Library API Homework: Building a RESTful API with Fake Data

Objective: To design and implement a simple RESTful API for a library system using fake data. This exercise will help you understand API design principles, HTTP methods, and how to serve data through endpoints.

Background: You are tasked with creating a backend for a small, imaginary library. This API will allow front-end applications to interact with library resources (books, authors, users). Since we don't have a database, you will generate and manage all data in memory.

Requirements:

1. **Technology:** You can use any programming language or framework you are familiar with (e.g., Node.js with Express, Python with Flask/FastAPI, Ruby on Rails, Go, etc.).
2. **Fake Data:** All data should be generated programmatically within your application (e.g., hardcoded arrays of objects, or using libraries like faker.js for Node.js, Faker for Python). You are not allowed to use a real database for this assignment.
3. **API Endpoints:** Your API should expose the following endpoints:
    - **Books:**
        - GET /api/books: Retrieve a list of all books.
        - GET /api/books/{id}: Retrieve a single book by its ID.
        - POST /api/books: Add a new book. (Include validation for required fields like title, authorId, isbn).
        - PUT /api/books/{id}: Update an existing book by its ID.
        - DELETE /api/books/{id}: Delete a book by its ID.
    - **Authors:**
        - GET /api/authors: Retrieve a list of all authors.
        - GET /api/authors/{id}: Retrieve a single author by their ID.
        - POST /api/authors: Add a new author. (Include validation for required fields like name).
        - PUT /api/authors/{id}: Update an existing author by their ID.
        - DELETE /api/authors/{id}: Delete an author by their ID.
    - **Users (Library Members):**
        - GET /api/users: Retrieve a list of all library members.
        - GET /api/users/{id}: Retrieve a single user by their ID.
        - POST /api/users: Add a new user. (Include validation for required fields like name, email).
        - PUT /api/users/{id}: Update an existing user by their ID.

        - DELETE /api/users/{id}: Delete a user by their ID.
4. **Data Structure (Example):**
    - **Book Object:**

```
{
  "id": "string", // Unique identifier (e.g., UUID)
  "title": "string",
  "authorId": "string", // Reference to an author's ID
  "isbn": "string", // International Standard Book Number
  "publicationYear": "number",
  "genre": "string",
  "availableCopies": "number"
}
```

    - **Author Object:**

```
{
  "id": "string", // Unique identifier
  "name": "string",
  "bio": "string",
  "nationality": "string"
}
```

    - **User Object:**

```
{
  "id": "string", // Unique identifier
  "name": "string",
  "email": "string",
  "membershipDate": "string" // Date in ISO format (e.g., "YYYY-MM-DD")
}
```

5. **Error Handling:** Implement basic error handling for:
    - **404 Not Found:** When an ID does not correspond to an existing resource.
    - **400 Bad Request:** When POST or PUT requests have missing or invalid data.
    - **405 Method Not Allowed (Optional but recommended):** If you access an endpoint with an unsupported HTTP method.
6. **CORS (Optional but Recommended):** If you plan to test with a separate front-end, enable Cross-Origin Resource Sharing.

Instructions:

1. **Set up your project:** Create a new project for your chosen framework/language.

2. **Generate Fake Data:** Create initial arrays of objects (e.g., 5-10 books, 3-5 authors, 5-7 users) to populate your in-memory "database." Ensure relationships (e.g., book.authorId pointing to an author.id) are correctly established.
3. **Implement Endpoints:** Write the code for each required API endpoint, handling the appropriate HTTP methods and logic for interacting with your fake data.
4. **Testing:** Use a tool like Postman, Insomnia, or curl to test each endpoint thoroughly. Verify that data is correctly retrieved, added, updated, and deleted.
5. **Documentation:** Briefly explain your API design and how to run your application.

Submission:

- Your source code.
- Instructions on how to run your API.
- A brief write-up describing any challenges you faced and how you overcame them.

Evaluation Criteria:

- Correct implementation of all required API endpoints and HTTP methods.
- Proper handling of fake data (creating, reading, updating, deleting).
- Basic error handling (404, 400).
- Code quality, readability, and comments.
- Adherence to RESTful principles where applicable.

Good luck!