## **Designing a REST API**

### **Topic:** Catalogue (a library – books, authors, categories); 1. ****Functional Requirements****

* **Book Management**:

1. CRUD operations with books.
2. Filter, sort, and pagination of books.
3. Search by title or ISBN.
4. Borrow and return books, updating availability status.

* **Author Management**:

1. CRUD operations with authors.
2. List books by a an author.
3. Filter and paginate author lists by name.

* **Category Management**:

1. CRUD operations with categories.
2. List all books in a category.
3. Filter and paginate category lists.

### 2. ****Non-Functional Requirements****

* **Security**:

1. JWT-token authentication to secure endpoints.
2. Authorization controls for borrowing/returning books and creating/deleting resources.

* **Performance**:

1. Response times should be as low as it can be for paginated lists.
2. Implement caching for heavy endpoints, especially for GET /books, GET /authors, and GET /categories.

* **Reliability**:

1. Consistent response codes and error messages.
2. Use of ETags for caching.

* **Scalability**:

1. Support for pagination on large collections.

### 3. ****Entities and Operations****

#### ****Entities****

* **Book**

1. Attributes: id, title, isbn, publishDate, authorId, categoryId, summary, status (available/borrowed)

* **Author**

1. Attributes: id, name, biography, birthDate

* **Category**

1. Attributes: id, name, description

* **User**

1. Attributes: id, username, passwordHash, email, role (Admin or User)
2. **Roles**:

**Admin**: Full access to all operations, creating, updating, and deleting books, authors, and categories.

**User**: User can not perform create, update and delete book operations, same with authors and categories.

#### ****Operations****

##### ****Books****

* GET /books: Retrieve a list of books, supporting pagination, sorting, and filtering by author, category, and availability.
* GET /books/{id}: Retrieve a single book by ID.
* POST /books: Add a new book to the catalogue.
* PUT /books/{id}: Update details of a book.
* DELETE /books/{id}: Remove a book from the catalogue.
* PATCH /books/{id}/borrow: Change a book's status to "borrowed".
* PATCH /books/{id}/return: Change a book's status to "available".

##### ****Authors****

* GET /authors: Retrieve a list of authors, with pagination and filtering options.
* GET /authors/{id}: Get details of a specific author.
* GET /authors/{id}/books: Retrieve a list of books written by a specific author.
* POST /authors: Add a new author to the catalogue.
* PUT /authors/{id}: Update an author's details.
* DELETE /authors/{id}: Remove an author.

##### ****Categories****

* GET /categories: Retrieve a list of categories with pagination.
* GET /categories/{id}: Get details of a specific category.
* GET /categories/{id}/books: Retrieve a list of books in a specific category.
* POST /categories: Add a new category to the catalogue.
* PUT /categories/{id}: Update details of a category.
* DELETE /categories/{id}: Remove a category.

**User**

* POST /users: Register a new user (Admin access only).
* GET /users: List all users (Admin access only).
* GET /users/{id}: Retrieve user information (Admin can view all users; each User can only view their own profile).
* PUT /users/{id}: Update user information, such as changing passwords or role assignment (Admin access only).
* DELETE /users/{id}: Delete a user account (Admin access only).

**Admin Role**:

* Allowed to perform CRUD operations on all entities (Books, Authors, Categories, and Users).

**User Role**:

* Allowed to view Books, Authors, and Categories.
* Allowed to borrow or return books by interacting with the PATCH /books/{id}/borrow and PATCH /books/{id}/return endpoints.
* Restricted from creating, updating, or deleting books, authors, categories, or users.

### 4. ****Status Codes****

* **200 OK**: Success in GET, PATCH operations.
* **201 Created**: Resource successfully created.
* **204 No Content**: Resource deleted successfully.
* **400 Bad Request**: Invalid request, e.g., missing required fields.
* **401 Unauthorized**: Authentication required or failed.
* **403 Forbidden**: Insufficient permissions for action.
* **404 Not Found**: Resource does not exist.
* **409 Conflict**: Duplicate resource conflict.
* **500 Internal Server Error**: General server error.

### 5. ****Richardson Maturity Model****: ****Level 2**** Implementation

The API meets **Level 2** of the Richardson Maturity Model:

* Resources ( Books, Authors) are have unique endpoints.
* HTTP methods are used: GET, POST, PUT, DELETE, and PATCH.

### 6. ****Error Handling****

Error responses are structured with:

* status: HTTP status code.
* message: Description of the error.
* details: Optional field for detailed error information.
* errorId: Unique identifier for tracking errors in logs.

### 7. ****Authentication and Authorization****

* **JWT-Based Authentication**:

1. Users must authenticate via JWT tokens to access most endpoints.
2. Token required in the Authorization: Bearer <token> header.
3. Role-based access control:
   * + **Admin** role: Can create, update, and delete resources.
     + **User** role: Can view books and borrow/return them.

### 8. ****Pagination****

* **Pagination**:
  + Applied to all list endpoints using limit and offset or page and pageSize query parameters.
  + Example: /books?limit=10&offset=20

### 9. ****Caching****

* **Caching Strategy**:

1. Cache GET requests on books, authors, and categories endpoints.
2. Implement **ETags** and **Last-Modified** headers on GET requests.
3. Set **Cache-Control** headers with a short expiration ( Cache-Control: max-age=60).

## 