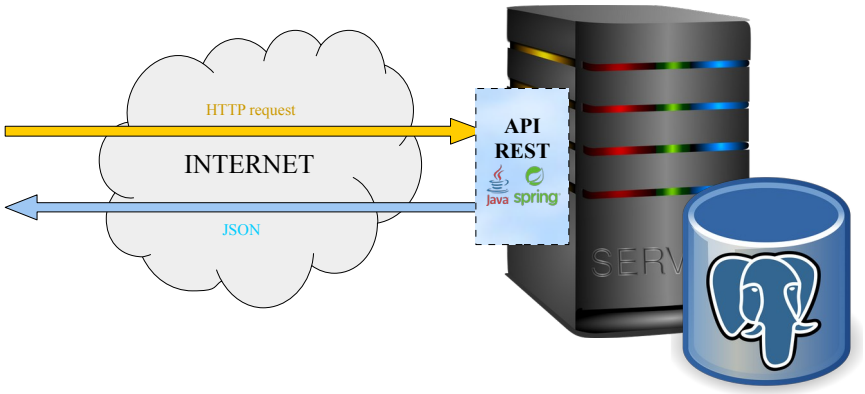
Tema 4

Proyecto API Frases Célebres

Objetivo

Desarrollar una API REST en Spring Boot que permita gestionar y consultar frases célebres, clasificadas por autor y categoría, persistidas en una base de datos relacional (PostgreSQL).

Este proyecto está orientado a poner en práctica lo aprendido sobre JPA, Hibernate, Spring Data JPA, relaciones entre entidades, validación y diseño REST.



1. Requisitos funcionales

1.1. Consultas

- Obtener la frase del día (la frase programada para una fecha concreta, por defecto, hoy).
- Listar autores y consultar un autor por id.
- Listar categorías y consultar una categoría por id.
- Listar frases y consultar una frase por id.
- Consultar frases por autor.
- Consultar frases por categoría.

1.2. Gestión del CRUD

- Crear, actualizar y eliminar autores.
- Crear, actualizar y eliminar categorías.
- Crear, actualizar y eliminar frases.

2. Modelo de datos

El diseño exacto es libre, pero debe cubrir las siguientes entidades y campos mínimos:

2.1 Entidad Autor

- id (PK, autogenerada)
- nombre (único, obligatorio)

- año de nacimiento (4 cifras, puede ser negativo para a.C.)
- año de fallecimiento (4 cifras, puede ser null en caso de que el autor siga vivo y negativo para a.C.)
- profesion (texto)

Relación: un autor puede tener muchas frases

2.2 Entidad Categoría

- id (PK, autogenerada)
- nombre (único, obligatorio)

Relación: una categoría puede tener muchas frases

2.3 Entidad Frase

- id (PK, autogenerada)
- texto (obligatorio)
- fechaProgramada (fecha en la que será la "frase del día")

3. Requisitos REST

A continuación se detallan los endpoints mínimos a desarrollar. Cualquier ampliación será bienvenida.

Todos los endpoints deben estar bajo el prefijo:

/api/v1

3.1 Autores

- GET /api/v1/autores
  Lista autores con paginación.
  - Query params: page y size
- GET /api/v1/autores/{id}
  Obtiene un autor por id.
- GET /api/v1/autores/{id}/frases
  Lista frases de un autor (con paginación).
- POST /api/v1/autores
  Crea un autor.
- PUT /api/v1/autores/{id}
  Actualiza un autor existente.

- DELETE /api/v1/autores/{id}

  Elimina un autor.

3.2 Categorías

- GET /api/v1/categorias
- GET /api/v1/categorias/{id}
- GET /api/v1/categorias/{id}/frases

  Lista frases de una categoría (con paginación).
- POST /api/v1/categorias
- PUT /api/v1/categorias/{id}
- DELETE /api/v1/categorias/{id}

3.3 Frases

- GET /api/v1/frases

  Lista frases (con paginación).
- GET /api/v1/frases/{id}

  Obtiene una frase por id.
- GET /api/v1/frases/dia

  Devuelve la frase programada para hoy.
- GET /api/v1/frases/dia?fecha=yyyy-MM-dd

  Devuelve la frase programada para esa fecha.
- POST /api/v1/frases

  Crea una frase.
- PUT /api/v1/frases/{id}

  Actualiza una frase.
- DELETE /api/v1/frases/{id}

  Elimina una frase.

