \_dbSet.FindAsync(id);

}

public async Task<TEntity> Insert(TEntity entity)

{

await \_dbSet.AddAsync(entity);

await \_context.SaveChangesAsync();

return entity;

}

public async Task<TEntity> Update(TEntity entity)

{

\_dbSet.Update(entity);

await \_context.SaveChangesAsync();

return entity;

}

public async Task DeleteById(int id)

{

var entity = await \_dbSet.FindAsync(id);

if (entity is not null)

{

\_dbSet.Remove(entity);

await \_context.SaveChangesAsync();

}

}

}

}

Dentro de las ventajas de usar un repositorio genérico tenemos:

* Reutilizas la lógica para todas las entidades (menos código repetido).
* Puedes extender con IGenericRepository<T> + IStudentRepository si necesitas funciones específicas más adelante.

**Paso 14: Registrar GenericRepository en la inyección de dependencias**

Crea una clase DependencyInjection en: CLAcademy.Infrastructure

using CLAcademy.Application.Interfaces;

using CLAcademy.Infrastructure.Persistence;

using CLAcademy.Infrastructure.Repositories;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

namespace CLAcademy.Infrastructure

{

public static class DependencyInjection

{

public static IServiceCollection AddInfrastructure(this IServiceCollection services,

IConfiguration configuration)

{

services.AddDbContext<AcademyDbContext>(options =>

options.UseSqlServer(configuration.GetConnectionString("")));

services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));

return services;

}

}

}

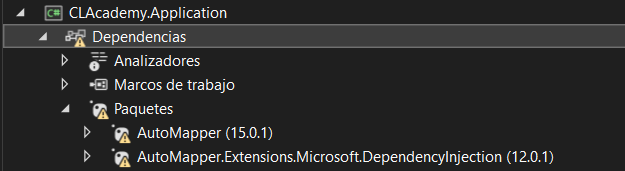
**Paso 15: Adicionar AutoMapper.**

Antes de crear los controladores debemos adicionar AutoMapper al proyecto CLAcademy.Application.

Debes hacer clic derecho sobre Dependencias del proyecto CLAcademy.Application, Administrar paquetes NuGet y agregar:

* AutoMapper
* AutoMapper.Extensions.Microsoft.DependencyInjection

**Nota**: Deja las misma versión. En el momento de hacer este taller se usó 12.0.1.

Es muy probable que te diga que AutoMapper.Extensions.Microsoft.DependencyInjection, está obsoleto, solo es una advertencia. Esto es porque será retirado o modificado en futuras versiones, debes estar atento.

### Paso 16: Crear el StudentProfile

En CLAcademy.Application, crea una nueva carpeta llamada Mappings y dentro crea la clase StudentProfile:

using AutoMapper;

using CLAcademy.Application.DTOs;

using CLAcademy.Domain.Entities;

namespace CLAcademy.Application.Mappings

{

public class StudentProfile : Profile

{

public StudentProfile()

{

// Mapeo de Student a StudentDTO

CreateMap<Student, StudentDTO>()

.ForMember(dest => dest.FullName,

opt => opt.MapFrom(src => $"{src.FirstName} {src.LastName}"));

// Mapeo inverso si lo deseas más adelante

//CreateMap<StudentDTO, Student>()

// .ForMember(dest => dest.FirstName, opt => opt.Ignore())

// .ForMember(dest => dest.LastName, opt => opt.Ignore());

}

}

}

**Notas**:

* El FullName se construye desde FirstName + LastName.
* En el mapeo inverso ignoramos FirstName y LastName porque no están en el DTO (puedes ajustar eso si decides crear un DTO de entrada más adelante, por ejemplo CreateStudentDTO).

**Paso 17: Registrar AutoMapper.**

Inserta esto exactamente después de AddControllers() en tu Program:

using CLAcademy.Application.Mappings; // este using va al inicio del archivo

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddAutoMapper(typeof(StudentProfile).Assembly); // Inserta esta línea

builder.Services.AddOpenApi();

typeof(StudentProfile).Assembly le dice a AutoMapper que busque todos los perfiles (Profile) en el ensamblado donde está definido StudentProfile.

