**IMPLEMENTACIÓN DE UNA API REST FULL EN VISUAL STUDIO 2019**

**Proceso:**

1. **Creamos un proyecto de Tipo Solución en blanco**
2. **Creamos la capa API que va tener nuestro proyecto.**
   1. **Nuevo proyecto Aplicación Web ASP.NET Core C#. Tipo API**
   2. **Colocar en el nombre el identificador que es un API**
3. **Creamos la capa Core y la capa Infraestructura**
   1. **Nuevo proyecto Clase Librería .NET Standart**
4. **Creamos las carpetas DTOs, Entities, Enumerations, Exceptions, Interfaces. Para el Core.**
5. **Creamos las carpetas Data, Filters, Repositories. Para la Infraestructure.**
6. **Creamos las referencias entre las capas.**
   1. **Clic derecho en la carpeta raíz de cada capa, agregar, referencia del proyecto**
7. **Creamos una carpeta con nombre test en la raíz del proyecto.**
8. **Agregamos dos proyectos de tipo XUnit Test Project .NET Core C#**
   1. **Uno con el identificador UnitTest en el nombre**
   2. **El otro con el identificador IntegrationTest**
9. **Eliminar de las tres capas las clases que se crean por defecto**
10. **Migrar la base de datos con la ingeniería inversa que nos brinda Scaffolding de entity framework (EF).**
    1. **Gestionar los paquetes de nuguets (instalar en Infraestructura) Microsoft EntityFrameworkCore, Microsoft EntityFrameworkCore SqlServer, Microsoft EntityFrameworkCore Tools, Microsoft EntityFrameworkCore Design (instalar en API).**
    2. **Procurar que sean la misma versión del Core del proyecto para evitar problemas de compatibilidad**
    3. **Utilizar este comando en la venta de comandos de los nuget, está en herramientas Scaffold-DbContext "Server=localhost;Database=DB\_ARTICULOS;Integrated Security = false; User ID=sa; Password=Sa1" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Data**
    4. **Verificar estar en la carpeta correcta antes de ejecutar el comando. Usar la lista desplegable proyecto predeterminado de la terminal de nuget**
    5. **Inicialmente migramos todo a Infraestructure/Data**
    6. **Sacamos los modelos de las tablas a excepción del context y los pegamos en la carpeta Core/Entities**
    7. **Cambiamos el namespace de los modelos que pasamos de infraestructura a core**
    8. **Al archivo context en Data le pasamos el using de core/entities**
11. **Vamos a cambiar la forma de conexión a la base datos, lo haremos a través de pasar un parámetro desde el startup para que la conexión se implemente en el archivo context y no que de expuesto, esto es inyección de dependencias.**
    1. **Eliminamos del archivo Infraestructura/Data/context el método OnConfiguring.**
    2. **En el archivo appsetting.json creamos lo siguiente “ConnectionStrings”:{“Nombre de la conexión”:”cadena de conexión”},**
    3. **En el archivo startup en el método configureservice vamos a pasar la conexión al archivo de contexto. Service.AddDbContext<Archivo contexto>(options => options.UseSqlServer(Configuration.GetConnectionStings(“Nombre que le dimos a la conexion en el appsettings”)));**
12. **Creamos un archivo tipo clase en la carpeta Infraestructure/Repository/nombre del archivo**
13. **Creamos un archivo tipo Api control vacio en la carpeta Api/Controller/nombre del archivo**
14. **En el archivo de control que creamos vamos a crear un método publico con referencia HttpGet con el nombre que querramos de tipo IActionResult va retornar ok() código 200 en la web**
15. **Las entities ya están creadas es lo que se hizo en el paso 10**
16. **El archivo repository lo abrimos y volvemos su clase publica, luego creamos un método publico de tipo IEnumerable implementando el modelo que esta en core/entities le ponemos el nombre que querramos y desde acá vamos a gestionar los datos de la DB.**
17. **Vamos a crear inyección de dependencias basándonos en el principio solid que nos indica que las clases no debe depender de implementaciones concretas (new) sino de abstracciones (interfaces)**
18. **En la carpeta Core/Interfaces/ creamos un archivo tipo interface el nombre debe iniciar con la I seguido de lo que querramos, debemos poner la clase publica, dentro de la clase hacemos la referencia a los métodos que ya tenemos creados usando la clase asíncrona ejemplo**
    1. **Task<IEnumerable<Producto>> GetItems();**
    2. **Podemos referenciar todos los métodos necesarios**
19. **En la clase del archivo repository vamos a implementar la interface que se creo antes en los métodos de este repository no debemos olvidar que son async y Task y poner un await Task.Delay(10); antes del return para esperar la invocación y respuesta.**
20. **En el archivo controller vamos a iniciar con la inyección de dependencias, creamos un método constructor que recibe un parámetro de tipo la Interface (es la que se creo), antes se declara una variable privada solo de lectura de tipo interface y dentro del constructor se le asigna el parámetro recibido, quitamos los new de los métodos que recuperan valores para mostrar o modificar datos y los remplazamos para la variable o abstracción que se lleno en el constructor.**
21. **Vamos a resolver las dependencias con un contenedor de dependencias en el caso de .NET Core se usa el archivo Startup ya que este ya tiene su contenedor de dependencias.**
22. **En el archivo repository hace una inyección de dependencia como la de #20 solo que esta no será con una iterface sino con el archivo context, con esa dependencia context yo puedo pedirle a la base de datos que me liste todos los productos en una listaasincrona, no debo olvidar poner el await para que espere resolución.**