LIBRARY MANAGEMENT SYSTEM

Project Report

Submitted to

STATE BOARD OF TECHNICAL EDUCATION AND TRAINING, ANDHRA PRADESH

In the fulfillment of the requirements for the award of the degree of

DIPLOMA IN

COMPUTER ENGINEERING

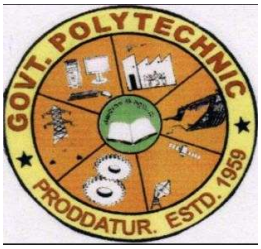
By

KANCHARLA SRINIVAS CHOWDARY	**(23022-CM-022)**
LAKKINENI NIKHIL	**(23022-CM-027)**
YALAGANI JAHNAVI	**(23022-CM-063)**
GADWALA SAHIFA BANU	**(23022-CM-012)**
RACHARLA SRAVAN	**(23184-CM-012)**
CHILLATHOTI ISWARYA	**(23022-CM-065)**
PITTU SIVAREDDY	**(23022-CM-046)**
C SIVARAJ	**(23022-CM-006)**
MALLELA LINGA SANTHOSH	**(23022-CM-029)**
NAKKA TIRUMALESH	**(23022-CM-039)**

Under The Esteemed Guidance of

T.NAVYA DURGA, B. Tech

BONAFIDE CERTIFICATE



Department of computer Engineering

This is to certify that the project entitled **LIBRARY MANAGEMENT SYSTEM** is a Bonafide work done by

KANCHARLA SRINIVAS CHOWDARY	**(23022-CM-022)**
LAKKINENI NIKHIL	**(23022-CM-027)**
YALAGANI JAHNAVI	**(23022-CM-063)**
GADWALA SAHIFA BANU	**(23022-CM-012)**
RACHARLA SRAVAN	**(23184-CM-012)**
CHILLATHOTI ISWARYA	**(23022-CM-065)**
PITTU SIVAREDDY	**(23022-CM-046)**
C. SIVARAJ	**(23022-CM-006)**
MALLELA LINGA SANTHOSH	**(23022-CM-029)**
NAKKA TIRUMALESH	**(23022-CM-039)**

In partial fulfillment of the requirements for the award of **Diploma in Computer Engineering** by the **Department of Technical Education and Training, Andhra Pradesh**, during the academic year **2023-2026** at **Government Polytechnic, Proddatur, Y.S.R Kadapa District**.

Project Guide
T.NAVYA DURGA, B.Tech.,

TIRUPATI-517501.

Head of the Department
M.VIJAYA KUMAR,M.Tech., HCMES
GPT,PRODDATUR-516360.

EXTERNAL EXAMINAR:

1)

2)

INTERNAL EXAMINAR:

1)

2)

Project Mentor :

DECLARATION

We hereby declare that the project report entitled "**LIBRARY MANAGEMENT SYSTEM**" done by us under the guidance of **T.NAVYA DURGA**, This report is the result of our effort and it has not been submitted to any other University or Institution for the award of any degree and diploma other than specified above.

DATE:
PLACE:

KANCHARLA SRINIVAS CHOWDARY	**(23022-CM-022)**
LAKKINENI NIKHIL	**(23022-CM-027)**
YALAGANI JAHNAVI	**(23022-CM-063)**
GADWALA SAHIFA BANU	**(23022-CM-012)**
RACHARLA SRAVAN	**(23184-CM-012)**
CHILLATHOTI ISWARYA	**(23022-CM-065)**
PITTU SIVAREDDY	**(23022-CM-046)**
C SIVARAJ	**(23022-CM-006)**
MALLELA LINGA SANTHOSH	**(23022-CM-029)**
NAKKA TIRUMALESH	**(23022-CM-039)**

ACKNOWLEDGEMENT

We are thankful to our guide **T.NAVYA DURGA,(B.Tech),** for her valuable guidance and encouragement. Her helping attitude and suggestions have helped in the successful completion of the project.

We would like to express our gratefulness and sincere thanks to **T.NAVYA DURGA(B.Tech),**for her kind help and encouragement during the course of study and in the successful completion of the project work.

We have a great pleasure in expressing our hearty thanks to our beloved **T.NAVYA DURGA(B.Tech),** for spending her valuable time with us to complete the project and providing all facilities in computer lab.

Successful completion of the project cannot be done without proper support and encouragement for providing all the necessary facilities computer lab during course of study.

We sincerely thank the all staff and students, project mates for their support.

By

KANCHARLA SRINIVAS CHOWDARY	**(23022-CM-022)**
LAKKINENI NIKHIL	**(23022-CM-027)**
YALAGANI JAHNAVI	**(23022-CM-063)**
GADWALA SAHIFA BANU	**(23022-CM-012)**
RACHARLA SRAVAN	**(23184-CM-012)**
CHILLATHOTI ISWARYA	**(23022-CM-065)**
PITTU SIVAREDDY	**(23022-CM-046)**
C SIVARAJ	**(23022-CM-006)**
MALLELA LINGA SANTHOSH	**(23022-CM-029)**
NAKKA TIRUMALESH	**(23022-CM-039)**

INTRODUCTION:

- The Library Management System (LMS) is a digital platform designed to simplify and automate the day-to-day operations of a library. The main goal of this project is to maintain detailed information about books, authors, and students while tracking all book transactions such as issue and return.
- In a traditional library, managing hundreds of books manually is time-consuming and prone to errors. This project addresses those challenges by creating a system where the librarian can easily add, update, and monitor books, and students can register, log in, and borrow or return books through their accounts. The system also includes an automated fine calculation feature that charges ₹5 per day for late returns**.**

FEATURES:

1. Student Module Features

1. **Student Registration:**
   Students can create an account using their email, name, and password to access the system.
2. **Login Authentication:**
   Only registered students can log in with valid credentials.
3. **Book Search:**
   Students can search for books by title, author, or category.
4. **Book Issue Request:**
   Students can request to borrow available books online.
5. **Return Request:**
   When students finish reading, they can click the *Return* button to submit a return request.
6. **View Issued Books:**
   Displays all the books currently borrowed by the student with issue and due dates.
7. **Fine Calculation:**
   Automatically calculates ₹5 per day if the book is not returned within the due date.
8. **Profile Management:**
   Students can update their details such as email or password.

2. Librarian Module Features

1. **Librarian Login:**
   Secure login for librarians to access administrative features.
2. **Book Management:**
   Add, edit, delete, and view details of all books (title, author, quantity, price, etc.).
3. **Student Management:**
   View and manage registered student accounts.
4. **Issue Books:**
   Approve student requests and issue books to them by entering issue and return dates.
5. **Return Confirmation:**
   Librarian confirms the return of books to ensure accountability.
6. **Fine Management:**
   View fines for late book returns and maintain records of paid/unpaid fines.
7. **Generate Reports:**
   Create reports on total books, issued books, returned books, and defaulters.
8. **Database Backup:**
   Backup and restore the database to prevent data loss.

3. System-Level Features

1. **Email Verification:**
   Students receive confirmation emails during registration or transactions.
2. **Role-Based Access:**
   Different access levels for librarians and students ensure security.
3. **Real-Time Updates:**
   Changes made by the librarian (like book availability) are instantly updated.
4. **Activity Logs:**
   All transactions (issue, return, fine) are recorded for transparency.
5. **Search and Filter:**
   Easy navigation with advanced filters for book title, author, or category.
6. **Responsive Design:**
   The system can be accessed from desktops, laptops, or tablets.
7. **Fine Tracking System:**
   Automatically tracks the number of days delayed and calculates total fine amount.

PROJECT REQUIREMENTS:

1. Hardware Requirements

1. Processor: Intel Core i5 or higher for smooth execution and testing.
2. RAM: Minimum 8 GB recommended for running server and database efficiently.
3. Storage: At least 256 GB SSD or higher to handle MongoDB data and project files.
4. Display: Standard monitor with 1366×768 resolution or higher.
5. Input Devices: Keyboard and mouse for development and testing.
6. Internet: Stable connection for online database tools and dependency downloads.

2. Software Requirements

1. **Operating System:** Windows 10 / 11 or any OS compatible with Java and MongoDB.
2. **Code Editor:** Visual Studio Code (VS Code) for writing and running frontend and backend code.
3. **Backend Technology:** Java using **Spring Boot Framework** (developed using **Spring Tool Suite** or VS Code).
4. **Database:** MongoDB used to store book details, student records, and transaction history.
5. **Frontend Technologies:** HTML, CSS, and JavaScript for user interface and client-side interactions.
6. **Build Tool:** Maven or Gradle for dependency management and project builds.
7. **API Testing Tool:** Postman for testing RESTful APIs.
8. **Version Control:** Git or GitHub for managing source code and version history.
9. **Browser:** Google Chrome or Microsoft Edge for running and testing the web application.

3. Functional Requirements

1. Students can register and log in using their email and password.
2. Librarians can log in with admin credentials to manage the entire system.
3. Librarian can add, edit, and delete book details.
4. Students can search for books by title, author, or category.
5. Students can request to issue books online through their account.

4. Non-Functional Requirements

1. **Performance:** The system should respond within 2–3 seconds for most operations.
2. **Security:** Passwords must be stored in encrypted form using secure hashing algorithms.
3. **Reliability:** The system must maintain accurate and consistent records without data loss.
4. **Usability:** The interface should be simple, clean, and user-friendly for both students and librarians.
5. **Scalability:** The system should handle an increasing number of users and books efficiently.
6. **Maintainability:** The code should be modular and well-documented for easy updates or fixes.
7. **Compatibility:** The application should work across modern browsers and devices.
8. **Backup:** MongoDB data should be backed up regularly to prevent accidental loss.

CHALLENGES:

1. Managing a large number of books manually without errors.
2. Ensuring that students actually return the books they borrow.
3. Maintaining accurate records of book issue and return dates.
4. Reducing dependency on the librarian's physical presence.
5. Avoiding duplicate book entries and data inconsistencies.

TECHNOLOGIES USED :

1. **HTML:** Used to design the structure of all web pages in the project.
2. **CSS:** Used to style the web pages and make them look attractive.
3. JavaScript: Used to make the web pages interactive and responsive.
4. **Java (Spring Boot):** Used as the backend programming language to handle all the logic and server operations.
5. **Spring Tool Suite (STS):** Used to develop and run the Java Spring Boot application easily.
6. **MongoDB:** Used as the database to store all book details, student information, and transaction records.
7. **VS Code:** Used as the main code editor for writing frontend and backend code.

BENEFITS :

1. Automation: The system reduces manual work by automating the issue, return, and fine calculation process.
2. Transparency: Both students and librarians can track transactions easily.
3. Security: Only registered users can log in using email and password.
4. Accountability: The return process is validated by the librarian, ensuring that books are not falsely marked as returned.
5. Time Efficiency: Quick book search, registration, and tracking save time for both students and librarians.

ADVANTAGES:

1. Easy Monitoring: Librarians can view all book and transaction details in one place.
2. Fine Calculation System: Automatically adds ₹5 per day for overdue books.
3. Digital Access: Students can request and manage books online.
4. Reduced Errors: Eliminates manual record-keeping errors.
5. Improved Communication: The system ensures both librarian and student confirmations for each transaction.

DISADVANTAGES:

1. Requires a stable internet or local server connection.
2. Installing and configuring the system may require time and resources.

EXPLANATION :

1. The Library Management System is a comprehensive web-based application designed to automate all operations of a library.
2. It allows librarians and students to perform their respective tasks efficiently, digitally, and securely.
3. The traditional library system requires manual record keeping, which is prone to errors, time-consuming, and difficult to maintain for a large number of books.
4. This project aims to provide a fully digital platform that tracks books, students, issue/return transactions, and fines.
5. The system ensures accuracy in all library operations while providing transparency for both students and librarians.
6. One of the main objectives is to **eliminate misuse or false claims of book returns** by implementing a **return confirmation system** verified by the librarian.
7. The system is built using **HTML, CSS, JavaScript** for frontend, **Java Spring Boot** for backend, and **MongoDB** for database management.
8. Students can access the system through a secure login, view available books, request issue, and request returns.
9. The librarian manages all the books, confirms issue and return requests, and monitors fines for overdue books.

Objectives of the Project

1. To automate library operations and replace manual record-keeping with a digital system.
2. To provide a secure platform for students and librarians to perform their functions.
3. To maintain proper records of every book, student, and transaction in the library.
4. To ensure accountability by confirming book returns through the librarian.
5. To calculate fines automatically for late returns at ₹5 per day.
6. To allow students to search, issue, and request returns online.
7. To provide real-time updates on book availability and issued books.
8. To create a reliable database where all library operations are logged for transparency.
9. To allow the librarian to generate detailed reports of issued books, returns, pending fines, and student activity.
10. To design a scalable system that can handle an increasing number of students and books.

Purpose of the System

1. The primary purpose is to simplify and automate library operations for efficiency and accuracy.
2. To prevent errors caused by manual record-keeping, such as misplaced books or duplicate entries.
3. To provide a **special feature** that ensures students cannot mark a book as returned

without librarian confirmation.

4. To track overdue books and calculate fines automatically, reducing dependency on manual calculations.
5. To provide students with easy access to library information, including available books, issue history, and due dates.
6. To allow librarians to manage the entire library system from a single dashboard.
7. To ensure transparency and accountability in all transactions between students and librarians.
8. To maintain detailed logs of all activities for future reference and auditing.

Working of the System

1. Student Registration: Students create an account with email and password.
2. Student Login: Secure login ensures only authorized users can access their accounts.
3. Librarian Login: Librarians use separate credentials for full administrative access.
4. Dashboard Access: Students and librarians see different dashboards with relevant features.
5. Add New Books: Librarians can add book details like title, author, category, quantity.
6. Edit/Delete Books: Librarians can update or remove outdated book records.
7. Book Search: Students can search books by title, author, or category.
8. Book Availability Check: The system displays whether the book is available for issue.
9. Issue Request by Student: Students request to borrow a book online.
10. Approval by Librarian: Librarian verifies availability and issues the book, recording the issue date.
11. Return Request by Student: After reading, the student clicks the return button.
12. Return Confirmation by Librarian: The book is only marked returned after librarian verification.
13. Fine Calculation: The system automatically calculates ₹5 per day if the book is overdue.
14. Update Database: All issued, returned, and fine transactions are recorded in MongoDB.
15. Report Generation: Librarian can generate reports of issued books, returned books, fines, and pending transactions.
16. Real-Time Updates: Book availability updates automatically when issued or returned.
17. Transaction History: Complete history of every action is stored for auditing and transparency.
18. Search and Filter Reports: Librarians can filter reports by student, book, or date.
19. Secure Access: Only authorized users can perform specific actions.
20. Notifications (Optional): Students can receive alerts about due dates or fines

SOURCE CODE

FRONTEND CODE:

LOGIN PAGE:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Login Form</title>
   <style>
      *{
   margin: 0;
   padding: 0;
   box-sizing: border-box;
}

body{
  width: 100vw;
  height: 100vh;
   display: flex;
   flex-direction: column;
   justify-content: center;
   align-items: center;
   font-family: Arial,sans-serif;
   background-image:
url(https://media.istockphoto.com/id/949118068/photo/books.jpg?s=612x612&w=0&k=20&c
=1vbRHaA_aOl9tLIy6P2UANqQ27KQ_gSF-BH0sUjQ730=);
   background-repeat: no-repeat;
   background-size: cover;
  background-position: center;
}
 .login-form{
      background: white;
      align-items: center;
      padding: 30px 30px;
      box-shadow: 0px 4px 15px slateblue;
```

```
    border-radius: 10px;
    width: 350px;
    height: 300px;
}

.login-form style{
 height: fit-content;
}
 h2{
  font-size: 35px;
  margin-bottom: 30px;
  color: white;
}
.input{
  padding-top: 15px;
  gap: 2rem;
}
.input input{
  width: 100%;
  padding: 0.6rem;
  margin: 10px 0;
  border-radius: 3px;
  border: 1px solid #ccc;
}
.button{
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
  margin: 1.5rem;
}

.button button{
  width: 110px;
  height: 30px;
  margin: 2px;
  border-radius: 3px;
  font-size: 18px;
}

