Antes de Empezar

Fundamentos Básicos de Programación

Algoritmo

Qué es: Conjunto de pasos o instrucciones que se siguen para resolver un problema o realizar una tarea específica.

Por qué es importante: Todo programa es esencialmente un algoritmo que resuelve algún tipo de problema.

Variable

Qué es: Espacio en memoria que almacena datos que pueden cambiar durante la ejecución del programa.

Por qué es importante: Esencial para almacenar y manipular información.

Tipo de dato (Data type)

Qué es: Especifica el tipo de valor que una variable puede almacenar (por ejemplo, número entero, cadena de texto, etc.).

Por qué es importante: Ayuda a garantizar que los datos se manejen correctamente.

Condicional

(if, else , foreach, else if, switch, ?: , && (AND), || (OR) y ! (NOT),)

Qué es: Estructura que permite ejecutar diferentes bloques de código según ciertas condiciones.

Por qué es importante: Permite que el programa tome decisiones y ejecute acciones diferentes basadas en entradas o situaciones.

Bucle

(Loop, while, do-while)

Qué es: Estructura que permite ejecutar repetidamente un bloque de código mientras se cumpla una condición.

Por qué es importante: Facilita la repetición de tareas sin escribir código redundante.

Función

Qué es: Bloque de código que realiza una tarea específica y puede ser reutilizado en diferentes partes del programa.

Por qué es importante: Permite organizar el código y evitar la repetición de lógica.

Conceptos Avanzados

Recursión

Qué es: Técnica donde una función se llama a sí misma para resolver un problema.

Por qué es importante: Útil para resolver problemas complejos de manera eficiente.

Excepción (Exception)

Qué es: Evento que ocurre durante la ejecución del programa que interrumpe su flujo normal.

Por qué es importante: Permite manejar errores y problemas inesperados sin que el programa se caiga.

Depuración (Debugging)

Qué es: Proceso de identificar y corregir errores en el código.

Por qué es importante: Asegura que el programa funcione correctamente y ayuda a resolver problemas de ejecución.

Principios y Buenas Prácticas

DRY (Don't Repeat Yourself)

Qué es: Principio de programación que promueve la reutilización de código y evita la duplicación.

Por qué es importante: Mejora la mantenibilidad y reduce errores.

KISS (Keep It Simple, Stupid)

Qué es: Principio que recomienda mantener las soluciones lo más simples y claras posibles.

Por qué es importante: El código más sencillo es más fácil de entender y mantener.

YAGNI (You Aren't Gonna Need It)

Qué es: Principio que aconseja no agregar funcionalidades o complejidad innecesaria que no se necesita de inmediato.

Por qué es importante: Evita la sobreingeniería y hace que el código sea más manejable.

Conceptos de Software y Desarrollo

IDE (Integrated Development Environment)

Qué es: Herramienta que proporciona un entorno completo para escribir, compilar, y depurar código.

Por qué es importante: Facilita el desarrollo y mejora la productividad.

Versionamiento (Version Control)

Qué es: Sistema que permite gestionar las diferentes versiones del código fuente a lo largo del tiempo.

Por qué es importante: Permite mantener un historial del código y colaborar con otros programadores de manera efectiva.

Git

Qué es: Sistema de control de versiones distribuido que te permite gestionar y rastrear cambios en tu código.

Por qué es importante: Es una herramienta esencial para el trabajo en equipo y la gestión de código.

Repositorio (Repository)

Qué es: Ubicación donde se almacenan y gestionan los archivos de código fuente en un sistema de control de versiones.

Por qué es importante: Facilita el trabajo colaborativo y la organización del código.

API (Application Programming Interface)

Qué es: Conjunto de reglas y protocolos que permite que diferentes programas se comuniquen entre sí.

Por qué es importante: Facilita la integración y reutilización de servicios de software.

Framework

Qué es: Conjunto de herramientas y bibliotecas que facilita el desarrollo de aplicaciones.

Por qué es importante: Proporciona una estructura para desarrollar aplicaciones más rápido y de manera más organizada.

Biblioteca (Library)

Qué es: Colección de funciones y clases que pueden ser reutilizadas por diferentes programas.

Por qué es importante: Ayuda a no reinventar la rueda, ya que se pueden reutilizar soluciones ya desarrolladas.

