.Net Clean Architecture

Contenido

Creando Proyecto

```
dotnet new sln -o Proyecto      // crea el proyecto solución
```

Creación de las tres capas básicas para una arquitectura limpia

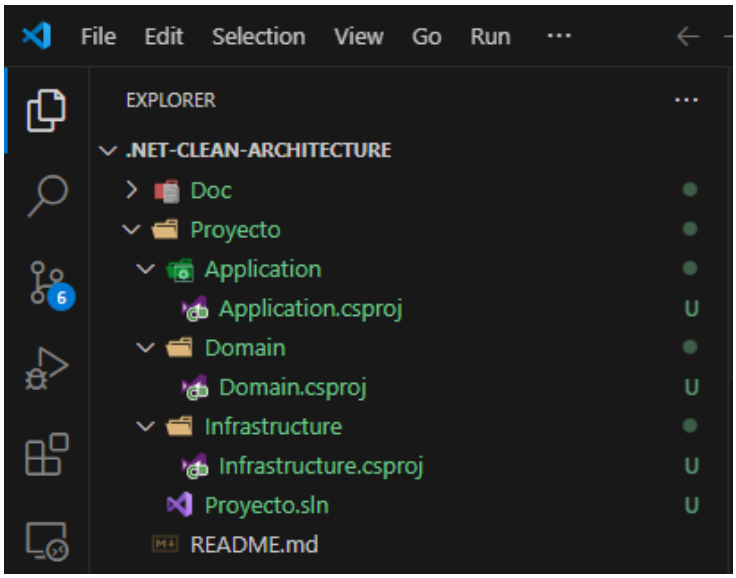
NOTA: elimina en un principio la clase que aparece por defecto y la carpeta obj

```
dotnet new classlib -o Domain -f net7.0
```

```
dotnet new classlib -o Application -f net7.0
```

```
dotnet new classlib -o Infrastructure-f net7.0
```

Vista:



Crea la API (no borres nada de ella)

`dotnet new webapi -o Web.API -f net7.0`

NOTA: comprobar que todo está bien hasta el momento → `dotnet build`

```
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet build
MSBuild version 17.5.1+f6fdcf537 for .NET
  Determinando los proyectos que se van a restaurar...
C:\Program Files\dotnet\sdk\7.0.203\NuGet.targets(132,5): warning : No se puede encontrar un proyecto para restaurar. [C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto\Proyecto.sln]

Compilación correcta.

C:\Program Files\dotnet\sdk\7.0.203\NuGet.targets(132,5): warning : No se puede encontrar un proyecto para restaurar. [C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto\Proyecto.sln]
    1 Advertencia(s)
    0 Errores

Tiempo transcurrido 00:00:00.45
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto>
```

Establecer relaciones (como, por ejemplo):

`dotnet add Application/Application.csproj reference .\Domain\Domain.csproj`

```
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet add Application/Application.csproj reference .\Domain\Domain.csproj
Se ha agregado la referencia "..\Domain\Domain.csproj" al proyecto.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet add .\Infrastructure\Infrastructure.csproj reference .\Domain\Domain.csproj
Se ha agregado la referencia "..\Domain\Domain.csproj" al proyecto.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet add .\Infrastructure\Infrastructure.csproj reference .\Application\Application.csproj
Se ha agregado la referencia "..\Application\Application.csproj" al proyecto.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto>
```

```
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet add .\Web.API\Web.API.csproj reference .\Application\Application.csproj .\Infrastructure\Infrastructure.csproj
Se ha agregado la referencia "..\Application\Application.csproj" al proyecto.
Se ha agregado la referencia "..\Infrastructure\Infrastructure.csproj" al proyecto.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet sln add .\Web.API\Web.API.csproj
Se ha agregado el proyecto "Web.API\Web.API.csproj" a la solución.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet sln add .\Application\Application.csproj
Se ha agregado el proyecto "Application\Application.csproj" a la solución.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet sln add .\Infrastructure\Infrastructure.csproj
Se ha agregado el proyecto "Infrastructure\Infrastructure.csproj" a la solución.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet sln add .\Domain\Domain.csproj
Se ha agregado el proyecto "Domain\Domain.csproj" a la solución.
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto>
```