Nota: El Program debe quedar así:

using CLAcademy.Application.Mappings;

using CLAcademy.Application.Services;

using CLAcademy.Infrastructure;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container

builder.Services.AddControllers();

// AutoMapper

builder.Services.AddAutoMapper(typeof(StudentProfile).Assembly);

// Agregar infraestructura (DbContext + Repositorios)

builder.Services.AddInfrastructure(builder.Configuration);

// Agregar Swagger estándar

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Servicios de aplicación

builder.Services.AddScoped<StudentService>();

var app = builder.Build();

// Middleware

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(); // Esto genera /swagger/index.html

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Paso 18: Implementar StudentService que consumirá el repositorio genérico.**

En la carpeta CLAcademy.Application.Services, crea la clase StudentService:

using AutoMapper;

using CLAcademy.Application.DTOs;

using CLAcademy.Application.Interfaces;

using CLAcademy.Domain.Entities;

namespace CLAcademy.Application.Services

{

public class StudentService

{

private readonly IGenericRepository<Student> \_studentRepository;

private readonly IMapper \_mapper;

public StudentService(IGenericRepository<Student> studentRepository, IMapper mapper)

{

\_studentRepository = studentRepository;

\_mapper = mapper;

}

public async Task<List<StudentDTO>> GetAllAsync()

{

var students = await \_studentRepository.GetAll();

return \_mapper.Map<List<StudentDTO>>(students);

}

public async Task<StudentDTO?> GetById(int id)

{

var student = await \_studentRepository.GetById(id);

return student == null ? null : \_mapper.Map<StudentDTO>(student);

}

// Métodos Create/Update/Delete pueden seguir usando Student directamente

public async Task<Student> Insert(Student student)

{

return await \_studentRepository.Insert(student);

}

public async Task<Student> Update(Student student)

{

return await \_studentRepository.Update(student);

}

public async Task DeleteById(int id)

{

await \_studentRepository.DeleteById(id);

}

}

}

**Paso 19: Crear el proyecto CLAcademy.Presentation.**

Antes de crear controladores, debemos crear el proyecto CLAcademy.Presentation, que en una arquitectura limpia suele ser un **proyecto tipo ASP.NET Core Web API**, ya que es el punto de entrada para las solicitudes externas (front-end, apps móviles, integraciones, etc.).

**Haz clic derecho sobre tu solución (**CLAcademy**) >** Agregar **>** Nuevo proyecto**.**

**** Elige **"ASP.NET Core Web API".**

 Nombra el proyecto:  
CLAcademy.Presentation

* Haz clic en "Siguiente" y selecciona:
* Framework: .NET 9.0

Marca solamente:

* Configurar para HHTPS
* Habilitarcompatibilidad OpenApi
* Utilizar controladores
* Haz clic en **Crear**.

**Nota**: Presentation debe hacer referencia a Infrastructure.

**Paso 20: Establecer el proyecto como proyecto de inicio y definir cadena de conexión.**

Clic derecho sobre CLAcademy.Presentation y elije Establecer como proyecto de inicio.

Ahora define la cadena de conexión el appsettings.json.

{

"ConnectionStrings": {

"DefaultConnection": "Server=DESKTOP-S3TGDJ8;Database=AcademyDB;User Id=SA;Password=12345;Encrypt=True;TrustServerCertificate=True"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

**Paso 21: Establecer referencias a proyectos.**

Haz clic derecho sobre el proyecto CLAcademy.Presentation > Agregar > Referencia de proyecto, y selecciona:

* CLAcademy.Application
* CLAcademy.Domain

**Paso 22: Crear el StudentController.**

using CLAcademy.Application.Services;

using CLAcademy.Domain.Entities;

using Microsoft.AspNetCore.Mvc;

namespace CLAcademy.Presentation.Controllers

{

[ApiController]

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

public class StudentController : ControllerBase

{

private readonly StudentService \_studentService;

public StudentController(StudentService studentService)

{

\_studentService = studentService;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var students = await \_studentService.GetAllAsync();

return Ok(students);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

var student = await \_studentService.GetById(id);

if (student == null)

return NotFound();

return Ok(student);

}

[HttpPost]

public async Task<IActionResult> Create(Student student)

{

var created = await \_studentService.Insert(student);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, Student student)

{

if (id != student.Id)

return BadRequest();

var updated = await \_studentService.Update(student);

return Ok(updated);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_studentService.DeleteById(id);

return NoContent();

}

}

}

**Paso 23: Referencia a Suwashbucle**

 Haz clic derecho sobre el proyecto CLAcademy.Presentation.

 Selecciona "Administrar paquetes NuGet".

 Ve a la pestaa "Examinar".

 Busca Swashbuckle.AspNetCore e instalalo.