Desarrollo Web y Front-End

HTML (Hypertext Markup Language)

Qué es: Lenguaje de marcado utilizado para estructurar contenido en la web.

Por qué es importante: Es la base de todas las páginas web.

CSS (Cascading Style Sheets)

Qué es: Lenguaje utilizado para definir el estilo visual de una página web (colores, fuentes, disposición, etc.).

Por qué es importante: Controla la apariencia de las páginas web y mejora la experiencia del usuario.

JavaScript

Qué es: Lenguaje de programación que se utiliza principalmente para agregar interactividad a las páginas web.

Por qué es importante: Esencial para el desarrollo de aplicaciones web dinámicas.

Bases de Datos

Base de Datos Relacional (RDBMS)

Qué es: Sistema de almacenamiento de datos organizado en tablas relacionadas entre sí.

Por qué es importante: Es la forma más común de almacenar y gestionar datos estructurados.

SQL (Structured Query Language)

Qué es: Lenguaje utilizado para interactuar con bases de datos relacionales.

Por qué es importante: Permite crear, leer, actualizar y borrar datos en una base de datos.

Metodologías y Gestión de Proyectos

Agile

Qué es: Metodología de desarrollo de software basada en iteraciones rápidas y la adaptación continua.

Por qué es importante: Promueve la flexibilidad, la colaboración y el enfoque en entregar valor al usuario rápidamente.

Scrum

Qué es: Marco de trabajo ágil para la gestión de proyectos basado en iteraciones llamadas "sprints".

Por qué es importante: Ayuda a gestionar equipos de desarrollo y proporciona una estructura para completar proyectos de software.

Conceptos de Seguridad

Cifrado (Encryption)

Qué es: Proceso de convertir datos en un formato seguro para que no puedan ser leídos sin una clave.

Por qué es importante: Protege los datos sensibles y asegura la privacidad.

Autenticación

Qué es: Proceso de verificar la identidad de un usuario o sistema.

Por qué es importante: Garantiza que solo los usuarios autorizados tengan acceso a los recursos.

Autorización

Qué es: Proceso de definir qué recursos o acciones un usuario puede realizar después de ser autenticado.

Por qué es importante: Controla el acceso a los sistemas y datos.

1. Instalar git y crear cuenta github
2. Crear repositorio de github y añadir a la carpeta del entorno de trabajo del proyecto.
3. Comandos importantes para el uso de git

git config --global user.name "Tu Nombre"

git config --global user.email [tu\_email@example.com](mailto:tu_email@example.com)

git init

git commit -m "Primer commit"

git branch -M master

git remote add origin <https://github.com/siulserrotam/Manantial.git>

git pull origin master

git push -u origin master

git branch develop

git checkout develop

NOTA 1

UML (Unified Modeling Language): Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar sistemas de software.

MVC (Model-View-Controller): Es un patrón de diseño de software que separa una aplicación en tres componentes: el modelo (lógica de datos), la vista (interfaz de usuario) y el controlador (gestión de interacciones).

WCF (Windows Communication Foundation): Es una plataforma de comunicación de Microsoft que permite crear aplicaciones distribuidas y servicios web.

WPF (Windows Presentation Foundation): Es una tecnología de Microsoft para desarrollar aplicaciones de escritorio con interfaces gráficas avanzadas en Windows.

Web API (SOAP y REST):

SOAP (Simple Object Access Protocol): Es un protocolo estándar basado en XML para la comunicación entre aplicaciones a través de la web. SOAP es más formal y estructurado, utilizado generalmente en servicios empresariales.

REST (Representational State Transfer): Es un estilo arquitectónico para servicios web que utiliza HTTP y permite la comunicación entre aplicaciones de forma más sencilla, generalmente utilizando los métodos GET, POST, PUT y DELETE. REST es más flexible y se usa ampliamente en aplicaciones modernas.

ASP.NET: Es un marco de desarrollo de aplicaciones web de Microsoft. Permite construir sitios web dinámicos y aplicaciones web mediante tecnologías como HTML, CSS, JavaScript y C#. Es parte de la plataforma .NET.

ADO.NET: Es una parte de .NET que permite acceder y manipular datos de bases de datos. ADO.NET proporciona una forma de conectar, consultar y actualizar bases de datos mediante objetos como Connection, Command, DataReader y DataAdapter.

