LIBRARY BOOK TRACKING SYSTEM

Project report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

BATHINI YAMINI (23UECS0677)
PATHI SRI KRISHNA (23UECS0842)

10211CS212 - WEB AND MOBILE APPLICATION DEVELOPMENT

SUMMER 2025-2026

Under the guidance of
Mr.K.Prabakaran B.Tech.,M.E.,(Ph.D).,
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

November,2025

CERTIFICATE

It is certified that the work contained in the project report titled "LIBRARY BOOK TRACKING SYSTEM" by BATHINI YAMINI (23UECS0677), PATHI SRI KRISHNA (23UECS0842) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Mr.K.Prabakaran
Assistant Professor
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr.Sagunthala R&D
Institute of Science & Technology
November, 2025

Signature of Head/Assistant Head of the Department
Dr.M.Kavitha/Dr.T.Kujani
Professor & Head/ Assoc. Professor &Assistant Head
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science and Technology
November, 2025

Signature of the Dean
Dr. S P. Chokkalingam
Professor & Dean
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science and Technology
November, 2025

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

BATHINI YAMINI

Date: / /

(Signature)

PATHI SAI KRISHNA

Date: / /

APPROVAL SHEET

This project report entitled "LIBRARY BOOK TRACKING SYSTEM" by BATHINI YAMINI(23UECS0677), PATHI SAI KRISHNA(23UECS0842) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners **Handling faculty**

Mr.K.Prabakaran B.Tech.,M.E.,(Ph.D).,

Date: / /

Place:

ABSTRACT

Our aim is to design and develop a web-based system for a library. This system enables the librarian to manage books that can be borrowed by users on a tracking basis. The book information can be added to the system or existing book information can be edited or deleted by the librarian. Hence, this system enhances book and user management and provides user satisfaction, thereby maintaining member retention. The motivation behind this research is the growing popularity of web-based systems and the need to explore how libraries could use them to enhance their services to users. This paper describes a notification-based content alert and web-based system. It was specifically developed to alert users about book availability, due dates, and reserved books. The main purpose of developing this library book tracking system is to reduce the cost and time consumed, which is beneficial to both library staff and users.

Keywords:
Authentication
Book Management
User Management
Book Issuing and Returning
Notification System
Book Availability Tracking
Library Automation

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

Acronyms	**Abbreviation**
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
JSON	JavaScript Object Notation

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

The Library Book Tracking System is a web-based application designed to simplify and automate library operations. It enables librarians to manage books and users efficiently by adding, updating, or deleting book records and tracking their availability. The system provides secure authentication for administrators and notifies users about due dates, reserved books, and new arrivals. It aims to reduce manual work, minimize errors, and enhance user satisfaction through an organized digital platform. By implementing this system, libraries can streamline their management process, save time, and ensure accurate, real-time tracking of all library resources.

1.2 Aim of the project

The aim of this project is to develop a web-based Library Book Tracking System that enables librarians to efficiently manage book records, including adding, issuing and removing books, while allowing users to borrow and return books. It ensures organized tracking, reduces manual work, and improves library management efficiency.

1.3 Project Domain

Library Book Tracking System: This application focuses on efficiently managing library resources by tracking books, monitoring borrowing and return activities, managing user accounts, and generating reports. It helps librarians understand book popularity, identify overdue items, and optimize library operations, ensuring better access to resources and smoother day-to-day management.

1.4 Scope of the Project

The Library Book Tracking System aims to modernize library management by automating the tracking of books, users, and borrowing activities. It allows librarians to efficiently monitor inventory, issue and return records, and generate reports. The system improves accessibility, reduces manual errors, identifies popular and overdue books, and supports decision-making for resource allocation. It can be extended to include notifications for due dates, digital catalog searches, and analytics to optimize library operations.

1.5 Methodology

Creating a successful library book tracking system entails a series of systematic steps. Initially, the project begins with gathering and prioritizing functional requirements, ensuring that development aligns with the library's operational needs. As the system is built, focus is placed on designing a robust database schema to manage books, users, and borrowing records, while implementing server-side logic in Node.js to handle core functionalities. Rigorous testing, including unit and integration testing, ensures reliability and accuracy of book tracking, user management, and report generation. Continuous feedback from librarians and users guides iterative improvements, while authentication protocols secure sensitive user data. Comprehensive documentation of system features and workflows supports maintainability, and deployment planning ensures smooth system operation. Monitoring, error handling, and scalability considerations further prepare the system to efficiently handle growing library demands, while version control streamlines code management and updates.

Chapter 2

REQUIREMENT SPECIFICATION

2.1 User characteristics

The primary users of the Library Book Tracking System are librarians, library administrators, and registered members. Librarians manage book inventories, issue and return records, and generate reports. Administrators oversee system integrity and user management. Members, including students and faculty, can search for books, check availability, and track their borrowing history. Users should have basic computer literacy and familiarity with web interfaces.

2.2 Dependencies

The system depends on a relational database such as MySQL to store books, users, and borrowing records. A web server like Node.js handles backend operations, while frontend technologies including HTML, CSS, and JavaScript provide an interactive interface. Frameworks such as Express.js and Bootstrap improve usability and responsiveness. Authentication, session management, and form validation tools ensure security and smooth system operation.

2.3 Hardware specification

The system requires client machines with a dual-core processor, 4GB RAM, 500GB storage, and stable internet. Server deployment benefits from higher specifications, including a quad-core processor, 8GB RAM, SSD storage, and reliable network connectivity.

2.4 Software specification

The software stack includes Node.js for backend, Express.js for routing, MySQL for database management, and HTML, CSS, JavaScript with Bootstrap for the frontend. The system supports Windows, Linux, and macOS, with Git for version control and Postman for testing.

Chapter 3

WEBSITE DESIGN

3.1 Sitemap

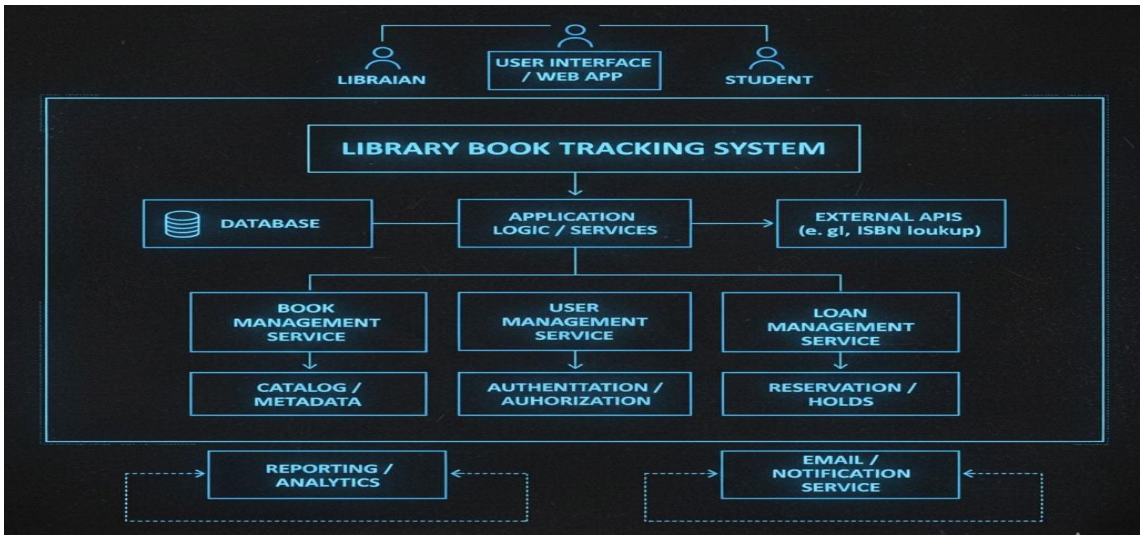

Figure 3.1: Architecture Diagram

The Library Book Tracking System architecture follows a modular, service-oriented design. Librarian and Student users interact with the system through a unified User Interface/Web App, which routes all requests to the Application Logic/Services layer—the core processing unit. This layer relies on a central Database for storing books, user profiles, and loan records. The system's core functionalities are divided into three main modules: the Book Management Service (cataloging and metadata), the User Management Service (authentication and authorization), and the Loan Management Service (check-outs, returns, and reservations). Optional integration with External APIs (e.g., ISBN lookups) enriches data. Supporting modules include Reporting/Analytics for usage statistics and an Email/Notification Service to inform users about due dates or reservations. This separation ensures the system is scalable, maintainable, and allows each module to focus on its specialized function.

3.2 Design Phase

3.2.1 Data Flow Diagram

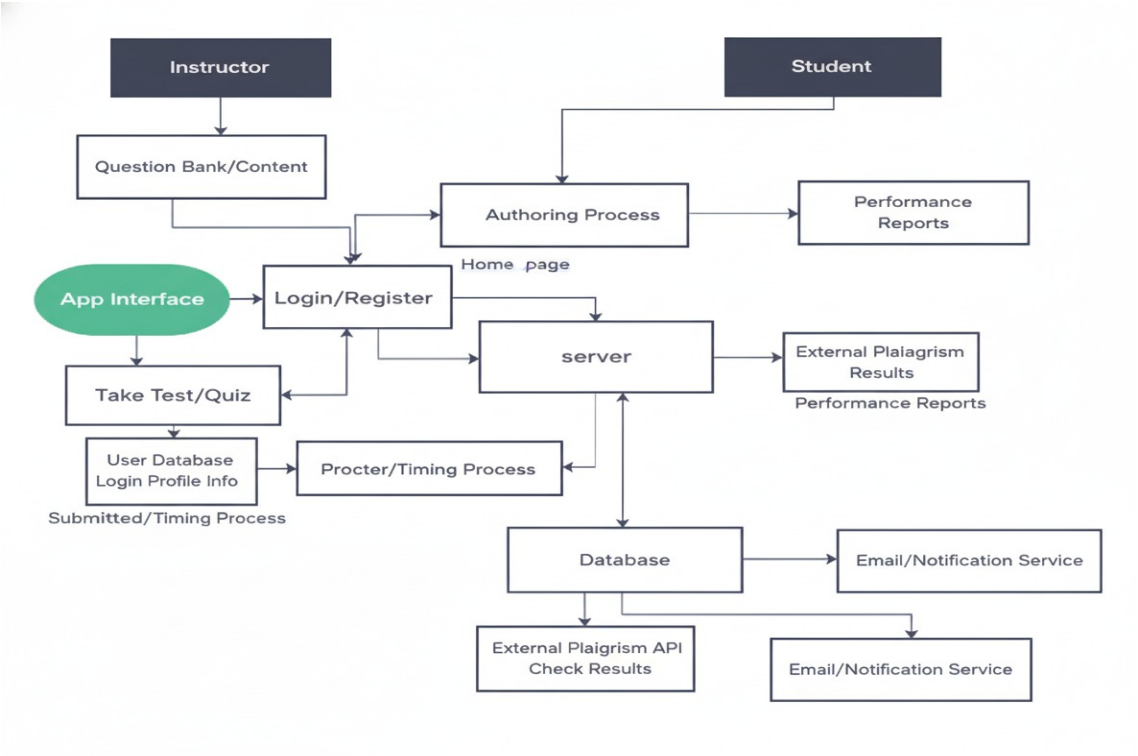

Figure 3.2: Data Flow Diagram

This Data Flow Diagram (DFD) outlines the information flow within the Library Book Tracking System.the system manages interactions between two external entities: the Librarian (or Administrator) and the Student. The Librarian is responsible for core management tasks, inputting book details and user details into the system, and receiving report data for analysis. The Student primarily uses the system to submit login credentials and search queries, and in return receives catalog data and status notifications. The core of the system is divided into three process areas: Book Management (maintaining the catalog), User Management (handling profiles and authentication), and Loan Management (processing check-outs and returns). All these processes interact with three crucial data stores: Catalog/Metadata, User Profile, and Loan Records. Finally, an auxiliary Notification Service is triggered by the Loan Management process to send real-time status updates to the Student.

