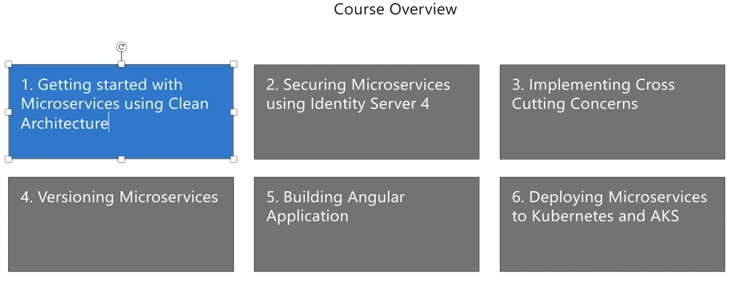
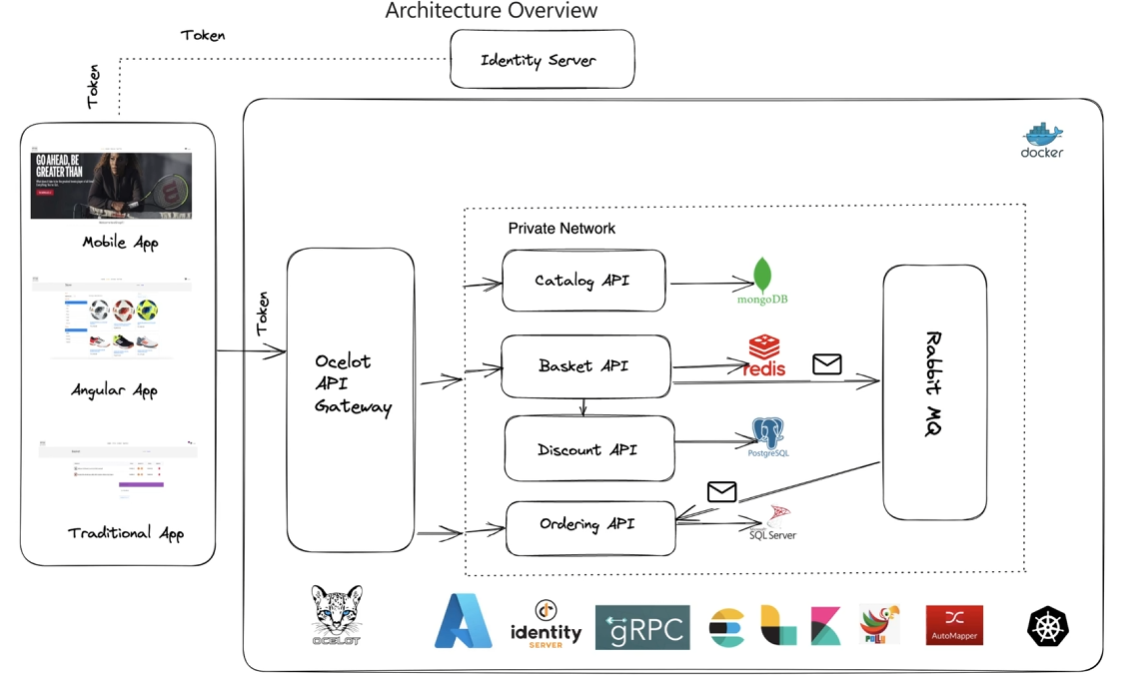
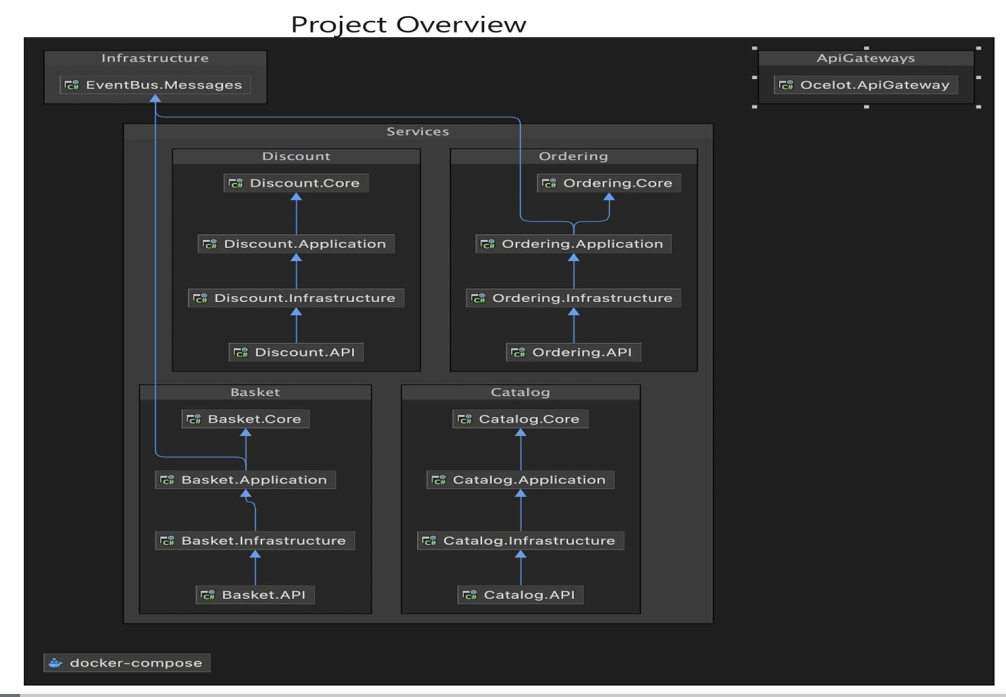
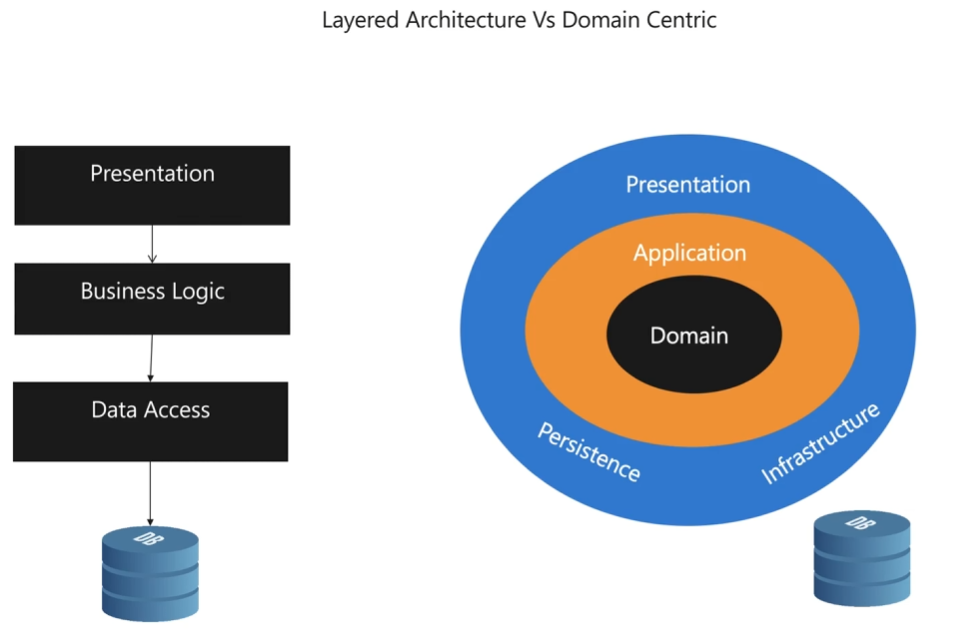
Curso 🡪 <https://github.com/rahulsahay19/eShopping>









una diferencia entre la arquitectura en capas y la arquitectura centrada en el dominio. Como puedes ver aquí en la arquitectura en capas, todo apunta hacia lo inverso y eso es hacia

datos en sí. Entonces, en algún momento, ¿qué pasará con esta arquitectura en capas como, ya sabes, todas estas

Los componentes, ya sabes, se acoplarán estrechamente y eso a su vez estará, ya sabes, estrechamente acoplado.

hacia los datos en sí, en los que se agregan nuevas funciones o cualquier cosa, ya sabes, distinción o tal vez. la pieza de refactorización también con las tecnologías más nuevas, se vuelve un poco, ya sabes, ya sabes,

difícil para eso o desafiante para los desarrolladores.

Entonces, si su aplicación va a ser de tipo complejo, que tendrá una capa de dominios o yo lo haría

digamos subsistema donde cada dominio está separado y puede administrarse individualmente.

Ahí es donde los microservicios serán una muy buena pieza como esta que implementarás.

Y para implementar ese diseño de microservicios, es necesario comprender a fondo su patrón.