VB.NET: Es un lenguaje de programación basado en .NET. Se utiliza principalmente para crear aplicaciones de escritorio y aplicaciones web en la plataforma .NET. Se deriva de Visual Basic, pero con características más modernas.

SOLID es un conjunto de principios de diseño de software que ayudan a crear sistemas más flexibles, fáciles de mantener y escalables. Los principios son:

S: Single Responsibility Principle (Principio de Responsabilidad Única)

O: Open/Closed Principle (Principio de Abierto/Cerrado)

L: Liskov Substitution Principle (Principio de Sustitución de Liskov)

I: Interface Segregation Principle (Principio de Segregación de Interfaces)

D: Dependency Inversion Principle (Principio de Inversión de Dependencias)

SOLID ayuda a que el código sea más fácil de entender, probar y extender, reduciendo errores a largo plazo.

LINQ (Language Integrated Query) es una característica de C# y otros lenguajes .NET que permite realizar consultas a colecciones de datos (como listas, bases de datos, etc.) de manera sencilla y coherente, utilizando sintaxis de lenguaje integrado.

LINQ facilita trabajar con datos de manera más eficiente, reduciendo la necesidad de escribir código complejo para filtrar, ordenar y manipular colecciones.

Quiero una webapi que me estructures mi proyecto que llamaremos "Manantial" el cual quiero un entorno en asp.net 9.0 capa administración y otra de cliente también quiero la estructura de una api restful de angular con Bootstrap integrado css3, quiero que te bases con las siguiente base de datos y sus respectivas tablas, no quiero explicacion solo las estructuras: (modelo de capas)

ESTRUCTURA DE LA BASE DATOS EN SQL SERVER

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 2. Términos de Capas y Arquitectura de Software

API

Qué es: Capa que expone las funcionalidades de un sistema a través de una interfaz de programación de aplicaciones (API).

Para qué es: Permite que diferentes sistemas se comuniquen entre sí.

Por qué es importante: Aísla la lógica de negocio y permite la interoperabilidad.

Core

Qué es: Capa central que contiene la lógica de negocio esencial.

Para qué es: Alberga la funcionalidad central del sistema.

Por qué es importante: Es la base sobre la que se construye la aplicación.

Infraestructura

Qué es: Capa que se encarga de los servicios externos como bases de datos, redes, o almacenamiento.

Para qué es: Provee soporte para que el sistema funcione correctamente.

Por qué es importante: Maneja las interacciones con los recursos externos.

Application

Qué es: Capa que gestiona la lógica de aplicación, la interacción con el usuario y las interfaces.

Para qué es: Controla la ejecución del flujo de la aplicación.

Por qué es importante: Facilita la interacción entre la lógica de negocio y la interfaz de usuario.

Admin MVC

Qué es: Arquitectura basada en Model-View-Controller para el desarrollo de aplicaciones web, centrada en la gestión administrativa.

Para qué es: Separa la lógica de la aplicación, la vista y el control de flujo.

Por qué es importante: Organiza el código y facilita el mantenimiento y la escalabilidad.

