

# GUIA DE LABORATORIO

Desarrollo de aplicaciones  
Empresariales Avanzadas

## INFORMACIÓN DEL CURSO

DOCENTE:

ING. JHORDAN LEONARDO

MAYHUA UBILLAS

JMAYHUA@TECSUP.EDU.PE

<https://www.linkedin.com/in/jhordan-mayhua/>

.NET

## Laboratorio 04: Implementación de la Arquitectura de Capas con UnitOfWork y Repositorio

### Objetivos:

El objetivo de este laboratorio es :

- Aplicar los patrones de diseño **UnitOfWork** y **Repositorio** en la arquitectura de una API REST.
- Asegurar transacciones consistentes a través del uso del patrón **UnitOfWork**.
- Integrar estos patrones en un sistema escalable y desacoplado.
- Reforzar el uso de buenas prácticas en la estructuración de los servicios y controladores.

### Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

### Equipos y Materiales:

Una computadora con:

- Windows 10 o superior – Mac Os Ventura o superior
- Conexión a la red del laboratorio
- Rider IDE o Visual Studio 2022
- Base de datos a preferencia de cada estudiante (Postgres, SQL Server, mysql, MariaDB, etc.)

### Procedimiento:

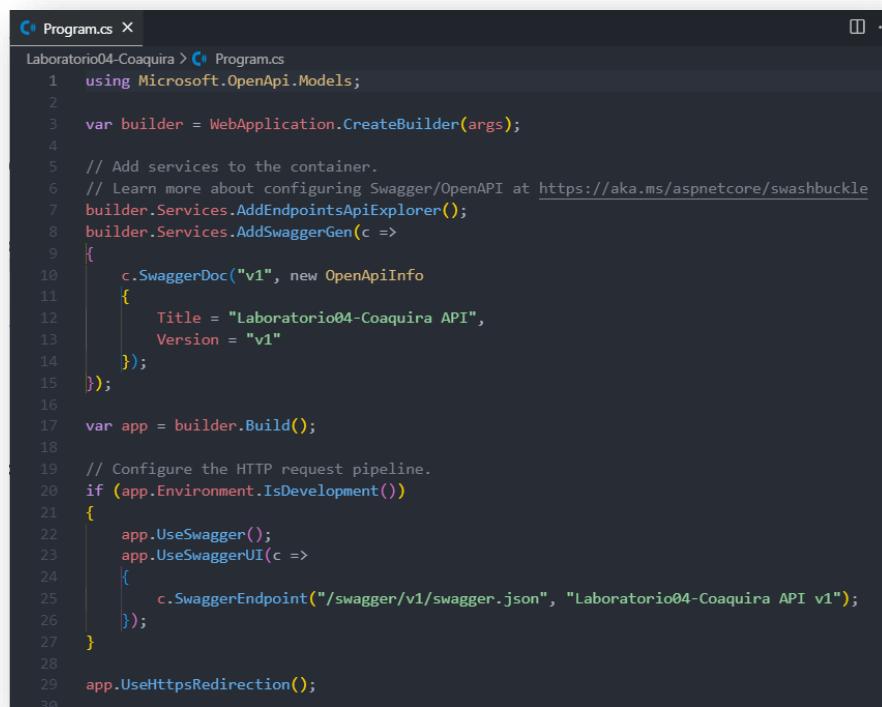
#### Paso 1: Creación de un Proyecto en .NET

- Crea un Proyecto de RestApi.
- Luego Selecciona en Web ASP.NET Core o ASP .NET Core. (Modelo Vista-Controlador)
- Nombra tu proyecto como LABX-TuNombre
- Incluye en tu proyecto nuevo Swagger y configúralo como vimos en las primeras sesiones
- Instala las dependencias para trabajar con EF Core vistas en el laboratorio 3

Captura de pantalla de los archivos de tu solución y despliega los paquetes instalados para verificar este paso:

### Incluyendo Swagger

Se añadieron metadatos(título/versión) y un endpoint explícito para nuestra IU y para la versión en formato JSON



```

 1  using Microsoft.OpenApi.Models;
 2
 3  var builder = WebApplication.CreateBuilder(args);
 4
 5  // Add services to the container.
 6  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
 7  builder.Services.AddEndpointsApiExplorer();
 8  builder.Services.AddSwaggerGen(c =>
 9  {
10      c.SwaggerDoc("v1", new OpenApiInfo
11      {
12          Title = "Laboratorio04-Coaquira API",
13          Version = "v1"
14      });
15  });
16
17  var app = builder.Build();
18
19  // Configure the HTTP request pipeline.
20  if (app.Environment.IsDevelopment())
21  {
22      app.UseSwagger();
23      app.UseSwaggerUI(c =>
24      {
25          c.SwaggerEndpoint("/swagger/v1/swagger.json", "Laboratorio04-Coaquira API v1");
26      });
27  }
28
29  app.UseHttpsRedirection();
30

```

## Instalando dependencias para trabajar con EF Core Tools

```

● ASUS@Beto ~\..\.NetCoreApp\2.1\Temp\dotnet\sdk\2.1.502\NuGet\5.0.1\tools\dotnet.exe add package Microsoft.EntityFrameworkCore.Tools --version 8.0.8
Determining projects to restore...
Writing C:\Users\ASUS\AppData\Local\Temp\tmpdxdq3l.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore.Tools' into project 'C:\Users\ASUS\RiderProjects\Laboratorio04-Coaquira\Laboratorio04-Coaquira\Laboratorio04-Coaquira.csproj'.
info : Restoring packages for C:\Users\ASUS\RiderProjects\Laboratorio04-Coaquira\Laboratorio04-Coaquira\Laboratorio04-Coaquira.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/index.json
info :     OK https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/index.json 828ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/8.0.8/microsoft.entityframeworkcore.tools.8.0.8.nupkg
info :     OK https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/8.0.8/microsoft.entityframeworkcore.tools.8.0.8.nupkg 190ms
info : Installed Microsoft.EntityFrameworkCore.Tools 8.0.8 from https://api.nuget.org/v3/index.json with content hash wjDNbLJk86QpZt2JxJuNVzpBKIBEqsgcJYHGeIFTBuK6NEgvJvyxgneg059HfSJmTVdInZ611T04sJGCffr7+w==.
info :   CACHE https://api.nuget.org/v3/vulnerabilities/index.json
info :   CACHE https://api.nuget.org/v3-vulnerabilities/2025.09.11.05.20.35/vulnerability.base.json

```

# Design

```
Coquira\Laboratorio04-Coquira.csproj (in 7.02 sec).
● ASUS@Beto ~\..\..\Laboratorio04-Coquira dotnet add package Microsoft.EntityFrameworkCore.Design --version 8.0.8
  Determining projects to restore...
  Writing C:\Users\ASUS\AppData\Local\Temp\tmps2lgyx.tmp
  info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
  info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
  info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore.Design' into project 'C:\Users\ASUS\RiderProjects\Laboratorio04-Coquira\Laboratorio04-Coquira\Laboratorio04-Coquira.csproj'.
  info : Restoring packages for C:\Users\ASUS\RiderProjects\Laboratorio04-Coquira\Laboratorio04-Coquira\Laboratorio04-Coquira.csproj...
  info : GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.design/index.json
  info : OK https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.design/index.json 1156ms
  info : GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.design/8.0.8/microsoft.entityframeworkcore.design.8.0.8.nupkg
  info : OK https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.design/8.0.8/microsoft.entityframeworkcore.design.8.0.8.nupkg 203ms
  info : GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.relational/index.json
  info : GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.mysql/index.json
```

# Mysql

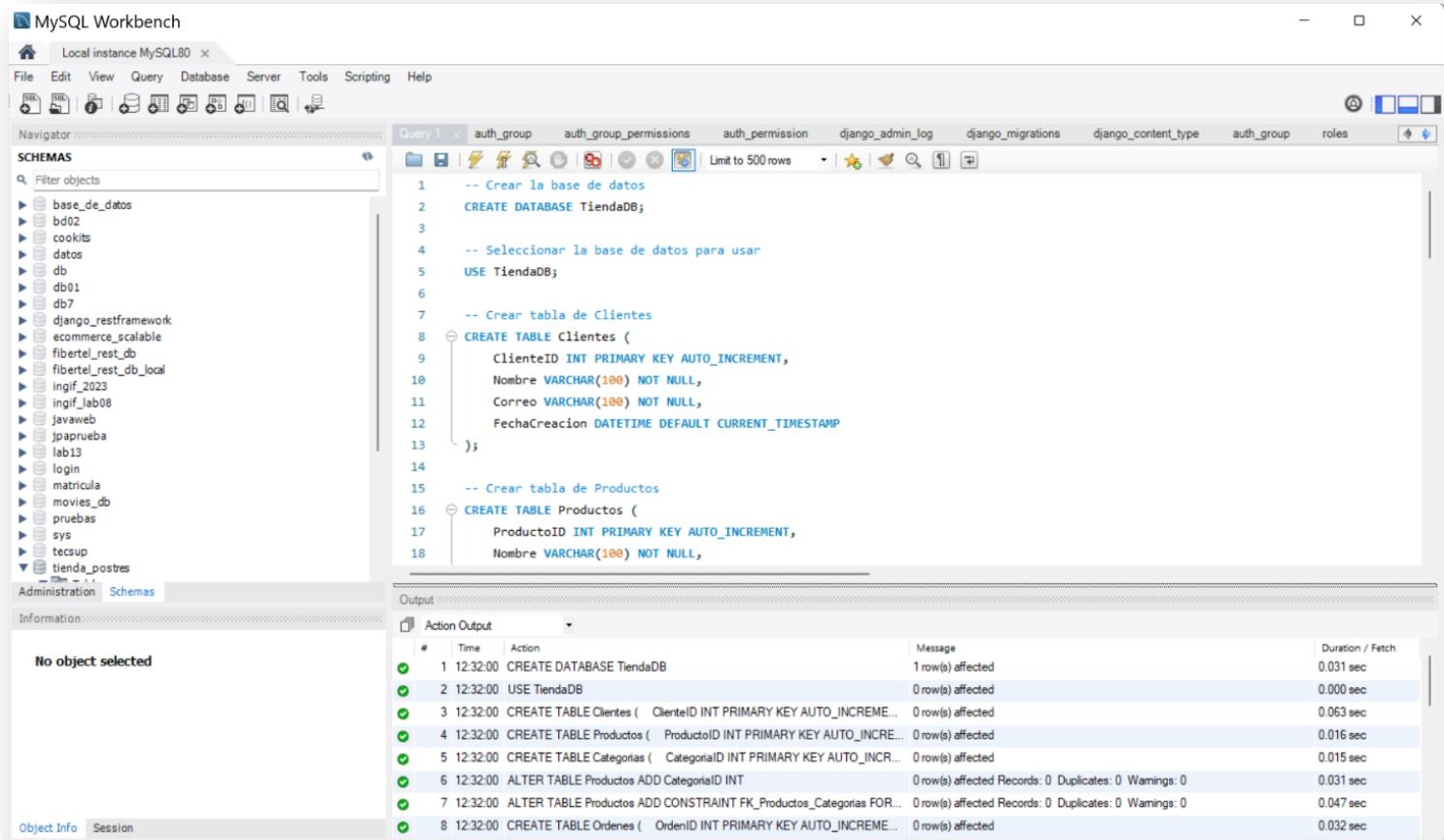
```
● ASUS@Beto ~\..\..\Laboratorio04-Coquira cd C:\Users\ASUS\RiderProjects\Laboratorio04-Coquira\Laboratorio04-Coquira && dotnet add package Pomelo.EntityFrameworkCore.MySql --version 8.0.2
  Determining projects to restore...
  Writing C:\Users\ASUS\AppData\Local\Temp\tmpxzch5y.tmp
  timestamping.
  info : Adding PackageReference for package 'Pomelo.EntityFrameworkCore.MySql' into project 'C:\Users\ASUS\RiderProjects\Laboratorio04-Coquira\Laboratorio04-Coquira\Laboratorio04-Coquira.csproj'.
  info : Restoring packages for C:\Users\ASUS\RiderProjects\Laboratorio04-Coquira\Laboratorio04-Coquira\Laboratorio04-Coquira.csproj...
  info : GET https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/index.json
  info : OK https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/index.json 158ms
  info : GET https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/8.0.2/pomelo.entityframeworkcore.mysql.8.0.2.nupkg
  info : OK https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/8.0.2/pomelo.entityframeworkcore.mysql.8.0.2.nupkg 177ms
  info : GET https://api.nuget.org/v3-flatcontainer/mysqlconnector/index.json
  info : OK https://api.nuget.org/v3-flatcontainer/mysqlconnector/index.json 193ms
```

## Paso 2: Configurar la Base de Datos

1. Descargar la base de datos de prueba:
  - En Canvas, encontrarás los archivos SQL para cada base de datos compatible: MySQL, PostgreSQL y SQL Server.
  - Descarga el archivo SQL correspondiente a la base de datos que vas a utilizar.
2. Seleccionar la base de datos de tu preferencia:
  - Si eliges MySQL, descarga el archivo TiendaDB\_MySQL.sql.
  - Si eliges PostgreSQL, descarga el archivo TiendaDB\_PostgreSQL.sql.
  - Si eliges SQL Server, descarga el archivo TiendaDB\_SQLServer.sql.
3. Ejecutar el archivo SQL:
  - Abre tu cliente de base de datos preferido (como MySQL Workbench, pgAdmin para PostgreSQL o SQL Server Management Studio para SQL Server).
  - Crea una nueva base de datos en tu cliente. Puedes nombrarla TiendaDB o cualquier nombre que prefieras.
  - Ejecuta el archivo SQL descargado en la base de datos que acabas de crear. Este archivo generará todas las tablas, relaciones y datos de prueba necesarios para realizar las consultas y operaciones de la práctica.
4. Verificar la creación de las tablas:
  - Despues de ejecutar el archivo, revisa que todas las tablas hayan sido creadas correctamente y que contengan los datos de prueba. Asegúrate de que las relaciones entre las tablas (como las Foreign Keys) estén correctamente configuradas.