Y ahí es donde, ya sabes, la capa de dominio entra en escena y esta capa de dominio tiene su propia

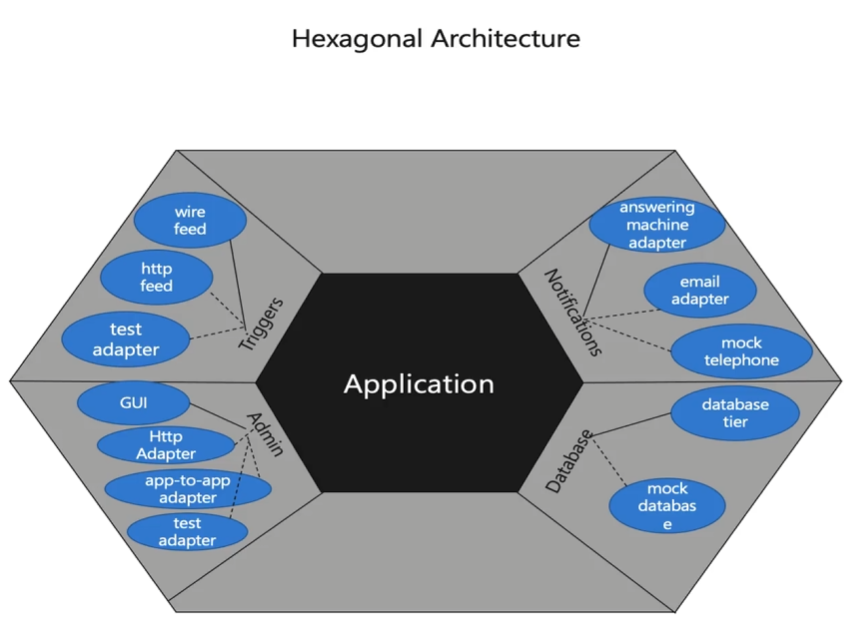
manera, su propia historia de implementación. Y todos los casos de uso están en el centro y otras cosas como los idiomas.

La presentación y la persistencia son sólo los detalles aquí que puedes ver en la propia periferia exterior.

Ahora, aquí el dominio es el centro que está encapsulado dentro de esta aplicación, en la cual

En este caso, digamos la capa de aplicación y todas las dependencias apuntan hacia el dominio mismo.

Dicho esto, ahora la capa de dominio en sí tiene una variedad de implementaciones.

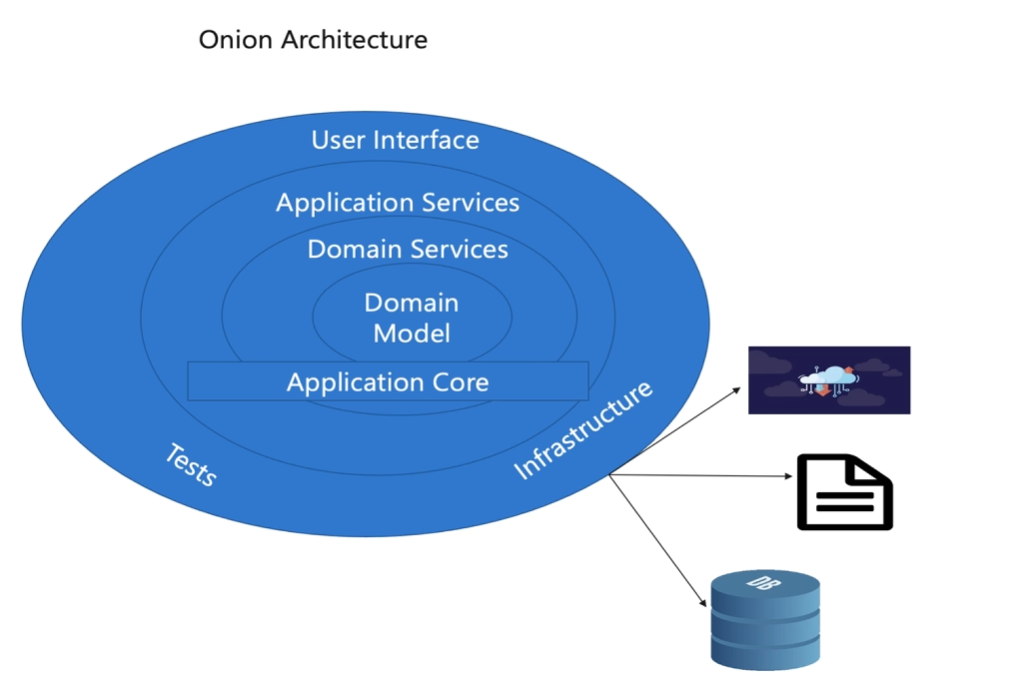


Así que ahora la esencia de este sistema en particular es hacer que su sistema, ya sabe, sea independiente de

cualquier tipo de dependencia.

Digamos que la interfaz de usuario es independiente de cualquier tipo de dependencia que pueda ejecutarse y aún está aislada.

No requiere que su sistema back-end esté en funcionamiento para probar esos sistemas en particular.

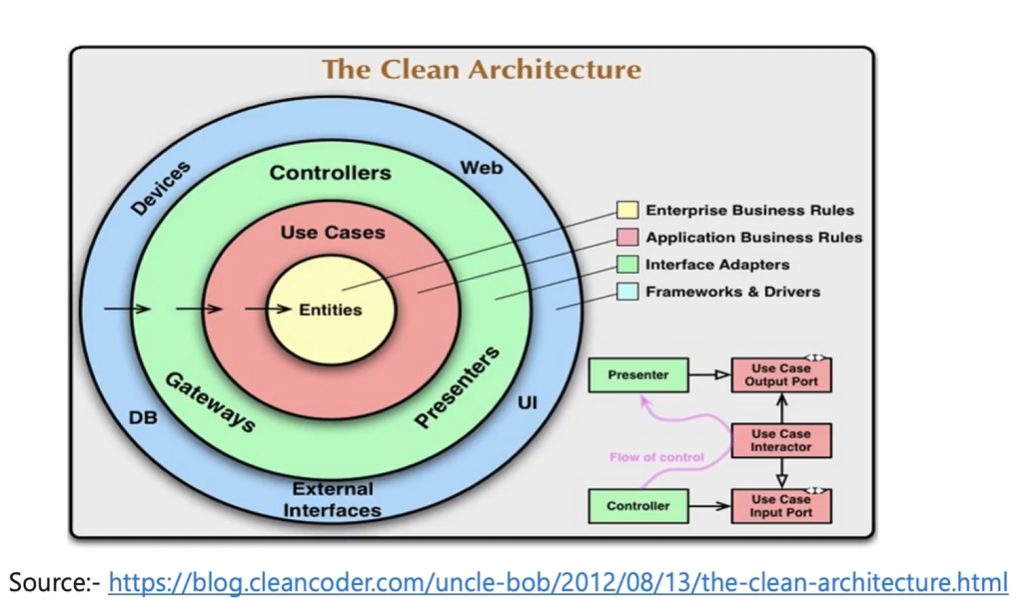


ves que todas las dependencias apuntan hacia la inversa,

lo que significa que no hay, ya sabes, dependencias, en cierto modo, todas las capas son interdependientes.

Entonces, la esencia completa de este segmento es que su sistema está diseñado de la manera que debería.

Ser independiente de cualquier tipo de dependencia externa o de la propia interfaz de usuario para que pueda probarse de forma aislada.



los dominios están básicamente representados por entidades rodeadas por sus casos de uso.

Y luego esos controladores o puertas de enlace o presentaciones, presentación, puedes ver cualquier cosa.

Así que estas son las cosas que son como en el caso del patrón centrado en el dominio, ya sabes, esto es lo mismo

cosa ordenada por un patrón diferente en sí mismo.

Y en este caso es esta arquitectura limpia.

Entonces, ¿qué ha hecho bien el tío Bob?

Así que ha incorporado este patrón arquitectónico de Eva Jacobsen también para explicar cómo

La capa de presentación y la capa de aplicación deben estar conectadas.

el patrón de puertos y adaptadores a través del cual, ya sabe, puede conectar y usar una especie de arquitectura.

