Documentación

CONSIDERACIONES INICIALES:

- La documentación que se encontrará en este documento estará complementada con el swagger que expone la API. Se puede acceder a través del endpoint: */swagger-ui.html* (*http://localhost:8080/swagger-ui.html*)

- El código se encuentra versionado en: *https://github.com/alejandrofduran/demoparrilla.git*

CARACTERÍSTICAS DE LA API

La API implementa 1 función:

- **Método:** *GET*
- **Path:** */v1.0/parrilla*
- **Parámetros:** *"Directorio con los archivos a procesar", el mismo es obligatorio.*
- **Descripción:** *Dado el directorio, (Ej: C:/demo/csvs,) lee los archivos* ***product.csv, size.csv y stock.csv****, los procesa y devuelve los identificadores de los productos que cumplen con los requisitos para ser vendidos.*
- **Validaciones:** *El presente método valida que el parámetro "Directorio con los archivos a procesar", sea un directorio válido y que contenga los archivos mencionados.*

INTERPRETACIONES ASUMIDAS DEL ENUNCIADO

- Se interpreta y asume del enunciado que los campos *id, sequence* del archivo ***product.csv***, *id, productId* del archivo ***size.csv*** y *sizeId, quantity* del archivo ***stock.csv*** siempre serán enteros y que el dato indicado siempre es correcto
- También se interpreta que los campos *backSoon y special* sólo pueden contener los textos true o false.
- Se entiende que si a un producto sólo se le indican talla especial no estará disponible para la venta dado que no se le indicó ninguna talla normal. Se asume que al no poseer ningún producto de talla normal no se satisface la condición 2 del enunciado.
- Finalmente, se entiende que cada *sizeId* es único, es decir no pueden existir 2 productos distintos que tengan el mismo *sizeId****.***

ESTRUCTURAS DE DATOS UTILIZADAS EN EL ALGORITMO

Para la resolución del algoritmo se crearon 3 entidades idénticas (ProductDTO, SizeDTO y StockDTO) a los datos proporcionados en los archivos y una particular para el procesamiento de los productos (ProductDataDTO), que cuenta con los siguientes atributos: normalSizeOKToPublish, specialSizeOkToPublish y hasSpecialSizes todos booleanos.

Los datos leídos de cada archivo se guardaron en ***ArrayList*** utilizando los DTO creados, y se utilizó un ***HashMap<idProducto, ProductDataDTO>***, ya que como un producto tiene múltiples tallas es necesario buscar dicho producto múltiples veces. En particular esta implementación del ***HashMap*** tiene una complejidad de O(1) tanto para la búsqueda como para la inserción de datos.

La lista de StockDTO, es pasada a un ***HashSet*** filtrando las tallas que tienen 0 stock, debido a que es necesario buscar cada talla y de esta forma la búsqueda se realizara más eficientemente. Es importante destacar que el ***HashSet*** tiene, en el peor caso, una complejidad para la búsqueda de O(n). Sin embargo, la complejidad esperada es O(1). Adicionalmente, elimina los elementos duplicados de la lista si los hubiera.

Finalmente, la respuesta, con los productos que están en condiciones de ser pintados en la parrilla de los frontales web, es un ***ArrayList***. El contenido del mismo será concatenado con el método ***String.join*** para devolver la respuesta en el formato indicado en el enunciado.

COMPLEJIDAD TEMPORAL DEL ALGORITMO

Para el cálculo de orden de complejidad diremos que:

- `La letra “n” representa la cantidad las tallas`
- `La letra “m” representa la cantidad de productos`

En el peor caso la complejidad del algoritmo será, $O(2m) + O(3n) + O(n^2) + O(m \log(m))$ que es equivalente a $O(n^2)$, esto quiere decir que mi algoritmo tiene una complejidad cuadrática.

En el caso esperado la complejidad del algoritmo será, $O(2m) + O(4n) + O(m \log(m))$ que es equivalente a $O(m \log(m))$ esto quiere decir que mi algoritmo tiene una complejidad casi lineal.

Dada la forma en que fueron presentados los datos, no he encontrado una forma mas eficiente para la resolución del algoritmo. Pero considero que podría ser diferente la respuesta si tuviera la visión de un equipo para analizar el algoritmo y proponer soluciones alternativas.

HERRAMIENTAS UTILIZADAS PARA LA IMPLEMENTACIÓN.

- Framework: Spring boot 2.5.4
- Swagger2: 2.9.2
- Java 11

CONFIGURACIÓN

El proyecto está configurado para utilizar el puerto 8080. Para cambiar esta propiedad debe modificarse la siguiente variable en el archivo **application.properties: server.port=8080**

Para la ejecución de manera local (sin usar Spring Tool Suite), se utiliza mvn y para ejecutar la API se debe de correr el comando, **mvn spring-boot:run**, una vez ubicados en la raíz del proyecto.