1. Library requirements

1) Functional requirements

a. Manage books (CRUD operations).
b. Manage authors (CRUD operations).
c. Manage categories (CRUD operations).
d. Support searching and filtering of books.
e. Allow pagination for large datasets.
f. Implement user authentication for modifying data.
g. Maintain API responses in JSON format.

2) Non-Functional requirements

a. Ensure security through authentication and authorization.
b. Implement proper error handling.
c. Ensure API scalability and performance optimization.
d. Support caching mechanisms for frequently accessed data.
e. Ensure adherence to RESTful principles.

2. Enities

1) Book

a. id (UUID, primary key)
b. title (string, required)
c. author_id (UUID, foreign key to Author)
d. category_id (UUID, foreign key to Category)
e. published_date (date)
f. isbn (string, unique)
g. available_copies (integer, default: 1)

2) Author

a. id (UUID, primary key)
b. name (string, required)
c. biography (text, optional)

3) Category

a. id (UUID, primary key)
b. name (string, required, unique)
c. description (text, optional)

3. API Operations

1) Book

a. **GET /books** (Retrieve all books with filters and pagination)

b. **GET /books/{id}** (Retrieve a single book by ID)
c. **POST /books** (Create a new book, requires authentication)
d. **PUT /books/{id}** (Update book details, requires authentication)
e. **DELETE /books/{id}** (Delete a book, requires authentication)

2) Author

d. **GET /authors** (Retrieve all authors with pagination)
e. **GET /authors/{id}** (Retrieve a single author by ID)
f. **POST /authors** (Create a new author, requires authentication)
g. **PUT /authors/{id}** (Update author details, requires authentication)
h. **DELETE /authors/{id}** (Delete an author, requires authentication)

3) Category

d. **GET /categories** (Retrieve all categories)
e. **GET /categories/{id}** (Retrieve a single category by ID)
f. **POST /categories** (Create a new category, requires authentication)
g. **PUT /categories/{id}** (Update category details, requires authentication)
h. **DELETE /categories/{id}** (Delete a category, requires authentication)

4. REST API Design

1) Filters

a. **GET /books?title={title}** (Filter books by title)
b. **GET /books?author={author_name}** (Filter books by author)
c. **GET /books?category={category_name}** (Filter books by category)
d. **GET /books?published_after={date}** (Filter books published after a certain date)

2) Pagination

Default page size: 10

a. GET /books?page=1&pageSize=10
b. GET /authors?page=1&pageSize=10
c. GET /categories?page=1&pageSize=10

3) Authentication

JWT-based authentication (OAuth2, Bearer Token)

a. **POST /auth/login** (User login, returns token)
b. **POST /auth/logout** (User logout)

4) Errors

a. 400 Bad Request: Invalid request data
b. 401 Unauthorized: Authentication required
c. 403 Forbidden: Access denied
d. 404 Not Found: Resource not found
e. 500 Internal Server Error: Server failure

5) Caching

a. **GET /books**: Cached for 10 minutes
b. **GET /authors**: Cached for 10 minutes
c. **GET /categories**: Cached for 10 minutes

ETag and Last-Modified headers for cache validation

6) Richardson Maturity Model Application

a. **Level 1**: Uses resource-based URIs (/books, /authors, /categories)
b. **Level 2**: Uses HTTP verbs (GET, POST, PUT, DELETE)
c. **Level 3**: Implements HATEOAS links for navigation