Captura de Pantalla: (*Toma una captura de pantalla de la consola o del cliente de base de datos mostrando que la base de datos y sus tablas han sido creadas correctamente. Asegúrate de incluir la verificación de las tablas Clientes, Productos, Categorías, Órdenes, DetallesOrden, y Pagos.*)

## Comprobacion:



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Shows the list of databases available on the local instance MySQL80. The 'tienda\_postres' database is currently selected.
- Query Editor:** Displays the SQL code used to create the database and its tables. The code is as follows:

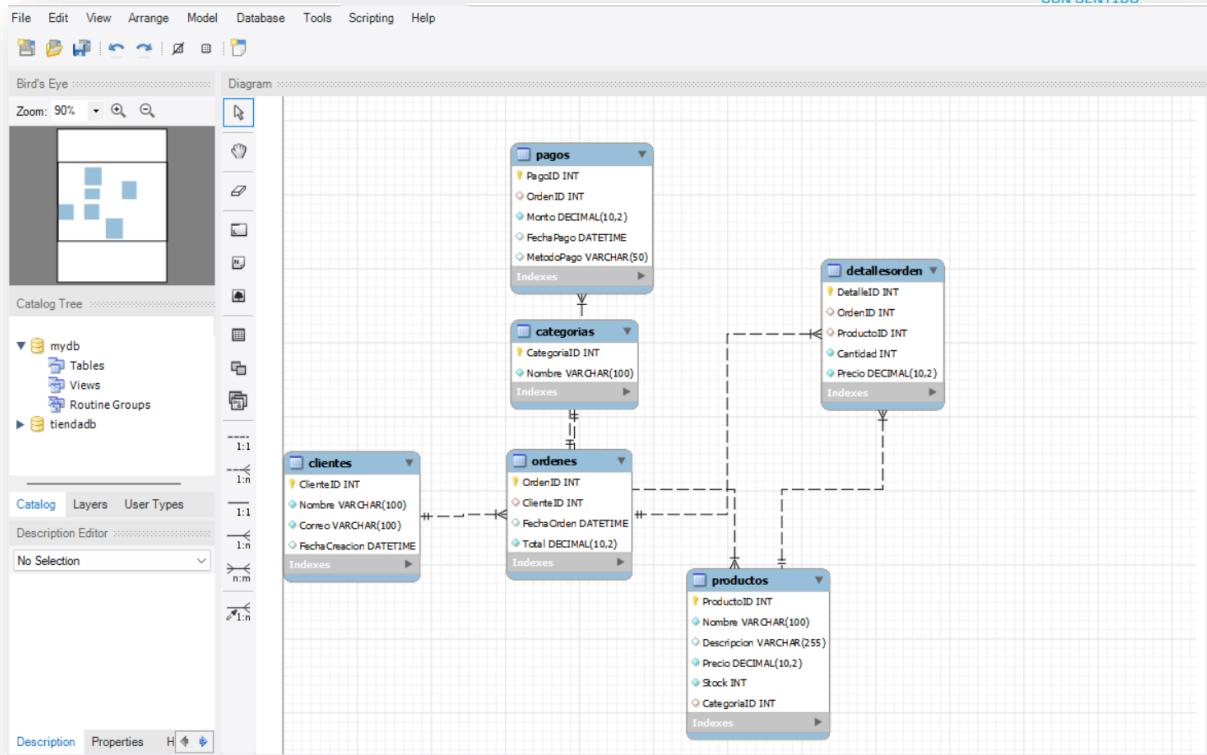
```

1 -- Crear la base de datos
2 CREATE DATABASE TiendaDB;
3
4 -- Seleccionar la base de datos para usar
5 USE TiendaDB;
6
7 -- Crear tabla de Clientes
8 CREATE TABLE Clientes (
9     ClienteID INT PRIMARY KEY AUTO_INCREMENT,
10    Nombre VARCHAR(100) NOT NULL,
11    Correo VARCHAR(100) NOT NULL,
12    FechaCreacion DATETIME DEFAULT CURRENT_TIMESTAMP
13 );
14
15 -- Crear tabla de Productos
16 CREATE TABLE Productos (
17     ProductoID INT PRIMARY KEY AUTO_INCREMENT,
18     Nombre VARCHAR(100) NOT NULL,

```

- Output:** Shows the results of the executed queries, detailing the action, time, message, duration, and fetch count.

#	Time	Action	Message	Duration / Fetch
1	12:32:00	CREATE DATABASE TiendaDB	1 row(s) affected	0.031 sec
2	12:32:00	USE TiendaDB	0 row(s) affected	0.000 sec
3	12:32:00	CREATE TABLE Clientes ( ClienteID INT PRIMARY KEY AUTO_INCREMENT,	0 row(s) affected	0.063 sec
4	12:32:00	Nombre VARCHAR(100) NOT NULL,	0 row(s) affected	0.016 sec
5	12:32:00	Correo VARCHAR(100) NOT NULL,	0 row(s) affected	0.015 sec
6	12:32:00	FechaCreacion DATETIME DEFAULT CURRENT_TIMESTAMP		
7	12:32:00	);		
8	12:32:00	CREATE TABLE Productos ( ProductoID INT PRIMARY KEY AUTO_INCREMENT,	0 row(s) affected	0.031 sec
9	12:32:00	Nombre VARCHAR(100) NOT NULL,	0 row(s) affected	0.047 sec
10	12:32:00	ALTER TABLE Productos ADD CategoriaID INT	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
11	12:32:00	ALTER TABLE Productos ADD CONSTRAINT FK_Productos_Categorias FOREIGN KEY (CategoriaID) REFERENCES Categorias (CategoriaID)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec
12	12:32:00	CREATE TABLE Ordenes ( OrdenID INT PRIMARY KEY AUTO_INCREMENT,	0 row(s) affected	0.032 sec



	CategoríaID	Nombre
▶	1	Electrónica
	2	Ropa
	3	Alimentos
*	HULL	HULL

Result Grid   Filter Rows:   Edit:   Export/Import			
	ClienteID	Nombre	Correo
▶	1	Juan Perez	juan.perez@email.com
	2	Maria Lopez	maria.lopez@email.com
	3	Carlos Garcia	carlos.garcia@email.com
*	HULL	HULL	HULL

Result Grid   Filter Rows:   Edit:   Export/Import					
	DetalleID	OrdenID	ProductoID	Cantidad	Precio
▶	1	1	1	1	750.00
	2	1	2	1	25.00
	3	2	2	1	25.00
*	HULL	HULL	HULL	HULL	HULL

Result Grid   Filter Rows:   Edit:   Export/Import			
	OrdenID	ClienteID	FechaOrden
▶	1	1	2025-09-11 12:32:00
	2	2	2025-09-11 12:32:00
*	HULL	HULL	HULL

Result Grid   Filter Rows:   Edit:   Export/Import					
	PagoID	OrdenID	Monto	FechaPago	MetodoPago
▶	1	1	775.00	2025-09-11 12:32:00	Tarjeta de Crédito
	2	2	30.00	2025-09-11 12:32:00	Efectivo
*	HULL	HULL	HULL	HULL	HULL

Result Grid   Filter Rows:   Edit:   Export/Import						
	ProductoID	Nombre	Descripcion	Precio	Stock	CategoríaID
▶	1	Laptop	Laptop con 8GB de RAM	750.00	10	1
	2	Camisa	Camisa de algodón	25.00	50	2
	3	Cereal	Cereal de avena	5.00	100	3
*	HULL	HULL	HULL	HULL	HULL	3

## Paso 3: Generación de Modelos y DBContext

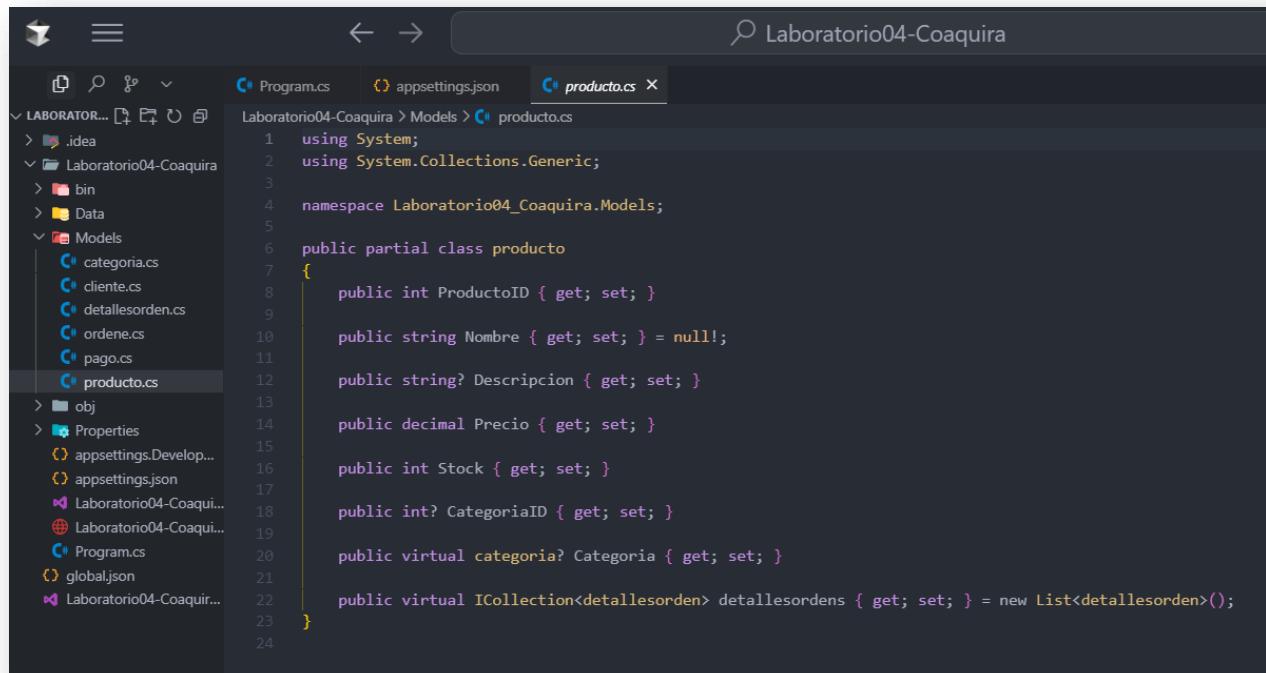
1. Agrega las dependencias necesarias y la configuración en el Program.cs para ejecutar el comando scaffold visto en la sesión 3.
2. Después de ejecutar el comando toma una captura de pantalla a tu carpeta donde generaste los modelos y el dbContext:

```
ASUS@Beto ~\..\.Laboratorio04-Coaquir> dotnet tool update --global dotnet-e
Skipping NuGet package signature verification.
Tool 'dotnet-e' was successfully updated from version '9.0.8' to v
ersion '9.0.9'.
ASUS@Beto ~\..\.Laboratorio04-Coaquir>
```

Ejecutamos el comando para la migracion inversa o scaffold

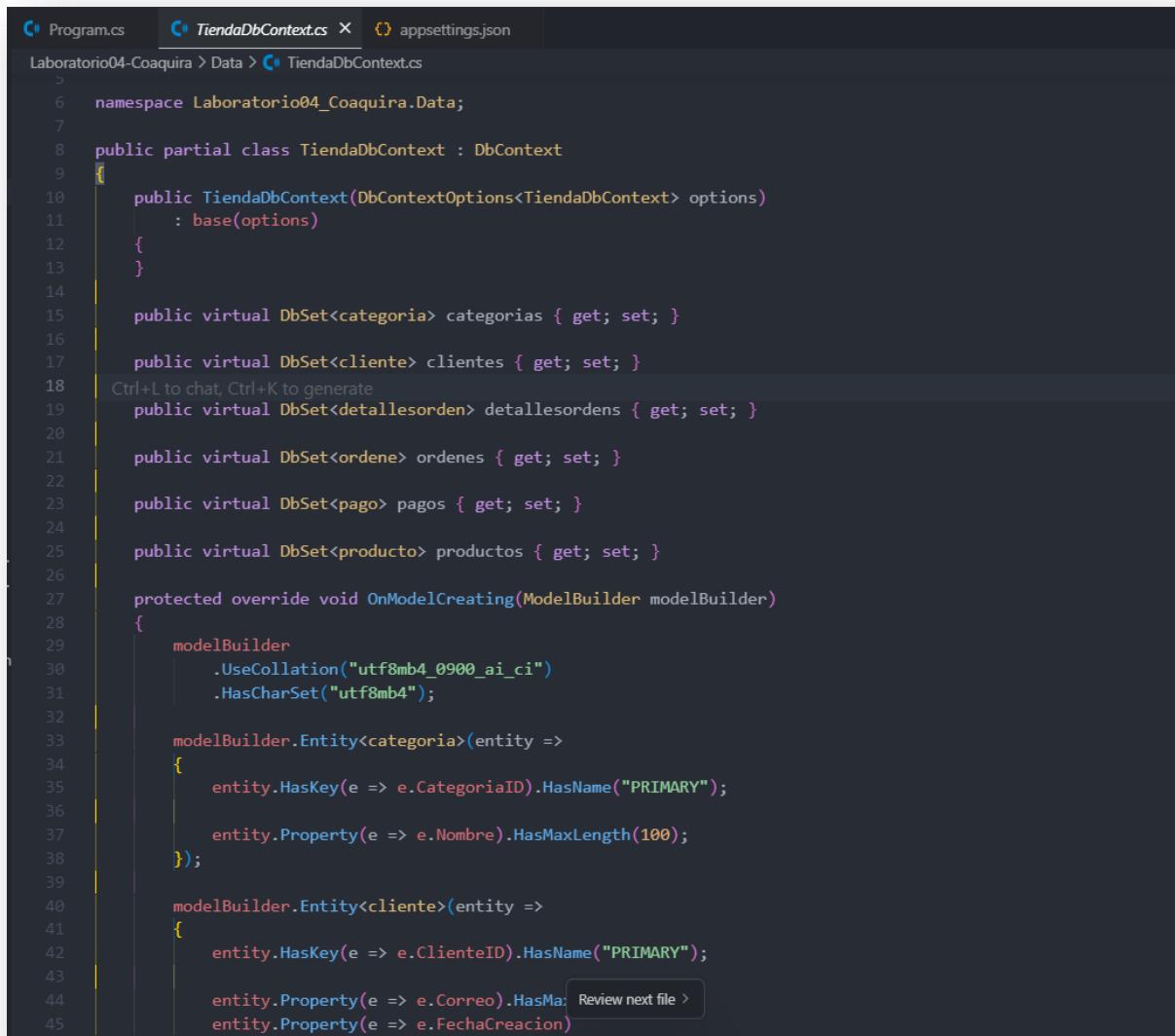
```
ASUS@Beto ~\..\.Laboratorio04-Coaquir> dotnet ef dbcontext scaffold "server=localhost;port=3306;database=tiendadb;user=root;password=tecsup2023;TreatTinyAsBoolean=false;SslMode=None;" Pomelo.EntityFrameworkCore.MySql --context TiendaDbContext --context-dir Data --output-dir Models --use-database-names --no-onconfiguring
Build started...
Build succeeded.
Using ServerVersion '8.0.34-mysql'.
```

## Resultado:



```
using System;
using System.Collections.Generic;
namespace Laboratorio04_Coaquir.Models;
public partial class producto
{
    public int ProductoID { get; set; }
    public string Nombre { get; set; } = null!;
    public string? Descripcion { get; set; }
    public decimal Precio { get; set; }
    public int Stock { get; set; }
    public int? CategoriaID { get; set; }
    public virtual categoria? Categoria { get; set; }
    public virtual ICollection<detallesorden> detallesordens { get; set; } = new List<detallesorden>();
}
```

