Los siguientes son los criterios de evaluación que se tendrán en cuenta durante la prueba técnica para aplicar como desarrollador Java en Castor:

Precisión: Respuestas acotadas, concretas y claras.

Fundamentación: Entendimiento de lo que se habla. Ser coherentes y consecuentes en todo el proceso de la prueba con argumentos que sustenten cada respuesta.

Calidad: Cumplir con los criterios de aceptación de cada prueba. Entregar lo que se solicita de la mejor forma posible.

Creatividad: La originalidad, formas nuevas y enfoques distintos en la presentación de lo solicitado se apreciará en la valoración final.

Oportunidad: Entregar en el tiempo convenido. Trabajar dentro de un *time-box* es muy importante cuando hablamos de principios ágiles.

El objetivo de esta prueba es evaluar tus habilidades como **desarrollador backend senior**, incluyendo diseño de APIs, persistencia de datos, microservicios, pruebas, calidad de código y despliegue en contenedores orquestados.

Contexto

Una empresa necesita un **sistema de gestión de facturación** para clientes corporativos. La aplicación backend debe exponer **APIs REST en formato JSON** para:

1. **Gestión de clientes**: CRUD completo.
2. **Gestión de facturas**: creación, listado y consulta de facturas por cliente.
3. **Validaciones de negocio**:
    - Una factura no puede crearse sin un cliente activo.
    - Los totales de las facturas deben calcularse correctamente, aplicando impuestos y descuentos.

Requerimientos técnicos

1. Lenguajes y frameworks

- Implementar el **servicio principal** en **Java con Spring Boot**.
- Crear un **microservicio en Python** que reciba los ítems de una factura (productos, precios y cantidades) y devuelva el cálculo de

impuestos/descuentos.

    - Este microservicio debe ser consumido desde el backend en Java.

2. Persistencia

- PostgreSQL: almacenar la información de clientes.
- Oracle (PL/SQL): almacenar las facturas.
- Debes implementar un **procedimiento almacenado en PL/SQL** que valide si el cliente existe antes de registrar una factura.

3. Servicios REST

- Todos los endpoints deben exponer y consumir datos en **JSON**.
- Ejemplos de endpoints:
    - `POST /clientes`
    - `GET /clientes/{id}`
    - `POST /facturas`
    - `GET /facturas/{id}`

4. Pruebas y calidad

- Implementar **pruebas unitarias** en Java usando **Mockito**.
- Configurar **JaCoCo** para generar reportes de cobertura.

- Integrar con SonarQube y mostrar el análisis de calidad de código.

5. Infraestructura

- Crear manifiestos de Kubernetes (YAML) para desplegar:
    - El backend en Spring Boot.
    - El microservicio en Python.
    - PostgreSQL y Oracle (pueden usarse imágenes para desarrollo).

6. Node.js

- Incluir un script en Node.js que actúe como cliente de prueba, enviando solicitudes a los endpoints principales y mostrando resultados por consola.

Entregables

1. Código fuente en repositorio (GitHub/GitLab).
2. Manifiestos de Kubernetes listos para levantar la solución.
3. Scripts SQL/PLSQL para inicializar las bases de datos.
4. README.md con:
    - Instrucciones de despliegue.
    - Ejemplo de uso de los endpoints.

    - Cómo correr las pruebas y ver reportes en SonarQube.

5. Reporte de cobertura JaCoCo y evidencia de integración con SonarQube.
6. Video explicando, donde se vea lo realizado y la cara del participante. Máximo de 10 minutos