Library Management System API Documentation

Introduction

This API provides functionalities for managing a library system, including book management, user management, borrowing, and searching. The API is designed for two main roles:

- **Librarian**: Responsible for managing books, users, and transactions.
- **Member**: Can search for books, borrow, reserve, and return them.
- **Sytems**: is not considered as role but it sends notification for users.

Note: Use this Base URL below http://127.0.0.1:5000 Example: http://127.0.0.1:5000/auth/register

---

Authentication Routes

1. Register a User

Endpoint: `/auth/register`
Method: `POST`
Access: Public

Description: Registers a new user (Librarian or Member) in the system.

Request Body:

```
{
  "name": "Ahmed Mohamed",
  "email": "ahmed.mohamed@example.com",
  "password": "password123",
  "role": "Member"  // or "Librarian"
}
```

Response:

- **Success** (`201`):

  ```
  {
    "message": "User registered successfully"
  }
  ```

- **Failure** (`400`):

  ```
  {
    "error": "A user with this email already exists"
  }
  ```

Notes:

- The `role` field must be either "Member" or "Librarian".
- Librarian emails must meet specific validation criteria.
- User barcode is automatically generated by the system, not provided during registration.

---

2. Login

Endpoint: `/auth/login`
Method: `POST`
Access: Public

Description: Logs in a user and generates JWT tokens.

Request Body:

```
{
  "email": "ahmed.mohamed@example.com",
  "password": "password123"
}
```

Response:

- **Success** (`200`):

  ```
  {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
  ```

- **Failure** (`401`):

  ```
  {
    "error": "Invalid credentials"
  }
  ```

Notes:

- The access token should be included in the Authorization header for protected routes.
- Format: `Authorization: Bearer <access_token>`

---

3. Refresh Token

Endpoint: `/auth/refresh`
Method: `POST`
Access: Requires a valid refresh token

Description: Generates a new access token using a valid refresh token.

Response:

- **Success** (`200`):

  ```
  {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
  ```

- **Failure** (`401`):

```
{
  "error": "Invalid refresh token"
}
```

User Management Routes

4. Get All Users

Endpoint: `/api/users`
Method: `GET`
Access: Public

Description: Fetches a list of all users in the system.

Response:

- **Success** (`200`):

```
[
  {
    "name": "Ahmed Mohamed",
    "email": "ahmed.mohamed@example.com",
    "role": "Member",
    "barcode": "USER12345"
  },
  {
    "name": "Mostafa Ali",
    "email": "mostafa.ali@example.com",
    "role": "Librarian",
    "barcode": "USER67890"
  }
]
```

- **Failure** (`500`):

```
{
  "error": "Database error occurred"
}
```

---

5. Get User Details by Barcode

Endpoint: `/api/users/barcode/<barcode>`
Method: `GET`
Access: Public

Description: Fetches details of a specific user by their barcode.

Response:

- **Success** (`200`):

```
{
  "name": "Ahmed Mohamed",
```

```
  "email": "ahmed.mohamed@example.com",
  "role": "Member",
  "barcode": "USER12345"
}
```

- **Failure** (`404`):

```
{
  "error": "User not found"
}
```

---

6. Delete a User

Endpoint: `/api/users`
Method: `DELETE`
Access: Public

Description: Deletes a user by their barcode.

Request Body:

```
{
  "barcode": "USER12345"
}
```

Response:

- **Success** (`200`):

```
{
  "message": "User deleted successfully"
}
```

- **Failure** (`404`):

```
{
  "error": "User not found"
}
```

- **Failure** (`400`):

```
{
  "error": "Cannot delete a Librarian"
}
```

Notes:

- Librarians cannot be deleted through this endpoint.
- The barcode must be provided in the request body.

---

7. Get User Notifications

Endpoint: `/api/users/<barcode>/notifications`
Method: `GET`

Access: Requires JWT

Description: Fetches notifications for a specific user.

Response:

- **Success** (`200`):

```
{
  "notifications": [
    {
      "id": 1,
      "subject": "Overdue Book",
      "body": "The book 'The Great Gatsby' is overdue."
    },
    {
      "id": 2,
      "subject": "Reservation Available",
      "body": "Your reserved book is now available."
    }
  ]
}
```

- **Failure** (`404`):

```
{
  "error": "User not found"
}
```

Book Management Routes

10. Get All Books

Endpoint: `/api/books`
Method: `GET`
Access: Requires JWT

Description: Fetches a list of all books in the catalog.

