

## PRUEBA TÉCNICA

### Backend Senior (Java + Spring Boot)

#### 1. Consideraciones generales

Obligatorio

- Utilizar Java en su última versión LTS (mínimo Java 17).
- Utilizar Spring Boot en su versión estable/LTS vigente.
- Gestión de dependencias y build mediante Maven.
- Persistencia en H2 en memoria.
- Entregar la solución en un repositorio Git (no es necesario publicarlo). Alternativamente, entregar un archivo comprimido con el repositorio completo.

Se valorará

- Aplicación de principios SOLID y diseño orientado a mantenimiento.
- Buenas prácticas de testing (TDD es un plus).
- Calidad de documentación y claridad de ejecución.

#### 2. Objetivo

Implementar una API REST para el mantenimiento (CRUD) de un catálogo de súper héroes. La API debe permitir la consulta, búsqueda, alta, modificación y eliminación de héroes, garantizando validaciones y un manejo de errores consistente.

#### 3. Requisitos funcionales

La API deberá exponer los siguientes endpoints bajo el prefijo:

Base path: /api/v1/heroes

##### 3.1 Consultar todos los héroes

- GET /api/v1/heroes
- Debe soportar paginación y ordenación mediante parámetros estándar (page, size, sort).

##### 3.2 Consultar un héroe por id

- GET /api/v1/heroes/{id}
- Si el recurso no existe, devolver 404 Not Found.

## PRUEBA TÉCNICA

### Backend Senior (Java + Spring Boot)

#### 3.3 Buscar héroes por coincidencia parcial en el nombre

- GET /api/v1/heroes/search?name={value}
- Debe devolver todos los héroes cuyo nombre contenga el valor indicado.
- La búsqueda debe ser case-insensitive.
- Si el parámetro name no es válido (vacío o insuficiente), devolver 400 Bad Request.

#### 3.4 Crear un héroe

- POST /api/v1/heroes
- Validar los campos de entrada.
- Si se crea correctamente, devolver 201 Created e incluir cabecera Location.

#### 3.5 Modificar un héroe

- PUT /api/v1/heroes/{id}
- Si el recurso no existe, devolver 404 Not Found.
- Debe aplicar validaciones equivalentes a las de creación.

#### 3.6 Eliminar un héroe

- DELETE /api/v1/heroes/{id}
- Si se elimina correctamente, devolver 204 No Content.

## 4. Modelo de datos

Definir una entidad Hero con, al menos, los siguientes atributos:

- id (Long, autogenerado)
- name (String, obligatorio, único) (index único, Non-Clustered )
- alias (String, opcional)
- universe (Enum: MARVEL, DC, OTHER)
- powerLevel (Integer, rango 1–100)
- active (Boolean, por defecto true)
- createdAt, updatedAt (auditoría)

## PRUEBA TÉCNICA

### Backend Senior (Java + Spring Boot)

#### Restricciones mínimas

- name obligatorio, sin espacios laterales, longitud razonable.
- Unicidad de name en base de datos (duplicados → 409 Conflict).
- powerLevel en rango permitido.

## 5. Requisitos técnicos

### 5.1 Persistencia

- Usar H2 in-memory.
- Se valorará el uso de Flyway o Liquibase para gestionar DDL y/o migraciones.

### 5.2 Estructura y arquitectura

- Separación por capas (mínimo):
  - Controller
  - Service
  - Repository
  - DTOs
  - Mapper
- No exponer entidades JPA directamente en los endpoints.

### 5.3 Gestión centralizada de excepciones

Implementar un manejador global (@ControllerAdvice) que devuelva errores con formato consistente, cubriendo como mínimo:

- 400 (validaciones/parámetros)
- 404 (recurso no encontrado)
- 409 (conflicto por duplicidad)
- 500 (error inesperado)

Se recomienda un modelo de error con campos como: timestamp, status, error, message, path.

## PRUEBA TÉCNICA

### Backend Senior (Java + Spring Boot)

#### 5.4 Documentación

- Documentar la API mediante OpenAPI/Swagger (p. ej. springdoc-openapi).

## 6. Pruebas automatizadas

### Obligatorio

- Incluir tests unitarios de al menos un servicio (JUnit 5).
- Deben cubrir, como mínimo:
  - Caso exitoso
  - Caso de recurso inexistente (404)
  - Caso de conflicto por duplicidad (409) si aplica

### Se valorará

- Tests de integración (@SpringBootTest + MockMvc/WebTestClient).
- Cobertura razonable sobre validaciones y manejo de errores.

## 7. Mejoras opcionales (valoradas)

- Cache en endpoints de lectura e invalidación tras escrituras.
- Dockerización (Dockerfile, opcional docker-compose).
- Seguridad (Basic Auth o JWT) con restricción de endpoints de escritura.

## 8. Entregable y criterios de evaluación

### Entregable

- Repositorio Git con:
  - Código fuente
  - Tests
  - README

### README mínimo esperado

- Requisitos de ejecución

## PRUEBA TÉCNICA

### Backend Senior (Java + Spring Boot)

- Cómo ejecutar la aplicación
- Cómo ejecutar tests
- Ruta de Swagger/OpenAPI
- Consideraciones relevantes (decisiones técnicas)

#### Criterios de evaluación

- Calidad de diseño y mantenibilidad (SOLID, separación de responsabilidades). 30%
- Pruebas automatizadas relevantes. 25%
- Manejo de errores consistente y orientado a cliente. 20%
- Correcta aplicación de validaciones. 15%
- Claridad de documentación y facilidad de ejecución. 10%