Library Borrowing Tracker

Final Report dedicated for Database Group Project

Group 7

Group Members:

Farrel Tsaqif Anindyo	(24/536735/PA/22769)
Faiq Athaya Urumsah	(24/546678/PA/23211)
Akhtar Gadang Abimanyu	(24/532855/PA/22547)

December 20, 2025

Contents

1 Introduction

1.1 Problem Background

Many traditional libraries still rely on manual logging methods or outdated spreadsheet systems to manage book loans. This manual approach leads to significant issues, including duplicate records, lost data, difficulty in tracking overdue books, and a general lack of visibility regarding book availability. Librarians often struggle to manage inventory efficiently, while borrowers lack a convenient way to check book status or history.

1.2 Objectives

The primary objective of the Library Borrowing Tracker is to digitize and automate core library operations. The specific goals are:

- To create a centralized relational database for storing book, user, and transaction data.
- To provide an interface for Borrowers to browse catalogs and request loans.
- To provide an interface for Librarians to manage inventory, approve requests, and process returns.
- To ensure data integrity through proper database normalization and constraints.

1.3 Users and Use Cases

The system is designed for two distinct user roles:

- **Borrower (Student/User):** Can register, log in, view available books, request to borrow books, and view their active loan status.
- **Librarian (Admin):** Can add new books, remove books, approve borrowing requests (setting loan duration), and process book returns.

2 Database Design

2.1 Entity Relationship Diagram (ERD)

The conceptual design of the database focuses on the relationships between Borrowers, Books, and Librarians.

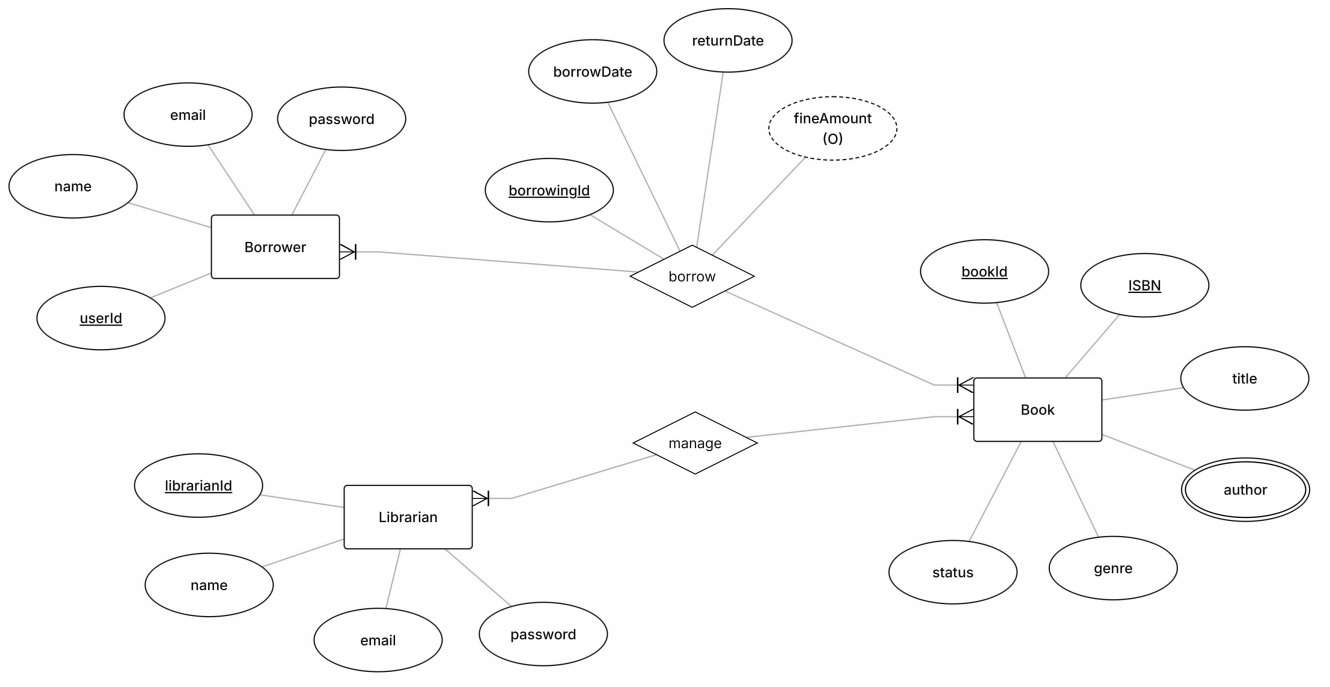

Figure 1: Entity Relationship Diagram (ERD)

2.2 Entity Explanations

- **Borrower:** Stores user credentials. Attributes: `userID` (PK), `borrowerName`, `borrowerEmail`, `borrowerPass`.
- **Book:** Represents the library inventory. Attributes: `bookID` (PK), `ISBN`, `bookTitle`, `bookGenre`, `bookStatus` (e.g., AVAILABLE, BORROWED).
- **Librarian:** Stores admin credentials. Attributes: `librarianID` (PK), `libName`, `libEmail`, `libPass`.
- **Borrowing (Transaction):** An associative entity tracking the loan process. Attributes: `borrowID` (PK), `borrowDate`, `returnDate`, `status` (PENDING, APPROVED, RETURNED), `fineAmount`.

