**Guía de creación proyecto multi capa .Net Core**

Proyecto que consta de 3 capas:

**Presentación** (Capa que muestra la información mediante MVC, capa web en este caso)

**Negocio** (Capa encargada de la lógica, el core de la aplicación)

**Datos** (Capa encargada de interactuar con la base de datos)

**Consola** (Capa adicional que actuaría en este caso como capa encargada de proporcionar datos a la capa de negocio donde se gestionarían y se insertarían a través de Datos si es necesario.

## Obtener datos de BD y mostrarlos en la pantalla de la web

\*Es necesario crear una base de datos en local llamada Cuadros, las tablas se crearan mediante migrations(update-database)

1. **Creación de contextos y tablas de BD**

En la capa de datos voy a crear los DbContext(aquello que interactúa con la base de datos e implementa las acciones propias de un repositorio y un UoW).

Voy a basarme en la metodología de trabajo CodeFirst que consiste en la creación de las entidades de datos en el propio código y la posterior replica en la base de datos mediante comandos.

Para esto necesitamos tener instaladas las siguientes librerías:

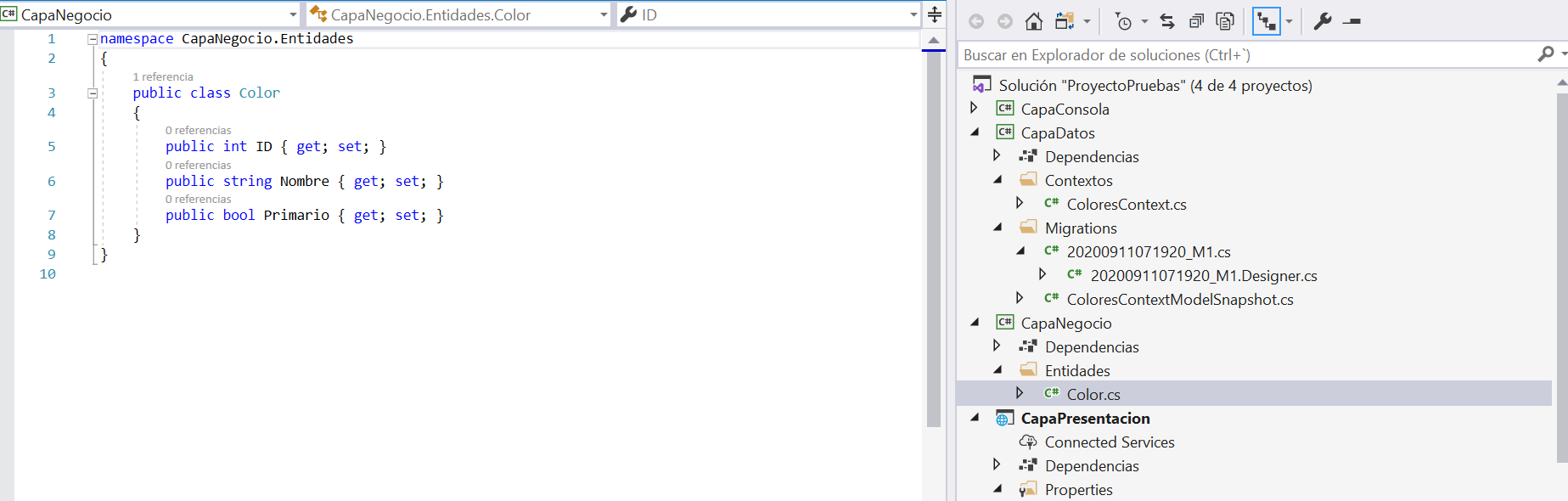
Microsoft.EntityFrameworkCore.SqlServer (**DATOS Y PRESENTACION**)-> -> Permitir las conexiones a base de datos, utilizar cadenas de conexión en las opciones del DbContext,etc..

