I’ll use a Library Catalogue system, which includes books, authors, categories, etc. If you'd like to go with a different topic, the structure will remain similar, and we can adapt as needed.

Identify Entities

For a Library Catalogue, these might include:

* Book (Attributes: id, title, author\_id, category\_id, published\_date, isbn, summary, status (available, checked-out))
* Author (Attributes: id, name, bio, birthdate, books (a list of authored book IDs))
* Category (Attributes: id, name, description, parent\_category\_id (for nested categories))
* User (Attributes: id, name, email, membership\_id, borrowed\_books (list of currently checked-out book IDs))

Define Operations

Common operations for each entity:

* Books
  + GET /books - Retrieve all books (supports filtering, pagination)
  + POST /books - Add a new book
  + GET /books/{id} - Retrieve a specific book by ID
  + PUT /books/{id} - Update details of a specific book
  + DELETE /books/{id} - Delete a book by ID
* Authors
  + GET /authors - Retrieve all authors (supports filtering, pagination)
  + POST /authors - Add a new author
  + GET /authors/{id} - Retrieve an author by ID
  + PUT /authors/{id} - Update author details
  + DELETE /authors/{id} - Delete an author by ID
* Categories
  + GET /categories - Retrieve all categories
  + POST /categories - Add a new category
  + GET /categories/{id} - Retrieve a specific category by ID
  + PUT /categories/{id} - Update category details
  + DELETE /categories/{id} - Delete a category by ID
* Users
  + GET /users - Retrieve all users (supports filtering, pagination)
  + POST /users - Add a new user
  + GET /users/{id} - Retrieve user by ID
  + PUT /users/{id} - Update user details
  + DELETE /users/{id} - Delete a user by ID

Design the REST API

a) Collections and Filters

For collections, each entity (e.g., books, authors) should support operations like GET, POST, PUT, and DELETE for flexibility.

Filters:

* For /books, filter by author\_id, category\_id, status (e.g., available or checked-out).
* For /authors, filter by name or part of a bio.
* For /categories, filter by parent\_category\_id.

b) Pagination

Use pagination to limit responses:

* Example: GET /books?limit=10&page=2 - Retrieves the second page of books, with 10 books per page.

c) Error Handling

Use standard HTTP status codes:

* 200 OK - Success on GET, PUT, or DELETE requests
* 201 Created - Success on POST requests
* 204 No Content - Success when deleting a resource
* 400 Bad Request - Invalid request syntax
* 404 Not Found - Resource not found
* 401 Unauthorized - Not authenticated
* 403 Forbidden - Authenticated but lacks permissions
* 500 Internal Server Error - Unexpected server error

d) Authentication

A common method is JWT (JSON Web Token) authentication, where clients pass a token in the request headers, e.g., Authorization: Bearer <token>. This ensures secure access to resources.

e) Richardson Maturity Model

To ensure RESTful compliance:

1. Level 1: Use resources (/books, /authors)
2. Level 2: Utilize proper HTTP methods (GET, POST, PUT, DELETE)
3. Level 3: Implement HATEOAS (Hypermedia as the Engine of Application State), adding URLs in responses for easy client navigation.

f) Caching

* GET requests should be cached to improve response times.
* Set Cache-Control headers, such as Cache-Control: public, max-age=3600 for frequently accessed data.
* Use ETags to cache and validate resources.

Final REST API Documentation Example

Here's a summary of an endpoint for the Book entity:

GET /books

* Retrieves all books.
* Query Parameters:
  + author\_id: Filter books by author
  + category\_id: Filter by category
  + status: Filter by availability (e.g., available, checked-out)
  + limit and page: Pagination controls

POST /books

* Creates a new book.
* Body:

json

Копировать код

{

"title": "string",

"author\_id": "integer",

"category\_id": "integer",

"published\_date": "date",

"isbn": "string",

"summary": "string",

"status": "available"

}

Responses:

* 200 OK: Successful retrieval
* 201 Created: Book created
* 400 Bad Request: Invalid data
* 404 Not Found: Book not found (for specific book requests)

Зображення, що містить текст, схема, ряд, План

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис