Objetivos y comentarios de la prueba:

- Poner en práctica los conocimientos adquiridos para mostrar una aproximación lo más real posible a un trabajo diario.
- Para ello se aconseja tener en cuenta todos y cada uno de los criterios de evaluación, así como el documento adjunto: “guidance_java_developer_Inditex”
- La prueba habrá que enfocarla desde un punto de vista orientado a la eficiencia, no solo debe funcionar, sino que debe funcionar con el menor número de recursos posibles (memoria, tiempo de respuesta, etc).
- Los criterios de evaluación son importantes, entre paréntesis encontraréis información muy valiosa para comprender los criterios y poder realizar un mejor desarrollo del test.
- La prueba en sí no es compleja, la complejidad radica en aplicar buenas prácticas, así como en centrar el diseño y desarrollo en la eficiencia y ser capaz de justificarlo en el readme.

Criterios de evaluación de la prueba:

- Un buen readme explicando tecnologías, arquitecturas y patrones utilizados, ejemplo de los tests, además de cosas que debe tener un readme (es una muy buena oportunidad y se valora altamente la explicación de las decisiones de diseño tomadas, ya que esto ayuda a comprender la lógica que has seguido para la toma de decisiones, pudiendo remarcar y demostrar así el seniority sobre la materia):
- Claridad del código, se tiene en cuenta lo fácil que resulte leerlo y entenderlo (utilizando principios SOLID):
- Número de alertas que tenga cuando se ejecute un linter como por ejemplo Sonar:
- Separación en capas y que cada capa contenga únicamente su responsabilidad (importante que el diseño sea escalable y modular):
- Uso correcto de la arquitectura elegida para la desarrollar la solución (la elección de arquitectura es libre, pero habría que explicar en el readme las ventajas y el por qué elegir ésta):
- Se valorará el desarrollo orientado a la eficiencia, no sólo debe funcionar, sino que las consultas a las bases de datos y/o algoritmia utilizados deben ser lo más eficientes posibles (aquí es importante elegir bien las estructuras de datos correctas, teniendo en cuenta operaciones como lectura, inserción o borrado de datos son más eficientes colas, hashmaps, ArrayList, LinkedList, treemap, etc):
- Uso de las herramientas que proporciona Github o similares para la gestión del código (utilizar nomenclatura estándar de commits, ramas, etc):
- Que pasen correctamente los tests unitarios y al menos un test de integración para ver el desarrollo del mismo:
- Uso de controllerAdvice y gestión de excepciones:
- Uso de jacoco u otra herramienta para monitorizar la cobertura de código de test:
- Que el ejercicio se pueda probar y cumpla exactamente con lo que se pide (se puede extender funcionalidad, pero nunca debe comprometer o modificar el funcionamiento principal):

Enunciado de la prueba:

- Realizar un microservicio en SpringBoot que tenga 3 endpoints:
    - 1. ejecute un algoritmo que enriquezca unos datos obtenidos a través de un API (2 endpoints), para posteriormente guardarlos en una base de datos en memoria H2.
    - 2. ejecute un algoritmo que enriquezca unos datos obtenidos a través de un API (2 endpoints) sin posibilidad de utilizar base de datos y los devuelva en la petición.
    - 3. GET de la base de datos en memoria H2.
- Por una parte tenemos “albums” y por otra las “photos”.
- Tenemos que obtener los “albums” en este endpoint: https://jsonplaceholder.typicode.com/albums
- Tenemos que obtener las “photos” en este endpoint: https://jsonplaceholder.typicode.com/photos
- Debemos enriquecer cada “álbum” para que tenga todas las “photos”.
- Realizar al menos un test unitario por funcionalidad.
- Realizar al menos un test de integración.
- IMPORTANTE: la eficiencia es lo más importante (elegir las estructuras de datos correctamente).