Comprobamos que todo está bien

`dotnet build`

Arrancar la API

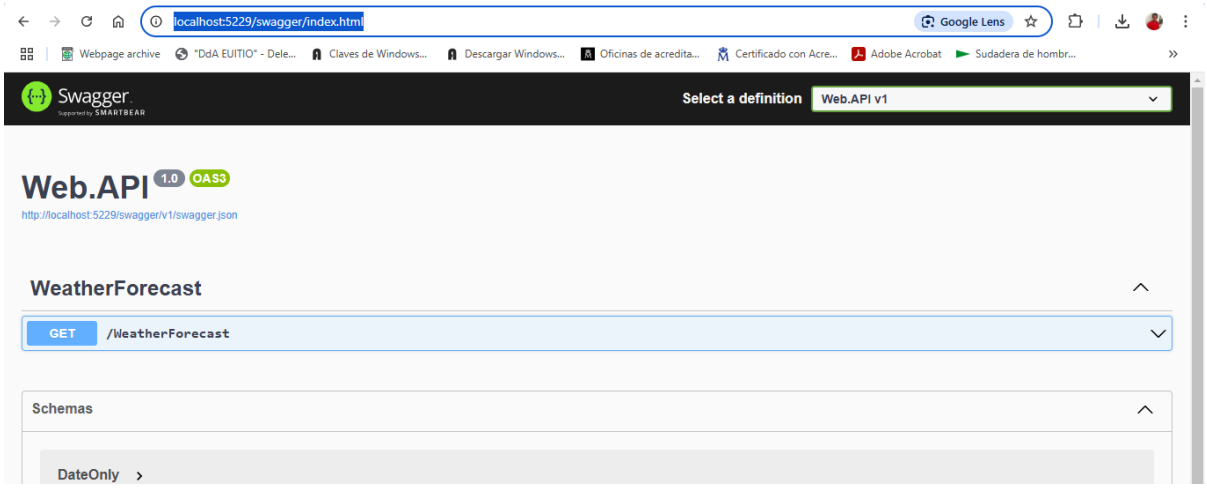
`dotnet run -p .\Web.API\`

Vista:

```
PS C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto> dotnet run -p .\Web.API\
Advertencia NETSDK1174: La abreviatura de -p para --project está en desuso. Use --project.
Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5229
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\germa\Desktop\DDD\.Net-Clean-Architecture\Proyecto\Web.API
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
      Failed to determine the https port for redirect.
```

NOTA: abre el navegador en el localhost, es verdad que no aparecerá nada, debes de añadir a la url el swagger:

http://localhost:5229/swagger/index.html

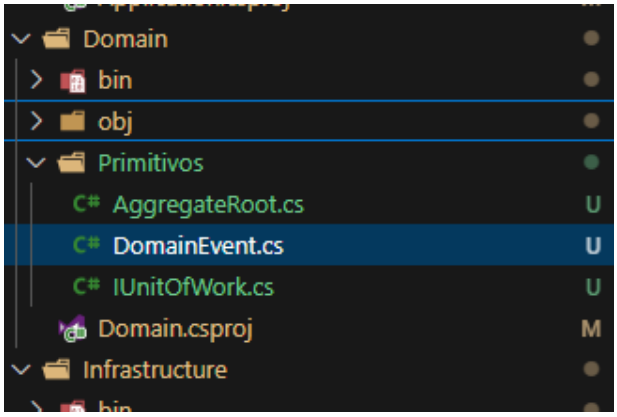


Definir el dominio

Añadir primitivos

- Creas una carpeta primitivos en dominio
- Creas una clase AggregateRoot.cs
- Creas una clase DomainEvents.cs
    - Instala Nuget Gallery – MediatR- instalar en Domain y en Application

- Creas una Interfaz UnitOfWork.cs



AggregateRoot.cs

```
namespace Domain.Primitivos;

public abstract class AggregateRoot
{
    // lista de eventos de dominio
    private readonly List<DomainEvent> _domainEvents = new();

    // propiedad de solo lectura para acceder a la lista de eventos
    public ICollection<DomainEvent> DomainEvents => _domainEvents;

    // método para levantar eventos de dominio
    protected void Raise(DomainEvent domainEvent) =>
_domainEvents.Add(domainEvent);
}
```

DomainEvent.cs

```
using MediatR;

namespace Domain.Primitivos;

public record DomainEvent(Guid Id) : INotification;
```

IUnitOfWork.cs

```
namespace Domain.Primitivos;

public interface IUnitOfWork
{
    // método para guardar los cambios en la base de datos
    Task<bool> SaveChangesAsync(CancellationToken cancellationToken =
default);
}
```

Objetos de Valor (Object Value)

Valores sin identidad, pero hay que implementarlos iguales -> deben ser inmutables

Enlace: https://learn.microsoft.com/es-es/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/implement-value-objects

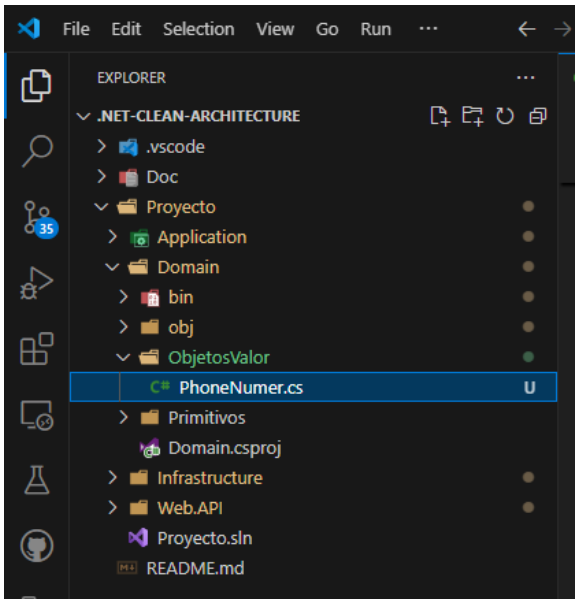
Creamos una carpeta para los objetos valor en Domain