# DdContext:



```

 1  namespace Laboratorio04_Coaquira > Data > TiendaDbContext.cs
 2
 3  public partial class TiendaDbContext : DbContext
 4  {
 5      public TiendaDbContext(DbContextOptions<TiendaDbContext> options)
 6          : base(options)
 7      {
 8      }
 9
10
11      public virtual DbSet<categoria> categorias { get; set; }
12
13      public virtual DbSet<cliente> clientes { get; set; }
14
15      public virtual DbSet<detallesorden> detallesordens { get; set; }
16
17      public virtual DbSet<ordene> ordenes { get; set; }
18
19      public virtual DbSet<pago> pagos { get; set; }
20
21      public virtual DbSet<producto> productos { get; set; }
22
23
24      protected override void OnModelCreating(ModelBuilder modelBuilder)
25  {
26
27          modelBuilder
28              .UseCollation("utf8mb4_0900_ai_ci")
29              .HasCharSet("utf8mb4");
30
31
32          modelBuilder.Entity<categoria>(entity =>
33          {
34              entity.HasKey(e => e.CategoríaID).HasName("PRIMARY");
35
36              entity.Property(e => e.Nombre).HasMaxLength(100);
37          });
38
39
40          modelBuilder.Entity<cliente>(entity =>
41          {
42              entity.HasKey(e => e.ClienteID).HasName("PRIMARY");
43
44              entity.Property(e => e.Correo).HasMa: Review next file >
45              entity.Property(e => e.FechaCreacion)
        
```

## Paso 4: Generación de Repositorios para conexión de base de datos

1. Crea el repositorio de Cliente:
  - 1.1. la implementación ClienteRepository

```

public class ClienteRepository : IClienteRepository
{
    private readonly DbContext _context;
    public ClienteRepository(DbContext context)
    {
        _context = context;
    }

    public Cliente GetById(int id)
    {
        return _context.Set<Cliente>().Find(id);
    }

    public IEnumerable<Cliente> GetAll()
    {
        return _context.Set<Cliente>().ToList();
    }

    public void Add(Cliente cliente)
    {
        _context.Set<Cliente>().Add(cliente);
    }

    public void Update(Cliente cliente)
    {
        _context.Set<Cliente>().Update(cliente);
    }

    public void Delete(int id)
    {
        var cliente = _context.Set<Cliente>().Find(id);
        if (cliente != null)
        {
            _context.Set<Cliente>().Remove(cliente);
        }
    }
}
        
```

## 1.2. y la interfaz ICliente Repository

```
public interface IClienteRepository
{
    Cliente GetById(int id);
    IEnumerable<Cliente> GetAll();
    void Add(Cliente cliente);
    void Update(Cliente cliente);
    void Delete(int id);
}
```

2. Ahora crea de igual forma UnitOfWork con su interfaz e implementación

```
public interface IUnitOfWork : IDisposable
{
    IClienteRepository Clientes { get; }
    int SaveChanges();
}
```

```
public class UnitOfWork : IUnitOfWork
{
    private readonly DbContext _context;
    public IClienteRepository Clientes { get; }

    public UnitOfWork(DbContext context, IClienteRepository clienteRepository)
    {
        _context = context;
        Clientes = clienteRepository;
    }

    public int SaveChanges()
    {
        return _context.SaveChanges();
    }

    public void Dispose()
    {
        _context.Dispose();
    }
}
```

## Paso 5: Uso del patrón Unit of Work en un servicio o controlador

En el controlador o servicio de la aplicación, podemos utilizar el patrón Unit of Work para realizar operaciones en la base de datos.

```
public class ClienteController : Controller
{
    private readonly IUnitOfWork _unitOfWork;

    public ClienteController(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }

    public IActionResult CrearCliente(Cliente cliente)
    {
        _unitOfWork.Cuentas.Add(cliente);
        _unitOfWork.SaveChanges();
        return Ok("Cliente creado con éxito");
    }

    public IActionResult ObtenerClientes()
    {
        var clientes = _unitOfWork.Cuentas.GetAll();
        return View(clientes);
    }
}
```

## Nota

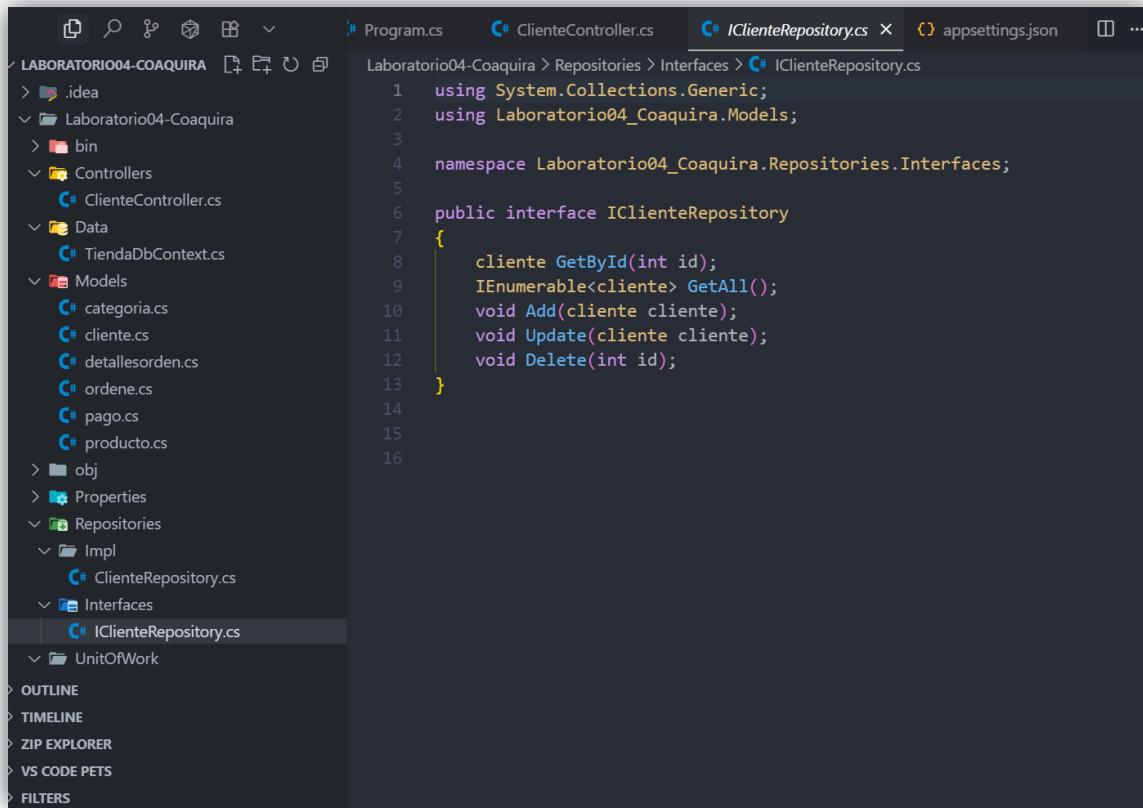
*Cadena de conexión siempre se define en appsettings.json (lo registre en el archivo main ósea el programs.css lo cual estuvo mal)*

## Explicación

- **Repositorio:** Cada repositorio (como `IClienteRepository`) encapsula el acceso a la base de datos para una entidad específica (`Cliente`).
- **Unit of Work:** La clase `UnitOfWork` actúa como un “contorno” de las transacciones, gestionando los repositorios y asegurando que todas las operaciones se guarden correctamente al finalizar la transacción.
- **Controlador:** El controlador de ASP.NET utiliza `IUnitOfWork` para acceder a los repositorios y realizar las operaciones de manera transaccional.

# Adjunta captura de Pantalla de tus archivos:

## Interfaz de Cliente



```

    # Program.cs      C# ClienteController.cs      C# IClienteRepository.cs      appsettings.json      ...
    LABORATORIO04-COAQUIRA      Laboratorio04-Coaquiria > Repositories > Interfaces > C# IClienteRepository.cs
    .idea      ClienteController.cs      IClienteRepository.cs      ...
    bin      Data      Models      ...
    Controllers      TiendaDbContext.cs      categoria.cs      cliente.cs      ...
    Data      detallesorden.cs      ordene.cs      pago.cs      producto.cs
    Models      ...
    obj      Properties      ...
    Repositories      ...
    Impl      ClienteRepository.cs      ...
    Interfaces      ...
    IClienteRepository.cs      ...
    UnitOfWork      ...
    OUTLINE      ...
    TIMELINE      ...
    ZIP EXPLORER      ...
    VS CODE PETS      ...
    FILTERS      ...
  
```

The screenshot shows the Visual Studio IDE interface. The left sidebar displays the project structure for 'LABORATORIO04-COAQUIRA'. The 'Repositories' folder contains 'Interfaces' and 'Impl' subfolders. The 'Interfaces' folder has a file named 'IClienteRepository.cs'. The right pane shows the code for this interface, which defines a public interface with methods for getting a client by ID, getting all clients, adding a client, updating a client, and deleting a client.

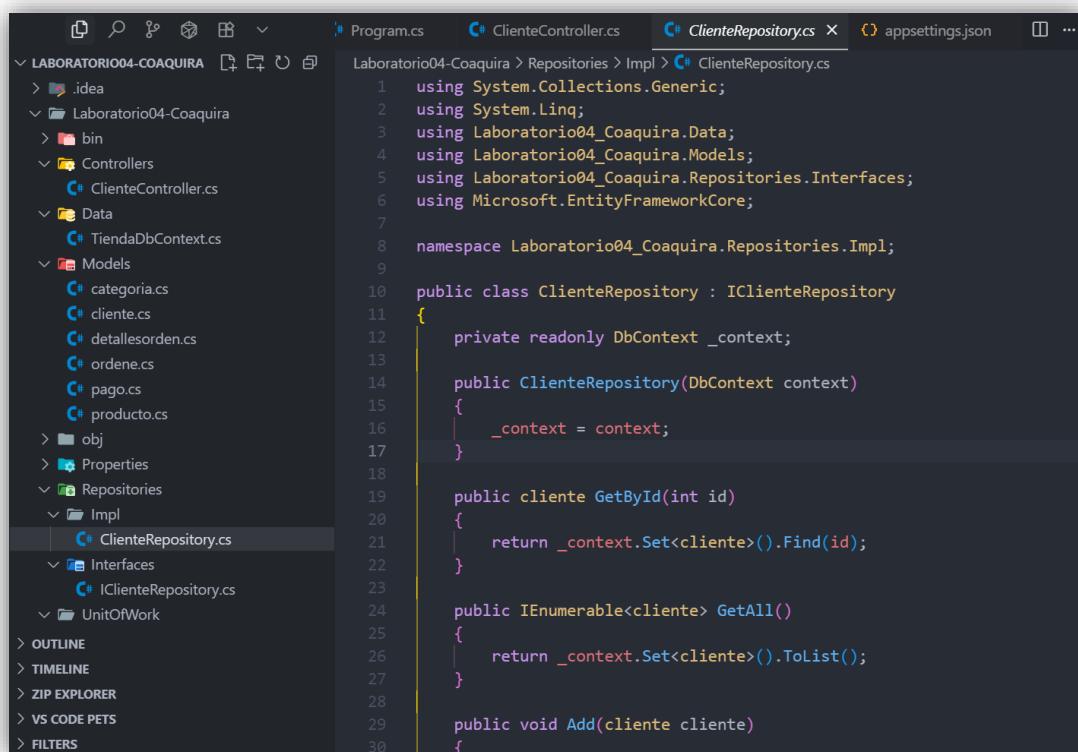
```

using System.Collections.Generic;
using Laboratorio04_Coaquia.Models;

namespace Laboratorio04_Coaquia.Repositories.Interfaces;

public interface IClienteRepository
{
    cliente GetById(int id);
    IEnumerable<cliente> GetAll();
    void Add(cliente cliente);
    void Update(cliente cliente);
    void Delete(int id);
}
  
```

## Implementación de Cliente



```

    # Program.cs      C# ClienteController.cs      C# ClienteRepository.cs      appsettings.json      ...
    LABORATORIO04-COAQUIRA      Laboratorio04-Coaquiria > Repositories > Impl > C# ClienteRepository.cs
    .idea      ClienteController.cs      ...
    bin      Data      Models      ...
    Controllers      ...
    Data      TiendaDbContext.cs      ...
    Models      ...
    obj      Properties      ...
    Repositories      ...
    Impl      ClienteRepository.cs      ...
    Interfaces      ...
    IClienteRepository.cs      ...
    UnitOfWork      ...
    OUTLINE      ...
    TIMELINE      ...
    ZIP EXPLORER      ...
    VS CODE PETS      ...
    FILTERS      ...
  
```

The screenshot shows the Visual Studio IDE interface. The left sidebar displays the project structure for 'LABORATORIO04-COAQUIRA'. The 'Repositories' folder contains 'Impl' subfolder, which contains a file named 'ClienteRepository.cs'. This file implements the 'IClienteRepository' interface defined in the previous screenshot. The code uses Entity Framework Core to interact with the database.

```

using System.Collections.Generic;
using System.Linq;
using Laboratorio04_Coaquia.Data;
using Laboratorio04_Coaquia.Models;
using Laboratorio04_Coaquia.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;

namespace Laboratorio04_Coaquia.Repositories.Impl;

public class ClienteRepository : IClienteRepository
{
    private readonly DbContext _context;

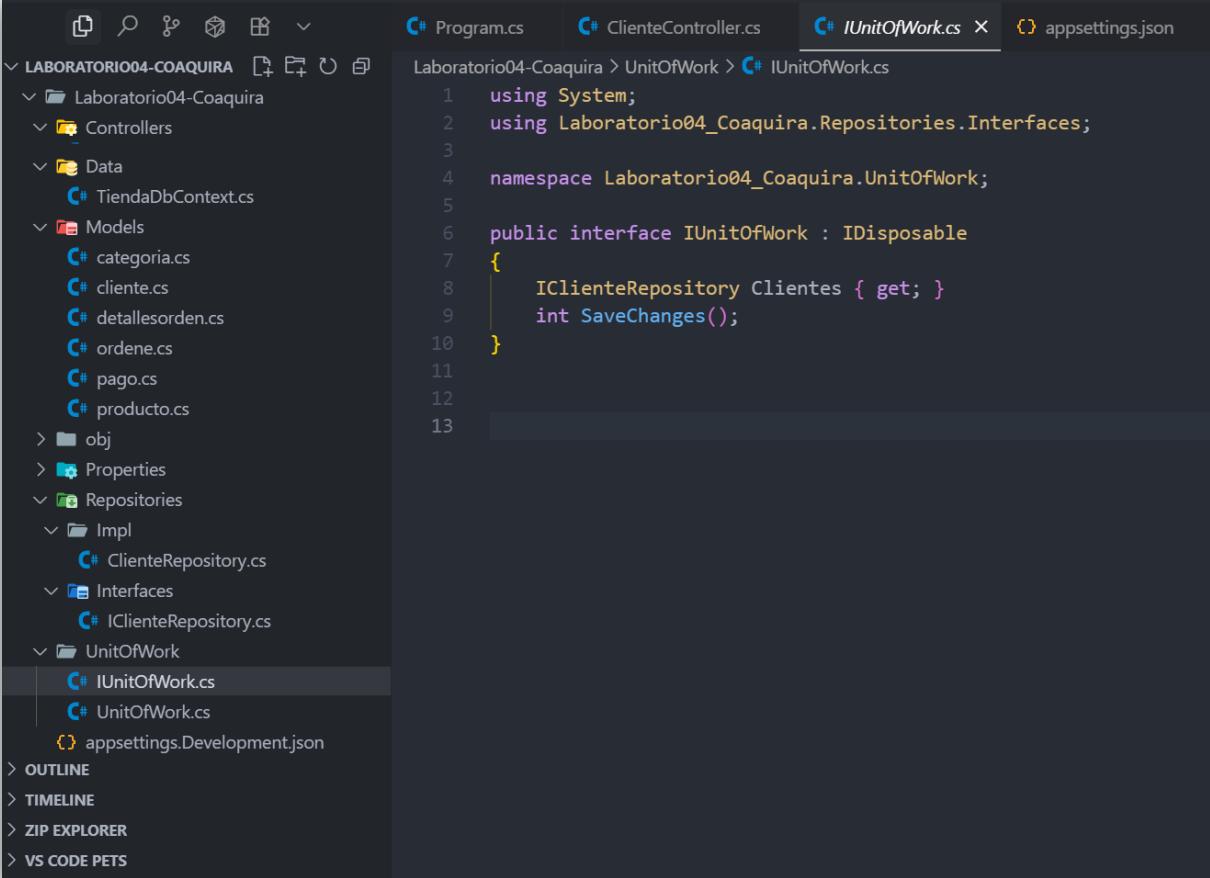
    public ClienteRepository(DbContext context)
    {
        _context = context;
    }

    public cliente GetById(int id)
    {
        return _context.Set<cliente>().Find(id);
    }

    public IEnumerable<cliente> GetAll()
    {
        return _context.Set<cliente>().ToList();
    }

    public void Add(cliente cliente)
    {
    }
}
  