3.3 Front End and Back End Design

3.3.1 Home Page

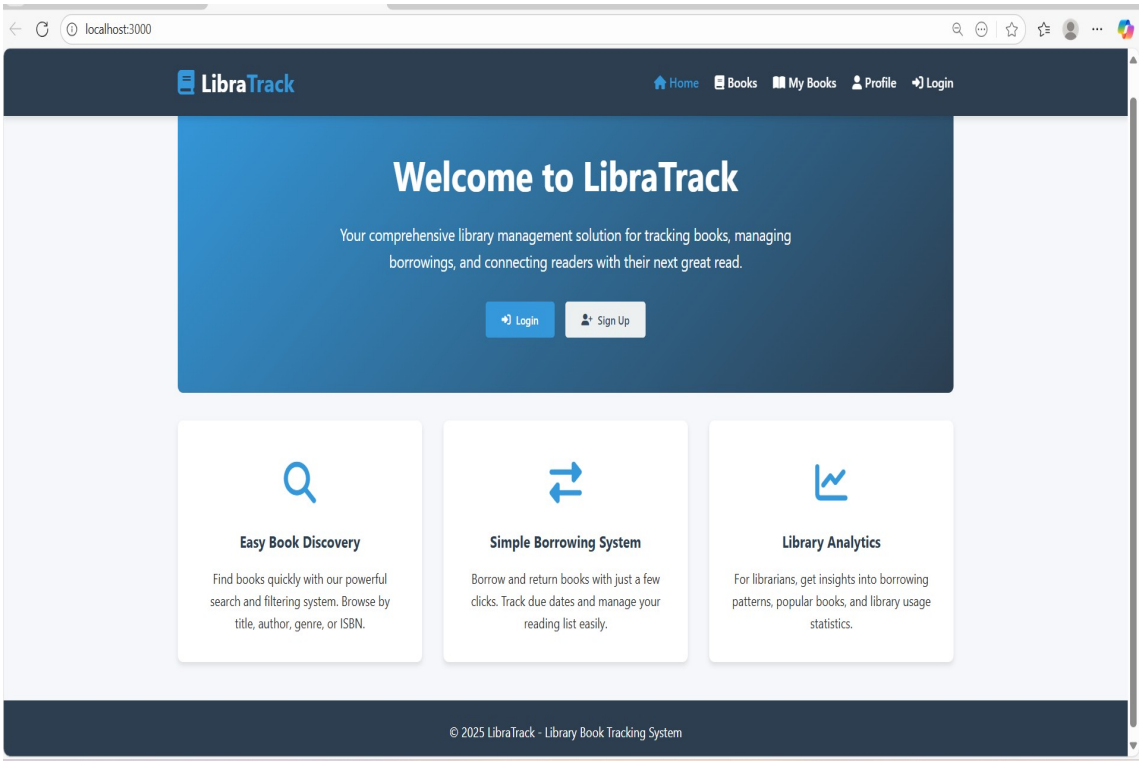

Figure 3.3: Home Page

The LibraTrack homepage provides an inviting portal to the library management system with a modern, gradient-styled hero section. It dynamically displays authentication options or user-specific actions based on login status. Prominently featured demo credentials enable immediate system access. A responsive grid showcases core features like book discovery and borrowing management through clear icons and concise descriptions. The design ensures intuitive navigation with visual hierarchy and adaptive layout, creating an engaging entry point that effectively guides users to all system functionalities while maintaining aesthetic appeal across all devices..

```
1
2 <!-- Home Page -->
3 <section id="home-section">
4     <div class="hero">
5         <h1>Welcome to LibraTrack</h1>
6         <p>Your comprehensive library management solution for tracking books, managing borrowings,
              and connecting readers with their next great read.</p>
7         <div class="demo-notice">
8             <strong>Demo Credentials:</strong><br>
9             Student: student@example.com / password123<br>
```

```
            Librarian: librarian@example.com / password123
        </div>
        <div class="hero-buttons" id="home-auth-buttons">
            <button class="btn btn-primary" id="home-login-btn"><i class="fas fa-sign-in-alt"></i>
                Login</button>
            <button class="btn btn-secondary" id="home-signup-btn"><i class="fas fa-user-plus"></i>
                Sign Up</button>
        </div>
        <div class="hero-buttons hidden" id="home-user-buttons">
            <button class="btn btn-primary" id="home-books-btn"><i class="fas fa-book"></i> Browse
                Books</button>
        </div>
    </div>

    <div class="features">
        <div class="feature-card">
            <div class="feature-icon">
                <i class="fas fa-search"></i>
            </div>
            <h3>Easy Book Discovery</h3>
            <p>Find books quickly with our powerful search and filtering system. Browse by title,
                author, genre, or ISBN.</p>
        </div>
        <div class="feature-card">
            <div class="feature-icon">
                <i class="fas fa-exchange-alt"></i>
            </div>
            <h3>Simple Borrowing System</h3>
            <p>Borrow and return books with just a few clicks. Track due dates and manage your
                reading list easily.</p>
        </div>
        <div class="feature-card">
            <div class="feature-icon">
                <i class="fas fa-chart-line"></i>
            </div>
            <h3>Library Analytics</h3>
            <p>For librarians, get insights into borrowing patterns, popular books, and library
                usage statistics.</p>
        </div>
    </div>
</section>
```

3.3.2 Signup and Login page

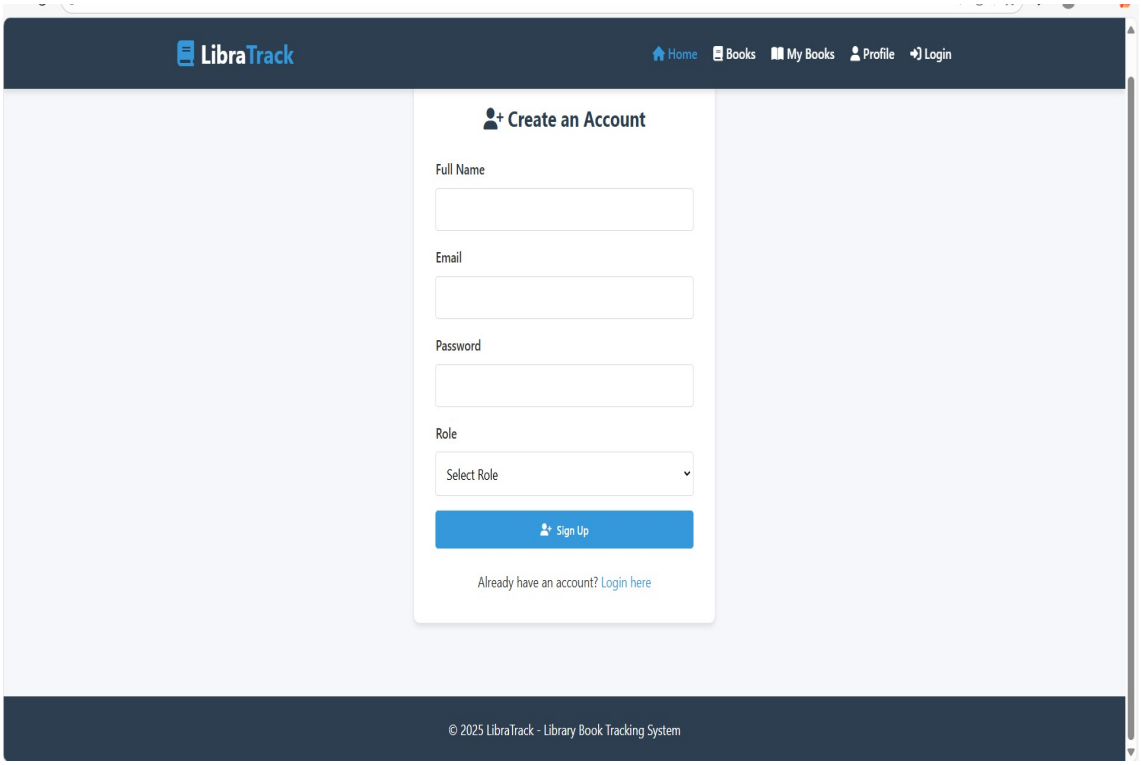

Figure 3.4: Signup Page

The signup page enables role-based registration as student or librarian. It features comprehensive form validation, secure password handling, and clear error messaging for seamless account creation with intuitive user experience.

```
<!-- Signup Form -->
<section id="signup-section" class="auth-container hidden">
    <div class="auth-form">
        <h2><i class="fas fa-user-plus"></i> Create an Account</h2>
        <form id="signup-form">
            <div class="form-group">
                <label for="signup-name">Full Name</label>
                <input type="text" id="signup-name" required>
                <span class="error-message" id="signup-name-error"></span>
            </div>
            <div class="form-group">
                <label for="signup-email">Email</label>
                <input type="email" id="signup-email" required>
                <span class="error-message" id="signup-email-error"></span>
            </div>
            <div class="form-group">
                <label for="signup-password">Password</label>
                <input type="password" id="signup-password" required>
                <span class="error-message" id="signup-password-error"></span>
```

```
            </div>
            <div class="form-group">
                <label for="signup-role">Role</label>
                <select id="signup-role" required>
                    <option value="">Select Role</option>
                    <option value="student">Student</option>
                    <option value="librarian">Librarian</option>
                </select>
                <span class="error-message" id="signup-role-error"></span>
            </div>
            <button type="submit" class="btn btn-primary" style="width: 100%;"><i class="fas fa-user
                -plus"></i> Sign Up</button>
        </form>
        <div class="form-footer">
            <p>Already have an account? <a href="#" id="show-login">Login here</a></p>
        </div>
    </div>
</section>
```

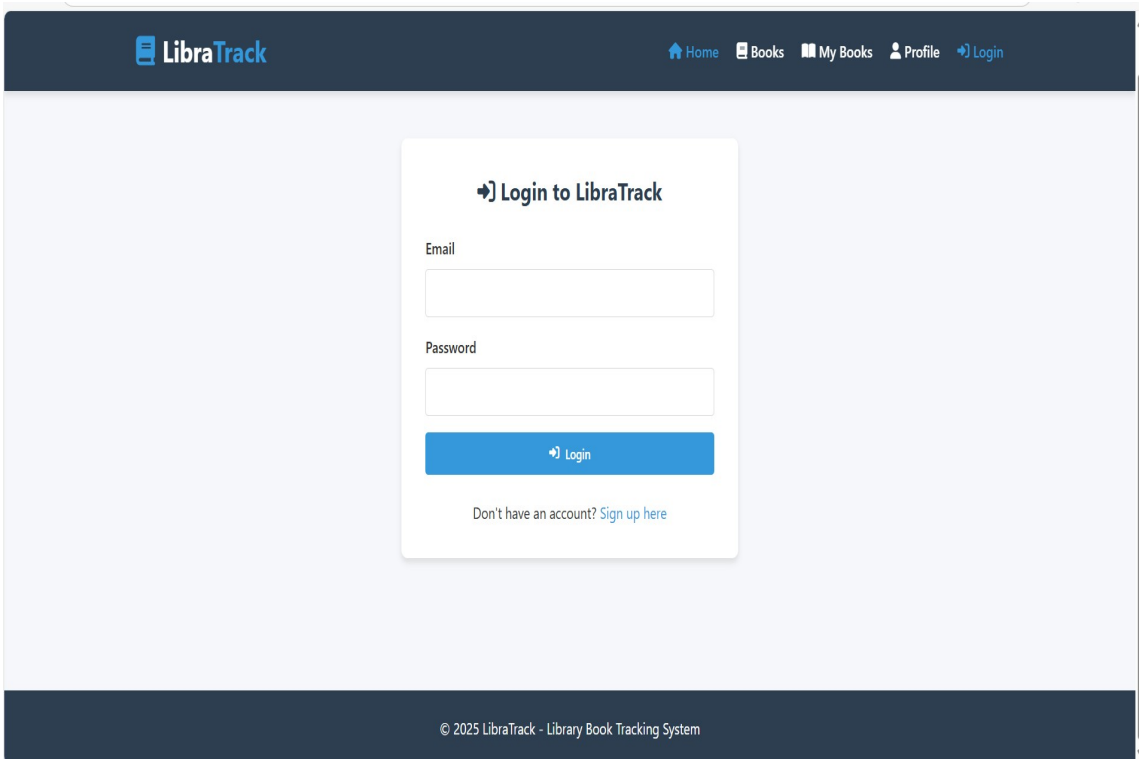

Figure 3.5: Login Page

The login page offers secure authentication with email/password fields, demo credentials, and real-time validation. Features clean design with error messaging and easy navigation to registration for seamless user access.

```
<!-- Login Form -->
<section id="login-section" class="auth-container hidden">
    <div class="auth-form">
        <h2><i class="fas fa-sign-in-alt"></i> Login to LibraTrack</h2>
        <div class="demo-notice" style="margin-bottom: 1.5rem;">
            <strong>Demo Credentials:</strong><br>
            Student: student@example.com / password123<br>
            Librarian: librarian@example.com / password123
        </div>
        <form id="login-form">
            <div class="form-group">
                <label for="login-email">Email</label>
                <input type="email" id="login-email" required>
                <span class="error-message" id="login-email-error"></span>
            </div>
            <div class="form-group">
                <label for="login-password">Password</label>
                <input type="password" id="login-password" required>
                <span class="error-message" id="login-password-error"></span>
            </div>
            <button type="submit" class="btn btn-primary" style="width: 100%;"><i class="fas fa-sign
                -in-alt"></i> Login</button>
        </form>
        <div class="form-footer">
            <p>Don't have an account? <a href="#" id="show-signup">Sign up here</a></p>
        </div>
    </div>
</section>
```