Creamos una clase Objeto Valor llamada PhoneNumber.cs

PhoneNumber.cs

```
using System.Reflection.Metadata;
using System.Text.RegularExpressions;

namespace Domain.ObjetosValor;

public partial record PhoneNumber
{
    private const int DefaultLength = 9; // 9 digitos
    private const string Pattern = @"^\d{9}$"; // 9 digitos
    private PhoneNumber(string value) => Value = value;

    public string Value { get; init; }

    /// <summary>
    /// Crea un objeto PhoneNumber si el valor es valido
    /// </summary>
    /// <param name="value"></param>
    /// <returns></returns>
    public static PhoneNumber? Create(string value)
    {
        if (string.IsNullOrEmpty(value) ||
!PhoneNumberRegex().IsMatch(value) || value.Length != DefaultLength)
        {
            return null;
        }
        return new PhoneNumber(value);
    }
    /// <summary>
    /// Expresion regular para validar el valor
    /// </summary>
    /// <returns></returns>
    [GeneratedRegex(Pattern)]
    private static partial Regex PhoneNumberRegex();
}
```

Vista:



Aggreate Root

Creamos una nueva carpeta

Creamos una Clase Customer también

Creamos una identidad Customer

Creamos una interfaz CustomerRepository

Customer.cs

```
using Domain.ObjetosValor;
using Domain.Primitivos;

namespace Domain.Customer;

// Customer es una entidad, por lo que hereda de AggregateRoot
public sealed class Customer : AggregateRoot
{
    public Customer(CustomerId id, string name, string lastName, string
email, PhoneNumber phoneNumber, Address address)
    {
        Id = id;
        Name = name;
        LastName = lastName;
        Email = email;
        PhoneNumber = phoneNumber;
        Address = address;
    }
    public Customer()
    {
    }
```

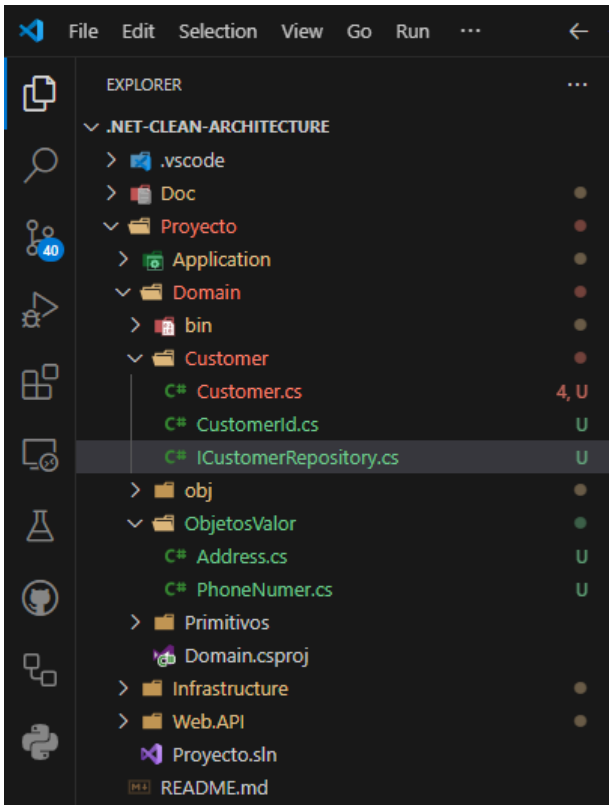
```
    public CustomerId Id { get; private set; } // Value Object
    public string Name { get; private set; } = string.Empty; // Propiedad
    public string LastName { get; set; } = string.Empty; // Propiedad

    public string FullName => $"{Name} {LastName}"; // Propiedad de solo
lectura
    public string Email { get; private set; } = string.Empty; //
Propiedad

    public PhoneNumber PhoneNumber { get; private set; }; // Value Object

    public Address Address { get; private set; } // Value Object

    public bool IsActive { get;  set; } // Propiedad
}
```

CustomerId.cs

```
namespace Domain.Customer;

/// <summary>
/// Identificador de cliente
/// </summary>
/// <param name="value"></param>
public record class CustomerId(Guid value);// Value Object
```

ICustomerRepository.cs

```
namespace Domain.Customer;

public interface ICustomerRepository
{
    Task<Customer?> GetByIdAsync(CustomerId id); // Metodo

    Task Add(Customer customer); // Metodo
}
```