```

# Interfaz de UnitOfWork



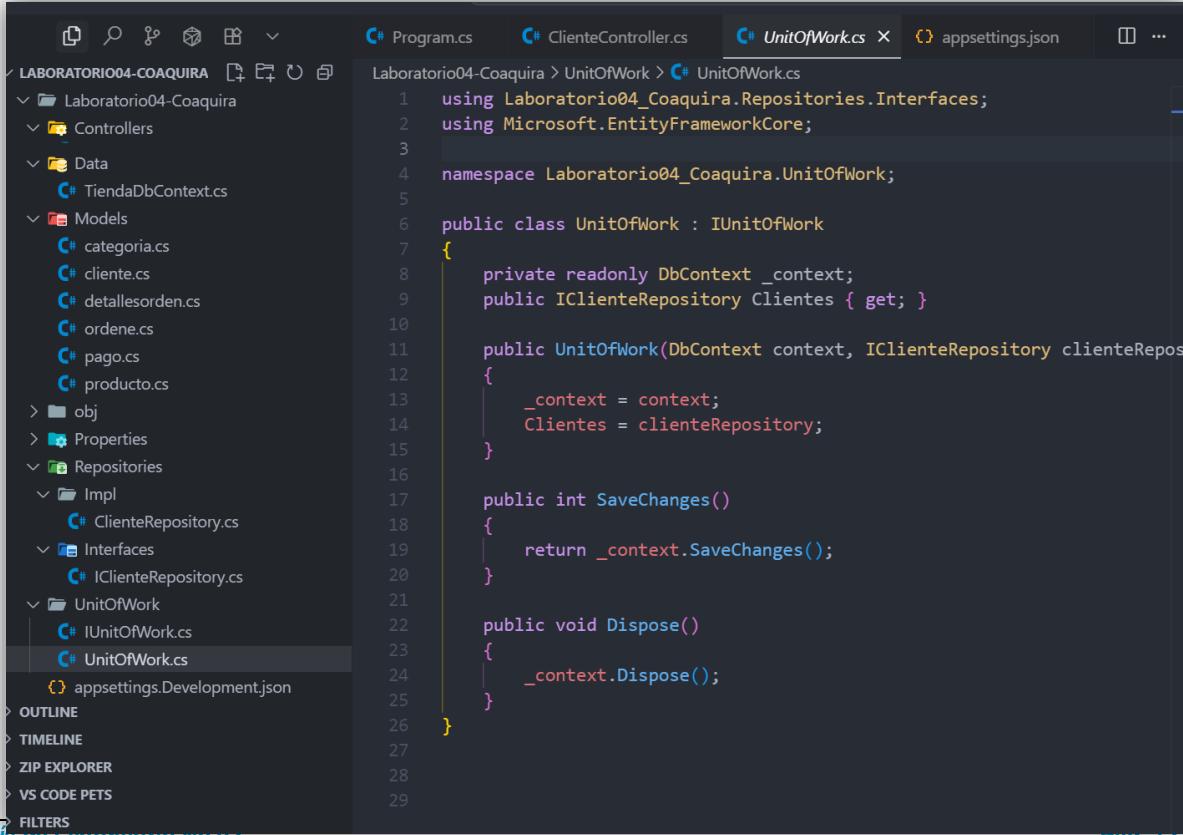
```

    C# Program.cs      C# ClienteController.cs      C# IUnitOfWork.cs      C# appsettings.json
Laboratorio04-Coaquir > UnitOfWork > C# IUnitOfWork.cs
1  using System;
2  using Laboratorio04_Coaquir.Repositories.Interfaces;
3
4  namespace Laboratorio04_Coaquir.UnitOfWork;
5
6  public interface IUnitOfWork : IDisposable
7  {
8      IClienteRepository Clientes { get; }
9      int SaveChanges();
10 }
11
12
13

```

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays the project structure for 'LABORATORIO04-COAQUIRA'. The 'UnitOfWork' folder contains 'IUnitOfWork.cs' and 'UnitOfWork.cs'. The 'IUnitOfWork.cs' file is currently open in the code editor, showing its interface definition. The 'UnitOfWork.cs' file is also listed in the tabs above. The code editor has syntax highlighting for C# and shows line numbers from 1 to 13.

# Implementación de UnitOfWork



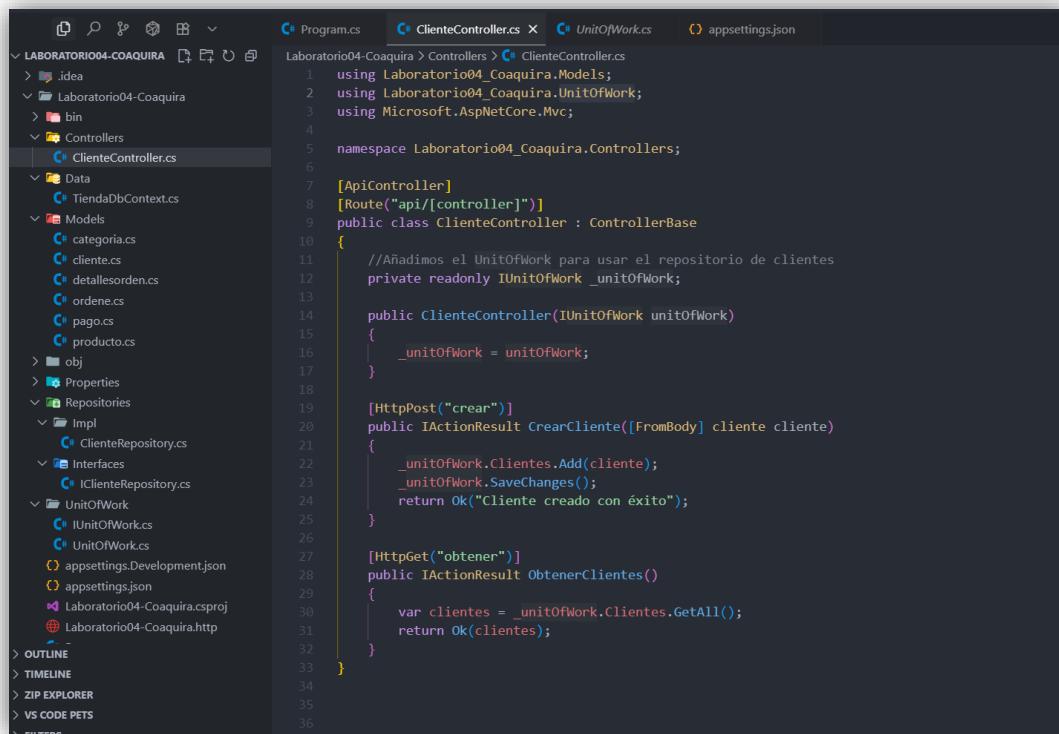
```

    C# Program.cs      C# ClienteController.cs      C# UnitOfWork.cs      C# appsettings.json
Laboratorio04-Coaquir > UnitOfWork > C# UnitOfWork.cs
1  using Laboratorio04_Coaquir.Repositories.Interfaces;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Laboratorio04_Coaquir.UnitOfWork;
5
6  public class UnitOfWork : IUnitOfWork
7  {
8      private readonly DbContext _context;
9      public IClienteRepository Clientes { get; }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

```

The screenshot shows the Visual Studio IDE interface. The project structure is identical to the previous screenshot. The 'UnitOfWork' folder now contains both 'IUnitOfWork.cs' and 'UnitOfWork.cs'. The 'UnitOfWork.cs' file is open in the code editor, showing its implementation. The code defines a class 'UnitOfWork' that implements the 'IUnitOfWork' interface. It includes a private field '\_context' of type 'DbContext', a public property 'Clientes' of type 'IClienteRepository', and methods for saving changes and disposing of the context. The code editor shows line numbers from 1 to 29.

# Controlador de Cliente:



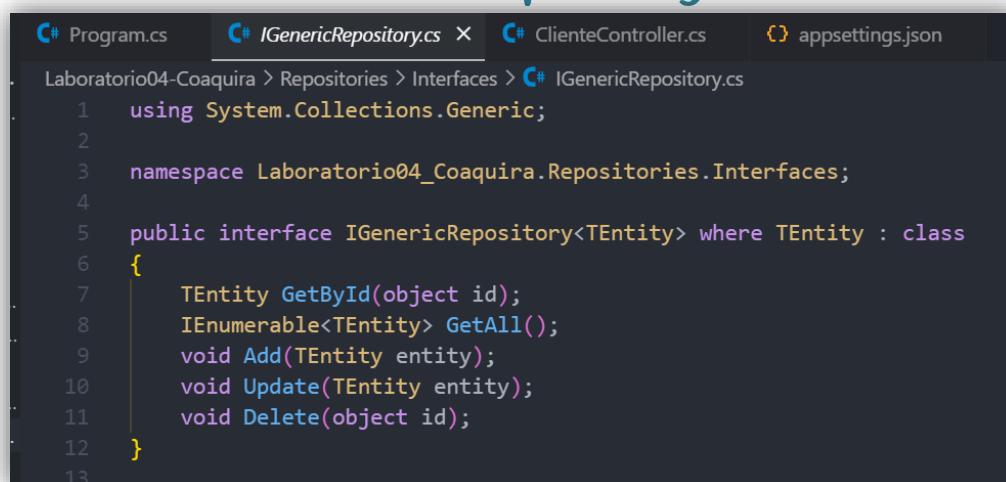
```

    ClientController.cs
    1  using Laboratorio04_Coaquia.Models;
    2  using Laboratorio04_Coaquia.UnitOfWork;
    3  using Microsoft.AspNetCore.Mvc;
    4
    5  namespace Laboratorio04_Coaquia.Controllers;
    6
    7  [ApiController]
    8  [Route("api/{controller}")]
    9  public class ClienteController : ControllerBase
    10 {
    11     //Añadimos el UnitOfWork para usar el repositorio de clientes
    12     private readonly IUnitOfWork _unitOfWork;
    13
    14     public ClienteController(IUnitOfWork unitOfWork)
    15     {
    16         _unitOfWork = unitOfWork;
    17     }
    18
    19     [HttpPost("crear")]
    20     public IActionResult CrearCliente([FromBody] cliente cliente)
    21     {
    22         _unitOfWork.Clientes.Add(cliente);
    23         _unitOfWork.SaveChanges();
    24         return Ok("Cliente creado con éxito");
    25     }
    26
    27     [HttpGet("obtener")]
    28     public IActionResult ObtenerClientes()
    29     {
    30         var clientes = _unitOfWork.Clientes.GetAll();
    31         return Ok(clientes);
    32     }
    33
    34
    35
    36
  
```

## Paso 6: Ejercicio de Aplicación

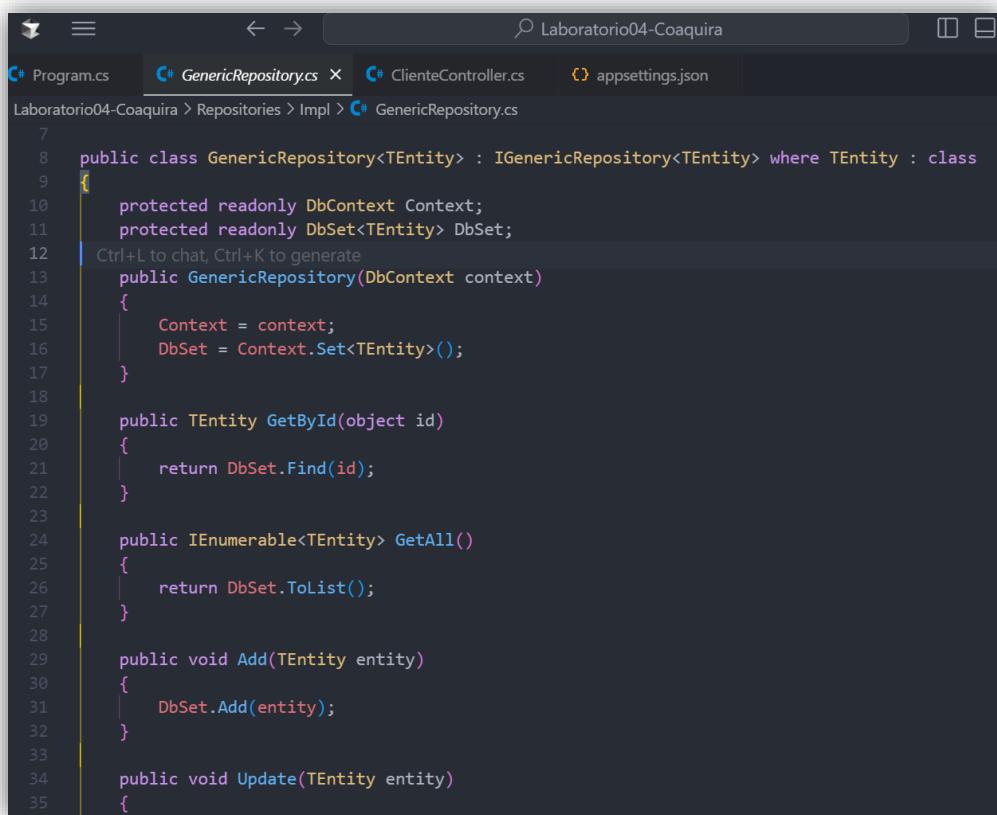
- Como evidencia, crea el CRUD completo de las entidades del SQL trabajadas durante el laboratorio
- Genera la demostración con capturas de Swagger y la inserción a la base de datos.
- Si deseas agregar más campos a alguna entidad para demostrar o practicar, no dudes en hacerlo.
- Puedes utilizar opcionalmente el modelo de repositorio genérico que vimos en la sesión 3