.submit-btn{
```

```
  background-color: rgb(43, 43, 206);
  color: white;
  border-color: rgb(43, 43, 206);
}

.register-btn{
  border: 1px solid #ccc
}

.register-form{
    display: none;
    background: white;
    align-items: center;
    padding: 30px 40px;
    box-shadow: 0px 4px 15px rgba(0, 0, 0,2);
    border-radius: 10px;
    width: 380px;
    height: 350px;
}

.register-form h2{
  padding-left: 80px;
  color: black;
  font-size: 2rem;
}

.register-form input{
  width: 100%;
  padding: 10px;
  margin: 5px;
  border-radius: 3px;
  border: 1px solid #ccc;
}

.register-form button{
  width: 100%;
  padding: 5px;
  margin: 5px;
  font-size: large;
  border-radius: 3px;
  color: white;
```

```
  background-color: rgb(43, 43, 206);
  border-color: rgb(43, 43, 206);;
}

.register-form p{
  padding: 15px;
  margin: 5px;
}

.register-form a{
  text-decoration: none;
}

  </style>
</head>
<body>
  <h2>LibraryManagementSystem</h2>

  <!-- Login Form -->
  <div class="login-form" id="login-form">
    <form>
      <div class="input">
        <label for="email">Email:</label>
        <input type="email" name="email" id="email" placeholder="Enter email"
required><br>
        <label for="password">Password:</label>
        <input type="password" name="password" id="password" placeholder="Enter
password" required><br>
      </div>
      <div class="button" style="overflow: auto;">
        <button type="submit" class="submit-btn">Submit</button>
        <button type="button" onclick="showregister()" id="show-register">Register</button>
      </div>
    </form>
  </div>

  <!-- Register Form -->
  <div class="register-form" id="register-form" style="display: none;">
    <h2>Register</h2>
    <form>
      <input type="text" name="username" id="reg-username" placeholder="Username"
```

```
required><br>
        <input type="email" name="email" id="reg-email" placeholder="Email" required><br>
        <input type="password" name="password" id="reg-password" placeholder="Password"
required><br>

        <!-- CRITICAL: Role Selection sent to the Java backend -->
        <select name="role" id="reg-role" required>
          <option value="" disabled selected>Select Role</option>
          <option value="Student">Student</option>
          <option value="Librarian">Librarian</option>
        </select><br>

        <button type="submit">Register</button><br>
        <p>Already registered? <a href="#" id="showlogin" onclick="showlogin()">Login
Here</a></p>
      </form>
   </div>
  <script>
   //  Form toggle functions (critical fix)
function showregister() {
   document.getElementById('login-form').style.display = 'none';
   document.getElementById('register-form').style.display = 'block';
}

function showlogin() {
   document.getElementById('register-form').style.display = 'none';
   document.getElementById('login-form').style.display = 'block';
}

// Main logic
window.onload = function() {
   document.getElementById("register-form").style.display = "none";

   const loginForm = document.querySelector('#login-form form');
   const registerForm = document.querySelector('#register-form form');
   const API_BASE = 'http://localhost:8080/api/auth';

   //  LOGIN EVENT
   loginForm.addEventListener('submit', async (e) => {
      e.preventDefault();
      const email = document.querySelector('#login-form input[name="email"]').value.trim();
```

```
  const password = document.querySelector('#login-form input[name="password"]').value;

  try {
    const res = await fetch(`${API_BASE}/login`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password })
    });

    const data = await res.json();

    if (res.ok && data.success) {
      alert(data.message + '\nWelcome ' + data.username);

      // Save user info locally
      localStorage.setItem('userId', data.id);
      localStorage.setItem('role', data.role);
      localStorage.setItem('username', data.username);
      localStorage.setItem('email', data.email);

      // Redirect based on role
      if (data.role === "Librarian") {
        window.location.href = "managebook.html";
      } else {
        window.location.href = "main.html";
      }
    } else {
      alert(data.message || 'Invalid email or password.');
    }
  } catch (err) {
    console.error('Login Fetch Error:', err);
    alert('Could not reach server. Please ensure your Java backend is running on port 8080.');
  }
});

// REGISTER EVENT
registerForm.addEventListener('submit', async (e) => {
  e.preventDefault();

  const username = document.querySelector('#reg-username').value.trim();
  const email = document.querySelector('#reg-email').value.trim();
```

```
    const password = document.querySelector('#reg-password').value;
    const role = document.getElementById('reg-role').value;

    if (!role) {
      alert('Please select a role (Student or Librarian) before registering.');
      return;
    }

    try {
      const res = await fetch(`${API_BASE}/register`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, email, password, role })
      });

      const data = await res.json();

      if (res.ok && data.success) {
        alert((data.message || 'Registered successfully!') + '\nYou can now login as ' + role +
'.');
        showlogin(); // Go back to login form
      } else {
        alert(data.message || 'Registration failed. Please try again.');
      }
    } catch (err) {
      console.error('Registration Fetch Error:', err);
      alert('Could not reach server. Please ensure your Java backend is running on port 8080.');
    }
  });
};

  </script>
</body>
</html>
```

MAIN PAGE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Dashboard - Library Management</title>
  <script src="https://cdn.tailwindcss.com"></script>
  <style>
                                                                                @import
url('https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap');
     body {
       font-family: 'Inter', sans-serif;
       background-color: #f7f9fc;
     }
     .card {
       box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1),
              0 2px 4px -2px rgba(0, 0, 0, 0.1);
     }   body {
    font-family: Arial, sans-serif;
     margin: 20px;
  }
  .header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background: #0d6efd;
    padding: 10px;
    color: white;
    border-radius: 5px;
  }
  table {
    width: 100%;
     border-collapse: collapse;
     margin-top: 20px;
    }
  th, td {
   border: 1px solid #ddd;
    padding: 10px;
    text-align: center;
```

```
  }
  th {
    background-color: #f2f2f2;
    }
  button {
   background-color: #0d6efd;
    color: white;
     border: none;
      padding: 8px 12px;
      border-radius: 4px;
      cursor: pointer;
  }
  button:disabled {
    background-color: #aaa;
    }
  .sidebar {
   margin-top: 20px;
  }

  </style>
</head>
<body>
  <div class="min-h-screen flex flex-col">
    <!-- Navigation Bar -->
    <nav class="bg-indigo-600 text-white shadow-lg">
      <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div class="flex justify-between items-center h-16">
          <div class="flex-shrink-0 text-2xl font-bold">
            Student Portal
          </div>
          <div class="flex space-x-4">
            <span id="welcome-message" class="text-indigo-200 self-center"></span>
             <button id="logout-button" class="bg-indigo-700 hover:bg-indigo-800 text-white
font-medium py-2 px-4 rounded-lg transition duration-150">
               Logout
            </button>
          </div>
        </div>
      </div>
    </nav>
```

```
    <!-- Main Content -->
    <main class="flex-grow max-w-7xl mx-auto py-10 px-4 sm:px-6 lg:px-8 w-full">
      <h1 class="text-4xl font-extrabold text-gray-900 mb-8">Welcome to the Library!</h1>

      <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        <!-- Card 1: Browse Books -->
        <div class="card bg-white p-6 rounded-xl border border-gray-200">
          <h2 class="text-2xl font-semibold text-indigo-600 mb-3">Browse Catalog</h2>
          <p class="text-gray-600 mb-4">Search and discover available books.</p>
           <button onclick="window.location.href='viewbooks.html'" class="w-full bg-green-
500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded-lg transition duration-
150">View Books</button>
        </div>

        <!-- Card 2: My Borrowed Books -->
        <div class="card bg-white p-6 rounded-xl border border-gray-200">
          <h2 class="text-2xl font-semibold text-indigo-600 mb-3">My Borrowings</h2>
          <p class="text-gray-600 mb-4">Check due dates and request renewals.</p>
            <button onclick="window.location.href='myborrowings.html'" class="w-full bg-
yellow-500 hover:bg-yellow-600 text-white font-bold py-2 px-4 rounded-lg
transition duration-150">View Loans</button>
        </div>

        <!-- Card 3: Account Info -->
        <div class="card bg-white p-6 rounded-xl border border-gray-200">
          <h2 class="text-2xl font-semibold text-indigo-600 mb-3">Account Settings</h2>
          <p class="text-gray-600 mb-4">Update your profile information.</p>
            <button onclick="window.location.href='profile.html'" class="w-full bg-gray-500
hover:bg-gray-600 text-white font-bold py-2 px-4 rounded-lg transition duration-
150">Profile</button>
        </div>
      </div>

       <div id="status-message" class="mt-8 text-lg font-medium text-center text-indigo-
600">
        You are currently viewing the student dashboard.
      </div>
    </main>
  </div>
  <script>
    // Display username
    const storedUsername = localStorage.getItem('username') || 'Student';
```

```
                document.getElementById('welcome-message').textContent = `Hello, ${storedUsername}`;

    // Logout functionality
    document.getElementById('logout-button').addEventListener('click', () => {
      alert("Logging out...");
      localStorage.removeItem('userToken');
      localStorage.removeItem('username');
      //  Redirect to login page (make sure login.html exists)
      window.location.href = "login.html";
    });

    // Fetch books and borrowed books when needed
    async function fetchBooks() {
      try {
        const response = await fetch("http://localhost:8080/api/books");
        const books = await response.json();
        renderBooks(books);
      } catch (error) {
        console.error("Error fetching books:", error);
      }
    }

    function renderBooks(books) {
      const tbody = document.querySelector("#booksTable tbody");
      if (!tbody) return; // prevents errors if table doesn't exist
      tbody.innerHTML = "";
      books.forEach(book => {
        const row = `
          <tr>
            <td>${book.id}</td>
            <td>${book.title}</td>
            <td>${book.author}</td>
            <td>${book.available ? "Yes" : "No"}</td>
             <td><button onclick="borrowBook('${book.id}')" ${!book.available ? "disabled" : ""}>Borrow</button></td>
          </tr>`;
        tbody.innerHTML += row;
      });
    }
```

```
    async function borrowBook(id) {
      try {
            const response = await fetch(`http://localhost:8080/api/books/borrow/${id}`, {
method: "PUT" });
        if (response.ok) {
          alert("Book borrowed successfully!");
          fetchBooks();
          loadBorrowedBooks();
        } else {
          const data = await response.json();
          alert(data.message || "Failed to borrow book.");
        }
      } catch (error) {
        console.error("Error borrowing book:", error);
        alert("Could not reach server to borrow book.");
      }
    }

    async function loadBorrowedBooks() {
      try {
        const response = await fetch("http://localhost:8080/api/books/borrowed");
        const books = await response.json();
        const list = document.getElementById("borrowedBooks");
        if (!list) return; // prevents error if element missing
        list.innerHTML = "";
        books.forEach(b => {
          list.innerHTML += `<li>${b.title}</li>`;
        });
      } catch (error) {
        console.error("Error loading borrowed books:", error);
      }
    }

    function searchBooks() {
      const input = document.getElementById("search");
      if (!input) return;
      let value = input.value.toLowerCase();
      let rows = document.querySelectorAll("#booksTable tbody tr");
      rows.forEach(row => {
        row.style.display = row.innerText.toLowerCase().includes(value) ? "" : "none";
      });
```

```
    }

    document.addEventListener('DOMContentLoaded', () => {
      fetchBooks();
      loadBorrowedBooks();
    });
  </script>
</body>
</html>
```

VEIW BOOK PAGE:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width,initial-scale=1" />
 <title>Library Management — View Books</title>
 <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100 min-h-screen p-8">
 <div class="max-w-6xl mx-auto">
  <div class="flex justify-between items-center mb-6">
   <h1 class="text-3xl font-bold">Available Books</h1>
   <div>
       <button onclick="window.history.back()" class="px-3 py-2 bg-gray-300 rounded">Back</button>
   </div>
  </div>

  <div class="mb-4 flex gap-2">
    <input id="search" oninput="searchBooks()" class="border p-2 rounded w-full" placeholder="Search by title or author" />
     <button onclick="refreshAll()" class="px-4 py-2 bg-indigo-600 text-white rounded">Refresh</button>
```

```
  </div>

  <table id="booksTable" class="min-w-full bg-white rounded shadow overflow-
hidden">
    <thead class="bg-gray-50">
     <tr>
      <th class="px-4 py-2 text-left">Title</th>
      <th class="px-4 py-2 text-left">Author</th>
      <th class="px-4 py-2 text-left">Copies</th>
      <th class="px-4 py-2 text-left">Action</th>
     </tr>
    </thead>
    <tbody></tbody>
  </table>

  <!-- <h2 class="text-2xl font-semibold mt-8">My Borrowed Books</h2>
  <ul id="borrowedBooks" class="list-disc ml-6 mt-2"></ul> -->
 </div>

<script>
 // User id (for local testing). Replace with real auth flow in production.
 const userId = localStorage.getItem('userId') || 'demoUser';
 localStorage.setItem('userId', userId);

 const API_BASE = 'http://localhost:8080/api/books';

 // Load all books
 async function loadBooks() {
  try {
   const res = await fetch(`${API_BASE}`);
   if (!res.ok) throw new Error('Failed to load books');
   const books = await res.json();
   renderBooks(books);
  } catch (e) {
   console.error(e);
   alert('Could not load books from server.');
```

```
  }
 }

 function renderBooks(books) {
  const tbody = document.querySelector('#booksTable tbody');
  tbody.innerHTML = '';
  books.forEach(b => {
   const tr = document.createElement('tr');
   tr.className = 'border-t';
   tr.innerHTML = `
    <td class="px-4 py-3">${escapeHtml(b.title)}</td>
    <td class="px-4 py-3">${escapeHtml(b.author)}</td>
    <td class="px-4 py-3">${b.copies}</td>
    <td class="px-4 py-3">
      <button class="px-3 py-1 rounded ${b.copies > 0 ? 'bg-green-500 text-white' : 'bg-gray-300 text-gray-700'}"
        ${b.copies <= 0 ? 'disabled' : ''}
        onclick="borrow('${b.id}')">
       Borrow
      </button>
    </td>
   `;
   tbody.appendChild(tr);
  });
 }

 // Borrow endpoint
 async function borrow(bookId) {
  try {
                                    const res = await fetch(`${API_BASE}/borrow/${encodeURIComponent(bookId)}?userId=${encodeURIComponent(userId)}`, {
     method: 'POST'
    });
    const data = await res.json();
    if (res.ok && data.success) {
```

```
      // After successful borrow, reload both lists
      await refreshAll();
    } else {
      alert(data.message || 'Failed to borrow');
    }
  } catch (e) {
    console.error(e);
    alert('Error contacting server');
  }
}

// Load borrowed books with details DTO from backend
async function loadBorrowed() {
  try {
    const res = await fetch(`${API_BASE}/borrowed/details/${encodeURIComponent(userId)}`);
    if (!res.ok) throw new Error('Failed to load borrowed details');
    const borrowed = await res.json(); // [{ loanId, bookId, title, dueAt }]
    const list = document.getElementById('borrowedBooks');
    list.innerHTML = '';
    if (!Array.isArray(borrowed) || borrowed.length === 0) {
      list.innerHTML = '<li>No books borrowed yet.</li>';
      return;
    }
    borrowed.forEach(item => {
      const li = document.createElement('li');
      li.className = 'mb-2';
      li.innerHTML = `
        <span>${escapeHtml(item.title)} — due <b>${escapeHtml(item.dueAt)}</b></span>
        <button class="ml-3 px-2 py-1 bg-red-500 text-white rounded" onclick="returnBook('${escapeHtml(item.loanId)}')">Return</button>
      `;
      list.appendChild(li);
    });
  } catch (e) {
```

```
    console.error(e);
    const list = document.getElementById('borrowedBooks');
    list.innerHTML = '<li>Unable to load borrowed books.</li>';
   }
  }

