Library Management System

Backend

The Library Management System is designed to allow **Librarians** and **Members** to manage books and their borrowing history. The system facilitates:

* Managing a collection of books.
* Recording book borrowing and returning events.
* Assigning different roles (Librarian and Member) to users, with different access controls.
* Tracking the borrowing history of books by members.

Features:

* Proper CRUD operation using REST API’s
* JWT based authentication in each protected endpoint.
* There are two roles in the system; LIBRARIAN and MEMBER
* Member can sign up either as LIBRARIAN and MEMBER using username and password
* Member can login using username/password and get JWT access token
* Librarian can add, update, and remove Books from the system
* Librarian can update, view, and remove Member from the system
* Librarian can view history of all the members(issue and return of books)
* Librarian can view deleted and active members.
* Membercan view, borrow, and return available Books
* Once a book is borrowed, its status will change to BORROWED
* Once a book is returned, its status will change to AVAILABLE
* Member can delete my own account.
* Member can view the history of books borrowed.

**Technical Stack**

* **Backend:**
  + **Node.js** with **Express.js** for building the server-side.
  + **Sequelize ORM** for interacting with the database.
  + **Vercel DB** for database storage.
  + **JWT (JSON Web Token)** for authentication.
  + **Hosting on Vercel**

**Database Design:**

**User Table:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| **id** | **UUID** | **Primary Key** |
| **username** | **STRING** | **Unique, Not NULL** |
| **email** | **STRING** | **Unique, Not NULL** |
| **password** | **STRING** | **Not NULL** |
| **role** | **ENUM(LIBRARIAN, MEMBER)** | **Not NULL** |
| **status** | **ENUM(ACTIVE, INACTIVE** | **Default: ACTIVE** |

**Book Model:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| **id** | **UUID** | **Primary Key** |
| **title** | **STRING** | **Not NULL** |
| **author** | **STRING** | **Not NULL** |
| **status** | **ENUM(AVAILABLE, BORROWED)** | **Default: AVAILABLE** |

**History Model:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Constraint** |
| **id** | **UUID** | **Primary Key** |
| **borrowDate** | **DATE** | **Default: Now** |
| **returnDate** | **DATE** | **NULL** |
| **userId** | **UUID** | **Foreign Key (user)** |
| **bookId** | **UUID** | **Foreign Key (Book)** |
| **status** | **ENUM(BORROWED, RETURNED)** | **Default: BORROWED** |

**Associations:**

User.belongsToMany(Book, { through: BorrowHistory, foreignKey: 'userId', as: 'borrowedBooks' });

Book.belongsToMany(User, { through: BorrowHistory, foreignKey: 'bookId', as: 'borrowers' });

User.hasMany(BorrowHistory, { foreignKey: 'userId', sourceKey: 'id', as: 'borrowHistory' });

BorrowHistory.belongsTo(User, { foreignKey: 'userId', targetKey: 'id', as: 'user' });