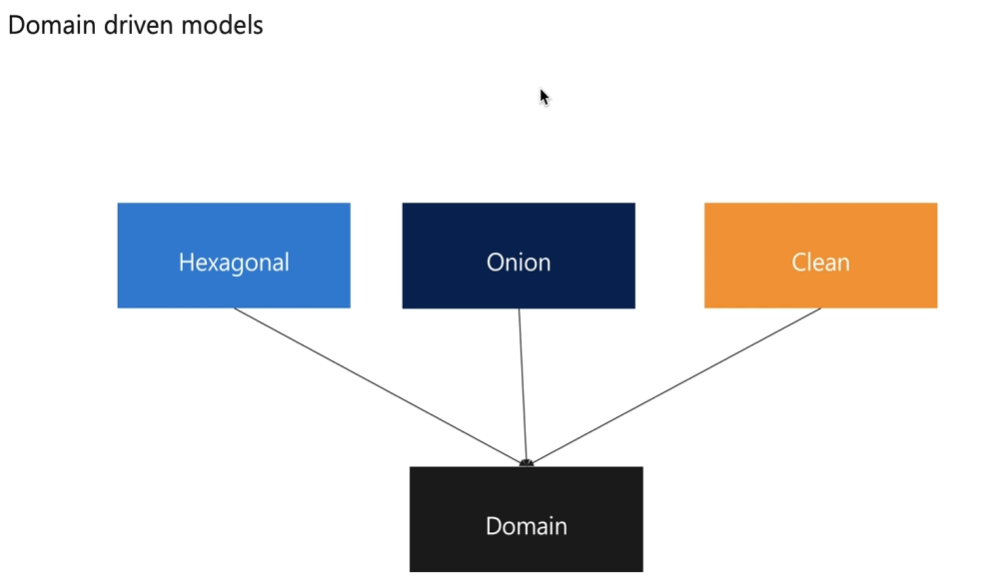
como si pudiera inyectar las dependencias como se esperaba.

Y en cierto modo hace que esta aplicación sea tan sólida que es independiente de cualquier tipo de, ya sabes,

dependencias.

Como todo sistema, se puede probar de forma aislada.

Esa es la esencia de todo este diseño impulsado por el dominio.

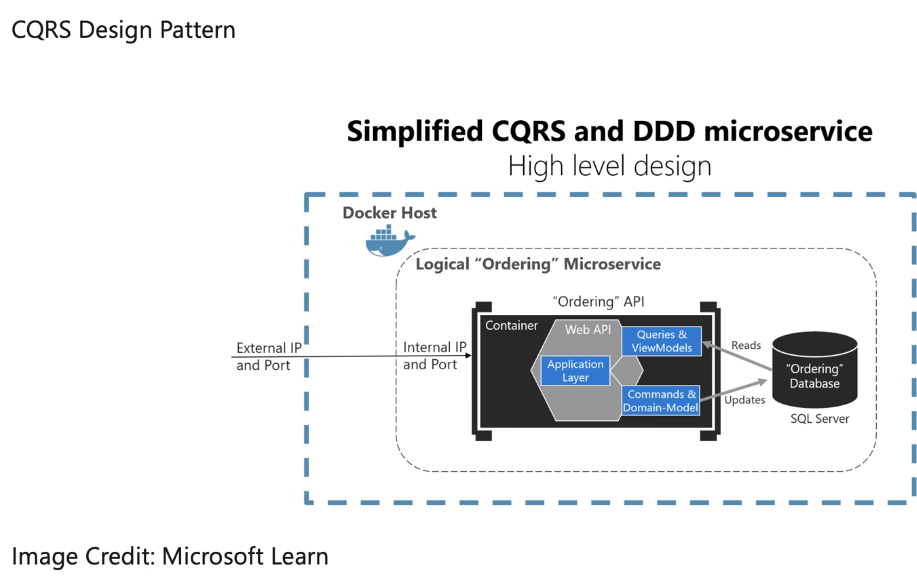


Todos estos patrones de arquitectura mantienen el dominio en el nivel central.

Y lo único que difiere aquí es que hay algunos pequeños cambios de implementación allí.

Dicho esto, en este caso, nos apegaremos a esta arquitectura limpia e implementaremos.

**CQRS Pattern**

****

este patrón de diseño y significa Segregación de Responsabilidad de Comandos y Consultas.

Y este patrón de diseño va a ser muy importante en nuestro curso porque estamos implementando todos

microservicios que utilizan una arquitectura limpia, en la que aplicará este patrón de diseño completamente en

cada microservicio, sea lo que sea que vayas a escribir.

Y este principio importante es que básicamente segrega su consulta en una

sección y, ya sabes, comandos en otra sección.

Entonces, la idea básica es que siempre que una consulta sea similar a estas consultas, devolver un resultado que no

cambian el estado del sistema y están libres de cualquier tipo de efectos secundarios en los que se ejecutan los comandos.

Básicamente significa cambiar el estado del sistema. Entonces, antes de eso, si ves esta imagen en particular, directamente, ya sabes, te dice cómo

externo y el puerto está actuando.

Y tiene consultas y ve modelos en un lugar, que puede ver aquí y en otro lugar que

están teniendo sus comandos y modelo de dominio.

Entonces, una sección es responsable de que usted lea la consulta de lectura, sea lo que sea que solicite la consulta,

diga obtener una lista de productos, en cuyo caso la consulta es esa, por lo que le devolverá esa respuesta

de la respuesta del producto.

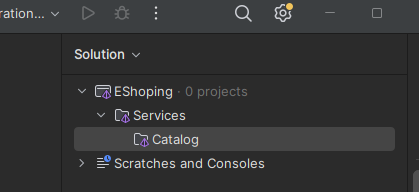
Y digamos que va a crear un nuevo producto o actualizar un producto o digamos revisar la cesta,

entonces, en ese caso, básicamente estás, ya sabes, emitiendo un comando para cambiar el estado del

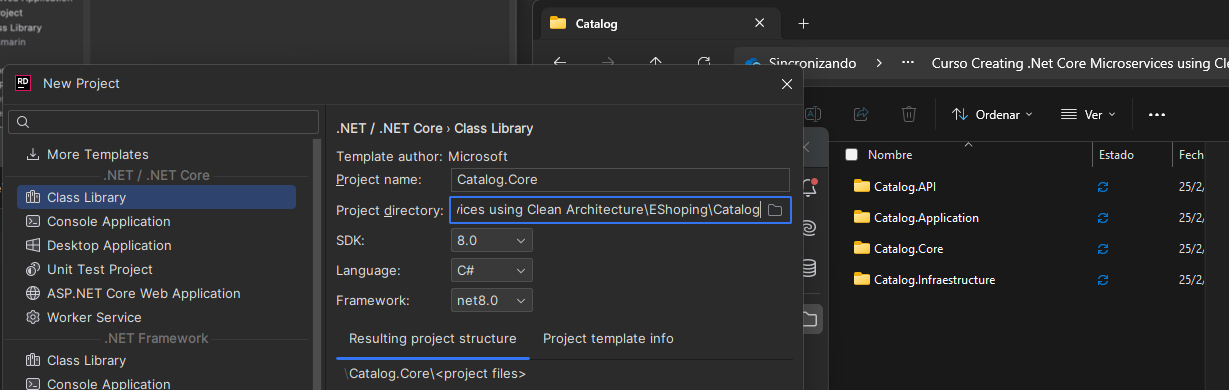
sistema.

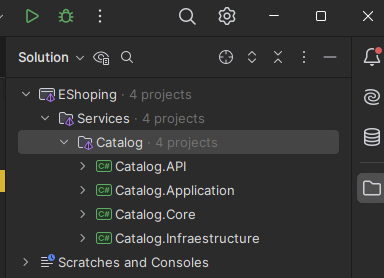
Ahí es donde ocurre esta actualización y ahí es donde hay un ejemplo de la base de datos de pedidos, que

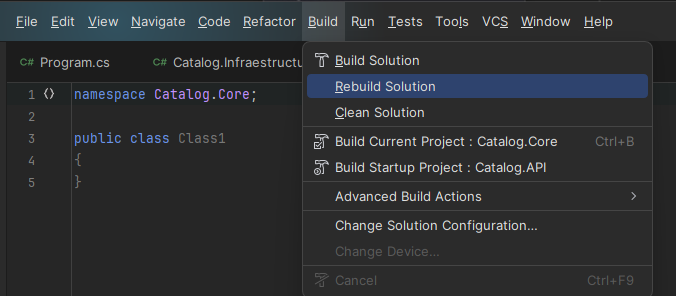
está basado en SQL Server.