  // Return endpoint
  async function returnBook(loanId) {
   try {
                                        const        res        =        await
fetch(`${API_BASE}/return/${encodeURIComponent(loanId)}`, { method: 'POST'
});
    const data = await res.json();
    if (res.ok && data.success) {
     await refreshAll();
    } else {
     alert(data.message || 'Failed to return book');
    }
   } catch (e) {
    console.error(e);
    alert('Error contacting server');
   }
  }

  function refreshAll() {
   loadBooks();
   loadBorrowed();
  }

  function searchBooks() {
   const q = document.getElementById('search').value.toLowerCase();
   document.querySelectorAll('#booksTable tbody tr').forEach(tr => {
    const text = tr.innerText.toLowerCase();
    tr.style.display = text.includes(q) ? '' : 'none';
   });
  }
```

```
 function escapeHtml(s) {
                return (s + '').replace(/[&<>"'`]/g, c => ({'&':'&amp;','<':'&lt;','>':'&gt;','"':'&quot;',"'":'&#39;','`':'&#96;'}[c]));
 }

 document.addEventListener('DOMContentLoaded', refreshAll);
</script>
</body>
</html>
```

MY BORROWING PAGE

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>My Borrowed Books</title>
 <script src="https://cdn.tailwindcss.com"></script>
 <style>
  body {
   font-family: 'Inter', sans-serif;
   background-color: #f9fafb;
  }
 </style>
</head>
<body class="min-h-screen p-10">
 <h1 class="text-3xl font-bold text-center mb-6 text-indigo-700">
  My Borrowed Books
 </h1>

 <div class="flex justify-center mb-6">
  <!--  Fixed path (absolute) -->
  <button
   onclick="navigateToDashboard()"
   class="bg-gray-500 hover:bg-gray-600 text-white px-4 py-2 rounded-lg"
  >
   Back
  </button>
```

```
</div>

<div class="max-w-4xl mx-auto bg-white shadow-md rounded-lg p-6">
 <table class="w-full border-collapse">
  <thead class="bg-indigo-600 text-white">
   <tr>
    <th class="p-3 text-left">Book ID</th>
    <th class="p-3 text-left">Borrowed Date</th>
    <th class="p-3 text-left">Due Date</th>
    <th class="p-3 text-center">Action</th>
   </tr>
  </thead>
  <tbody id="loanTableBody"></tbody>
 </table>
</div>

<script>
 // Robust navigation function
 function navigateToDashboard() {
  // If using Spring Boot static files

  // OR if using frontend folder structure
  window.location.href = "../html/main.html";
 }

 const userId = localStorage.getItem("userId"); // Set this at login
 const loanTable = document.getElementById("loanTableBody");

 async function fetchLoans() {
  try {
   const response = await fetch(`http://localhost:8080/api/loans/user/${userId}`);
   const loans = await response.json();

   loanTable.innerHTML = "";
   if (loans.length === 0) {
    loanTable.innerHTML = `
     <tr><td colspan="4" class="text-center p-4 text-gray-500">
      No borrowed books found.
     </td></tr>`;
    return;
   }
```

```
      loans.forEach(loan => {
        const row = document.createElement("tr");
        row.innerHTML = `
          <td class="border p-3">${loan.bookId}</td>
          <td class="border p-3">${loan.borrowedAt}</td>
          <td class="border p-3">${loan.dueAt}</td>
          <td class="border p-3 text-center">
            <button onclick="returnBook('${loan.id}')"
              class="bg-green-500 hover:bg-green-600 text-white px-3 py-1 rounded">
              Return
            </button>
          </td>`;
        loanTable.appendChild(row);
      });
    } catch (error) {
      console.error("Error fetching loans:", error);
      loanTable.innerHTML = `
        <tr><td colspan="4" class="text-center p-4 text-red-500">
          Failed to load loans.
        </td></tr>`;
    }
  }

  async function returnBook(loanId) {
    try {
      await fetch(`http://localhost:8080/api/loans/${loanId}/return`, { method: "PUT" });
      alert("Book returned successfully!");
      fetchLoans();
    } catch (error) {
      alert("Error returning book!");
    }
  }

  fetchLoans();
 </script>
</body>
</html>
```

PROFILE PAGE

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width,initial-scale=1" />
 <title>Profile</title>
 <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100 min-h-screen p-8">
 <div class="max-w-2xl mx-auto bg-white p-6 rounded shadow">
  <div class="flex justify-between items-center mb-4">
   <h1 class="text-2xl font-bold">My Profile</h1>
          <button onclick="window.history.back()" class="px-3 py-2 bg-gray-200
rounded">Back</button>
  </div>

  <form id="profileForm" class="space-y-4">
   <div>
    <label class="block text-sm font-medium">Username</label>
    <input id="username" class="mt-1 block w-full border p-2 rounded" />
   </div>

   <div>
    <label class="block text-sm font-medium">Email</label>
    <input id="email" type="email" class="mt-1 block w-full border p-2 rounded" />
   </div>

   <div>
       <label class="block text-sm font-medium">Password (leave empty to keep
current)</label>
    <input id="password" type="password" class="mt-1 block w-full border p-2 rounded" />
   </div>

   <div class="flex gap-2">
            <button type="submit" class="px-4 py-2 bg-indigo-600 text-white
rounded">Save</button>
         <button type="button" onclick="loadProfile()" class="px-4 py-2 bg-gray-200
rounded">Reload</button>
   </div>
```

```
  </form>
 </div>

<script>
 const userId = localStorage.getItem('userId') || null;

 async function loadProfile(){
  if (!userId) {
       alert("No userId in localStorage. Set localStorage.setItem('userId','<your-id>') for testing.");
    return;
  }
  try {
                                        const res = await fetch(`http://localhost:8080/api/profile/${encodeURIComponent(userId)}`);
    if (!res.ok) { alert("Profile not found"); return; }
    const user = await res.json();
    document.getElementById('username').value = user.username || '';
    document.getElementById('email').value = user.email || '';
  } catch (e) {
    console.error(e);
    alert("Could not load profile");
  }
 }

 document.getElementById('profileForm').addEventListener('submit', async (e) => {
  e.preventDefault();
  if (!userId) { alert("No logged in user"); return; }
  const payload = {
   username: document.getElementById('username').value,
   email: document.getElementById('email').value
  };
  const pw = document.getElementById('password').value;
  if (pw) payload.passwordHash = pw; // NOTE: hash in production
  try {
                                        const res = await fetch(`http://localhost:8080/api/profile/${encodeURIComponent(userId)}`, {
     method: 'PUT',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify(payload)
    });
```

```
    if (res.ok) {
      alert('Profile updated');
      // optionally update stored username
      localStorage.setItem('username', payload.username);
    } else {
      alert('Failed to update profile');
    }
  } catch (e) {
    console.error(e);
    alert('Error contacting server');
  }
});

 document.addEventListener('DOMContentLoaded', loadProfile);
</script>
</body>
</html>
```

MANAGING BOOK PAGE:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Library Management System</title>
 <style>
   *{
 margin: 0;
 padding: 0;
 box-sizing: border-box;
}
 body {
 font-family: Arial, sans-serif;
 background: #f5f7fa;
 margin: 0;
}
header {
 display: flex;
 justify-content: space-between;
 background: #dde6f1;
 padding: 15px 40px;
```

```
}
header h1 {
  margin: 0;
}
header button {
  padding: 8px 16px;
  border: 1px solid #555;
  border-radius: 4px;
  background: white;
}
.container {
  display: flex;
  padding: 20px;
}
aside {
  width: 200px;
  display: flex;
  flex-direction: column;
  gap: 10px;
}
aside button {
  padding: 10px;
  border: none;
  background: white;
  border-radius: 5px;
  cursor: pointer;
}
aside button.active, aside button:hover {
  background: #e0eaf6;
}
main {
  flex: 1;
  margin-left: 40px;
  background: white;
  padding: 20px;
  border-radius: 8px;
}
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
```

```
}
th, td {
  border: 1px solid #ddd;
  padding: 10px;
}
td button {
  margin-right: 8px;
  padding: 5px 10px;
  border-radius: 5px;
}
button.edit {
  background: #eee;
}
button.delete {
  background: #f44;
  color: white;
}
.modal {
  display: none;
  position: fixed;
  top: 0; left: 0;
  width: 100%; height: 100%;
  background: rgba(0,0,0,0.3);
  justify-content: center;
  align-items: center;
}
.modal-content {
  background: white;
  padding: 20px;
  border-radius: 8px;
  width: 300px;
}
.modal-content label {
  display: block;
  margin-top: 10px;
}
.buttons {
  margin-top: 15px;
  display: flex;
  justify-content: space-between;
}
```

```
 </style>
</head>
<body>
  <header>
    <h1>Library Management System</h1>
    <button id="logoutBtn" onclick="logout()">Logout</button>
  </header>

  <div class="container">
    <aside>
      <button onclick="showAddForm()">Add Book</button>
                    <button    class="active"    onclick="navigateTo('managebooks')">Manage
Books</button>
      <button onclick="navigateTo('manageusers')">Manage Users</button>
      <button onclick="navigateTo('transactions')">Transactions</button>
    </aside>

    <main>
      <h2>Manage Books</h2>
      <table id="bookTable">
        <thead>
          <tr>
            <th>#</th>
            <th>Title</th>
            <th>Author</th>
            <th>Copies</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody></tbody>
      </table>

      <div id="bookForm" class="modal">
        <div class="modal-content">
          <h3 id="formTitle">Add Book</h3>
          <form onsubmit="saveBook(event)">
            <input type="hidden" id="bookId">
            <label>Title:</label>
            <input type="text" id="title" required>
            <label>Author:</label>
            <input type="text" id="author" required>
```

```
            <label>Copies:</label>
            <input type="number" id="copies" min="0" required>
            <div class="buttons">
              <button type="submit">Save</button>
              <button type="button" onclick="closeForm()">Cancel</button>
            </div>
          </form>
        </div>
      </div>
    </main>
  </div>

  <script>
    //  Correct API endpoint for your Spring Boot backend
const API_URL = "http://localhost:8080/api/books";

// --- Sidebar Navigation ---
function navigateTo(page) {
  if (page === 'managebooks') window.location.href = 'managebook.html';
  else if (page === 'manageusers') window.location.href = 'ManageUser.html';
  else if (page === 'transactions') window.location.href = 'transaction.html';
}

function logout() {
  alert("Logging out...");
  window.location.href = "login.html";
}

// --- Load Books from Backend ---
async function loadBooks() {
  try {
    const res = await fetch(API_URL);
    const books = await res.json();
    const tbody = document.querySelector("#bookTable tbody");
    tbody.innerHTML = "";

    books.forEach((b, i) => {
      const row = document.createElement("tr");
      row.innerHTML = `
        <td>${i + 1}</td>
        <td>${b.title}</td>
```

```
          <td>${b.author}</td>
          <td>${b.copies}</td>
          <td>
                    <button  class="edit"  onclick="editBook('${b.id}',  '${b.title}',  '${b.author}',
${b.copies})">Edit</button>
            <button class="delete" onclick="deleteBook('${b.id}')">Delete</button>
          </td>
        `;
        tbody.appendChild(row);
      });
  } catch (error) {
    console.error("Error loading books:", error);
    alert("Failed to load books from server.");
  }
}

// --- Add or Edit Book ---
function showAddForm() {
  document.getElementById("formTitle").innerText = "Add Book";
  document.getElementById("bookId").value = "";
  document.getElementById("title").value = "";
  document.getElementById("author").value = "";
  document.getElementById("copies").value = "";
  document.getElementById("bookForm").style.display = "flex";
}

function closeForm() {
  document.getElementById("bookForm").style.display = "none";
}

function editBook(id, title, author, copies) {
  document.getElementById("formTitle").innerText = "Edit Book";
  document.getElementById("bookId").value = id;
  document.getElementById("title").value = title;
  document.getElementById("author").value = author;
  document.getElementById("copies").value = copies;
  document.getElementById("bookForm").style.display = "flex";
}

// --- Save (POST or PUT) ---
async function saveBook(e) {
```

```
  e.preventDefault();

  const id = document.getElementById("bookId").value;
  const book = {
    title: document.getElementById("title").value,
    author: document.getElementById("author").value,
    copies: parseInt(document.getElementById("copies").value)
  };

  const method = id ? "PUT" : "POST";
  const url = id ? `${API_URL}/${id}` : API_URL;

  try {
    const res = await fetch(url, {
      method,
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(book)
    });

    if (res.ok) {
      closeForm();
      loadBooks();
    } else {
      alert("Failed to save book. Server responded with status " + res.status);
    }
  } catch (error) {
    console.error("Error saving book:", error);
    alert("Could not reach server to save book.");
  }
}

// --- Delete Book ---
async function deleteBook(id) {
  if (!confirm("Delete this book?")) return;
  try {
    const res = await fetch(`${API_URL}/${id}`, { method: "DELETE" });
    if (res.ok) {
      loadBooks();
    } else {
      alert("Failed to delete book.");
    }
```

```
  } catch (error) {
    console.error("Error deleting book:", error);
    alert("Could not reach server to delete book.");
  }
}

// Load books when page opens
window.onload = loadBooks;

  </script>
</body>
</html>
```

MANAGE USER PAGE

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Library Management — Users</title>
 <meta name="viewport" content="width=device-width,initial-scale=1">
 <style>
  *{
   margin: 0;
   padding: 0;
   box-sizing: border-box;
}
/* simple, clean layout */
:root {
   --accent:#2b2b2b;
    --muted:#dde6f1;
    --danger:#d9534f;
  }
body {
   font-family: Arial, sans-serif;
   margin:0;
   background: #f5f7fa;
   color:var(--accent);
   }
.site-header {
```

```
    display: flex;
    justify-content: space-between;
    background:var(--muted);
    padding:20px;
   }

   .site-header button{
    padding: 8px 16px;
    border: 1px solid #555;
    border-radius: 4px;
    background: white;
   }

.container {
   display:flex;
    gap:20px;
    padding:30px;
   }

.sidebar{
 width: 200px;
 display: flex;
 flex-direction: column;
 gap: 10px;
}

 .sidebar button{
 padding: 10px;
 border: none;
 background: white;
 border-radius: 5px;
 cursor: pointer;
 }

 .sidebar button.active, aside button:hover {
 background: #e0eaf6;
}

.content {
    flex:1;
    }
```

```
#user-form {
    display:flex;
    gap:8px;
    margin-bottom:16px;
     align-items:center;
   }

#user-form input, #user-form select {
   padding:8px;
    border-radius:6px;
     border:1px solid #ddd;
   }

#user-form button {
    padding:8px 12px;
    border-radius:6px;
     border:1px solid #ccc;
     background:white;
     cursor:pointer;
   }

#users-table {
    width:100%;
    border-collapse:collapse;
    background:#fff;
     border-radius:8px;
      overflow:hidden;
      box-shadow:0 1px 3px rgba(0,0,0,.06);
    }
#users-table th, #users-table td {
   padding:18px;
    text-align:left;
    border-bottom:1px solid #eee;
   }
.action-btn {
    margin-right:8px;
    padding:6px 10px;
    border-radius:6px;
    cursor:pointer;
    border:1px solid #ccc;
```

```
    background:white;
   }
.remove {
    background:var(--danger);
    color:white;
    border-color:rgba(0,0,0,0);
    }
 </style>
</head>
<body>
 <header class="site-header">
  <h1>Library Management System</h1>
  <button id="logoutBtn" onclick="logout()">Logout</button>
 </header>

 <main class="container">
  <aside class="sidebar">
   <button onclick="navigateTo('managebooks')">Manage Books</button>
   <button class="active" onclick="navigateTo('manageusers')">Manage Users</button>
   <button onclick="navigateTo('transactions')">Transactions</button>
  </aside>

  <section class="content">
   <h2>Manage Users</h2>

   <form id="user-form">
    <input type="hidden" id="user-id">
    <input type="text" id="name" placeholder="Name" required>
    <select id="role">
     <option>Student</option>
     <option>Librarian</option>
    </select>
    <button type="submit">Save</button>
    <button type="button" id="reset-btn">Reset</button>
   </form>

   <table id="users-table">
    <thead>
     <tr><th>ID</th><th>Name</th><th>Role</th><th>Action</th></tr>
    </thead>
    <tbody></tbody>
```

```
    </table>
   </section>
  </main>

  <script>
const baseURL = 'http://localhost:8080/api/users';

