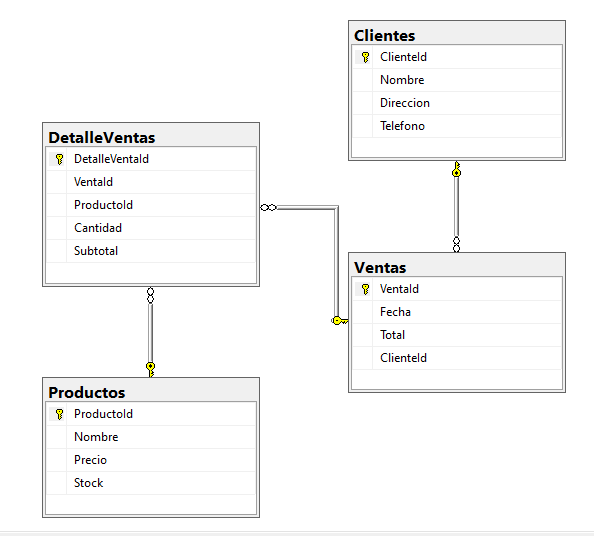
**TALLER .Net**

**CRUD con Visual Studio .Net (C#) con SQL Server**

Realizaremos un CRUD utilizando visual Studio .Net con C# y la arquitectura MVC, considerando la siguiente base de datos:



**Requisitos:**

1. Visual Studio 2019 o superior instalado.
2. .NET 6.0 o superior.
3. Librerías y paquetes adicionales: Entity Framework Core, SQL Server (o una base de datos compatible).

**Pasos para Crear la Aplicación Web**

1. **Crear el proyecto en Visual Studio**:
   * Abrir Visual Studio y seleccionar Crear un nuevo proyecto.
   * Elegir Aplicación Web ASP.NET Core y darle un nombre (por ejemplo: SalesManagementApp).
   * Seleccionar el Modelo-Vista-Controlador (MVC) y asegurarse de que está habilitado Individual User Accounts si se quiere autenticación básica.
2. **Configurar Entity Framework Core**:
   * En Tools > NuGet Package Manager > Package Manager Console, ejecutar:

powershell

Copiar código

Install-Package Microsoft.EntityFrameworkCore

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

* + Configurar la cadena de conexión en el archivo appsettings.json para conectar a SQL Server (u otra base de datos):

json

Copiar Código:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"ConnectionStrings": {

"DefaultConnection": "Server=EDICCSON\\SQLEXPRESS;Database=SalesManagementDB;Trusted\_Connection=True;Encrypt=False;"

},

"AllowedHosts": "\*"

}

1. **Crear el modelo de datos**:
   * Crear una carpeta Models en el proyecto y añadir clases para Producto, Cliente, Venta y DetalleVenta. Aquí están las definiciones básicas:

csharp

Copiar código

// Models/Producto.cs

using System.ComponentModel.DataAnnotations.Schema;

namespace SalesManagementApp.Models

{

public class Producto

{

public int ProductoId { get; set; }

public required string Nombre { get; set; }

[Column(TypeName = "decimal(18,2)")]

public decimal Precio { get; set; }

public int Stock { get; set; }

}

}

// Models/Cliente.cs

namespace SalesManagementApp.Models

{

public class Cliente

{

public int ClienteId { get; set; }

public required string Nombre { get; set; }

public required string Direccion { get; set; }

public required string Telefono { get; set; }

}

}

// Models/Venta.cs

using System.ComponentModel.DataAnnotations.Schema;

namespace SalesManagementApp.Models

{

public class Venta

{

public int VentaId { get; set; }

public DateTime Fecha { get; set; }

[Column(TypeName = "decimal(18,2)")]

public decimal Total { get; set; }

public int ClienteId { get; set; }

public required Cliente Cliente { get; set; }

//public required ICollection<DetalleVenta> Detalles { get; set; }

public required List<DetalleVenta> DetalleVentas { get; set; } // Relación con DetalleVenta

}

}

// Models/DetalleVenta.cs

// Models/DetalleVenta.cs

using System.ComponentModel.DataAnnotations.Schema;

namespace SalesManagementApp.Models

{

public class DetalleVenta

{

public int DetalleVentaId { get; set; }

public int VentaId { get; set; }

public required Venta Venta { get; set; }

public int ProductoId { get; set; }

public required Producto Producto { get; set; }

public int Cantidad { get; set; }

// Especificar el tipo de columna con precisión y escala

[Column(TypeName = "decimal(18,2)")]

public decimal Subtotal { get; set; }

}

}

1. **Configurar el contexto de la base de datos**:
   * Crear la clase ApplicationDbContext en la carpeta Data:

csharp

Copiar código

// Data/ApplicationDbContext.cs

//Recordemos que el EntityFrameworkCore proporciona funcionalidades

// para trabajar con las base de datos en .net

using Microsoft.EntityFrameworkCore;

//Importamos la libreria que contiene los modelos de la aplicación

using WebEmpresa.Models;

namespace WebEmpresa.Data

{

using Microsoft.EntityFrameworkCore;

//Importamos la libreria que contiene los modelos de la aplicación

using WebEmpresa.Models;

//Recordar que DbContext es la clase de EntityFramework para

// trabajar con la base de datos

public class ApplicationDbContext: DbContext

{

//ApplicationDbContext permite la inyeccion dependencias y la configuracion

// de la base de datos

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options) {

}

//Definimos una propiedad dbSet que representa una colección de todas las entidad