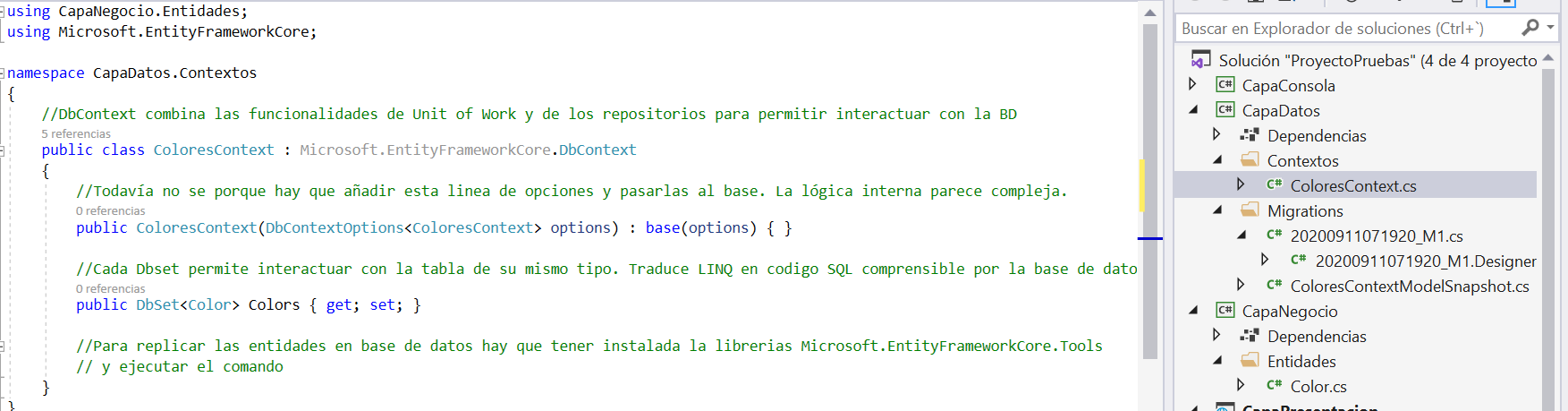
Microsoft.EntityFrameworkCore.Tools(**DATOS Y PRESENTACION**)-> Permite crear migraciones y replicar las entidades en la base de datos.. Add-Migration,Update-Database

Microsoft.EntityFrameworkCore(**SE INSTALA CON LA MISMA SQL**) -> Implementar los DBContext

Lo siguiente es crear el modelo en aquella capa donde este el contexto(Datos) mientras que las entidades pueden estar en la capa de Negocio o incluso en un proyecto transversal.

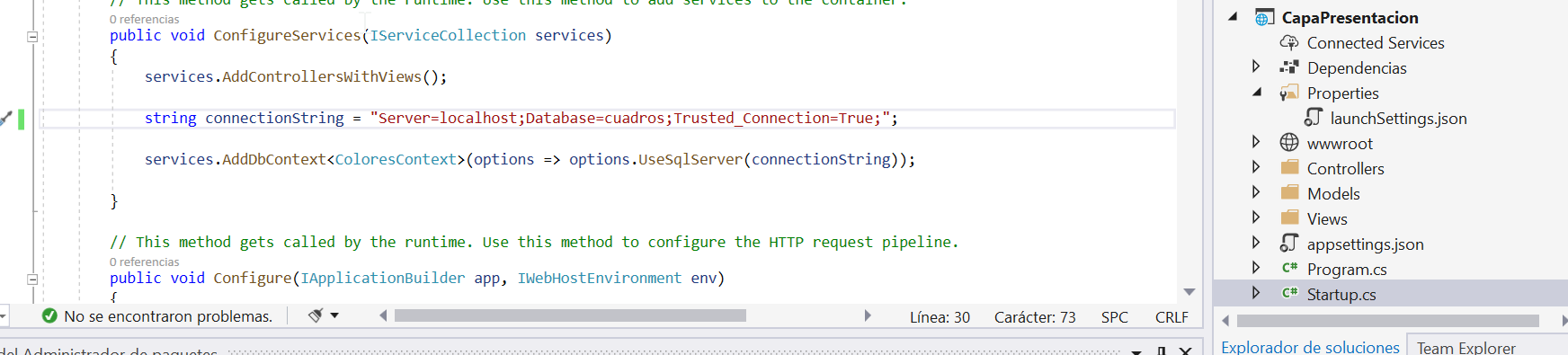
En la capa de datos deberemos tener los contextos de la base de datos y en la capa negocio las entidades vinculadas a estos contextos.