ESTRUCTURA EN CAPAS USANDO CHATGPT

Manantial

│

├── Api

│ ├── Controllers

│ │ ├── CategoriaController.cs

│ │ ├── MarcaController.cs

│ │ ├── ProductoController.cs

│ │ ├── ClienteController.cs

│ │ ├── CarritoController.cs

│ │ ├── VentaController.cs

│ │ ├── DetalleVentaController.cs

│ │ ├── UsuarioController.cs

│ │ ├── DepartamentoController.cs

│ │ ├── CiudadController.cs

│ │ └── BarrioController.cs

│ ├── Models

│ │ ├── Categoria.cs

│ │ ├── Marca.cs

│ │ ├── Producto.cs

│ │ ├── Cliente.cs

│ │ ├── Carrito.cs

│ │ ├── Venta.cs

│ │ ├── DetalleVenta.cs

│ │ ├── Usuario.cs

│ │ ├── Departamento.cs

│ │ ├── Ciudad.cs

│ │ └── Barrio.cs

│ ├── Dtos

│ │ ├── CategoriaDto.cs

│ │ ├── MarcaDto.cs

│ │ ├── ProductoDto.cs

│ │ ├── ClienteDto.cs

│ │ ├── CarritoDto.cs

│ │ ├── VentaDto.cs

│ │ ├── DetalleVentaDto.cs

│ │ ├── UsuarioDto.cs

│ │ ├── DepartamentoDto.cs

│ │ ├── CiudadDto.cs

│ │ └── BarrioDto.cs

│ ├── Repositories

│ │ ├── ICategoriaRepository.cs

│ │ ├── IMarcaRepository.cs

│ │ ├── IProductoRepository.cs

│ │ ├── IClienteRepository.cs

│ │ ├── ICarritoRepository.cs

│ │ ├── IVentaRepository.cs

│ │ ├── IDetalleVentaRepository.cs

│ │ ├── IUsuarioRepository.cs

│ │ ├── IDepartamentoRepository.cs

│ │ ├── ICiudadRepository.cs

│ │ └── IBarrioRepository.cs

│ ├── Services

│ │ ├── CategoriaService.cs

│ │ ├── MarcaService.cs

│ │ ├── ProductoService.cs

│ │ ├── ClienteService.cs

│ │ ├── CarritoService.cs

│ │ ├── VentaService.cs

│ │ ├── DetalleVentaService.cs

│ │ ├── UsuarioService.cs

│ │ ├── DepartamentoService.cs

│ │ ├── CiudadService.cs

│ │ └── BarrioService.cs

│ ├── AppSettings

│ │ └── appsettings.json

│ ├── Startup.cs

│ └── Program.cs

│

├── Admin

│ ├── Controllers

│ │ ├── CategoriaController.cs

│ │ ├── MarcaController.cs

│ │ ├── ProductoController.cs

│ │ ├── ClienteController.cs

│ │ ├── CarritoController.cs

│ │ ├── VentaController.cs

│ │ ├── DetalleVentaController.cs

│ │ ├── UsuarioController.cs

│ │ ├── DepartamentoController.cs

│ │ ├── CiudadController.cs

│ │ └── BarrioController.cs

│ ├── Views

│ │ ├── Categoria

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Marca

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Producto

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Cliente

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Carrito

│ │ │ ├── Index.cshtml

│ │ ├── Venta

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── DetalleVenta

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Usuario

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Departamento

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Ciudad

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ │ ├── Barrio

│ │ │ ├── Index.cshtml

│ │ │ ├── Create.cshtml

│ │ │ └── Edit.cshtml

│ ├── Layout.cshtml

│ ├── \_ViewImports.cshtml

│ └── \_ViewStart.cshtml

│

├── Client

│ ├── src

│ │ ├── app

│ │ │ ├── components

│ │ │ │ ├── categoria

│ │ │ │ ├── marca

│ │ │ │ ├── producto

│ │ │ │ ├── cliente

│ │ │ │ ├── carrito

│ │ │ │ ├── venta

│ │ │ │ ├── detalle-venta

│ │ │ │ ├── usuario

│ │ │ │ ├── departamento

│ │ │ │ ├── ciudad

│ │ │ │ └── barrio

│ │ │ ├── services

│ │ │ │ ├── categoria.service.ts

│ │ │ │ ├── marca.service.ts

│ │ │ │ ├── producto.service.ts

│ │ │ │ ├── cliente.service.ts

│ │ │ │ ├── carrito.service.ts

│ │ │ │ ├── venta.service.ts

│ │ │ │ ├── detalle-venta.service.ts

│ │ │ │ ├── usuario.service.ts

│ │ │ │ ├── departamento.service.ts

│ │ │ │ ├── ciudad.service.ts