2.3 Normalization Steps

To ensure the database is efficient and free of redundancy:

- **1NF (First Normal Form):** All attributes contain atomic values (e.g., no multiple authors in a single cell). Each record is unique.
- **2NF (Second Normal Form):** All non-key attributes are fully dependent on the primary key. We separated "Borrowing History" into its own table so that transaction data doesn't clutter the User or Book tables.

- **3NF (Third Normal Form):** Transitive dependencies were removed. For example, book details are tied strictly to `bookID`, not to the borrower who currently has the book.

3 Database Implementation

3.1 Relational Schema

The physical implementation of the database in MySQL follows this schema structure.

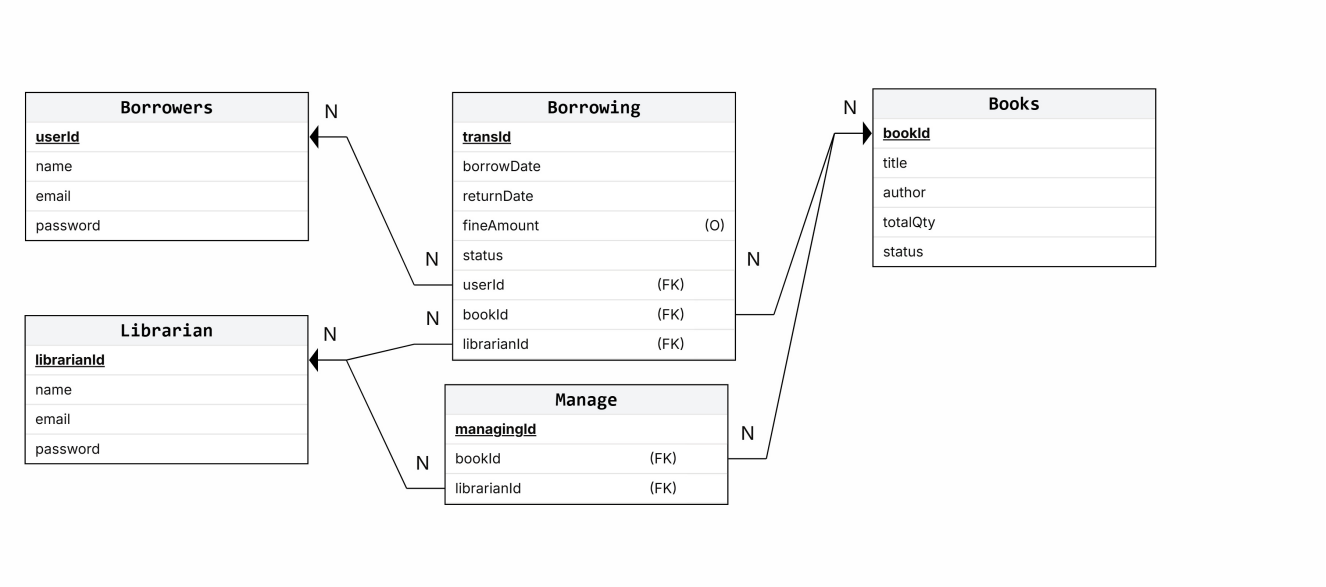

Figure 2: Relational Schema

3.2 Key Constraints and SQL

The system utilizes Primary Keys (PK) for unique identification and Foreign Keys (FK) to link the Borrow table to Borrower and Book.

Example SQL for Table Creation:

```sql
CREATE DATABASE librarydb;
USE librarydb;

CREATE TABLE Borrower (
  userID INT NOTNULL AUTO_INCREMENT PRIMARY KEY,
  borrowerName VARCHAR(100) NOT NULL,
  borrowerEmail VARCHAR(100) NOT NULL UNIQUE,
  borrowerPass VARCHAR(255) NOT NULL
);

```

```sql
CREATE TABLE Librarian (
  librarianID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  librarianName VARCHAR(100) NOT NULL,
  librarianEmail VARCHAR(100) NOT NULL UNIQUE,
  librarianPass VARCHAR(255) NOT NULL
);

CREATE TABLE Book (
  bookID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  bookISBN VARCHAR(20) NOT NULL,
  bookTitle VARCHAR(255) NOT NULL,
  bookAuthor VARCHAR(255) NOT NULL,
  bookGenre VARCHAR(100) NOT NULL,
  bookStatus ENUM('available', 'pending', 'borrowed') DEFAULT '
    available',
  totalCopies INT DEFAULT 1,
  availableCopies INT DEFAULT 1
);

CREATE TABLE Borrowing (
  borrowID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  borrowDate DATE NOT NULL,
  returnDate DATE NOT NULL,
  fine DECIMAL(10,2) DEFAULT 0.00,
  bookID INT,
  userID INT,
  librarianID INT,
  dueDate DATE,
  status ENUM('pending', 'approved', 'return_requested', 'returned')
    DEFAULT 'pending',
  FOREIGN KEY (bookID) REFERENCES Book(bookID),
  FOREIGN KEY (userID) REFERENCES Borrower(userID) ON DELETE CASCADE,
  FOREIGN KEY (librarianID) REFERENCES Librarian(librarianID) ON DELETE
    SET NULL
);
```

Listing 1: Creating the Borrow Transaction Table

Stored Procedures/Logic: When a book is requested, the system inserts a record into `borrowing_history` with a status of 'pending'. When approved by a librarian, the books table updates the `bookStatus` to 'UNAVAILABLE'.

