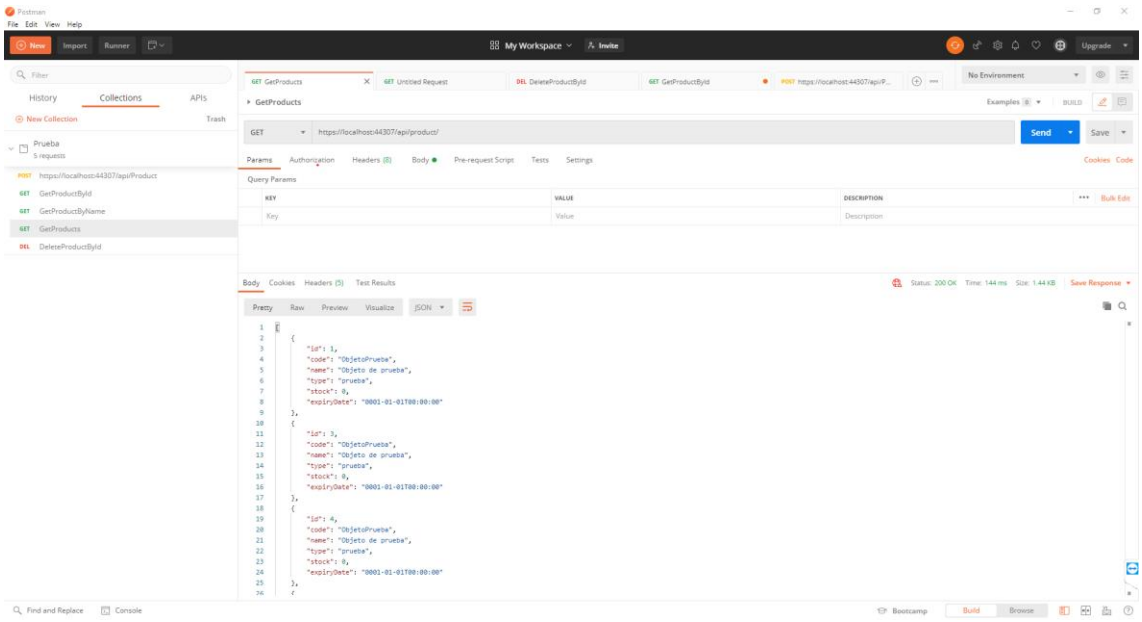
INIDICE

1 .- Instrucciones de ejecución

El código de la aplicación está alojado en un repositorio de GitHub, en la siguiente dirección: https://github.com/z-hdh/PruebaNivel

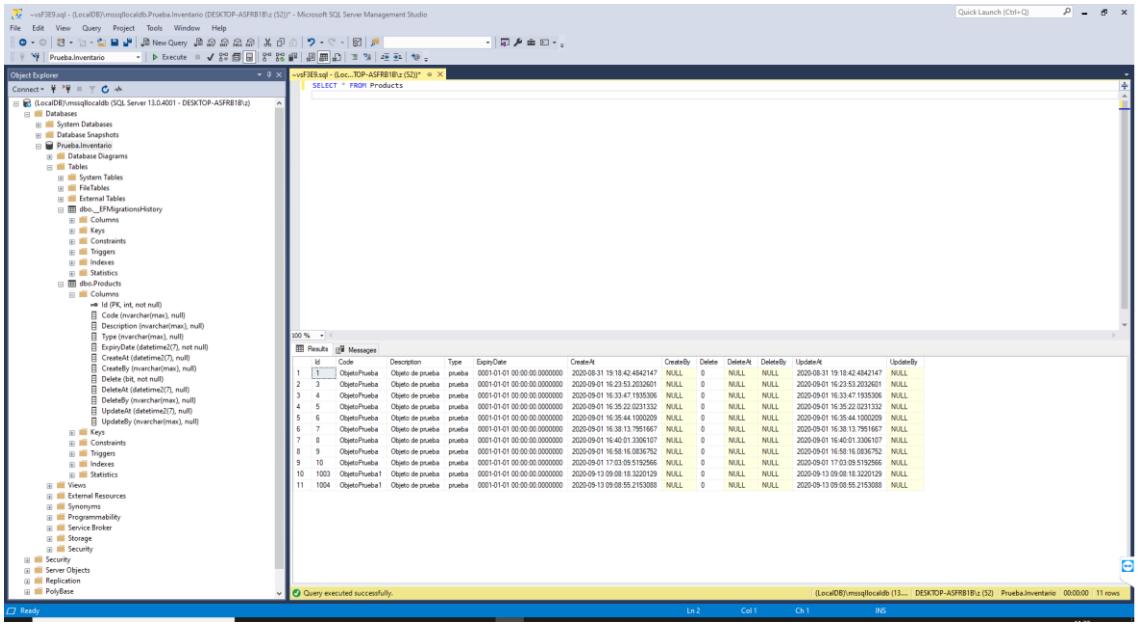
La aplicación se inicia como un API (sin cliente), para hacer pruebas he utilizado Postman.