Configuración Aplicación

Creamos:

```
using Microsoft.Extensions.DependencyInjection;
using FluentValidation.AspNetCore;
using FluentValidation;

namespace Application;

// Clase que contiene los métodos de extensión para la inyección de
dependencias.
public static class DependencyInjection
{
    // Método de extensión que añade los servicios de la aplicación.
    public static IServiceCollection AddApplication(this
IServiceCollection services)
    {

        // añadiendo los servicios de MediatR
        services.AddMediatR(config =>
        {
            config.RegisterServicesFromAssemblyContaining<ApplicationAsse
mblyReference>();
        });

        // añadiendo los servicios de FluentValidation
```

```
        services.AddValidatorsFromAssemblyContaining<ApplicationAssemblyR
eference>();

        return services;
    }

}
```

ApplicationAssemblyReference.cs

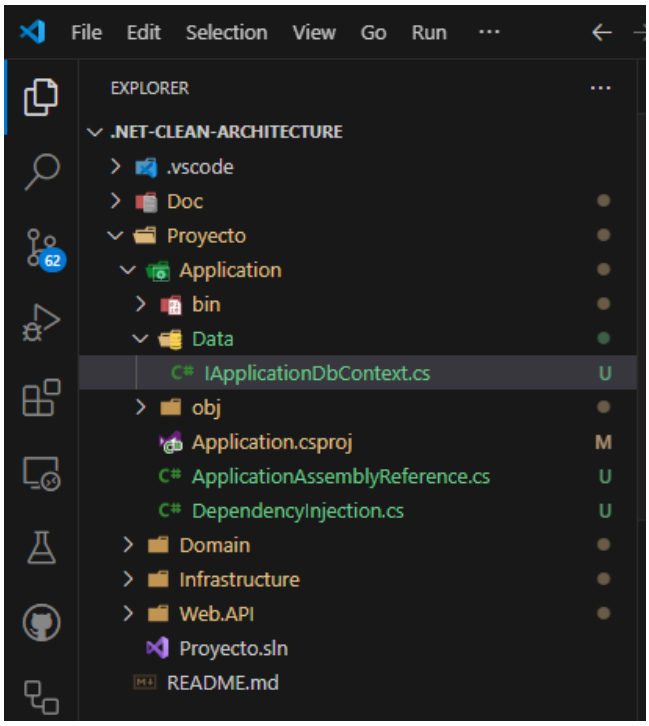
```
using System.Reflection;

namespace Application;
/// <summary>
/// Esta clase se utiliza para obtener la referencia a la asamblea de la
aplicación.
/// </summary>
public class ApplicationAssemblyReference
{
    // Esta propiedad estática se utiliza para obtener la referencia a la
asamblea de la aplicación.
    internal static readonly Assembly Assembly =
typeof(ApplicationAssemblyReference).Assembly;
}
```

Creamos una carpeta Data y dentro de ella una interfaz

IApplicationDbContext.cs

```
namespace Application;

using Domain.Customer;
using Microsoft.EntityFrameworkCore;

/// <summary>
/// Interfaz que define el contexto de la aplicación.
/// </summary>
public interface IApplicationDbContext
{
    public DbSet<Customer> Customers { get; set; } // Propiedad que
representa la tabla de clientes en la base de datos.
    // Método que guarda los cambios en la base de datos.
    public Task<int> SaveChangesAsync(CancellationToken cancellationToken
= default);
}
```

Vista final:



NOTA: Atiende a las versiones de los paquetes instalados porque pueden darte problemas según la versión de .net con la que trabajes;

