# Documentación del Proyecto Spring Boot: API de Libros

## Tabla de Contenidos

1. **Introducción**
2. **Requisitos**
3. **Descripción del Proyecto**
4. **Configuración del Proyecto**
5. **Estructura del Proyecto**
6. **Correr el Proyecto**
7. **Endpoint de la API**
8. **Consumiendo la API desde JavaScript**
9. **Notas Adicionales**

## 1. Introducción

Este proyecto es una aplicación Spring Boot que expone una API RESTful para gestionar una lista de libros. La API permite crear, leer, actualizar y eliminar libros. Esta documentación cubre todos los aspectos necesarios para ejecutar, configurar y consumir la API.

## 2. Requisitos

Para ejecutar el proyecto, necesitas lo siguiente:

* JDK 8 o superior.
* Maven o Gradle para la gestión de dependencias.
* Una aplicación de terminal o IDE como IntelliJ IDEA, Eclipse, etc.
* Un entorno de ejecución de Java como Spring Boot.

## 3. Descripción del Proyecto

El proyecto spring-boot-books-api consiste en una aplicación Spring Boot que maneja una lista de libros. Los datos de los libros están almacenados en memoria y la API expone los siguientes endpoints:

* **GET /api/books**: Obtiene todos los libros.
* **GET /api/books/{id}**: Obtiene un libro por su ID.
* **POST /api/books**: Crea un nuevo libro.
* **PUT /api/books/{id}**: Actualiza un libro existente.
* **DELETE /api/books/{id}**: Elimina un libro por su ID.

## 4. Configuración del Proyecto

Para correr el proyecto localmente, sigue estos pasos:

1. **Clonar el proyecto desde GitHub**:

bash

Copiar código

git clone https://github.com/tu-usuario/spring-boot-books-api.git

1. **Importar el proyecto en tu IDE**:
   * Importa el proyecto como un proyecto Maven o Gradle en tu IDE.
2. **Configuración de la base de datos (opcional)**:
   * Si deseas integrar la API con una base de datos, configura los detalles de conexión en application.properties o application.yml. Aquí hay un ejemplo básico para conectar a una base de datos H2 en memoria:

properties

Copiar código

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

1. **Ejecutar la aplicación**:
   * Corre la aplicación desde tu IDE haciendo clic en el botón Run o Debug.
   * O si prefieres usar la línea de comandos:

arduino

Copiar código

mvn spring-boot:run

## 5. Estructura del Proyecto

La estructura básica del proyecto se ve así:

lua

Copiar código

|-- src

| |-- main

| | |-- java

| | | `-- com

| | | `-- example

| | | `-- books

| | | |-- Book.java

| | | |-- BookController.java

| | | |-- BookRepository.java

| | | `-- BookService.java

| | `-- resources

| | |-- application.properties

| `-- test

| `-- java

| `-- com

| `-- example

| `-- books

| `-- BookControllerTest.java

|-- pom.xml (Maven) / build.gradle (Gradle)

* **Book.java**: Representa la entidad Book con propiedades como id, title, author, y category.
* **BookController.java**: Exposa los endpoints RESTful para la API.
* **BookRepository.java**: Define los métodos CRUD para la entidad Book.
* **BookService.java**: Implementa la lógica de negocio para la gestión de libros.
* **application.properties**: Configuración de la aplicación (base de datos, puertos, logging, etc.).

## 6. Correr el Proyecto

Para correr la aplicación:

1. Abre tu terminal.
2. Navega al directorio del proyecto.
3. Ejecuta el comando Maven:

arduino

Copiar código

mvn spring-boot:run

1. La aplicación debería estar disponible en http://localhost:8080.

## 7. Endpoint de la API

### Endpoints disponibles:

* **GET /api/books**: Devuelve una lista de todos los libros.
* **GET /api/books/{id}**: Devuelve un libro específico basado en su ID.
* **POST /api/books**: Crea un nuevo libro.
* **PUT /api/books/{id}**: Actualiza un libro existente basado en su ID.
* **DELETE /api/books/{id}**: Elimina un libro basado en su ID.