Explanation: The provided SQL script successfully establishes the physical schema for the Library Borrowing Tracker by creating four normalized tables: Borrower, Librarian, Book, and Borrowing. The Borrower and Librarian tables manage user and staff credentials, while the Book table tracks inventory, including the crucial availableCopies and the book's bookStatus (available, pending, or borrowed). The central Borrowing ta-

ble connects these entities using Foreign Keys (bookID, userID, librarianID), maintaining the transaction details such as borrowDate, dueDate, fine amount, and the current status of the loan (pending, approved, return requested, or returned), with constraints ensuring data integrity during deletion, such as setting the associated librarianID to NULL if a librarian is removed.

4 Application Implementation

The application uses a Node.js/Express backend connected to a MySQL database, with a frontend interface for user interaction.

4.1 Borrower Dashboard

Borrowers can view the catalog and see the status of their requests.

- **Logic:** The user clicks "Borrow," which sends a POST request to the API. The status initially shows as "PENDING."

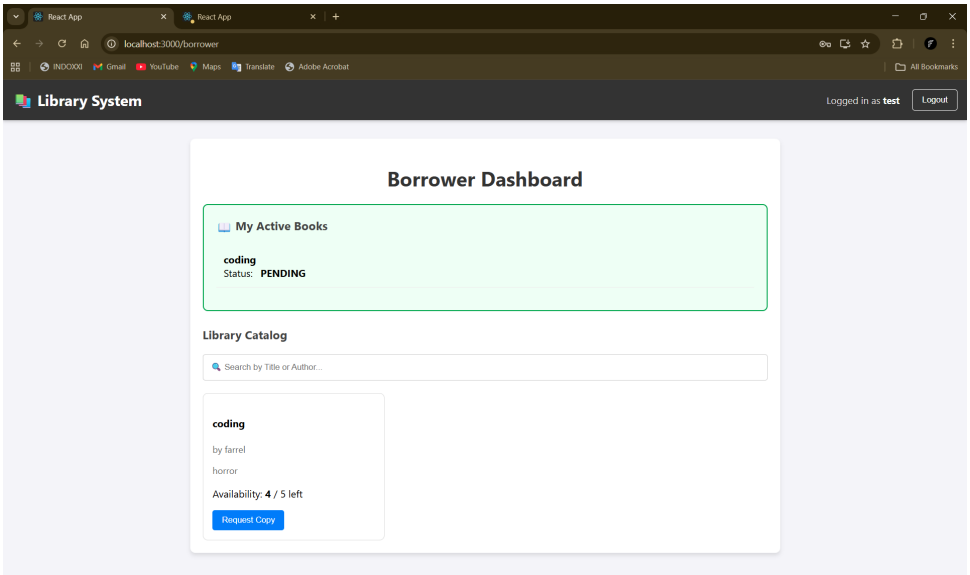

Figure 3: Borrower Dashboard showing a Pending Request

4.2 Librarian Dashboard (Approvals)

Librarians view a list of pending requests. They can approve a request and set the borrowing duration (e.g., 7 days).

- **Logic:** Upon approval, the `returnDate` is calculated, and the book status is updated in the database.

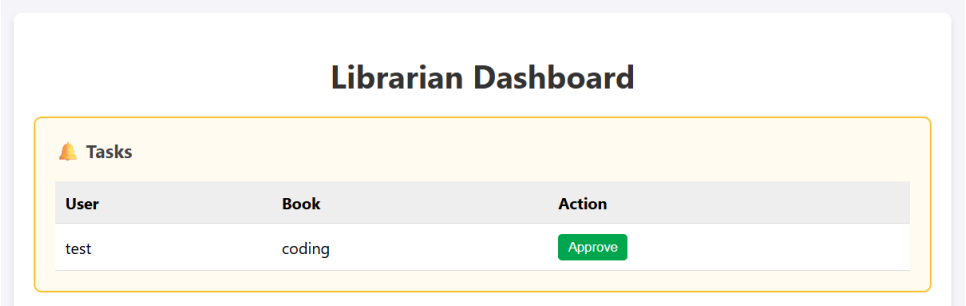

Figure 4: Librarian Approval Interface

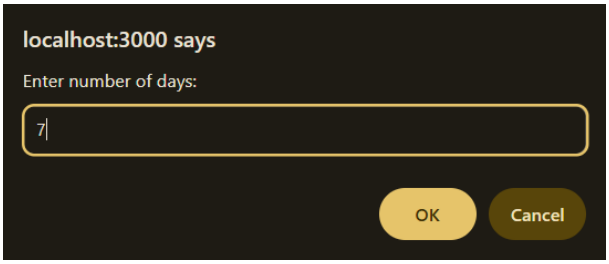

Figure 5: calculate the returnDate

4.3 Return Process

There are two ways to return the book. First, the librarian can enter the `borrowID` to process a return. Second, the borrower can request the return through their dashboard, and then the librarian can approve the request. This updates the transaction to "returned" and makes the book "AVAILABLE" again in the catalog. If the 'returnDate' is past the 'dueDate', the fine will be calculated.

4.3.1 Librarian Dashboard

- **Librarian Return Dashboard:**

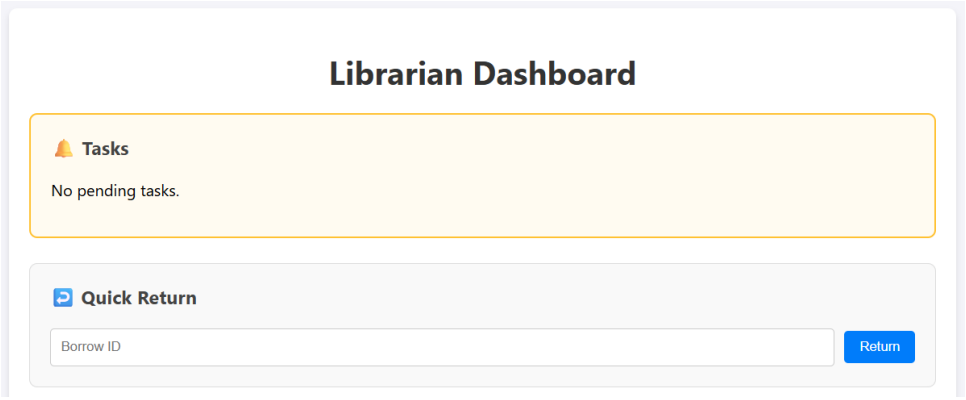

Figure 6: Librarian Dashboard

	borrowID	borrowDate	returnDate	fine	bookID	userID	librarianID	dueDate	status
▶	1	2025-12-03	2025-12-24	70.00	1	1	1	2025-12-10	returned
	2	2025-12-24	NULL	0.00	1	1	1	2025-12-31	approved

Figure 7: Check the borrowID

- **Processing 'Return' by entering the borrowID:**

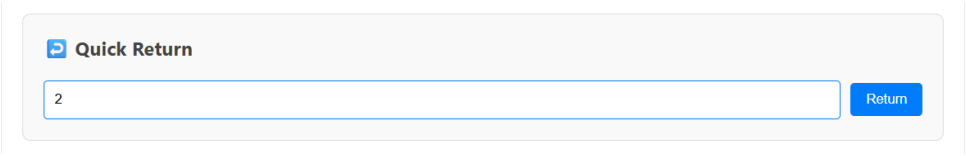

Figure 8: Enter borrowID to 'Return'

