**GUIDE FULLSTACK**

¿Qué vas a aprender en esta guía?

**SQL SERVER**

Es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado por Microsoft. Se usa para almacenar, administrar y recuperar datos de manera eficiente. Soporta T-SQL, procedimientos almacenados y consultas avanzadas.

**Descargar:** Ve a <https://www.microsoft.com/en-us/sql-server/sql-server-downloads> y descarga la versión adecuada (Developer o Express para pruebas).

**Instalar**: Ejecuta el instalador, elige "Instalación personalizada" o "Instalación básica".

Configurar: Define la autenticación (Windows o mixta) y selecciona los componentes necesarios (Motor de base de datos, SSMS, etc.).

**Instalar** SQL Server Management Studio (SSMS): <https://learn.microsoft.com/en-us/ssms/download-sql-server-management-studio-ssms>

**Descárgalo** e instálalo para administrar tus bases de datos visualmente.

**NOTA.**

**RDBMS** (Relational Database Management System) → Sistema de gestión de bases de datos relacional que organiza la información en tablas con claves y relaciones (Ej: SQL Server, MySQL, Oracle).

**T-SQL** (Transact-SQL) → Extensión de SQL utilizada en SQL Server que incluye programación estructurada como variables, condicionales y bucles.

SSMS (SQL Server Management Studio) → Herramienta de Microsoft para administrar bases de datos SQL Server mediante una interfaz gráfica y consultas SQL.

**Framework** → Conjunto de herramientas, bibliotecas y reglas que facilitan el desarrollo de software (Ej: .NET, Angular, Spring Boot).

**SDK** (Software Development Kit) → Paquete de herramientas, bibliotecas y documentación para desarrollar aplicaciones en una plataforma específica (Ej: .NET SDK, Android SDK).

**C#**

Es un lenguaje de programación desarrollado por Microsoft, orientado a objetos y utilizado para desarrollar aplicaciones de escritorio, web, móviles y videojuegos.

**.NET**

Es un framework de desarrollo que proporciona herramientas y bibliotecas para crear aplicaciones en diferentes plataformas. Incluye .NET Core (multiplataforma) y .NET Framework (Windows).

Descargar e instalar .NET SDK desde <https://dotnet.microsoft.com/es-es/>

Instalar un IDE como Visual Studio o Visual Studio Code: <https://code.visualstudio.com/>

Probar la instalación ejecutando en la terminal:

dotnet –versión

**ANGULAR**

Es un framework de JavaScript desarrollado por Google para construir aplicaciones web de una sola página (SPA) con TypeScript. Es ideal para crear interfaces dinámicas y escalables.

Instalar Node.js (incluye npm) desde <https://nodejs.org/es>

Instalar Angular CLI (herramienta de línea de comandos):

npm install -g @angular/cli

**JAVASCRIPT (JS)**

Es un lenguaje de programación interpretado y orientado a objetos que se usa principalmente para crear páginas web interactivas. Se ejecuta en el navegador y permite manipular el DOM, hacer peticiones a servidores y manejar eventos.

Lenguaje de alto nivel y dinámico.

Compatible con HTML y CSS.

Permite programación asincrónica con promesas y async/await.

Utilizado tanto en el frontend (con frameworks como Angular, React, Vue) como en el backend (con Node.js).

console.log("Hola, JavaScript!");

**BOOTSTRAP**

Es un framework de CSS desarrollado por Twitter que facilita la creación de sitios web responsivos y con un diseño atractivo sin necesidad de escribir mucho código CSS.

Basado en HTML, CSS y JavaScript.

Diseño responsivo con un sistema de rejilla (grid).

Componentes listos para usar (botones, alertas, modales, etc.).

Compatible con frameworks como Angular y React.

npm install Bootstrap

<https://startbootstrap.com/>

**NOTA.**

**IDE** (Integrated Development Environment) → Entorno de desarrollo que incluye editor de código, depurador y herramientas para programar (Ej: Visual Studio, IntelliJ, VS Code).

**Terminal** → Interfaz de línea de comandos donde puedes ejecutar órdenes y scripts en tu sistema operativo o entorno de desarrollo.

**NPM** (Node Package Manager) → Administrador de paquetes para JavaScript y Node.js que permite instalar librerías y frameworks.

**DOM** (Document Object Model) → Representación estructurada en forma de árbol de un documento HTML que permite manipular sus elementos con JavaScript.

**HTML** (HyperText Markup Language) → Lenguaje de marcado utilizado para estructurar el contenido de las páginas web.

**CSS** (Cascading Style Sheets) → Lenguaje de estilos que define la apariencia y diseño de los elementos HTML en una web.