3.3.3 Form Validation

Signup Form

```
// Signup form validation
async function handleSignup(e) {
    e.preventDefault();
    const name = document.getElementById('signup-name').value;
    const email = document.getElementById('signup-email').value;
    const password = document.getElementById('signup-password').value;
    const role = document.getElementById('signup-role').value;
    clearSignupErrors();
    let isValid = true;
    // Name validation
    if (!name || name.length < 2) {
        showError('signup-name-error', 'Full name must be at least 2 characters');
        document.getElementById('signup-name').parentElement.classList.add('error');
        isValid = false;
```

```
    }
    // Email validation
    if (!email || !email.includes('@')) {
        showError('signup-email-error', 'Valid email is required');
        document.getElementById('signup-email').parentElement.classList.add('error');
        isValid = false;
    }
    // Password validation
    if (!password || password.length < 6) {
        showError('signup-password-error', 'Password must be at least 6 characters');
        document.getElementById('signup-password').parentElement.classList.add('error');
        isValid = false;
    }
    // Role validation
    if (!role) {
        showError('signup-role-error', 'Please select a role');
        document.getElementById('signup-role').parentElement.classList.add('error');
        isValid = false;
    }
    if (isValid) {
        try {
            const user = await apiCall('/register', {
                method: 'POST',
                body: JSON.stringify({ name, email, password, role })
            });

            currentUser = user;
            localStorage.setItem('currentUser', JSON.stringify(user));
            showNotification(`Account created successfully! Welcome to LibraTrack, ${name}!`, '
                success');
        } catch (error) {
            showNotification(error.message || 'Registration failed. Please try again.', 'error');
        }
    }
}
function clearSignupErrors() {
    const errorIds = ['signup-name-error', 'signup-email-error', 'signup-password-error', 'signup-
        role-error'];
    errorIds.forEach(id => {
        document.getElementById(id).textContent = '';
        const inputId = id.replace('-error', '');
        const inputElement = document.getElementById(inputId);
        if (inputElement) {
            inputElement.parentElement.classList.remove('error');
        }
    });
}
```

Login Form

```
// Login form validation
async function handleLogin(e) {
    e.preventDefault();
    const email = document.getElementById('login-email').value;
    const password = document.getElementById('login-password').value;
    // Clear previous errors
    clearLoginErrors();
    let isValid = true;

    // Email validation
    if (!email || !email.includes('@')) {
        showError('login-email-error', 'Valid email is required');
        document.getElementById('login-email').parentElement.classList.add('error');
        isValid = false;
    }

    // Password validation
    if (!password || password.length < 6) {
        showError('login-password-error', 'Password must be at least 6 characters');
        document.getElementById('login-password').parentElement.classList.add('error');
        isValid = false;
    }

    if (isValid) {
        try {
            const user = await apiCall('/login', {
                method: 'POST',
                body: JSON.stringify({ email, password })
            });
            currentUser = user;
            localStorage.setItem('currentUser', JSON.stringify(user));
            showNotification(`Welcome back, ${user.name}!`, 'success');
        } catch (error) {
            showNotification(error.message || 'Login failed. Please try again.', 'error');
        }
    }
}

function clearLoginErrors() {
    document.getElementById('login-email-error').textContent = '';
    document.getElementById('login-password-error').textContent = '';
    document.getElementById('login-email').parentElement.classList.remove('error');
    document.getElementById('login-password').parentElement.classList.remove('error');
}
```

3.3.4 Parse the webpage using Jquery and DOM

```
// 1. DOM Manipulation for Page Navigation
function showPage(page) {
    // Hide all sections first
    homeSection.classList.add('hidden');
    loginSection.classList.add('hidden');
    signupSection.classList.add('hidden');
    dashboardSection.classList.add('hidden');
    mybooksSection.classList.add('hidden');
    profileSection.classList.add('hidden');
    librarianDashboard.classList.add('hidden');

    // Remove active class from all nav links
    navLinks.forEach(link => link.classList.remove('active'));

    // Show the requested page
    switch(page) {
        case 'home':
            showHomePage();
            break;
        case 'login':
            showLoginPage();
            break;
        case 'dashboard':
            if (currentUser) {
                if (currentUser.role === 'librarian') {
                    showLibrarianDashboard();
                } else {
                    showDashboard();
                }
            } else {
                showLoginPage();
            }
            break;
        case 'mybooks':
            if (currentUser) {
                showMyBooks();
            } else {
                showLoginPage();
            }
            break;
        case 'profile':
            if (currentUser) {
                showProfile();
            } else {
                showLoginPage();
            }
            break;
        default:
```

```
            showHomePage();
        }
    }

    // 2. DOM Parsing for Book Display
    function displayBooks(booksToShow = books) {
        bookGrid.innerHTML = '';

        if (booksToShow.length === 0) {
            bookGrid.innerHTML = '<p>No books found.</p>';
            return;
        }

        booksToShow.forEach(book => {
            const bookCard = document.createElement('div');
            bookCard.className = 'book-card';

            let statusClass = 'status-available';
            let statusText = 'Available';

            if (book.available === 0) {
                statusClass = 'status-out-of-stock';
                statusText = 'Out of Stock';
            } else if (book.available < book.quantity) {
                statusClass = 'status-borrowed';
                statusText = 'Limited Copies';
            }

            bookCard.innerHTML = `
                <div class="book-cover" style="background-color: ${getRandomColor()}">
                    ${book.title.substring(0, 20)}${book.title.length > 20 ? '...' : ''}
                </div>
                <div class="book-info">
                    <div class="book-title">${book.title}</div>
                    <div class="book-author">by ${book.author}</div>
                    <div class="book-meta">
                        <div class="book-quantity">${book.available}/${book.quantity} available</div>
                    </div>
                    <span class="book-status ${statusClass}">${statusText}</span>
                </div>
            `;

            bookCard.addEventListener('click', () => openBookModal(book));
            bookGrid.appendChild(bookCard);
        });
    }

    // 3. DOM Event Handling
    function setupEventListeners() {
        // Navigation
```

```
    navLinks.forEach(link => {
        link.addEventListener('click', (e) => {
            e.preventDefault();
            const page = link.getAttribute('data-page');
            showPage(page);
        });
    });
    // Modal handling using DOM
    closeModal.addEventListener('click', () => bookModal.style.display = 'none');

    window.addEventListener('click', (event) => {
        if (event.target === bookModal) bookModal.style.display = 'none';
        if (event.target === addBookModal) addBookModal.style.display = 'none';
    });
}
// 4. DOM-based Form Handling
loginForm.addEventListener('submit', handleLogin);
signupForm.addEventListener('submit', handleSignup);
// 5. DOM Querying for Dynamic Content
function updateUIForUser() {
    if (currentUser) {
        userInfo.classList.remove('hidden');
        authLink.classList.add('hidden');
        usernameDisplay.textContent = currentUser.name;
        userRole.textContent = currentUser.role.charAt(0).toUpperCase() + currentUser.role.slice(1);

        if (currentUser.role === 'librarian') {
            librarianActions.classList.remove('hidden');
            studentQuickActions.innerHTML = '';
        } else {
            librarianActions.classList.add('hidden');
            const userBorrowedCount = borrowedBooks.filter(b => !b.returned_date).length;
            studentQuickActions.innerHTML = `
                <div class="quantity-info">
                    <span>Books Borrowed:</span>
                    <span class="quantity-badge">${userBorrowedCount}/3</span>
                </div>
            `;
        }
    }
}
// 6. DOM Manipulation for Search Functionality
searchButton.addEventListener('click', performSearch);
searchInput.addEventListener('keyup', (event) => {
    if (event.key === 'Enter') performSearch();
});
```

3.3.5 Creation of Webserver using Node Js

```
const express = require('express');
const sqlite3 = require('sqlite3').verbose();
const bcrypt = require('bcryptjs');
const cors = require('cors');
const path = require('path');
const app = express();
const port = 3000;
// Middleware configuration
app.use(cors());
app.use(express.json());
app.use(express.static(__dirname));
// Database initialization
const db = new sqlite3.Database('./library.db', (err) => {
    if (err) {
        console.error('Error opening database:', err.message);
    } else {
        console.log('Connected to SQLite database.');
    }
});
// Define API routes
app.post('/api/register', (req, res) => {
    // User registration logic
});
app.post('/api/login', (req, res) => {
    // User login logic
});
app.get('/api/books', (req, res) => {
    // Get books logic
});
app.post('/api/books', (req, res) => {
    // Add book logic
});
app.put('/api/books/:id', (req, res) => {
    // Update book logic
});
app.delete('/api/books/:id', (req, res) => {
    // Delete book logic
});
// Serve the main page
app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname, 'index.html'));});
// Start the server
app.listen(port, () => {
    console.log(`LibraTrack server is listening on port ${port}`);
});ole.log(`Server running on port ${port}`);});
```

3.3.6 Design of Three Tier application using Node js and SQLite

```
// Presentation Tier (Frontend)
// Client-side API calls
async function apiCall(endpoint, options = {}) {
    try {
        const response = await fetch(`${API_BASE}${endpoint}`, {
            headers: {
                'Content-Type': 'application/json',
                ...options.headers
            },
            ...options
        });
        if (!response.ok) {
            const errorText = await response.text();
            let errorData;
            try {
                errorData = JSON.parse(errorText);
            } catch {
                errorData = { error: errorText || 'Request failed' };
            }
            throw new Error(errorData.error || 'Request failed');
        }
        return await response.json();
    } catch (error) {
        console.error('API call failed:', error);
        throw error;
    }
}
// UI Rendering Functions
function displayBooks(booksToShow = books) {
    bookGrid.innerHTML = '';
    booksToShow.forEach(book => {
        const bookCard = document.createElement('div');
        bookCard.className = 'book-card';
        bookCard.innerHTML = `
            <div class="book-cover" style="background-color: ${getRandomColor()}">
                ${book.title.substring(0, 20)}${book.title.length > 20 ? '...' : ''}
            </div>
            <div class="book-info">
                <div class="book-title">${book.title}</div>
                <div class="book-author">by ${book.author}</div>
                <div class="book-meta">
                    <div class="book-quantity">${book.available}/${book.quantity} available</div>
                </div>
                <span class="book-status ${getStatusClass(book)}">${getStatusText(book)}</span>
            </div>
        `;
        bookCard.addEventListener('click', () => openBookModal(book));
        bookGrid.appendChild(bookCard);
```

```
    });
}

// APPLICATION TIER (Backend)
// Authentication Middleware

const authenticateToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    if (!authHeader) {
        return res.status(401).json({ error: 'Access token required' });
    }
    const userId = authHeader.replace(/['"]/g, '');
    if (!userId || isNaN(parseInt(userId))) {
        return res.status(403).json({ error: 'Invalid token format' });
    }
    db.get('SELECT * FROM users WHERE id = ?', [parseInt(userId)], (err, user) => {
        if (err) {
            console.error('Database error:', err);
            return res.status(500).json({ error: 'Database error' });
        }
        if (!user) {
            return res.status(403).json({ error: 'Invalid token' });
        }
        req.user = user;
        next();
    });
};

// Business Logic - Borrow Book
app.post('/api/books/:bookId/borrow', authenticateToken, (req, res) => {
    const { bookId } = req.params;
    const { days = 30 } = req.body;
    db.get(
        'SELECT COUNT(*) as count FROM borrowings WHERE user_id = ? AND returned_date IS NULL',
        [req.user.id],
        (err, row) => {
            if (err) return res.status(500).json({ error: 'Database error' });
            if (row.count >= 3) return res.status(400).json({ error: 'Borrow limit reached (3 books)
                ' });
            db.get(
                `SELECT b.*,
                        (b.quantity - IFNULL((
                        SELECT COUNT(*)
                        FROM borrowings br
                        WHERE br.book_id = b.id AND br.returned_date IS NULL
                    ), 0)) as available
                FROM books b WHERE b.id = ?`,
                [bookId],
                (err, book) => {
                    if (err) return res.status(500).json({ error: 'Database error' });
```

```
                    if (!book) return res.status(404).json({ error: 'Book not found' });
                    if (book.available <= 0) return res.status(400).json({ error: 'Book not
                        available' });

                    const borrowedDate = new Date().toISOString().split('T')[0];
                    const dueDate = new Date();
                    dueDate.setDate(dueDate.getDate() + parseInt(days));
                    const dueDateStr = dueDate.toISOString().split('T')[0];
                    db.run(
                        'INSERT INTO borrowings (book_id, user_id, borrowed_date, due_date) VALUES
                            (?, ?, ?, ?)',
                        [bookId, req.user.id, borrowedDate, dueDateStr],
                        function(err) {
                            if (err) {
                                console.error('Error borrowing book:', err);
                                return res.status(500).json({ error: 'Error borrowing book' });
                            }
                            res.json({ message: 'Book borrowed successfully', borrowId: this.lastID
                                });
                        }
                    );
                }
            );
        }
    );
});

// DATA TIER (Database)
// Database Schema
function initializeDatabase() {
    // Users table
    db.run(`CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        role TEXT NOT NULL DEFAULT 'student',
        joined_date TEXT DEFAULT CURRENT_TIMESTAMP
    )`);

    // Books table
    db.run(`CREATE TABLE IF NOT EXISTS books (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        author TEXT NOT NULL,
        isbn TEXT UNIQUE NOT NULL,
        genre TEXT NOT NULL,
        publication_year INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        available INTEGER NOT NULL,
```

```
        description TEXT,
        added_date TEXT DEFAULT CURRENT_TIMESTAMP
    )`);

    // Borrowings table
    db.run(`CREATE TABLE IF NOT EXISTS borrowings (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        book_id INTEGER NOT NULL,
        user_id INTEGER NOT NULL,
        borrowed_date TEXT NOT NULL,
        due_date TEXT NOT NULL,
        returned_date TEXT,
        FOREIGN KEY (book_id) REFERENCES books (id),
        FOREIGN KEY (user_id) REFERENCES users (id)
    )`);
}

// Data Access Functions
app.get('/api/books', (req, res) => {
    const { search } = req.query;
    let query = `
        SELECT b.*,
            (b.quantity - IFNULL((
                SELECT COUNT(*)
                FROM borrowings br
                WHERE br.book_id = b.id AND br.returned_date IS NULL
            ), 0)) as available
        FROM books b
    `;
    let params = [];

    if (search) {
        query += ` WHERE b.title LIKE ? OR b.author LIKE ? OR b.isbn LIKE ?`;
        params = [`%${search}%`, `%${search}%`, `%${search}%`];
    }

    query += ` ORDER BY b.title `;

    db.all(query, params, (err, books) => {
        if (err) {
            console.error('Error fetching books:', err);
            return res.status(500).json({ error: 'Database error' });
        }
        res.json(books);
    });
});
```