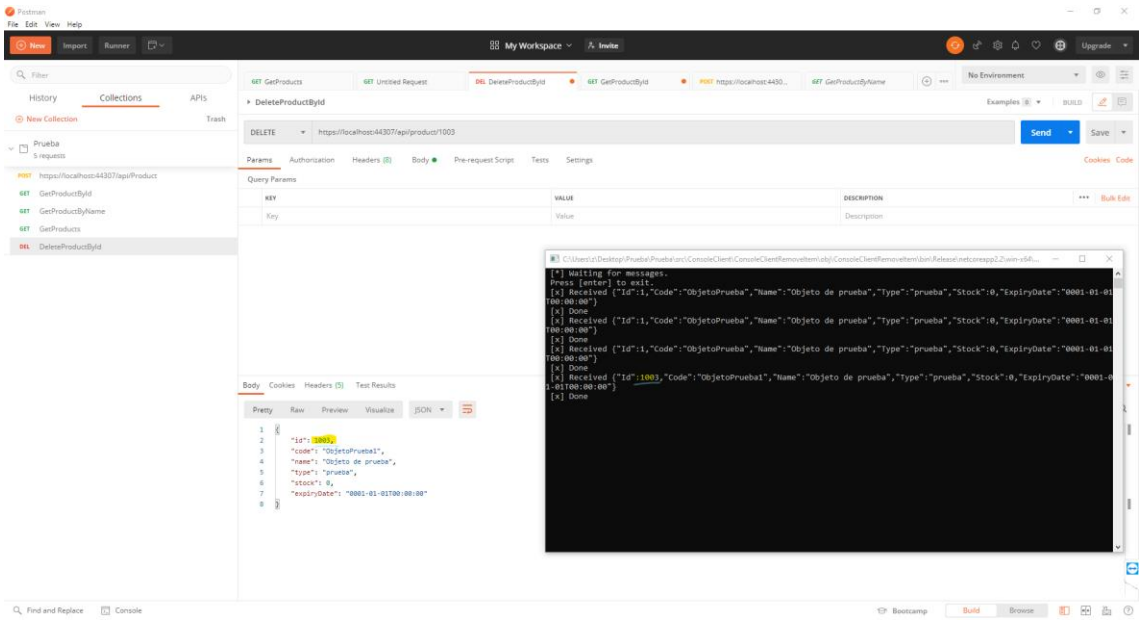
Se ha implementado funcionalidad para:

- Consultar todos los productos: https://localhost:44307/api/product/ (GET)
- Consultar productos por nombre: https://localhost:44307/api/product/***name/NombreProducto*** (GET)
- Consultar productos por Id: https://localhost:44307/api/Product/***Id*** (GET)
- Añadir productos: https://localhost:44307/api/Product (POST)
- Borrar productos: https://localhost:44307/api/product/***Id*** (DELETE)

Para el almacenamiento de datos he utilizado un base de datos local y Entity Framework (Code First). Tras la descarga de la aplicación, será **necesario hacer** una **migración inicial** para crear la base de datos.

