***PROCESO DE CREACIÓN DE UNA WEB API***

..

1. Crea una nueva solución en blanco para partir de cero en la construcción del proyecto.
2. Luego agrega el nuevo proyecto a crear en este caso un web api con asp.net core
3. También se puede agregar un proyecto tipo biblioteca, por ejemplo, en el proyecto sistema de venta se creo una biblioteca **DAL** para todo el manejo de datos
4. Capa de negocios (biblioteca **BLL**)
5. Capa de modelos (biblioteca **MODEL**)
6. Capa de **IOC** (inversión de control), nos permite separa la inyección de dependencias de nuestro proyecto.
7. Capa de **DTO** (), nos permite crear clases similares a nuestros modelos ya que vamos a compartir la información a la aplicación
8. Capa de **UTILITY,** se crean todas las utilidades que vamos a utilizar en nuestro proyecto.
9. Si se maneja la capa de modelos es importante tener en cuenta que se debe poner en proyecto de inicio esta capa para poder instalar los paquetes que esta va a utilizar.
10. En appsettings agregar la conexión de la base de datos

..

1. Se crea una carpeta en api llamada utilidad y dentro se crea una clase response con el fin de que son sirva como respuesta a todas las peticiones de nuestras apis
2. Se crean los controllers necesarios para la api
3. Para activar los cors hay que dirigirse a la clase program.cs y agregar builder.Services.AddCors(options =>

{

options.AddPolicy("NuevaPolitica", app =>

{

app.AllowAnyOrigin()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

app.UseCors("NuevaPolitica");

*Paquetes de instalación*

1. EntityFrameworkCore.SqlServer (nos permite conectarnos con nuestra base de datos).
2. EntityFrameworkCore.Tools (acompañante de sqlserver)

*Comando para conectar la base de datos al web api*

1. Scaffold-DbContext "Server=(local); DataBase=dbventa; Trusted\_Connection=True; TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer (asegurar de tener en Proyecto predeterminado a la capa de model)
2. Si se aplico en anterior comando mover la clase **DBcontext** a la capa de **DAL**
3. De igual forma tener presente agregar las referencias de las capas que está utilizando (click en dependencias y click derecho sobre la capa y ya)

..

*CAPA IOC*

1. Se crea una clase en la cual se va a crear un método para inyectar las dependencias

*CAPA DAL*

1. Se crea una carpeta **Repositorios** y dentro un contrato donde se crearán las interfaces con los métodos que nos van a permitir interactuar con la base de datos.
2. Crea una clase genérica para poder implementar la interfaz con las consultas, las cuales deben estas en el mismo orden que la interfaz y en la clase se aplican los try catch

*CAPA DTO*

1. Se crean todas las clases que van a representar a nuestros modelos de la base de datos, pero que en la api que las propiedades que estén dentro de esta clase van a ser visibles en el web api.

*CAPA UTILITY*

1. Se añaden las dependencias (automapper, automapper.extensions.dependencyInjection)
2. Se crea una clase para mapear las clases con modelos
3. Luego de hacer todos los mapeos necesarios implementamos las dependencias de auto mapeo en la capa ioc que es la inyecta las dependencias a nuestro proyecto

*CAPA BLL*

1. Se crea una carpeta **Servicos** y dentro un **contrato** donde se crearán las interfaces con los métodos que nos van a permitir interactuar con la base de datos.
2. Crean clases para poder implementar la interfaz con las consultas, las cuales deben estas en el mismo orden que la interfaz y en la clase se aplican los try catch
3. implementamos las dependencias en la capa ioc que es la inyecta las dependencias a nuestro proyecto

Se puede anejar una interfaz genérica para la obtención de datos de la base de datos

La interfaz genérica se puede reutilizar para hacer diferentes consultas como obtener, consultar, eliminar, editar …, y luego se crea la clase en la cual se va a implementar la interfaz mencionada anteriormente (en esta también de debe hacer mención al contexto de nuestra base de datos)