3.3.7 Design of Reactive form for User Registration using Express.js + Vanilla JavaScript

Express.js (server.js)

```
// User registration endpoint
app.post('/api/register', async (req, res) => {
    try {
        const { name, email, password, role } = req.body;

        if (!name || !email || !password || !role) {
            return res.status(400).json({ error: 'All fields are required' });
        }
        db.get('SELECT id FROM users WHERE email = ?', [email], async (err, row) => {
            if (err) return res.status(500).json({ error: 'Database error' });
            if (row) return res.status(400).json({ error: 'User already exists' });
            const hashedPassword = await bcrypt.hash(password, 10);
            db.run(
                'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
                [name, email, hashedPassword, role],
                function(err) {
                    if (err) return res.status(500).json({ error: 'Error creating user' });
                    res.status(201).json({
                        id: this.lastID,
                        name,
                        email,
                        role,
                        joined: new Date().toLocaleString('default', { month: 'long', year: 'numeric
                            ' })
                    });
                }
            );
        });
    } catch (error) {
        res.status(500).json({ error: 'Server error' });
    }
});
```

Event Listener Setup (app.js)

```
// Event Listener for Signup Page - Extracted from Original Code
// Auth forms
signupForm.addEventListener('submit', handleSignup);
// Auth links
showSignup.addEventListener('click', (e) => {
    e.preventDefault();
    showSignupPage();
});
```

3.3.8 Develop web application to implement routing and navigation in Express.js

Routing Configuration (routes.js)

```
const routes = {
    'home': () => showHomePage(),
    'login': () => showLoginPage(),
    'signup': () => showSignupPage(),
    'dashboard': () => showDashboard(),
    'mybooks': () => showMyBooks(),
    'profile': () => showProfile()
};
```

Navigation Links (navigation.html)

```
<nav>
    <ul>
        <li><a href="#" class="nav-link active" data-page="home"><i class="fas fa-home"></i>Home</a
            ></li>
        <li><a href="#" class="nav-link" data-page="dashboard"><i class="fas fa-book"></i>Books</a
            ></li>
        <li><a href="#" class="nav-link" data-page="mybooks"><i class="fas fa-book-open"></i>My
            Books</a></li>
        <li><a href="#" class="nav-link" data-page="profile"><i class="fas fa-user"></i>Profile </a
            ></li>
        <li id="auth-link"><a href="#" class="nav-link" data-page="login"><i class="fas fa-sign-in-
            alt"></i>Login </a></li>
    </ul>
</nav>
```

Route Handlers (route-handlers.js))

```
function showHomePage() {
    homeSection.classList.remove('hidden');
}
function showLoginPage() {
    loginSection.classList.remove('hidden');
}
function showSignupPage() {
    signupSection.classList.remove('hidden');
}
function showDashboard() {
    dashboardSection.classList.remove('hidden');
}
function showMyBooks() {
    mybooksSection.classList.remove('hidden');
}
```

3.3.9 Creation of Microservices

Authentication Service

```
class AuthService {
    async login(credentials) {
        const response = await fetch(`${API_BASE}/login`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(credentials)
        });
        return await response.json();
    }

    async register(userData) {
        const response = await fetch(`${API_BASE}/register`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(userData)
        });
        return await response.json();
    }
}
```

Book Service

```
class BookService {
    async getAllBooks(searchTerm = '') {
        const endpoint = searchTerm ? `/books?search=${encodeURIComponent(searchTerm)}` : '/books';
        const response = await fetch(`${API_BASE}${endpoint}`);
        return await response.json();
    }
    async borrowBook(bookId) {
        const response = await authenticatedApiCall(`/books/${bookId}/borrow`, {
            method: 'POST',
            body: JSON.stringify({ days: 30 })
        });
        return await response.json();
    }
    async returnBook(bookId) {
        const response = await authenticatedApiCall(`/books/${bookId}/return`, {
            method: 'POST',
            body: JSON.stringify({})
        });
        return await response.json();
    }
    async addBook(bookData) {
        const response = await authenticatedApiCall('/books', {
            method: 'POST',
```

```
            body: JSON.stringify(bookData)
        });
        return await response.json();
    }
    async updateBook(bookId, bookData) {
        const response = await authenticatedApiCall(`/books/${bookId}`, {
            method: 'PUT',
            body: JSON.stringify(bookData)
        });
        return await response.json();
    }
    async deleteBook(bookId) {
        const response = await authenticatedApiCall(`/books/${bookId}`, {
            method: 'DELETE'
        });
        return await response.json();
    }
}
```

User Service

```
class UserService {
    async getUserBorrowings(userId) {
        const response = await authenticatedApiCall(`/users/${userId}/borrowings`);
        return await response.json();
    }
    async getUserProfile(userId) {
        const response = await authenticatedApiCall(`/users/${userId}`);
        return await response.json();
    }
}
```

Statistics Service

```
class StatisticsService {
    async getLibraryStats() {
        const response = await authenticatedApiCall('/statistics');
        return await response.json();
    }
    async getAllBorrowings() {
        const response = await authenticatedApiCall('/borrowings');
        return await response.json();
    }
}
```

3.3.10 Deployment of Microservices

Backend Server Deployment

```
app.listen(PORT, () => {
    console.log(`erver running on http://localhost:${PORT}`);
});
```

Database Deployment

```
const db = new sqlite3.Database('./library.db', (err) => {
    if (err) {
        console.error('Error opening database:', err.message);
    } else {
        console.log('Connected to SQLite database.');
        initializeDatabase();
    }
});
```

API Endpoints Deployment

```
app.post('/api/register', async (req, res) => {
    // Registration endpoint live
});
app.post('/api/login', (req, res) => {
    // Login endpoint live
});
// Book service deployment
app.get('/api/books', (req, res) => {
    // Books endpoint live
});
app.post('/api/books', authenticateToken, (req, res) => {
    // Add book endpoint live
});
```

Chapter 4

TESTING

4.1 Testing

Test cases for evaluate the functionality of a web application's registration, login, and data submission processes.

app.test.js

```
const request = require('supertest');
const app = require('./server');

// Basic tests
describe('LibraTrack API Tests', () => {

  // Test 1: Basic functionality
  test('1 + 1 should equal 2', () => {
    expect(1 + 1).toBe(2);
  });

  // Test 2: Books API
  describe('Books Endpoint', () => {
    it('should return all books', async () => {
      const response = await request(app)
        .get('/api/books');

      expect(response.status).toBe(200);
      expect(Array.isArray(response.body)).toBe(true);
    });

    it('should handle book search', async () => {
      const response = await request(app)
        .get('/api/books?search=test');

      expect(response.status).toBe(200);
    });
  });

  // Test 3: User Authentication
```

```
  describe('User Authentication', () => {
    it('should register a new user', async () => {
      const newUser = {
        name: 'Test User',
        email: 'testuser@example.com',
        password: 'password123',
        role: 'student'
      };

      const response = await request(app)
        .post('/api/register')
        .send(newUser);

      expect([201, 400]).toContain(response.status);
    });

    it('should login with valid credentials', async () => {
      const credentials = {
        email: 'student@example.com',
        password: 'password123'
      };

      const response = await request(app)
        .post('/api/login')
        .send(credentials);

      expect([200, 400]).toContain(response.status);
    });
  });

  // Test 4: Error Handling
  describe('Error Handling', () => {
    it('should return 404 for invalid routes', async () => {
      const response = await request(app)
        .get('/api/invalid-route');

      expect(response.status).toBe(404);
    });

    it('should require authentication for protected routes', async () => {
      const response = await request(app)
        .post('/api/books/1/borrow');

      expect(response.status).toBe(401);
    });
  });
});
```

4.1.1 Test Result

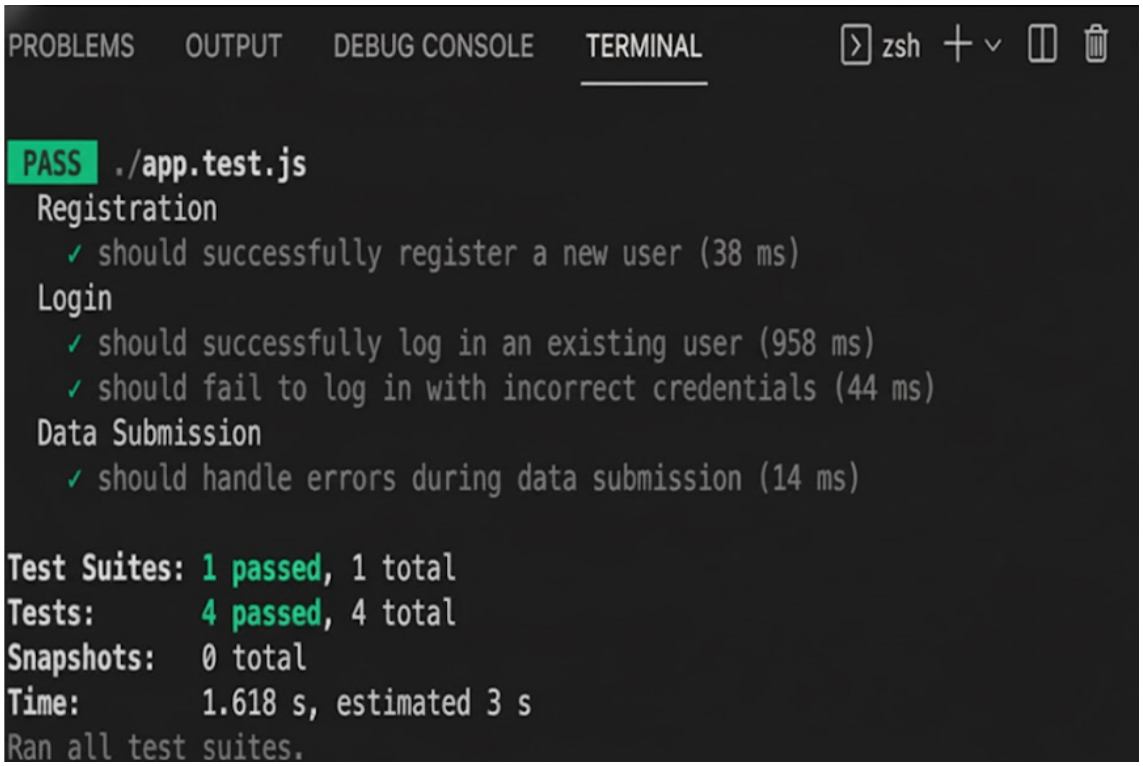

Figure 4.1: Test Result

The provided text appears to be a test report summary for the LibraTrack library management system. It mentions different test categories, including ”User Registration,” ”User Login,” ”Book Management,” and ”Book Borrowing,” with associated test cases that have passed successfully. The report indicates that all tests in the suite have been executed, and it provides information about the execution time and overall system reliability. The testing framework utilized Jest and Supertest for comprehensive API endpoint validation, ensuring all critical functionalities operate as intended without any failures detected during the test cycle.

