# 1. Overview

The Library Catalogue API provides a way to manage and query information about books, authors, and categories. The API supports full CRUD (Create, Read, Update, Delete) operations on these resources and includes filtering, pagination, and sorting mechanisms. It is designed as a RESTful service adhering to Level 3 of the Richardson Maturity Model by using hypermedia controls (HATEOAS) in responses.

---

# 2. Functional Requirements

- **Resource Management:**
  - **Books:** Manage books with details such as title, ISBN, publication date, summary, associated author(s), and category.
  - **Authors:** Manage author details including name, biography, and list of books.
  - **Categories:** Manage categories (e.g., Fiction, Non-fiction, Science, etc.) with names and descriptions.
- **Operations:**
  - **CRUD Operations:** Provide endpoints to create, read, update, and delete books, authors, and categories.
  - **Filtering & Sorting:** Allow filtering of collections (e.g., books by publication year, author, or category) and sorting (e.g., alphabetically by title).
  - **Pagination:** All collection endpoints must support pagination through query parameters.
  - **Search:** Optionally, support text-based search on resources.
- **Interactivity:**
  - Use hypermedia links (HATEOAS) in responses to guide clients to available next actions.

---

# 3. Non-Functional Requirements

- **Meaningful HTTP Status Codes:**
  - Use codes such as 200 (OK), 201 (Created), 204 (No Content), 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), and 500 (Internal Server Error).
- **Authentication & Authorization:**
  - The API uses a token-based authentication mechanism (e.g., Bearer tokens in the `Authorization` header). Unauthorized or forbidden access returns appropriate error codes.
- **Error Handling:**
  - Error responses include a JSON body with an error code, message, and, optionally, additional context (e.g., validation errors).

- **Caching:**
  - GET endpoints should be cached using standard HTTP headers (e.g., `Cache-Control`, `ETag`, and `Last-Modified`). Caching rules are applied to minimize load and enhance performance.
- **Pagination & Filtering:**
  - All endpoints that return collections are paginated and allow query parameters for page number, page size, filtering, and sorting.

---

# 4. Entities & Data Model

### Book

- **Properties:**
  - `id` (UUID/integer, unique identifier)
  - `title` (string)
  - `isbn` (string)
  - `publicationDate` (ISO 8601 date)
  - `summary` (string)
  - `authorId` (reference to Author)
  - `categoryId` (reference to Category)

### Author

- **Properties:**
  - `id` (UUID/integer)
  - `name` (string)
  - `biography` (string)
  - `books` (collection of Book IDs or embedded links)

### Category

- **Properties:**
  - `id` (UUID/integer)
  - `name` (string)
  - `description` (string)

---

# 5. REST API Endpoints

All endpoints return JSON and follow a consistent URI structure. The endpoints include hypermedia links to support HATEOAS.

## Books

- **List Books:**
  - **Endpoint:** `GET /books`
  - **Query Parameters:**
    - `page` (number, default: 1)
    - `pageSize` (number, default: 20)
    - `sort` (e.g., `title,asc` or `publicationDate,desc`)
    - `authorId` (filter by author)
    - `categoryId` (filter by category)
  - **Response:** 200 OK
    - Returns a paginated list of books with hypermedia links for next/prev pages and details on each book.
- **Get Book Details:**
  - **Endpoint:** `GET /books/{bookId}`
  - **Response:** 200 OK (or 404 Not Found)
    - Returns detailed book data along with links to the associated author and category.
- **Create a Book:**
  - **Endpoint:** `POST /books`
  - **Body:** JSON with book properties
  - **Response:** 201 Created (with Location header pointing to the new resource)
  - **Errors:** 400 Bad Request if validation fails.
- **Update a Book:**
  - **Endpoint:** `PUT /books/{bookId}`
  - **Body:** JSON with updated book properties
  - **Response:** 200 OK (or 204 No Content)
  - **Errors:** 400 Bad Request, 404 Not Found.
- **Delete a Book:**
  - **Endpoint:** `DELETE /books/{bookId}`
  - **Response:** 204 No Content
  - **Errors:** 404 Not Found.