**Async** (Asynchronous) → Palabra clave en JavaScript que indica que una función trabaja de forma asíncrona, sin bloquear la ejecución del código.

**Await** → Palabra clave en JavaScript usada dentro de funciones async para pausar la ejecución hasta que una promesa se resuelva.

async function obtenerDatos() {

let respuesta = await fetch('https://api.example.com/datos');

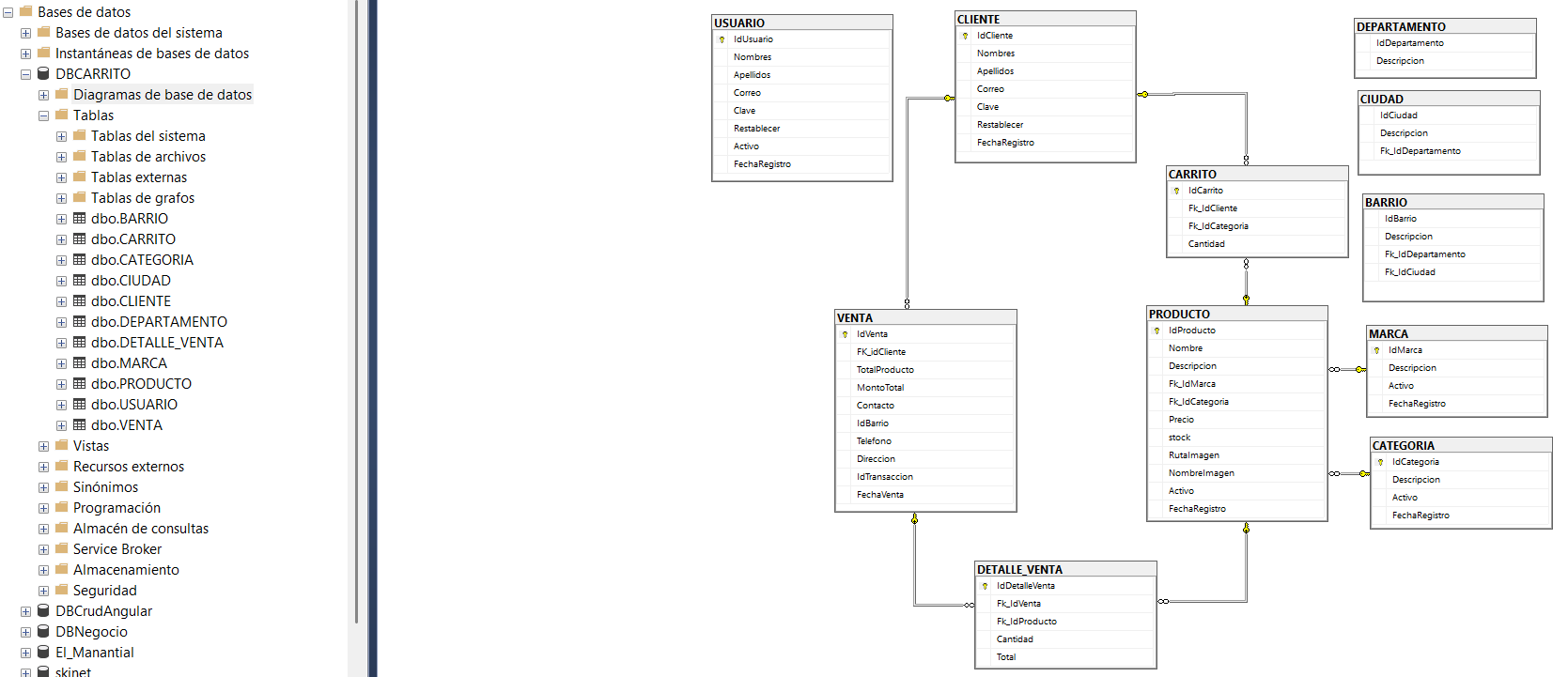
let datos = await respuesta.json();

console.log(datos);

}

SQL SERVER

1. Creamos el diagrama de la base de datos que utilizaremos



1. En un nuevo Query crearemos todas las características de nuestra base de datos.

create database MANANTIAL

Go

use MANANTIAL

Go

create table CATEGORIA(

IdCategoria int primary key identity,

Descripcion varchar(100),

Activo bit default 1,

FechaRegistro datetime default getdate()

)

Go

create table MARCA(

IdMarca int primary key identity,

Descripcion varchar(100),

Activo bit default 1,

FechaRegistro datetime default getdate()

)

