# Spring Boot Project: Advanced Library Management System

This project is designed to challenge students with a more advanced backend implementation in Spring Boot. The goal is to manage a library system with features like book and author management, borrowing functionality, and enhanced data integrity through validations.

## Project Objective

Build a RESTful API for a Library Management System that manages:

1. **Books** (CRUD operations, validations, search by title, author, or category, with pagination).
2. **Authors** (CRUD operations, search by name or nationality, and duplicate prevention).
3. **Borrow Records** (track borrowing/returning of books, with checks for availability and validation of borrow/return dates).

---

## Project Requirements

- **Backend Framework**: Spring Boot
- **Database**: PostgreSQL
- **JDK Version**: Java 17+
- **Project Structure**: Clearly defined layers (Controller, Service, Repository, Exception Handling).

---

## Project Features

1. **Books**:
    - **Attributes**: id, title, category, publicationDate, authorId.
    - **Validations**:
        - Prevent duplicate entries of the same title and authorId.
        - title and category fields must not be blank.
    - **Search Features**:
        - By title (case-insensitive partial match).
        - By author name (join with the author table).
        - By category (exact match).
    - **Pagination**:
        - Add pagination support to fetch books in batches.
    - **Endpoints**:
        - GET /api/books (with pagination).
        - POST /api/books (create a new book).
        - PUT /api/books/{id} (update book details).
        - DELETE /api/books/{id} (delete a book).
        - GET /api/books/search (search by title, author, or category).
2. **Authors**:
    - **Attributes**: id, name, nationality.
    - **Validations**:
        - Prevent duplicate entries of the same name (case-insensitive).
        - name must not be blank.
    - **Search Features**:
        - By name (case-insensitive partial match).
        - By nationality (case-insensitive exact match).
    - **Endpoints**:
        - GET /api/authors.
        - POST /api/authors (create a new author).
        - PUT /api/authors/{id} (update author details).

        - DELETE /api/authors/{id} (delete an author).
        - GET /api/authors/search (search by name or nationality).
3. **Borrow Records**:
    - **Attributes**: id, bookId, borrowerName, borrowDate, returnDate, isReturned.
    - **Validations**:
        - Prevent borrowing a book that is already borrowed and not returned.
        - Ensure returnDate is after borrowDate.
        - borrowerName must not be blank.
    - **Endpoints**:
        - GET /api/borrow-records (fetch all borrow records).
        - POST /api/borrow-records (create a new borrow record).
        - PUT /api/borrow-records/{id} (mark a book as returned).
        - GET /api/borrow-records/search (search records by borrower name or book title).
4. **Custom Responses**:
    - Use ResponseEntity to return:
        - HTTP 200 for successful operations with data.
        - HTTP 404 when data is not found (e.g., book or author not found).
        - HTTP 400 for bad requests (e.g., validation failures, duplicate entries).
        - HTTP 500 for unexpected errors.
5. **Exception Handling**:
    - Create a GlobalExceptionHandler using @ControllerAdvice.
    - Handle exceptions like:
        - DuplicateEntryException: Thrown when attempting to insert duplicate data.
        - ValidationException: Thrown when validations fail.
        - EntityNotFoundException: Thrown when a requested entity is not found.

---

**Class and Layer Descriptions**

**1. Model Layer**

1. **Book**:
    - Fields: id, title, category, publicationDate, authorId.
    - Relationships: Foreign key authorId referencing the Author table.
2. **Author**:
    - Fields: id, name, nationality.
    - Relationships: None (authors are independent).
3. **BorrowRecord**:
    - Fields: id, bookId, borrowerName, borrowDate, returnDate, isReturned.
    - Relationships: Foreign key bookId referencing the Book table.

**2. Repository Layer**

- Create JpaRepository interfaces for each entity (BookRepository, AuthorRepository, BorrowRecordRepository).
- Add custom query methods:
    - findByTitleContainingIgnoreCase(String title) (BookRepository).
    - findByNameContainingIgnoreCase(String name) (AuthorRepository).
    - findByBorrowerNameContainingIgnoreCase(String borrowerName) (BorrowRecordRepository).

**3. Service Layer**

1. **BookService**:
    - Methods:
        - List<Book> getAllBooks(Pageable pageable): Fetch paginated books.
        - Book getBookById(Long id): Fetch book by ID.
        - Book addBook(Book book): Add a new book with duplicate checks.

        - Book updateBook(Long id, Book bookDetails): Update book details.
        - void deleteBook(Long id): Delete a book by ID.
        - List<Book> searchBooks(String title, String authorName, String category): Search books.
2. **AuthorService**:
    - Methods:
        - List<Author> getAllAuthors(): Fetch all authors.
        - Author getAuthorById(Long id): Fetch author by ID.
        - Author addAuthor(Author author): Add a new author with duplicate checks.
        - Author updateAuthor(Long id, Author authorDetails): Update author details.
        - void deleteAuthor(Long id): Delete an author by ID.
        - List<Author> searchAuthors(String name, String nationality): Search authors.
3. **BorrowRecordService**:
    - Methods:
        - List<BorrowRecord> getAllBorrowRecords(): Fetch all borrow records.
        - BorrowRecord borrowBook(BorrowRecord record): Add a new borrow record.
        - BorrowRecord returnBook(Long id, LocalDate returnDate): Mark a book as returned.
        - List<BorrowRecord> searchRecords(String borrowerName, String bookTitle): Search borrow records.

**4. Controller Layer**

1. **BookController**:
    - Endpoints for book management (as detailed above).
    - Return appropriate ResponseEntity responses.
2. **AuthorController**:
    - Endpoints for author management.
    - Handle validation errors with meaningful messages.
3. **BorrowRecordController**:
    - Endpoints for managing borrow records.
    - Check book availability before allowing borrowing.

**5. Exception Handling**

1. **CustomException**: Base exception class.
2. **DuplicateEntryException**: Thrown for duplicate entries.
3. **GlobalExceptionHandler**:
    - Use @ControllerAdvice to handle exceptions globally.
    - Return structured JSON responses for errors, including:
        - timestamp
        - status (HTTP status code)
        - message (error description)

---

**What Students Will Learn**

1. Designing a multi-entity backend system with relationships.
2. Implementing validations to ensure data integrity.
3. Writing meaningful custom error messages and responses.
4. Managing exceptions at a global level for a consistent API.
5. Leveraging Spring Data JPA for database operations.
6. Writing clean and modular code using service and repository layers.