**LIBRARY MANAGEMENT SYSTEM**

**DOCUMENTATION**

**1. Database Structure**

This system uses MongoDB, a NoSQL database, which structures data as documents rather than traditional relational tables. However, we can treat collections and fields similarly to tables and columns.

**Collections (Tables)**

**1.1 Users Collection**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| \_id | ObjectId | Unique identifier for each user (auto-generated). |
| username | String | The username for logging in. Unique for each user. |
| password | String | The password for the user (hashed). |
| role | String | Role of the user, either LIBRARIAN or MEMBER. |
| createdAt | Date | The date the account was created (auto-generated). |

**1.2 Books Collection**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| \_id | ObjectId | Unique identifier for each book (auto-generated). |
| title | String | Title of the book. |
| author | String | Author of the book. |
| isbn | String | ISBN (International Standard Book Number) for the book. Unique. |
| status | String | The current status of the book. Either AVAILABLE or BORROWED. |
| createdAt | Date | The date the book was added to the library. |

**1.3 History Collection**

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| \_id | ObjectId | Unique identifier for each history entry (auto-generated). |
| memberId | ObjectId | Reference to the member who borrowed or returned the book. |
| bookId | ObjectId | Reference to the book that was borrowed or returned. |
| actions | Enum | Enum value to track the book's status, either Borrowed or available. |
| date | Date | The date when the action (borrow/return) took place. |

**2. Database Diagram**

Here’s a basic diagram to represent the relationships between the collections:

+-------------+ +----------------+ +--------------+

| Users | | Books | | History |

+-------------+ +--------------- + +--------------+

| \_id | <--------| borrowedBy | | userId |

| username | | \_id | <-------- | bookId |

| password | | title | | borrowedAt|

| role | | author | | returnedAt |

+-------------+ +----------------+ +--------------+

**3. API Documentation**

**3.1 Authentication (Auth) API**

* **Signup (POST /api/auth/signup)**:
  + **Requirements**:
    - username (string)
    - password (string)
    - role (string, either LIBRARIAN or MEMBER)
  + **Output**:
    - On success: { success: true, message: 'User registered successfully' }
    - On failure: { success: false, message: 'User already exists' }
  + **Errors**:
    - 400: User already exists.
    - 500: Server error.
* **Login (POST /api/auth/login)**:
  + **Requirements**:
    - username (string)
    - password (string)
  + **Output**:
    - On success: { token: 'JWT token' }
    - On failure: { message: 'Invalid credentials' }
  + **Errors**:
    - 400: Invalid credentials.
    - 500: Server error.

**3.2 Books API**

* **View All Books (GET /api/books/view)**:
  + **Requirements**:
* Auth token (JWT)
  + **Output**:
    - List of books, including title, author, ISBN, and status.
  + **Errors**:
    - 403: Authorization failed.
    - 500: Server error.
* **Add New Book (POST /api/books/add)**:
  + **Requirements**:
    - title (string)
    - author (string)
    - isbn (string, unique)
    - Auth token (JWT)
  + **Output**:
    - On success: { success: true, message: 'Book added successfully' }
  + **Errors**:
    - 400: Validation errors (e.g., missing fields).
    - 500: Server error.
* **Update Book (PUT /api/books/update/:id)**:
  + **Requirements**:
    - title (string)
    - author (string)
    - isbn (string)
    - Auth token (JWT)
  + **Output**:
    - On success: { success: true, message: 'Book updated successfully' }
  + **Errors**:
    - 404: Book not found.
    - 500: Server error.
* **Delete Book (DELETE /api/books/remove/:id)**:
  + **Requirements**:
* Auth token (JWT)
  + **Output**:
    - On success: { success: true, message: 'Book removed successfully' }
  + **Errors**:
    - 404: Book not found.
    - 500: Server error.

**3.3 History API**

* **View Borrow/Return History (GET /api/history/view)**:
  + **Requirements**:
* Auth token (JWT)
  + **Output**:
    - List of all history entries (including borrowedAt and returnedAt dates).
  + **Errors**:
    - 500: Server error.
* **Borrow a Book (POST /api/history/borrow/:id):**
* **Requirements**:
* bookId (in the URL): The ID of the book being borrowed.
* Auth token (JWT) from the user.
* **Output**:
* On success: { success: true, message: 'Book borrowed successfully', historyId: <historyId> }
* **Errors**:
* 404: Book not found or already borrowed.
* 500: Server error.
* **Return a Book (POST /api/history/return/:id)**
* **Requirements**:
* bookId (in the URL): The ID of the book being returned.
* Auth token (JWT) from the user.
* **Output**:
* On success: { success: true, message: 'Book returned successfully' }
* **Errors**:
* 404: Book not found or not borrowed by this user.
* 500: Server error.

**4. Frontend Documentation and Flow**

**4.1 Login/Signup Flow**

* The **login** and **signup** pages allow users to enter their credentials and role.
* Once logged in, a **JWT token** is stored in localStorage, and the user is redirected to the home page.

**4.2 Books Flow**

* The **books** page lists all books, and based on the user's role (LIBRARIAN or MEMBER):
  + **Members**: Can borrow books (status changes to BORROWED).
  + **Librarians**: Can add, update, or delete books.
* A **search bar** allows users to filter books by title or author.

**4.3 Members Flow**

* The **members** page is accessible only to **librarians**.
* Librarians can view the list of members, update member details, or delete member accounts.

**4.4 History Flow**

* Users can view their borrowing history on the **history page**.
* Librarians can view the borrowing and returning history of all members.

**5. Hosting Instructions**

**5.1 Backend Hosting (Render.com)**

1. **Account Creation**:
   * Create an account on [Render.com](https://render.com).
2. **Create New Web Service**:
   * In Render, create a new **Web Service** and connect it to the GitHub repository that contains the backend code.
3. **Environment Variables**:
   * Add the following environment variables in Render's settings:
     + PORT: The port number (e.g., 5000).
     + MONGODB\_URI: The MongoDB connection string (e.g., mongodb+srv://<user>:<password>@cluster.mongodb.net/database).
     + JWT\_SECRET: A secure secret string for JWT token signing.
4. **Deploy**:
   * Render will automatically build and deploy the backend service.
5. **Live Link**:
   * After deployment, Render provides a live link to your backend (e.g., https://your-backend-service.onrender.com).
   * **Live Link**: https://library-backend-x387.onrender.com/ (since it is a free hosting platform, it takes some time to load)

**5.2 Frontend Hosting (Netlify)**

1. **Account Creation**:
   * Create an account on [Netlify](https://netlify.com).
2. **Create New Web Service**:
   * In Netlify, create a new **Web Service** and connect it to the GitHub repository that contains the frontend code.
3. **Deploy**:
   * Netlify will automatically deploy the frontend from the connected GitHub repository.
4. **Live Link**:
   * Netlify will provide a live link to your frontend website (e.g., https://your-frontend-service.netlify.app).
   * **Live Link**: https://library-frontend1.netlify.app/