**Creamos un repositorio genérico con interfaz e implementación `IGenericRepository` y `GenericRepository`**



```

    IGenericRepository.cs
    1  using System.Collections.Generic;
    2
    3  namespace Laboratorio04_Coaquia.Repositories.Interfaces;
    4
    5  public interface IGenericRepository<TEntity> where TEntity : class
    6  {
    7      TEntity GetById(object id);
    8      IEnumerable<TEntity> GetAll();
    9      void Add(TEntity entity);
   10      void Update(TEntity entity);
   11      void Delete(object id);
   12  }
   13
  
```

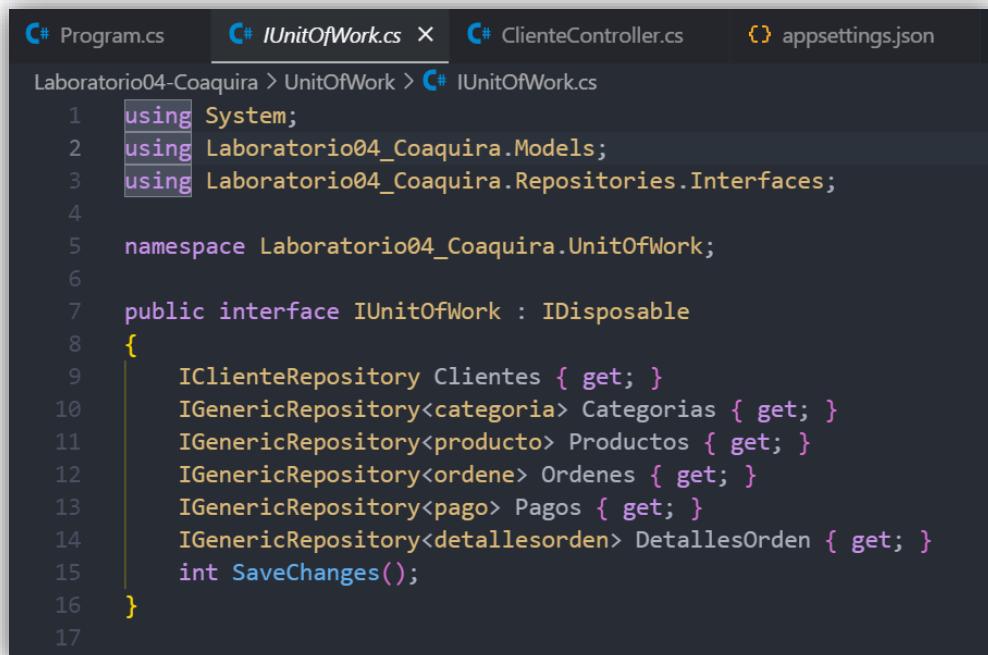


```

7
8     public class GenericRepository<TEntity> : IGenericRepository<TEntity> where TEntity : class
9     {
10         protected readonly DbContext Context;
11         protected readonly DbSet<TEntity> DbSet;
12         Ctrl+L to chat, Ctrl+K to generate
13         public GenericRepository(DbContext context)
14         {
15             Context = context;
16             DbSet = Context.Set<TEntity>();
17         }
18
19         public TEntity GetById(object id)
20         {
21             return DbSet.Find(id);
22         }
23
24         public IEnumerable<TEntity> GetAll()
25         {
26             return DbSet.ToList();
27         }
28
29         public void Add(TEntity entity)
30         {
31             DbSet.Add(entity);
32         }
33
34         public void Update(TEntity entity)
35         {

```

## Extendemos UnitOfWork para exponer repos genericos por entidad



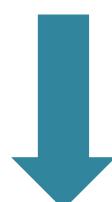
```

1     using System;
2     using Laboratorio04_Coaquira.Models;
3     using Laboratorio04_Coaquira.Repositories.Interfaces;
4
5     namespace Laboratorio04_Coaquira.UnitOfWork;
6
7     public interface IUnitOfWork : IDisposable
8     {
9         IClienteRepository Clientes { get; }
10        IGenericRepository<categoria> Categorias { get; }
11        IGenericRepository<producto> Productos { get; }
12        IGenericRepository<ordene> Ordenes { get; }
13        IGenericRepository<pago> Pagos { get; }
14        IGenericRepository<detallesorden> DetallesOrden { get; }
15        int SaveChanges();
16    }
17

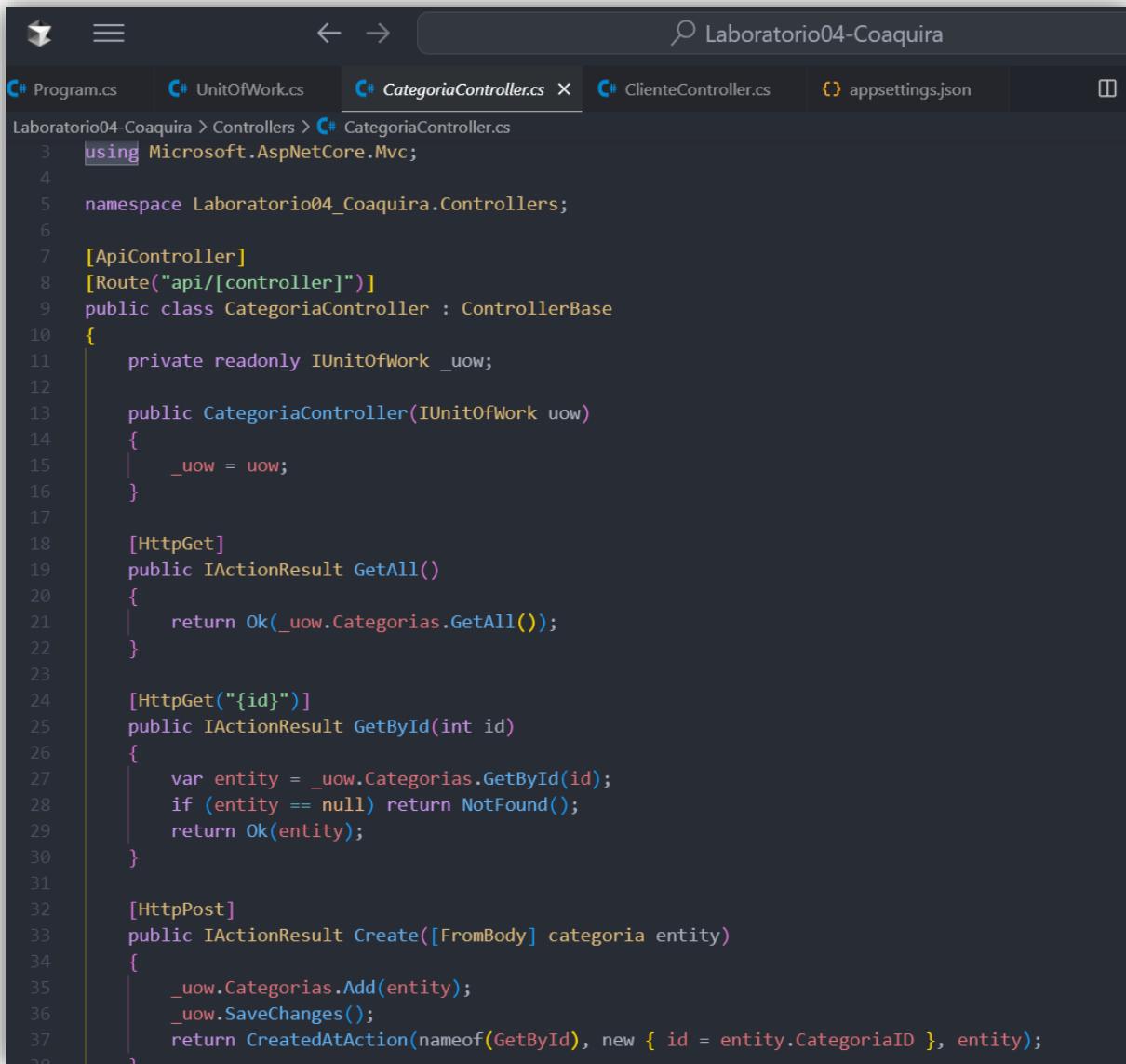
```

```
C# Program.cs      C# UnitOfWork.cs X  C# ClienteController.cs    appsettings.json
Laboratorio04-Coaquira > UnitOfWork > C# UnitOfWork.cs
1  using Laboratorio04_Coaquira.Repositories;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Laboratorio04_Coaquira.UnitOfWork;
5
6  public class UnitOfWork : IUnitOfWork
7  {
8      private readonly DbContext _context;
9
10     public IClienteRepository Clientes { get; }
11     public IGenericRepository<categoria> Categorias { get; }
12     public IGenericRepository<producto> Productos { get; }
13     public IGenericRepository<ordene> Ordenes { get; }
14     public IGenericRepository<pago> Pagos { get; }
15     public IGenericRepository<detallesorden> DetallesOrden { get; }
16
17
18     public UnitOfWork(DbContext context, IClienteRepository clienteRepository)
19     {
20         _context = context;
21         Clientes = clienteRepository;
22         Categorias = new GenericRepository<categoria>(_context);
23         Productos = new GenericRepository<producto>(_context);
24         Ordenes = new GenericRepository<ordene>(_context);
25         Pagos = new GenericRepository<pago>(_context);
26         DetallesOrden = new GenericRepository<detallesorden>(_context);
27     }
28
29     public int SaveChanges()
30     {
31         return _context.SaveChanges();
32     }
33
34     public void Dispose()
35     {
36         _context.Dispose();
37     }
38 }
```

Creamos controladores CRUD para cada una de nuestras tablas



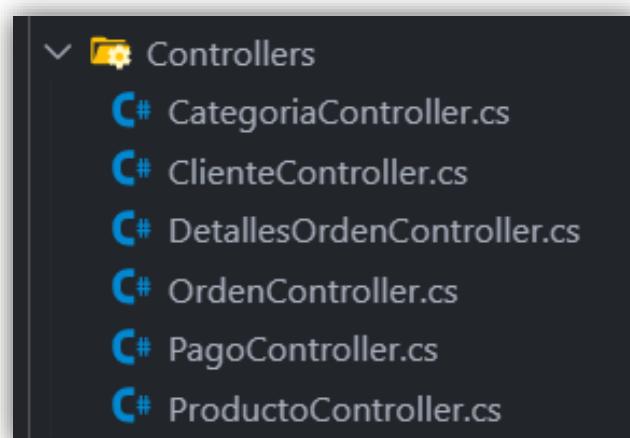
# Ejemplo CRUD para nuestra entidad Categoría



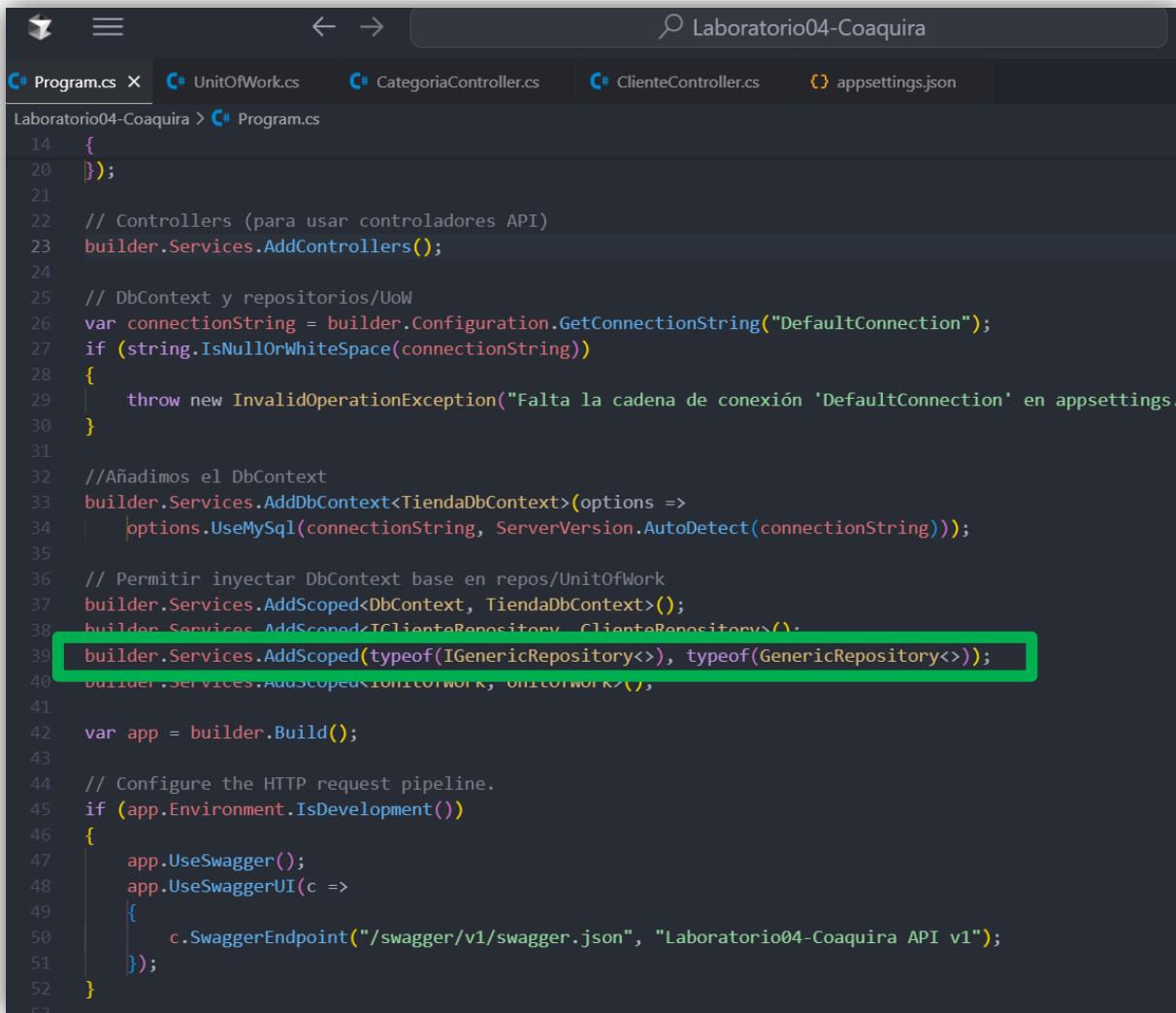
```

1  using Microsoft.AspNetCore.Mvc;
2
3  namespace Laboratorio04_Coaquia.Controllers;
4
5  [ApiController]
6  [Route("api/[controller]")]
7  public class CategoriaController : ControllerBase
8  {
9      private readonly IUnitOfWork _unitOfWork;
10
11     public CategoriaController(IUnitOfWork unitOfWork)
12     {
13         _unitOfWork = unitOfWork;
14     }
15
16     [HttpGet]
17     public IActionResult GetAll()
18     {
19         return Ok(_unitOfWork.Categorias.GetAll());
20     }
21
22     [HttpGet("{id}")]
23     public IActionResult GetById(int id)
24     {
25         var entity = _unitOfWork.Categorias.GetById(id);
26         if (entity == null) return NotFound();
27         return Ok(entity);
28     }
29
30     [HttpPost]
31     public IActionResult Create([FromBody] categoria entity)
32     {
33         _unitOfWork.Categorias.Add(entity);
34         _unitOfWork.SaveChanges();
35         return CreatedAtAction(nameof(GetById), new { id = entity.CategoriaID }, entity);
36     }
37
38 }
```

Y Así con las demás ...