Go

create table PRODUCTO(

IdProducto int primary key identity,

Nombre varchar(500),

Descripcion varchar(500),

Fk\_IdMarca int references MARCA(IdMarca),

Fk\_IdCategoria int references CATEGORIA(IdCategoria),

Precio decimal(10,2) default 0,

stock int,

RutaImagen varchar(100),

NombreImagen varchar(100),

Activo bit default 1,

FechaRegistro datetime default getdate()

)

Go

create table CLIENTE(

IdCliente int primary key identity,

Nombres varchar(100),

Apellidos varchar(100),

Correo varchar(100),

Clave varchar(150),

Restablecer bit default 0,

FechaRegistro datetime default getdate()

)

Go

create table CARRITO(

IdCarrito int primary key identity,

Fk\_IdCliente int references CLIENTE(IdCliente),

Fk\_IdCategoria int references PRODUCTO(IdProducto),

Cantidad int

)

Go

create table VENTA(

IdVenta int primary key identity,

FK\_idCliente int references CLIENTE(idCliente),

TotalProducto int,

MontoTotal decimal(10,2),

Contacto varchar(50),

FK\_IdBarrio varchar(10),

Telefono varchar(50),

Direccion varchar(500),

IdTransaccion varchar(50),

FechaVenta datetime default getdate()

)

Go

create table DETALLE\_VENTA(

IdDetalleVenta int primary key identity,

Fk\_IdVenta int references VENTA(IdVenta),

Fk\_IdProducto int references PRODUCTO(IdProducto),

Cantidad int,

Total decimal(10,2)

)

Go

create table USUARIO(

IdUsuario int primary key identity,

Nombres varchar(100),

Apellidos varchar(100),

Correo varchar(100),

Clave varchar(150),

Restablecer bit default 1,

Activo bit default 1,

FechaRegistro datetime default getdate()

)

Go

create table DEPARTAMENTO(

IdDepartamento varchar(2) NOT NULL,

Descripcion varchar (45) NOT NULL

)

Go

create table CIUDAD(

IdCiudad varchar(4) NOT NULL,

Descripcion varchar (45) NOT NULL,

Fk\_IdDepartamento varchar (2) NOT NULL

)

Go

create table BARRIO(

IdBarrio varchar(6) NOT NULL,

Descripcion varchar (45) NOT NULL,

Fk\_IdDepartamento varchar (2) NOT NULL,

Fk\_IdCiudad varchar (4) NOT NULL

)

Go

**NOTA.**

**Conceptos Fundamentales**

UML → Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar sistemas de software.

Base de datos → Conjunto de datos organizados.

Tablas → Estructuras que almacenan datos en filas y columnas.

Registros (Filas) → Cada entrada en una tabla.

Columnas (Campos) → Definen el tipo de datos en una tabla.

Tipos de datos → INT, VARCHAR, DATE, DECIMAL, etc.

Clave primaria (PRIMARY KEY) → Identifica de manera única cada registro.

Clave foránea (FOREIGN KEY) → Relaciona una tabla con otra.

Índices → Mejoran la velocidad de búsqueda en la base de datos.

**Consultas y Manipulación de Datos (DML - Data Manipulation Language)**

Querys → Consultas SQL para recuperar, insertar, actualizar o eliminar datos.

SELECT → Recuperar datos de la base de datos.

INSERT → Agregar nuevos registros en una tabla.

UPDATE → Modificar registros existentes.

DELETE → Eliminar registros de una tabla.

MERGE → Combina INSERT, UPDATE y DELETE en una sola operación.

**Filtrado y Ordenación de Datos**

FROM → Especifica la tabla de origen de los datos.

WHERE → Filtra registros según condiciones.

HAVING → Filtra resultados después de aplicar GROUP BY.

ORDER BY → Ordena los resultados.

GROUP BY → Agrupa registros para funciones de agregación.

DISTINCT → Elimina valores duplicados.

**Operadores de Comparación**

= (Igual a)

<> o != (Diferente de)

> (Mayor que)

< (Menor que)

>= (Mayor o igual que)

<= (Menor o igual que)

BETWEEN (Filtra un rango de valores)

**Operadores Lógicos**

AND (Ambas condiciones deben cumplirse)

OR (Al menos una condición debe cumplirse)

NOT (Niega una condición)

**Búsqueda de Texto y Listas**

LIKE → Busca patrones en texto.

IN → Busca valores dentro de una lista.

NOT IN → Excluye valores de una lista.

EXISTS → Verifica si una subconsulta devuelve resultados.

**Combinación de Datos**

JOIN → Combina datos de múltiples tablas.

