Sistema de Gestión de Libros - Prueba Técnica MegaSoft

Sistema integral para administración de libros, construido sobre una arquitectura en capas utilizando Java 8, Spring Boot 2.7.18 y una base de datos H2 en memoria. El objetivo es demostrar buenas prácticas de ingeniería, manejo de patrones de diseño, uso de la Stream API y un enfoque full-stack.

El proyecto implementa un **CRUD completo** para la gestión de libros, habilitando operaciones de creación, consultas avanzadas, actualización y eliminación.

Se estructura bajo una arquitectura escalable (DAO → Service → Controller) y expone tanto un **frontend interactivo** como una **API RESTful**.
El sistema incorpora conceptos clave como:

- Optional y manejo seguro de nullability
- Lambda expressions y Method References
- Consultas enriquecidas mediante Stream API
- Excepciones personalizadas orientadas a gobernanza del dominio
- Base de datos en memoria para aprovisionamiento ágil del entorno

Características Principales

- **CRUD completo** de libros
- **Búsquedas avanzadas** con Stream API
- **Arquitectura empresarial en capas**
- **Base de datos H2** (in-memory) para despliegue local
- **Interfaz web dinámica** basada en Fetch API (AJAX)
- **Excepciones personalizadas** (BookNotFoundException)
- **Uso de Optional, lambdas y method references**

Instalación y Ejecución

1. Clonar el repositorio
2. Abrir el proyecto con **Spring Tools Suite (STS)** o Eclipse
3. Ejecutar la clase principal `DemoApplication`
   → Run As → Spring Boot App

Acceso a la Aplicación

Una vez iniciada la aplicación:

- **Interfaz Web**: http://localhost:8080

Estructura del Proyecto

```
library-management/
├── src/
│   ├── main/
│   │   ├── java/com/library/
│   │   │   ├── LibraryApplication.java        # Clase principal
│   │   │   ├── model/
│   │   │   │   └── Book.java                  # Entidad Book
│   │   │   ├── dao/
│   │   │   │   ├── BookDAO.java               # Interfaz DAO
│   │   │   │   └── BookDAOImpl.java           # Implementación DAO
│   │   │   ├── service/
│   │   │   │   ├── BookService.java           # Interfaz Service
│   │   │   │   └── BookServiceImpl.java       # Implementación Service
│   │   │   ├── controller/
│   │   │   │   ├── BookController.java        # REST Controller
│   │   │   │   └── WebController.java         # Web Controller
│   │   │   └── exception/
│   │   │       └── BookNotFoundException.java  # Excepción personalizada
│   │   └── resources/
│   │       ├── application.properties         # Configuración
│   │       ├── schema.sql                     # Esquema de BD
│   │       ├── data.sql                       # Datos iniciales
│   │       └── static/
│   │           ├── index.html                 # Interfaz principal
│   │           ├── app.js                     # Lógica frontend
│   │           └── styles.css                 # Estilos CSS
│   └── test/
│       └── java/com/library/
│           └── BookServiceTest.java           # Pruebas unitarias
```

```
├── build.gradle                    # Configuración Gradle
├── settings.gradle                 # Configuración proyecto
└── README.md                       # Este archivo
```

API REST – Endpoints CRUD

Método	Endpoint	Descripción
POST	/api/books	Crear un nuevo libro
GET	/api/books/{id}	Obtener libro por ID
PUT	/api/books/{id}	Actualizar libro
DELETE	/api/books/{id}	Eliminar libro
GET	`/api/books`	Listar todos los libros

Búsquedas Avanzadas (Stream API)

Método	Endpoint	Descripción
GET	/api/books/search/title?q={query}	Buscar por título (case-insensitive)
GET	/api/books/search/author?author={name}	Buscar por autor exacto
GET	/api/books/search/price-range?min={min}&max={max}	Filtrar por rango de precios
GET	/api/books/sorted/publication-date	Ordenar por fecha de publicación

Arquitectura en Capas

1. Capa de Modelo

Define la entidad Book con anotaciones JPA.

2. Capa DAO

Manejo transaccional y acceso a datos vía `EntityManager`.

3. Capa de Servicio

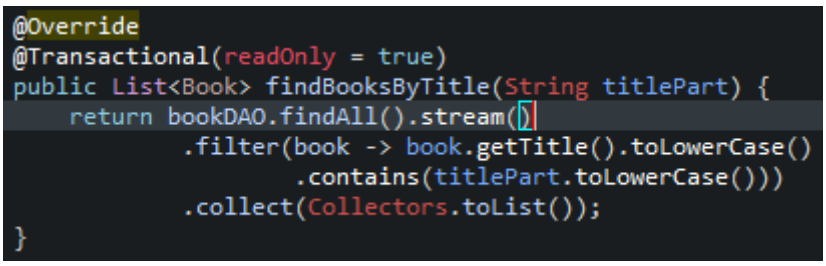
Lógica de negocio, validaciones y consultas enriquecidas.

4. Capa de Controlador

Exposición de REST API + controlador web para UI.

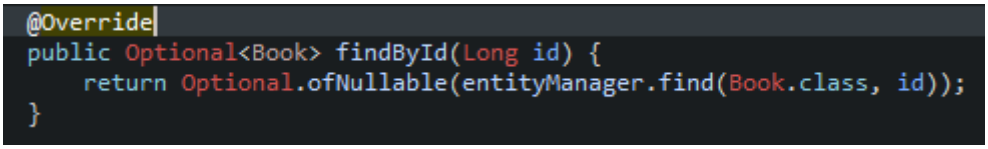
Uso de Stream API, Optional y Method References

1. Stream API ejemplo de uso



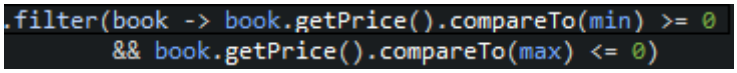
```
@Override
@Transactional(readOnly = true)
public List<Book> findBooksByTitle(String titlePart) {
    return bookDAO.findAll().stream()
            .filter(book -> book.getTitle().toLowerCase()
                    .contains(titlePart.toLowerCase()))
            .collect(Collectors.toList());
}
```

2. Optional



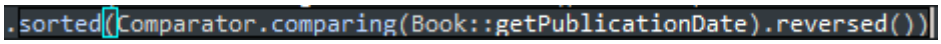
```
@Override
public Optional<Book> findById(Long id) {
    return Optional.ofNullable(entityManager.find(Book.class, id));
}
```

3. Lambda Expressions



```
.filter(book -> book.getPrice().compareTo(min) >= 0
        && book.getPrice().compareTo(max) <= 0)
```

4. Method References



```
.sorted(Comparator.comparing(Book::getPublicationDate).reversed())
```

BookNotFoundException

Excepción personalizada lanzada cuando:

- Se intenta obtener un libro que no existe
- Se intenta actualizar un libro inexistente
- Se intenta eliminar un libro inexistent

Frontend

- **Fetch API**: Para llamadas AJAX a la REST API
- **Vanilla JavaScript**: Sin dependencias adicionales
- **DOM Manipulation**: Actualización dinámica del contenido

Diagrama Lógico de la Arquitectura