- **After the book is returned, the database will be updated**

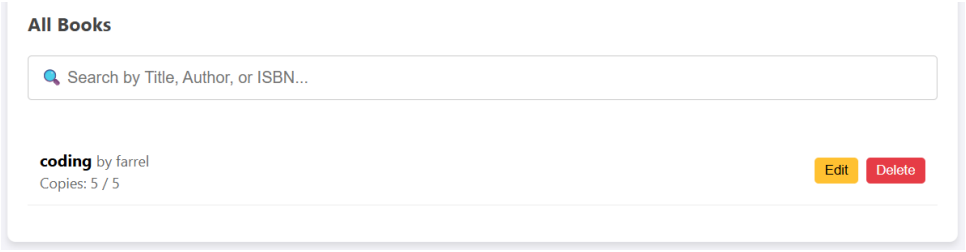

Figure 9: Number of copies will be increased by 1 after returning

4.3.2 Borrower Dashboard

- **Borrower Return Dashboard:**

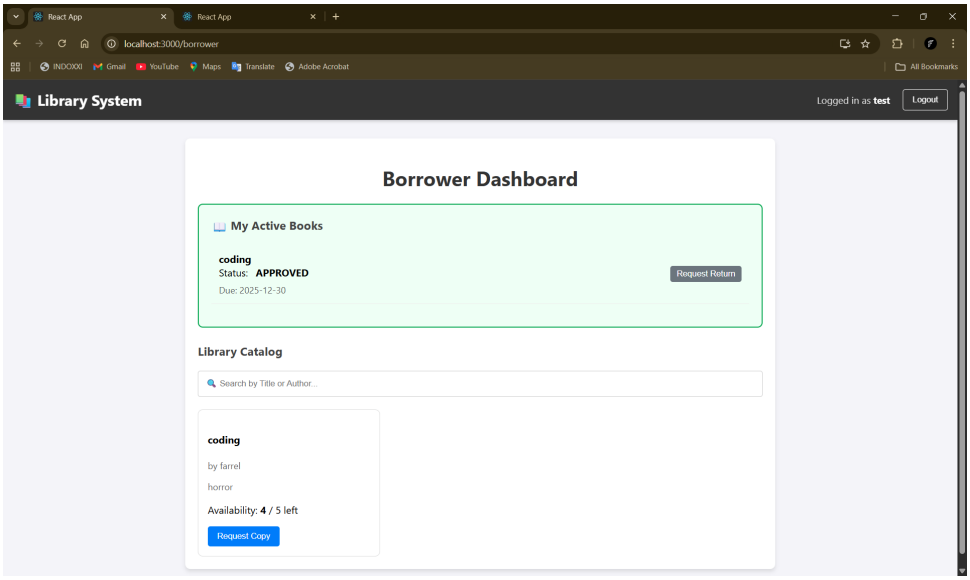

Figure 10: Borrower Dashboard

- **Processing 'Return' by request the return**

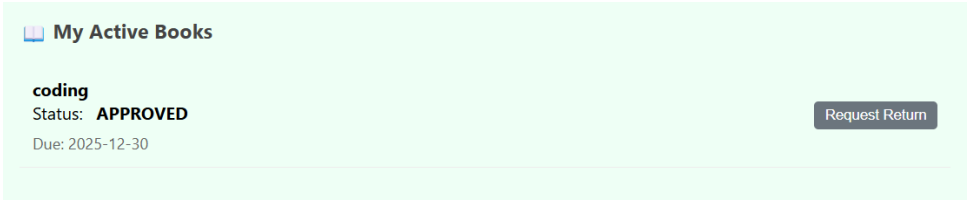

Figure 11: request the return

- **Approve the return request**

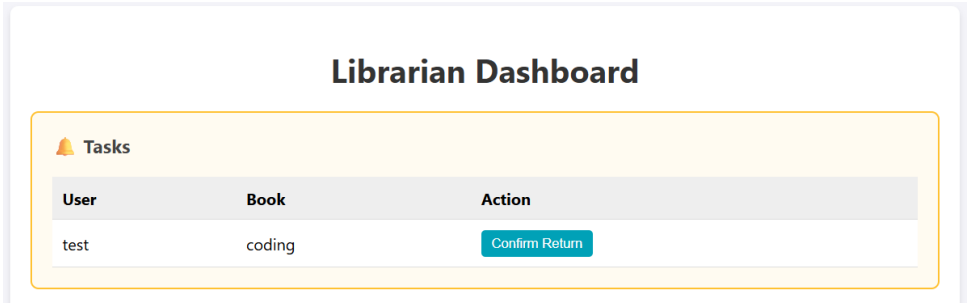

Figure 12: Librarian approves the return request

5 Testing and Results

We performed CRUD (Create, Read, Update, Delete) testing using Postman to verify the backend API before integrating it with the frontend.

5.1 User Registration (CREATE)

Tested the `POST /borrower/register` endpoint.

- **Input:** JSON containing name, email, and password.
- **Result:** 200 OK, "Borrower registered."

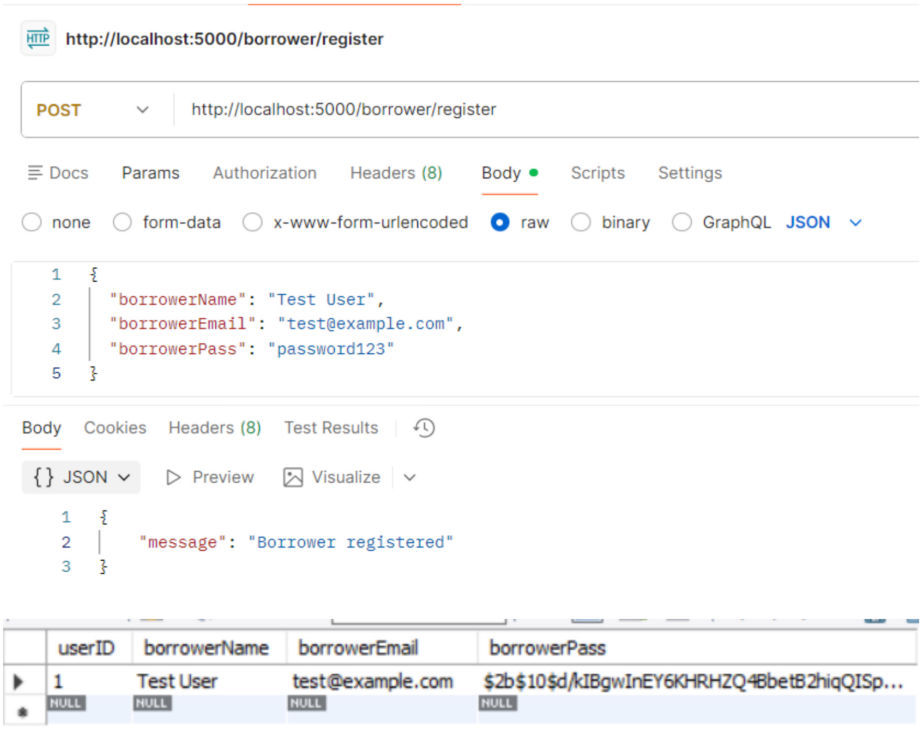

Figure 13: API Testing - Register Borrower

- **Register Librarian and Login:** A librarian, "Boss," was registered and logged in to acquire an authorization token required for privileged actions.