INNER JOIN → Devuelve solo coincidencias.

LEFT JOIN → Devuelve todos los registros de la tabla izquierda y las coincidencias en la derecha.

RIGHT JOIN → Devuelve todos los registros de la tabla derecha y las coincidencias en la izquierda.

FULL JOIN → Devuelve todos los registros de ambas tablas.

CROSS JOIN → Devuelve todas las combinaciones posibles.

**Definición y Estructura de la Base de Datos (DDL - Data Definition Language)**

CREATE DATABASE → Crea una base de datos.

ALTER DATABASE → Modifica una base de datos.

DROP DATABASE → Elimina una base de datos.

CREATE TABLE → Crea una tabla nueva.

ALTER TABLE → Modifica una tabla existente.

DROP TABLE → Elimina una tabla.

TRUNCATE TABLE → Elimina todos los registros de una tabla sin afectar la estructura.

**Objetos de Base de Datos Avanzados**

Views (Vistas) → Consultas almacenadas que actúan como tablas virtuales.

Stored Procedures (Procedimientos almacenados) → Bloques de código SQL reutilizables.

Functions (Funciones) → Devuelven un valor basado en parámetros.

Triggers (Disparadores) → Procedimientos que se ejecutan automáticamente en respuesta a eventos (INSERT, UPDATE, DELETE).

**Control y Seguridad (DCL - Data Control Language)**

GRANT → Otorga permisos a usuarios o roles.

REVOKE → Revoca permisos concedidos.

DENY → Niega permisos específicos.

Usuarios y roles → Controlan el acceso a la base de datos.

**Control de Transacciones (TCL - Transaction Control Language)**

BEGIN TRANSACTION → Inicia una transacción.

COMMIT → Guarda los cambios de una transacción.

ROLLBACK → Revierte los cambios de una transacción.

SAVEPOINT → Crea un punto de restauración dentro de una transacción.

**Administración y Mantenimiento de Base de Datos**

Índices → Mejoran la velocidad de las consultas.

Backups y recuperación → Para evitar la pérdida de datos.

BACKUP DATABASE → Crea una copia de seguridad.

RESTORE DATABASE → Restaura una copia de seguridad.

CHECKPOINT → Guarda los cambios en disco para mejorar el rendimiento.

**Funciones Útiles en SQL Server**

Funciones Agregadas

COUNT

SUM

AVG

MIN

MAX

Funciones de Texto

LEN

UPPER

LOWER

SUBSTRING

REPLACE

Funciones de Fecha y Hora

GETDATE

DATEADD

DATEDIFF

FORMAT

Funciones de Conversión

CAST

CONVERT

**NOTA.**

**Antes de comenzar:**

OOP (Programación Orientada a Objetos): Paradigma basado en clases y objetos.

Algoritmo: Conjunto de pasos para resolver un problema.

Variable: Espacio en memoria para almacenar datos.

Condicional: Estructura que ejecuta código según una condición.

Bucle: Repetición de código mientras se cumpla una condición.

Función: Bloque de código reutilizable con una tarea específica.

Recursión: Función que se llama a sí misma para resolver un problema.

Excepción: Error en tiempo de ejecución que puede manejarse.

Depuración: Proceso de encontrar y corregir errores en el código.

DRY (Don't Repeat Yourself): Principio de evitar código duplicado.

KISS (Keep It Simple, Stupid): Escribir código simple y claro.

YAGNI (You Aren’t Gonna Need It): No programar funciones innecesarias.

Postman: Herramienta para probar APIs.

Git: Sistema de control de versiones distribuido.

Repositorio: Espacio donde se almacena código y cambios.

Library (Librería): Conjunto de funciones reutilizables.

Agile: Metodología de desarrollo flexible e iterativa.

Scrum: Marco de trabajo ágil basado en sprints.

Encryption (Encriptación): Proceso de codificar datos para protegerlos.

Autenticación: Verificación de identidad de un usuario.

Autorización: Permisos otorgados a un usuario autenticado.