# Registraremos DI de repositorio generico en Program.cs



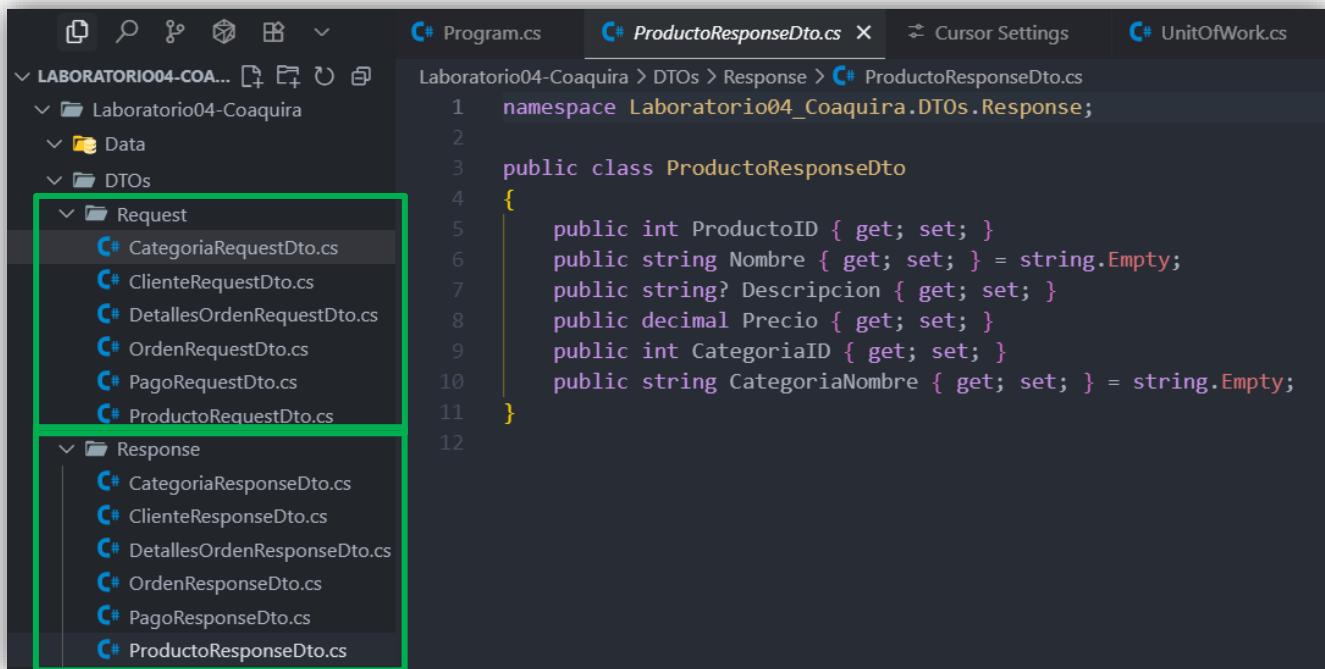
```
C# Program.cs X C# UnitOfWork.cs C# CategoriaController.cs C# ClienteController.cs C# appsettings.json
Laboratorio04-Coaquira > C# Program.cs
14  {
15  });
16
17 // Controllers (para usar controladores API)
18 builder.Services.AddControllers();
19
20 // DbContext y repositorios/UoW
21 var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
22 if (string.IsNullOrWhiteSpace(connectionString))
23 {
24     throw new InvalidOperationException("Falta la cadena de conexión 'DefaultConnection' en appsettings.");
25 }
26
27 //Añadimos el DbContext
28 builder.Services.AddDbContext<TiendaDbContext>(options =>
29     options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString)));
30
31
32 // Permitir injectar DbContext base en repos/UnitOfWork
33 builder.Services.AddScoped<DbContext, TiendaDbContext>();
34 builder.Services.AddScoped<IClienteRepository, ClienteRepository>();
35
36 builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));
37 builder.Services.AddScoped<UnitOfWork, UnitOfWork>();
38
39 var app = builder.Build();
40
41 // Configure the HTTP request pipeline.
42 if (app.Environment.IsDevelopment())
43 {
44     app.UseSwagger();
45     app.UseSwaggerUI(c =>
46     {
47         c.SwaggerEndpoint("/swagger/v1/swagger.json", "Laboratorio04-Coaquira API v1");
48     });
49 }
50
51 }
52 }
```

## Complementando con DTOs

Configuramos el proyecto en este punto para implementar las DTOs es decir el traslado de datos de una capa a otra en este caso modificaremos los controladores para que usen los DTOs en vez de usar directamente las entidades en las respuestas y solicitudes

## Crear Carpeta DTOs

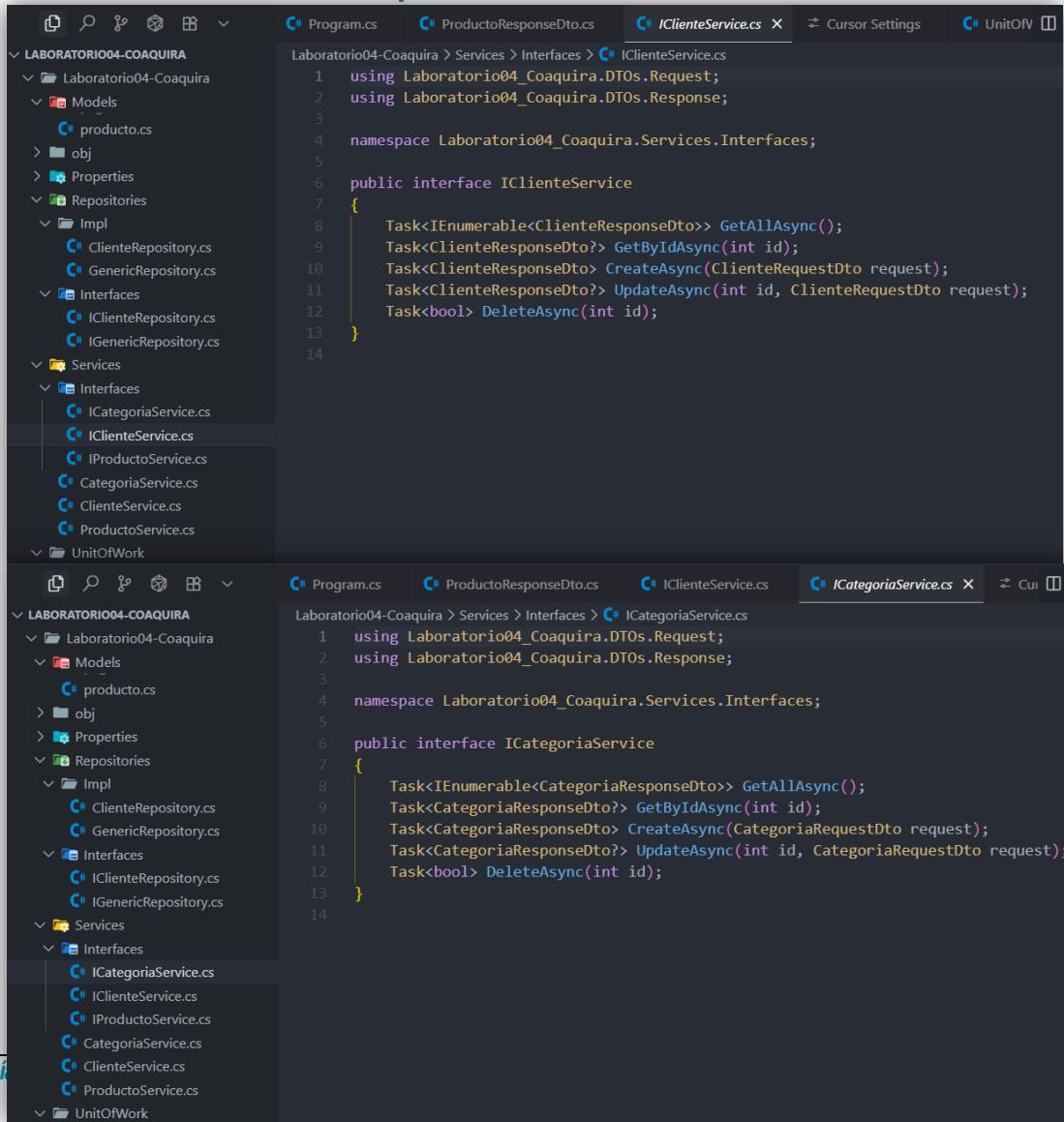
Creamos una carpeta exclusiva para ellas, separándolas por módulos Request y Response para mayor organización



```

    C# Program.cs          C# ProductoResponseDto.cs X      Cursor Settings      C# UnitOfWork.cs
    Laboratorio04-Coaquira > DTOs > Response > C# ProductoResponseDto.cs
    1  namespace Laboratorio04_Coaquia.DTOs.Response;
    2
    3  public class ProductoResponseDto
    4  {
    5      public int ProductoID { get; set; }
    6      public string Nombre { get; set; } = string.Empty;
    7      public string? Descripcion { get; set; }
    8      public decimal Precio { get; set; }
    9      public int CategoriaID { get; set; }
   10     public string CategoriaNombre { get; set; } = string.Empty;
   11 }
   12
  
```

## Crear Servicios de Aplicación:



```

    C# Program.cs          C# ProductoResponseDto.cs      Cursor Settings      C# UnitOfWork.cs
    Laboratorio04-Coaquira > Services > Interfaces > C# IClienteService.cs
    1  using Laboratorio04_Coaquia.DTOs.Request;
    2  using Laboratorio04_Coaquia.DTOs.Response;
    3
    4  namespace Laboratorio04_Coaquia.Services.Interfaces;
    5
    6  public interface IClienteService
    7  {
    8      Task<IEnumerable<ClienteResponseDto>> GetAllAsync();
    9      Task<ClienteResponseDto?> GetByIdAsync(int id);
   10     Task<ClienteResponseDto> CreateAsync(ClienteRequestDto request);
   11     Task<ClienteResponseDto?> UpdateAsync(int id, ClienteRequestDto request);
   12     Task<bool> DeleteAsync(int id);
   13 }
   14

    C# Program.cs          C# ProductoResponseDto.cs      Cursor Settings      C# ICategoríaService.cs X
    Laboratorio04-Coaquira > Services > Interfaces > C# ICategoríaService.cs
    1  using Laboratorio04_Coaquia.DTOs.Request;
    2  using Laboratorio04_Coaquia.DTOs.Response;
    3
    4  namespace Laboratorio04_Coaquia.Services.Interfaces;
    5
    6  public interface ICategoríaService
    7  {
    8      Task<IEnumerable<CategoriaResponseDto>> GetAllAsync();
    9      Task<CategoriaResponseDto?> GetByIdAsync(int id);
   10     Task<CategoriaResponseDto> CreateAsync(CategoriaRequestDto request);
   11     Task<CategoriaResponseDto?> UpdateAsync(int id, CategoriaRequestDto request);
   12     Task<bool> DeleteAsync(int id);
   13 }
   14
  
```

**Guía** **Ág. 18**

```

    1  using Laboratorio04_Coaquira.DTOs.Request;
    2  using Laboratorio04_Coaquira.DTOs.Response;
    3
    4  namespace Laboratorio04_Coaquira.Services.Interfaces;
    5
    6  public interface IProductoService
    7  {
    8      Task<IEnumerable<ProductoResponseDto>> GetAllAsync();
    9      Task<ProductoResponseDto?> GetByIdAsync(int id);
   10     Task<ProductoResponseDto> CreateAsync(ProductoRequestDto request);
   11     Task<ProductoResponseDto?> UpdateAsync(int id, ProductoRequestDto request);
   12     Task<bool> DeleteAsync(int id);
   13 }
   14
  
```

## Crear Mappers:

```

    1  using Laboratorio04_Coaquira.DTOs.Request;
    2  using Laboratorio04_Coaquira.DTOs.Response;
    3  using Laboratorio04_Coaquira.Models;
    4
    5  namespace Laboratorio04_Coaquira.Mappers;
    6
    7  public interface IClienteMapper
    8  {
    9      cliente ToEntity(ClienteRequestDto dto);
   10     ClienteResponseDto ToResponseDto(cliente entity);
   11     void UpdateEntity(cliente entity, ClienteRequestDto dto);
   12 }
   13
  
```

```

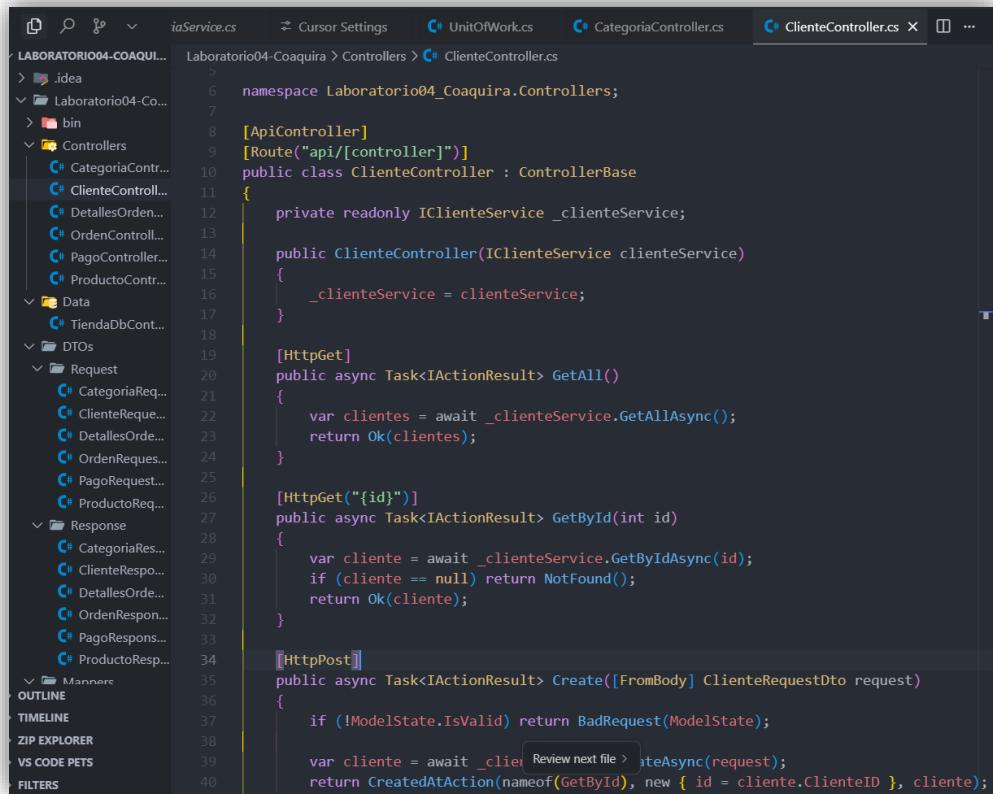
    1  using Laboratorio04_Coaquira.DTOs.Request;
    2  using Laboratorio04_Coaquira.DTOs.Response;
    3  using Laboratorio04_Coaquira.Models;
    4
    5  namespace Laboratorio04_Coaquira.Mappers;
    6
    7  public class ClienteMapper : IClienteMapper
    8  {
    9      public cliente ToEntity(ClienteRequestDto dto)
   10     {
   11         return new cliente
   12         {
   13             Nombre = dto.Nombre,
   14             Correo = dto.Correo,
   15             FechaCreacion = DateTime.Now
   16         };
   17     }
   18
   19     public ClienteResponseDto ToResponseDto(cliente entity)
   20     {
   21         return new ClienteResponseDto
   22         {
   23             ClienteID = entity.ClienteID,
   24             Nombre = entity.Nombre,
   25             Correo = entity.Correo,
   26             FechaCreacion = entity.FechaCreacion
   27         };
   28     }
   29
   30     public void UpdateEntity(cliente entity, ClienteRequestDto dto)
   31     {
   32         entity.Nombre = dto.Nombre;
   33         entity.Correo = dto.Correo;
   34     }
   35 }
   36
  