// --- Navigation Functions for Sidebar ---
function navigateTo(page) {
   if (page === 'managebooks') window.location.href = 'managebook.html';
   else if (page === 'manageusers') window.location.href = 'ManageUser.html';
   else if (page === 'transactions') window.location.href = 'transaction.html';
}

function logout() {
   alert("Logging out...");
   window.location.href = "login.html";
}
// ----------------------------------------

document.addEventListener('DOMContentLoaded', () => {
   const tbody = document.querySelector('#users-table tbody');
   const form = document.getElementById('user-form');
   const nameInput = document.getElementById('name');
   const roleInput = document.getElementById('role');
   const userIdInput = document.getElementById('user-id');
   const resetBtn = document.getElementById('reset-btn');

   async function loadUsers(){
      try {
         // Uses the new baseURL
         const res = await fetch(baseURL);
         const data = await res.json();
         tbody.innerHTML = '';
         data.forEach(u => {
            const tr = document.createElement('tr');
            tr.innerHTML = `
              <td>${u.id}</td>
              <td>${escapeHtml(u.name)}</td>
              <td>${escapeHtml(u.role)}</td>
              <td>
```

```
          <button class="action-btn" data-id="${u.id}" data-action="edit">Edit</button>
                                <button class="action-btn remove" data-id="${u.id}" data-
action="delete">Remove</button>
           </td>`;
          tbody.appendChild(tr);
        });
      } catch (err) {
        console.error(err);
        tbody.innerHTML = '<tr><td colspan="4">Failed to load users</td></tr>';
      }
    }

    form.addEventListener('submit', async (e) => {
      e.preventDefault();
      const name = nameInput.value.trim();
      const role = roleInput.value;
      const id = userIdInput.value;

      if (!name) return alert('Enter name');

      try {
        if (id) {
          // update (Uses the new baseURL)
          const res = await fetch(`${baseURL}/${id}`, {
            method:'PUT',
            headers:{ 'Content-Type':'application/json' },
            body: JSON.stringify({ name, role })
          });
          if (!res.ok) throw new Error('Update failed');
        } else {
          // create (Uses the new baseURL)
          const res = await fetch(baseURL, {
            method:'POST',
            headers:{ 'Content-Type':'application/json' },
            body: JSON.stringify({ name, role })
          });
          if (!res.ok) throw new Error('Create failed');
        }
        resetForm();
        loadUsers();
      } catch (err) {
```

```
      console.error(err);
      alert('Error saving user');
    }
});

tbody.addEventListener('click', async (e) => {
  const btn = e.target.closest('button');
  if (!btn) return;
  const id = btn.dataset.id;
  const action = btn.dataset.action;
  if (action === 'edit') {
    // fetch single user then populate form (Uses the new baseURL)
    try {
      const res = await fetch(`${baseURL}/${id}`);
      const user = await res.json();
      userIdInput.value = user.id;
      nameInput.value = user.name;
      roleInput.value = user.role;
    } catch (err) { console.error(err); alert('Failed to fetch user'); }
  } else if (action === 'delete') {
    if (!confirm('Delete user?')) return;
    // delete (Uses the new baseURL)
    try {
      const res = await fetch(`${baseURL}/${id}`, { method:'DELETE' });
      if (!res.ok) throw new Error('Delete failed');
      loadUsers();
    } catch (err) { console.error(err); alert('Delete failed'); }
  }
});

resetBtn.addEventListener('click', resetForm);

function resetForm(){
  userIdInput.value = '';
  nameInput.value = '';
  roleInput.value = 'Student';
}

function escapeHtml(s){
  return s.replace(/[&<>"']/g, c => ({'&':'&amp;','<':'&lt;','>':'&gt;','"':'&quot;',"'":'&#39;'}[c]));
}
```

```
  // initial load
  loadUsers();
});
 </script>
</body>
</html>
```

TRANSACTION PAGE

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Library Management - Transactions</title>
<script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100 min-h-screen">
 <header class="bg-blue-100 p-4 flex justify-between items-center shadow">
  <h1 class="text-2xl font-bold">Library Management System</h1>
            <button class="bg-red-500 text-white px-4 py-2 rounded" onclick="logout()">Logout</button>
 </header>

 <div class="flex">
  <!-- Sidebar -->
  <aside class="w-1/5 bg-white shadow p-4">
      <button class="block w-full mb-3 py-2 rounded bg-gray-200 hover:bg-gray-300" onclick="navigateTo('managebooks')">Manage Books</button>
      <button class="block w-full mb-3 py-2 rounded bg-gray-200 hover:bg-gray-300" onclick="navigateTo('manageusers')">Manage Users</button>
            <button class="block w-full py-2 rounded bg-blue-500 text-white">Transactions</button>
  </aside>

  <!-- Main content -->
  <main class="flex-1 p-8">
   <h2 class="text-3xl font-semibold mb-6">Transactions</h2>

   <!-- Issue Book Section -->
   <div class="bg-white shadow rounded-lg p-6 mb-6">
```

```
      <h3 class="text-xl font-semibold mb-4">Issue Book</h3>
      <div class="flex flex-wrap gap-4 items-center">
       <select id="book" class="border p-2 rounded w-64" required>
        <option value="">-- Select Book --</option>
       </select>

       <select id="borrower" class="border p-2 rounded w-64" required>
        <option value="">-- Select Borrower --</option>
       </select>

       <input type="date" id="issueDate" class="border p-2 rounded" required>
       <input type="date" id="dueDate" class="border p-2 rounded" required>
          <button id="saveBtn" class="bg-green-500 text-white px-4 py-2 rounded hover:bg-
green-600">Save</button>
      </div>
    </div>

    <!-- Transactions Table -->
    <div class="bg-white shadow rounded-lg p-4 overflow-x-auto">
     <table id="transactionsTable" class="min-w-full text-left border">
      <thead class="bg-gray-50 border-b">
       <tr>
        <th class="px-4 py-2 border-r">ID</th>
        <th class="px-4 py-2 border-r">Book</th>
        <th class="px-4 py-2 border-r">Borrower</th>
        <th class="px-4 py-2 border-r">Issue Date</th>
        <th class="px-4 py-2 border-r">Due Date</th>
        <th class="px-4 py-2 border-r">Fine (₹)</th>
        <th class="px-4 py-2">Action</th>
       </tr>
      </thead>
      <tbody></tbody>
     </table>
    </div>
   </main>
  </div>

<script>
const API_URL = "http://localhost:8080/api/transactions";
const BOOKS_URL = "http://localhost:8080/api/books";
const USERS_URL = "http://localhost:8080/api/users";
```

```
// NOTE: delete endpoint uses same base as API_URL
document.addEventListener("DOMContentLoaded", () => {
 loadDropdowns();
 loadTransactions();
 document.getElementById("saveBtn").addEventListener("click", issueBook);
});

// Load books and users into dropdowns
async function loadDropdowns() {
 try {
        const [booksRes, usersRes] = await Promise.all([fetch(BOOKS_URL), fetch(USERS_URL)]);
    if (!booksRes.ok || !usersRes.ok) throw new Error("Failed to load books/users for dropdowns");

  const books = await booksRes.json();
  const users = await usersRes.json();

  const bookSelect = document.getElementById("book");
  const borrowerSelect = document.getElementById("borrower");

  bookSelect.innerHTML = `<option value="">-- Select Book --</option>` +
      books.map(b => `<option value="${b.id}">${escapeHtml(b.title || b.name || b.id)}</option>`).join("");

  borrowerSelect.innerHTML = `<option value="">-- Select Borrower --</option>` +
            users.map(u => `<option value="${u.id}">${escapeHtml(u.name || u.id)}</option>`).join("");
 } catch (err) {
  console.error(err);
    alert("Error loading books or users. Check backend endpoints /api/books and /api/users.");
 }
}

// Issue book (POST)
async function issueBook() {
 const book = document.getElementById("book").value;
 const borrower = document.getElementById("borrower").value;
 const issueDate = document.getElementById("issueDate").value;
```

```
  const dueDate = document.getElementById("dueDate").value;

  if (!book || !borrower || !issueDate || !dueDate) {
    alert("Please fill all fields.");
    return;
  }

  // Build transaction object
  const transaction = { book, borrower, issueDate, dueDate };

  try {
    const res = await fetch(API_URL, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(transaction)
    });

    if (!res.ok) {
      // read server error message if available
      const text = await res.text();
      throw new Error(text || "Error issuing book");
    }

    alert("Book issued successfully!");
    // reset to defaults
    document.getElementById("book").selectedIndex = 0;
    document.getElementById("borrower").selectedIndex = 0;
    document.getElementById("issueDate").value = "";
    document.getElementById("dueDate").value = "";
    loadTransactions();
  } catch (err) {
    console.error(err);
    alert("Error issuing book: " + err.message);
  }
}

// Load transactions (GET)
async function loadTransactions() {
  const tbody = document.querySelector("#transactionsTable tbody");
  tbody.innerHTML = "<tr><td colspan='7' class='p-4'>Loading...</td></tr>";
```

```
 try {
  const res = await fetch(API_URL);
  if (!res.ok) {
   const text = await res.text();
   throw new Error(text || "Failed to load transactions");
  }
  const transactions = await res.json();
  tbody.innerHTML = "";

  if (!transactions || transactions.length === 0) {
      tbody.innerHTML = "<tr><td colspan='7' class='p-4 text-center'>No transactions found.</td></tr>";
   return;
  }

  transactions.forEach(t => {
   const row = document.createElement("tr");
   row.className = "border-t";
   const fine = (typeof t.fine === "number") ? t.fine.toFixed(2) : "0.00";
   // Escape text before inserting to avoid HTML injection
   row.innerHTML = `
    <td class="px-4 py-2">${escapeHtml(t.id)}</td>
    <td class="px-4 py-2">${escapeHtml(t.bookTitle || t.book || "N/A")}</td>
        <td class="px-4 py-2">${escapeHtml(t.borrowerName || t.borrower || "user name")}</td>
    <td class="px-4 py-2">${escapeHtml(t.issueDate || "")}</td>
    <td class="px-4 py-2">${escapeHtml(t.dueDate || "")}</td>
    <td class="px-4 py-2 text-red-600 font-semibold">${fine}</td>
    <td class="px-4 py-2">
          <button class="bg-red-500 text-white px-3 py-1 rounded hover:bg-red-600" onclick="deleteTransaction('${t.id}')">Return</button>
    </td>
   `;
   tbody.appendChild(row);
  });
 } catch (err) {
  console.error(err);
  tbody.innerHTML = "<tr><td colspan='7' class='p-4 text-center text-red-500'>Error loading transactions</td></tr>";
 }
}
```

```
// Delete transaction (return book) — use same base API_URL
async function deleteTransaction(id) {
 if (!confirm("Confirm return of this book?")) return;
 try {
  const res = await fetch(`${API_URL}/${id}`, { method: "DELETE" });
  if (!res.ok) {
   const txt = await res.text();
   throw new Error(txt || "Error deleting transaction");
  }
  alert("Book returned successfully!");
  loadTransactions();
 } catch (err) {
  console.error(err);
  alert("Error deleting transaction: " + err.message);
 }
}

function logout() {
 localStorage.clear();
 window.location.href = "login.html";
}
function navigateTo(page) {
 if (page === "managebooks") window.location.href = "../html/managebook.html";
 if (page === "manageusers") window.location.href = "../html/manageuser.html";
 if (page === "transactions") window.location.href = "../html/transaction.html";
}

// small helper to escape user-provided text for insertion into HTML
function escapeHtml(str) {
 if (str === null || str === undefined) return "";
 return String(str)
  .replace(/&/g, "&amp;")
  .replace(/</g, "&lt;")
  .replace(/>/g, "&gt;")
  .replace(/"/g, "&quot;")
  .replace(/'/g, "&#039;");
}
</script>
</body>
</html>
```

Backend Code:

com.example.LibraryManagementSystem(package):

LibraryManagementSystemApplication.java

```
package com.example.LibraryManagementSystem;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LibraryManagementSystemApplication {
 public static void main(String[] args) {
  SpringApplication.run(LibraryManagementSystemApplication.class, args);
  System.out.println("Welcome LibraryManagementSystem");
 }
}
```

com.example.LibraryManagementSystem.book(package):

Book.java

```
package com.example.LibraryManagementSystem.book;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "books")
public class Book {

  @Id
  private String id;
  private String title;
  private String author;
  private String category;
  private int copies;

  public Book() {}

  public Book(String title, String author, String category, int copies) {
```

```
    this.title = title;
    this.author = author;
    this.category = category;
    this.copies = copies;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getAuthor() {
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}

public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

public int getCopies() {
    return copies;
}
```

```
    public void setCopies(int copies) {
        this.copies = copies;
    }
}
```

Librarian.java

```
package com.example.LibraryManagementSystem.book;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "librarians")
public class Librarian {

        @Id
private String id;
private String username;
private String email;
private String password;
private String role;


public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getUsername() { return username; }
public void setUsername(String username) { this.username = username; }
public String getEmail() { return email; }
public void setEmail(String email) { this.email = email; }
public String getPassword() { return password; }
public void setPassword(String password) { this.password = password; }
public String getRole() { return role; }
public void setRole(String role) { this.role = role; }


// Constructors
public Librarian() {}

public Librarian(String username, String email, String password, String role) {
```

```
  this.username = username;
  this.email = email;
  this.password = password;
  this.role = role;
}

   // CRITICAL FIX: Ensure getName() returns String and uses the display name (username).
   public String getName() {
      return username;
   }
}
```

Loan.java

```
package com.example.LibraryManagementSystem.book;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.time.LocalDate;

@Document(collection = "loans")
public class Loan {
   @Id
   private String id;
   private String userId;
   private String bookId;
   private LocalDate borrowedAt;
   private LocalDate dueAt;
   private boolean returned;

   public String getId() { return id; }
   public void setId(String id) { this.id = id; }

   public String getUserId() { return userId; }
   public void setUserId(String userId) { this.userId = userId; }

   public String getBookId() { return bookId; }
   public void setBookId(String bookId) { this.bookId = bookId; }

   public LocalDate getBorrowedAt() { return borrowedAt; }
   public void setBorrowedAt(LocalDate borrowedAt) { this.borrowedAt = borrowedAt; }
```

```
    public LocalDate getDueAt() { return dueAt; }
    public void setDueAt(LocalDate dueAt) { this.dueAt = dueAt; }

    public boolean isReturned() { return returned; }
    public void setReturned(boolean returned) { this.returned = returned; }
}
```

ManageUser.java

```
package com.example.LibraryManagementSystem.book;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.time.Instant;

@Document(collection = "manageusers")
public class ManageUser {

    @Id
    private String id;
    private String name;
    private String role;
    private Instant createdAt;

    public ManageUser() {
        this.createdAt = Instant.now();
    }

    public ManageUser(String name, String role) {
        this.name = name;
        this.role = role;
        this.createdAt = Instant.now();
    }

    // --- Getters and Setters ---
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getName() { return name; }
```

```
    public void setName(String name) { this.name = name; }

    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }

    public Instant getCreatedAt() { return createdAt; }
    public void setCreatedAt(Instant createdAt) { this.createdAt = createdAt; }
}
```

Transaction.java

```
package com.example.LibraryManagementSystem.book;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import com.fasterxml.jackson.annotation.JsonFormat;
import java.time.LocalDate;

@Document(collection = "transactions")
public class Transaction {

    @Id
    private String id;
    private String book;       // Book ID
    private String borrower;    // User ID
    private String bookTitle;   // For display (populated server-side)
    private Object borrowerName;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private LocalDate issueDate;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private LocalDate dueDate;
    private double fine;

    public Transaction() {}

    public Transaction(String book, String borrower, LocalDate issueDate, LocalDate
dueDate) {
        this.book = book;
        this.borrower = borrower;
        this.issueDate = issueDate;
```

```
        this.dueDate = dueDate;
        this.fine = 0.0;
    }

    // Getters & Setters
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getBook() { return book; }
    public void setBook(String book) { this.book = book; }

    public String getBorrower() { return borrower; }
    public void setBorrower(String borrower) { this.borrower = borrower; }

    public String getBookTitle() { return bookTitle; }
    public void setBookTitle(String bookTitle) { this.bookTitle = bookTitle; }

    public Object getBorrowerName() { return borrowerName; }
    public void setBorrowerName(Object borrowerName2) { this.borrowerName =
borrowerName2; }

    public LocalDate getIssueDate() { return issueDate; }
    public void setIssueDate(LocalDate issueDate) { this.issueDate = issueDate; }

    public LocalDate getDueDate() { return dueDate; }
    public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }

    public double getFine() { return fine; }
    public void setFine(double fine) { this.fine = fine; }
}
```

