**Ejercicio 2**

Diseña, documenta e implementa una PoC (entregable y ejecutable en docker)de un sistema para poder visualizar en tiempo real la evolución de creación,modificación, parada y destrucción de Items.

Los Items podrán ser de tipo bebida, comida, salsas, especies. Cualquiera de los items podrá tener de manera aleatoria las siguientes

características:
- •Precisa nevera
- •no precisa nevera

Cualquiera de los items podrá tener una capacidad de:
- •100 gr
- •1000 gr

Cualquiera de los items podrá tener un envase de:
- •botella
- •caja

Cualquiera de los items tendrá:
- •Nombre
- •Identificador numérico único

Se deberá guardar:
- •El nombre del cliente que ha lanzado el comando de creación
- •Se deberá guardar el ID del item
- •Se deberá guardar la hora + timestamp de la operación
- •Se deberá guardar el estado
  - o WAITING: Está procesando una petición (creacion, eliminación, cambio, etc..)
  - o CREATED: creado ok
  - o DELETED: Está eliminado

Finalmente se deberá implementar un sistema de simulación que lanzando peticiones aleatorias a la API cree, elimine, modifique items, para que estos cambios se puedan monitorizar en tiempo real en el dashboard.

En resumen, el sistema deberá proveer:
•Una API restful para gestionar el recurso “item” sobre una base de datos. (Diseño Openapi 3)

•Una App para gestionar desde el backend el recurso item implementando la API.

•Una interficie gráfica para visualizar y analizar (drill down) en tiempo real los cambios en la colección de items desde el origen de datos.
•Una aplicación de test para lanzar la creación automática y aleatoria de items (CRUD completo) contra la API de N elementos en tiempo real.

**Solución:**

1. Creación de **api-service**
   Es una API RESTful para gestionar el recurso item sobre una base de datos H2.
   Esta api se suscribe un dashboard-service al cual le notificará cualquier operación de los items.

2. Creación de un **dashboard-service:**
   Servidor web que visualizará los cambios de las distintas operaciones de los items.
   Se encarga de recibir las notificaciones, en esta caso el **api-service** estará suscrito, y distribuirlas en los distintos clientes que estén visualizando los cambios.

3. Creación de **console-management-app**
   Aplicación en modo consola que gestiona los item (CRUD).
   Hace la llamadas al **apli-service**, para la gestión de estos.

4. Creación de **test-services**
   Aplicación en modo consola que para lanzar la creación automática y aleatoria de items haciendo llamada al **api-services**, para la gestión de estos.

**Entorno de desarrollo:**

**SO:** Ubuntu 22.04.4 LTS
**Lenguaje:** jdk 17
**IDE:** netbeans 22
**Maven:** apache-maven-3.9.8
**Docker:** Docker version 27.2.1, build 9e34c9b
**Orquestador:** docker-compose version 1.29.2
**Generador de código:** chatgpt 4

**Tecnologías aplicadas:**

**Base de datos:** H2
**Framework:** spring boot v3.3.3
**OpenAi:** springdoc 2.5.0
**JPA:** spring data-jpa v3.3.3
**Servidor web:** spring boot v3.3.3
**WebSocket:** spring websocket v.3.3.3
**Broker:** spring websocket v.3.3.3 (STOMP)

**Requisitos para la ejecución:**
Linux
Maven
jdk 17
docker
docker-compose

**Instalación y ejecución en linux:**

Desde una terminal descomprimir:

```
$ tar -xvcf ejercicio2.tgz
$ cd ejercicio2_v2
```

Para la instalación ejecutaremos un script que realizará un build de los 4 proyectos y lanzará el orquestador docker-compose para levantar los 2 microservicios. El orquestador creará 2 dockers, **api-service** y **dashboar-service**, y la red **vincle-network**

$ ./install.sh

En la url **http://localhost:8081/items** visualizaremos la lista de items en tiempo real.

En la url **http://localhost:8080/swagger-ui/index.html** visualizaremos el swagger del servicio api-services

En la url http://localhost:8080/v3/api-docs visualizaremos el swagger en formato JSON del servicio api-services

Para la ejecución de la **console-management-app** desde un terminal:
$ java -jar console-management-app/target/console-management-app-0.0.1-SNAPSHOT.jar

La consola nos mostrara las distintas operaciones que debemos realizar.

Para la ejecucioń del **test-servicea**
$ java -jar test-service/target/test-service-0.0.1-SNAPSHOT.jar
La consola nos pedirá el número operaciones que queremos realizar.

**Observaciones:**

Me hubiera gustado poder realizar un entregable mejor, pero dado que el tamaño de la aplicación compilada son unos 70Mb y la tengo que enviar por email, me parecía muy excesivo, por eso he creado scripts en bash para facilitar la instalación.

Debido a mis limitaciones de hardware esto sólo ha sido probado en un entorno localhost. Es probable que desde otro host no funcione la página **http://localhost:8081/items** de visualización en tiempo real de los items. El motivo podría ser que no pueda conectarse al **dashboard-services**. Otro motivo podría ser que los navegadores devolvieran errores de **CORS**. Existen soluciones para los 2 casos.

La base de datos he escogido H2 por simplificar el desarrollo. Podría haber utilizado en otro docker otra base de datos (mysql, mariadb, PostgreSQL, etc). La aplicación está implementada con JPA, con lo cual no debería suponer un problema al cambiar de bdd en caso de fuera necesario.

El diseño de la bdd, sólo he utilizado una tabla también para simplificar el desarrollo. Se puede mejorar creando tablas relacionales como, tipo de producto y tipo de envase.

Se podría mejora la aplicación utilizando otro brocker que no sea el spring como Kafka o rabbitmq. Siendo sincero, no estoy familiariza con estos 2 brokers.