**2) Declarar el contexto y la cadena de conexión.**

En la capa de Presentación, en el archivo startup es donde se declaran los componentes de la aplicación y es aquí donde declararemos los contextos, de la siguiente manera:



Además, aquí declararemos la cadena de conexión a base de datos que vayamos a usar, en mi caso voy a utilizar una base de datos local llamada cuadros.

La base de datos debemos crearla a mano en el servidor, las tablas las crearemos mediante comandos.

Una vez tengamos esta estructura debemos ejecutar los siguientes comandos en la consola de comandos:

**Add-Migration**(nos creara un archivo basado en las entidades que tengamos, una especie de modelo de datos que sirve para ejecutar en la base de datos real)

**Update-Database(**Actualiza la base de datos con la última migración activa)

**3) Creación de un repositorio**

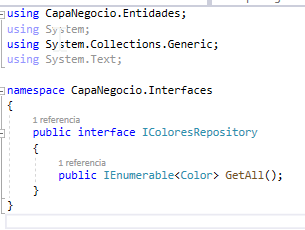
Para tener las funciones más comunes de acceso a datos voy a implementar un repositorio que utilice el dbContext de colores.

Este repositorio implementara una interfaz que estará ubicada en la capa de Negocio para así poder ser utilizada desde las capas superiores de presentación, servicios web etc..

Creo un método GetAll que traiga los registros de la tabla Colores.

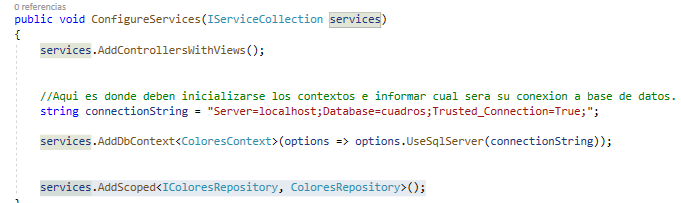


La interfaz se ubica en la capa de negocio:



**4) Inyección dependencias**

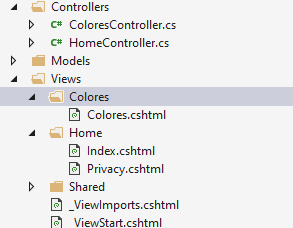
Una vez tengamos el repositorio de colores hay que declararlo en el Startup de la aplicación para poder inyectarlo en los controladores desde los que lo vayamos a usar.



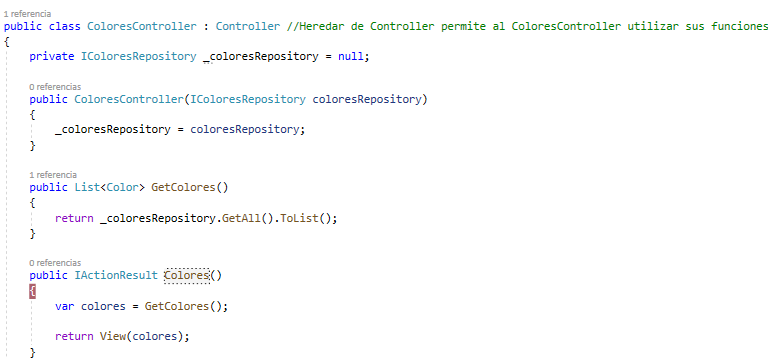
**5) Utilizar el repositorio desde controlador, obtener y mostrar datos**

Ahora ya podemos utilizarlo en controladores. Creo un controlador llamado colores y desde este llamo a la función GetAll del repositorio. Creo también una vista llamada colores en una carpeta llamada vistas. Todo esto en la capa de Presentación.