**BACKEND CON C# Y .NET**

1. Crearemos una carpeta la cual será nuestro espacio de trabajo.
2. Agregamos dos nuevas herramientas que utilizaremos a lo largo del curso
3. Posman: <https://www.postman.com/downloads/>
4. Git: <https://git-scm.com/downloads>
5. Crear una cuenta en GITHUB https://github.com/
6. Ten encuenta estos comando git para el curso:

git config --global user.name "Tu Nombre" → Configurar nombre de usuario

git config --global user.email tu\_email@example.com → Configurar correo electrónico

git init → Inicializar un repositorio

git add . → Agregar todos los archivos al área de preparación

git commit -m "Mensaje del commit" → Guardar cambios con un mensaje

git branch -M master → Renombrar la rama principal a "master"

git remote add origin URL\_DEL\_REPOSITORIO → Vincular con un repositorio remoto

git pull origin master → Obtener cambios del repositorio remoto

git push -u origin master → Subir cambios al repositorio remoto

git branch nombre\_de\_rama → Crear una nueva rama

git checkout nombre\_de\_rama → Cambiar de rama

git switch nombre\_de\_rama → Cambiar de rama (comando moderno)

git switch -c nombre\_de\_rama → Crear y cambiar a una nueva rama

git status → Ver cambios pendientes

git log --oneline --graph --decorate --all → Ver historial de commits de forma compacta

git restore archivo.txt → Restaurar un archivo antes del commit

git restore --staged archivo.txt → Quitar un archivo del área de preparación

git reset --soft HEAD~1 → Revertir el último commit sin perder cambios

git reset --hard HEAD~1 → Eliminar el último commit perdiendo cambios

git revert ID\_DEL\_COMMIT → Crear un nuevo commit que deshace un commit anterior

git rm --cached archivo.txt → Eliminar un archivo solo del repositorio sin borrar localmente

git merge --abort → Cancelar un merge con conflictos

git fetch --all → Obtener todos los cambios remotos sin fusionar

git reset --hard origin/master → Forzar la actualización del código local con el remoto

git branch -d nombre\_de\_rama → Eliminar una rama localmente (si ya fue fusionada)

git branch -D nombre\_de\_rama → Forzar la eliminación de una rama local

git push origin --delete nombre\_de\_rama → Eliminar una rama en el repositorio remoto

1. Iniciaremos nuestro proyecto, abriendo VSC en la terminal insertamos:

dotnet new sln -n Manantial

1. capa mvc de administracion:

dotnet new mvc -n Admin

1. capa webapi del e-commer

dotnet new webapi -n Api

1. capa cliente angular

ng new Client --style=css

Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N) N

1. En la Capa Api instalar

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Swashbuckle.AspNetCore

1. Agregar nueva capa Core, Infraestructure y Application

dotnet new classlib -n Core

dotnet new classlib -n Infraestructure

dotnet new classlib -n Application

1. Agregar los proyectos a la solución

dotnet sln add Api/Api.csproj

dotnet sln add Application/Application.csproj

dotnet sln add Core/Core.csproj

dotnet sln add Infraestructure/Infraestructure.csproj

dotnet sln add Admin/Admin.csproj

dotnet sln add Client

1. Referenciar Application en Api, Core y Infraestructure

dotnet add Api/Api.csproj reference Application/Application.csproj

dotnet add Application/Application.csproj reference Core/Core.csproj

dotnet add Application/Application.csproj reference Infraestructure/Infraestructure.csproj

1. Referenciar Infraestructure en Core.

dotnet add Infraestructure/Infraestructure.csproj reference Core/Core.csproj

1. Verificar Referencias.

dotnet list Api/Api.csproj reference

dotnet list Application/Application.csproj reference

dotnet list Infraestructure/Infraestructure.csproj reference

1. Restaurar dependencias

dotnet restore

1. Instalar dependencias en Infraestructure

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

1. Instalar AutoMapper en la Capa Application (para mapeo de DTOs)

dotnet add Api/Api.csproj package AutoMapper.Extensions.Microsoft.DependencyInjection

1. Instalar Bootstrap para el cliente (Capa Client)

npm install bootstrap

1. Angular.json

"styles": [

"node\_modules/bootstrap/dist/css/bootstrap.min.css",

"src/styles.css"

],

1. Instalar Angular HTTP Client (para consumir la API)

npm install @angular/common@latest

1. se modifica el archivo appsettings.json

"ConnectionStrings": {

"CadenaConexion": "Server=SIULSERROTAM\\MYSQL;Initial Catalog=Manantial;User Id=sa;Password=luis1994;Encrypt=False;TrustServerCertificate=True;"

}

1. Crear en la carpeta principal el archivo .gitignore

obj

bin

appsettings.json

\*.db

**NOTA.**

**Tecnologías y Patrones**

UX/UI: Experiencia (UX) y diseño visual (UI) de usuario.

MVC (Model-View-Controller): Patrón de diseño que separa lógica, vista y control.

WCF (Windows Communication Foundation): Framework de Microsoft para servicios web.

WPF (Windows Presentation Foundation): Framework para aplicaciones de escritorio en .NET.