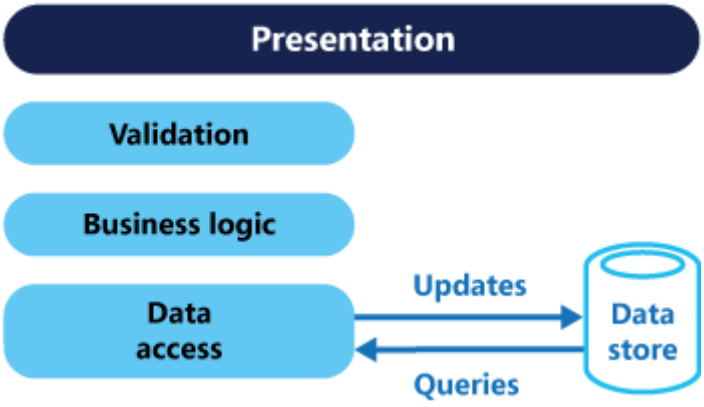
Patrón CQRS

Arquitectura limpia

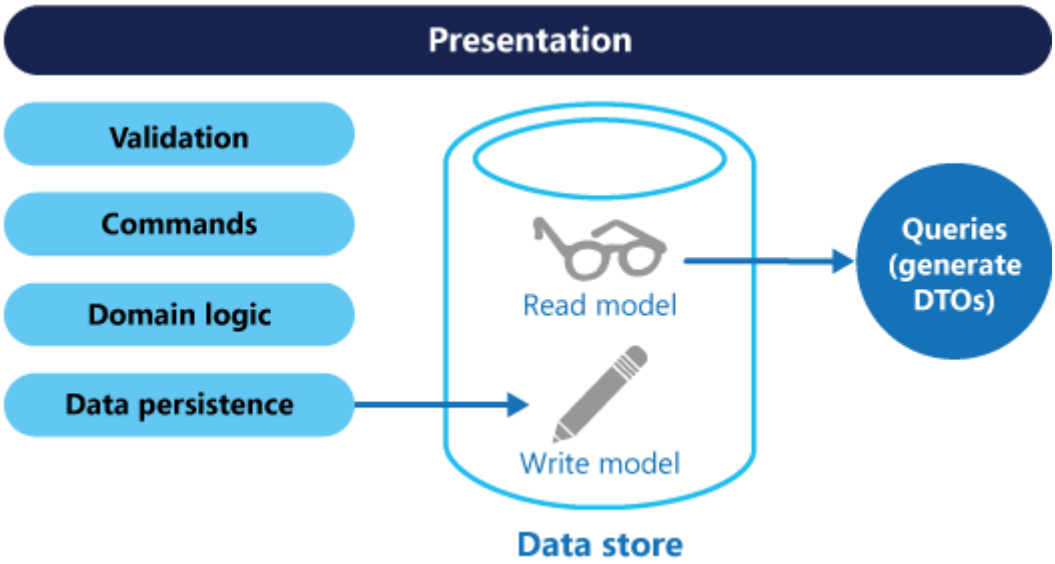
SRP, escalabilidad, extensión, etc.

Enlace: https://learn.microsoft.com/es-es/azure/architecture/patterns/cqrs

Inicio (CRUD):

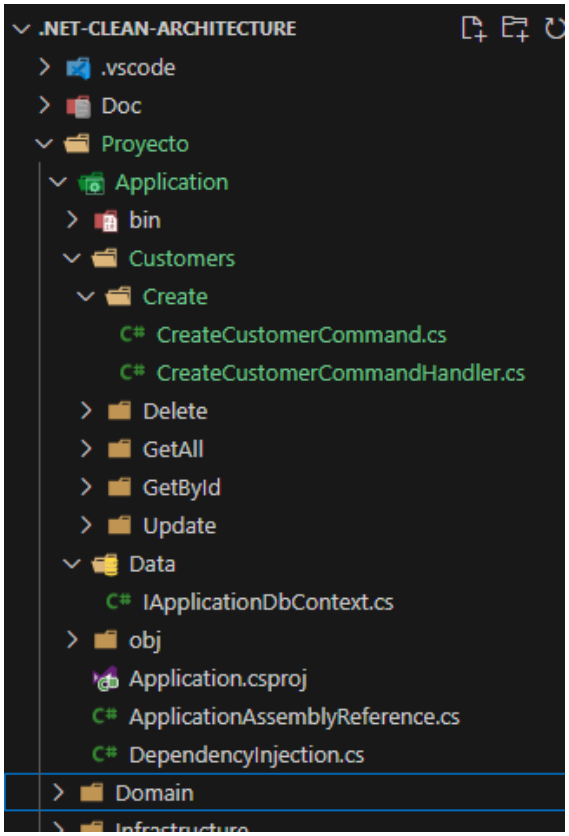


Solución (CQRS):



Creación CRUD carpetas. Primero creamos una carpeta Customer en Application y según las responsabilidades (créate, update, etc.) creamos sus correspondientes carpetas.

Vista:



Creamos las clases:

CreateCustomerCommand.cs

```
using MediatR;

namespace Application.Customers.Create
```

```
{
    // Es una clase sellada, es decir, no puede ser heredada
    public record CreateCustomerCommand(
        string Name,
        string LastName,
        string Email,
        string PhoneNumber,
        string Country,
        string State,
        string City,
        string Street,
        string ZipCode
    ) : IRequest<Unit>;

}
```