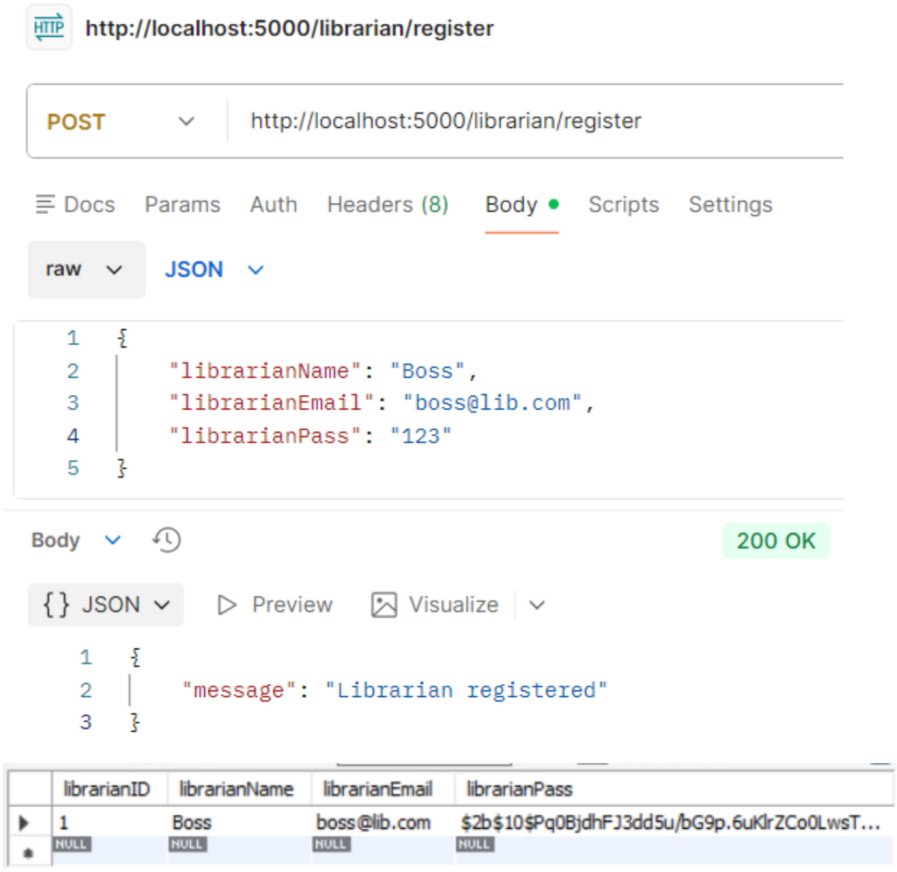

Figure 14: API Testing - Register Librarian

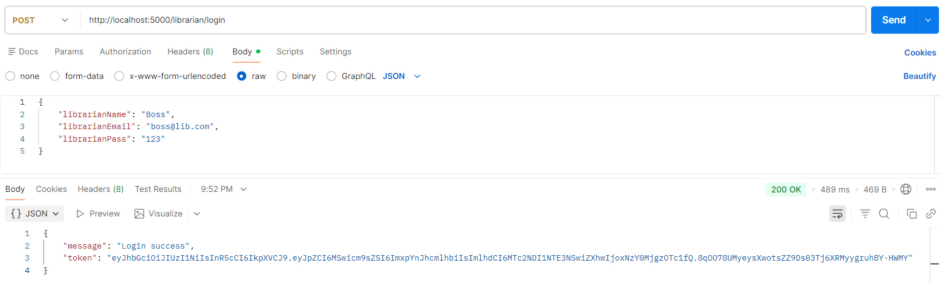

Figure 15: API Testing - Librarian Login

- **Add Book:** A book titled "Database Introduction" was successfully added using the librarian's token.