SOAP (Simple Object Access Protocol): Protocolo de comunicación basado en XML.

REST (Representational State Transfer): Estilo de arquitectura para servicios web.

VB.NET: Lenguaje de programación basado en .NET.

ADO.NET: Tecnología de acceso a bases de datos en .NET.

ASP.NET: Framework de Microsoft para aplicaciones web.

SOLID: Principios de diseño para software mantenible y escalable.

LINQ (Language Integrated Query): Consulta de datos integrada en C#.

**Arquitectura de Software**

Capa Admin: Parte de la arquitectura encargada de la gestión administrativa.

Capa API: Capa que expone funcionalidades a otras aplicaciones.

Capa Core: Contiene la lógica central del negocio.

Capa Infrastructure: Implementa la infraestructura (BD, servicios externos, etc.).

Capa Application: Orquesta la lógica de negocio y la infraestructura.

Capa Client: Interfaz de usuario o frontend.

**Conceptos de Programación**

Entidad: Objeto que representa datos en un sistema.

Abstract (Clase Abstracta): Clase que no se puede instanciar directamente.

Class (Clase): Plantilla para crear objetos en OOP.

CRUD (Create, Read, Update, Delete): Operaciones básicas en bases de datos.

1. EntidadBase (namespace Core.Entities)

Entidad: Representa un objeto del dominio con atributos y comportamientos.

Clase: Modelo que define un conjunto de propiedades y métodos.

Interfaces: Definen contratos sin implementar lógica, estableciendo reglas para las clases.

1. IEspecificacion (namespace Core.Interfaces)

Inyección de Dependencias: Técnica para gestionar dependencias de clases y mejorar la mantenibilidad del código.

using System.Linq.Expressions: Permite construir consultas dinámicas en LINQ mediante expresiones lambda.

1. IRepositorioGenerico (namespace Core.Interfaces)

Método: Bloque de código que realiza una tarea específica.

Task: Representa una operación asíncrona en .NET, utilizada en programación concurrente.

1. IRepositorioProducto (namespace Core.Interfaces)

Herencia: Mecanismo que permite que una clase derive de otra y reutilice su funcionalidad.

Polimorfismo: Capacidad de una clase de implementar métodos de diferentes formas.

Nota. En las interfaces s obligatorio agregar todos los métodos creados en una herencia.

1. EspecificacionBase (namespace Core.Specifications)

using System.Linq.Expressions: Facilita la creación de expresiones lambda para consultas.

Propiedad: Característica de una clase que tiene un valor (getter/setter).

Void: Tipo de retorno que indica que un método no devuelve valor.

Controller: En el patrón MVC, gestiona la lógica de la aplicación y responde a las peticiones.

1. ControladorBaseApi (namespace API.Controllers)

using Microsoft.AspNetCore.Mvc: Proporciona funcionalidades para construir API REST en ASP.NET Core.

Router: Define rutas para dirigir solicitudes a controladores o funciones.

1. Categoria, Marca y Producto (namespace Core.Entities)

Objeto: Instancia de una clase que contiene datos y métodos.

throw: Instrucción que lanza una excepción en .NET para manejar errores.

Any: Método de LINQ que verifica si una colección tiene al menos un elemento.

DTOs: (Data Transfer Objects) Clases diseñadas para transferir datos entre capas de una aplicación.

1. DtoProducto (namespace Application.DTOs)

|  |  |  |
| --- | --- | --- |
| Característica | Entities (Entidades) | Models (DTOs / Modelos de Vista) |
| Ubicación | Core/Entities | Application/Models |
| Propósito | Mapea tablas de la BD | Transfiere datos en la API |
| Incluye validaciones? | No | Sí (puede incluirlas) |
| Contiene datos sensibles? | Sí | No (se pueden ocultar datos) |
| Relaciones con otras tablas? | Sí (con EF) | No (solo muestra lo necesario) |

Models: Clases que representan datos en una aplicación, generalmente en MVC o API.

1. wwroot\images

Carpeta donde se almacenan archivos estáticos, como imágenes, en ASP.NET Core.

1. EspecificacionProductosConCategoriaYMarca (namespace Core.Specifications)

Filtrar Productos por características especificas

1. EvaluadorDeEspecificaciones (namespace Infraestructure.Data)

Constructor: Método especial que inicializa una instancia de una clase.

Var: Tipo implícito en C# que permite inferir el tipo de una variable.

TEntity: Parámetro genérico que representa una entidad en un repositorio o consulta.