### Ejemplos de uso:

1. **Obtener todos los libros**:

bash

Copiar código

GET http://localhost:8080/api/books

* + Respuesta:

json

Copiar código

[

{

"id": 1,

"title": "Cien años de soledad",

"author": "Gabriel García Márquez",

"category": "Ficción"

},

{

"id": 2,

"title": "1984",

"author": "George Orwell",

"category": "Distopía"

},

{

"id": 3,

"title": "El principito",

"author": "Antoine de Saint-Exupéry",

"category": "Fantasía"

}

]

1. **Obtener un libro por ID**:

bash

Copiar código

GET http://localhost:8080/api/books/1

* + Respuesta:

json

Copiar código

{

"id": 1,

"title": "Cien años de soledad",

"author": "Gabriel García Márquez",

"category": "Ficción"

}

1. **Crear un nuevo libro**:

bash

Copiar código

POST http://localhost:8080/api/books

* + Cuerpo de la solicitud (JSON):

json

Copiar código

{

"title": "El Señor de los Anillos",

"author": "J.R.R. Tolkien",

"category": "Fantástica"

}

* + Respuesta:

json

Copiar código

{

"id": 4,

"title": "El Señor de los Anillos",

"author": "J.R.R. Tolkien",

"category": "Fantástica"

}

1. **Actualizar un libro**:

bash

Copiar código

PUT http://localhost:8080/api/books/1

* + Cuerpo de la solicitud (JSON):

json

Copiar código

{

"title": "Cien años de soledad (edición revisada)",

"author": "Gabriel García Márquez",

"category": "Ficción"

}

* + Respuesta:

json

Copiar código

{

"id": 1,

"title": "Cien años de soledad (edición revisada)",

"author": "Gabriel García Márquez",

"category": "Ficción"

}

1. **Eliminar un libro**:

bash

Copiar código

DELETE http://localhost:8080/api/books/1

* + Respuesta:

json

Copiar código

{

"message": "Book with id 1 deleted successfully"

}

## 8. Consumir la API desde JavaScript

Para consumir la API desde un cliente JavaScript, puedes utilizar la función fetch. Aquí hay un ejemplo básico:

html

Copiar código

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Consultar API con JavaScript</title>

<script>

function consultarAPI() {

const url = 'http://localhost:8080/api/books'; // URL de la API

fetch(url)

.then(response => response.json())

.then(data => {

const resultadoDiv = document.getElementById('resultado');

resultadoDiv.innerHTML = ''; // Limpiar contenido anterior

data.forEach(book => {

resultadoDiv.innerHTML += `

<div class="book">

<h3>Título:</h3>

<p>${book.title}</p>

<h3>Categoría:</h3>

<p>${book.category}</p>

<h3>Autor:</h3>

<p>${book.author}</p>

</div>

`;

});

})

.catch(error => {

console.error('Error al consultar la API:', error);

document.getElementById('resultado').innerHTML = 'Hubo un error al consultar la API.';

});

}

</script>

</head>

<body>

<h1>Consultar API</h1>

<button onclick="consultarAPI()">Consultar API</button>

<div id="resultado">

<!-- Aquí se mostrará el contenido de la API -->

</div>

</body>

</html>

## 9. Notas Adicionales

1. **Depuración**:
   * Para depurar la aplicación en un entorno de desarrollo, puedes configurar tu IDE para usar los puntos de interrupción y las opciones de depuración específicas.
2. **Seguridad**:
   * Para mejorar la seguridad, considera usar autenticación y autorización en tus endpoints RESTful utilizando JWT, OAuth, o Basic Authentication.
3. **Pruebas**:
   * Puedes realizar pruebas automáticas para tus endpoints utilizando herramientas como Postman, Swagger UI, o escribiendo tests con JUnit y Mockito.
4. **Integración continua y despliegue**:
   * Para implementar la aplicación en producción, puedes utilizar herramientas como Jenkins o GitHub Actions para la integración continua y el despliegue.