User.java

```
package com.example.LibraryManagementSystem.book;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "users")
```

```
public class User
{
        @Id
        private String id;
        private String username;
        @Indexed(unique = true)
        private String email;

       //  Ensure this field exists for consistency with getName()
  private String name;

  private String passwordHash;
  private String role;

  // Constructors
  public User() {}

  public User(String username, String email, String passwordHash, String role) {
    this.username = username;
    this.email = email;
    this.passwordHash = passwordHash;
    this.role = role;
    // Note: The 'name' field is not set here, which could be an issue if it's required for transactions.
    // If 'name' is empty in the database, the transaction service will use 'username' as a fallback if implemented.
  }

  // getters & setters
  public String getId() {
    return id;
  }
  public void setId(String id) {
    this.id = id;
  }
  public String getUsername(){
    return username;
  }
  public void setUsername(String username) {
    this.username = username;
  }
```

```
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPasswordHash() {
        return passwordHash;
    }
    public void setPasswordHash(String passwordHash) {
        this.passwordHash = passwordHash;
    }

    // Getter and Setter for Role
    public String getRole() {
        return role;
    }
    public void setRole(String role) { this.role = role; }

    public String getPassword() { return passwordHash; }

    //  CRITICAL FIX: Ensure getName() returns String and provides a name.
    public String getName() {
        // Use 'name' if available, otherwise fall back to 'username'
        return (name != null && !name.isEmpty()) ? name : username;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

com.example.LibraryManagementSystem.BookController(package):

AuthController.java

```
package com.example.LibraryManagementSystem.BookController;

import com.example.LibraryManagementSystem.book.User;
import com.example.LibraryManagementSystem.book.Librarian;
import com.example.LibraryManagementSystem.BookRepository.UserRepository;
```

```
import com.example.LibraryManagementSystem.BookRepository.LibrarianRepository;
import com.example.LibraryManagementSystem.payload.AuthRequest;
import com.example.LibraryManagementSystem.payload.AuthResponse;
import com.example.LibraryManagementSystem.payload.RegisterRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.bcrypt.BCrypt;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "*")
public class AuthController {

  @Autowired
  private UserRepository userRepository;

  @Autowired
  private LibrarianRepository librarianRepository;

  //  REGISTER ENDPOINT (for both Student and Librarian)
  @PostMapping("/register")
  public ResponseEntity<?> register(@RequestBody RegisterRequest registerRequest) {
    String role = registerRequest.getRole();

    // Check if email already exists in either collection
    if (userRepository.existsByEmail(registerRequest.getEmail()) ||
      librarianRepository.findByEmail(registerRequest.getEmail()).isPresent()) {
      return ResponseEntity.badRequest().body(
          new AuthResponse(false, "Email is already in use")
      );
    }

    String hashed = BCrypt.hashpw(registerRequest.getPassword(), BCrypt.gensalt(12));

    if ("Librarian".equalsIgnoreCase(role)) {
      Librarian librarian = new Librarian(
          registerRequest.getUsername(),
```

```
            registerRequest.getEmail(),
            hashed,
            "Librarian"
        );
        librarianRepository.save(librarian);
        return ResponseEntity.ok(new AuthResponse(true,
            "Librarian registered successfully",
            librarian.getUsername(),
            librarian.getEmail(),
            librarian.getRole(),
            librarian.getId()));
    } else {
        User user = new User(
            registerRequest.getUsername(),
            registerRequest.getEmail(),
            hashed,
            "Student"
        );
        userRepository.save(user);
        return ResponseEntity.ok(new AuthResponse(true,
            "User registered successfully",
            user.getUsername(),
            user.getEmail(),
            user.getRole(),
            user.getId()));
    }
}

// 🔑 LOGIN ENDPOINT (checks both collections)
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody AuthRequest authRequest) {

    // 1️⃣ Check User collection
    Optional<User> maybeUser = userRepository.findByEmail(authRequest.getEmail());
    if (maybeUser.isPresent()) {
        User user = maybeUser.get();
        if (BCrypt.checkpw(authRequest.getPassword(), user.getPasswordHash())) {
            return ResponseEntity.ok(new AuthResponse(true,
                "Login successful",
                user.getUsername(),
                user.getEmail(),
```

```
                user.getRole(),
                user.getId()));
        }
    }

    // Check Librarian collection
    Optional<Librarian> maybeLibrarian =
librarianRepository.findByEmail(authRequest.getEmail());
    if (maybeLibrarian.isPresent()) {
        Librarian librarian = maybeLibrarian.get();
        if (BCrypt.checkpw(authRequest.getPassword(), librarian.getPassword())) {
            return ResponseEntity.ok(new AuthResponse(true,
                "Login successful",
                librarian.getUsername(),
                librarian.getEmail(),
                librarian.getRole(),
                librarian.getId()));
        }
    }

    // If neither matched
    return ResponseEntity.badRequest()
        .body(new AuthResponse(false, "Invalid email or password"));
  }
}
```

BookController.java

```
package com.example.LibraryManagementSystem.BookController;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import com.example.LibraryManagementSystem.BookRepository.BookRepository;
import com.example.LibraryManagementSystem.BookRepository.LoanRepository;
import com.example.LibraryManagementSystem.BookRepository.TransactionRepository;
import com.example.LibraryManagementSystem.book.Book;
import com.example.LibraryManagementSystem.book.Loan;
import com.example.LibraryManagementSystem.book.Transaction;
```

```
import java.time.LocalDate;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/books")
@CrossOrigin(origins = "*")
public class BookController {

  @Autowired
  private BookRepository bookRepository;

  @Autowired
  private LoanRepository loanRepository;

  @Autowired
  private TransactionRepository transactionRepository;

  // Get all books
  @GetMapping
  public List<Book> getAllBooks() {
    return bookRepository.findAll();
  }

  // Get book by ID
  @GetMapping("/{id}")
  public Optional<Book> getBookById(@PathVariable String id) {
    return bookRepository.findById(id);
  }

  // Add new book
  @PostMapping
  public Book addBook(@RequestBody Book book) {
    return bookRepository.save(book);
  }

  // Update book
  @PutMapping("/{id}")
  public Book updateBook(@PathVariable String id, @RequestBody Book updatedBook) {
    return bookRepository.findById(id)
```

```
            .map(book -> {
                book.setTitle(updatedBook.getTitle());
                book.setAuthor(updatedBook.getAuthor());
                book.setCopies(updatedBook.getCopies());
                return bookRepository.save(book);
            })
            .orElseGet(() -> {
                updatedBook.setId(id);
                return bookRepository.save(updatedBook);
            });
    }

    // Delete book
    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable String id) {
        bookRepository.deleteById(id);
    }

    //  Borrow a book
    @PostMapping("/borrow/{bookId}")
    public Response borrowBook(@PathVariable String bookId, @RequestParam String
userId) {
        Optional<Book> opt = bookRepository.findById(bookId);
        if (opt.isEmpty()) return new Response(false, "Book not found");
        Book book = opt.get();

        if (book.getCopies() <= 0) return new Response(false, "No copies available");

        book.setCopies(book.getCopies() - 1);
        bookRepository.save(book);

        Loan loan = new Loan();
        loan.setBookId(bookId);
        loan.setUserId(userId);
        loan.setBorrowedAt(LocalDate.now());
        loan.setDueAt(LocalDate.now().plusWeeks(2));
        loan.setReturned(false);
        loanRepository.save(loan);

        Transaction transaction = new Transaction(
            book.getTitle(),
```

```
        userId,
        LocalDate.now(),
        LocalDate.now().plusWeeks(2)
    );
    transactionRepository.save(transaction);

    return new Response(true, "Book borrowed successfully");
}

//  Delete transaction (return book)
@DeleteMapping("/transaction/{transactionId}")
public Response deleteTransaction(@PathVariable String transactionId) {
    Optional<Transaction> transOpt = transactionRepository.findById(transactionId);
    if (transOpt.isEmpty()) return new Response(false, "Transaction not found");

    Transaction transaction = transOpt.get();
    transactionRepository.deleteById(transactionId);

    // Increase book copies
    List<Book> books = bookRepository.findAll();
    Book targetBook = books.stream()
        .filter(b -> b.getTitle().equalsIgnoreCase(transaction.getBook()))
        .findFirst()
        .orElse(null);

    if (targetBook != null) {
      targetBook.setCopies(targetBook.getCopies() + 1);
      bookRepository.save(targetBook);
    }

    // Mark loan as returned
    List<Loan> loans = loanRepository.findAll();
    loans.stream()
        .filter(l -> l.getUserId().equals(transaction.getBorrower())
            && targetBook != null
            && l.getBookId().equals(targetBook.getId())
            && !l.isReturned())
        .forEach(l -> {
          l.setReturned(true);
          loanRepository.save(l);
        });
```

```
    return new Response(true, "Transaction deleted and book returned successfully");
  }

  // Get borrowed books
  @GetMapping("/borrowed/{userId}")
  public List<Loan> getBorrowedByUser(@PathVariable String userId) {
    return loanRepository.findByUserIdAndReturnedFalse(userId);
  }

  @GetMapping("/borrowed/details/{userId}")
  public List<BorrowedBookDto> getBorrowedDetails(@PathVariable String userId) {
    List<Loan> loans = loanRepository.findByUserIdAndReturnedFalse(userId);
    return loans.stream()
        .map(loan -> {
          Book book = bookRepository.findById(loan.getBookId()).orElse(null);
          if (book == null) return null;
          return new BorrowedBookDto(loan.getId(), loan.getBookId(), book.getTitle(),
loan.getDueAt());
        })
        .filter(dto -> dto != null)
        .collect(Collectors.toList());
  }

  @PostMapping("/return/{loanId}")
  public Response returnBook(@PathVariable String loanId) {
    Optional<Loan> loanOpt = loanRepository.findById(loanId);
    if (loanOpt.isEmpty()) return new Response(false, "Loan not found");

    Loan loan = loanOpt.get();
    if (loan.isReturned()) return new Response(false, "Book already returned");

    loan.setReturned(true);
    loanRepository.save(loan);

    Optional<Book> bookOpt = bookRepository.findById(loan.getBookId());
    bookOpt.ifPresent(book -> {
      book.setCopies(book.getCopies() + 1);
      bookRepository.save(book);
    });
```

```
    return new Response(true, "Book returned successfully");
  }

  static class BorrowedBookDto {
    public String loanId;
    public String bookId;
    public String title;
    public LocalDate dueAt;
    public BorrowedBookDto(String loanId, String bookId, String title, LocalDate dueAt) {
      this.loanId = loanId;
      this.bookId = bookId;
      this.title = title;
      this.dueAt = dueAt;
    }
  }

  static class Response {
    public boolean success;
    public String message;
    public Response(boolean success, String message) {
      this.success = success;
      this.message = message;
    }
  }
}
```

LibrarianController.java

```
package com.example.LibraryManagementSystem.BookController;

//src/main/java/com/yourpackage/controller/LibrarianController.java
//NOTE: For a clean application, book/user management logic should be in separate controllers.
//This example groups management functions under one logical role, the Librarian.


import org.springframework.web.bind.annotation.*;

import com.example.LibraryManagementSystem.book.Book;
```

```
import org.springframework.http.ResponseEntity;

//Assuming the base path for librarian/management APIs is separate
@RestController
@RequestMapping("/api/books")
@CrossOrigin(origins = "*")
public class LibrarianController {

 // You would inject services here (e.g., BookService, UserService, TransactionService)
 // @Autowired private BookService bookService;
 // @Autowired private UserService userService;

 // ----------------------------------------
 // BOOK MANAGEMENT (Called by managebook.js)
 // ----------------------------------------

 // POST /api/management/books (or /api/books if shared)
 @PostMapping("/api/books")
 public ResponseEntity<?> addBook(@RequestBody Book book) {
    // Logic to save a new book (only accessible by Librarians)
    // return ResponseEntity.ok(bookService.save(book));
    return ResponseEntity.ok("Book added successfully (Librarian only)");
 }

 // PUT /api/management/books/{id} (or /api/books/{id})
 @PutMapping("/api/books/{id}")
 public ResponseEntity<?> updateBook(@PathVariable String id, @RequestBody Book
bookDetails) {
    // Logic to update an existing book
    // return ResponseEntity.ok(bookService.update(id, bookDetails));
    return ResponseEntity.ok("Book updated successfully (Librarian only)");
 }

 // DELETE /api/management/books/{id} (or /api/books/{id})
 @DeleteMapping("/api/books/{id}")
 public ResponseEntity<?> deleteBook(@PathVariable String id) {
    // Logic to delete a book
    // bookService.delete(id);
    return ResponseEntity.ok("Book deleted successfully (Librarian only)");
 }
```

```
// ----------------------------------------
// USER MANAGEMENT (Called by ManageUser.js)
// ----------------------------------------

// GET /api/management/users (or /api/users if shared)
@GetMapping("/users")
public ResponseEntity<?> getAllUsers() {
   // Logic to fetch all users (Students and/or other Librarians)
   // return ResponseEntity.ok(userService.findAll());
   return ResponseEntity.ok("List of all users (Librarian only)");
}

// ... Implement POST, PUT, DELETE for /api/users here ...

// ----------------------------------------
// TRANSACTION MANAGEMENT (Called by transaction.js)
// ----------------------------------------

// GET /api/management/transactions (or /api/transactions if shared)
@GetMapping("/transactions")
public ResponseEntity<?> getAllTransactions() {
   // Logic to fetch all transactions
   // return ResponseEntity.ok(transactionService.findAll());
   return ResponseEntity.ok("List of all transactions (Librarian only)");
}
}
```

LoanController.java

```
package com.example.LibraryManagementSystem.BookController;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.example.LibraryManagementSystem.book.Loan;
import com.example.LibraryManagementSystem.book.Book;
import com.example.LibraryManagementSystem.BookRepository.LoanRepository;
import com.example.LibraryManagementSystem.BookRepository.BookRepository;
```

```
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/loans")
@CrossOrigin(origins = "*")
public class LoanController {

  @Autowired
  private LoanRepository loanRepository;

  @Autowired
  private BookRepository bookRepository;

  //  Get all active (not returned) loans for a user
  @GetMapping("/user/{userId}")
  public List<Loan> getLoansByUser(@PathVariable String userId) {
    return loanRepository.findByUserIdAndReturnedFalse(userId);
  }

  //  Return book: mark loan returned + increment book copies
  @PutMapping("/{loanId}/return")
  public ResponseEntity<ReturnResponse> markAsReturned(@PathVariable String loanId) {
    Optional<Loan> loanOpt = loanRepository.findById(loanId);
    if (loanOpt.isEmpty()) {
      return ResponseEntity.badRequest().body(new ReturnResponse(false, "Loan not
found"));
    }

    Loan loan = loanOpt.get();
    if (loan.isReturned()) {
      return ResponseEntity.badRequest().body(new ReturnResponse(false, "Book already
returned"));
    }

    // Mark the loan as returned
    loan.setReturned(true);
    loanRepository.save(loan);

    // Increment copies on the associated book
```

```
    Optional<Book> bookOpt = bookRepository.findById(loan.getBookId());
    if (bookOpt.isPresent()) {
      Book book = bookOpt.get();
      book.setCopies(book.getCopies() + 1);
      bookRepository.save(book);
    } else {
      // Book missing — still return success for the loan
      return ResponseEntity.ok(
        new ReturnResponse(true, "Loan marked returned; book not found to increment copies")
      );
    }

    return ResponseEntity.ok(new ReturnResponse(true, "Book returned successfully"));
  }

  //  Simple response DTO for frontend feedback
  static class ReturnResponse {
    public boolean success;
    public String message;

    public ReturnResponse(boolean success, String message) {
      this.success = success;
      this.message = message;
    }
  }
}
```

ManageUserController.java

