Designing a REST API - Library

Roman Kuzmin
2024-08-03

The REST API description

1. **Collection (Entities)**

- Users (Readers, Librarians, Admin)
- Books
- Orders
- Loans
- Reading-room

2. **Models**

Entities	Example	Notes

User	{
	"id": 1,
	"username": "reader1",
	"email":
	"reader1@example.com",
	"role": "reader",
	"blocked": false,
	"loans": [
	{
	"book_id": 1,
	"due_date": "2023-12-01",
	"fine": 0
	}
	"reading-room": [
	{
	"book_id": 2,
	"due_date": "2023-11-01",
	"fine": 0
	}
	]
	}	

Book	{
	"id": 1,
	"title": "Book Title",
	"author": "John Doe",
	"publication": "Publication Name",
	"publication_date": "2020-01-01",
	"copies_available": 3
	}	

Order	{
	"id": 1,
	"book_id": 1,
	"user_id": 2,
	"status": "pending"
	}	

Loan	{	

	"id": 1, "order_id": 1, "book_id": 1, "user_id": 2, "issue_date": "2023-11-01", "due_date": "2023-12-01", "return_date": null, "fine": 0 }	

ReadingRoom	{
	"id": 1,
	"order_id": 2,
	"book_id": 2,
	"user_id": 2,
	"issue_date": "2023-11-01",
	"due_date": "2023-11-01",
	"return_date": null,
	"fine": 0
	}	

3. **API Endpoints**

Authentication
POST /register: Register a new user (reader).
POST /login: Authenticate a user and return a JWT token.

Users
GET /users/me: Get the authenticated user's profile.
PATCH /users/me: Update the authenticated user's profile.
GET /users/{id}: Get details of a specific user (admin or librarian only).

Books
GET /books: Get a list of books with filters and sorting options by default.
GET /books?sort-field={sort-field}&sort-dir={sort-dir}: Get a list of books with cusmom filters and sorting options.
GET /books/{id}: Get details of a specific book.
GET /books/search?author={auhtor}: Get a list of book by author
GET /books/search?title={title}: Get a list of book by title.
POST /books: Add a new book (admin only).
PATCH /books/{id}: Update a book (admin only).
DELETE /books/{id}: Delete a book (admin only).

Orders
POST /orders: Place a new order (authenticated readers only).
GET /orders: Get a list of orders (librarian only).
PATCH /orders/rejects/{id}: Update an order status to “rejected” (librarian only).
PATCH /orders/issues/{id}: Update an order status to “issues” (librarian only).

Loans
GET /loans: Get a list of loans (librarian only).
GET /loans/me: Get a list of the authenticated reader's loans.
POST /loans/{id}: Pay a loan (authenticated readers only).
PATCH /loans/{id}: Accept a book (librarian only)

Reading-room
GET /reading-room: Get a list of books in the reading room (librarian only).
GET /reading-room/me: Get a list of the authenticated reader's books in.
POST /reading-room: Issue a book (librarian only)
PATCH /reading-room/{id}: Accept a book (librarian only)

Admin
POST /admin/librarians: Create a new librarian (admin only).
DELETE /admin/librarians/{id}: Delete a librarian (admin only).
PATCH /admin/users/{id}/block: Block a user (admin only).
PATCH /admin/users/{id}/unblock: Unblock a user (admin only).

4. **API Example Implementation**

Endpoint	Request Example	Response Example	Bad Response Examples
Authentication	**Authentication**		
POST /register	POST /register { "username": "reader1", "password": "password123", "email": "reader1@example.com" }	201 Created { "id": 1, "username": "reader1", "email": "reader1@example.com" }	400 Bad Request { "error": "Email already exists" } or 400 Bad Request { "error": "Invalid data" }
POST /login	POST /login { "username": "reader1", "password": "password123" }	200 OK { "token": "jwt_token_here" }	401 Unauthorized { "error": "Invalid credentials" }
Users	**Users**		
GET /users/me	GET /users/me Authorization: Bearer <token>	200 OK { "id": 1, "username": "reader1", "email": "reader1@example.com", "role": "reader" }	401 Unauthorized { "error": "Invalid token" }
PATCH /users/me	PATCH /users/me Authorization: Bearer <token> { "email": "newemail@example.com" }	200 OK { "id": 1, "username": "reader1", "email": "newemail@example.com", "role": "reader" }	400 Bad Request { "error": "Invalid data" } or 401 Unauthorized { "error": "Invalid token" }
GET /users/{id}	GET /users/2 Authorization: Bearer <admin-token>	200 OK { "id": 2, "username": "reader2", "email": "reader2@example.com", "role": "reader" }	403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "User not found" }
Books	**Books**		
GET /books	GET /books? page=1&page_size=10	200 OK { "page": 1, "page_size": 10, "total": 100, "books": [{ "id": 1, "title": "Book Title", "author": "John Doe", "publication": "Publication Name", "publication_date": "2020-01-01" }, ...], "links": { "self": "/books?page=1&page_size=10", "next": "/books?page=2&page_size=10", "prev": null } }	400 Bad Request { "error": "Invalid pagination parameters" }
GET /books?sort-field={sort-field}&sort-dir={sort-dir}	GET /books?sort-field=title&sort-dir=asc&page=1&page_size=10	200 OK { "page": 1, "page_size": 10, "total": 100, "books": [{ "id": 1, "title": "Book Title", "author": "John Doe", "publication": "Publication Name", "publication_date": "2020-01-01" }, ...] }	400 Bad Request { "error": "Invalid sort parameters" }
GET /books/{id}	GET /books/1	200 OK { "id": 1, "title": "Book Title", "author": "John Doe", "publication": "Publication Name", "publication_date": "2020-01-01", "copies_available": 3 }	404 Not Found { "error": "Book not found" }
GET	GET /books/search?	200 OK { "books": [{ "id": 1,	400 Bad Request { "error":