Response:

- **Success** (`200`):

```
{
  "books": [
    {
      "id": 1,
      "barcode": "BOOK12345",
      "title": "The Great Gatsby",
      "author": "F. Scott Fitzgerald",
      "subject_category": "Fiction",
      "publication_date": "1925-04-10",
      "is_available": true
    },
```

```
    {
      "id": 2,
      "barcode": "BOOK67890",
      "title": "To Kill a Mockingbird",
      "author": "Harper Lee",
      "subject_category": "Fiction",
      "publication_date": "1960-07-11",
      "is_available": false
    }
  ]
}
```

- **Failure** (`500`):

```
{
  "error": "An error occurred while fetching books"
}
```

Notes:

- The `is_available` field indicates whether any copies of the book are currently available for borrowing.

---

11. Get Book by Barcode

Endpoint: `/api/books/<barcode>`
Method: `GET`
Access: Requires JWT

Description: Fetches details of a specific book by its barcode.

Response:

- **Success** (`200`):

```
{
  "id": 1,
  "barcode": "BOOK12345",
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "subject_category": "Fiction",
  "publication_date": "1925-04-10",
  "available_copies": 2,
  "total_copies": 3
}
```

- **Failure** (`404`):

```
{
  "error": "Book not found"
}
```

---

12. Add a New Book

Endpoint: `/api/books`
Method: `POST`
Access: Librarian

Description: Adds a new book to the catalog.

Request Body:

```
{
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "subject_category": "Fiction",
  "publication_date": "1925-04-10"
}
```

Response:

- **Success** (`201`):

  ```
  {
    "message": "Book added successfully",
    "barcode": "BOOK12345"
  }
  ```

- **Failure** (`400`):

  ```
  {
    "error": "Invalid input data"
  }
  ```

Notes:

- The system automatically generates a unique barcode for each new book.
- The publication date format should be YYYY-MM-DD.

---

13. Update a Book

Endpoint: `/api/books/<barcode>`
Method: `PUT`
Access: Librarian

Description: Updates information for an existing book.

Request Body:

```
{
  "title": "Updated Title",
  "author": "Updated Author",
  "subject_category": "Updated Category",
  "publication_date": "2020-01-01"
}
```

Response:

- **Success** (`200`):

```
{
  "message": "Book updated successfully"
}
```

- **Failure** (`404`):

```
{
  "error": "Book not found"
}
```

- **Failure** (`400`):

```
{
  "error": "Invalid input data"
}
```

Notes:

- You only need to include the fields you want to update.
- The barcode cannot be changed once assigned.

---

14. Delete a Book

Endpoint: `/api/books/<barcode>`
Method: `DELETE`
Access: Librarian

Description: Deletes a book and all its copies from the catalog.

Response:

- **Success** (`200`):

```
{
  "message": "Book deleted successfully"
}
```

- **Failure** (`404`):

```
{
  "error": "Book not found"
}
```

- **Failure** (`400`):

```
{
  "error": "Cannot delete a book that has checked-out copies"
}
```

Notes:

- This operation will also delete all copies of the book.
- Books with currently checked-out copies cannot be deleted.

Book Copies Management

15. Add a Book Copy

Endpoint: `/api/book_copies/<barcode>`
Method: `POST`
Access: Librarian

Description: Adds a new copy of an existing book.

Request Body:

```
{
  "rack_location": "A1"
}
```

Response:

- **Success** (`201`):

  ```
  {
    "message": "Book copy added successfully",
    "id": 3
  }
  ```

- **Failure** (`404`):

  ```
  {
    "error": "Book not found"
  }
  ```

- **Failure** (`400`):

  ```
  {
    "error": "Invalid rack location"
  }
  ```

Notes:

- The `rack_location` specifies where the physical book can be found in the library.
- New book copies are automatically marked as available.

---

16. Get All Copies of a Book

Endpoint: `/api/book_copies/<barcode>`
Method: `GET`
Access: Librarian

Description: Fetches all copies of a specific book.