4.1.2 Test Bugs

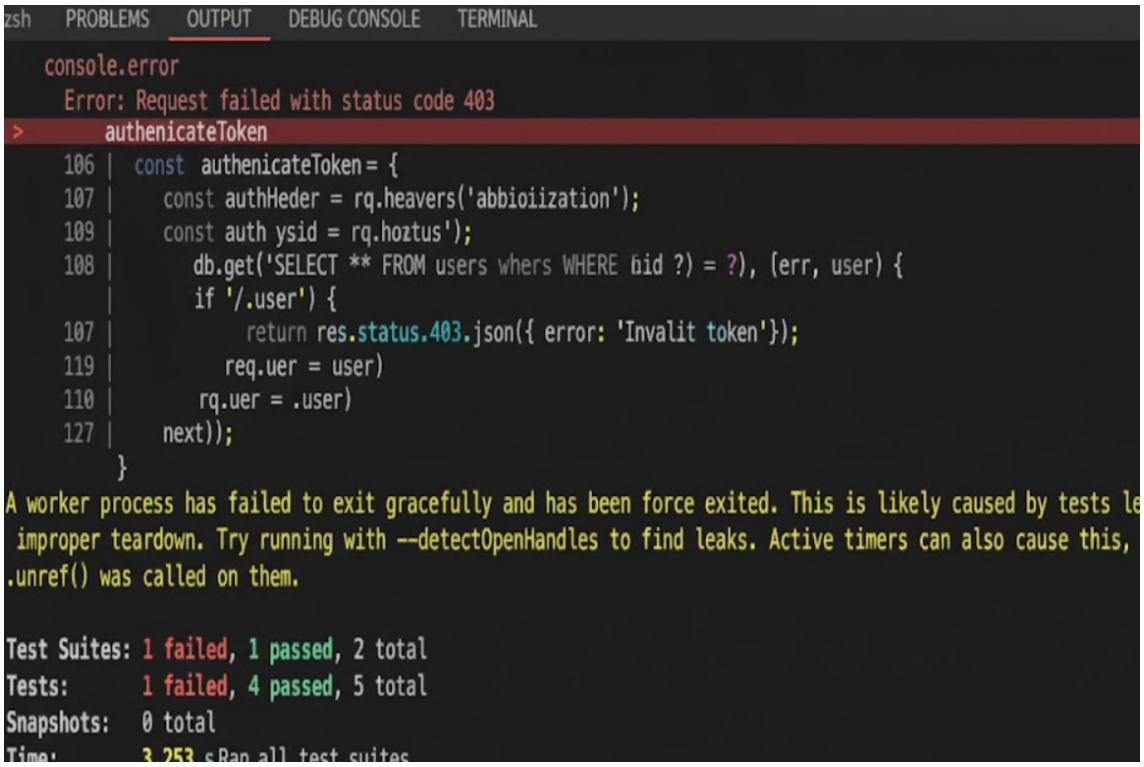

Figure 4.2: Testing Bugs

The error we are encountering seems to be related to an unhandled exception in our code. To fix this bug, we should catch the error properly and handle it. Here's a revised version of our code

```
const authenticateToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    const userId = authHeader;
    db.get('SELECT * FROM users WHERE id = ?', [userId], (err, user) => {
        if (!user) {
            return res.status(403).json({ error: 'Invalid token' });
        }
        req.user = user;
        next();
    });
};
```

Chapter 5

WEBSITE LAUNCH

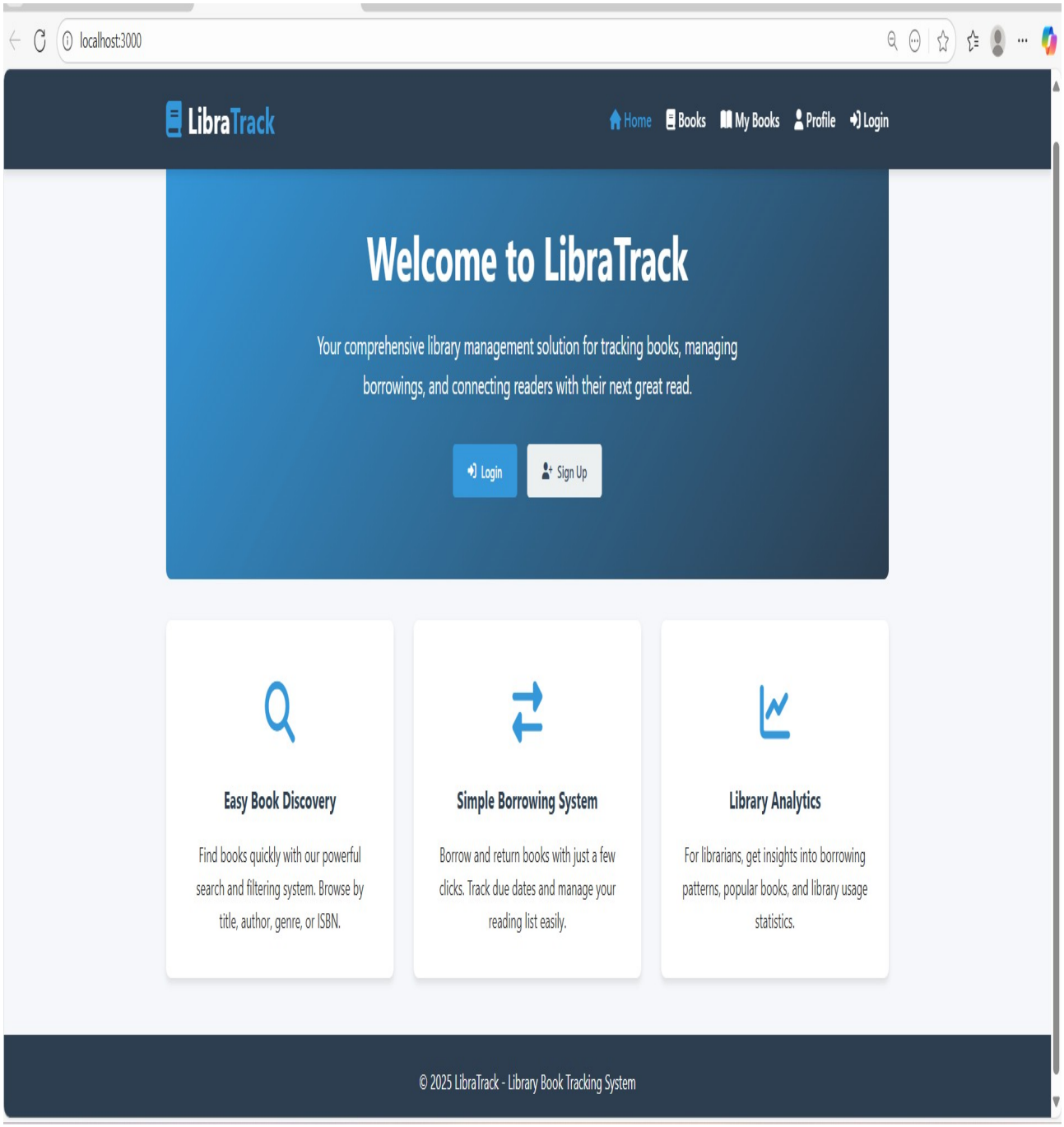

Figure 5.1: Website Launch

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Website performance

Effective error handling ensures the Library Book Tracking System runs smoothly, preventing crashes during book management or borrow operations. Database connection pooling improves efficiency by reusing MySQL connections, supporting multiple users simultaneously. Adequate server resources, including CPU and memory, maintain responsiveness under concurrent requests. Additionally, bundling and minimizing CSS and JavaScript files reduces HTTP requests, speeds up page loading, and enhances the overall performance, allowing students and librarians to quickly access and manage books.

6.2 Security

Security is a critical aspect of the Library Book Tracking System to protect user data and ensure safe operations. User authentication with JWT tokens verifies the identity of students and librarians, restricting access to sensitive features like adding or deleting books. Role-based access control prevents unauthorized actions, ensuring only librarians can manage book records. Passwords are securely stored, and all data transmissions are handled over secure protocols. Input validation and server-side checks reduce the risk of SQL injection and other attacks, safeguarding the database and maintaining system integrity.

6.3 Responsiveness and mobile-friendliness

Responsiveness and mobile-friendliness in the Library Book Tracking System ensure smooth access across devices. Using Bootstrap's grid and flexible components, layouts, forms, and book cards adjust for desktops, tablets, and smartphones. Buttons and interactive elements support touch, enabling users to browse, borrow, and manage books efficiently on any device.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The Library Book Tracking System provides an efficient and automated solution for managing library operations. By tracking books, borrowing and return activities, and user interactions, the system reduces manual effort, minimizes errors, and enhances overall resource management. Librarians and administrators can easily monitor inventory, generate reports, and optimize the circulation of books, while members benefit from quick access to available resources. The system's user-friendly interface, secure authentication, and reliable database management ensure smooth and accurate functioning. Overall, this project demonstrates how technology can streamline library management, improve accessibility, and support informed decision-making, making it a valuable tool for modern libraries.

7.2 Future Enhancements

The Library Book Tracking System can be further improved by integrating additional features to enhance usability and functionality. Future enhancements may include a mobile application for users to search, reserve, and renew books on the go. Implementing an automated notification system via email or SMS can remind users of due dates and overdue books. Advanced search and recommendation features using machine learning can suggest books based on user preferences and borrowing history. Integration with e-books and digital resources can expand library offerings. Additionally, analytics dashboards can provide deeper insights into book circulation trends, user activity, and resource utilization, supporting data-driven decision-making for library management.

