A library

Entities:

Book	
ID	Integer
Title	String
authorID	Integer
Category	String
publishedDate	Date
isbn	String

Author	
ID	Integer
name	String
birthDate	Date
nationality	String

User	
ID	Integer
name	String
email	String
password	String
role	Enum (LIBRARIAN, MEMBER)

Borrowing	
id	Integer
bookId	Integer
userId	Integer
borrowDate	Date
returnDate	Date (nullable)

Functional requirements

1. The system must allow users to add, update, delete, and retrieve books.
2. Each book must have attributes like title, author, category, published date, and ISBN.
3. The system must support searching and filtering books by title, author, category, and published date.
4. The system must allow users to view book details by providing the book's ID.

5. The system must allow users to add, update, delete, and retrieve authors.

6. Each author must have attributes like name, birth date, and nationality.
7. The system must support searching and filtering authors by name and nationality.
8. The system must allow users to view author details by providing the author's ID.

9. The system must allow users to borrow and return books.
10. The system must track the borrowing history of each book, including borrower details and borrowing/return dates.
11. The system must prevent borrowing if the book is currently unavailable.

12. The system must allow user registration and authentication.

Non-Functional Requirements

1. The system must handle up to 100 concurrent users with minimal latency.

2. The system should efficiently manage large datasets, supporting up to 1 million books without performance degradation.

3. The system must enforce role-based access control, ensuring only authorized users can perform specific actions

4. The system must implement OAuth2 for user authentication and authorization.

5. The codebase must follow clean code principles and be well-documented.

REST API Design

Books Management

GET /books

Description: Retrieve a list of books.

Query Parameters:

1. title: Filter by book title.

2. authorId: Filter by author ID.

3. category: Filter by category.

4. publishedDate: Filter by published date (supports ranges, e.g., publishedDate[gte]=2020-01-01&publishedDate[lte]=2023-01-01).

5. page: Page number for pagination.

6. size: Number of records per page.

7. sort: Sort by fields (e.g., title,asc).

Response: Paginated list of books.

POST /books

Description: Create a new book.

Request Body:

```
{
"title": "Example Book",
"authorId": "123e4567-e89b-12d3-a456-426614174000",
"category": "Fiction",
"publishedDate": "2024-01-01",
"isbn": "978-3-16-148410-0"
}
```

Response: 201 Created, with the location of the newly created book.

GET /books/{id}

Description: Retrieve details of a specific book by its ID.

Response: Details of the book.

PUT /books/{id}

Description: Update an existing book.

Request Body: Same as POST /books.

Response: 200 OK, with updated book details.

DELETE /books/{id}

Description: Delete a specific book by its ID.

Response: 204 No Content.

Authors Management

GET /authors

Description: Retrieve a list of authors.

Query Parameters:

1. name: Filter by author name.

2. nationality: Filter by nationality.

3. page: Page number for pagination.

4. size: Number of records per page.

5. sort: Sort by fields (e.g., name,asc).

Response: Paginated list of authors.

POST /authors

Description: Create a new author.

Request Body:

```
{
"name": "Jack Dee",
"birthDate": "1978-08-13",
 "nationality": "American"
}
```

Response: 201 Created, with the location of the newly created author.

GET /authors/{id}

Description: Retrieve details of a specific author by their ID.

Response: Details of the author.

PUT /authors/{id}

Description: Update an existing author.

Request Body: Same as POST /authors.

Response: 200 OK, with updated author details.

DELETE /authors/{id}

Description: Delete a specific author by their ID.

Response: 204 No Content.

Borrowing Management

POST /borrowings

Description: Borrow a book.

Request Body:

```
{

"bookId": "123e4567-e89b-12d3-a456-426614174000",

"userId": "987e6543-e21b-34f3-a789-426614174111"

}
```

Response: 201 Created, with borrowing details.

Error: 409 Conflict if the book is already borrowed.

PUT /borrowings/{id}/return

Description: Return a borrowed book.

Response: 200 OK, with updated borrowing details.

GET /borrowings

Description: Retrieve borrowing history.

Query Parameters:

1. userId: Filter by user ID.

2. bookId: Filter by book ID.

3. page: Page number for pagination.

4. size: Number of records per page.

5. sort: Sort by fields (e.g., borrowDate,desc).

Response: Paginated list of borrowing records.

User Management

POST /users

Description: Register a new user.

Request Body:

{

"name": "Jane Rose",

"email": "jane.rose@gmail.com",

"password": "pass_jane_rose",

"role": "MEMBER"

}

Response: 201 Created, with user details.

POST /auth/login

Description: Authenticate a user and generate an OAuth2 token.

Request Body:

{

"email": " jane.rose@gmail.com ",

"password": "pass_jane_rose"

}

Response: 200 OK, with an access token.

Security

OAuth2 Authentication

All endpoints (except /auth/login and /users) require an OAuth2 token.

Role-based access control:

1. LIBRARIAN can manage books, authors, and borrowing records.
2. MEMBER can borrow/return books and view their own borrowing history.

Caching

GET /books, **GET /authors**, and **GET /borrowings** should implement caching to improve performance.

Cache invalidation must occur on create/update/delete operations.

Pagination and Sorting

Pagination: Use page and size query parameters.

Sorting: Use sort query parameter, e.g., ?sort=title,asc.

Error Handling

Meaningful HTTP status codes:

200 OK for successful operations.

201 Created for successful creation.

204 No Content for successful deletion.

400 Bad Request for validation errors.

401 Unauthorized for authentication failures.

403 Forbidden for authorization failures.

404 Not Found for missing resources.

409 Conflict for business rule violations (e.g., book already borrowed).