**API’s:**

* 1. User Register:
     + Endpoint: /api/users/register
     + Method: POST
     + Requirement : Body: { email, username, password, role }
     + Output: Response 200: Ok, Success Message and redirection to login page
     + Errors:
       - Mandatory Fields not entered
       - User already exists
       - Username/ email already taken
       - Server error
  2. User Login:
     + Endpoint: /api/users/login
     + Method: POST
     + Requirement: Body: { username, password }
     + Output: Response 200: Ok, success message, user Details, jwt token
     + Errors:
       - Invalid Credentials
       - User doesnot exist
       - Account not active
       - Server error
  3. Show User Profile:
     + Endpoint: /api/users/profile
     + Method: GET
     + Requirement: Authorization: { token }
     + Output: Response 200: user details including the borrow history
     + Errors:
       - User not Logged In
       - Token expire
       - User not found
       - Server error
  4. Update Self User Details:
     + Endpoint: /api/users/update/self
     + Method: PUT
     + Requirement: Authorization: { token }, body: { username, email, password }
     + Output: Response 200: Success Message and updated UserData Object
     + Errors:
       - Token expire
       - User not found
       - UserName/email already taken
       - Server error
  5. Delete Self User:
     + Endpoint: /api/users/delete/self
     + Method: DELETE
     + Requriement: Authorization: { token }
     + Output: Response 200: Success Message, User Deleted
     + Errors:
       - Token error
       - User not found
       - Server error
  6. Librarian access: see active members:
     + Endpoint: /api/users/activeMembers
     + Method: GET
     + Requirement: Authorization { token }: token must belong to librarian
     + Output: Get the list of all active members
     + Errors:
       - Token error of Librarian
       - Server error
  7. Librarian access: see active members:
     + Endpoint: /api/users/deletedMembers
     + Method: GET
     + Requirement: Authorization { token }: token must belong to librarian
     + Output: Get the list of all deleted members
     + Errors:
       - Token error of Librarian
       - Server error
  8. Librarian access: see the user’s details:
     + Endpoint: /api/users/member/userId
     + Method: GET
     + Requirement: Authorization: { Librarian Token }, Params: { userId }
     + Output: Get user Details, along with borrow History
     + Errors:
       - Librarian Token error
       - Server error
       - User not found/ invalid user id
  9. Librarian access: update the user’s details:
     + Endpoint: /api/users/update/:userId
     + Method: PUT
     + Requirement: Authorization: {Librarian Token}, Params: {userId}, body: { username, email }
     + Output: Get user’s updated details
     + Errors:
       - Librarian Token error
       - Server error
       - User not found
       - Username/ email already taken
  10. Librarian access: Delete a user
      + Endpoint: /api/users/delete/:userId
      + Method: DELETE
      + Requirement: Authorization: {Librarian Token}, Params: (userId}
      + Output: Success Message
      + Errors:
        - Librarian Token error
        - Server error
        - User Not found
  11. Librarian access: add a new book
      + Endpoint: /api/books/add
      + Method: POST
      + Requirement: Authorization: {Librarian Token}, Body: {title, author}
      + Output: Response 200: Success message and book details
      + Errors:
        - Librarian Token error
        - Server error
  12. Librarian access: update a book details
      + Endpoint: /api/books/update/:bookId
      + Method: PUT
      + Requirement: Authorization: {Librarian Token}, Body: {updateTitle, updatedAuthor}, Params: {bookId}
      + Output: Response 200: Success message
      + Errors:
        - Librarian Token error
        - Book not found error
        - Server error
  13. Librarian access: delete a book
      + Endpoint: /api/books/delete/:bookId
      + Method: DELETE
      + Requirement: Authorization: {Librarian Token}, Params: {bookId}
      + Output: Response 200: Success message
      + Errors:
        - Librarian Token error
        - Book not found error
        - Server error
  14. Get all books
      + Endpoint: /api/books/
      + Method: GET
      + Requirement: Authorization: {token}
      + Output: Response 200: all book details object
      + Errors:
        - Token error
        - Server error
  15. Get a book by id:
      + Endpoint: /api/books/:bookId
      + Method: GET
      + Requirement: Authorization: { token }, Params: { bookId }
      + Output: Response 200: book details object
      + Errors:
        - Token error
        - Book not found error
        - Server error
  16. Borrow Book:
      + Endpoint: /api/books/borrow/:bookId/user/:userId
      + Method: POST
      + Requirement: Authorization: { token }, Params: { userId, bookId }
      + Output: Response 200: bookHistory object containing the details of book
      + Errors:
        - Token error
        - Book is already borrowed by the person requesting it
        - Book is borrowed by other person (not available)
        - User not found
        - User validation error
        - Server error
  17. Return Book:
      + Endpoint: /api/books/return/:bookId/user/:userId
      + Method: PUT
      + Requirement: Authorization: { token }, Params: { userId, bookId }
      + Output: Response 200: bookHistory object containing the details of book
      + Errors:
        - Token error
        - Book is already borrowed by the person requesting it
        - Book is borrowed by other person (not available)
        - User not found
        - User validation error
        - Server error

Future Aspects:  
Proper Handling of deleted user- Thought- Create new table for the deleted user and move them there, for now, I am just marking the flag inactive (didn’t click me when I was creating, but clicked when I was documenting)

Handling the case sensitivity in the api, if same thing is written twice but with different case, it will be considered two different things.

Add photos of the book