La estructura de carpetas debe parecerse ahora mismo a algo así:

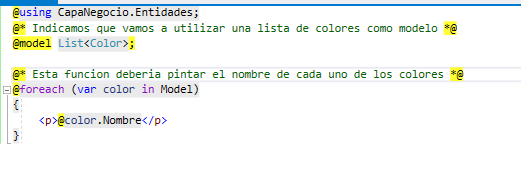


El controlador.



Para poder utilizar la función IActionResult es necesario que herede de Controller(ColoresController: Controller), de esta forma podemos enviar a la vista colores una lista de colores y pintarlos.

La Vista

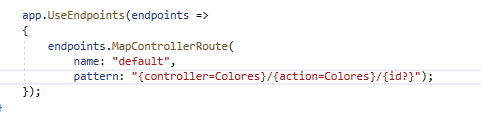


**6) Cambio de la función de inicio**

Cambiando esto la función que se ejecute al iniciarse la web será la que definamos.

En el archivo StartUp, en la función Configure, al final del todo, añadir lo siguiente:

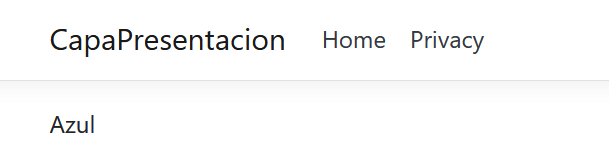
(O modificarlo, si es que ya existe, para que este tal que así)



**7) Mostrar datos**

Cuando la función Colores se ejecute ira a buscar la vista siguiendo la siguiente estructura(controller/función) Colores/Colores, por tanto, buscara en la carpeta colores una vista llamada colores.

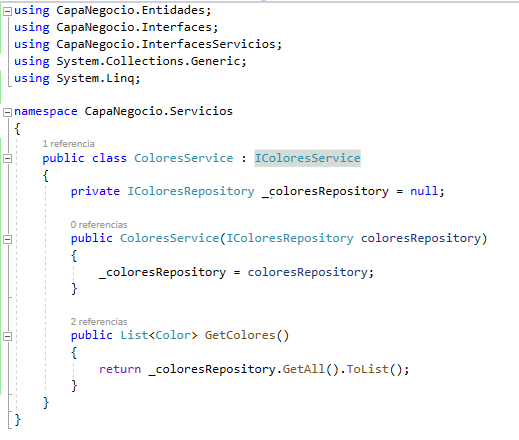
He añadido a mano un registro en base de datos (Azul)



**8) Utilizar la capa de negocio**

Con esto ya valdría para traernos datos desde base de datos, pero para respetar el sistema de capas voy a crear un servicio en la capa de negocio (Que se llame ColoresService) que será el encargado de la lógica y las llamadas a los repositorios (en este caso la lógica es mínima, pero si tuviésemos que hacer algo mas complejo lo mas correcto creo que es utilizar la capa de negocio.

Así conseguimos desvincular la capa superior del origen de datos dándonos igual de donde saque los datos: la capa de presentación solo pinta cosas y gestiona vistas.



De esta manera podemos reutilizar este servicio desde otras capas 1(servicios web o consola como es el caso del proyecto que voy a crear ahora)

\*Recordad añadir este nuevo servicio en la inyección de dependencias: (Startup -> ConfigureServices)



## Capa Consola

Voy a leer un fichero de texto, copiar sus datos, que serán una lista de colores, e insertarlos en la base de datos.

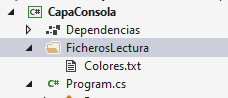
**1) Crear proyecto de consola**

Al crear un proyecto de consola automáticamente se genera una clase llamada program que es la que voy a utilizar:



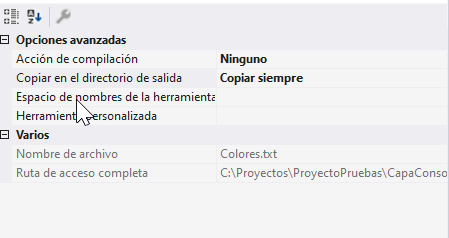
**2) Leer el fichero**

Crear una carpeta y añadir un archivo de texto con una lista de colores separados por coma o por lo que se quiera.