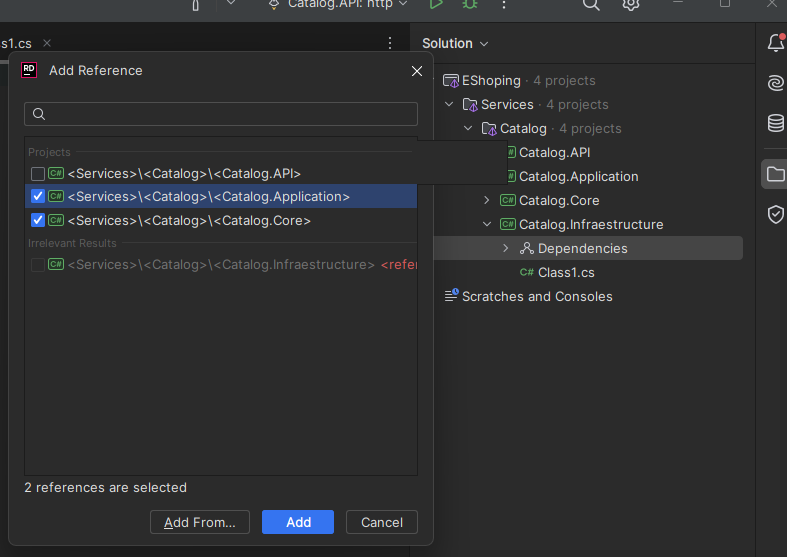


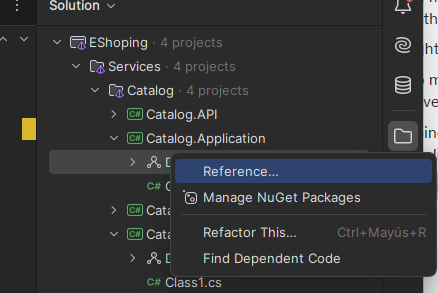


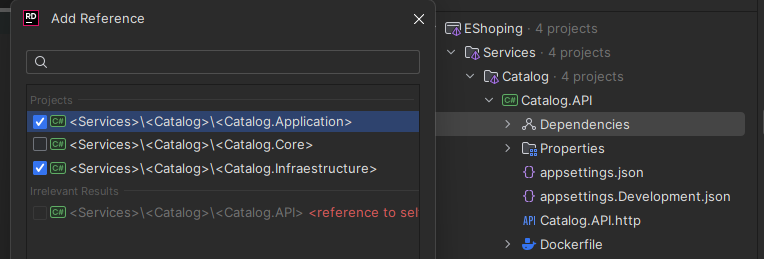
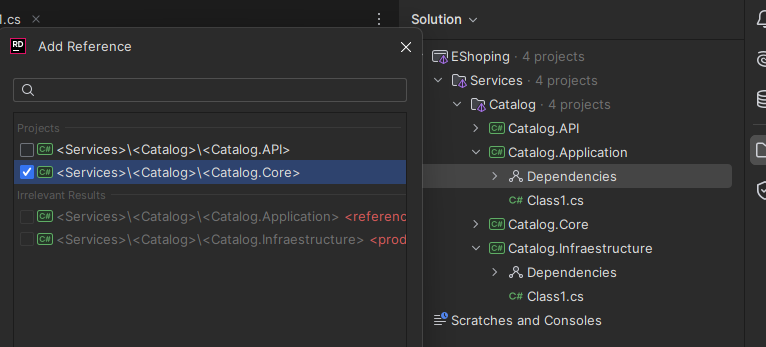




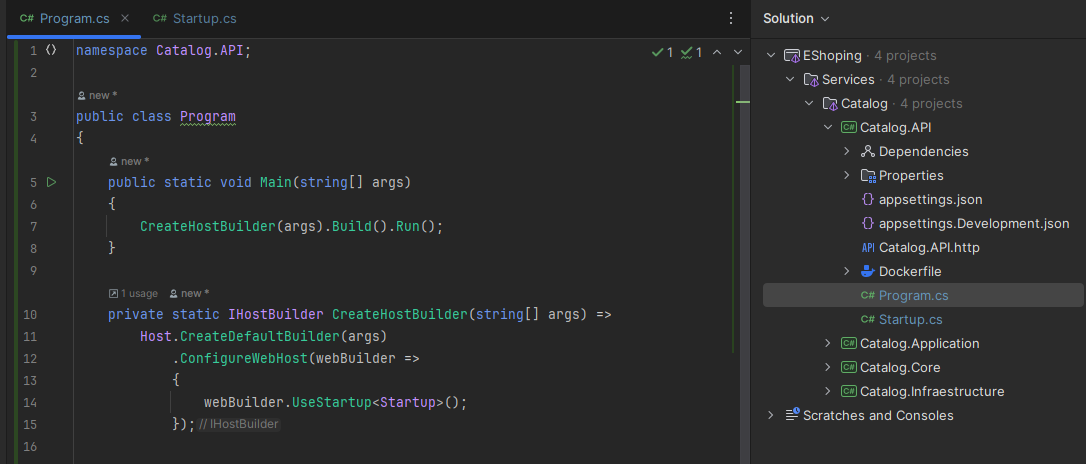


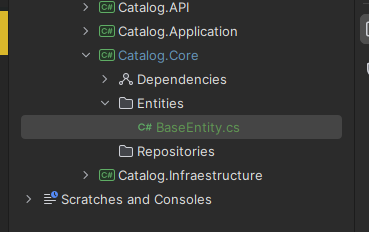
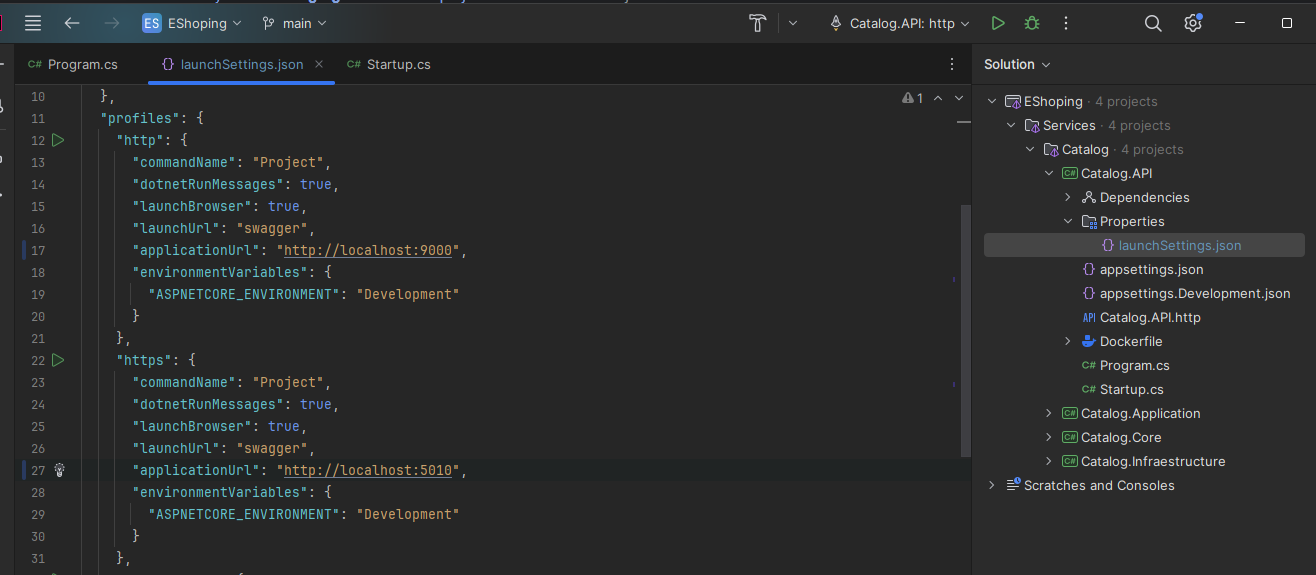




