Table - User

Column	Data Type	Constraints
id	Integer	Primary Key, Auto-increment
password	Varchar	Not Null
last_login	Datetime	Nullable
is_superuser	Boolean	Not Null, Default: False
username	Varchar	Unique, Not Null
first_name	Varchar	Not Null
last_name	Varchar	Not Null
email	Varchar	Not Null
is_staff	Boolean	Not Null, Default: False
is_active	Boolean	Not Null, Default: True
date_joined	Datetime	Not Null
role	Varchar	Not Null, Choices: ('LIBRARIAN', 'MEMBER')
groups	Many-to-Many (Group)	Not Null
user_permissions	Many-to-Many (Permission)	Not Null

Table - Book

Column	Data Type	Constraints
id	Integer	Primary Key, Auto-increment
name	Varchar	Not Null
author	Varchar	Not Null
status	Varchar	Not Null, Default: 'AVAILABLE'

Table – Transaction of books

Column	Data Type	Constraints
id	Integer	Primary Key, Auto-increment
user_id	Integer	Foreign Key, References user_id, Not Null
book_id	Integer	Foreign Key, References book_id,Not Null
issue_date	Date	Not Null
return_date	Date	Nullable

Detailed structure of table is given above

Frontend and Flow Documentation

1. Home Page

Description: The home page serves as the entry point for all users. Here, users can either log in if they already have an account, or sign up if they are new.

Features:

Login: Users enter their username and password to log in.

Sign Up: New users can create an account by entering their details.

Flow:

User Visits Home Page: Presented with options to either log in or sign up.

User Signs Up: If a new user, they fill in their details (username, email, password, role).

Post Sign Up: Once signed up, the user is prompted to log in using their newly created credentials.

2. Login

Description: After signing up, users must log in to access their account based on their role (either MEMBER or LIBRARIAN).

Features:

Username & Password Input: Fields for users to enter their login credentials.

JWT Authentication: Secure token-based authentication for validating users.

Flow:

User Logs In: Enters username and password, which are authenticated.

Role Check: Upon successful login, the system checks the user's role.

Redirection: User is redirected to the appropriate page:

Librarian Page: For users with the LIBRARIAN role.

Member Page: For users with the MEMBER role.

3. Librarian Page

Description: Librarians can manage the library system, including adding, updating, and removing books, as well as managing members.

Features:

Book Management: Add, update, and delete books.

Member Management: View and manage members, including adding and removing members.

View History: View borrowing and return history of all members.

Flow:

Book Operations:

Librarian can add new books.

Update book details.

Remove books from the system.

Member Operations:

View list of members.

Add or remove members.

View the borrowing history of members.

4. Member Page

Description: Members can browse and borrow books, return borrowed books, delete their account, and view their borrowing history.

Features:

Browse Books: View available books in the library.

Borrow Books: Borrow available books.

Return Books: Return borrowed books.

Account Management: Delete own account.

View History: See the borrowing history of the user.

Flow:

Browsing and Borrowing:

Members browse the list of available books.

Borrow a book, changing its status to BORROWED.

Returning Books:

Return previously borrowed books, changing their status to AVAILABLE.

Account Operations:

Members can delete their own account.

View their own borrowing history.

Summary

This documentation outlines the user flow and frontend components of the library management system. The system ensures a seamless experience for both librarians and members, catering to their specific needs and roles.

API with requirements, output and various errors.

1. Home Page

Function: home

Endpoint: / Method: GET

Requirements:

No specific requirements.

Output:

Renders the index.html template.

Errors:

None, as this is a straightforward page render.

2. User Logout

Function: usr_logout

Endpoint: /logout/

Method: GET

Requirements:

User should be logged in.

Output:

200 OK: Returns {"status":"login_sucess"} on successful logout.

Errors:

None, as this simply logs out the user.

3. Member Page

Function: member

Endpoint: /member/ Method: GET

Requirements:

User should be authenticated.

Output:

Renders the member.html template.

Errors:

None, as this is a straightforward page render.

4. Librarian Page

Function: librarian

Endpoint: /librarian/

Method: GET

Requirements:

User should be authenticated and have the LIBRARIAN role.

Requires IsAuthenticated and CheckMember permissions.

Output:

Renders the librarian.html template with status context.