```
package com.example.LibraryManagementSystem.BookController;

import com.example.LibraryManagementSystem.book.ManageUser;
import com.example.LibraryManagementSystem.payload.ManageUserService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.util.List;
```

```
@RestController
@RequestMapping("/api/users")
@CrossOrigin(origins = "*") // Allow frontend requests; set specific origin in production
public class ManageUserController {

  private final ManageUserService userService;

  public ManageUserController(ManageUserService userService) {
    this.userService = userService;
  }

  // GET /api/users?q=search
  @GetMapping
  public List<ManageUser> list(@RequestParam(required = false) String q) {
    return userService.getAll(q);
  }

  // GET /api/users/{id}
  @GetMapping("/{id}")
  public ResponseEntity<ManageUser> get(@PathVariable String id) {
    return userService.getById(id)
        .map(ResponseEntity::ok)
        .orElse(ResponseEntity.notFound().build());
  }

  // POST /api/users
  @PostMapping
  public ResponseEntity<ManageUser> create(@RequestBody ManageUser payload) {
    ManageUser created = userService.create(payload);
    return ResponseEntity.created(URI.create("/api/users/" + created.getId()))
        .body(created);
  }

  // PUT /api/users/{id}
  @PutMapping("/{id}")
  public ResponseEntity<ManageUser> update(@PathVariable String id, @RequestBody
ManageUser payload) {
    try {
      ManageUser updated = userService.update(id, payload);
      return ResponseEntity.ok(updated);
    } catch (RuntimeException ex) {
```

```
        return ResponseEntity.notFound().build();
      }
    }

    // DELETE /api/users/{id}
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable String id) {
      userService.delete(id);
      return ResponseEntity.noContent().build();
    }
}
```

TransactionController.java

```
package com.example.LibraryManagementSystem.BookController;

import com.example.LibraryManagementSystem.book.Transaction;
import com.example.LibraryManagementSystem.payload.TransactionService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.net.URI;
import java.util.List;

@RestController
@RequestMapping("/api/transactions")
@CrossOrigin(origins = "*")
public class TransactionController {

    private final TransactionService service;

    public TransactionController(TransactionService service) {
      this.service = service;
    }

    @GetMapping
    public List<Transaction> all() {
      return service.getAll();
    }

    @GetMapping("/{id}")
```

```
  public ResponseEntity<Transaction> get(@PathVariable String id) {
    return service.getById(id)
        .map(ResponseEntity::ok)
        .orElse(ResponseEntity.notFound().build());
  }

  @PostMapping
  public ResponseEntity<?> add(@RequestBody Transaction t) {
    try {
      Transaction created = service.add(t);
      return ResponseEntity.created(URI.create("/api/transactions/" +
created.getId())).body(created);
    } catch (RuntimeException e) {
      return ResponseEntity.badRequest().body(e.getMessage());
    }
  }

  @DeleteMapping("/{id}")
  public ResponseEntity<Void> delete(@PathVariable String id) {
    service.delete(id);
    return ResponseEntity.noContent().build();
  }
}
```

UserProfileController.java

```
package com.example.LibraryManagementSystem.BookController;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.example.LibraryManagementSystem.BookRepository.UserRepository;
import com.example.LibraryManagementSystem.book.User;

import java.util.Optional;

@RestController
@RequestMapping("/api/profile")
@CrossOrigin(origins = "*")
```

```
public class UserProfileController {

  @Autowired
  private UserRepository userRepository;

  // GET profile by user id
  @GetMapping("/{id}")
  public ResponseEntity<User> getProfile(@PathVariable String id) {
    Optional<User> u = userRepository.findById(id);
    return u.map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
  }

  // UPDATE profile (username, email, passwordHash). For password you should hash in production.
  @PutMapping("/{id}")
  public ResponseEntity<User> updateProfile(@PathVariable String id, @RequestBody User payload) {
    return userRepository.findById(id).map(user -> {
      if (payload.getUsername() != null) user.setUsername(payload.getUsername());
      if (payload.getEmail() != null) user.setEmail(payload.getEmail());
      if (payload.getPasswordHash() != null && !payload.getPasswordHash().isEmpty()) {
        // In production: hash the password using BCrypt before saving
        user.setPasswordHash(payload.getPasswordHash());
      }
      userRepository.save(user);
      return ResponseEntity.ok(user);
    }).orElse(ResponseEntity.notFound().build());
  }
}
```

com.example.LibraryManagementSystem.BookRepository(package):

BookRepository.java

```
package com.example.LibraryManagementSystem.BookRepository;

import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.LibraryManagementSystem.book.Book;

public interface BookRepository extends MongoRepository<Book, String> {}
```

LibrarianRepository.java

```
package com.example.LibraryManagementSystem.BookRepository;

//src/main/java/com/yourpackage/repository/LibrarianRepository.java

import com.example.LibraryManagementSystem.book.Librarian;
import org.springframework.data.mongodb.repository.MongoRepository;
import java.util.Optional;

public interface LibrarianRepository extends MongoRepository<Librarian, String> {

 // Method to find a Librarian by email for login/checks
 Optional<Librarian> findByEmail(String email);
}
```

LoanRepository.java

```
package com.example.LibraryManagementSystem.BookRepository;

import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.LibraryManagementSystem.book.Loan;

public interface LoanRepository extends MongoRepository<Loan, String> {
   List<Loan> findByUserIdAndReturnedFalse(String userId);
}
```

ManageUserRepository.java

```
package com.example.LibraryManagementSystem.BookRepository;

import com.example.LibraryManagementSystem.book.ManageUser;
import org.springframework.data.mongodb.repository.MongoRepository;
```

```
import java.util.List;

public interface ManageUserRepository extends MongoRepository<ManageUser, String> {

   List<ManageUser>
findByNameContainingIgnoreCaseOrRoleContainingIgnoreCase(String name, String role);
}
```

TransactionRepository.java

```
package com.example.LibraryManagementSystem.BookRepository;

import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.LibraryManagementSystem.book.Transaction;

public interface TransactionRepository extends MongoRepository<Transaction, String> {}
```

UserRepository.java

```
package com.example.LibraryManagementSystem.BookRepository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;
import com.example.LibraryManagementSystem.book.User;

public interface UserRepository extends MongoRepository<User, String> {
   Optional<User> findByEmail(String email);
   boolean existsByEmail(String email);
         List<User>
findByUsernameContainingIgnoreCaseOrRoleContainingIgnoreCase(String q, String q2);
}
```

com.example.LibraryManagementSystem.payload(package):

AuthRequest.java

```
package com.example.LibraryManagementSystem.payload;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;

public class AuthRequest {
  @NotBlank @Email
  private String email;

  @NotBlank
  private String password;

  // getters & setters
  public String getEmail() { return email; }
  public void setEmail(String email) { this.email = email; }

  public String getPassword() { return password; }
  public void setPassword(String password) { this.password = password; }
}
```

AuthResponse.java

```
package com.example.LibraryManagementSystem.payload;

public class AuthResponse {
  private boolean success;
  private String message;
  private String username;
  private String email;
  private String role;
  private String id;

  public AuthResponse(boolean success, String message) {
    this.success = success;
    this.message = message;
  }

  public AuthResponse(boolean success, String message, String username, String email,
String role, String id) {
    this.success = success;
```

```
        this.message = message;
        this.username = username;
        this.email = email;
        this.role = role;
        this.id = id;
    }

    // Getters and setters
    public boolean isSuccess() { return success; }
    public void setSuccess(boolean success) { this.success = success; }

    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
}
```

ManageUserService.java

```
package com.example.LibraryManagementSystem.payload;

import com.example.LibraryManagementSystem.book.ManageUser;
import com.example.LibraryManagementSystem.BookRepository.ManageUserRepository;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class ManageUserService {
```

```
    private final ManageUserRepository repo;

    public ManageUserService(ManageUserRepository repo) {
        this.repo = repo;
    }

    public List<ManageUser> getAll(String q) {
        if (q == null || q.isBlank()) {
            return repo.findAll();
        }
        return repo.findByNameContainingIgnoreCaseOrRoleContainingIgnoreCase(q, q);
    }

    public Optional<ManageUser> getById(String id) {
        return repo.findById(id);
    }

    public ManageUser create(ManageUser user) {
        return repo.save(user);
    }

    public ManageUser update(String id, ManageUser updated) {
        return repo.findById(id).map(existing -> {
            existing.setName(updated.getName());
            existing.setRole(updated.getRole());
            return repo.save(existing);
        }).orElseThrow(() -> new RuntimeException("User not found"));
    }

    public void delete(String id) {
        repo.deleteById(id);
    }
}
```

RegisterRequest.java

```
package com.example.LibraryManagementSystem.payload;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
```

```
import jakarta.validation.constraints.Size;

public class RegisterRequest {
   @NotBlank @Size(min = 3, max = 50)
   private String username;

   @NotBlank @Email
   private String email;

   @NotBlank @Size(min = 6, max = 100)
   private String password;

   // getters & setters
   public String getUsername() { return username; }
   public void setUsername(String username) { this.username = username; }

   public String getEmail() { return email; }
   public void setEmail(String email) { this.email = email; }

   public String getPassword() { return password; }
   public void setPassword(String password) { this.password = password; }
        public String getRole() {
                // TODO Auto-generated method stub
                return null;
        }
}
```

TransactionService.java

```
package com.example.LibraryManagementSystem.payload;

import com.example.LibraryManagementSystem.book.Transaction;
import com.example.LibraryManagementSystem.book.Book;
import com.example.LibraryManagementSystem.book.User;
// Necessary imports
import com.example.LibraryManagementSystem.book.Librarian;
import com.example.LibraryManagementSystem.BookRepository.LibrarianRepository;
import com.example.LibraryManagementSystem.BookRepository.TransactionRepository;
import com.example.LibraryManagementSystem.BookRepository.BookRepository;
```

```
import com.example.LibraryManagementSystem.BookRepository.UserRepository;
import org.springframework.stereotype.Service;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.List;
import java.util.Optional;

@Service
public class TransactionService {
  private final TransactionRepository transactionRepo;
  private final BookRepository bookRepo;
  private final UserRepository userRepo;
  // Inject the Librarian repository
  private final LibrarianRepository librarianRepo;

  private static final double FINE_PER_DAY = 5.0;

  // UPDATED CONSTRUCTOR: Must include LibrarianRepository
  public TransactionService(TransactionRepository transactionRepo, BookRepository bookRepo,
                UserRepository userRepo, LibrarianRepository librarianRepo) {
    this.transactionRepo = transactionRepo;
    this.bookRepo = bookRepo;
    this.userRepo = userRepo;
    this.librarianRepo = librarianRepo;
  }

  public List<Transaction> getAll() {
    List<Transaction> list = transactionRepo.findAll();
    // Populate display names and compute fine
    list.forEach(this::populateNames);
    list.forEach(this::calculateFine);
    return list;
  }

  public Optional<Transaction> getById(String id) {
    Optional<Transaction> opt = transactionRepo.findById(id);
    opt.ifPresent(t -> {
      populateNames(t);
      calculateFine(t);
    });
```

```
    return opt;
  }

  public Transaction add(Transaction t) {
    Book book = bookRepo.findById(t.getBook()).orElseThrow(() -> new
RuntimeException("Book not found"));

    // FIX 1: Consolidated lookup for borrower
    String borrowerName = findBorrowerNameById(t.getBorrower());
    if (borrowerName == null) {
      // This is the error message seen in your screenshot
      throw new RuntimeException("Error issuing book: User not found with ID " +
t.getBorrower());
    }

    if (book.getCopies() <= 0) {
      throw new RuntimeException("No copies available");
    }

    // Update copies
    book.setCopies(book.getCopies() - 1);
    bookRepo.save(book);

    // Set names for display
    t.setBookTitle(book.getTitle());
    t.setBorrowerName(borrowerName); // Set the name from the lookup
    t.setFine(0.0);

    return transactionRepo.save(t);
  }

  public Transaction update(String id, Transaction t) {
    return transactionRepo.findById(id)
        .map(existing -> {
          existing.setBook(t.getBook());
          existing.setBorrower(t.getBorrower());
          existing.setIssueDate(t.getIssueDate());
          existing.setDueDate(t.getDueDate());
          populateNames(existing);
          calculateFine(existing);
          return transactionRepo.save(existing);
```

```
        })
        .orElseThrow(() -> new RuntimeException("Transaction not found"));
  }

  public void delete(String id) {
    transactionRepo.findById(id).ifPresent(transaction -> {
      bookRepo.findById(transaction.getBook()).ifPresent(book -> {
        book.setCopies(book.getCopies() + 1);
        bookRepo.save(book);
      });
    });
    transactionRepo.deleteById(id);
  }

  // MODIFIED HELPER: Uses consolidated lookup for table display
  private void populateNames(Transaction t) {
    if (t.getBook() != null) {
      bookRepo.findById(t.getBook()).ifPresent(b -> t.setBookTitle(b.getTitle()));
    }
    if (t.getBorrower() != null) {
      String borrowerName = findBorrowerNameById(t.getBorrower());
      // This is crucial for displaying the name in the table
      if (borrowerName != null) {
        t.setBorrowerName(borrowerName);
      } else {
        // If the name is not found, keep the borrower ID in borrowerName temporarily
        // so the front-end has something, though the ID should usually be used.
        // However, the front-end fallback logic will handle it: ${t.borrowerName ||
t.borrower}
        t.setBorrowerName(null);
      }
    }
  }

  // CONSOLIDATED LOOKUP FUNCTION (Checks both User and Librarian)
  private String findBorrowerNameById(String id) {
    // 1. Check Student (User) collection
    Optional<User> userOpt = userRepo.findById(id);
    if (userOpt.isPresent()) {
      String name = userOpt.get().getName();
      // Return only if the name is not null or empty
```

```
        if (name != null && !name.trim().isEmpty()) return name;
      }

      // 2. Check Librarian collection
      Optional<Librarian> librarianOpt = librarianRepo.findById(id);
      if (librarianOpt.isPresent()) {
        String name = librarianOpt.get().getName();
        // Return only if the name is not null or empty
        if (name != null && !name.trim().isEmpty()) return name;
      }

      return null; // ID not found in either collection, or name/username field is empty
    }


    private void calculateFine(Transaction t) {
      LocalDate due = t.getDueDate();
      if (due != null && LocalDate.now().isAfter(due)) {
        long daysLate = ChronoUnit.DAYS.between(due, LocalDate.now());
        t.setFine(daysLate * FINE_PER_DAY);
      } else {
        t.setFine(0.0);
      }
    }
}
```

application.properties