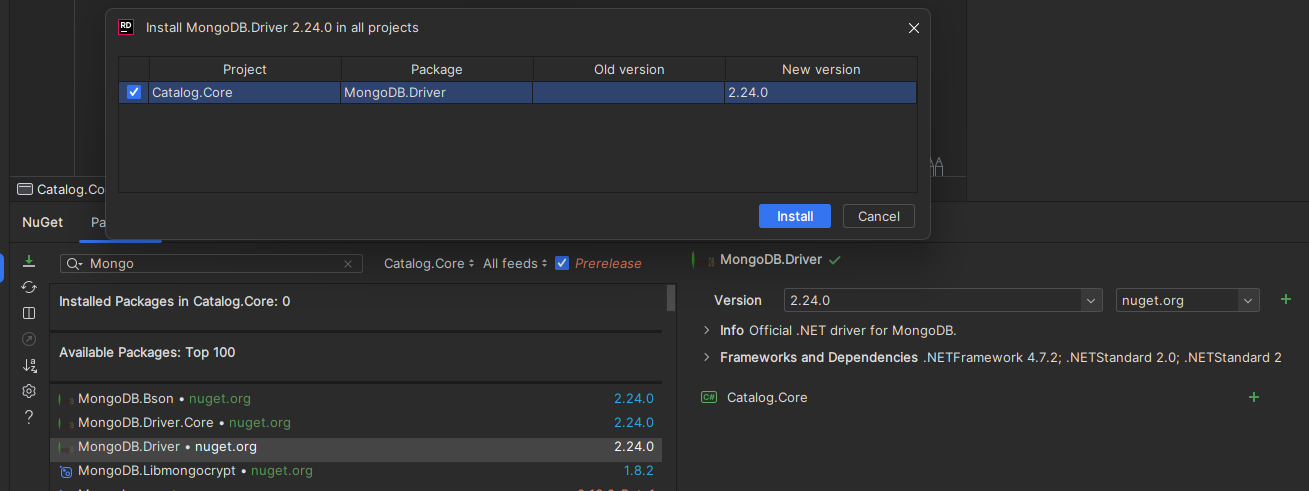


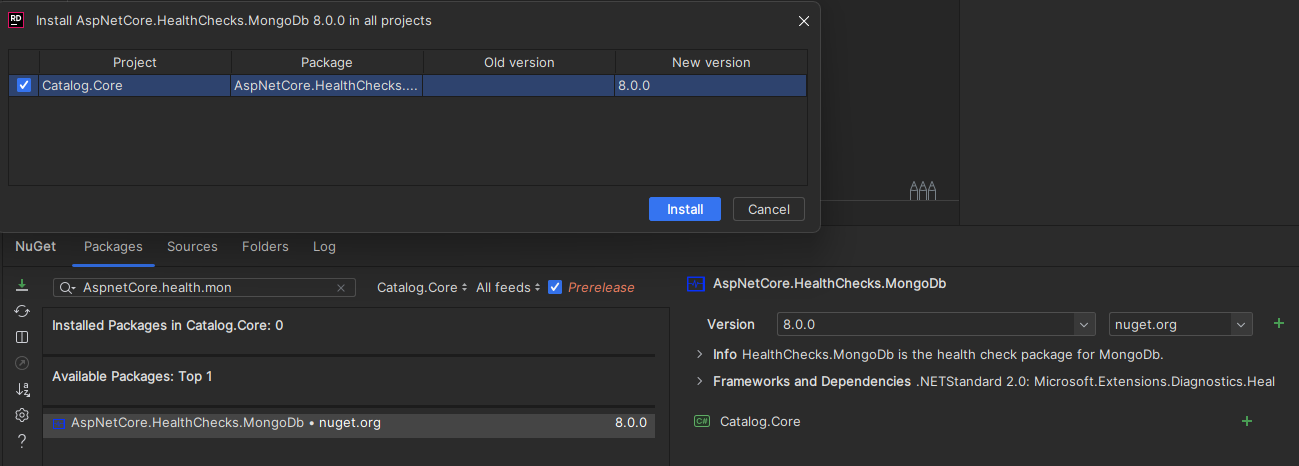




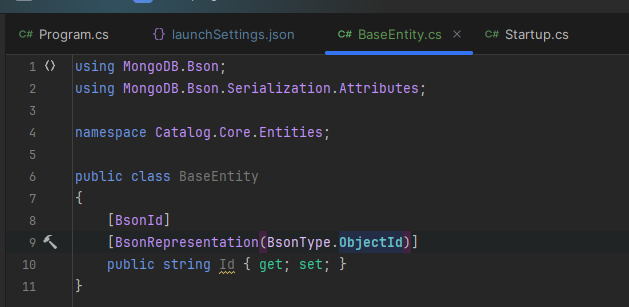


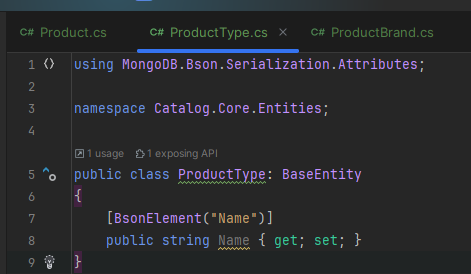
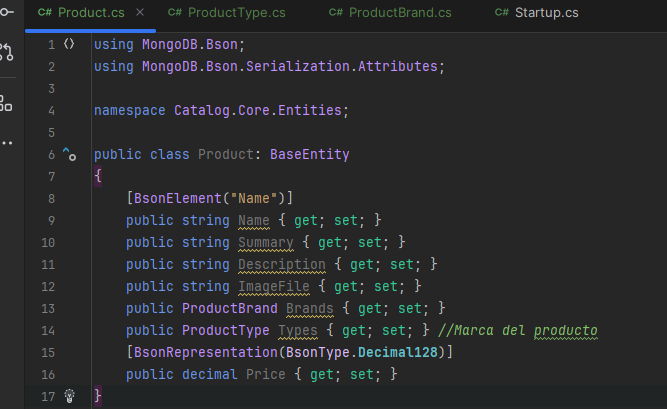
Hagamos rápidamente el controlador MongoDB e instalémoslo.