│ │ │ │ └── barrio.service.ts

│ │ │ ├── models

│ │ │ │ ├── categoria.model.ts

│ │ │ │ ├── marca.model.ts

│ │ │ │ ├── producto.model.ts

│ │ │ │ ├── cliente.model.ts

│ │ │ │ ├── carrito.model.ts

│ │ │ │ ├── venta.model.ts

│ │ │ │ ├── detalle-venta.model.ts

│ │ │ │ ├── usuario.model.ts

│ │ │ │ ├── departamento.model.ts

│ │ │ │ ├── ciudad.model.ts

│ │ │ │ └── barrio.model.ts

│ │ │ └── app.module.ts

│ └── package.json

│

├── Core

│ ├── Entities

│ │ ├── BaseEntity.cs

│ │ ├── Categoria.cs

│ │ ├── Marca.cs

│ │ ├── Producto.cs

│ │ ├── Cliente.cs

│ │ ├── Carrito.cs

│ │ ├── Venta.cs

│ │ ├── DetalleVenta.cs

│ │ ├── Usuario.cs

│ │ ├── Departamento.cs

│ │ ├── Ciudad.cs

│ │ └── Barrio.cs

│ ├── Interfaces

│ | ├── ISpecification.cs

│ │ ├── IGenericRepository.cs

│ │ ├── ICategoriaRepository.cs

│ │ ├── IMarcaRepository.cs

│ │ ├── IProductoRepository.cs

│ │ ├── IClienteRepository.cs

│ │ ├── ICarritoRepository.cs

│ │ ├── IVentaRepository.cs

│ │ ├── IDetalleVentaRepository.cs

│ │ ├── IUsuarioRepository.cs

│ │ ├── IDepartamentoRepository.cs

│ │ ├── ICiudadRepository.cs

│ │ └── IBarrioRepository.cs

└── Specifications

│ ├── BaseSpecification.cs

│ ├── BarrioSpecification.cs

│ ├── CategoriaSpecification.cs

│ ├── CiudadSpecification.cs

│ ├── ClienteSpecification.cs

│ ├── DepartamentoSpecification.cs

│ ├── DetalleVentaSpecification.cs

│ ├── MarcaSpecification.cs

│ ├── ProductoSpecification.cs

│ ├──UsuarioSpecification.cs

│ ├── VentaSpecification.cs

│ └── Services

│ ├── CategoriaService.cs

│ ├── MarcaService.cs

│ ├── ProductoService.cs

│ ├── ClienteService.cs

│ ├── CarritoService.cs

│ ├── VentaService.cs

│ ├── DetalleVentaService.cs

│ ├── UsuarioService.cs

│ ├── DepartamentoService.cs

│ ├── CiudadService.cs

│ └── BarrioService.cs

│

├── Application

│ ├── Dtos

│ │ ├── CategoriaDto.cs

│ │ ├── MarcaDto.cs

│ │ ├── ProductoDto.cs

│ │ ├── ClienteDto.cs

│ │ ├── CarritoDto.cs

│ │ ├── VentaDto.cs

│ │ ├── DetalleVentaDto.cs

│ │ ├── UsuarioDto.cs

│ │ ├── DepartamentoDto.cs

│ │ ├── CiudadDto.cs

│ │ └── BarrioDto.cs

│ ├── Services

│ │ ├── CategoriaService.cs

│ │ ├── MarcaService.cs

│ │ ├── ProductoService.cs

│ │ ├── ClienteService.cs

│ │ ├── CarritoService.cs

│ │ ├── VentaService.cs

│ │ ├── DetalleVentaService.cs

│ │ ├── UsuarioService.cs

│ │ ├── DepartamentoService.cs

│ │ ├── CiudadService.cs

│ │ └── BarrioService.cs

│ └── Mappers

│ ├── CategoriaMapper.cs

│ ├── MarcaMapper.cs

│ ├── ProductoMapper.cs

│ ├── ClienteMapper.cs

│ ├── CarritoMapper.cs

│ ├── VentaMapper.cs

│ ├── DetalleVentaMapper.cs

│ ├── UsuarioMapper.cs

│ ├── DepartamentoMapper.cs

│ ├── CiudadMapper.cs

│ └── BarrioMapper.cs

│

├── Infrastructure

│ ├── Data

│ │ ├── ApplicationDbContext.cs

│ │ └── Migration

│ │ ├── 20250320010100\_InitialMigration.cs

│ │ └── ApplicationDbContextModelSnapshot.cs

│ └── Repositories

│ ├── CategoriaRepository.cs

│ ├── MarcaRepository.cs

│ ├── ProductoRepository.cs

│ ├── ClienteRepository.cs

│ ├── CarritoRepository.cs

│ ├── VentaRepository.cs

│ ├── DetalleVentaRepository.cs

│ ├── UsuarioRepository.cs

│ ├── DepartamentoRepository.cs

│ ├── CiudadRepository.cs

│ └── BarrioRepository.cs

NOTA 3.

Fundamentos de OOP (Programación Orientada a Objetos)

OOP (Object-Oriented Programming)

Qué es: Paradigma de programación basado en objetos que contienen datos y métodos.

Para qué es: Organiza el código en unidades reutilizables, facilitando la escalabilidad y el mantenimiento.