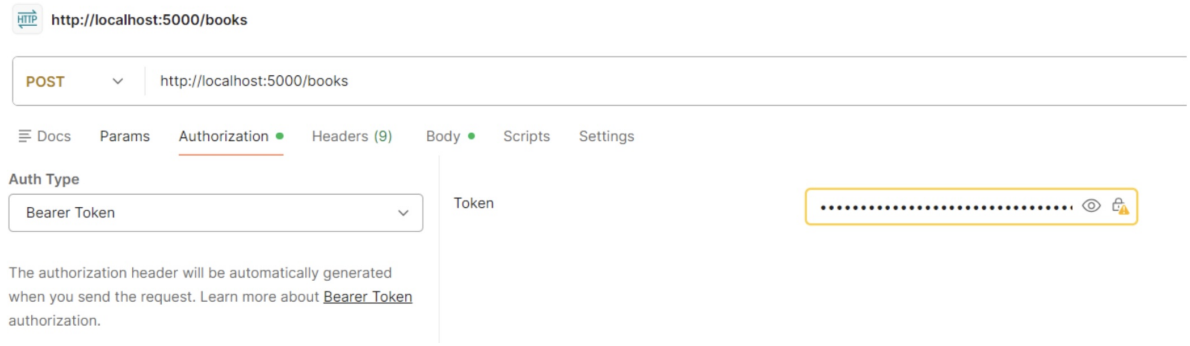

Figure 16: API Testing - Librarian's token to add book

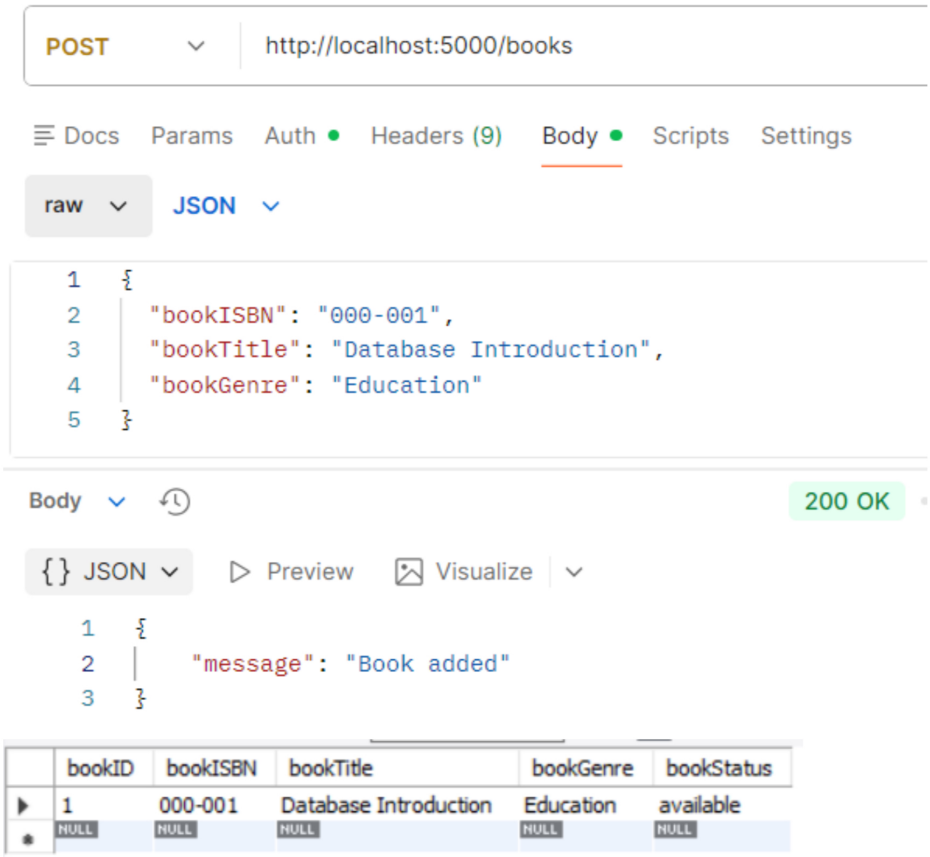

Figure 17: API Testing - Add book

5.2 Get Books (READ)

This operation tests the system's ability to retrieve and display current library inventory.

- **Endpoint:** GET http://localhost:5000/books.
- **Result:** A 200 OK response was received, returning a JSON array containing the newly added book, confirming it is in the database. The output showed:
  - bookID: 1
  - bookTitle: "Database Introduction"
  - bookStatus: "available"

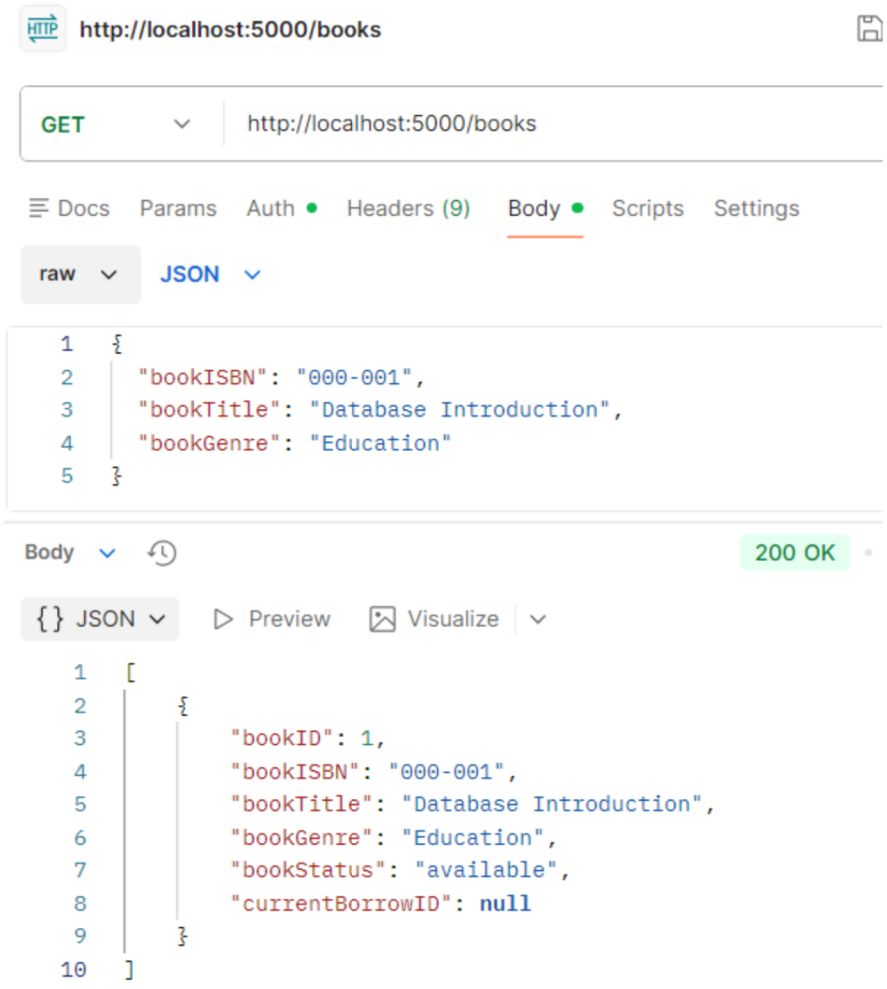

Figure 18: Read books information

5.3 Borrowing Flow (UPDATE)

Tested the lifecycle of a book loan.

- **Request:** `POST /borrow` → Message: "Request sent."