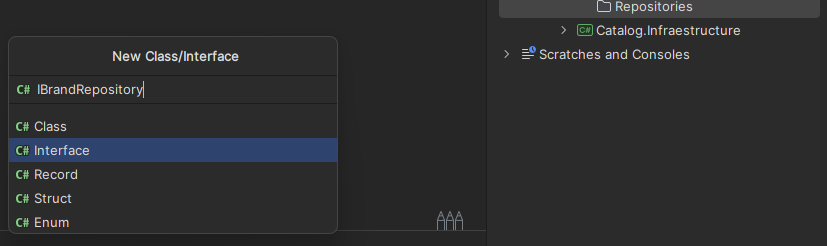
Crear un clase base de las entidades

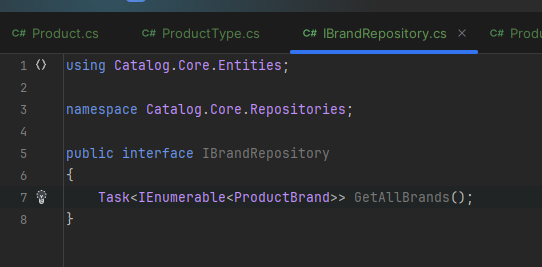


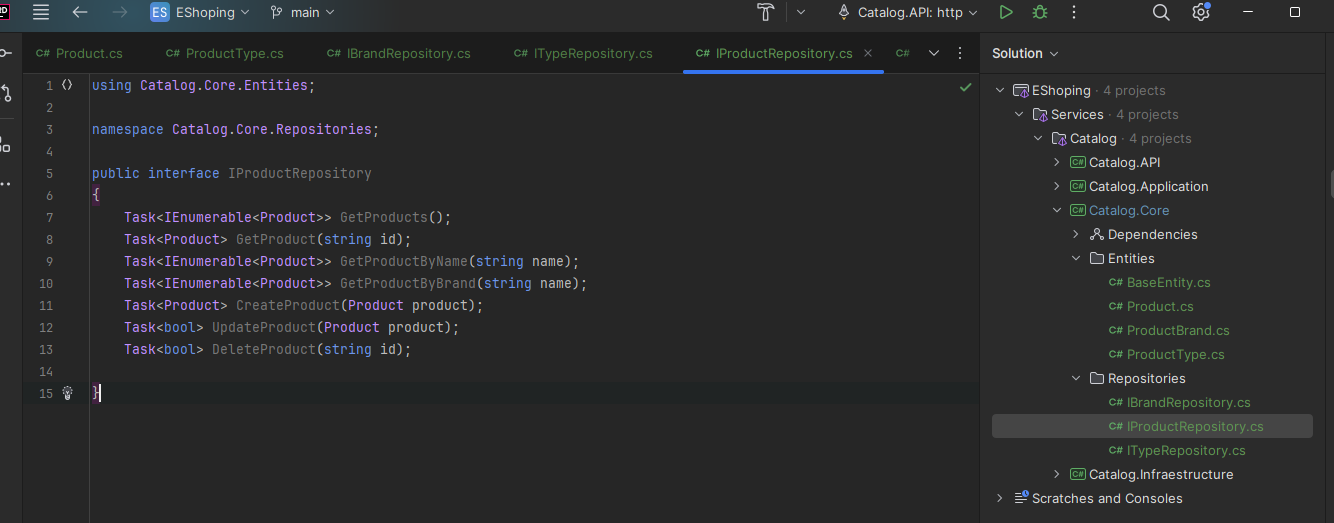


Lo primero que voy a crear es, digamos, un repositorio de marca

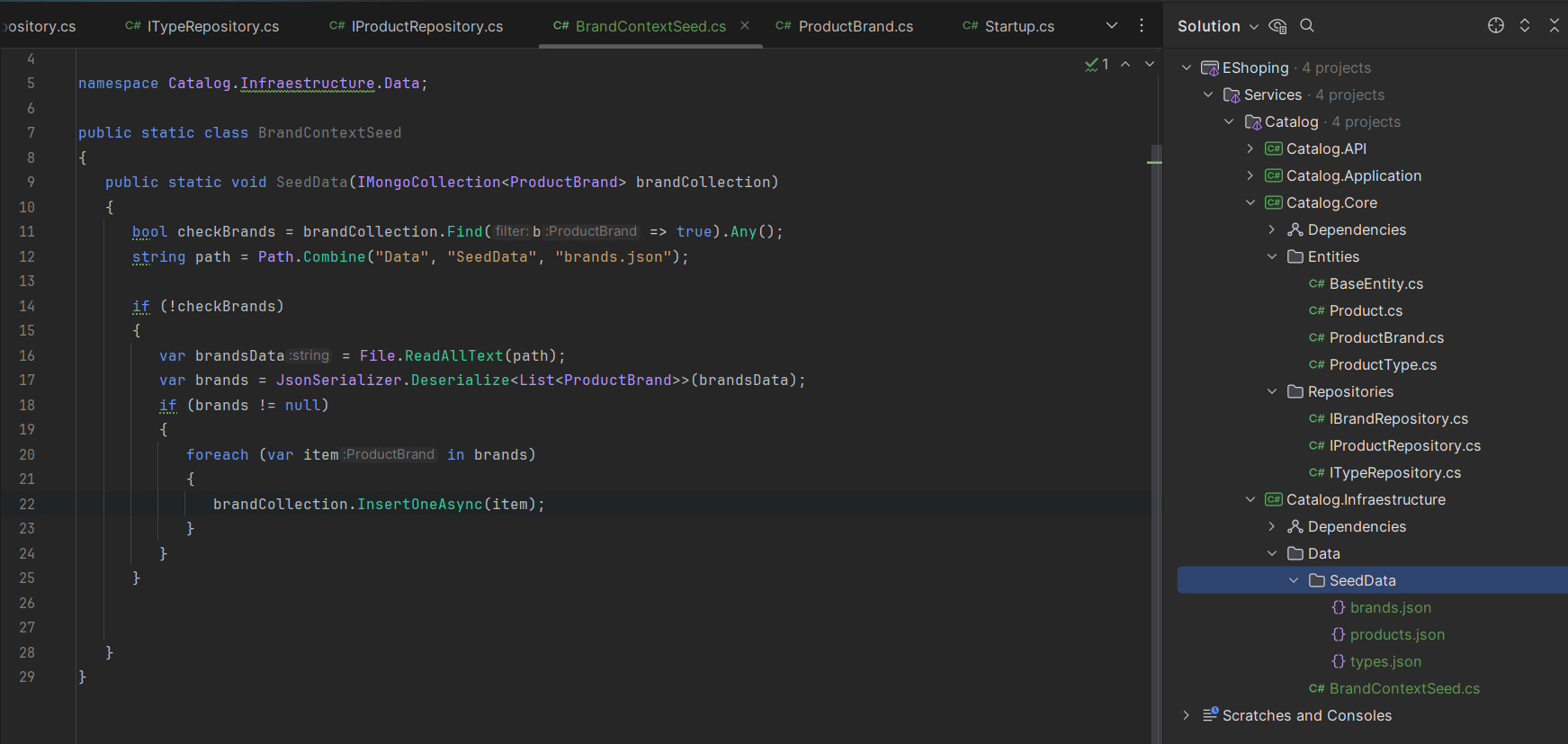
Entonces estas interfaces no son más que los contratos que implementaremos como parte del patrón de repositorio.





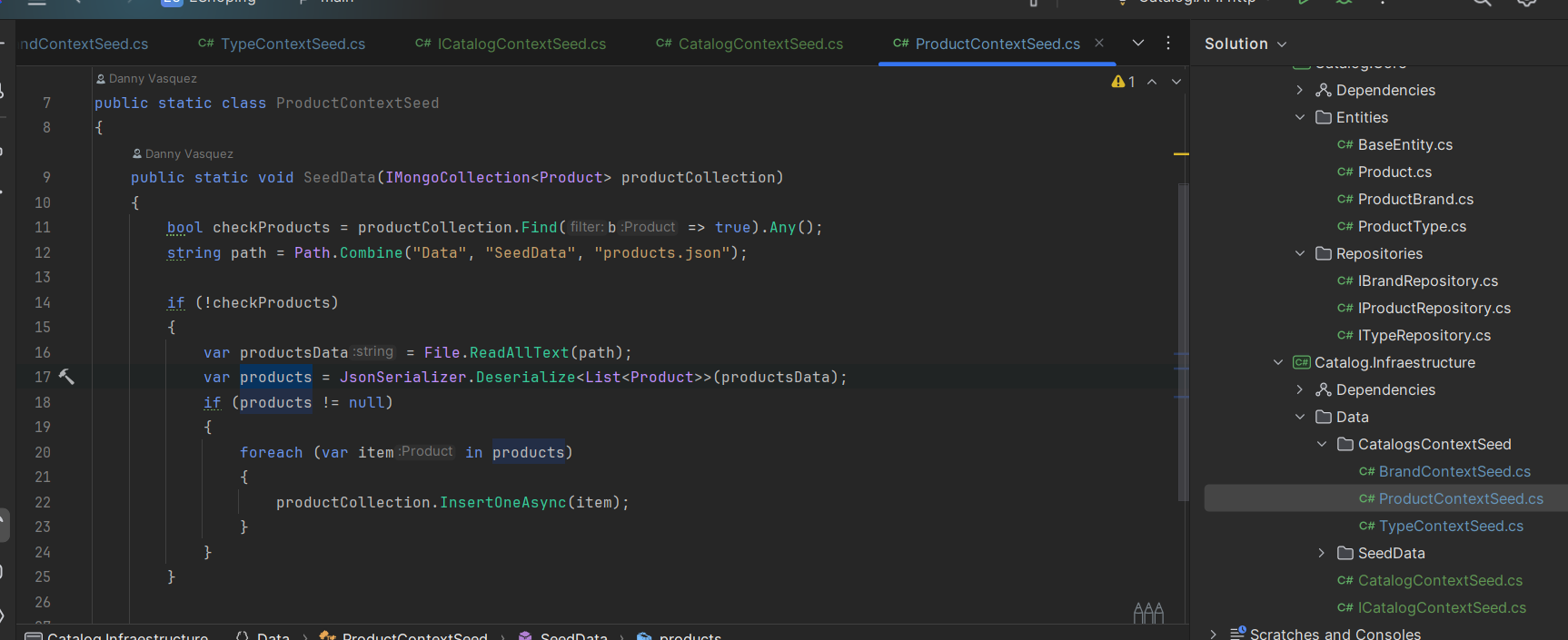


Agregar datos semilla para la base de Catalogos



Crear un contrato(mongo), incluir cadena de conexión

Entonces, lo que básicamente estoy tratando de hacer es leer las cadenas de conexión de la base de datos desde la configuración de la aplicación.