Por qué es importante: Permite manejar proyectos grandes y complejos de manera más estructurada y modular.

Clase

Qué es: Plantilla para crear objetos, define propiedades y métodos.

Para qué es: Estructura base de los objetos, donde se definen sus comportamientos y características.

Por qué es importante: Define la estructura de los objetos de una manera organizada y reutilizable.

Método

Qué es: Función dentro de una clase que define el comportamiento de los objetos.

Para qué es: Realiza operaciones o acciones sobre los datos de la clase.

Por qué es importante: Permite que los objetos realicen acciones o procesos específicos.

Propiedad

Qué es: Variable asociada a una clase que guarda datos específicos de un objeto.

Para qué es: Almacena información del estado del objeto.

Por qué es importante: Define las características o atributos de un objeto.

Herencia

Qué es: Mecanismo que permite que una clase herede propiedades y métodos de otra.

Para qué es: Promueve la reutilización del código.

Por qué es importante: Facilita la creación de nuevas clases sin tener que reescribir código.

Polimorfismo

Qué es: Capacidad de un objeto para tomar diferentes formas, generalmente en métodos.

Para qué es: Permite que un mismo método se comporte de manera distinta según el objeto que lo invoque.

Por qué es importante: Ofrece flexibilidad y extensibilidad en el código.

BACKEND

Terminal

1. Instalar dotnet-sdk 8.0 y crear en la carpeta del proyecto desde la terminal
2. crear la solución:

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 carpeta 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 api

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.Tools

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 (opcional para mapeo de DTOs)

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

1. Instalar Bootstrap para el cliente (Client)

Cd 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. En la carpeta Infraestructure: se crea carpeta data y ApplicationDbContext.cs

using Microsoft.EntityFrameworkCore;

namespace Infraestructure.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

}

}

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. se modifica el archivo program.cs para la conexión de la base de datos
2. Crear en la carpeta principal el archivo .gitignore

obj

bin

appsettings.json

\*.db

NOTA 4.

Conceptos de Diseño y Arquitectura

Entidad

Qué es: Objeto que representa un modelo de datos en una base de datos.

Para qué es: Maneja y manipula los datos dentro de un sistema.

Por qué es importante: Es esencial para la persistencia y organización de la información.

Abstract

Qué es: Modificador que se usa para declarar clases o métodos que no pueden ser instanciados directamente.

Para qué es: Define un comportamiento común sin implementación concreta.

Por qué es importante: Permite definir una interfaz común para clases derivadas.

Class (Clase)

Qué es: Definición o plantilla para crear objetos.

Para qué es: Permite organizar el código en objetos con propiedades y métodos.

Por qué es importante: Facilita la organización y reutilización del código.

CRUD

Qué es: Conjunto de operaciones básicas de base de datos: Crear, Leer, Actualizar y Borrar.

Para qué es: Permite realizar las acciones fundamentales de gestión de datos.

Por qué es importante: Es esencial para interactuar con bases de datos de manera estructurada.

Interfaces

Qué es: Contrato que define los métodos que una clase debe implementar sin proporcionar detalles de implementación.

Para qué es: Permite la interoperabilidad entre diferentes clases.

Por qué es importante: Promueve la flexibilidad y el desacoplamiento del código.

Inyección de Dependencias

Qué es: Patrón de diseño que permite pasar las dependencias necesarias a un objeto en lugar de que el objeto las cree.

Para qué es: Facilita el testing y la mantenibilidad del código.

Por qué es importante: Mejora la modularidad y reduce el acoplamiento entre clases.

Services (Servicios)

Qué es: Clases o componentes que realizan operaciones o procesos específicos en el sistema.

Para qué es: Encapsula la lógica de negocio.

Por qué es importante: Aísla la lógica de la aplicación, lo que facilita su mantenimiento.

1. Añadir las entidades a la carpeta Entites que se debe crear en Core

NOTA 5.

Conceptos de Programación Asíncrona y Funcional

Async

Qué es: Palabra clave que permite la ejecución de operaciones asincrónicas sin bloquear el hilo principal.

Para qué es: Mejora el rendimiento de la aplicación al no bloquear la ejecución mientras se espera una tarea.

Por qué es importante: Permite aplicaciones más rápidas y eficientes.

Await

Qué es: Utilizada con async para esperar la finalización de una operación asincrónica.

Para qué es: Hace que una función asíncrona espere el resultado de una tarea sin bloquear el hilo.