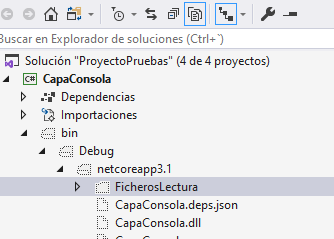




Es importante que a este fichero le indiquéis en sus propiedades que se copie siempre al compilar pues de lo contrario no encontrareis el fichero en la ruta deseada.



Al compilar con copiar siempre:

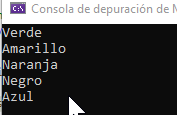


\*Para ver los archivos ocultos se debe marcar el botón: 

En local usando Base directory + ruta del fichero se va a buscarlo a 

->En otros entornos habrá que buscarle una ruta determinada en el servidor...

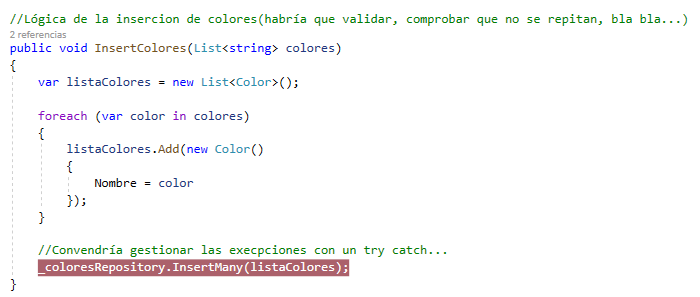
Al ejecutar el programa tal que así se deberían mostrar en consola los colores leídos desde el fichero:



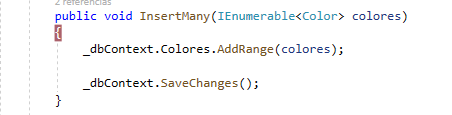
**3) Inserción en BD(crear funciones)**

El siguiente paso será insertar estos colores en base de datos. Para esto vamos a utilizar el servicio que creamos anteriormente (ColoresService)

En el servicio ColoresService he creado la siguiente función:



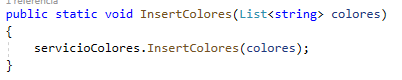
Que llama a insertMany del repositorio



\*Cada vez que se añade una función al repo hay que añadirla también a la interfaz que es la que realmente vamos a utilizar.

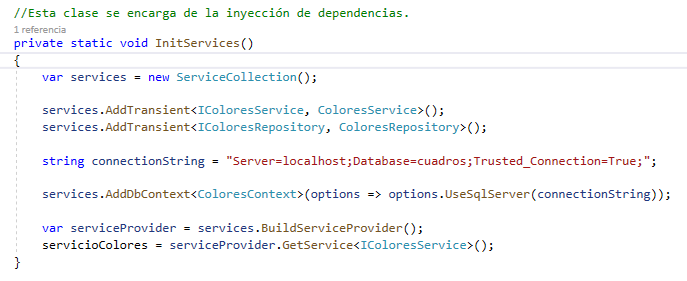


Añado la función que llama al servicio de inserción:



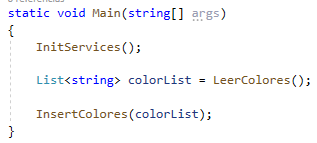
**4) Declarar servicios/inyección dependencias**

Para utilizar servicios, repositorios, etc desde el proyecto de consola hay que declarar los servicios y la inyección de dependencias de la siguiente manera. (como se hacía en el startup del proyecto CapaPresentacion)

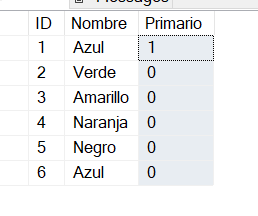


**5) Ejecutar y guardar los datos del fichero.**

Y el main lo dejo de esta manera:



De esta manera lee del fichero los colores y los pasa al servicio este a su vez al repo y este otro los inserta en base de datos.



Faltaría validar duplicados, nombres, etc.… y controlar excepciones mediante try-catch…