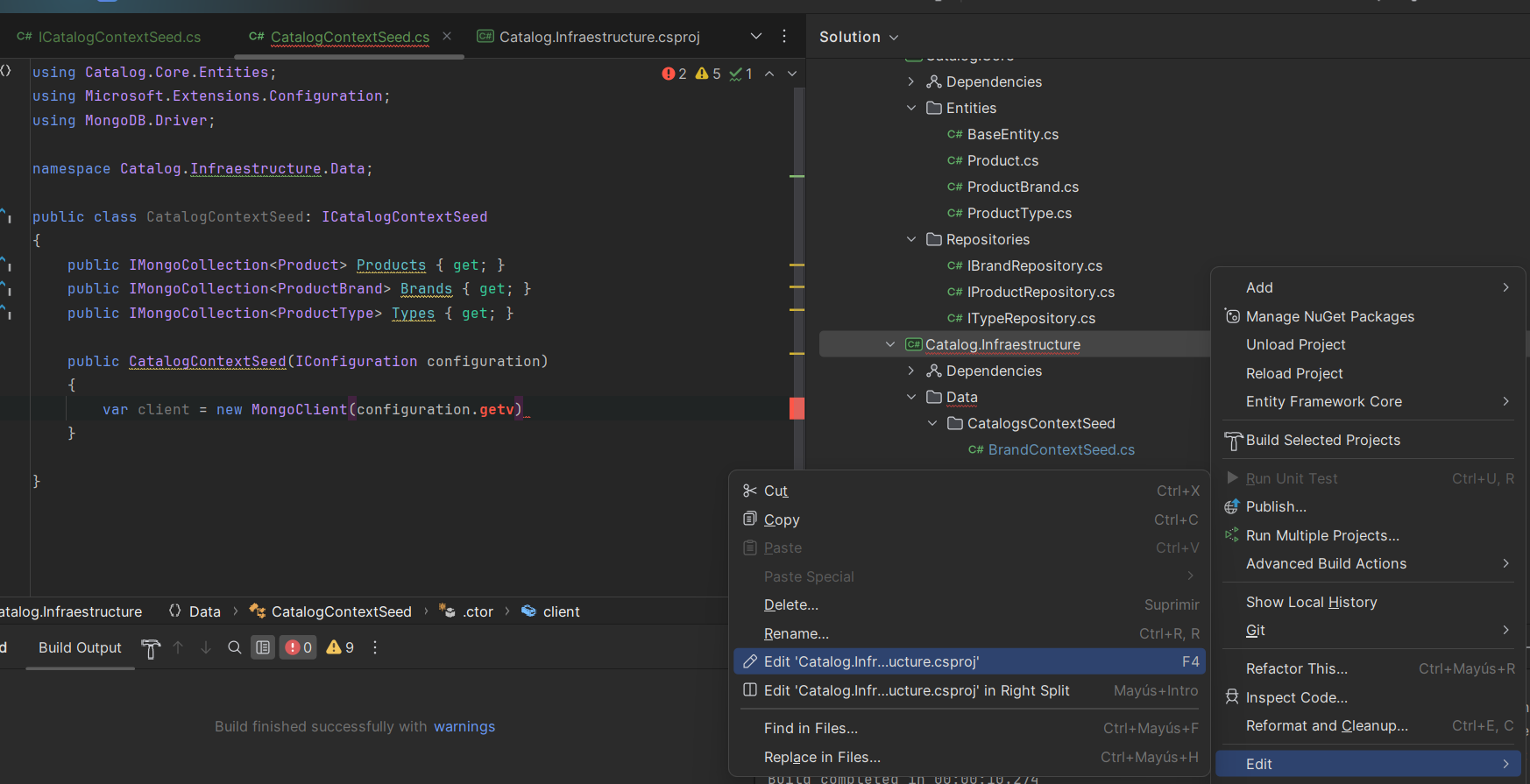


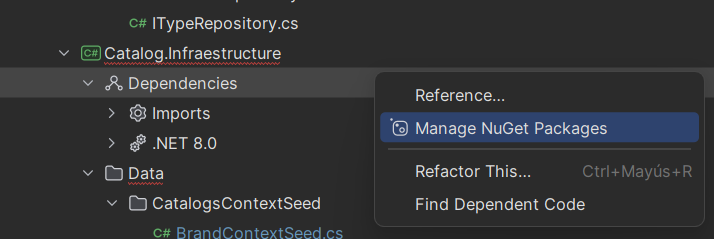
Entonces, lo que básicamente estoy tratando de hacer es leer las cadenas de conexión de la base de datos desde la configuración de la aplicación.

Entonces, ¿cómo podemos hacer eso?

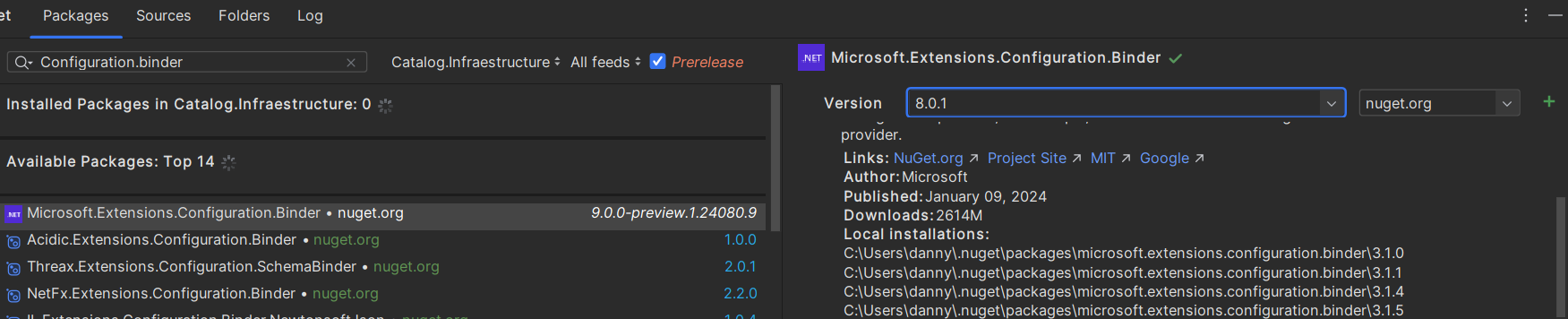
Y podríamos hacer eso con algo llamado con ese valor de obtención y este valor de obtención no viene

porque probablemente no he incluido la biblioteca, que es la carpeta de este.



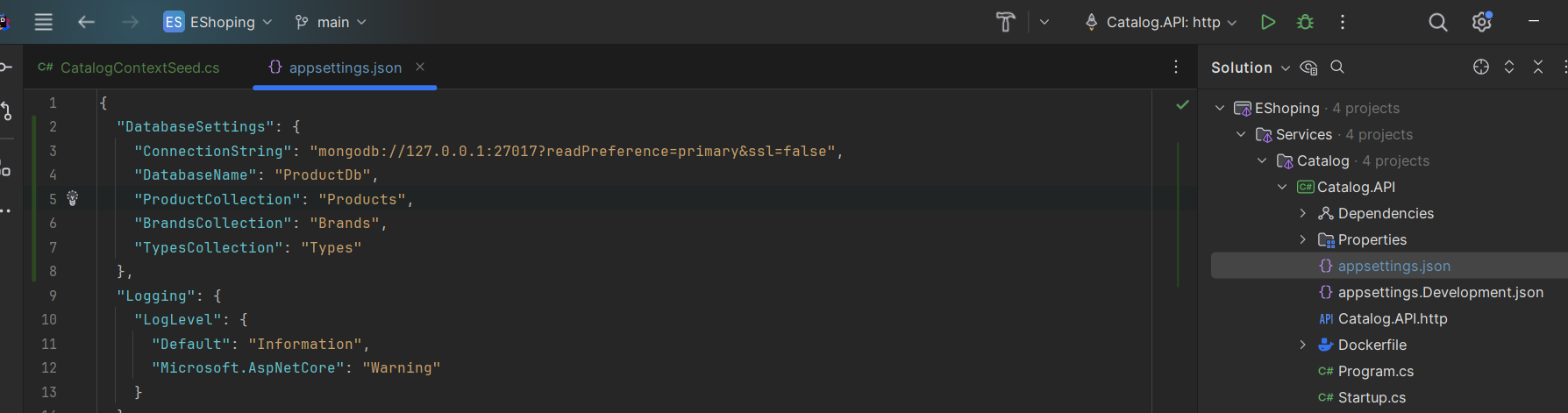


Instalar de la versión SDK(net 8)