4. Paginación

- page (por defecto 0)
- size (por defecto 10)

Ejemplos:

- /api/v1/autores?page=0&size=10
- /api/v1/frases?page=1&size=20

5. Validación y reglas de negocio

5.1 Validación

Aplicar anotaciones como:

- @NotBlank en nombres y texto de frase

- @Size para limitar longitudes razonables
- Validar año de nacimiento y año de fallecimiento como "enteros de 4 cifras" (admite negativos):
  Ejemplos válidos: 1990, -1350

5.2 Unicidad

- El nombre de la Categoria debe ser único
- Para una fecha dada, sólo puede existir una frase del día

6. DTOs y diseño de capas

La aplicación debe seguir una arquitectura por capas. La estructura de paquetes recomendada es la siguiente:

- config (configuración)
- controller (REST)
- service (lógica de negocio)
- repository (Spring Data JPA)
- model (JPA)
- dto (request/response)
- exception (gestión global de excepciones)

No se deben exponer entidades directamente en los controladores.

7. Gestión de errores

La aplicación debe implementar un manejador global de excepciones mediante `@RestControllerAdvice`, siguiendo un esquema similar al desarrollado en clase.

Los tipos de errores que deben gestionarse son:

- Errores de **validación** (@Valid)
  Cuando fallen las validaciones de los DTO:
  - Código HTTP: 400 Bad Request.

  Ejemplos:
  - Nombre vacío
  - Texto de frase demasiado corto
  - Año fuera de rango
- Errores de **integridad** de datos
  Cuando se viole una restricción de base de datos (por ejemplo valores únicos o campos obligatorios):
  - Código HTTP: 409 Conflict

  Casos típicos:
  - Autor con nombre duplicado
  - Categoría duplicada
  - Frase con referencias inexistentes

8. Autenticación

La API REST debe protegerse mediante Spring Security

8.1 Tipos de usuario

Se deben definir al menos dos roles:

- ADMIN. Podrá accedera a cualquiera de los endpoints (cualquier operación del CRUD)
- STANDARD. Solo podrá acceder a endpoints de consulta.

Cada usuario tendrá:

- id
- username (único)
- password (almacenada cifrada con Bcrypt)
- role

9. Conjunto de datos iniciales

Crear un archivo data.sql (puedes exportar las tablas de PostgreSQL), con al menos los siguientes datos:

- 3 Categorías
- 6 Autores
- 12 Frases
- Usuarios. Se debe crear al menos
  - Un usuario ADMIN
  - Un usuario STANDARD

10. Documentación

- Fichero README.md en la carpeta principal del proyecto explicando:
  - requisitos (Java, Gradle, PostgreSQL)
  - puesta en marcha
  - 3 capturas de pantalla con ejemplos de llamadas (a tu elección)
  - usuarios y contraseñas de ADMIN y STANDARD
- Swagger/OpenAPI

11. Entregables

- Documento de texto con la URL al repositorio del proyecto
- Archivo data.sql con el conjunto de datos iniciales para poder probar la aplicación

12. Criterios de evaluación

- Modelos y persistencia (25%)

  Relaciones correctas, restricciones y diseño de entidades.

- API REST y endpoints (30%)
  Convenciones REST, versionado, códigos HTTP correctos.
- Servicios, DTOs y validación (20%)
  Capas, reglas de negocio y validación.
- Seguridad con Spring Security (15%)
  Definición de roles, autenticación y gestión de permisos
- Manejo de errores + documentación (10%)
  @RestControllerAdvice y README.md claro

13. Ampliaciones

Posibilidad de conseguir hasta un máximo de **+1 punto**

- **Frases más valoradas** (hasta +1)
  Añadir entidad para "likes/dislikes" y endpoints:
  - POST /api/v1/frases/{id}/valoraciones (like/dislike)
  - GET /api/v1/frases/top?limit=10
- **Búsqueda por texto** (hasta +0,25)
  Permitir buscar frases que contengan una palabra o cadena:
  - GET /api/v1/frases/search?texto=vida
- **Contador de visualizaciones** (hasta +0,50)
  Cada vez que se consulta una frase incrementar un campo visitas.