```

```
Laboratorio04-Coaquirá > Mappers > C# ICategoríaMapper.cs
1  using Laboratorio04_Coaquirá.DTOs.Request;
2  using Laboratorio04_Coaquirá.DTOs.Response;
3  using Laboratorio04_Coaquirá.Models;
4
5  namespace Laboratorio04_Coaquirá.Mappers;
6
7  public interface ICategoríaMapper
8  {
9      categoría ToEntity(CategoríaRequestDto dto);
10     CategoríaResponseDto ToResponseDto(categoría entity);
11     void UpdateEntity(categoría entity, CategoríaRequestDto dto);
12 }
13
```

## Implementar servicios

```
Laboratorio04-Coaquirá > Services > C# ClienteService.cs
8
9  public class ClienteService : IClienteService
10 {
11     private readonly IUnitOfWork _unitOfWork;
12     private readonly IClienteMapper _mapper;
13
14     public ClienteService(IUnitOfWork unitOfWork, IClienteMapper mapper)
15     {
16         _unitOfWork = unitOfWork;
17         _mapper = mapper;
18     }
19
20     public async Task<IEnumerable<ClienteResponseDto>> GetAllAsync()
21     {
22         var clientes = _unitOfWork.Clientes.GetAll();
23         return clientes.Select(_mapper.ToResponseDto);
24     }
25
26     public async Task<ClienteResponseDto?> GetByIdAsync(int id)
27     {
28         var cliente = _unitOfWork.Clientes.GetById(id);
29         return cliente == null ? null : _mapper.ToResponseDto(cliente);
30     }
31
32     public async Task<ClienteResponseDto> CreateAsync(ClienteRequestDto request)
33     {
34         var cliente = _mapper.ToEntity(request);
35         _unitOfWork.Clientes.Add(cliente);
36         _unitOfWork.SaveChanges();
37         return _mapper.ToResponseDto(cliente);
38     }
39
40     public async Task<ClienteResponseDto?> UpdateAsync(int id, ClienteRequestDto request)
41     {
42         var cliente = _unitOfWork[Review next file >] /Id(id);
43         if (cliente == null) return null;
```

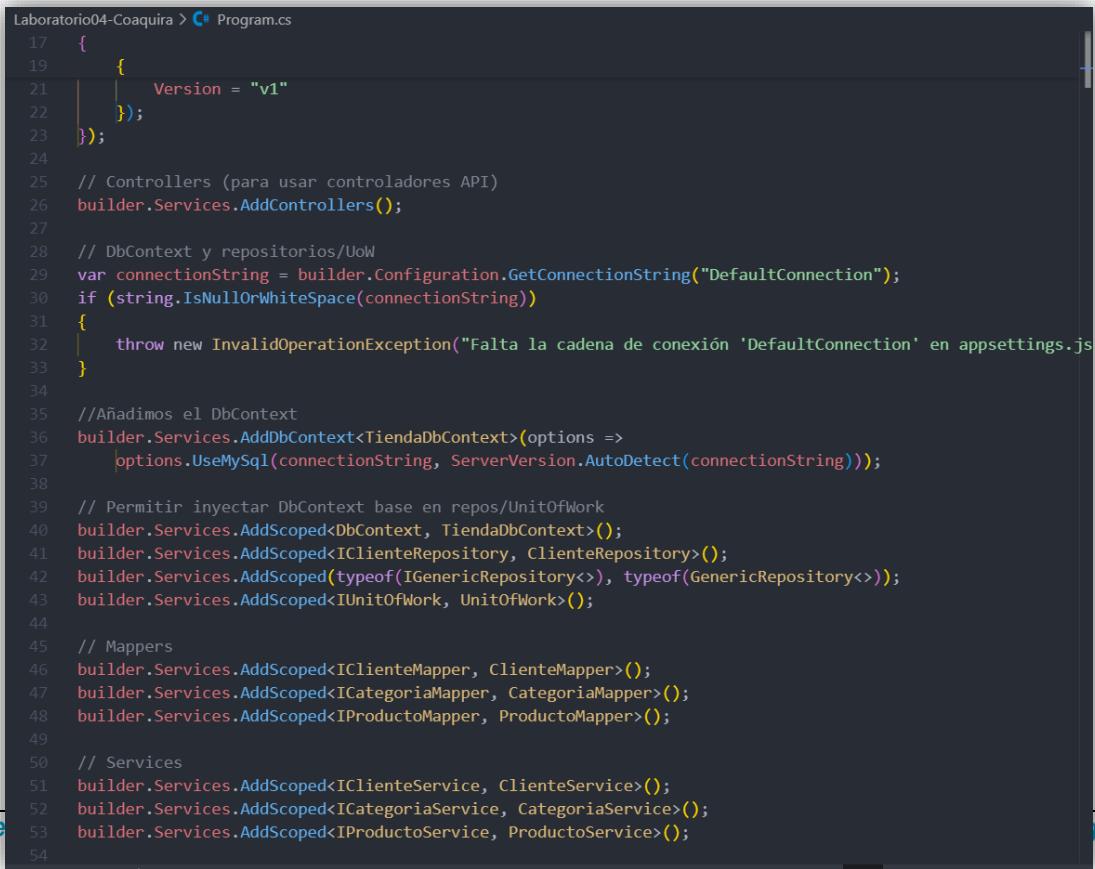
# Actualizar controladores para usar DTOs



```

1  // Clientes
2  // -----
3  // 
4  // 
5  // 
6  namespace Laboratorio04_Coaquia.Controllers;
7
8  [ApiController]
9  [Route("api/[controller]")]
10 public class ClienteController : ControllerBase
11 {
12     private readonly IClienteService _clienteService;
13
14     public ClienteController(IClienteService clienteService)
15     {
16         _clienteService = clienteService;
17     }
18
19     [HttpGet]
20     public async Task<IActionResult> GetAll()
21     {
22         var clientes = await _clienteService.GetAllAsync();
23         return Ok(clientes);
24     }
25
26     [HttpGet("{id}")]
27     public async Task<IActionResult> GetById(int id)
28     {
29         var cliente = await _clienteService.GetByIdAsync(id);
30         if (cliente == null) return NotFound();
31         return Ok(cliente);
32     }
33
34     [HttpPost]
35     public async Task<IActionResult> Create([FromBody] ClienteRequestDto request)
36     {
37         if (!ModelState.IsValid) return BadRequest(ModelState);
38
39         var cliente = await _clienteService.CreateAsync(request);
40         return CreatedAtAction(nameof(GetById), new { id = cliente.ClienteID }, cliente);
41     }
42
43 }
```

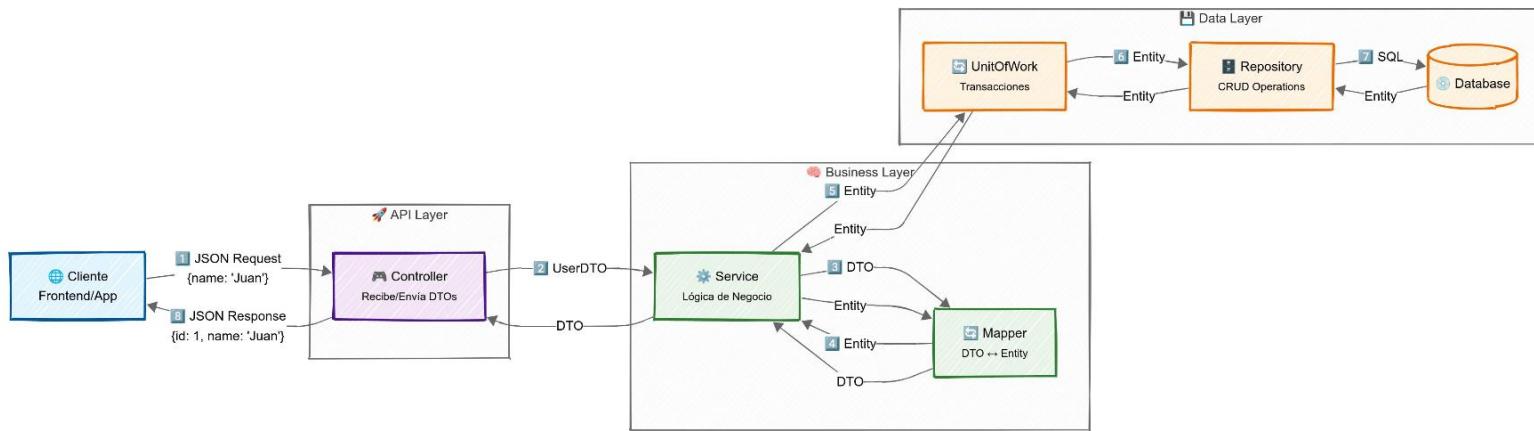
# Registrar servicios en Program.cs



```

1  // Program.cs
2
3  builder.Services.AddControllers();
4
5  // DbContext y repositorios/UoW
6  var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
7  if (string.IsNullOrWhiteSpace(connectionString))
8  {
9      throw new InvalidOperationException("Falta la cadena de conexión 'DefaultConnection' en appsettings.json");
10 }
11
12 // Añadimos el DbContext
13 builder.Services.AddDbContext<TiendaDbContext>(options =>
14     options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString)));
15
16 // Permitir inyectar DbContext base en repos/UnitOfWork
17 builder.Services.AddScoped<DbContext, TiendaDbContext>();
18 builder.Services.AddScoped<IClienteRepository, ClienteRepository>();
19 builder.Services.AddScoped<IGenericRepository<>, GenericRepository<>>();
20 builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
21
22 // Mappers
23 builder.Services.AddScoped<IClienteMapper, ClienteMapper>();
24 builder.Services.AddScoped<ICategoríaMapper, CategoríaMapper>();
25 builder.Services.AddScoped<IProductoMapper, ProductoMapper>();
26
27 // Services
28 builder.Services.AddScoped<IClienteService, ClienteService>();
29 builder.Services.AddScoped<ICategoríaService, CategoríaService>();
30 builder.Services.AddScoped<IProductoService, ProductoService>();
31
32 }
```

## Resultado de Flujo:



## Nota importante:

**Controllers:** Solo manejan HTTP y validación de entrada

**Services:** Contienen la lógica de negocio

**Mappers:** Convierten entre DTOs y entidades

**Repositories:** Acceso a datos puro

## Probandolo en Swagger:

Link: <https://youtu.be/mETTRNdL8No>

(Ver el video es mucho mejor que las imágenes)

## Listando

**Cliente**

GET /api/Cliente

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5188/api/Cliente' \
  -H 'accept: */*'
```

Request URL

<http://localhost:5188/api/Cliente>

Server response

Code	Details
200	<p>Response body</p> <pre>[   {     "clienteID": 1,     "nombre": "Juan Perez",     "correo": "juan.perez@email.com",     "fechaCreacion": "2025-09-11T12:32:00"   },   {     "clienteID": 2,     "nombre": "Maria Lopez",     "correo": "maria.lopez@email.com",     "fechaCreacion": "2025-09-11T12:32:00"   },   {     "clienteID": 3,     "nombre": "Carlos Garcia",     "correo": "carlos.garcia@email.com",     "fechaCreacion": "2025-09-11T12:32:00"   },   {     "clienteID": 4,     "nombre": "betito",     "correo": "Beto@gmail.com",     "fechaCreacion": "2025-09-15T23:31:45"   } ]</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Tue, 16 Sep 2025 04:34:23 GMT server: Kestrel transfer-encoding: chunked</pre>

Responses

Code	Description	Links
------	-------------	-------

**MySQL Workbench**

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- fiberotel\_rest\_db
- fiberotel\_rest\_db\_local
- ingif\_2023
- ingif\_lab08
- javaweb
- jpaprueba
- lab13
- login
- matricula
- movies\_db
- pruebas
- sys
- tecsup
- tienda\_postres
- tiendadb

Tables

- categorias
- clientes
- detalle\_pedidos
- pedidos
- productos
- categorias
- clientes
- tiendadb

queries

clientes

detalle\_pedidos

pedidos

productos

categorias

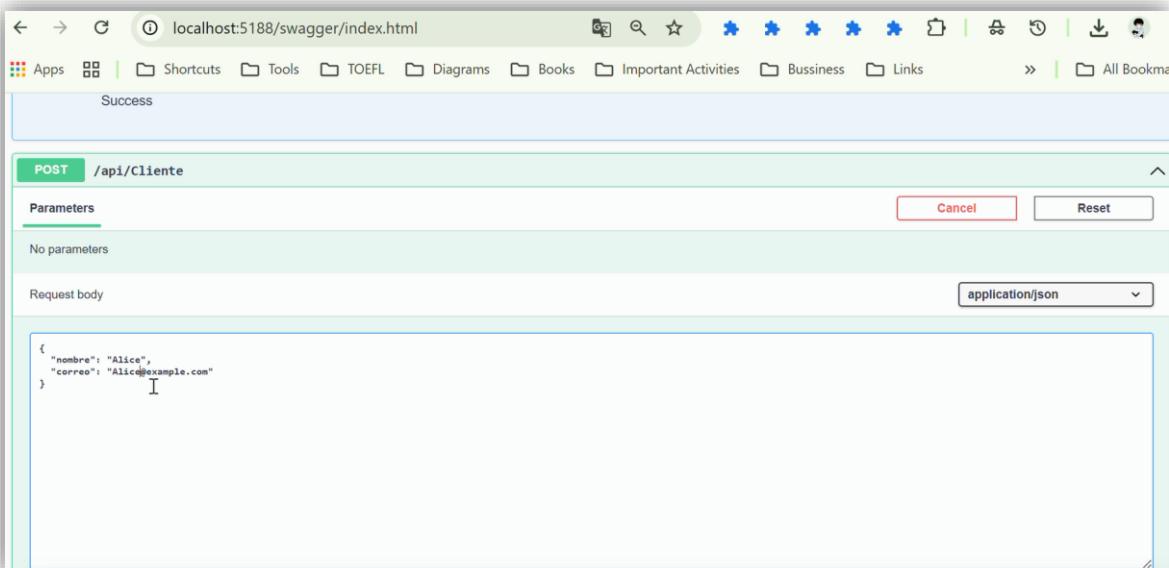
clientes

tiendadb

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

ClienteID	Nombre	Correo	FechaCreacion
1	Juan Perez	juan.perez@email.com	2025-09-11 12:32:00
2	Maria Lopez	maria.lopez@email.com	2025-09-11 12:32:00
3	Carlos Garcia	carlos.garcia@email.com	2025-09-11 12:32:00
4	betito	Beto@gmail.com	2025-09-15 23:31:45
*	HULL	HULL	HULL

# Agregando



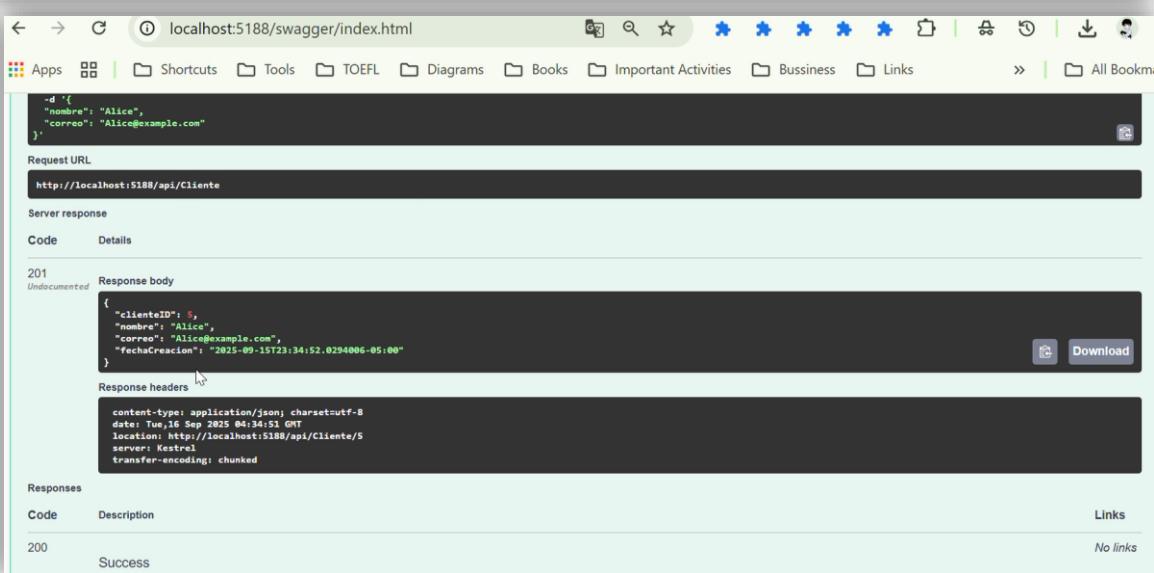
**POST /api/Cliente**

**Parameters**

No parameters

**Request body** (application/json)