## Authors

- **List Authors:**
  - **Endpoint:** `GET /authors`
  - **Query Parameters:** Same as books (pagination and optional sorting/filtering).
  - **Response:** 200 OK
- **Get Author Details:**
  - **Endpoint:** `GET /authors/{authorId}`
  - **Response:** 200 OK or 404 Not Found
- **Create an Author:**
  - **Endpoint:** `POST /authors`
  - **Response:** 201 Created
- **Update an Author:**

    - **Endpoint:** `PUT /authors/{authorId}`
    - **Response:** 200 OK
- **Delete an Author:**
    - **Endpoint:** `DELETE /authors/{authorId}`
    - **Response:** 204 No Content

### Categories

- **List Categories:**
    - **Endpoint:** `GET /categories`
    - **Query Parameters:** Supports pagination and sorting.
    - **Response:** 200 OK
- **Get Category Details:**
    - **Endpoint:** `GET /categories/{categoryId}`
    - **Response:** 200 OK or 404 Not Found
- **Create a Category:**
    - **Endpoint:** `POST /categories`
    - **Response:** 201 Created
- **Update a Category:**
    - **Endpoint:** `PUT /categories/{categoryId}`
    - **Response:** 200 OK
- **Delete a Category:**
    - **Endpoint:** `DELETE /categories/{categoryId}`
    - **Response:** 204 No Content

---

## 6. Error Handling

For all endpoints, errors are returned in a standardized JSON format:

```
{
  "error": "Error Title",
  "code": 400,
  "message": "Detailed error message explaining what went wrong.",
  "details": { ... } // optional additional context
}
```

- **Common HTTP Status Codes:**
    - **400 Bad Request:** Invalid input data.
    - **401 Unauthorized:** Missing or invalid authentication token.
    - **403 Forbidden:** Authenticated but lacking permissions.
    - **404 Not Found:** Resource not found.
    - **500 Internal Server Error:** Unexpected error on the server.

---

## 7. Authentication

- **Method:** Token-based authentication (e.g., OAuth 2.0 or API Keys).
- **Usage:**
  - The client must include a valid token in the `Authorization` header with the prefix `Bearer`.
  - Endpoints will return a **401 Unauthorized** response if the token is missing or invalid.

---

## 8. Pagination, Filtering, and Caching

- **Pagination:**
  - All collection endpoints support query parameters such as `page` and `pageSize` (or `limit` and `offset`) to control the number of records returned.
  - Hypermedia links for navigating to the next and previous pages are included in responses.
- **Filtering & Sorting:**
  - Query parameters (e.g., `authorId`, `categoryId`, `sort`) are supported on list endpoints to allow clients to filter and sort data.
- **Caching:**
  - GET endpoints include HTTP caching headers such as `Cache-Control` to define client caching policies.
  - The API supports ETag headers to enable clients to make conditional requests.
  - This caching strategy helps reduce server load and improves performance for frequently accessed data.

---

## 9. Richardson Maturity Model Considerations

- **Level 0 (The Swamp of POX):**
  - Not applicable; the design avoids RPC-like endpoints.
- **Level 1 (Resources):**
  - Resources are defined as `/books`, `/authors`, and `/categories`.
- **Level 2 (HTTP Verbs):**
  - Operations are mapped to HTTP methods (GET, POST, PUT, DELETE).
- **Level 3 (Hypermedia Controls):**
  - Responses include HATEOAS links that guide clients to available next actions (e.g., links to update, delete, or related resources).

---

# 10. Conclusion

This RESTful API design for a Library Catalogue System meets both the functional and non-functional requirements. It provides complete and unambiguous endpoints with meaningful HTTP status codes, thorough error handling, secure token-based authentication, and supports advanced features like filtering, pagination, and caching. The use of hypermedia controls places the API at Level 3 of the Richardson Maturity Model, ensuring a robust, scalable, and easy-to-use interface for clients.

This comprehensive design can now serve as a blueprint for implementation.