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.

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)

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), --distrito /barrio /vereda

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

--provincia / ciudad/ municipio

create table CIUDAD(

IdCiudad varchar(4) NOT NULL,

Descripcion varchar (45) NOT NULL,

Fk\_IdDepartamento varchar (2) NOT NULL

)

go

--distrito / barrio/ vereda

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

ESTRUCTURA EN CAPAS USANDO CHATGPT

**Estructura del Proyecto ASP.NET Core 9.0**

1. **Api** (API RESTful)
   * **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**
     + CategoriaRepository.cs
     + MarcaRepository.cs
     + ProductoRepository.cs
     + ClienteRepository.cs
     + CarritoRepository.cs
     + VentaRepository.cs
     + DetalleVentaRepository.cs
     + UsuarioRepository.cs
     + DepartamentoRepository.cs
     + CiudadRepository.cs
     + BarrioRepository.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
     + appsettings.Development.json
   * **Startup.cs**
   * **Program.cs**
2. **Admin** (Capa Administrativa)
   * **Controllers**
     + CategoriaAdminController.cs
     + MarcaAdminController.cs
     + ProductoAdminController.cs
     + ClienteAdminController.cs
     + CarritoAdminController.cs
     + VentaAdminController.cs
     + DetalleVentaAdminController.cs
     + UsuarioAdminController.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
       - Create.cshtml
       - Edit.cshtml
     + **Venta**
       - Index.cshtml
       - Create.cshtml
       - Edit.cshtml
     + **DetalleVenta**
       - Index.cshtml
       - Create.cshtml
       - Edit.cshtml
     + **Usuario**
       - Index.cshtml
       - Create.cshtml
       - Edit.cshtml
   * **Layout.cshtml**
   * **\_ViewImports.cshtml**
   * **\_ViewStart.cshtml**
3. **Client** (Capa de Cliente Angular)
   * **src/app**
     + **components**
       - categoria
         * categoria.component.ts
         * categoria.component.html
         * categoria.component.css
       - marca
         * marca.component.ts
         * marca.component.html
         * marca.component.css
       - producto
         * producto.component.ts
         * producto.component.html
         * producto.component.css
       - carrito
         * carrito.component.ts
         * carrito.component.html
         * carrito.component.css
       - venta
         * venta.component.ts
         * venta.component.html
         * venta.component.css
       - detalle-venta
         * detalle-venta.component.ts
         * detalle-venta.component.html
         * detalle-venta.component.css
     + **services**
       - categoria.service.ts
       - marca.service.ts
       - producto.service.ts
       - carrito.service.ts
       - venta.service.ts
       - detalle-venta.service.ts
     + **models**
       - categoria.model.ts
       - marca.model.ts
       - producto.model.ts
       - carrito.model.ts
       - venta.model.ts
       - detalle-venta.model.ts
     + **app.module.ts**
     + **app.component.ts**
     + **app.component.html**
     + **app.component.css**
   * **assets**
     + **images**
     + **styles**
       - styles.css
     + **bootstrap**
       - (Incluir Bootstrap 5 o versión deseada)
   * **index.html**
   * **angular.json**
   * **package.json**

public class ApplicationDbContext : DbContext

{

public DbSet<Categoria> Categorias { get; set; }

public DbSet<Marca> Marcas { get; set; }

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

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

public DbSet<Carrito> Carritos { get; set; }

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

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

public DbSet<Usuario> Usuarios { get; set; }

public DbSet<Departamento> Departamentos { get; set; }

public DbSet<Ciudad> Ciudades { get; set; }

public DbSet<Barrio> Barrios { get; set; }

}

BACKEND

Terminal

1. 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. Agregar los proyectos a la solución

dotnet sln add Api/Api.csproj

dotnet sln add Admin/Admin.csproj

dotnet sln add Client/Client.angular.json

1. Agregar nueva Capa services

dotnet new classlib -n Manantial.Services

1. Agregar la capa Repositories

dotnet new classlib -n Repositories

1. Agregando a la solución las capas

dotnet sln add Services/Services.csproj

dotnet sln add Repositories/Repositories.csproj

1. Referenciar Services y Repositories desde Api

dotnet add Api/Api.csproj reference Services/Services.csproj

dotnet add Api/Api.csproj reference Repositories/Repositories.csproj

1. Referenciar Services y Repositories desde Admin

dotnet add Admin/Admin.csproj reference Services/Services.csproj

dotnet add Admin/Admin.csproj reference Repositories/Repositories.csproj

1. Instalar dependencias adicionales

dotnet add Api/Api.csproj package Microsoft.EntityFrameworkCore.SqlServer

dotnet add Api/Api.csproj package Microsoft.EntityFrameworkCore.Design

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

Faltan las capas

CORE, INFRAESTRUCTURE Y APPLICATION

1. se crea carpeta data y ApplicationDbContext.cs
2. se modica 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. se genera la primera migración

dotnet ef migrations add InitialCreate --project Api/Api.csproj

1. Actualizar datos

dotnet ef database update --project Api/Api.csproj

1. Se corre la api

dotnet 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

Instalar dotnet-sdk 8.0 y crear en la carpeta del proyecto desde la terminal

dotnet new webapi -n API

1. En la carpeta Api instalar

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Microsoft.EntityFrameworkCore.Design

1. En la carpeta principal agregar las librerías y eliminar las class por defecto.

Dotnet new classlib -o Core

Dotnet new classlib -o Infraestructure

Dotnet new classlib -o Application

Dotnet sln add Core

Dotnet sln add Infraestructure

Dotnet sln add Application

1. Agregamos las Referencias de las Capas

# Agregar referencias de Api

Cd Api

dotnet add reference ../Application/Application.csproj

dotnet add reference ../Core/Core.csproj

dotnet add reference ../Infraestructure/Infraestructure.csproj

# Agregar referencia de Application a Core

cd ../Application

dotnet add reference ../Core/Core.csproj

# Agregar referencias de Infraestructure a Core y Application

cd ../ Infraestructure

dotnet add reference ../Core/Core.csproj

dotnet add reference ../Application/Application.csproj

# Restaurar dependencias

dotnet restore

1. Crear la carpeta DATA agregar referencia

dotnet new classlib -n Data

dotnet add reference ../ Data/Data.csproj En la carpeta Api

1. Crear Archivo ApplicationDbContext Eliminar la Class por defecto

using Microsoft.EntityFrameworkCore;

namespace ManantialAPI.Data

{

    public class ApplicationDbContext : DbContext

    {

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

            : base(options) { }

        public DbSet<Categoria> Categorias { get; set; }

        public DbSet<Marca> Marcas { get; set; }

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

        // Aquí defines las demás tablas (CLIENTE, CARRITO, etc.)

    }

}

1. Modificar el archivo appsettings.json de la carpeta API

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  },

  "ConnectionStrings": {

    "cadena": "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

1. Modificar el archivo Program.cs de la carpeta Api

* Crear el proyecto de la API usando dotnet new webapi.
* Configurar la conexión a la base de datos en appsettings.json y Program.cs.
* Crear los modelos de entidad que representarán las tablas de la base de datos.
* Crear los controladores de la API para manejar las solicitudes HTTP.
* Configurar Swagger para probar la API de manera interactiva.