Caso Práctico Backend Tools

Dado un listado de productos que se muestra en una categoría de camisetas se necesita implementar un algoritmo que permita ordenar ese listado en base a una serie de criterios de ordenación. Cada criterio de ordenación tendrá un peso asociado de manera que la puntuación de cada producto a ordenar vendrá dada por la suma ponderada de los criterios.

Los criterios de ordenación definidos serán el criterio de ventas por unidades y el criterio de ratio de stock, puede que a futuro se añadan nuevos criterios.
Nota: **A futuro podrían añadirse nuevos criterios**, por lo que la solución debe ser **abierta y extensible**.

El criterio de ventas por unidades dará una puntuación a cada producto basado en el número de unidades vendidas.

El criterio de ratio de stock dará una puntuación en función de las tallas que contengan stock en ese momento.

El listado de productos es el siguiente:

id	name	sales_units	stock
1	V-NECH BASIC SHIRT	100	S: 4 / M:9 / L:0
2	CONTRASTING FABRIC T-SHIRT	50	S: 35 / M:9 / L:9
3	RAISED PRINT T-SHIRT	80	S: 20 / M:2 / L:20
4	PLEATED T-SHIRT	3	S: 25 / M:30 / L:10
5	CONTRASTING LACE T-SHIRT	650	S: 0 / M:1 / L:0
6	SLOGAN T-SHIRT	20	S: 9 / M:2 / L:5

Requisitos funcionales:

- La funcionalidad debe exponerse mediante un **servicio REST**.
- El servicio debe **recibir los pesos por cada criterio** como entrada.
- Debe devolver el listado de productos ordenado según la puntuación total.

Recomendaciones técnicas y de calidad del desarrollo

1. Diseño y construcción del servicio

- Aplicar una **arquitectura hexagonal**, estructurando la solución mediante **módulos o paquetes claramente definidos**.
- Uso de una **nomenclatura clara y coherente** en clases, métodos y paquetes.
- **Orientación funcional** aprovechando características de **Java 17** (streams, records, sealed classes, etc.).
- Exposición del servicio usando **Swagger/OpenAPI** para facilitar su uso y documentación.

2. Calidad de código

- Aplicación de los principios **SOLID**.
- Buenas prácticas de **Clean Code**: código legible, mantenible, con funciones pequeñas y nombres expresivos.

3. Cobertura y validación

- **Pruebas unitarias** usando **JUnit 5** y **Mockito** (mocking de dependencias).
- **Pruebas de integración** que aseguren la correcta comunicación entre capas.
- Validación de que los resultados devueltos sean **correctos y reproducibles**.

Otros puntos importantes

- Uso el patrón que consideres más adecuado para implementar los criterios de forma desacoplada.
- Entrega empaquetada en un repositorio (GitHub/GitLab) con instrucciones para ejecutar la API.
- README claro que incluya ejemplos de llamada, datos esperados y cómo ejecutar los tests.