1. ContextoTienda (namespace Infraestructure.Data)

using System.Reflection: Permite obtener información sobre ensamblados, clases y métodos en tiempo de ejecución.

using Microsoft.EntityFrameworkCore: Biblioteca de acceso a datos para .NET basada en ORM.

Override: Modificador que indica que un método sobrescribe una implementación en una clase base.

Restrict: Opción en restricciones de clave foránea que impide eliminar registros relacionados.

Cascade: Propiedad de relaciones en bases de datos que permite eliminar o actualizar registros en cascada.

Assembly: Unidad de compilación en .NET que contiene metadatos, código y recursos.

1. RespuestaApi (namespace API.Errors)

Clase que define respuestas estándar para errores en la API.

1. MiddlewareExcepciones (namespace Api.Errors)

using System.Net: Espacio de nombres que proporciona funciones para trabajar con redes y protocolos.

using System.Text.Json: Biblioteca para serialización y deserialización de JSON en .NET.

1. RespuestaErrorValidacionApi (namespace API.Errors)

IEnumerable: Interfaz que permite iterar sobre una colección de elementos.

1. ExcepcionApi (namespace API.Errors)

Middleware: Componente que procesa solicitudes HTTP en ASP.NET, permitiendo manejar errores y autenticación.

1. MiddlewareExcepcion (namespace Api.Middleware)

using System.Net: Contiene tipos para comunicación en red.

using System.Text.Json: Biblioteca para manejar JSON en .NET.

Readonly: Modificador que impide modificar una variable después de su inicialización.

Try: Bloque que captura excepciones en código.

Catch: Maneja errores capturados en un bloque try.

Helpers: Clases o métodos auxiliares que facilitan tareas comunes.

1. ProductoUrlResolver (namespace Application.Helpers)

using AutoMapper: Biblioteca para mapear objetos de diferentes tipos.

using Microsoft.Extensions.Configuration: Proporciona acceso a configuraciones de la aplicación.

destMember: Parámetro usado en mapeo de objetos para definir el miembro de destino.

Empty: Indica una estructura vacía, como string.Empty para cadenas vacías.

source: En mapeo o consultas, representa el origen de los datos.

1. PerfilesDeMapeo (namespace Application.Helpers)

using AutoMapper: Define perfiles para transformar objetos entre capas de la aplicación.

1. BuggyController (namespace API.Controllers)

using Microsoft.AspNetCore.Mvc: Indica que es un controlador de API en ASP.NET Core.

1. ErrorController (namespace API.Controllers)

using Microsoft.AspNetCore.Mvc: Maneja errores en la API.

1. ProductosController (namespace API.Controllers)

using AutoMapper: Facilita la conversión entre entidades y DTOs.

using Microsoft.AspNetCore.Mvc: Permite la gestión de solicitudes HTTP.

Extensions: Métodos de extensión que agregan funcionalidades a tipos existentes sin modificar su código fuente.

1. RepositorioGenerico (namespace Infraestructure.Repositories)

|  |  |  |
| --- | --- | --- |
| **Aspecto** | **IRepositorioGenerico (Interfaz)** | **RepositorioGenerico (Clase)** |
| **Propósito** | Definir el contrato de los métodos. | Implementar la lógica concreta de acceso a datos. |
| **Abstracción** | Proporciona abstracción, no contiene lógica. | Contiene la lógica real de las operaciones. |
| **Flexibilidad** | Permite cambiar la implementación sin afectar otras clases. | Concreta la implementación y lógica de acceso a datos. |
| **Dependencias** | Se utiliza para desacoplar el código que usa el repositorio. | Depende de la interfaz para garantizar la flexibilidad. |

using Microsoft.EntityFrameworkCore: Interactúa con la base de datos usando Entity Framework Core.

FindAsync: Método para buscar registros de manera asíncrona en la base de datos.

1. RepositorioProducto (namespace Infraestructure.Repositories)

using Microsoft.EntityFrameworkCore: Permite la interacción con la base de datos mediante ORM.

1. ApplicationServicesExtensions (namespace API.Extensions)

using Microsoft.AspNetCore.Mvc: Proporciona funcionalidades para construir APIs en ASP.NET Core.

static: Modificador que indica que una clase o método pertenece a la clase en sí, no a una instancia.

Services: Clases que encapsulan lógica de negocio o de aplicación.

1. ServicioCategoria (namespace Manantial.Application.Services)

Servicio que maneja la lógica de negocio relacionada con categorías de productos.

1. ServicioMarca (namespace Application.Services)

Servicio que administra la lógica de negocio para marcas.

1. ServicioProducto (namespace Application.Services)

Servicio encargado de la lógica de productos dentro de la aplicación.