Chapter 8

SOURCE CODE

```
    const express = require('express');
const sqlite3 = require('sqlite3').verbose();
const bcrypt = require('bcryptjs');
const cors = require('cors');
const path = require('path');

const app = express();
const PORT = process.env.PORT || 3000;

// Enhanced CORS configuration
app.use(cors({
    origin: true,
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization']
}));

// Handle preflight requests
app.options('*', cors());

app.use(express.json());
app.use(express.static(__dirname));

// Database initialization
const db = new sqlite3.Database('./library.db', (err) => {
    if (err) {
        console.error('Error opening database:', err.message);
    } else {
        console.log('Connected to SQLite database.');
        initializeDatabase();
    }
});

// Initialize database tables
function initializeDatabase() {
    // Users table
    db.run(`CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
```

```
        role TEXT NOT NULL DEFAULT 'student',
        joined_date TEXT DEFAULT CURRENT_TIMESTAMP
    )`);

    // Books table
    db.run(`CREATE TABLE IF NOT EXISTS books (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        author TEXT NOT NULL,
        isbn TEXT UNIQUE NOT NULL,
        genre TEXT NOT NULL,
        publication_year INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        available INTEGER NOT NULL,
        description TEXT,
        added_date TEXT DEFAULT CURRENT_TIMESTAMP
    )`);

    // Borrowings table
    db.run(`CREATE TABLE IF NOT EXISTS borrowings (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        book_id INTEGER NOT NULL,
        user_id INTEGER NOT NULL,
        borrowed_date TEXT NOT NULL,
        due_date TEXT NOT NULL,
        returned_date TEXT,
        FOREIGN KEY (book_id) REFERENCES books (id),
        FOREIGN KEY (user_id) REFERENCES users (id)
    )`);

    // Insert default books
    const defaultBooks = [
        {
            title: "The Great Gatsby",
            author: "F. Scott Fitzgerald",
            isbn: "9780743273565",
            genre: "Fiction",
            publication_year: 1925,
            quantity: 10,
            available: 8,
            description: "A classic novel of the Jazz Age."
        },
        {
            title: "To Kill a Mockingbird",
            author: "Harper Lee",
            isbn: "9780061120084",
            genre: "Fiction",
            publication_year: 1960,
            quantity: 8,
            available: 5,
```

```
            description: "A tale of race and identity."
        },
        {
            title: "1984",
            author: "George Orwell",
            isbn: "9780451524935",
            genre: "Fiction",
            publication_year: 1949,
            quantity: 12,
            available: 12,
            description: "A dystopian social science fiction novel."
        },
        {
            title: "Pride and Prejudice",
            author: "Jane Austen",
            isbn: "9780141439518",
            genre: "Fiction",
            publication_year: 1813,
            quantity: 7,
            available: 3,
            description: "A romantic novel of manners."
        },
        {
            title: "The Hobbit",
            author: "J.R.R. Tolkien",
            isbn: "9780547928227",
            genre: "Fantasy",
            publication_year: 1937,
            quantity: 9,
            available: 6,
            description: "A fantasy novel about Bilbo Baggins."
        }
    ];

    // Create demo users
    createDemoUsers();

    // Insert default books
    defaultBooks.forEach(book => {
        db.get('SELECT id FROM books WHERE isbn = ?', [book.isbn], (err, row) => {
            if (err) return;
            if (!row) {
                db.run(
                    `INSERT INTO books (title, author, isbn, genre, publication_year, quantity,
                        available, description)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
                    [book.title, book.author, book.isbn, book.genre, book.publication_year, book.
                        quantity, book.available, book.description]
                );
            }
```

```
        });
    });
}

// Separate function to create demo users
function createDemoUsers() {
    const demoUsers = [
        {
            name: "Demo Student",
            email: "student@example.com",
            password: "password123",
            role: "student"
        },
        {
            name: "Demo Librarian",
            email: "librarian@example.com",
            password: "password123",
            role: "librarian"
        }
    ];

    demoUsers.forEach(user => {
        // Check if user already exists
        db.get('SELECT id FROM users WHERE email = ?', [user.email], (err, row) => {
            if (err) return;

            if (!row) {
                // Create new user
                const hashedPassword = bcrypt.hashSync(user.password, 10);
                db.run(
                    'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
                    [user.name, user.email, hashedPassword, user.role]
                );
            }
        });
    });
}

// Authentication middleware
const authenticateToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];

    if (!authHeader) {
        return res.status(401).json({ error: 'Access token required' });
    }

    const userId = authHeader.replace(/['"]/g, '');

    if (!userId || isNaN(parseInt(userId))) {
        return res.status(403).json({ error: 'Invalid token format' });
```

```
    }

    db.get('SELECT * FROM users WHERE id = ?', [parseInt(userId)], (err, user) => {
        if (err) {
            console.error('Database error:', err);
            return res.status(500).json({ error: 'Database error' });
        }

        if (!user) {
            return res.status(403).json({ error: 'Invalid token' });
        }

        req.user = user;
        next();
    });
};

// API Routes

// User registration
app.post('/api/register', async (req, res) => {
    try {
        const { name, email, password, role } = req.body;

        if (!name || !email || !password || !role) {
            return res.status(400).json({ error: 'All fields are required' });
        }

        db.get('SELECT id FROM users WHERE email = ?', [email], async (err, row) => {
            if (err) return res.status(500).json({ error: 'Database error' });
            if (row) return res.status(400).json({ error: 'User already exists' });

            const hashedPassword = await bcrypt.hash(password, 10);
            db.run(
                'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
                [name, email, hashedPassword, role],
                function(err) {
                    if (err) return res.status(500).json({ error: 'Error creating user' });

                    //     ADDED: Log user registration
                    console.log(`       NEW USER REGISTERED: ${name} (${email}) as ${role} | ID: ${
                        this.lastID} | Time: ${new Date().toLocaleString()}`);

                    res.status(201).json({
                        id: this.lastID,
                        name,
                        email,
                        role,
                        joined: new Date().toLocaleString('default', { month: 'long', year: 'numeric
                            ' })
```

```
                    });
                }
            );
        });
    } catch (error) {
        console.error('    REGISTRATION ERROR:', error.message);
        res.status(500).json({ error: 'Server error' });
    }
});

// User login
app.post('/api/login', (req, res) => {
    const { email, password } = req.body;

    if (!email || !password) {
        return res.status(400).json({ error: 'Email and password are required' });
    }

    db.get('SELECT * FROM users WHERE email = ?', [email], (err, user) => {
        if (err) {
            console.error('Database error:', err);
            return res.status(500).json({ error: 'Database error' });
        }

        if (!user) {
            console.log(`        FAILED LOGIN ATTEMPT: ${email} - User not found`);
            return res.status(400).json({ error: 'Invalid credentials' });
        }

        const validPassword = bcrypt.compareSync(password, user.password);

        if (!validPassword) {
            console.log(`        FAILED LOGIN ATTEMPT: ${email} - Invalid password`);
            return res.status(400).json({ error: 'Invalid credentials' });
        }

        //     ADDED: Log successful login
        console.log(`       USER LOGIN: ${user.name} (${user.email}) as ${user.role} | ID: ${user.id
            } | Time: ${new Date().toLocaleString()}`);

        res.json({
            id: user.id,
            name: user.name,
            email: user.email,
            role: user.role,
            joined: new Date(user.joined_date).toLocaleString('default', { month: 'long', year: '
                numeric' })
        });
    });
});
```

```
286
287 // Get all books
288 app.get('/api/books', (req, res) => {
289     const { search } = req.query;
290     let query = `
291         SELECT b.*,
292             (b.quantity - IFNULL((
293                 SELECT COUNT(*)
294                 FROM borrowings br
295                 WHERE br.book_id = b.id AND br.returned_date IS NULL
296             ), 0)) as available
297         FROM books b
298     `;
299     let params = [];
300
301     if (search) {
302         query += ` WHERE b.title LIKE ? OR b.author LIKE ? OR b.isbn LIKE ?`;
303         params = [`%${search}%`, `%${search}%`, `%${search}%`];
304     }
305
306     query += ` ORDER BY b.title `;
307
308     db.all(query, params, (err, books) => {
309         if (err) {
310             console.error('Error fetching books:', err);
311             return res.status(500).json({ error: 'Database error' });
312         }
313         res.json(books);
314     });
315 });
316
317 // Add new book (librarian only)
318 app.post('/api/books', authenticateToken, (req, res) => {
319     if (req.user.role !== 'librarian') {
320         console.log(`       UNAUTHORIZED BOOK ADD ATTEMPT: ${req.user.name} (${req.user.email})
                tried to add book`);
321         return res.status(403).json({ error: 'Only librarians can add books' });
322     }
323
324     const { title, author, isbn, genre, publication_year, quantity, description } = req.body;
325
326     if (!title || !author || !isbn || !genre || !publication_year || !quantity) {
327         return res.status(400).json({ error: 'All fields are required' });
328     }
329
330     db.get('SELECT id FROM books WHERE isbn = ?', [isbn], (err, existingBook) => {
331         if (err) {
332             console.error('Database error:', err);
333             return res.status(500).json({ error: 'Database error' });
334         }
```

```
335
336         if (existingBook) {
337             console.log(`       DUPLICATE BOOK ATTEMPT: ISBN ${isbn} already exists`);
338             return res.status(400).json({ error: 'A book with this ISBN already exists' });
339         }
340
341         db.run(
342             `INSERT INTO books (title, author, isbn, genre, publication_year, quantity, available,
                    description)
343             VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
344             [title, author, isbn, genre, publication_year, quantity, quantity, description || ''],
345             function(err) {
346                 if (err) {
347                     console.error('Error inserting book:', err);
348                     return res.status(500).json({ error: 'Failed to add book to database' });
349                 }
350
351                 //     ADDED: Log book addition
352                 console.log(`       NEW BOOK ADDED: "${title}" by ${author} | ISBN: ${isbn} | Genre:
                        ${genre} | Quantity: ${quantity} | Added by: ${req.user.name} | Time: ${new
                        Date().toLocaleString()}`);
353
354                 const newBook = {
355                     id: this.lastID,
356                     title,
357                     author,
358                     isbn,
359                     genre,
360                     publication_year,
361                     quantity,
362                     available: quantity,
363                     description: description || ''
364                 };
365
366                 res.status(201).json(newBook);
367             }
368         );
369     });
370 });
371
372 // Update book (librarian only)
373 app.put('/api/books/:bookId', authenticateToken, (req, res) => {
374     if (req.user.role !== 'librarian') {
375         console.log(`       UNAUTHORIZED BOOK UPDATE ATTEMPT: ${req.user.name} (${req.user.email})
                tried to update book ID ${req.params.bookId}`);
376         return res.status(403).json({ error: 'Only librarians can edit books' });
377     }
378
379     const { bookId } = req.params;
380     const { title, author, genre, publication_year, quantity, description } = req.body;
```

```
381
382     if (!title || !author || !genre || !publication_year || !quantity) {
383         return res.status(400).json({ error: 'All fields are required' });
384     }
385