Response:

- **Success** (`200`):

```
{
  "book_copies": [
    {
      "id": 1,
      "rack_location": "A1",
      "is_available": true
    },
    {
      "id": 2,
      "rack_location": "B3",
      "is_available": false
    }
  ]
}
```

- **Failure** (`404`):

```
{
  "error": "Book not found"
}
```

Notes:

- The `is_available` field indicates whether the copy is currently available for borrowing.

---

17. Update a Book Copy

Endpoint: `/api/book_copies/<barcode>/<copy_id>`
Method: `PUT`
Access: Librarian

Description: Updates information for a specific book copy.

Request Body:

```
{
  "rack_location": "C4",
  "is_available": false
}
```

Response:

- **Success** (`200`):

```
{
  "message": "Book copy updated successfully"
}
```

- **Failure** (`404`):

```
{
  "error": "Book copy not found"
}
```

Notes:

- Setting `is_available` to false manually can be useful for marking damaged or lost copies.

---

18. Delete a Book Copy

Endpoint: `/api/book_copies/<barcode>/<copy_id>`
Method: `DELETE`
Access: Librarian

Description: Deletes a specific copy of a book.

Response:

- **Success** (`200`):

```
{
  "message": "Book copy deleted successfully"
}
```

- **Failure** (`404`):

```
{
  "error": "Book copy not found"
}
```

- **Failure** (`400`):

```
{
  "error": "Cannot delete a book copy that is currently checked out"
}
```

Notes:

- This endpoint is useful for removing damaged or lost copies from the system.
- You cannot delete a copy that is currently checked out.

Search Routes

19. Search for Books

Endpoint: `/api/search/books`
Method: `GET`
Access: Public

Description: Searches for books based on various criteria.

Query Parameters:

- `title` (optional): Search by title
- `author` (optional): Search by author
- `subject_category` (optional): Search by subject category
- `publication_date` (optional): Search by publication date

Example Request:

```
GET /api/search/books?title=Great&author=Fitzgerald
```

Response:

- **Success** (`200`):

```
{
  "books": [
    {
      "id": 1,
      "barcode": "BOOK12345",
      "title": "The Great Gatsby",
      "author": "F. Scott Fitzgerald",
      "subject_category": "Fiction",
      "publication_date": "1925-04-10",
      "available_copies": 2,
      "total_copies": 3
    }
  ]
}
```

- **No Results** (`200`):

```
{
  "books": []
}
```

Notes:

- The search is case-insensitive and performs partial matching.
- Multiple parameters create an AND condition (all criteria must match).
- If no parameters are provided, all books are returned.

Borrow Routes

20. Issue a Book

Endpoint: `/api/issue`
Method: `POST`
Access: Librarian

Description: Issue a book to a user (librarian-initiated checkout).

Request Body:

```
{
  "user_barcode": "USER12345",
  "book_barcode": "BOOK67890"
}
```

Response:

- **Success** (`201`):

```
{
  "message": "Book issued successfully",
  "transaction_id": 1,
  "due_date": "2025-03-28"
}
```

- **Failure** (`404`):

```
{
  "error": "User not found"
}
```

- **Failure** (`400`):

```
{
  "error": "User has reached the maximum number of books allowed (5)"
}
```

Notes:

- Only librarians can issue books to users.
- The system automatically selects an available copy of the book.
- Due date is set to 10 days from the issue date.
- Users can have a maximum of 5 books checked out at one time.

---

21. Return a Book

Endpoint: `/api/return/<transaction_id>`
Method: `PUT`
Access: Librarian

Description: Process a book return.

Response:

- **Success** (`200`):

```
{
  "message": "Book returned successfully",
  "transaction_id": 1
}
```

- **Success with Fine** (`200`):

```
{
  "message": "Book returned successfully, with a fine of $2.50",
  "transaction_id": 1,
  "fine_amount": 2.5
}
```

- **Failure** (`404`):

```
{
  "error": "Transaction not found"
}
```

Notes:

- Only librarians can process returns.
- Fines are calculated at $0.50 per day overdue.
- The system automatically marks the book copy as available upon return.
- Users with pending reservations for the book are automatically notified.

---

22. Reserve a Book

Endpoint: `/api/reserve`
Method: `POST`
Access: Requires JWT

Description: Reserve a book that is currently unavailable.