1. ConfiguracionProducto (Infraestructure.Data.Config)

using Microsoft.EntityFrameworkCore: Espacio de nombres que proporciona herramientas para interactuar con bases de datos mediante Entity Framework Core.

using Microsoft.EntityFrameworkCore.Metadata.Builders: Permite configurar entidades y sus propiedades en la base de datos utilizando Fluent API.

1. IServicioCorreo (namespace Application.Interfaces)

Definir la lógica del negocio especifico de la Aplicación.

|  |  |  |
| --- | --- | --- |
| **Características** | **Interfaz de Repositorio** | **Interfaz de Servicio** |
| **Responsabilidad** | Define los métodos de acceso y manipulación de datos (CRUD). | Define la lógica de negocio, orquestando los repositorios. |
| **Operaciones** | Métodos como GetByIdAsync(), AddAsync(), DeleteAsync(), etc. | Métodos como Create(), Update(), Process(), etc. |
| **Nivel de abstracción** | Bajo nivel (acceso a datos) | Alto nivel (casos de uso y reglas de negocio) |
| **Ubicación** | Capa **Core** | Capa **Application** |
| **Uso principal** | Interacción con la base de datos. | Implementación de casos de uso y lógica de negocio. |

1. Program.cs (API)

using Microsoft.EntityFrameworkCore: Se usa para configurar y utilizar Entity Framework Core en la aplicación.

using Microsoft.OpenApi.Models: Proporciona herramientas para documentar APIs con Swagger en ASP.NET Core.

using Microsoft.AspNetCore.Diagnostics: Contiene middleware para manejar errores y excepciones en la aplicación.

using Microsoft.AspNetCore.Http: Proporciona funcionalidades para manejar solicitudes y respuestas HTTP.

using System.Text.Json: Biblioteca para la serialización y deserialización de JSON en .NET.

using Microsoft.Extensions.DependencyInjection: Permite registrar servicios y gestionar la inyección de dependencias.

using Microsoft.AspNetCore.Mvc: Se usa para definir controladores y manejar solicitudes HTTP en ASP.NET Core.

1. Se Implementa la migración

Dotnet ef database drop -p Infraestructure -s API

Dotnet ef migrations remove -p Infraestructure -s API

dotnet ef migrations add InitialCreate -p Infraestructure -s API -o Data/Migrations

Nota. Detener y Eliminar la Migración en Infraesturcture y la BD en Sql Server

dotnet ef database update en la Capa Api

1. Se corre la api

dotnet watch run --project Api/Api.csproj

**ESTRUCTURA BACKEND**

Api

Controllers

BuggyController.cs

ControladorBaseApi.cs

ErrorController.cs

ProductosControlador.cs

Errors

ExcepcionApi.cs

MiddlewareExcepciones.cs

RespuestaApi.cs

RespuestaErrorValidacionApi.cs

Extensions

ApplicationServicesExtensions.cs

SwaggerServicesExtensions.cs

Middleware

MiddlewareExcepcion.cs

Properties

launchSettings.json

Api.csproj

Api.http

appsettings.Development.json

Program.cs

Application

Dtos

DtoProducto.cs

Helpers

PerfilesDeMapeo.cs

ProductoUrlResolver.cs

Interfaces

IServicioCorreo.cs

Models

(vacio)

Services

ServicioProducto.cs

wwwroot/images

(vacio)

Application.csproj

Core

Entities

EntidadBase.cs

Producto.cs

Interfaces

IEspecificacion.cs

IRepositorioGenerico.cs

IRepositorioProducto.cs

Specifications

EspecificacionBase.cs

EspecificacionProductosConCategoriaYMarca.cs

Core.csproj

Infraestructura

Data

Config

ConfiguracionProducto.cs

Migrations

20250329021620\_InitialCreate.cs

20250329021620\_InitialCreate.Designer.cs

ContextoTiendaModelSnapshot.cs

Seed

categorias.json

marcas.json

productos.json

ContextoTienda.cs

EvaluadorDeEspecificaciones.cs

SemillaContextoTienda.cs

Repositories

RepositorioGenerico.cs

RepositorioProducto.cs

Infraestructure.csproj

.gitignore

Manantial.sln

README.md

1. Infraestructura (Base de datos y migraciones).
2. Core (Entidades, interfaces, specification y lógica de negocio).
3. Aplicación (DTOs y servicios de aplicación).
4. API (Controladores, rutas, Helpers, Middleware, Extensions y Models
5. Administración (Backend para gestión del sistema).
6. Cliente (Frontend: Angular, React, etc.).