Endpoint	Request Example	Response Example	Bad Response Examples
/books/search?author={author}	author=John	"title": "Book Title", "author": "John Doe", "publication": "Publication Name", "publication_date": "2020-01-01" }, ...] }	"Invalid author parameter" }
GET /books/search?title={title}	GET /books/search?title=Book+Title	200 OK { "books": [{ "id": 1, "title": "Book Title", "author": "John Doe", "publication": "Publication Name", "publication_date": "2020-01-01" }, ...] }	400 Bad Request { "error": "Invalid title parameter" }
POST /books	POST /books Authorization: Bearer <admin-token> { "title": "New Book", "author": "New Author", "publication": "New Publication", "publication_date": "2023-01-01", "copies_available": 5 }	201 Created { "id": 2, "title": "New Book", "author": "New Author", "publication": "New Publication", "publication_date": "2023-01-01", "copies_available": 5 }	400 Bad Request { "error": "Invalid book data" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" }
PATCH /books/{id}	PATCH /books/1 Authorization: Bearer <admin-token> { "title": "Updated Title" }	200 OK { "id": 1, "title": "Updated Title", "author": "John Doe", "publication": "Publication Name", "publication_date": "2020-01-01", "copies_available": 3 }	400 Bad Request { "error": "Invalid book data" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "Book not found" }
DELETE /books/{id}	DELETE /books/1 Authorization: Bearer <admin-token>	204 No Content	401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "Book not found" }
Orders	**Orders**		
POST /orders	POST /orders Authorization: Bearer <token> { "book_id": 1 }	201 Created { "id": 1, "book_id": 1, "user_id": 1, "status": "pending" }	400 Bad Request { "error": "Invalid book ID" } or 401 Unauthorized { "error": "Invalid token" }
GET /orders	GET /orders?page=1&page_size=10 Authorization: Bearer <librarian-token>	200 OK { "page": 1, "page_size": 10, "total": 50, "orders": [{ "id": 1, "book_id": 1, "user_id": 1, "status": "pending" }, ...] }	401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" }
PATCH /orders/rejects/{id}	PATCH /orders/rejects/1 Authorization: Bearer <librarian-token> { "status": "rejected" }	200 OK { "id": 1, "book_id": 1, "user_id": 1, "status": "rejected" }	400 Bad Request { "error": "Invalid order ID" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "Order not found" }
PATCH /orders/issues/{id}	PATCH /orders/issues/1 Authorization: Bearer <librarian-token> { "status": "issued" }	200 OK { "id": 1, "book_id": 1, "user_id": 1, "status": "issued" }	400 Bad Request { "error": "Invalid order ID" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "Order not found" }
Loans	**Loans**		