Request Body:

```
{
  "user_barcode": "USER12345",
  "book_barcode": "BOOK67890"
}
```

Response:

- **Success** (`201`):

  ```
  {
    "message": "Book reserved successfully",
    "reservation_id": 1
  }
  ```

- **Failure** (`400`):

  ```
  {
    "error": "This book is currently available, no need to reserve"
  }
  ```

- **Failure** (`400`):

  ```
  {
    "error": "You already have a pending reservation for this book"
  }
  ```

Notes:

- Reservations can only be made for books that have no available copies.
- Users are notified when a reserved book becomes available.
- Users cannot reserve the same book multiple times.

---

23. Cancel Reservation

Endpoint: `/api/cancel-reservation/<reservation_id>`
Method: `PUT`
Access: Requires JWT

Description: Cancel a pending book reservation.

Response:

- **Success** (`200`):

```
{
  "message": "Reservation cancelled successfully"
}
```

- **Failure** (`404`):

```
{
  "error": "Reservation not found"
}
```

- **Failure** (`400`):

```
{
  "error": "This reservation is already cancelled"
}
```

Notes:

- Only pending reservations can be cancelled.
- Fulfilled or already cancelled reservations cannot be cancelled again.

---

24. Renew a Book

Endpoint: `/api/renew/<transaction_id>`
Method: `PUT`
Access: Requires JWT

Description: Extend the due date for a borrowed book.

Response:

- **Success** (`200`):

```
{
  "message": "Book renewed successfully",
  "new_due_date": "2025-04-07"
}
```

- **Failure** (`400`):

```
{
  "error": "Overdue books cannot be renewed"
}
```

- **Failure** (`404`):

```
{
  "error": "Transaction not found"
}
```

Notes:

- Renewal extends the due date by 10 days from the current due date.
- Overdue books cannot be renewed.
- Books that have already been returned cannot be renewed.

---

25. Check Overdue Books

Endpoint: `/api/overdue-books`
Method: `GET`
Access: Librarian

Description: Retrieve a list of all overdue books and calculate fines.

Response:

- **Success** (`200`):

```
{
  "overdue_books": [
    {
      "transaction_id": 1,
      "user_name": "Ahmed Mohamed",
      "user_email": "ahmed@example.com",
      "book_title": "The Great Gatsby",
      "days_overdue": 5,
      "fine_amount": 2.5,
      "due_date": "2025-03-15"
    },
    {
      "transaction_id": 2,
      "user_name": "Mostafa Ali",
      "user_email": "mostafa@example.com",
      "book_title": "To Kill a Mockingbird",
      "days_overdue": 3,
      "fine_amount": 1.5,
      "due_date": "2025-03-17"
    }
  ]
}
```

Notes:

- Only librarians can access this endpoint.
- Fines are calculated at $0.50 per day overdue.
- The system automatically generates notifications for users with overdue books.

---

26. Checkout Book

Endpoint: `/api/checkout`
Method: `POST`
Access: Member

Description: Allow members to check out a book for themselves.

Request Body:

```
{
  "user_barcode": "USER12345",
  "book_barcode": "BOOK67890"
}
```

Response:

- **Success** (`201`):

  ```
  {
    "message": "Book issued successfully",
    "transaction_id": 1,
    "due_date": "2025-03-28"
  }
  ```

- **Failure** (`403`):

  ```
  {
    "error": "Unauthorized: Member role required"
  }
  ```

Notes:

- Only members can use this endpoint.
- Members can only check out books for themselves.
- Due date is set to 10 days from the checkout date.
- Members can have a maximum of 5 books checked out at one time.

---

27. Get Borrowing History

Endpoint: `/api/users/<user_id>/borrowing-history`
Method: `GET`
Access: Requires JWT

Description: Retrieve the complete borrowing history for a specific user.

Response:

- **Success** (`200`):

  ```
  {
    "borrowing_history": [
      {
        "transaction_id": 1,
        "book_title": "The Great Gatsby",
  ```

```
      "checkout_date": "2025-03-01",
      "due_date": "2025-03-11",
      "return_date": "2025-03-10",
      "fine_amount": 0
    },
    {
      "transaction_id": 2,
      "book_title": "To Kill a Mockingbird",
      "checkout_date": "2025-03-05",
      "due_date": "2025-03-15",
      "return_date": null,
      "fine_amount": 0
    }
  ]
}
```


- **Failure** (`404`):


```
{
  "error": "User not found"
}
```


Notes:

- The `return_date` field is `null` for books that are still checked out.
- History includes both current and past transactions.