CreateCustomerCommandHandler.cs

```
using Domain.Customer;
using Domain.ObjetosValor;
using Domain.Primitivos;
using MediatR;

namespace Application.Customers.Create;

// Clase sellada que implementa la interfaz IRequestHandler
internal sealed class CreateCustomerCommandHAndler :
IRequestHandler<CreateCustomerCommand, Unit>
{
    private readonly ICustomerRepository _customerRepository;
    private readonly IUnitOfWork _unitOfWork;

    public CreateCustomerCommandHAndler(ICustomerRepository
customerRepository, IUnitOfWork unitOfWork)
    {
        _customerRepository = customerRepository ?? throw new
ArgumentNullException(nameof(customerRepository));
        _unitOfWork = unitOfWork ?? throw new
ArgumentNullException(nameof(unitOfWork));
    }


    // Método que se encarga de manejar la solicitud
    public async Task<Unit> Handle(CreateCustomerCommand request,
CancellationToken cancellationToken)
    {
        // Se valida que el nombre no sea nulo o vacío
        if (PhoneNumber.Create(request.PhoneNumber) is not PhoneNumber
phoneNumber)
```

```
        {
            throw new Exception("Phone number is required. " +
nameof(PhoneNumber));
        }

        var address = Address.Create(request.Street, request.City,
request.State, request.Country, request.ZipCode);
        // Se valida que la dirección no sea nula
        if (address is null)
        {
            throw new Exception("Address is required. " +
nameof(Address));
        }

        var customer = new Customer(new CustomerId(Guid.NewGuid()),
request.Name, request.LastName, request.Email, phoneNumber, address);
        if (customer is null)
        {
            throw new Exception("Customer is required. " +
nameof(Customer));
        }

        await _customerRepository.Add(customer); // Se agrega el cliente
        await _unitOfWork.SaveChangesAsync(cancellationToken); // Se
guardan los cambios en la base de datos

        return Unit.Value;
    }
}
```

Capa Infractructura

Crear carpeta en infraestructura llamada Persistencia

ApplicationDbContext.cs

```
using Application;
using Domain.Customer;
using Domain.Primitivos;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Persistence
{
    // clase que implementa la interfaz IApplicationDbContext y la
interfaz IUnitOfWork
    public class ApplicationDbContext : DbContext, IApplicationDbContext,
IUnitOfWork
```

```
    {
        private readonly IPublisher _publisher; // propiedad de solo
lectura para acceder al publicador
        public ApplicationDbContext(DbContextOptions options, IPublisher
publisher) : base(options)
        {
            // asignar el publicador a la propiedad
            _publisher = publisher ?? throw new
ArgumentNullException(nameof(publisher));
        }

        public DbSet<Customer> Customers { get; set; } // propiedad para
acceder a la tabla de clientes

        // método para guardar los cambios en la base de datos
        public override async Task<int>
SaveChangesAsync(CancellationToken cancellationToken = new
CancellationToken())
        {
            // obtener los eventos de dominio de las entidades que
implementan AggregateRoot
            var domainEvents = ChangeTracker.Entries<AggregateRoot>()
                .Select(e => e.Entity)
                .Where(e => e.GetDomainEvents().Any())
                .SelectMany(e => e.GetDomainEvents());

            // guardar los cambios en la base de datos
            var result = await base.SaveChangesAsync(cancellationToken);

            foreach (var domainEvent in domainEvents)
            {
                await _publisher.Publish(domainEvent, cancellationToken);
// publicar los eventos de dominio
            }

            return result;
        }
    }
}
```

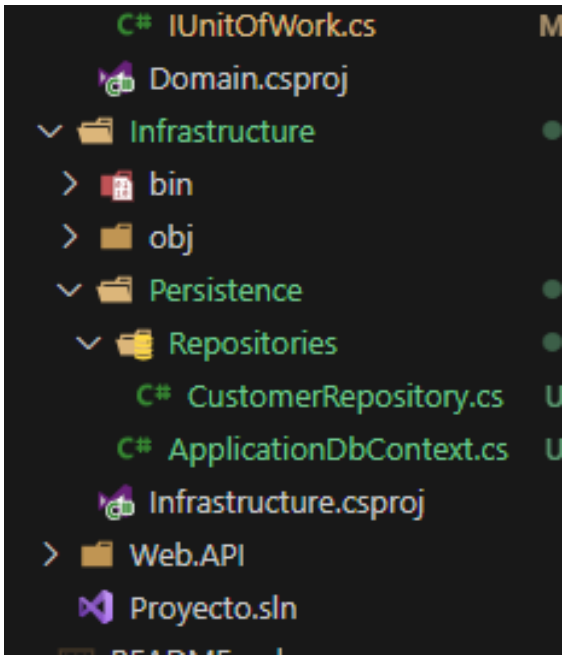
Crea otra carpeta dentro de Persistencia llamada Repositories.

Crea una clase **CustomerRepository.cs**

```
using Domain.Customer;
using Microsoft.EntityFrameworkCore;

namespace Infrastructure.Persistence.Repositories
{
```

```
    // interfaz para el repositorio de clientes
    public class CustomerRepository : ICustomerRepository
    {
        private readonly ApplicationDbContext _context; // propiedad de
solo lectura para acceder al contexto de la aplicación

        public CustomerRepository(ApplicationDbContext context)
        {
            _context = context ?? throw new
ArgumentNullException(nameof(context)); // asignar el contexto a la
propiedad
        }

        public async Task<Customer?> GetByIdAsync(CustomerId id) => await
_context.Customers.SingleOrDefaultAsync(c => c.Id == id); // obtener un
cliente por su identificador

        public async Task Add(Customer customer) => await
_context.Customers.AddAsync(customer); // agregar un cliente al contexto

    }
}
```

Vista:



Configuración de entidades (Mapeo)

Se crea una carpeta en Persistence llamada Configuration y en ella se crea la siguiente clase para el mapeo

ConfigurationCustomer.cs

```
using Domain.Customer;
using Domain.ObjetosValor;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Infrastructure.Persistence.Configuration
{
    // Clase para configurar la entidad Customer
    public class CustomerConfiguration :
IEntityTypeConfiguration<Customer>
    {
        // Configuración de la entidad Customer
        public void Configure(EntityTypeBuilder<Customer> builder)
        {

            // builder.ToTable("Customers"); // Configuración de la tabla
Customers

            builder.HasKey(c => c.Id); // Primary Key
            // Configuración de la propiedad Id
            builder.Property(c => c.Id).HasConversion(
                id => id.Value,
                value => new CustomerId(value) // Value Object
            );

            builder.Property(c => c.Name).IsRequired().HasMaxLength(50);
// Configuración de la propiedad Name

            builder.Property(c => c.LastName).HasMaxLength(50); //
Configuración de la propiedad LastName

            builder.Ignore(c => c.FullName); // Ignorar propiedad
FullName

            builder.Property(c => c.Email).HasMaxLength(255); //
Configuración de la propiedad Email

            builder.HasIndex(c => c.Email).IsUnique(); // Configuración
de índice para la propiedad Email

            builder.Property(c => c.PhoneNumber).HasConversion(
```

```
                phone => phone.Value,
                value => PhoneNumber.Create(value)! // Value Object
            ).HasMaxLength(9); // Configuración de la propiedad Phone

            builder.OwnsOne(c => c.Address, a =>
            {
                a.Property(a => a.Street).HasMaxLength(100); //
Configuración de la propiedad Street
                a.Property(a => a.City).HasMaxLength(50); //
Configuración de la propiedad City
                a.Property(a => a.State).HasMaxLength(50); //
Configuración de la propiedad State
                a.Property(a => a.ZipCode).HasMaxLength(10).IsRequired();
// Configuración de la propiedad ZipCode
            });

        }
    }
}
```

Migraciones

Se crea la clase siguiente en la carpeta Infrastructura

DependencyInjection.cs

```
using Application;
using Domain.Customer;
using Domain.Primitivos;
using Infrastructure.Persistence;
using Infrastructure.Persistence.Repositories;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Infrastructure;

//(1) Se crea una clase estática llamada DependencyInjection
public static class DependencyInjection
{
    //(2) Se crea un método de extensión llamado AddInfrastructure
    public static IServiceCollection AddInfrastructure(this
IServiceCollection services, IConfiguration configuration)
    {
        services.AddPersistence(configuration); //Se llama al método
AddPersistence
        return services;
    }
```

```
    //(3) Se crea un método de extensión llamado AddPersistence
    private static IServiceCollection AddPersistence(this
IServiceCollection services, IConfiguration configuration)
    {
        //Se agrega el contexto de la base de datos
        services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(configuration.GetConnectionString("Database")));

        //Se agregan los servicios necesarios para la inyección de
dependencias
        services.AddScoped<IApplicationDbContext>(sp =>
                sp.GetRequiredService<ApplicationDbContext>());
        // Se agrega el UnitOfWork como servicio
        services.AddScoped<IUnitOfWork>(sp =>
                sp.GetRequiredService<ApplicationDbContext>());

        //Se agregan los repositorios necesarios para la inyección de
dependencias
        services.AddScoped<ICustomerRepository, CustomerRepository>();

        return services;
    }
}
```

NOTA: Como denomines en el archivo de configuración siguiente a las credenciales de la BBDD: “`services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(configuration.GetConnectionString("Database")));`”

Busca el archivo ‘appsettings.Development.json’ en la carpeta Web.API

```
{
  "ConnectionStrings": {
    "Database": "Data Source=DESKTOP-3829VRG;Initial
Catalog=tutorial;Integrated Security=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

NOTA: en este caso se ha usado la configuración de SQLserver.

En la carpeta Web.API debemos de crear una carpeta llamada Extensions. Dentro de ella tenemos que crear una clase:

MigrationsExtensions.cs

```
using Infrastructure.Persistence;
using Microsoft.EntityFrameworkCore;

namespace Web.API.Extensions;

// Clase de extensión para aplicar las migraciones a la base de datos
public static class MigrationExtensions
{
    // Método para aplicar las migraciones a la base de datos
    public static void ApplyMigrations(this WebApplication app){
        // Crear un alcance para acceder a los servicios
        using var scope = app.Services.CreateScope();
        // Obtener el contexto de la aplicación
        var dbContext =
scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
        // Aplicar las migraciones a la base de datos
        dbContext.Database.Migrate();
    }
}
```

Debemos de crear otra clase en esta carpeta, Web.API:

DependencyInjections.cs

```
using Infrastructure.Persistence;
using Microsoft.EntityFrameworkCore;

namespace Web.API.Extensions;