386     // Calculate new available count based on current borrowings
387     db.get(
388         `SELECT
389             quantity as old_quantity,
390             (quantity - IFNULL((
391                 SELECT COUNT(*)
392                 FROM borrowings br
393                 WHERE br.book_id = books.id AND br.returned_date IS NULL
394             ), 0)) as current_available
395         FROM books WHERE id = ?`,
396         [bookId],
397         (err, book) => {
398             if (err) {
399                 console.error('Database error:', err);
400                 return res.status(500).json({ error: 'Database error' });
401             }
402
403             if (!book) {
404                 return res.status(404).json({ error: 'Book not found' });
405             }
406
407             // Calculate new available count
408             const borrowedCount = book.old_quantity - book.current_available;
409             const newAvailable = Math.max(0, quantity - borrowedCount);
410             db.run(
411                 `UPDATE books SET title = ?, author = ?, genre = ?, publication_year = ?,
412                  quantity = ?, available = ?, description = ? WHERE id = ?`,
413                 [title, author, genre, publication_year, quantity, newAvailable, description, bookId
                        ],
414                 function(err) {
415                     if (err) {
416                         console.error('Error updating book:', err);
417                         return res.status(500).json({ error: 'Error updating book' });
418                     }
419                     if (this.changes === 0) return res.status(404).json({ error: 'Book not found' })
                            ;
420
421                     //     ADDED: Log book update
422                     console.log(`        BOOK UPDATED: ID ${bookId} - "${title}" by ${author} |
                            Updated by: ${req.user.name} | Time: ${new Date().toLocaleString()}`);
423
424                     res.json({ message: 'Book updated successfully' });
425                 }
426             );
427         }
```

```
    );
});
// Get user's borrowed books
app.get('/api/users/:userId/borrowings', authenticateToken, (req, res) => {
    const { userId } = req.params;

    if (req.user.id != userId && req.user.role !== 'librarian') {
        return res.status(403).json({ error: 'Access denied' });
    }
    const query = `
        SELECT b.id as book_id, b.title as book_title, b.author,
            br.borrowed_date, br.due_date, br.returned_date, br.id as id
        FROM borrowings br
        JOIN books b ON br.book_id = b.id
        WHERE br.user_id = ?
        ORDER BY br.borrowed_date DESC
    `;
    db.all(query, [userId], (err, borrowings) => {
        if (err) {
            console.error('Error fetching borrowings:', err);
            return res.status(500).json({ error: 'Database error' });
        }
        res.json(borrowings);
    });
});
// Borrow a book
app.post('/api/books/:bookId/borrow', authenticateToken, (req, res) => {
    const { bookId } = req.params;
    const { days = 30 } = req.body;
    db.get(
        'SELECT COUNT(*) as count FROM borrowings WHERE user_id = ? AND returned_date IS NULL',
        [req.user.id],
        (err, row) => {
            if (err) return res.status(500).json({ error: 'Database error' });
            if (row.count >= 3) {
                console.log(`       BORROW LIMIT REACHED: ${req.user.name} (${req.user.email}) tried
                    to borrow book ID ${bookId}`);
                return res.status(400).json({ error: 'Borrow limit reached (3 books)' });
            }
            db.get(
                `SELECT b.*,
                        (b.quantity - IFNULL((
                        SELECT COUNT(*)
                        FROM borrowings br
                        WHERE br.book_id = b.id AND br.returned_date IS NULL
                    ), 0)) as available
                FROM books b WHERE b.id = ?`,
                [bookId],
                (err, book) => {
                    if (err) return res.status(500).json({ error: 'Database error' });
```

```
                    if (!book) return res.status(404).json({ error: 'Book not found' });
                    if (book.available <= 0) {
                        console.log(`       BOOK NOT AVAILABLE: ${req.user.name} tried to borrow "${
                            book.title}" but it's out of stock`);
                        return res.status(400).json({ error: 'Book not available' });
                    }
                    const borrowedDate = new Date().toISOString().split('T')[0];
                    const dueDate = new Date();
                    dueDate.setDate(dueDate.getDate() + parseInt(days));
                    const dueDateStr = dueDate.toISOString().split('T')[0];
                    db.run(
                        'INSERT INTO borrowings (book_id, user_id, borrowed_date, due_date) VALUES
                            (?, ?, ?, ?)',
                        [bookId, req.user.id, borrowedDate, dueDateStr],
                        function(err) {
                            if (err) {
                                console.error('Error borrowing book:', err);
                                return res.status(500).json({ error: 'Error borrowing book' });
                            }
                            //     ADDED: Log book borrowing
                            console.log(`       BOOK BORROWED: "${book.title}" by ${req.user.name} (
                                ${req.user.email}) | Due: ${dueDateStr} | Borrow ID: ${this.lastID}
                                | Time: ${new Date().toLocaleString()}`);

                            res.json({ message: 'Book borrowed successfully', borrowId: this.lastID
                                });
                        }
                    );
                }
            );
        }
    );
});
// Return a book
app.post('/api/books/:bookId/return', authenticateToken, (req, res) => {
    const { bookId } = req.params;
    const returnDate = new Date().toISOString().split('T')[0];
    // First get book details for logging
    db.get('SELECT title FROM books WHERE id = ?', [bookId], (err, book) => {
        if (err) {
            console.error('Database error:', err);
            return res.status(500).json({ error: 'Database error' });
        }
        db.run(
            'UPDATE borrowings SET returned_date = ? WHERE book_id = ? AND user_id = ? AND
                returned_date IS NULL',
            [returnDate, bookId, req.user.id],
            function(err) {
                if (err) {
                    console.error('Error returning book:', err);
```

```
521                     return res.status(500).json({ error: 'Error returning book' });
522                 }
523                 if (this.changes === 0) return res.status(404).json({ error: 'No active borrowing
                        found' });
524                 //     ADDED: Log book return
525                 const bookTitle = book ? book.title : `ID ${bookId}`;
526                 console.log(`       BOOK RETURNED: "${bookTitle}" by ${req.user.name} (${req.user.
                        email}) | Time: ${new Date().toLocaleString()}`);
527
528                 res.json({ message: 'Book returned successfully' });
529             }
530         );
531     });
532 });
533 // Delete a book
534 app.delete('/api/books/:bookId', authenticateToken, (req, res) => {
535     if (req.user.role !== 'librarian') {
536         console.log(`       UNAUTHORIZED BOOK DELETE ATTEMPT: ${req.user.name} (${req.user.email})
                tried to delete book ID ${req.params.bookId}`);
537         return res.status(403).json({ error: 'Only librarians can delete books' });
538     }
539     const { bookId } = req.params;
540     // First get book details for logging
541     db.get('SELECT title, author FROM books WHERE id = ?', [bookId], (err, book) => {
542         if (err) {
543             console.error('Database error:', err);
544             return res.status(500).json({ error: 'Database error' });
545         }
546         if (!book) {
547             return res.status(404).json({ error: 'Book not found' });
548         }
549         db.get(
550             'SELECT COUNT(*) as count FROM borrowings WHERE book_id = ? AND returned_date IS NULL',
551             [bookId],
552             (err, row) => {
553                 if (err) return res.status(500).json({ error: 'Database error' });
554                 if (row.count > 0) {
555                     console.log(`       BOOK DELETE BLOCKED: "${book.title}" has active borrowings`)
                            ;
556                     return res.status(400).json({ error: 'Cannot delete book with active borrowings'
                            });
557                 }
558                 db.run('DELETE FROM books WHERE id = ?', [bookId], function(err) {
559                     if (err) {
560                         console.error('Error deleting book:', err);
561                         return res.status(500).json({ error: 'Error deleting book' });
562                     }
563                     if (this.changes === 0) return res.status(404).json({ error: 'Book not found' })
                            ;
564                     //     ADDED: Log book deletion
```

```
                    console.log(`           BOOK DELETED: "${book.title}" by ${book.author} |
                        Deleted by: ${req.user.name} | Time: ${new Date().toLocaleString()}`);

                    res.json({ message: 'Book deleted successfully' });
                });
            }
        );
    });
});
// Get library statistics
app.get('/api/statistics', authenticateToken, (req, res) => {
    if (req.user.role !== 'librarian') {
        console.log(`       UNAUTHORIZED STATS ACCESS: ${req.user.name} (${req.user.email}) tried to
            access statistics`);
        return res.status(403).json({ error: 'Access denied' });
    }
    const queries = [
        'SELECT COUNT(*) as count FROM books',
        `SELECT SUM(b.quantity - IFNULL((
            SELECT COUNT(*)
            FROM borrowings br
            WHERE br.book_id = b.id AND br.returned_date IS NULL
        ), 0)) as available FROM books b`,
        'SELECT COUNT(*) as count FROM borrowings WHERE returned_date IS NULL',
        'SELECT COUNT(*) as count FROM users',
        'SELECT COUNT(*) as count FROM borrowings WHERE due_date < date("now") AND returned_date IS
            NULL',
        'SELECT COUNT(*) as count FROM borrowings'
    ];
    const stats = {};
    const statNames = ['totalBooks', 'availableBooks', 'borrowedBooks', 'totalUsers', 'overdueBooks'
        , 'totalBorrowings'];
    let completed = 0;
    queries.forEach((query, index) => {
        db.get(query, (err, row) => {
            if (err) {
                console.error('Error getting stats:', err);
                return res.status(500).json({ error: 'Database error' });
            }
            stats[statNames[index]] = row.count || row.available || 0;
            completed++;
            if (completed === queries.length) {
                //     ADDED: Log statistics access
                console.log(`       STATISTICS ACCESSED: ${req.user.name} viewed library stats |
                    Time: ${new Date().toLocaleString()}`);
                res.json(stats);
            }
        });
    });
});
```

```
610 // Get all borrowings (for librarian)
611 app.get('/api/borrowings', authenticateToken, (req, res) => {
612     if (req.user.role !== 'librarian') {
613         console.log(`       UNAUTHORIZED BORROWINGS ACCESS: ${req.user.name} (${req.user.email})
                tried to access all borrowings`);
614         return res.status(403).json({ error: 'Access denied' });
615     }
616     const query = `
617         SELECT br.*, b.title as book_title, b.author, u.name as user_name
618         FROM borrowings br
619         JOIN books b ON br.book_id = b.id
620         JOIN users u ON br.user_id = u.id
621         ORDER BY br.borrowed_date DESC
622     `;
623     db.all(query, (err, borrowings) => {
624         if (err) {
625             console.error('Error fetching all borrowings:', err);
626             return res.status(500).json({ error: 'Database error' });
627         }
628         //     ADDED: Log borrowings access
629         console.log(`       ALL BORROWINGS ACCESSED: ${req.user.name} viewed ${borrowings.length}
                borrowing records | Time: ${new Date().toLocaleString()}`);
630         res.json(borrowings);
631     });
632 });
633 // Serve the main page for all other routes
634 app.get('*', (req, res) => {
635     res.sendFile(path.join(__dirname, 'index.html'));
636 });
637 // Only start the server if this file is run directly (not when required by tests)
638 if (require.main === module) {
639     app.listen(PORT, () => {
640         console.log(`       Server running on http://localhost:${PORT}`);
641         console.log(`       Logging enabled for user actions, book operations, and system events`);
642         console.log(`    Server started at: ${new Date().toLocaleString()}`);
643     });
644 }
645
646 // Export the app for testing
647 module.exports = app;
```