```
spring.data.mongodb.uri=mongodb://localhost:27017/librarydb
server.port=8080

# optional: show mongo queries
spring.data.mongodb.show-sql=false
```

pom.xml

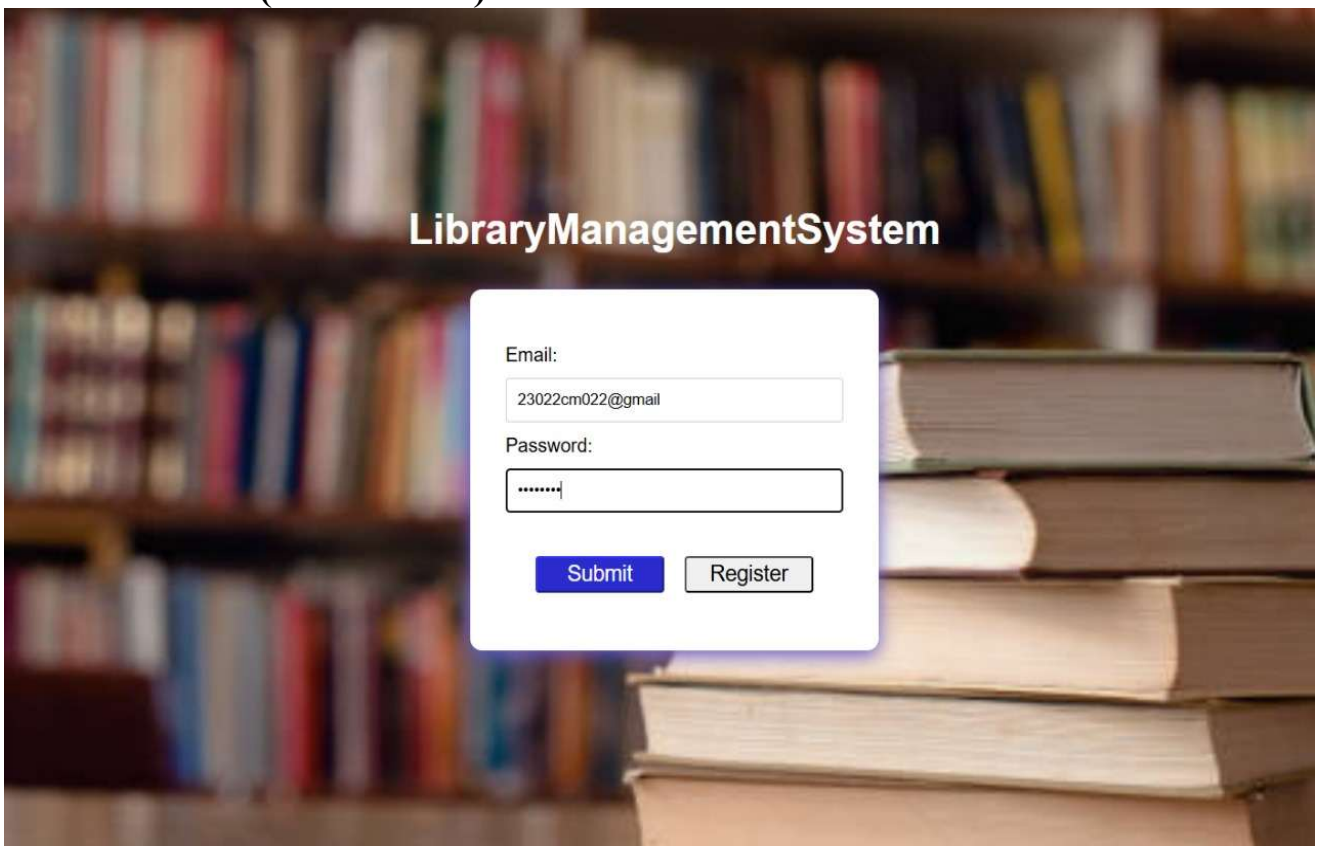
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

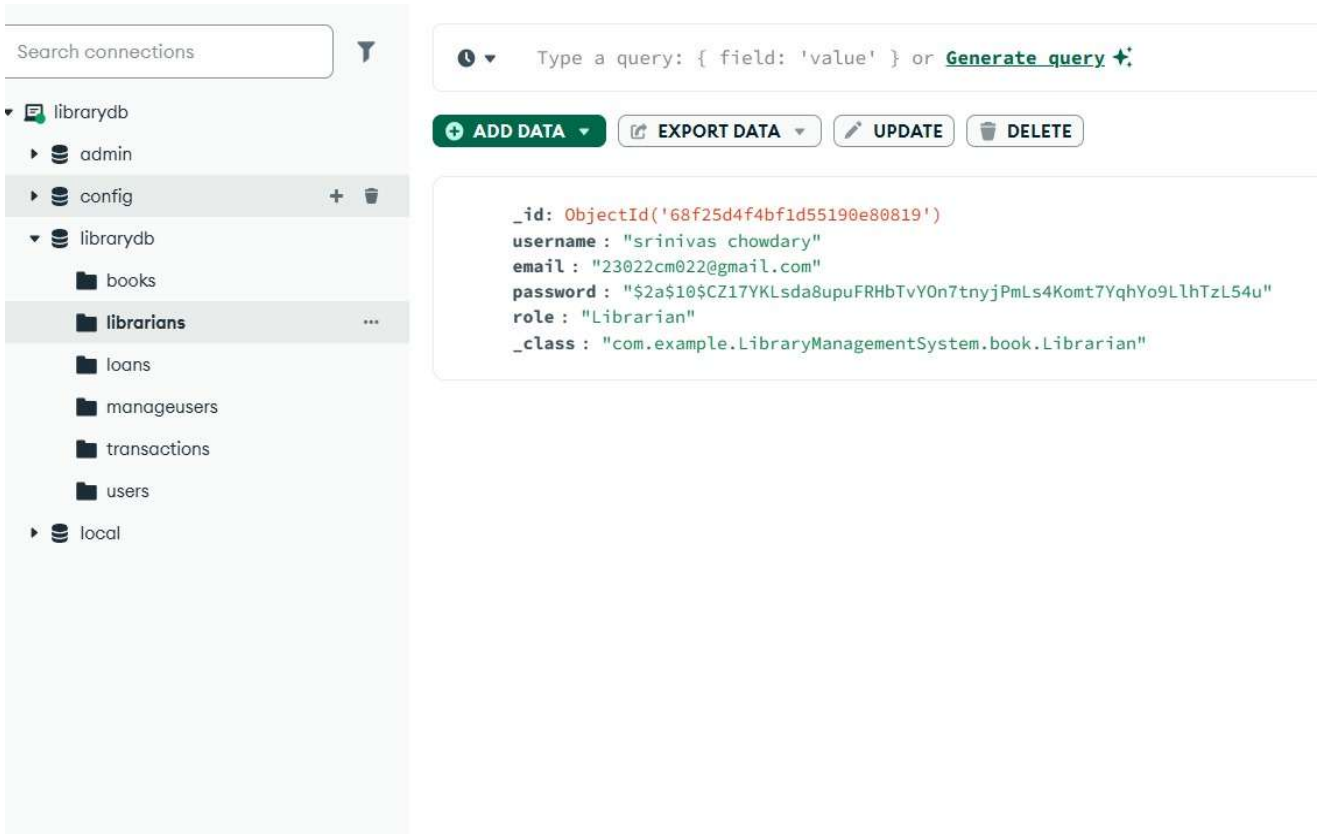
```
                http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>library-backend</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<properties>
 <java.version>17</java.version>
 <spring.boot.version>3.1.4</spring.boot.version>
</properties>

<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-dependencies</artifactId>
   <version>${spring.boot.version}</version>
   <type>pom</type>
   <scope>import</scope>
  </dependency>
 </dependencies>
</dependencyManagement>

<dependencies>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
 </dependency>

 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
 </dependency>

 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
 </dependency>
```

```
  <!-- BCrypt -->
  <dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-crypto</artifactId>
  </dependency>
  <!-- Lombok (optional, remove if you don't want Lombok) -->
  <dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <optional>true</optional>
  </dependency>

  <dependency>
   <groupId>com.fasterxml.jackson.module</groupId>
   <artifactId>jackson-module-parameter-names</artifactId>
  </dependency>