If authenticated as librarian: {"status": "OK"}

If not authenticated or not a librarian: {"status": "Error"}

Errors:

401 Unauthorized: If the user is not authenticated or does not have the LIBRARIAN role.

User Operations API

1. Register User

Function: register

Endpoint: /library/users/register/

Method: POST

Requirements:

username (string): The username for the new user.

password (string): The password for the new user.

role (string): The role of the user (LIBRARIAN or MEMBER).

first_name (string): The first name of the user.

Output:

201 Created: Returns user details on successful creation.

Errors:

403 Forbidden: If the input data is invalid.

2. User Login

Function: login

Endpoint: /library/users/login/

Method: POST

Requirements:

email (string): The email of the user.

password (string): The password for the user.

Output:

200 OK: Returns JWT tokens and user role on successful authentication.

refresh: Refresh token.

access: Access token.

role: User role in lowercase.

Errors:

400 Bad Request: If the credentials are invalid or input data is incorrect.

3. Remove Account

Function: remove_account

Endpoint: /library/users/remove_account/

Method: PATCH

Requirements:

Authorization header: Bearer token for authentication.

(Optional) user_id (integer): The ID of the user to deactivate (for admins).

Output:

200 OK: Confirms account has been deactivated.

Errors:

400 Bad Request: If there is an error in processing the request.

4. Update User

Function: update_user

Endpoint: /library/users/update_user/

Method: POST

Requirements:

user_id (integer): The ID of the user to update.

first_name (string): The new first name of the user.

Authorization header: Bearer token for authentication.

Output:

200 OK: Confirms account has been updated.

Errors:

400 Bad Request: If the input data is invalid.

403 Forbidden: If the user does not have permissions.

Book Operations API

1. Add New Book

Function: create_book

Endpoint: /library/books/

Method: POST

Requirements:

title (string): The title of the book.

author (string): The author of the book.

Authorization header: Bearer token for authentication.

Output:

201 Created: Returns details of the newly added book.

Errors:

400 Bad Request: If the input data is invalid.

403 Forbidden: If the user does not have permissions (must be a librarian).

2. Update Existing Book

Function: update_book

Endpoint: /library/books/{id}/

Method: PUT

Requirements:

id (integer): The ID of the book to update.

title (string, optional): The new title of the book.

author (string, optional): The new author of the book.

Authorization header: Bearer token for authentication.

Output:

200 OK: Returns the updated book details.

Errors:

400 Bad Request: If the input data is invalid.

403 Forbidden: If the user does not have permissions (must be a librarian).

404 Not Found: If the book with the given ID does not exist.

3. Remove Book

Function: delete_book

Endpoint: /library/books/{id}/

Method: DELETE

Requirements:

id (integer): The ID of the book to delete.

Authorization header: Bearer token for authentication.

Output:

204 No Content: On successful deletion.

Errors:

403 Forbidden: If the user does not have permissions (must be a librarian).

404 Not Found: If the book with the given ID does not exist.

400 Bad Request: If there are dependent transactions preventing deletion.

Transaction Operations API

1. Borrow Book

Function: borrow_book

Endpoint: /library/transactions/borrow/

Method: POST

Requirements:

book_id (integer): The ID of the book to borrow.

Authorization header: Bearer token for authentication.

Output:

201 Created: Returns transaction details on successful borrowing.

Errors:

400 Bad Request: If the book is not available or input data is invalid.

404 Not Found: If the book with the given ID does not exist.

2. Return Book

Function: return_book

Endpoint: /library/transactions/return/

Method: POST

Requirements:

transaction_id (integer): The ID of the transaction.

Authorization header: Bearer token for authentication.

Output:

200 OK: Returns transaction details on successful returning.

Errors:

400 Bad Request: If the book is not borrowed or input data is invalid.

404 Not Found: If the transaction with the given ID does not exist.

3. View Borrowing History

Function: history

Endpoint: /library/transactions/history/

Method: GET

Requirements:

Authorization header: Bearer token for authentication.

Output:

200 OK: Returns a list of transactions for the authenticated user.

Errors:

401 Unauthorized: If the user is not authenticated.

4. View All Borrowing History

Function: all_history

Endpoint: /library/transactions/all_history/

Method: GET

Requirements:

Authorization header: Bearer token for authentication with librarian permissions.

Output:

200 OK: Returns a list of all transactions.

Errors:

403 Forbidden: If the user does not have permissions (must be a librarian).