// de un tipo de dato que se almacenaran en la base de datos

public DbSet<Producto> Producto { get; set; }

//Ejemplo DbSet<Producto> representa la tabla producto en la base de datos

public DbSet<Cliente> Cliente { get; set; }

public DbSet<Venta> Venta { get; set; }

public DbSet<DetalleVenta> Detalle { get; set; }

//Sobreescribimos ciertas convenciones predeterminadas del entity framework

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//LLamar a la implementacion de la clase base para asegurarnos que

//cualquier configuracion realizada alli se aplique

base.OnModelCreating(modelBuilder);

//Podemos configurar la precision de la escala de las propiedades de decimales

modelBuilder.Entity<Producto>()

.Property(d => d.Precio)

.HasColumnType("decimal(18,2)");

modelBuilder.Entity<Venta>()

.Property(d => d.Total)

.HasColumnType("decimal(18,2)");

modelBuilder.Entity<DetalleVenta>()

.Property(d => d.SubTotal)

.HasColumnType("decimal(18,2)");

}

}

}

Explicación detallada:

// Importamos el espacio de nombres de Entity Framework Core, que nos proporciona funcionalidades para trabajar con bases de datos en .NET.

using Microsoft.EntityFrameworkCore*;*

// Importamos el espacio de nombres que contiene los modelos de la aplicación.

// Estos modelos probablemente representan las entidades o tablas de la base de datos.

using SalesManagementApp.Models*;*

namespace SalesManagementApp.Data

{

    // Definición de la clase *`ApplicationDbContext`* que hereda de *`DbContext`*, la clase base de Entity Framework Core para trabajar con la base de datos.

    public class ApplicationDbContext : DbContext

    {

        // Constructor de la clase *`ApplicationDbContext`* que recibe opciones de configuración y las pasa a la clase base *`DbContext`*.

        // Esto permite la inyección de dependencias o configuraciones como la cadena de conexión a la base de datos.

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

        // Definimos una propiedad *`DbSet`* para cada entidad. *`DbSet`* representa una colección de todas las entidades de un tipo dado que se almacenarán en la base de datos.

        // Es decir, *`DbSet<Producto>`* representa la tabla "Productos" en la base de datos.

        public DbSet<Producto> Productos { get; set; }

        // *`DbSet`* que representa la tabla "Clientes" en la base de datos.

        public DbSet<Cliente> Clientes { get; set; }

        // *`DbSet`* que representa la tabla "Ventas" en la base de datos.

        public DbSet<Venta> Ventas { get; set; }

        // *`DbSet`* que representa la tabla "DetalleVentas" en la base de datos.

        public DbSet<DetalleVenta> DetalleVentas { get; set; }

        // Método que permite configurar el modelo a nivel de la base de datos, especialmente para propiedades de las entidades.

        // Aquí podemos sobrescribir ciertas convenciones predeterminadas de Entity Framework.

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            // Llamamos a la implementación de la clase base para asegurarnos de que cualquier configuración hecha allí también se aplique.

            base.OnModelCreating(modelBuilder);

            // Configuramos manualmente la propiedad *`Subtotal`* de la entidad *`DetalleVenta`* para que sea de tipo decimal con precisión y escala específicas.

            // "decimal(18,2)" significa que el valor decimal puede tener hasta 18 dígitos en total, con 2 de ellos después del punto decimal.

            modelBuilder.Entity<DetalleVenta>()

                .Property(d => d.Subtotal)

                .HasColumnType("decimal(18,2)");

            // Con esto, estamos garantizando que la columna *`Subtotal`* en la tabla *`DetalleVenta`* tenga el tipo y formato adecuado.

        }

    }

}

Anexar la configuración de la base de datos y el contexto en la clase program.cs

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

// Configurar la cadena de conexión desde appsettings.json

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");

// Registrar el contexto de datos con el contenedor de dependencias

builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(connectionString));

// Agregar controladores con vistas (MVC)

builder.Services.AddControllersWithViews();

var app = builder.Build();

1. **Crear los controladores**:
   * Crear controladores para Producto, Cliente, Venta y DetalleVenta usando scaffolding (clic derecho en la carpeta Controllers, Add > Controller y seleccionar MVC Controller with views, using Entity Framework).
   * Seleccionar las respectivas clases y el contexto ApplicationDbContext.
2. **Crear las vistas**:
   * Visual Studio generará automáticamente las vistas CRUD para cada controlador.
3. **Configurar la barra de navegación**:
   * En la vista compartida \_Layout.cshtml, agregar enlaces a Productos, Clientes, Ventas:

html

Anexar opciones de navegación en código el layaout de la aplicación que contiene la barra de navegación en Views\_Shared\_\_Layout.cshtml

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Productos" **asp-action**="Index">Producto</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Clientes" **asp-action**="Index">Cliente</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="" **asp-controller**="Ventas" **asp-action**="Index">Ventas</**a**>

</li>

1. **Migrar la base de datos**:
   * En Package Manager Console, ejecutar:

powershell

Copiar código

Add-Migration InitialCreate

Update-Database

1. Esto creará la base de datos y las tablas necesarias.
2. **Ejecutar y probar la aplicación**:
   * Ejecutar la aplicación (F5 o Ctrl + F5) y verificar que se pueda navegar a las vistas de Productos, Clientes y Ventas.