Scenario:

You are developing an API for managing data related to a library. The API should handle information about books, authors, and users. Each book has a unique identifier (`id`), a title, an author, and a publication year. Users can borrow and return books. Each user has a unique identifier (`id`), a name, and a list of borrowed books.

Data:

Use the following complex data for testing your API:

let authors = [

{ id: 1, name: 'George Orwell' },

{ id: 2, name: 'Jane Austen' },

];

let books = [

{ id: 1, title: '1984', author: 'George Orwell', year: 1949 },

{ id: 2, title: 'Pride and Prejudice', author: 'Jane Austen', year: 1813 },

];

let users = [

{ id: 1, name: 'John Doe', borrowedBooks: [1] },

{ id: 2, name: 'Jane Doe', borrowedBooks: [2] },

];

**API Endpoints:**

1. Get All Books

- Endpoint: `GET /books`

- Description: Retrieve a list of all books.

- Example Response:

[

{ "id": 1, "title": "1984", "author": "George Orwell", "year": 1949 },

{ "id": 2, "title": "Pride and Prejudice", "author": "Jane Austen", "year": 1813 }

]

2. Get Book by ID

- Endpoint: `GET /books/:id`

- Description: Retrieve details of a specific book by its ID.

- Example Response:

{ "id": 1, "title": "1984", "author": "George Orwell", "year": 1949 }

3. Add a New Book

- Endpoint: `POST /books`

- Description: Add a new book to the library.

- Request Body:

{

"title": "New Book",

"author": "New Author",

"year": 2022

}

- Example Response:

{ "message": "Book added successfully", "books": [...updated book list...] }

4. Update Book by ID

- Endpoint: `PUT /books/:id`

- Description: Update details of a specific book by its ID.

- Request Body:

{

"title": "Updated Book",

"author": "Updated Author",

"year": 2023

}

- Example Response:

{ "message": "Book updated successfully", "books": [...updated book list...] }

5. Delete Book by ID

- Endpoint: `DELETE /books/:id`

- Description: Delete a book from the library by its ID.

- Example Response:

{ "message": "Book deleted successfully", "books": [...updated book list...] }

6. Get All Authors

- Endpoint: `GET /authors`

- Description: Retrieve a list of all authors.

- Example Response:

[

{ "id": 1, "name": "George Orwell" },

{ "id": 2, "name": "Jane Austen" }

]

```

7. Get Author by ID

- Endpoint: `GET /authors/:id`

- Description: Retrieve details of a specific author by their ID.

- Example Response:

{ "id": 1, "name": "George Orwell" }

8. Add a New User

- Endpoint: `POST /users`

- Description: Add a new user to the library system.

- Request Body:

{

"name": "New User",

"borrowedBooks": [1, 2]

}

- Example Response:

{ "message": "User added successfully", "users": [...updated user list...] }

9. Get User by ID

- Endpoint: `GET /users/:id`

- Description: Retrieve details of a specific user by their ID.

- Example Response:

{ "id": 1, "name": "John Doe", "borrowedBooks": [1] }

10. Return Book by User

- Endpoint: `PUT /users/:id/return`

- Description: Return a borrowed book for a specific user.

- Request Body:

{

"bookId": 1

}

- Example Response:

{ "message": "Book returned successfully", "users": [...updated user list...], "books": [...updated book list...] }

Submission:

1. Implement the Complex Data Management API based on the provided scenario and dummy data.

2. Test each API endpoint using tools like Postman.

3. Submit the complete code along with explanations and comments.