```
{
  "nombre": "Alice",
  "correo": "Alice@example.com"
}
```



**Request URL**: <http://localhost:5188/api/Cliente>

**Server response**

Code	Details
201	Response body

```
{
  "clienteID": 5,
  "nombre": "Alice",
  "correo": "Alice@example.com",
  "FechaCreacion": "2025-09-15T23:34:52.0294006-05:00"
}
```

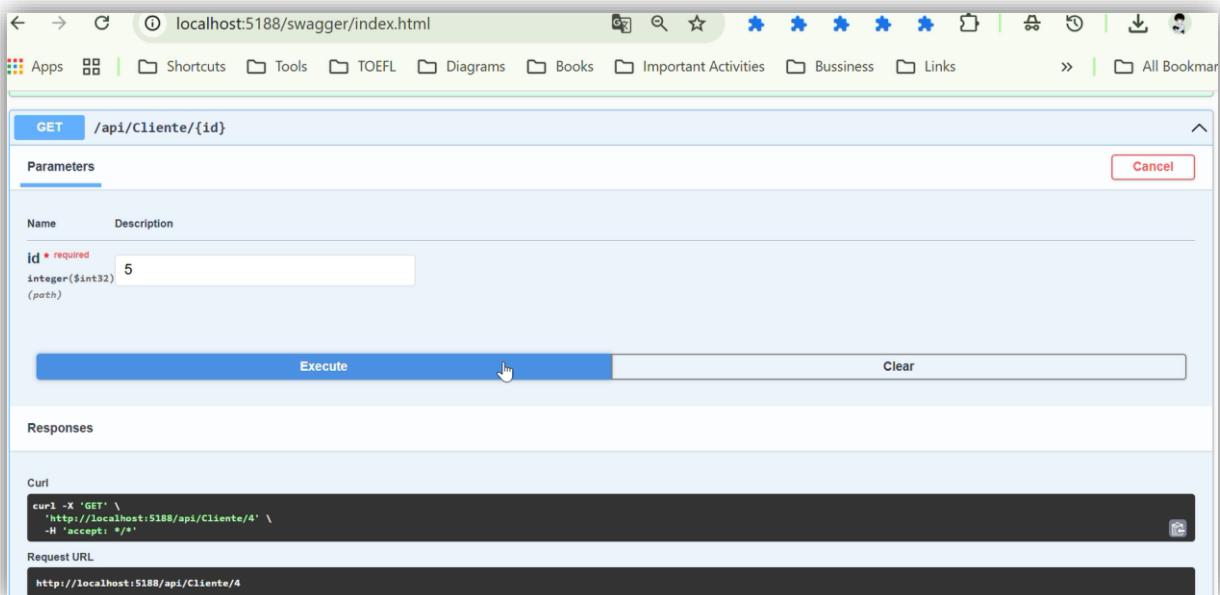
**Response headers**

```
content-type: application/json; charset=utf-8
date: Tue, 16 Sep 2025 04:34:51 GMT
location: http://localhost:5188/api/Cliente/5
server: Kestrel
transfer-encoding: chunked
```

**Responses**

Code	Description	Links
200	Success	No links

# Llamando por ID



**GET /api/Cliente/{id}**

**Parameters**

Name	Description
<b>id</b> * required	integer(\$int32) (path)

**Execute**  **Clear**

**Responses**

**Curl**

```
curl -X 'GET' \
'http://localhost:5188/api/Cliente/4' \
-H 'accept: */*
```

**Request URL**: <http://localhost:5188/api/Cliente/4>

localhost:5188/swagger/index.html

Execute Clear

### Responses

Curl

```
curl -X 'GET' \
'http://localhost:5188/api/Cliente/5' \
-H 'accept: */*'
```

Request URL

<http://localhost:5188/api/Cliente/5>

### Server response

Code	Details
200	<p>Response body</p> <pre>{   "clienteID": 5,   "nombre": "Alice",   "correo": "Alice@example.com",   "fechaCreacion": "2025-09-15T23:34:52" }</pre> <p>Download</p> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Tue, 16 Sep 2025 04:35:14 GMT server: Kestrel transfer-encoding: chunked</pre>

Responses

## Editando

localhost:5188/swagger/index.html

PUT /api/Cliente/{id}

Cancel Reset

### Parameters

Name	Description
<b>id</b> * required	integer(\$int32) (path)

Request body

application/json

```
{
  "nombre": "Alice",
  "correo": "Alice123@gmail.com"
}
```

localhost:5188/swagger/index.html

curl

```
curl -X 'PUT' \
'http://localhost:5188/api/Cliente/5' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
  "nombre": "Alice",
  "correo": "Alice123@gmail.com"
}'
```

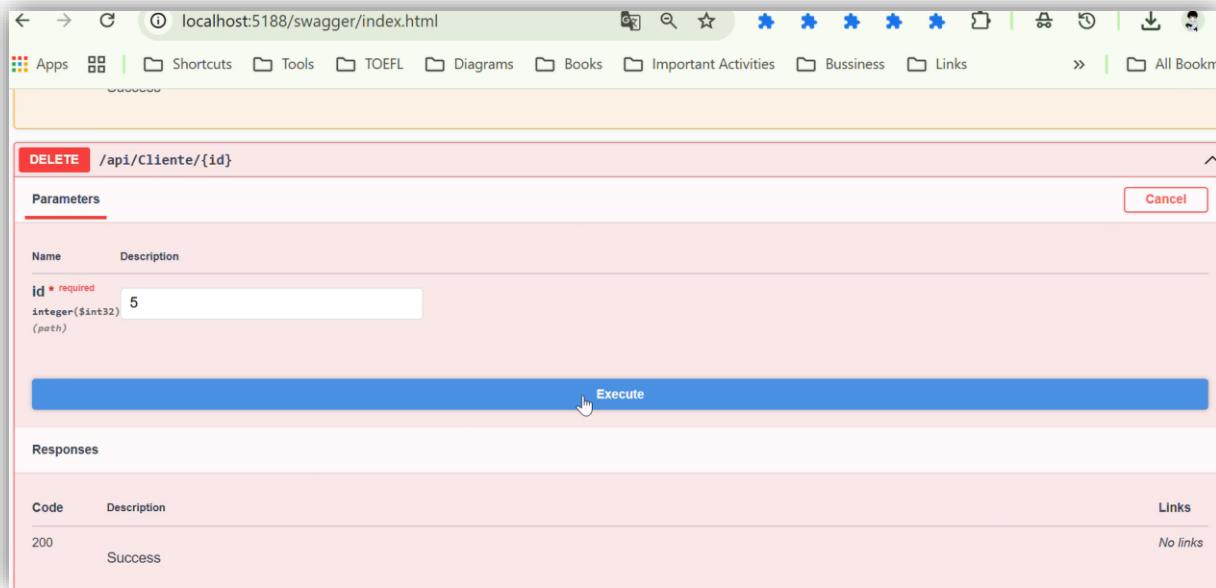
Request URL

<http://localhost:5188/api/Cliente/5>

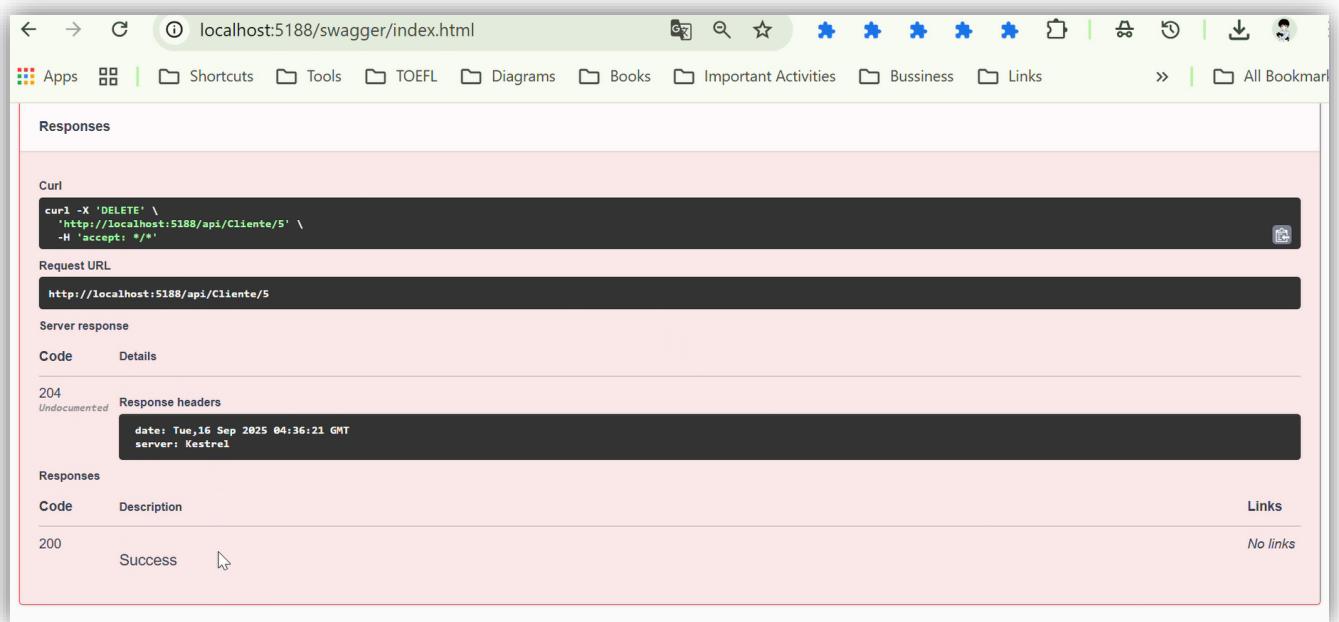
### Server response

Code	Details
200	<p>Response body</p> <pre>{   "clienteID": 5,   "nombre": "Alice",   "correo": "Alice123@gmail.com",   "fechaCreacion": "2025-09-15T23:34:52" }</pre> <p>Download</p> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Tue, 16 Sep 2025 04:35:14 GMT server: Kestrel transfer-encoding: chunked</pre>

# Eliminando



The screenshot shows a browser window with the URL `localhost:5188/swagger/index.html`. The main content is a Swagger UI interface for a DELETE endpoint at `/api/cliente/{id}`. The 'Parameters' section shows a required parameter `id` with the value `5`. Below it is a large blue 'Execute' button. The 'Responses' section shows a single response entry for code `200` with the description `Success` and a note 'No links'.



The screenshot shows a browser window with the URL `localhost:5188/swagger/index.html`. The main content is a Swagger UI interface for a DELETE endpoint. It shows the curl command, request URL (`http://localhost:5188/api/Cliente/5`), and the server response headers (`date: Tue, 16 Sep 2025 04:36:21 GMT`, `server: Kestrel`). The 'Responses' section shows a single response entry for code `200` with the description `Success` and a note 'No links'.

## Observaciones

- **Unit of Work** se posiciona como el coordinador central de transacciones, permitiendo que múltiples operaciones de repositorio trabajen de forma atómica y consistente.
- Los **Repositorios** actúan como una capa de abstracción efectiva que oculta completamente los detalles de Entity Framework Core a la lógica de negocio.
- Los **DTOs** funcionan como un filtro bidireccional que transforma datos entre el formato público (API) y el formato interno (entidades), manteniendo la seguridad y flexibilidad.

- El **DbContext** centraliza la configuración de la base de datos y facilita el mapeo objeto-relacional sin acoplar el código a tecnologías específicas.
- La separación en capas (**Controller → Service → Repository**) elimina dependencias directas y mejora la testabilidad del código.

## Conclusiones

- **La arquitectura implementada cumple su objetivo principal** al desacoplar los controladores de las entidades de base de datos, creando un sistema más mantible y seguro.
- **Los patrones aplicados demuestran su valor práctico** puesto que facilitan las validaciones automáticas, operaciones transaccionales consistentes y protección de datos sensibles.
- **La estructura modular resultante permite escalabilidad** sin comprometer la integridad del sistema, facilitando la adición de nuevas funcionalidades.
- **En conjunto, estos patrones establecen una base sólida** para el desarrollo de aplicaciones empresariales que requieren robustez, seguridad y mantenibilidad a largo plazo.