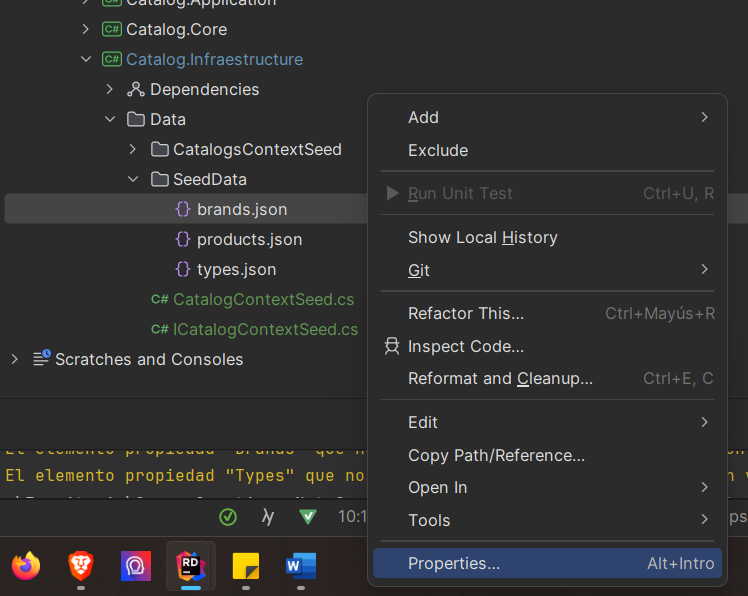


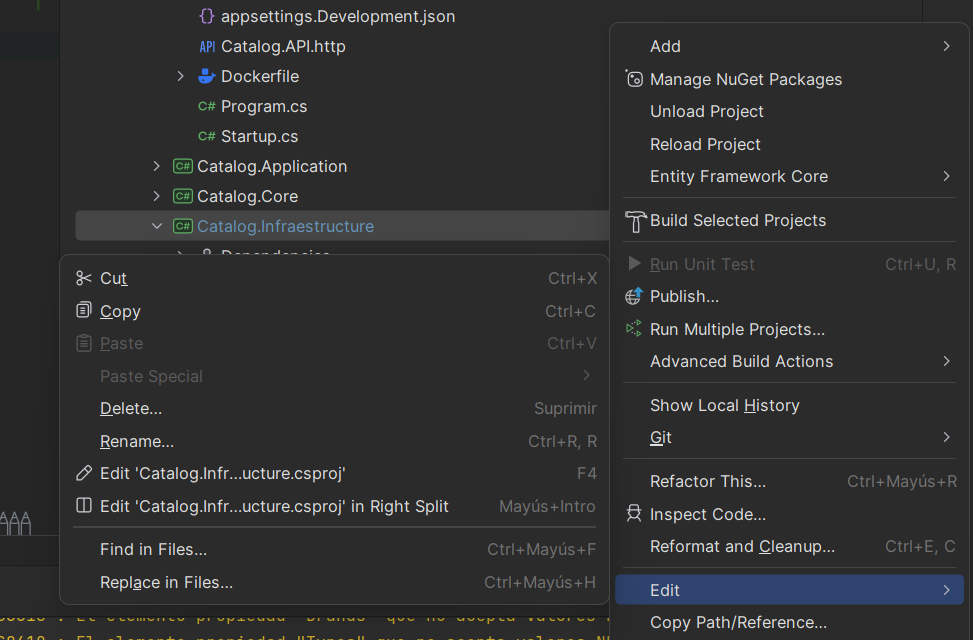
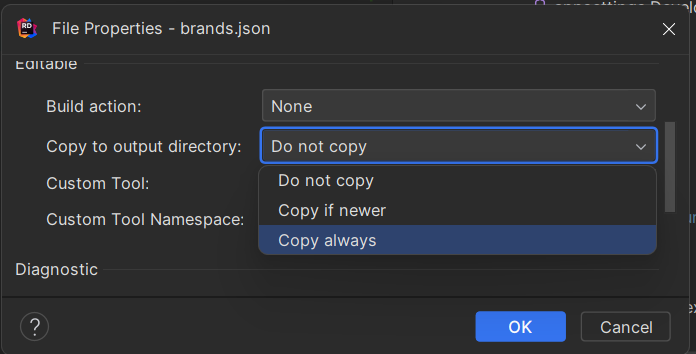
Entonces deberíamos ver esto en acción como si debería resolverse para mi tipo de obtención de valor. Necesitamos hacer eso como si básicamente necesitáramos la cadena de conexión para eso. Entonces ahora debería darme el valor de obtención y este es de tipo cadena.



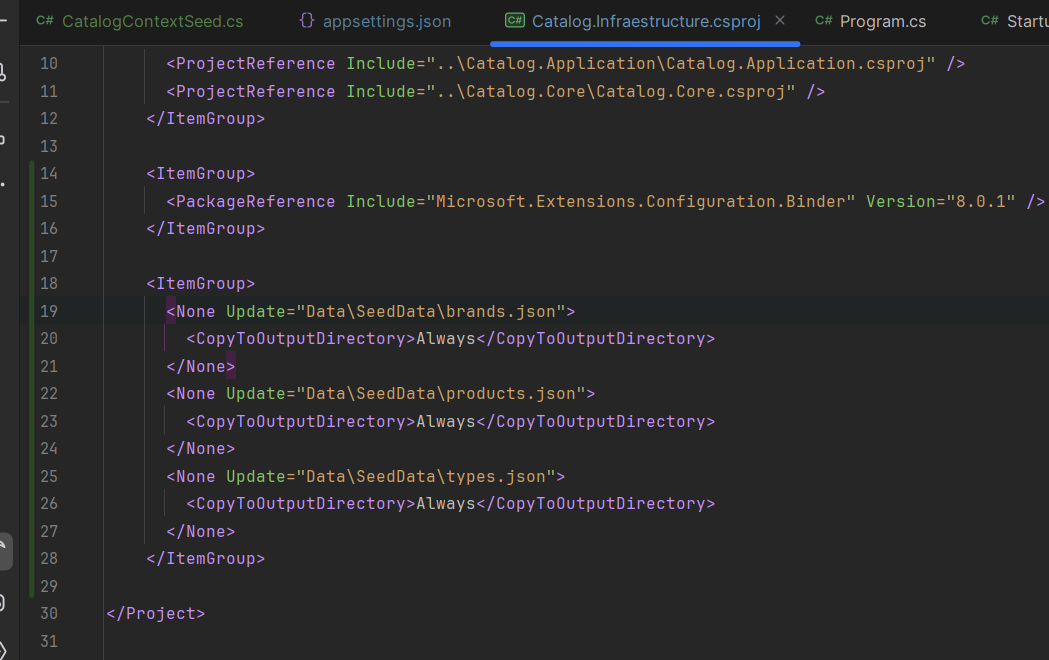


Cambiar las propiedades de los archivos json de catalogos, para q puedan acceder y cargarse en la base.

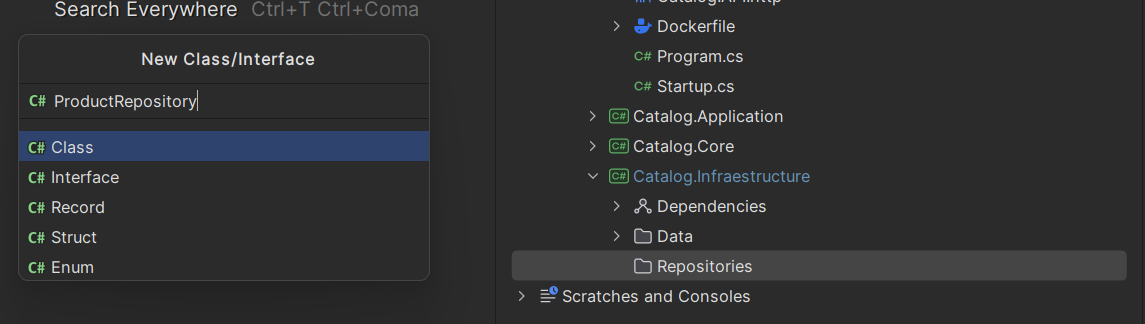


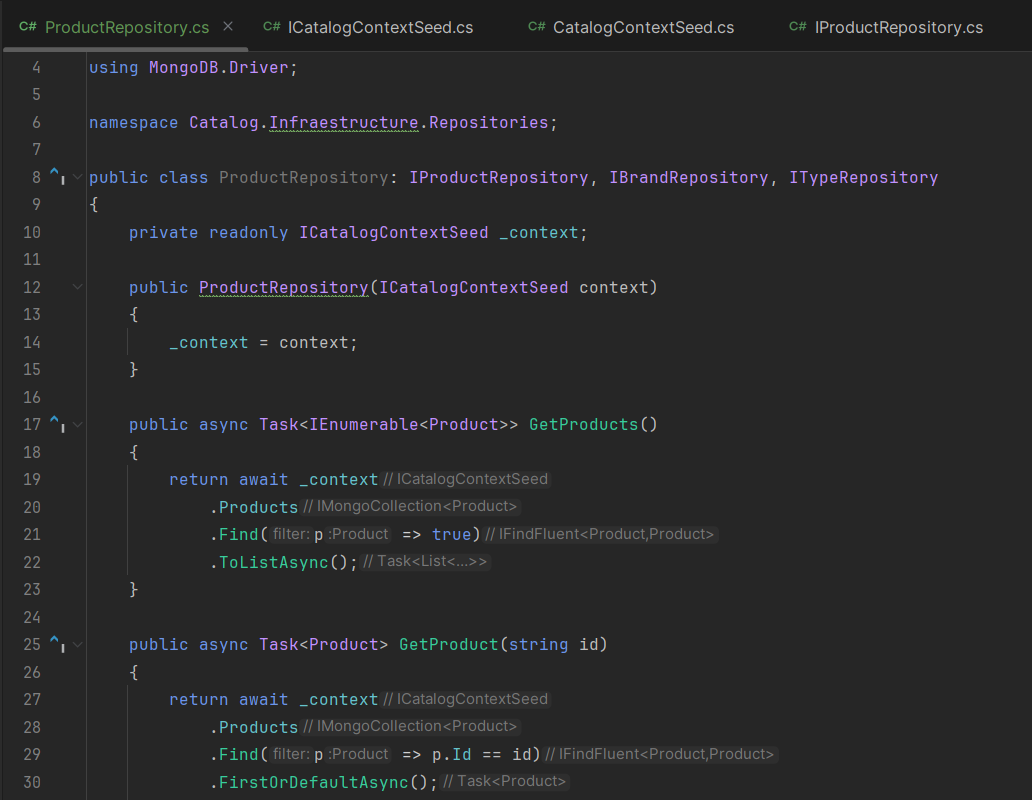


Se agregan automáticamente en el archivo del proyecto.



Trabajar con Repositorios





Para sacar los handler para la api

Crear carpetas para separar comandos de consultas, carpetas para handlers, responses, mappers, y usar el patrón mediador en sí, como la biblioteca Mediator, e instalar los mapeadores (Auto Mapper)

