Complete Guide to Testing User and Admin Features

This guide will walk you through testing the **user** and **admin** functionalities of your API using Postman, from account creation to accessing and manipulating book data.

---

1. Prerequisites

- Install Postman for API testing.
- Clone this repository https://github.com/jodahe1/BookApiStage-3.git (check readme to see how to clone and run it)
- Ensure your API server is running locally or deployed.

---

2. Testing Admin Features

a. Admin Sign Up

- **Purpose:** Create an admin account.
- **Method:** `POST`
- **Endpoint:** `/auth/signup`
- **Body (JSON):**
- ```
  {
    "username": "adminUser",
    "password": "adminPassword",
    "role": "admin"
  }
  ```
- **Expected Response:**
- ```
  {
    "message": "Welldone User created successfully!"
  }
  ```

b. Admin Log In

- **Purpose:** Log in as an admin and receive a token.
- **Method:** `POST`
- **Endpoint:** `/auth/login`
- **Body (JSON):**
- ```
  {
    "username": "adminUser",
    "password": "adminPassword"
  ```
  ```

- }
- **Expected Response:**
- {
-   "token": "<admin-jwt-token>"
- }
- **Copy the token** for use in subsequent requests.

c. Admin Access - Fetch All Books

- **Purpose:** Admins can fetch all books.
- **Method:** `GET`
- **Endpoint:** `/books/all`
- **Headers:**
  - **Key:** `Authorization`
  - **Value:** `Bearer <admin-jwt-token>`
- **Expected Response:**
- {
-   "books": [
-     {
-       "_id": "bookId1",
-       "title": "Book Title",
-       "author": "Author Name",
-       "isbn": "123456789",
-       "publishedYear": 2021
-     }
-   ]
- }

d. Admin Access - Delete a Book

- **Purpose:** Admins can delete a book.
- **Method:** `DELETE`
- **Endpoint:** `/books/:id`
  - Replace `:id` with the book's ID.
- **Headers:**
  - **Key:** `Authorization`
  - **Value:** `Bearer <admin-jwt-token>`
- **Expected Response:**
- {
-   "message": "Book deleted successfully"
- }

---

3. Testing User Features

a. User Sign Up

- **Purpose:** Create a regular user account.
- **Method:** `POST`
- **Endpoint:** `/auth/signup`
- **Body (JSON):**
- ```
  {
    "username": "user123",
    "password": "userPassword"
  }
  ```
- **Expected Response:**
- ```
  {
    "message": "Welldone User created successfully!"
  }
  ```

b. User Log In

- **Purpose:** Log in as a user and receive a token.
- **Method:** `POST`
- **Endpoint:** `/auth/login`
- **Body (JSON):**
- ```
  {
    "username": "user123",
    "password": "userPassword"
  }
  ```
- **Expected Response:**
- ```
  {
    "token": "<user-jwt-token>"
  }
  ```
- **Copy the token** for use in subsequent requests.

c. User Access - Fetch Accessible Books

- **Purpose:** Users can access the books that are available to them.
- **Method:** `GET`
- **Endpoint:** `/books`
- **Headers:**
  - **Key:** `Authorization`
  - **Value:** `Bearer <user-jwt-token>`
- **Expected Response:**
- ```
  {
    "books": [
      {
        "_id": "bookId1",
        "title": "Book Title",
        "author": "Author Name",
        "isbn": "123456789",
        "publishedYear": 2021
      }
  ```
  ```

-]
- }

d. User Access - Create a Book

- **Purpose:** Users can create new books.
- **Method:** `POST`
- **Endpoint:** `/books`
- **Headers:**
  - **Key:** `Authorization`
  - **Value:** `Bearer <user-jwt-token>`
- **Body (JSON):**
- {
-   "title": "New Book",
-   "author": "Author Name",
-   "isbn": "123456789",
-   "publishedYear": 2023
- }
- **Expected Response:**
- {
-   "message": "Book Added successfully !!",
-   "book": {
-     "_id": "bookId",
-     "title": "New Book",
-     "author": "Author Name",
-     "isbn": "123456789",
-     "publishedYear": 2023
-   }
- }

e. User Access - Fetch a Book by ID

- **Purpose:** Users can retrieve a specific book by its ID.
- **Method:** `GET`
- **Endpoint:** `/books/:id`
  - Replace `:id` with the actual book ID.
- **Headers:**
  - **Key:** `Authorization`
  - **Value:** `Bearer <user-jwt-token>`
- **Expected Response:**
- {
-   "book": {
-     "_id": "bookId",
-     "title": "Book Title",
-     "author": "Author Name",
-     "isbn": "123456789",
-     "publishedYear": 2021

- `    }`
- `}`

---

4. Postman Workflow for Both User and Admin

Step 1: Set Up Postman

1. Create a new Postman collection named **"Books API Testing"**.
2. Add environment variables:
    - **Key:** `baseURL`
    - **Value:** `http://localhost:3000` (or your deployed URL).
    - **Key:** `userToken`
    - **Value:** Leave empty initially.
    - **Key:** `adminToken`
    - **Value:** Leave empty initially.

Step 2: Test Admin Features

1. **Admin Sign Up**: Use the `POST /auth/signup` endpoint to create an admin account.
2. **Admin Log In**: Use the `POST /auth/login` endpoint to get the admin token.
3. **Admin Fetch Books**: Use the `GET /books/all` endpoint with the `Authorization` header to fetch all books.
4. **Admin Delete Book**: Use the `DELETE /books/:id` endpoint with the `Authorization` header to delete a book.

Step 3: Test User Features

1. **User Sign Up**: Use the `POST /auth/signup` endpoint to create a regular user account.
2. **User Log In**: Use the `POST /auth/login` endpoint to get the user token.
3. **User Fetch Books**: Use the `GET /books` endpoint with the `Authorization` header to fetch accessible books.
4. **User Create Book**: Use the `POST /books` endpoint with the `Authorization` header to create a book.

---

5. Error Handling

- **Invalid Token**: If you receive an "Access Denied" error, verify the token and ensure it’s being sent correctly.

- **Unauthorized Action**: If you receive "Access Denied" or "Forbidden" errors, ensure the correct role and permissions are set for the route.

Common Errors

- **Missing Authorization Header:**
- ```
  {
    "error": "Access Denied"
  }
  ```
- **Book Not Found (Invalid ID):**
- ```
  {
    "error": "Book not found"
  }
  ```

---

6. Conclusion

By following the above steps, you will be able to thoroughly test both **admin** and **user** functionalities for managing books, including signup, login, fetching, creating, and deleting books if you are using other testing tool like insomnia it’s almost similar just the Interface difference , not logic differences .