Por qué es importante: Mejora la legibilidad del código asíncrono.

Task

Qué es: Representa una operación asincrónica que se ejecuta en un hilo separado.

Para qué es: Maneja operaciones que pueden ejecutarse de manera independiente.

Por qué es importante: Permite una ejecución más eficiente de operaciones que no requieren bloquear el flujo principal.

Void

Qué es: Tipo de retorno de un método que no devuelve valor.

Para qué es: Se usa cuando una función realiza una acción sin necesidad de retornar algo.

Por qué es importante: Indica que el método no tiene un valor de salida.

¿Qué incluye una interfaz de repositorio?

Operaciones básicas de CRUD (Crear, Leer, Actualizar, Eliminar).

Métodos para obtener entidades según condiciones específicas.

Métodos para interactuar con la base de datos de forma abstracta.

¿Qué incluye una interfaz de servicio?

Métodos que implementan casos de uso o operaciones de negocio.

Métodos que realizan operaciones complejas combinando entidades o incluso llamando a múltiples repositorios.

Transformación de datos (por ejemplo, de DTO a entidades y viceversa).

los DTOs son utilizados para evitar la exposición directa de las entidades de la base de datos y facilitar el trabajo con datos de una manera más estructurada.

| **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. Crear las interfaces IGenericRepository y demás en la carpeta Core/Interfaces
2. Crear los servicios de las interfaces (CRUD)
3. Crear interfaces y servicios en la Carpeta Application/Interfaces o Services
4. Crear las clases especificasen la carpeta Core/Specifications
5. Modificar Programa para añadir el servicio SwaggerGen
6. Crear carpeta Controller y archivos en la capa Api
7. Crear el archivo ContextoTienda en la capa Infraestructure
8. Crear RepositorioGenerico.cs en la capa infra

| **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. |

1. Es necesario crear la carpeta Infra/Data/Seed(semilla)
2. Crear archivos .Json de los datos semilla ingresados
3. Crear el Archivo SemillaContextoTienda en la capa Infra
4. Añadir el EvaluadorDeEspecifaciones en la capa Infra

El archivo **EvaluadorDeEspecificaciones** se utiliza para aplicar **especificaciones** en las consultas a la base de datos. En términos simples, las especificaciones permiten crear filtros complejos para obtener solo los datos que cumplan con ciertos criterios o condiciones.

1. Crear carpeta Repositories y añadir el archivo RepositorioProducto
2. Incluir a la Spacifacation el archivo EspecificacionProductosConCategoriaYMarca.cs, borrar especificaciones no requeridas al momento.

Nota. Controller, router, readonly, Find, ToString, throw, AsNoTracking, constructor, propiedad, Helpers, Middleware, Extensions, Models, destMember, Empty, source, Find

1. Actualizar datos, corregir si es necesario el ContextoTienda.cs
2. Implementar llaves Primarias y foráneas si se requiere corrección en el ContextoTienda
3. Middleware crear archivos en la capa Api

El Middleware se añade en la capa API, ya que es donde se manejan las solicitudes HTTP antes de llegar a los controladores. Su propósito es interceptar y procesar peticiones y respuestas, permitiendo funcionalidades como autenticación, manejo de errores, logging, compresión, CORS, entre otros.

1. Model crear en la capa Application (Opcional – sin archivos)

Los Models definen las entidades o estructuras de datos que usa la aplicación, generalmente reflejando tablas de la base de datos o estructuras utilizadas en la lógica de negocio. Se utilizan para la transferencia de datos entre capas y garantizar consistencia en la aplicación.

Siempre usa Entities en Core/Entities/ para mapear la base de datos.

Siempre usa DTOs en Application/DTOs/ para transferir datos en la API.

Opcionalmente usa Models en Application/Models/ si necesitas estructurar datos antes de enviarlos.

Usa AutoMapper para convertir Entities a DTOs fácilmente.

|  |  |  |
| --- | --- | --- |
| 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) |

1. Helpers crear en la Capa Application

Los Helpers son clases estáticas que contienen métodos reutilizables para tareas comunes, como formateo de fechas, generación de tokens, cifrado de contraseñas, validaciones personalizadas, etc.

dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection

1. Crear archivo PerfilesDeMapeo.cs y ProductoUrlResolver.cs
2. Carpeta Extensions en la Capa Api

Extensions (clase base para otras extensiones)

Contiene métodos de extensión generales para mejorar la reutilización del código.