---

28. Get Checked-Out Books

Endpoint: `/api/users/<user_id>/checked-out-books`
Method: `GET`
Access: Requires JWT

Description: Retrieve the list of books currently checked out by a specific user.

Response:

- **Success** (`200`):


```
{
  "checked_out_books": [
    {
      "transaction_id": 2,
      "book_title": "To Kill a Mockingbird",
      "checkout_date": "2025-03-05",
      "due_date": "2025-03-15"
    },
    {
      "transaction_id": 3,
      "book_title": "1984",
      "checkout_date": "2025-03-07",
      "due_date": "2025-03-17"
    }
```

```
  ]
}
```

- **Failure** (`404`):

```
{
  "error": "User not found"
}
```

Notes:

- This endpoint only returns books that have not been returned yet.
- Users can use this to track their current borrowings and due dates.

General Notes

1. **Authentication**:
   - Most endpoints require authentication using JWT tokens.
   - Include the token in the Authorization header: `Authorization: Bearer <token>`
   - Librarian-specific endpoints require the user to have the Librarian role.
2. **Error Handling**:
   - All error responses include an `error` field with a descriptive message.
   - Common HTTP status codes:
     - `200 OK` : Request was successful
     - `201 Created` : Resource was created successfully
     - `400 Bad Request` : Invalid request data
     - `401 Unauthorized` : Missing or invalid authentication
     - `403 Forbidden` : Insufficient permissions
     - `404 Not Found` : Resource not found
     - `500 Internal Server Error` : Server-side error
3. **Date Formats**:
   - All dates should be in ISO format: YYYY-MM-DD
4. **Barcodes**:
   - Book barcodes are automatically generated by the system
   - User barcodes are also system-generated
   - Barcodes are used as unique identifiers for books and users

---

Summary of Library Management System API Endpoints

We have a total of **26 endpoints** across 6 main categories:

Authentication Routes (3 endpoints)

1. Register a User - POST `/auth/register`
2. Login - POST `/auth/login`
3. Refresh Token - POST `/auth/refresh`

User Management Routes (4 endpoints)

4. Get All Users - GET `/api/users`
5. Get User Details by Barcode - GET `/api/users/barcode/<barcode>`
6. Delete a User - DELETE `/api/users`
7. Get User Notifications - GET `/api/users/<barcode>/notifications`

Book Management Routes (5 endpoints)

8. Get All Books - GET `/api/books`
9. Get Book by Barcode - GET `/api/books/<barcode>`
10. Add a New Book - POST `/api/books`
11. Update a Book - PUT `/api/books/<barcode>`
12. Delete a Book - DELETE `/api/books/<barcode>`

Book Copies Management (4 endpoints) - in the same file of Book Management Routes

13. Add a Book Copy - POST `/api/book_copies/<barcode>`
14. Get All Copies of a Book - GET `/api/book_copies/<barcode>`
15. Update a Book Copy - PUT `/api/book_copies/<barcode>/<copy_id>`
16. Delete a Book Copy - DELETE `/api/book_copies/<barcode>/<copy_id>`

Search Routes (1 endpoint)

17. Search for Books - GET `/api/search/books`

Borrow Routes (9 endpoints)

18. Issue a Book - POST `/api/issue`
19. Return a Book - PUT `/api/return/<transaction_id>`
20. Reserve a Book - POST `/api/reserve`
21. Cancel Reservation - PUT `/api/cancel-reservation/<reservation_id>`
22. Renew a Book - PUT `/api/renew/<transaction_id>`
23. Check Overdue Books - GET `/api/overdue-books`
24. Checkout Book - POST `/api/checkout`
25. Get Borrowing History - GET `/api/users/<user_id>/borrowing-history`
26. Get Checked-Out Books - GET `/api/users/<user_id>/checked-out-books`

This comprehensive set of endpoints covers all the essential functionality for a library management system, including user authentication, book management, borrowing operations, search functionality, and advanced borrowing features like renewals and reservations.