  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-test</artifactId>
   <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
  </plugins>
 </build>
</project>
```

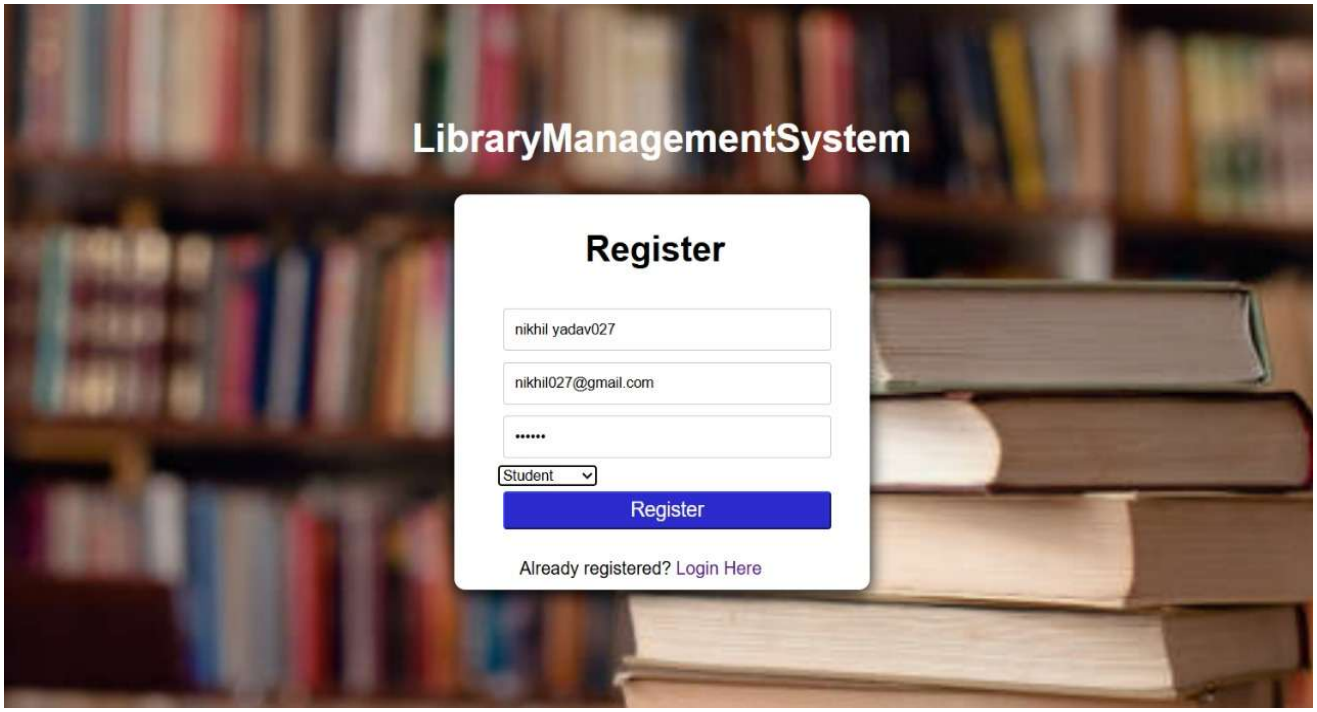
OUTPUT:

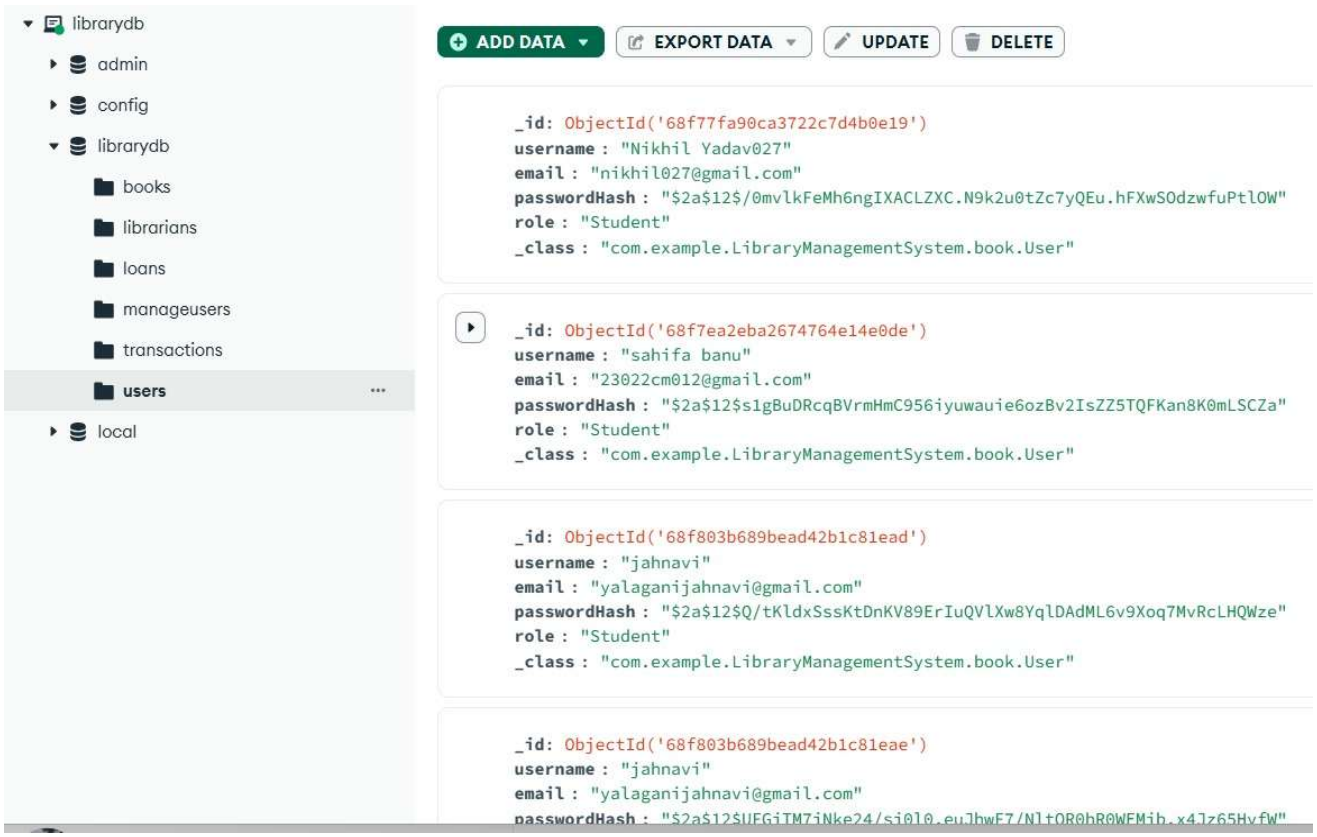
LOGIN PAGE(LIBRARIAN)



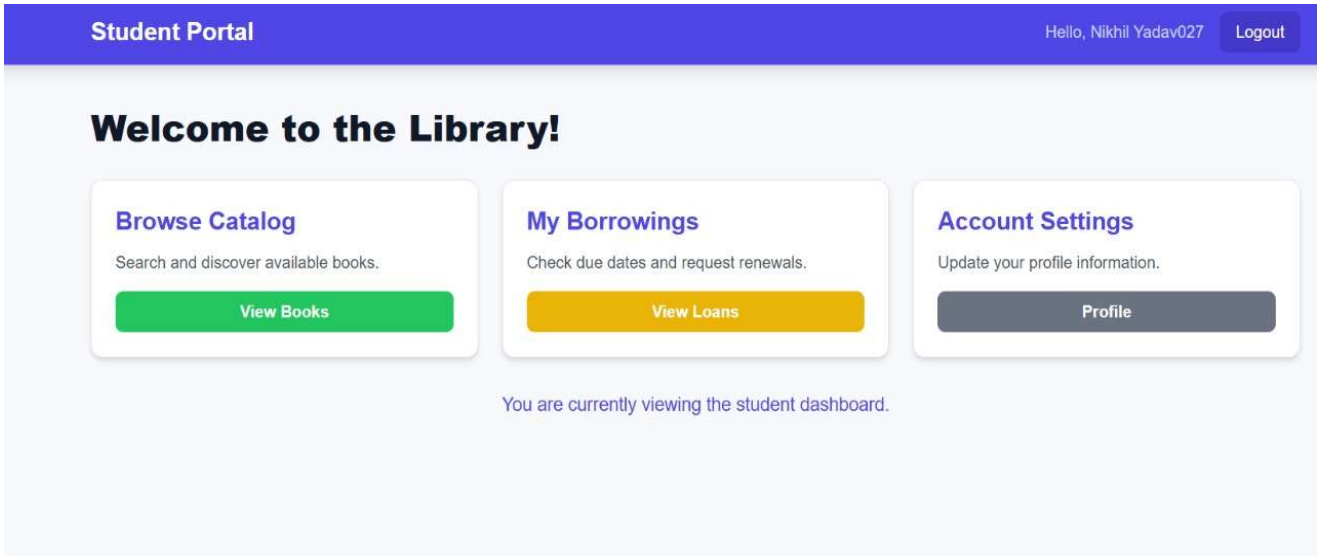


LOGIN PAGE(student):

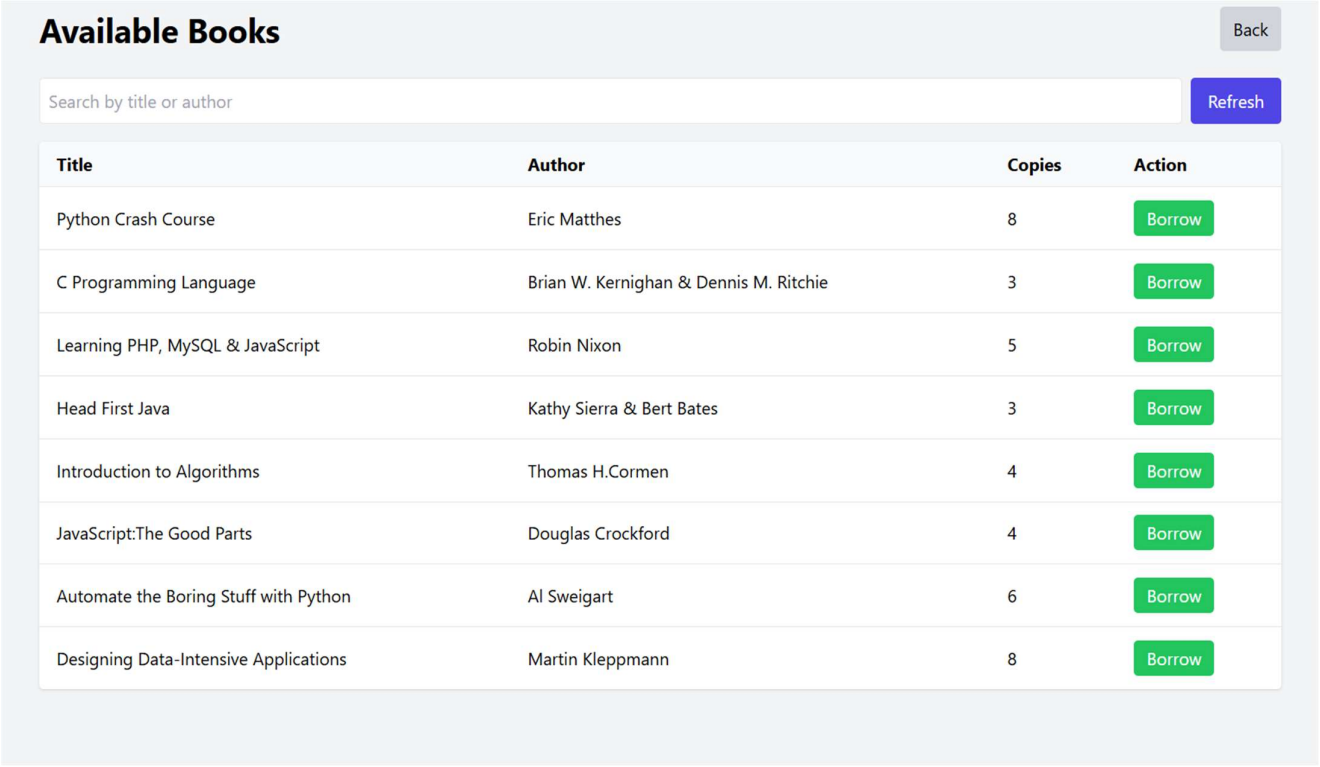




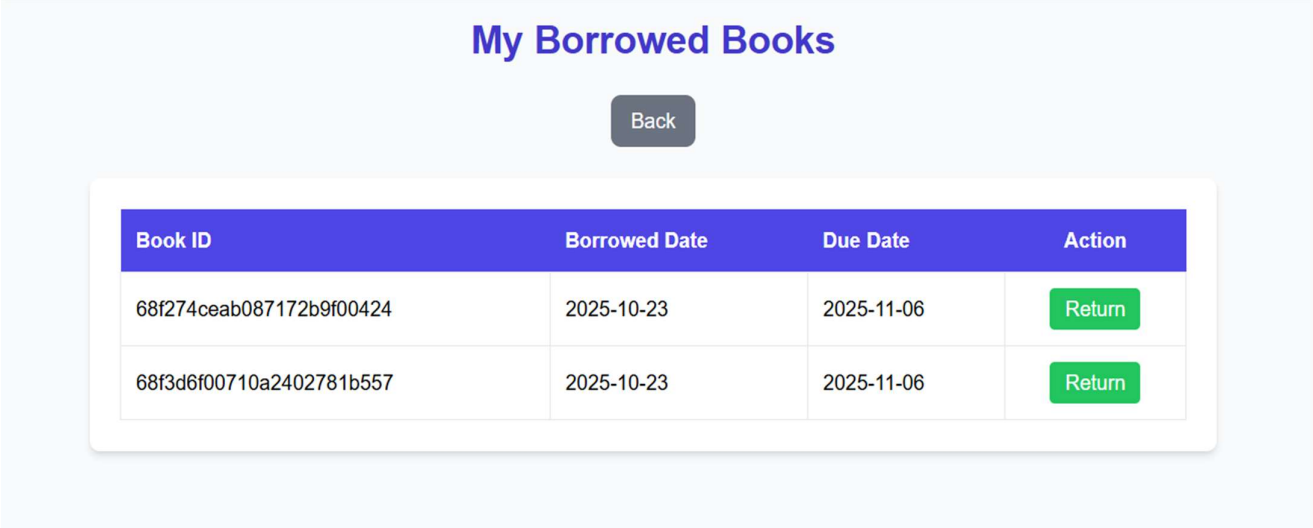
MAIN PAGE:



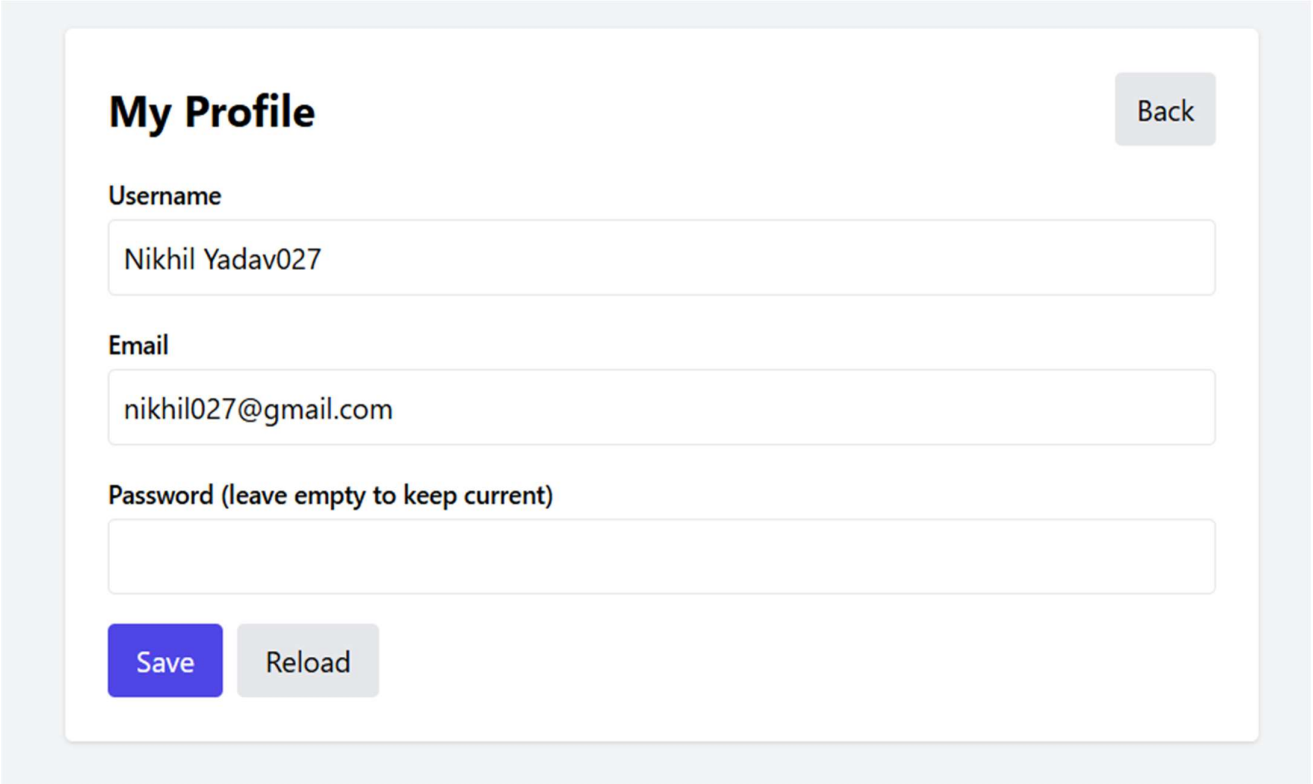
VIEW BOOK PAGE:



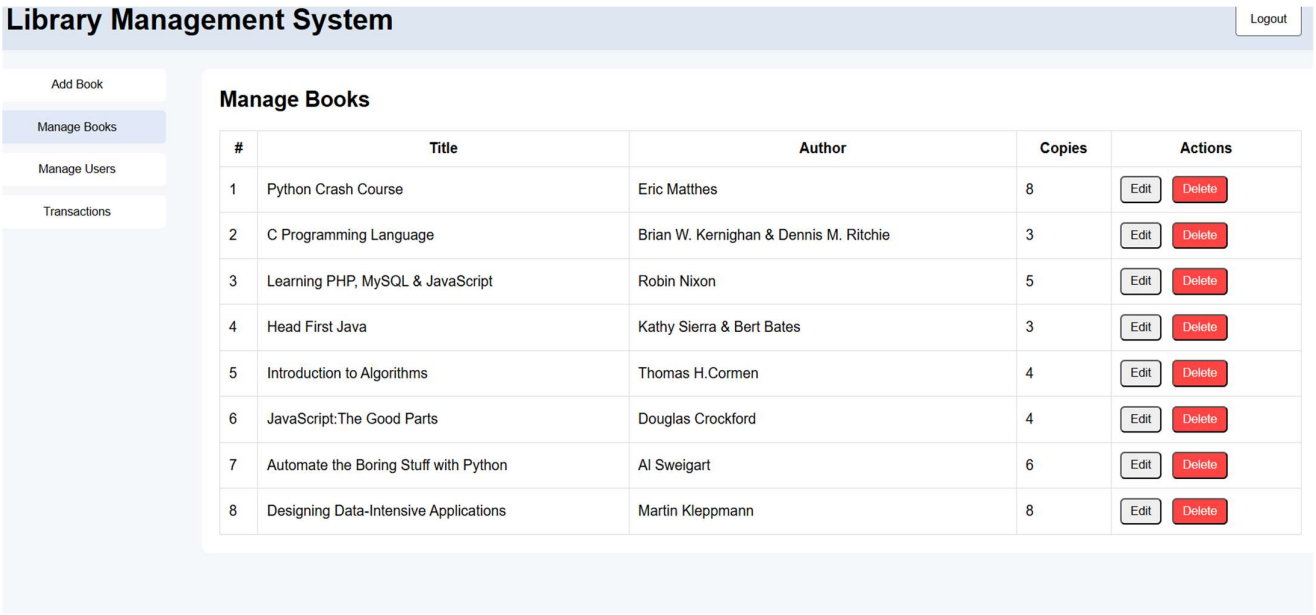
MY BORROWING PAGE:



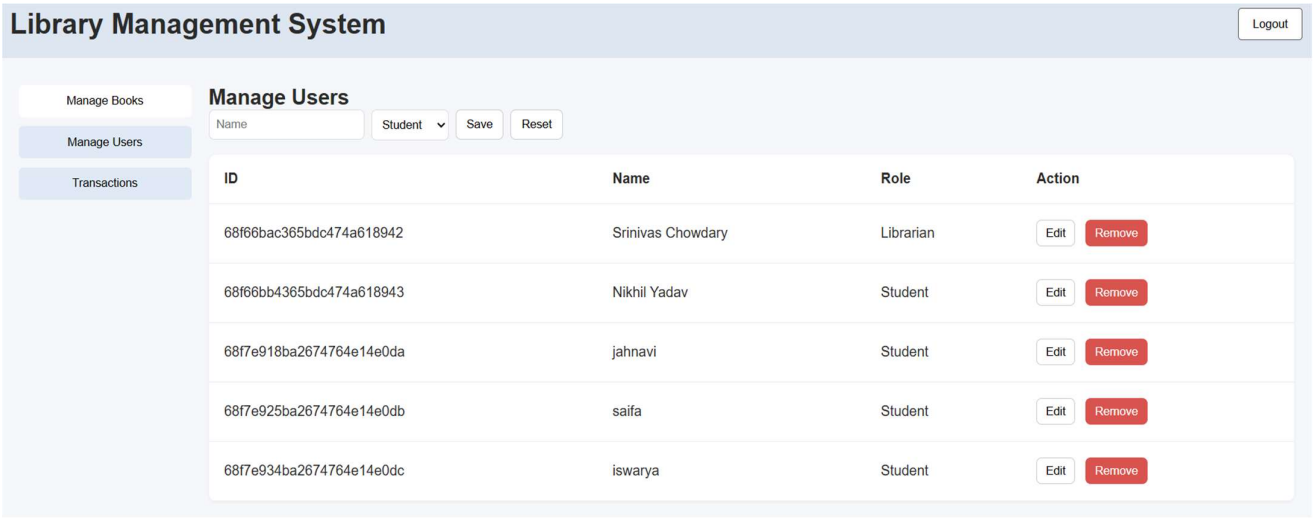
PROFILE PAGE:



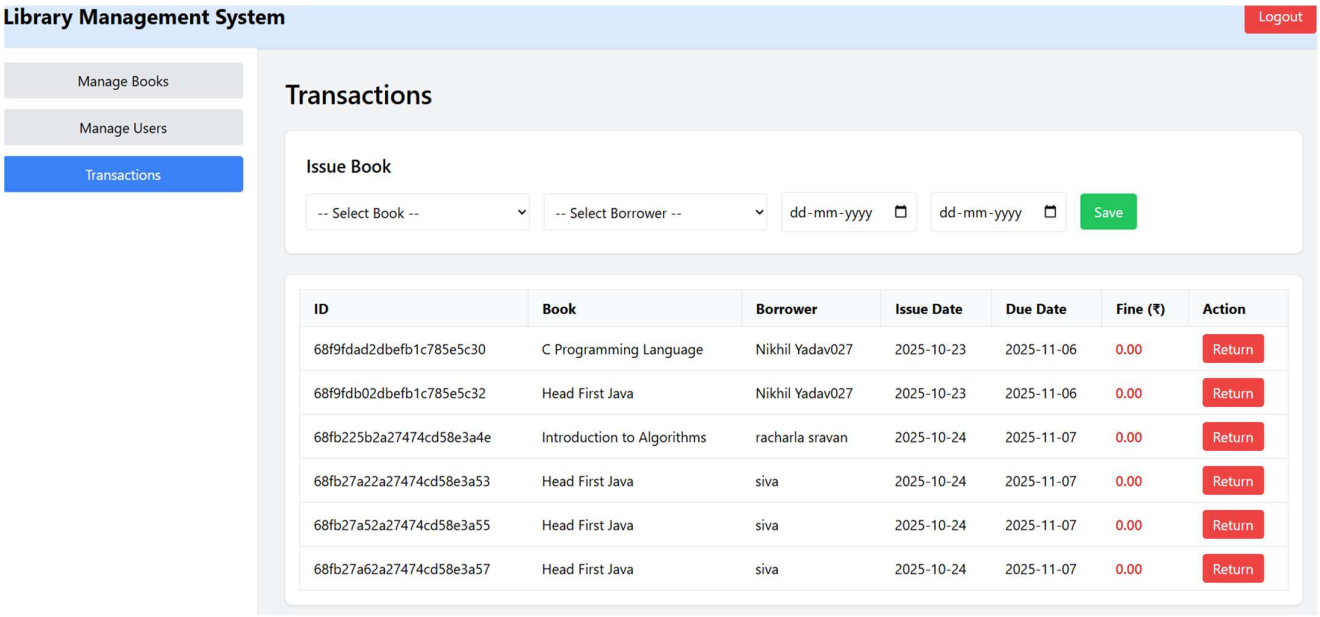
MANAGE BOOK PAGE (librarian):



MANAGE USER PAGE:



TRANSACTION PAGE:



Conclusion:

The Library Management System efficiently automates library operations, reducing manual effort and errors. It streamlines book management, student records, and transactions while ensuring accurate fine calculation making library services fast:er, more reliable, and user-friendly.