Endpoint	Request Example	Response Example	Bad Response Examples
GET /loans	GET /loans?page=1&page_size=10 Authorization: Bearer <librarian-token>	200 OK { "page": 1, "page_size": 10, "total": 30, "loans": [{ "id": 1, "book_id": 1, "user_id": 1, "due_date": "2023-12-01", "fine": 0 }, ...] }	401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" }
GET /loans/me	GET /loans/me?page=1&page_size=10 Authorization: Bearer <token>	200 OK { "page": 1, "page_size": 10, "total": 2, "loans": [{ "id": 1, "book_id": 1, "due_date": "2023-12-01", "fine": 0 }, ...] }	401 Unauthorized { "error": "Invalid token" }
POST /loans/{id}	POST /loans/1 Authorization: Bearer <token> { "payment": 100 }	200 OK { "id": 1, "book_id": 1, "user_id": 1, "due_date": "2023-12-01", "fine": 0 }	400 Bad Request { "error": "Invalid payment data" } or 401 Unauthorized { "error": "Invalid token" } or 404 Not Found { "error": "Loan not found" }
PATCH /loans/{id}	PATCH /loans/1 Authorization: Bearer <librarian-token> { "status": "accepted" }	200 OK { "id": 1, "book_id": 1, "user_id": 1, "due_date": "2023-12-01", "status": "accepted" }	400 Bad Request { "error": "Invalid loan ID" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "Loan not found" }
Reading-room	**Reading-room**		
GET /reading-room	GET /reading-room?page=1&page_size=10 Authorization: Bearer <librarian-token>	200 OK { "page": 1, "page_size": 10, "total": 20, "books": [{ "id": 1, "title": "Book Title", "author": "John Doe", "available": true }, ...] }	401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" }
GET /reading-room/me	GET /reading-room/me?page=1&page_size=10 Authorization: Bearer <token>	200 OK { "page": 1, "page_size": 10, "total": 3, "books": [{ "id": 1, "title": "Book Title", "author": "John Doe", "available": true }, ...] }	401 Unauthorized { "error": "Invalid token" }
POST /reading-room	POST /reading-room Authorization: Bearer <librarian-token> { "book_id": 1 }	201 Created { "id": 1, "book_id": 1, "user_id": 1, "status": "issued" }	400 Bad Request { "error": "Invalid book ID" } or 401 Unauthorized { "error": "Invalid token" }
PATCH /reading-room/{id}	PATCH /reading-room/1 Authorization: Bearer <librarian-token> { "status": "accepted" }	200 OK { "id": 1, "book_id": 1, "user_id": 1, "status": "accepted" }	400 Bad Request { "error": "Invalid reading room ID" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "Reading room entry not found" }
Admin	**Admin**		
POST /admin/librarians	POST /admin/librarians Authorization: Bearer <admin-token> { "username": "librarian1", "password": "password123", "email": "librarian1@example.com" }	201 Created { "id": 2, "username": "librarian1", "email": "librarian1@example.com" }	400 Bad Request { "error": "Invalid librarian data" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" }
DELETE /admin/librarians/{id}	DELETE /admin/librarians/1 Authorization: Bearer <admin-token>	204 No Content	401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404

Endpoint	Request Example	Response Example	Bad Response Examples
			Not Found { "error": "Librarian not found" }
PATCH /admin/users/{id}/block	PATCH /admin/users/2/block Authorization: Bearer <admin-token> { "blocked": true }	200 OK { "id": 2, "username": "reader2", "blocked": true }	400 Bad Request { "error": "Invalid user ID" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "User not found" }
PATCH /admin/users/{id}/unblock	PATCH /admin/users/2/unblock Authorization: Bearer <admin-token> { "blocked": false }	200 OK { "id": 2, "username": "reader2", "blocked": false }	400 Bad Request { "error": "Invalid user ID" } or 401 Unauthorized { "error": "Invalid token" } or 403 Forbidden { "error": "Access denied" } or 404 Not Found { "error": "User not found" }

5. **Error Handling in API Endpoints**

Authentication and Authorization Errors:

- Use 401 Unauthorized for issues related to token validity or credentials.
- Use 403 Forbidden for permission-related issues.

Validation and Data Errors: Use 400 Bad Request for invalid input or parameter errors.

Resource Issues: Use 404 Not Found when resources such as books or users are not found.

Conflicts: Use 409 Conflict when attempting to create resources that already exist.

Server Errors: Use 500 Internal Server Error for unhandled exceptions or server failures.

6. Authentication

JWT Structure:

- Header: Specifies the token type (JWT) and the signing algorithm (e.g., HS256).
- Payload: Contains the claims, which include:
  - sub: Subject of the token (user ID).
  - name: Username.
  - role: User role (e.g., reader, librarian, admin).
- Signature: Encoded by the header and payload using a secret key

Example JWT:

```
Header: {
  "alg": "HS256",
  "typ": "JWT"
}
Payload: {
  "sub": "userId123",
  "name": "john_doe",
  "role": "reader"
```

```
}
Signature: HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret_key
)
```

Authentication Flow:

- Registration: New users register by providing username, password, and email.
- Login: Users authenticate by sending their credentials (username and password). If valid, the server responds with a JWT.
- Access Protected Routes: For authenticated endpoints, include the JWT in the Authorization header as Bearer <token>.

7. Pagination

- All list endpoints support page and page_size query parameters (e.g., GET /books, GET /books?sort-field={sort-field}&sort-dir={sort-dir}, GET /books/search?author={auhtor}, GET /books/search?title={title}, GET /orders, GET /loans, GET /reading-room.)

8. Caching

- Cache responses for read-only endpoints (e.g., GET /books, GET /books/{id}) using a caching layer such as Redis.
- Implement cache invalidation on write operations (e.g., POST, PATCH, DELETE) to ensure data consistency.