Ejemplo: Métodos de configuración de CORS, JWT, middleware globales.

1. Crear las extensiones ApplicationServicesExtensions y SwaggerServicesExtensions

ApplicationServicesExtensions

Registra los servicios de aplicación en el contenedor de dependencias de ASP.NET Core.

Se usa en Program.cs para configurar AutoMapper, repositorios, servicios de aplicación, etc.

Ejemplo: Agregar AutoMapper, Unit of Work, etc.

SwaggerServicesExtensions

Configura Swagger/OpenAPI para documentar la API.

Define rutas y agrega autenticación a la documentación.

Ejemplo: Habilitar SwaggerUI y AddSwaggerGen().

1. Carpeta Errors en la Capa Api

La carpeta Errors se usa para manejar los errores de la aplicación de manera estructurada y proporcionar respuestas claras a los clientes de la API.

Centraliza el manejo de excepciones en la API.

Personaliza respuestas de error con códigos HTTP adecuados.

Facilita la depuración y el mantenimiento con mensajes detallados.

1. Crear ApiResponse.cs, ApiException.cs, ExceptionMiddleware.cs y RespuestaErrorValidacionApi
2. Carpeta Controllers en la Capa Api

Los Controllers en ASP.NET Core manejan las solicitudes HTTP y responden con datos en JSON o vistas. Son el punto de entrada de la API.

Reciben peticiones del cliente (Angular, Postman, etc.).

Llaman a los servicios de aplicación para procesar la lógica de negocio.

Devuelven respuestas en JSON con códigos HTTP adecuados.

1. Crear los siguientes controladores

BaseApiController → Proporciona operaciones CRUD genéricas para cualquier entidad.

ErrorController.cs → Maneja errores globales y devuelve respuestas personalizadas.

BuggyController → Simula diferentes errores HTTP (400, 404, 500) para pruebas y validación.

ProductoController.cs → Manejo de productos.

ClienteController.cs → Manejo de clientes.

VentaController.cs → Procesar ventas.

CarritoController.cs → Operaciones del carrito de compras.

UsuarioController.cs → Autenticación y usuarios.

1. Incorporado todas las clases en el archivo Program.cs
2. Archivo ConfiguraciónProducto en la carpeta Data/Config
3. wwwroot/images/Productos en la carpeta Application
4. 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

1. Cd client se inicializa

ng serve

Would you like to share pseudonymous usage data about this project with the Angular Team

at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more

details and how to change this setting, see <https://angular.dev/cli/analytics>.

N

Resumen de la secuencia recomendada

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.).

Nota. Es obligatorio agregar todos los métodos creados en una herencia.

JERARQUIA

**Interfaces (IRepositorioGenerico, IRepositorioProducto, IEspecificacion)**

* Definen contratos para repositorios y especificaciones, asegurando consistencia en la implementación.

**Clases base (EntidadBase, EspecificacionBase, ControladorBaseApi)**

* EntidadBase: Propiedades comunes a todas las entidades (ID, fechas, etc.).
* EspecificacionBase: Base para crear filtros o consultas avanzadas.
* ControllerBase y ControladorBaseApi: Base para controladores, centralizando configuraciones y métodos comunes.

**Entidades (EntidadCategoria, EntidadMarca, EntidadProducto)**

* Representan los objetos de la base de datos con sus propiedades y relaciones.

**DTOs (DtoProducto)**

* Objetos de transferencia de datos que estructuran la información para la API sin exponer entidades directamente.

**Implementaciones de especificaciones (EspecificacionProductosConCategoriaYMarca)**

* ContextoTienda.cs y EvaluadorDeEspecificaciones.cs (Data)
* Filtran o estructuran consultas avanzadas para repositorios, como buscar productos con categoría y marca.

**Repositorios concretos (RepositorioGenerico, RepositorioProducto)**

* RepositorioGenerico: Implementa operaciones CRUD comunes.
* RepositorioProducto: Métodos específicos para productos, extendiendo RepositorioGenerico.
* RespuestaApi.cs (API.Errors)

**Controladores (ControladorProducto)**

* Manejan las solicitudes HTTP y comunican la API con la lógica de negocio.
* Crear todos los Repositorios de acuerdo con el Generico