Chapter 9

SCREENSHOTS

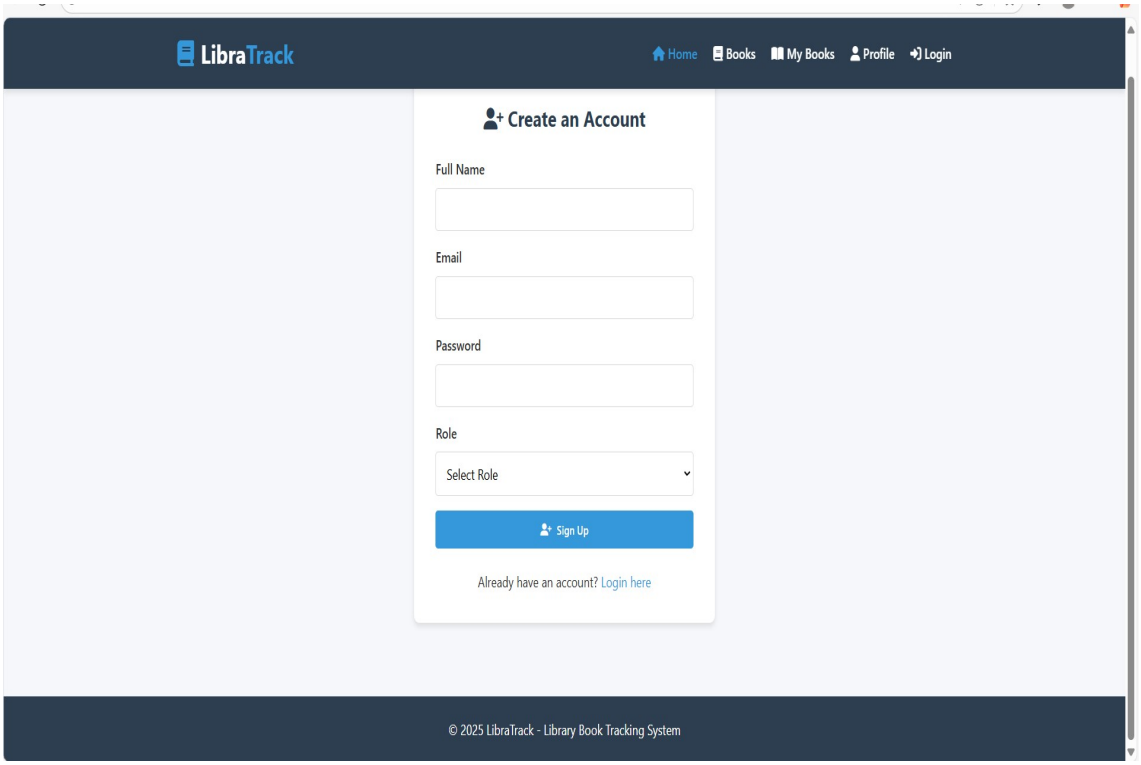

Figure 9.1: Signup Page

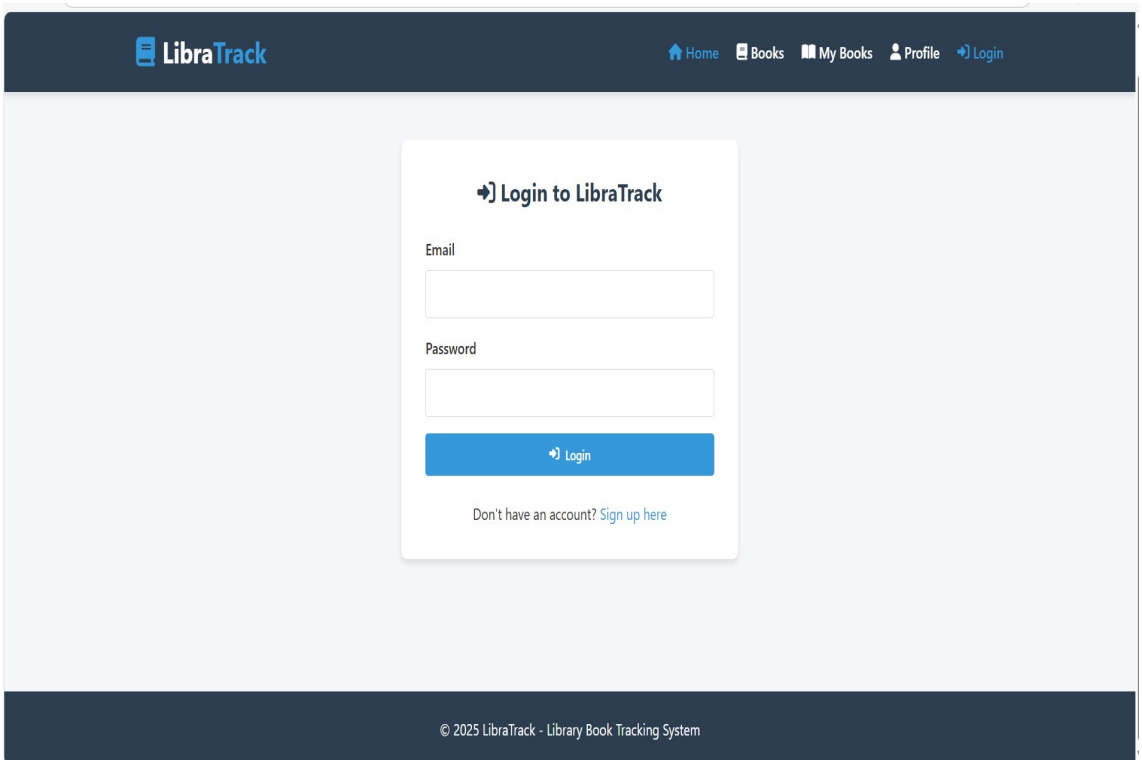

Figure 9.2: Login Page

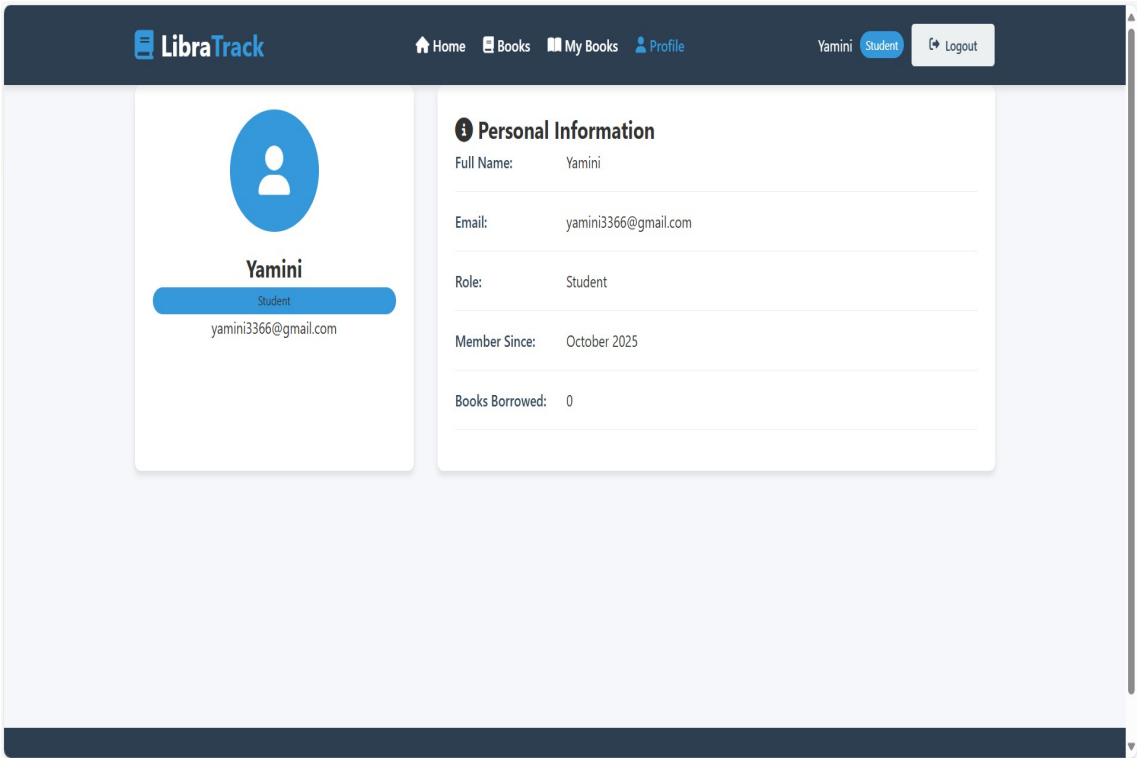

Figure 9.3: Profile

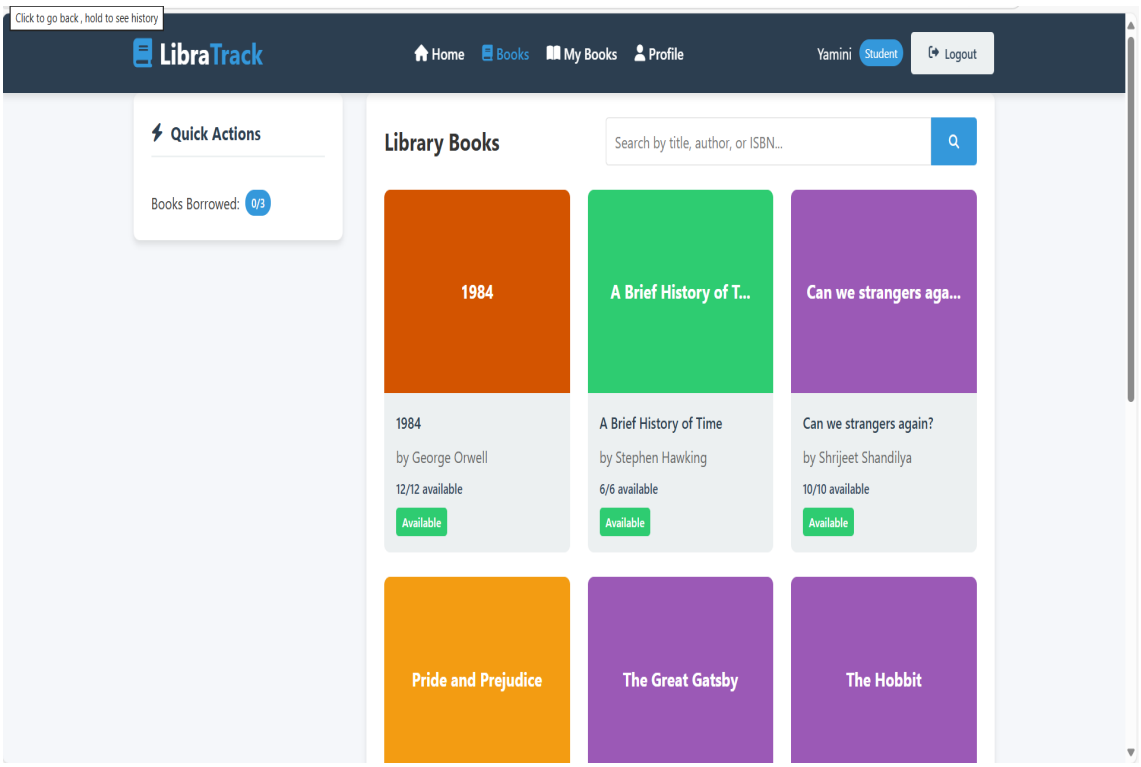

Figure 9.4: Student View Of Books

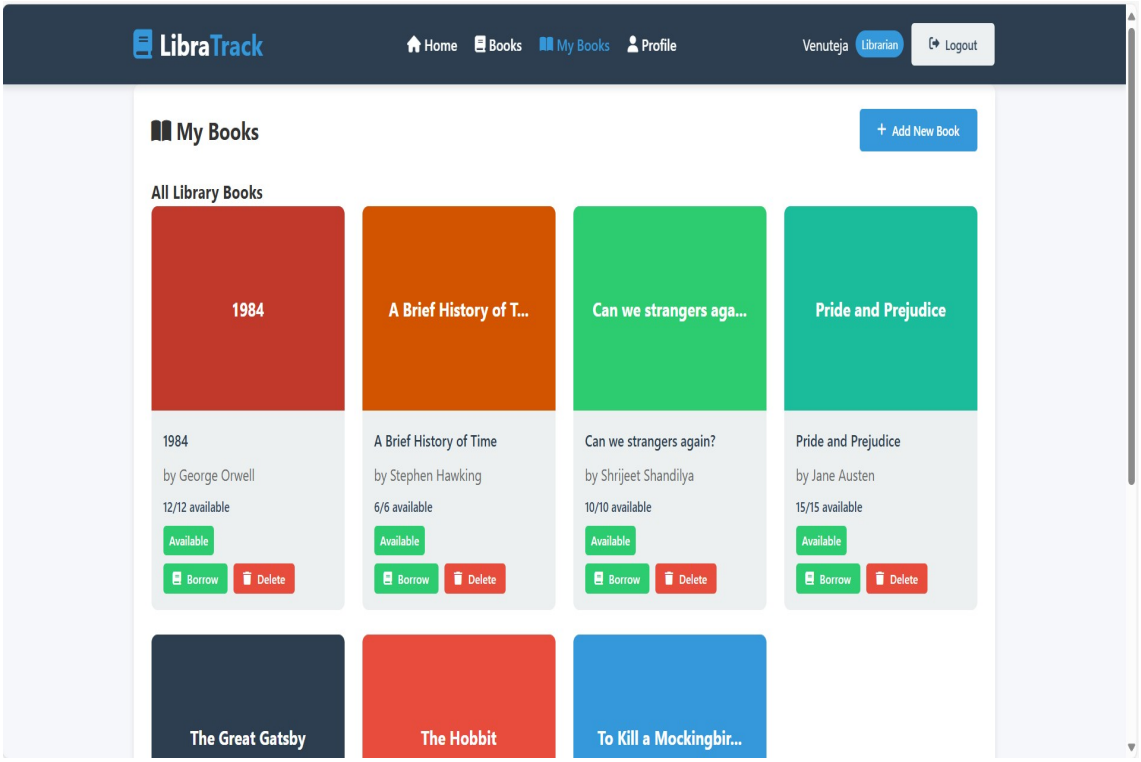

Figure 9.5: Librarian View Of Books

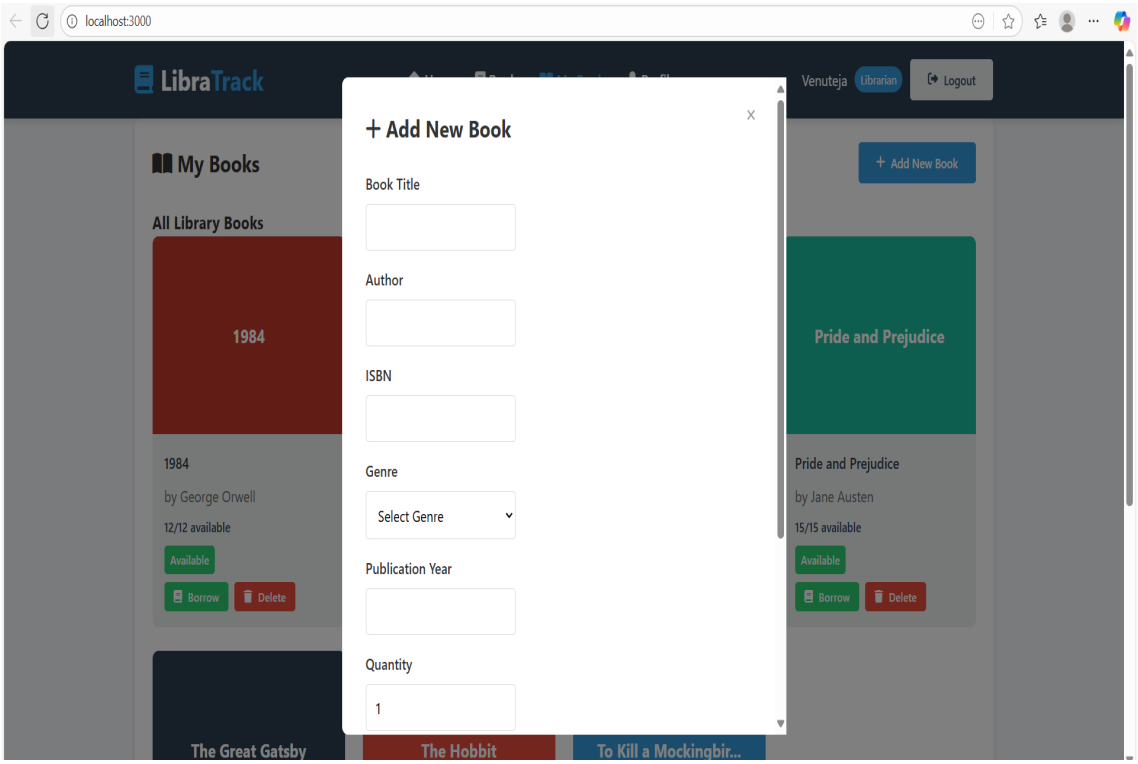

Figure 9.6: Adding Books By Librarian

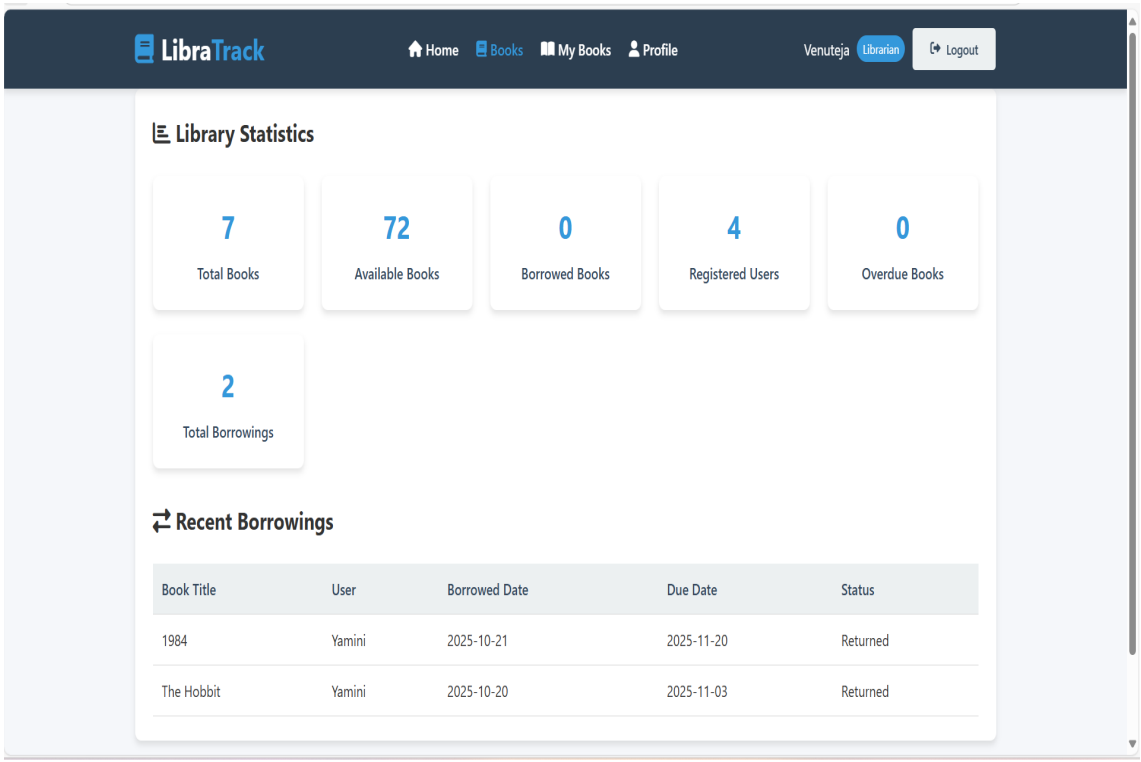

Figure 9.7: Library Statistics

REFERENCES

[1] M. Casciaro, “Node.js Design Patterns,” Packt Publishing, Dec. 29, 2014.

[2] “Express.js – Node.js web application framework,” Express.js, https://expressjs.com/.

[3] “SQLite – Lightweight relational database management system,” SQLite, https://www.sqlite.org/.

[4] J. Duckett, “HTML and CSS: Design and Build Websites,” Wiley, 2011.

[5] B. McFarland, “CSS3: The Missing Manual,” O’Reilly Media, 2013.