En la carpeta Controllers de CLAcademy.Presentation, crea una clase StudentController:

using CLAcademy.Application.Services;

using CLAcademy.Domain.Entities;

using Microsoft.AspNetCore.Mvc;

namespace CLAcademy.Presentation.Controllers

{

[ApiController]

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

public class StudentController : ControllerBase

{

private readonly StudentService \_studentService;

public StudentController(StudentService studentService)

{

\_studentService = studentService;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var students = await \_studentService.GetAllAsync();

return Ok(students);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

var student = await \_studentService.GetById(id);

if (student == null)

return NotFound();

return Ok(student);

}

[HttpPost]

public async Task<IActionResult> Create(Student student)

{

var created = await \_studentService.Insert(student);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, Student student)

{

if (id != student.Id)

return BadRequest();

var updated = await \_studentService.Update(student);

return Ok(updated);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_studentService.DeleteById(id);

return NoContent();

}

}

}

**Ejecuta**: https://localhost:xxxx/swagger/index.html

**Paso 24: Agregar DTO para insertar Student.**

En la carpeta CLAcademy.Application/DTOs, crea una clase CreateStudentDTO

using CLAcademy.Domain.Enums;

namespace CLAcademy.Application.DTOs

{

public class CreateStudentDTO

{

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

public string Email { get; set; } = string.Empty;

public StratumType Stratum { get; set; }

}

}

**Paso 25: Agrega el mapelo en StudentProfile**

En la clase CLAcademy.Application/Mappings/StudentProfile.cs

Agrega este nuevo mapeo:

CreateMap<CreateStudentDTO, Student>();

Ahora StudentProfile luce así:

using CLAcademy.Domain.Enums;

namespace CLAcademy.Application.DTOs

{

public class CreateStudentDTO

{

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

public string Email { get; set; } = string.Empty;

public StratumType Stratum { get; set; }

}

}

**Paso 26: Ajustar el método Insert en StudentService**

En CLAcademy.Application/Services/StudentService.cs

Cambia el método Insert para que reciba un CreateStudentDTO:

public async Task<Student> Insert(CreateStudentDTO dto)

{

var student = \_mapper.Map<Student>(dto);

return await \_studentRepository.Insert(student);

}

**Paso 27: Modificar el StudentController**

En la clase CLAcademy.Presentation/Controllers/StudentController, cambia el método Create para recibir un CreateStudentDTO:

HttpPost]

public async Task<IActionResult> Create(CreateStudentDTO dto)

{

var created = await \_studentService.Insert(dto);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

**Paso 28: Agregando una validación criolla porque Fluent Validator ahora será pago.**

Crea una nueva carpeta Validators en: CLAcademy.Application.

Dentro de la carpeta CLAcademy.Application.Validators, crea una nueva clase llamada CreateStudentValidator, así:

using CLAcademy.Application.DTOs;

using CLAcademy.Domain.Enums;

using System.Text.RegularExpressions;

namespace CLAcademy.Application.Validators

{

public class CreateStudentValidator

{

public static List<string> Validate(CreateStudentDTO dto)

{

var errors = new List<string>();

if(string.IsNullOrWhiteSpace(dto.FirstName) )

errors.Add("Nombre obligatorio.");

if(string.IsNullOrWhiteSpace(dto.LastName) )

errors.Add("Apellido obligatorio.");

if (!string.IsNullOrWhiteSpace(dto.Email) &&

!Regex.IsMatch(dto.Email, @"^[^@\s]+@[^@\s]+**\.**[^@\s]+$"));

errors.Add($"Email {dto.Email} no válido.");

if(!Enum.IsDefined(typeof(StratumType), dto.Stratum))

errors.Add($"El estrato {dto.Stratum} no existe.");

return errors;

}

}

}

Agrega por último estas líneas de código a la clase StudentServices:

var errors = CreateStudentValidator.Validate(dto);

if (errors.Any())

throw new ValidationException(string.Join("; ", errors));

Ahora esta clase debe lucir así:

using AutoMapper;

using CLAcademy.Application.DTOs;

using CLAcademy.Application.Interfaces;

using CLAcademy.Application.Validators;

using CLAcademy.Domain.Entities;

using System.ComponentModel.DataAnnotations;

namespace CLAcademy.Application.Services

{

public class StudentService

{

private readonly IGenericRepository<Student> \_studentRepository;

private readonly IMapper \_mapper;

public StudentService(IGenericRepository<Student> studentRepository, IMapper mapper)

{

\_studentRepository = studentRepository;

\_mapper = mapper;

}

public async Task<List<StudentDTO>> GetAllAsync()

{

var students = await \_studentRepository.GetAll();

return \_mapper.Map<List<StudentDTO>>(students);

}

public async Task<StudentDTO?> GetById(int id)

{

var student = await \_studentRepository.GetById(id);

return student == null ? null : \_mapper.Map<StudentDTO>(student);

}

// Métodos Update/Delete pueden seguir usando Student directamente

public async Task<Student> Insert(CreateStudentDTO dto)

{

var errors = CreateStudentValidator.Validate(dto);

if (errors.Any())

throw new ValidationException(string.Join("; ", errors));

var student = \_mapper.Map<Student>(dto);

return await \_studentRepository.Insert(student);

}

public async Task<Student> Update(Student student)

{

return await \_studentRepository.Update(student);

}

public async Task DeleteById(int id)

{

await \_studentRepository.DeleteById(id);

}

}

}