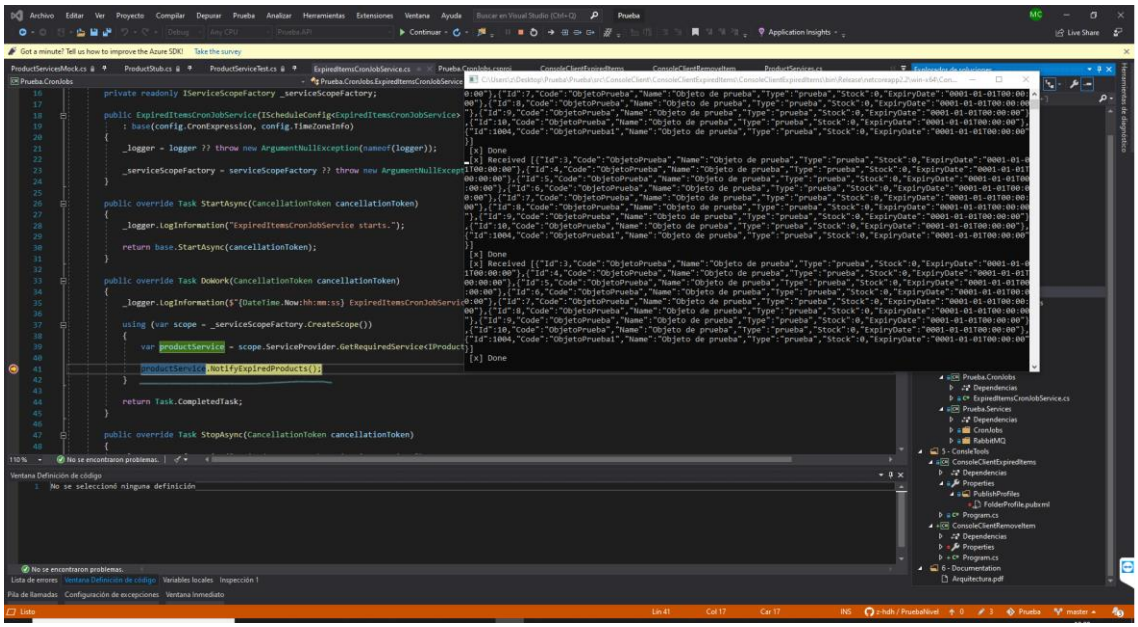


Para la notificación de cambios he utilizado RabbitMQ y una suscripción gratuita a https://www.cloudamqp.com/ para hacer las pruebas.



Par hacer pruebas, he creado una aplicación de consola (ConsoleClientRemoveItem) muy simple que está suscrita a la cola en la que se publican los mensajes. Esta aplicación se puede compilar y usar para hacer las pruebas.

Para implementar las tareas programadas, he utilizado **Cron** (para definir las expresiones que temporizan los procesos) y creado como en el caso anterior, una pequeña aplicación de consola para monitorizar lo que se publica en el bus (ConsoleClienteRemoveItem). Cada minuto se lanza un proceso que monta una lista con los productos caducados y la publica en el bus.



2.- Arquitectura

La arquitectura de la aplicación está basada en un DDD, dividido en las siguientes capas:

1 - WebAPI : Proyecto de WebApi que contiene los controladores que exponen los recursos y la configuración del proyecto.

2 - Application : La capa de aplicación contiene los DTOs, los servicios con la lógica de negocio y los helper que sirven de apoyo a los servicios, con lógica reutilizable.

3 – Domain : Esta capa contiene las entidades de negocio, las clases necesarias para implementar la auditoria por herencia y las interfaces de los repositorios.

4 – Infraestructure : Esta capa está divida en varios proyectos.

Data : En esta capa se encuentra la implementación del ORM (DbContext, migraciones, etc) y la implementación de los repositorios.

CrossCutting : En esta capa se encuentran las constantes, Excepciones y los métodos de extensión.

Mapping : En esta capa está la implementación de la librería para los mapeos entre entidades (AutoMapper) y los profiles que definen como se deben mapear los datos.

Services : En esta capa se encuentran los servicios que permiten implementar el bus de eventos y las tareas programadas.

5 – ConsoleTools : En esta carpeta se encuentran las aplicaciones de consola creadas para poder hacer pruebas. Estas aplicaciones están suscritas a las colas en las que se publican los mensajes de los requisitos 1.3 y 1.4. Estas aplicaciones se pueden compilar y lanzar los ejecutables para verificar que los mensajes se publican correctamente.

3.- Mejoras

Entre las posibles mejoras que se me ocurren a primera vista, estarían las siguientes:

1 – Sería necesario implementar un **sistema de autenticación**. Mi idea era implementar JWT, pero por falta de tiempo no me ha sido posible.

2 – El **control de excepciones** es muy pobre (esto es aplicable a todas las capas del proyecto). Desde mi punto de vista, sería necesario devolver en las llamadas al API, además del código correspondiente, información descriptiva sobre el error.

En las capas de servicios y repositorios se podrían crear excepciones personalizadas para los errores más comunes (en el caso de que no existan en el framework).

3 – En el caso del **bus de eventos**, para la prueba he creado una cuenta con un proveedor gratuito, que en ningún caso sería una solución factible para un entorno de producción.

4 – Sería recomendable **documentar el código** con comentarios XML. Esto además de permitir que el intelliSense de Visual Studio nos aporte información sobre los métodos, nos permitiría mediante herramientas externas crear una documentación de las clases del proyecto.

5 – Escribir test para toda la aplicación. Se ha creado algún test de ejemplo, pero por falta de tiempo ha sido imposible crearlos todos.

4.- Notas

Actualmente estoy trabajando en un proyecto que pretendemos desplegar en producción en unas dos semanas y tenemos una carga de trabajo bastante alta. Me hubiera gustado poder dedicarle más tiempo a la práctica, implementar algo más funcional y profundizar un poco más en la notificación entre procesos y las tareas programadas. Estos dos últimos puntos me han interesado especialmente ya que en mi día a día no trabajo con estas tecnologías.

Actualmente estoy trabajando con .net framework 4.5, en una aplicación con arquitectura MVC para la solución de front y DDD en back. Uno de los aspectos que me ha parecido más interesante de la oferta de trabajo es la posibilidad de trabajas con una arquitectura orientada a microservicios y un stack tecnológico moderno.