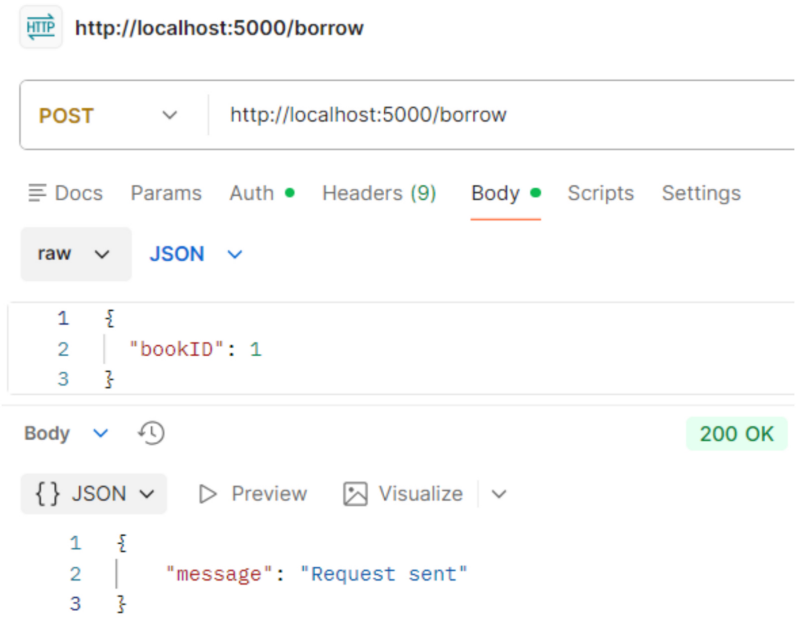

Figure 19: API Testing - Borrow request sent

- **Approve:** `POST /borrow/approve` → Input: `borrowID`, `days`. → Message: "Approved."

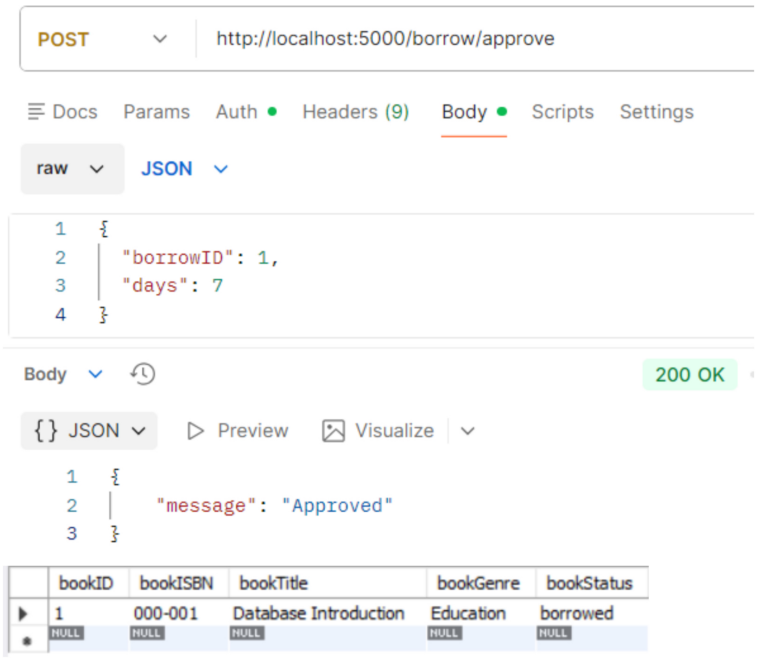

Figure 20: API Testing - Borrow request approved

5.4 Deleting a Book and its History (DELETE)

This operation tests the deletion functionality, ensuring relational integrity via constraints.

- Endpoint: DELETE http://localhost:5000/books/1 (requires the librarian's Bearer Token).
- Logic: The API endpoint handles the deletion of the book from the Book table and, due to the foreign key constraint on the Borrowing table, it also deletes the associated borrowing history.
- Result: A 200 OK response was returned with the message: "Book and history deleted".
- Verification: Post-deletion, a check confirmed that the book record (ID 1) no longer exists in the database.

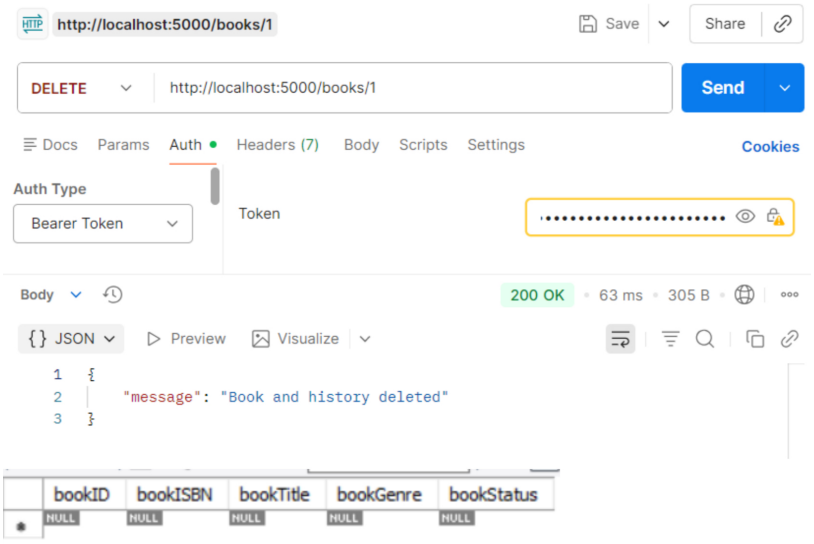

Figure 21: API Testing - Book and history succesfully deleted

6 Conclusion and Reflection

This project successfully implemented a functional Library Borrowing Tracker for conventional libraries. We learned valuable lessons in integrating a relational MySQL database with a Node.js backend, specifically regarding the handling of asynchronous queries and managing Foreign Key constraints (e.g., ensuring a book cannot be deleted if it is currently borrowed).

Challenges & Improvements: One challenge was synchronizing the status updates between the Books table and the Borrowing table to ensure the catalog accurately reflects availability. In the future, we could improve the system by adding automated fine calculations based on the `returnDate` and implementing email notifications for overdue books. Overall, the system meets the core objectives of digitizing the library's workflow.

7 References

[1] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill Education, 2020.