// Clase de extensión para aplicar las migraciones a la base de datos
public static class MigrationExtensions
{
    // Método para aplicar las migraciones a la base de datos
    public static void ApplyMigrations(this WebApplication app){
        // Crear un alcance para acceder a los servicios
        using var scope = app.Services.CreateScope();
        // Obtener el contexto de la aplicación
        var dbContext =
scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
        // Aplicar las migraciones a la base de datos
        dbContext.Database.Migrate();
    }
}
```

NOTA: mucho del código de esta clase ha sido refactorizado de la clase program.cs de esta misma carpeta

Nuestra clase program.cs debe de quedar de la siguiente manera:

Program.cs

```
using Application;
using Infrastructure;
using Web.API;
using Web.API.Extensions;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddPresentation() // Añadimos la inyección de
dependencias de presentación
                .AddInfrastructure(builder.Configuration) // Añadimos la
inyección de dependencias de infraestructura
                .AddApplication(); // Añadimos la inyección de
dependencias de aplicación

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
    app.ApplyMigrations(); // Añadimos la migración
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Comandos Terminal

NOTA: la secuencia de instrucciones siguientes no se asegura estar en el orden real, puede que sí o puede que no

NOTA: hubo muchas complicaciones respecto a las versiones de .net y los paquetes a instalar

- Montar las migraciones →

`dotnet ef migrations add InitialMigration -p .\Infrastructure\ -s .\Web.API\ -o .\Infrastructure\Persistence\Migrations\`

NOTA: Se crea en la carpeta Infrastructure\Persistence una carpeta Migrations con tres clases mapeadas con nuestras entidades para la BBDD

Otra forma muy similar (sin autocompletar):

`dotnet ef migrations add InitialMigration -p Infrastructure -s Web.API -o Persistence/Migrations`

- Lanzar la BBDD →

`dotnet ef database update -p Infrastructure -s Web.API`

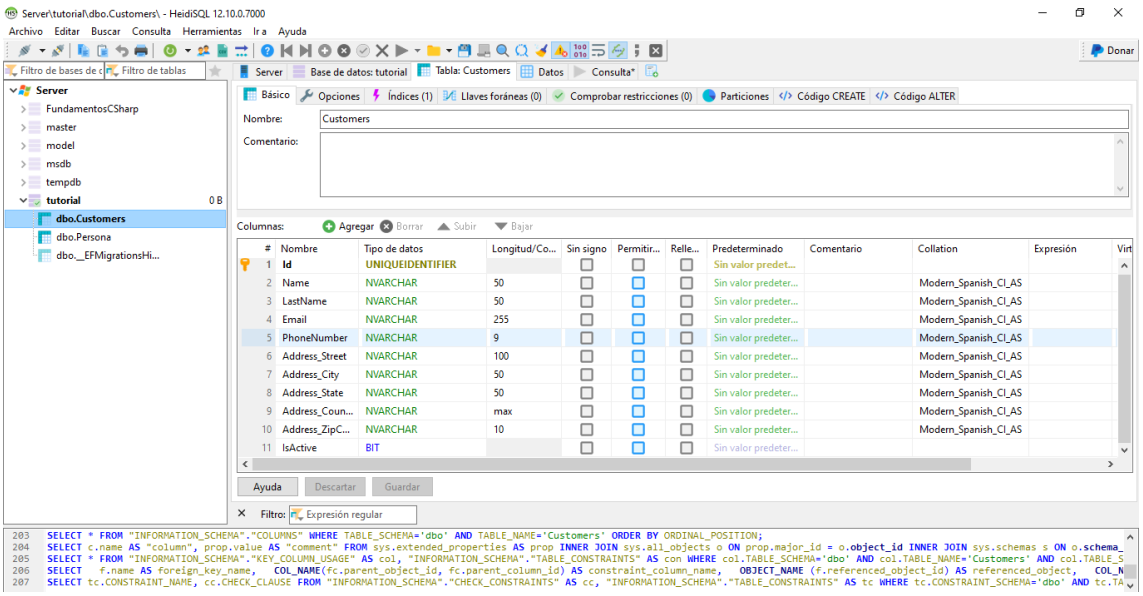
- Limpiar proyecto →

`dotnet clean`

- Restaurar proyecto →

`dotnet restore`

Resultado BBDD en HeidiSql:



Volver hacer una migración (borrar anterior migración, generar nueva migración)

Borra manualmente los archivos de la carpeta Migrations (esta incluida). También los obj (carpeta) de las distintas carpetas (Application, Domain, etc)

Haz una limpieza del proyecto → `dotnet clean`

Vuelve a construir el proyecto → `dotnet build`

Finalmente, vuelve hacer → dotnet ef migrations add InitialMigration -p Infrastructure -s Web.API -o Persistence/Migrations

Y lanzala → dotnet ef database update -p Infrastructure -s Web.API