Designing a REST API

REST API Design: Library Catalogue System

1. Requirements

1.1. Functional Requirements

- **Manage Books:** The system must allow creating, reading, updating, and deleting (CRUD) book records.
- **Search & Filter:** Users must be able to search books by title and filter by author or genre.
- **Manage Authors:** The system must allow viewing authors and their associated books.
- **Pagination:** Lists of books and authors must be paginated to handle large datasets.
- **Hypermedia:** The API must provide navigation links (HATEOAS) to related resources.

1.2. Non-functional Requirements

- **Scalability:** The system must handle high read loads efficiently using caching mechanisms.
- **Security:** Access to modification operations (POST, PUT, DELETE) must be secured via Token-Based Authentication (JWT).
- **Interoperability:** The API must consume and produce `application/json`.
- **Performance:** Responses for list endpoints should not exceed 200ms (achieved via pagination and limiting fields).

---

2. Model description

Entity: Author

- `id` (UUID): Unique identifier

- `fullname` (String): The author’s full name
- `bio` (String): Short biography
- `birthDate` (Date): In format DD.MM.YYYY

Entity: Book

- `id` (UUID): Unique identifier
- `title` (String): The book title
- `genre` (Enum or String): Fiction, Science, History, Tech
- `authorId` (UUID): Foreign Keys referencing the `Author` entity

3. Operations description

Method	URI	Description	Access
GET	`/api/v1/books`	Get a paginated list of books. Supports filtering	Public
POST	`/api/v1/books`	Create a new book entry	Auth required
GET	`/api/v1/books/{id}`	Get detailed info about a specific book	Public
PUT	`/api/v1/books/{id}`	Update an existing book completely	Auth required
PATCH	`/api/v1/books/{id}`	Update an existing book partialy	Auth required
DELETE	`/api/v1/books/{id}`	Remove a book from the catalogue	Auth required
GET	`/api/v1/authors`	Get a paginated list of authors	Public
GET	`/api/v1/authors/{id}`	Get specific author details and their books	Public

4. Meaningful status codes

The API uses standard HTTP status codes to indicate the result of operations:

- **200 OK:** Request succeeded (used for `GET`, `PUT`).
- **201 Created:** Resource successfully created (used for `POST`). Response includes `Location` header.
- **204 No Content:** Request succeeded, but no body is returned (used for `DELETE`).
- **304 Not Modified:** The resource has not changed since the last request (Caching/ETag).
- **400 Bad Request:** Validation error (e.g., missing required fields).
- **401 Unauthorized:** Missing or invalid authentication token.
- **403 Forbidden:** Valid token, but the user lacks permissions (e.g., Reader trying to Delete).
- **404 Not Found:** Resource with the specified ID does not exist.
- **429 Too Many Requests:** Rate limit exceeded.
- **500 Internal Server Error:** Unexpected server-side error.

---

5. Richardson model application

The API is designed at Level 3 (Hypermedia Controls). Every resource contains a _links object guiding the client on what actions are possible next

Example Response (GET `/api/v1/books/b1`):

```
{
  "id": "b1",
  "title": "Rest API Design",
  "genre": "Tech",
  "_links": {
    "self": { "href": "/api/v1/books/b1" },
    "update": { "href": "/api/v1/books/b1", "method": "PUT" },
    "delete": { "href": "/api/v1/books/b1", "method": "DELETE" },
    "author": { "href": "/api/v1/authors/a1" },
    "collection": { "href": "/api/v1/books" }
  }
}
```

---

6. Authentication and Errors

Authentication

We use **Bearer Token (JWT)** standard.

- **Mechanism:** Every request to a protected route (POST, PUT, DELETE) must include the header `Authorization: Bearer <your_token>` .
- **Token Payload:** The token contains the user's ID (`sub`), expiration time (`exp`), and roles (`scope`).

Error Handling & Security Errors

The API returns a consistent JSON structure. Specifically for authentication, we strictly distinguish between **401** and **403**:

1. **401 Unauthorized:** The user did not provide a token, or the token is invalid/expired
   - *Example Response:*

   ```
   {
     "timestamp": "2025-11-10T10:05:00Z",
     "status": 401,
     "error": "Unauthorized",
     "message": "Full authentication is required to access this resource",
     "path": "/api/v1/books"
   }
   ```

2. **403 Forbidden:** The user provided a valid token, but does not have the `ADMIN` role required to perform the action
   - *Example Response:*

   ```
   {
     "timestamp": "2025-11-10T10:06:00Z",
     "status": 403,
     "error": "Forbidden",
     "message": "Access is denied. User does not have ADMIN privileges.",
     "path": "/api/v1/books/b1"
   }
   ```

7. Pagination

All "Collection" resources (`GET /books` , `GET /authors`) enforce pagination to protect the system.

- **Request Parameters:**
    - `page` : Page number (0-based, default: 0).
    - `size` : Number of items per page (default: 20, max: 100).
    - `sort` : Sorting field (e.g., `title,asc`).
- **Response Structure:**
  Wraps the data in a `content` array and provides `page` metadata.

```
{
  "content": [ ...list of books... ],
  "page": {
    "size": 20,
    "totalElements": 500,
    "totalPages": 25,
    "number": 0
  },
  "_links": {
    "self": { "href": "/api/v1/books?page=0&size=20" },
    "next": { "href": "/api/v1/books?page=1&size=20" },
    "last": { "href": "/api/v1/books?page=24&size=20" }
  }
}
```

8. Caching

The API utilizes HTTP **Client-Side Caching** to minimize bandwidth and server load.

1. **Expiration Model (`Cache-Control`):**
    - `GET /books` (Lists): `Cache-Control: public, max-age=60` (Cache for 1 minute).
    - `GET /books/{id}` (Details): `Cache-Control: public, max-age=3600` (Cache for 1 hour).

2. **Validation Model (`ETag`):**
    - The server includes an `ETag` (hash of the resource) in the response.
    - Subsequent requests from the client send `If-None-Match: "hash_value"`.
    - If the data hasn't changed, the server returns **304 Not Modified** with an empty body.
3. **Invalidation:**
    - `POST`, `PUT`, and `DELETE` requests automatically invalidate the cache for the specific resource URI.