Library Management System

Project Report

O.Hashini Sandamini

8/11/2025

Assignment: Software Engineering Internship – Library Management System
Technology Stack: .NET 6 Backend + React TypeScript Frontend
Repository: https://github.com/SandaminiObadage/Library_MS.git

Table Of Contents

1. Executive Summary

This project is a **web application** that allows users to manage their personal book collections. It meets all the required features for the assignment and also adds **user registration and login** for better security.

Main Features:

- Add, view, update, and delete books.
- User authentication with **JWT tokens**.
- Secure password storage with **BCrypt**.
- Responsive, easy-to-use design.

2. Project Overview

Purpose:
To give each user their own private digital library where they can keep track of books securely.

2.1 Scope Delivered:

- **Required:** CRUD (Create, Read, Update, Delete) for books.
- **Optional:** Authentication system with login and registration.
- **Extra:** Responsive design, About page.

2.2 Technology Stack:

- **Backend:** ASP.NET Core 6, Entity Framework Core, SQLite, JWT, BCrypt.
- **Frontend:** React 18, TypeScript, Bootstrap 5, Axios, Context API.

3. System Architecture

3.1 High-Level Architecture

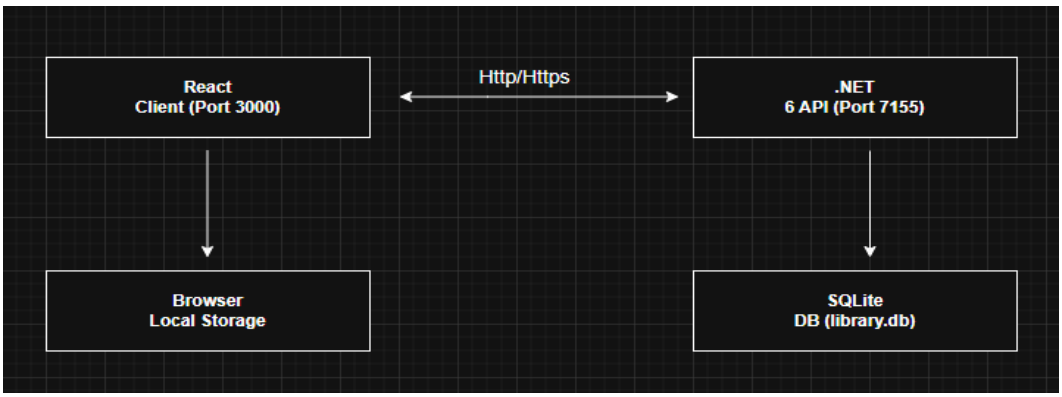

Figure 3.1

3.2 Application Layers

Presentation Layer (React Frontend):

- Components: UI elements.
- Services: API communication.
- Contexts: Authentication state.
- Utils: Helper functions.

Business Logic Layer (.NET Backend):

- Controllers: Handle API requests.
- Services: Contain core logic.
- DTOs: Transfer data between layers.
- Middleware: Handle cross-cutting concerns.

Data Access Layer:

- Entity Framework Core ORM.
- DbContext for database access.
- Models and migrations.

Data Storage Layer:

- SQLite for relational storage.
- Database file stored locally.

3.3 Communication Flow

1. User interacts with a React component.
2. Component calls a service (authService/bookService).
3. Service sends an HTTP request via Axios to the .NET API.
4. API Controller processes the request and calls business logic.
5. Business logic accesses data through Entity Framework.
6. Database responds, and API sends JSON back to the frontend.
7. Frontend updates the UI.

4. Backend Implementation

Structure:

Controllers → Services → Data → Database

Main Controllers:

- **AuthController:** Handles register and login.
- **BooksController:** Handles book operations (only for logged-in users).

Database Models:

- **User:** Stores username, email, password (hashed), and their books.
- **Book:** Stores book details and links to the user who owns it.

4.1 API Endpoints:

Method	Endpoint	Description	Auth Required
POST	/api/Auth/register	Register user	No
POST	/api/Auth/login	Login user	No
GET	/api/Books	Get all books	Yes
POST	/api/Books	Add a book	Yes
PUT	/api/Books/{id}	Update a book	Yes
DELETE	/api/Books/{id}	Delete a book	Yes

Table 4.1

5. Frontend Implementation

Main Components:

- **Header:** Navigation bar with login/logout.
- **HeroSection:** Landing page.
- **About:** Information about the app.
- **BookList:** Shows all books for the user.
- **BookForm:** Form to add or edit books.
- **LoginModal / RegisterModal:** For user authentication.
- **Footer:** Page footer.

State Management:

- Used **React Context API** to store login status and user details.
- Stored JWT token in local Storage for session persistence.

6. Database Design

Entity Relationship (ER) Diagram:

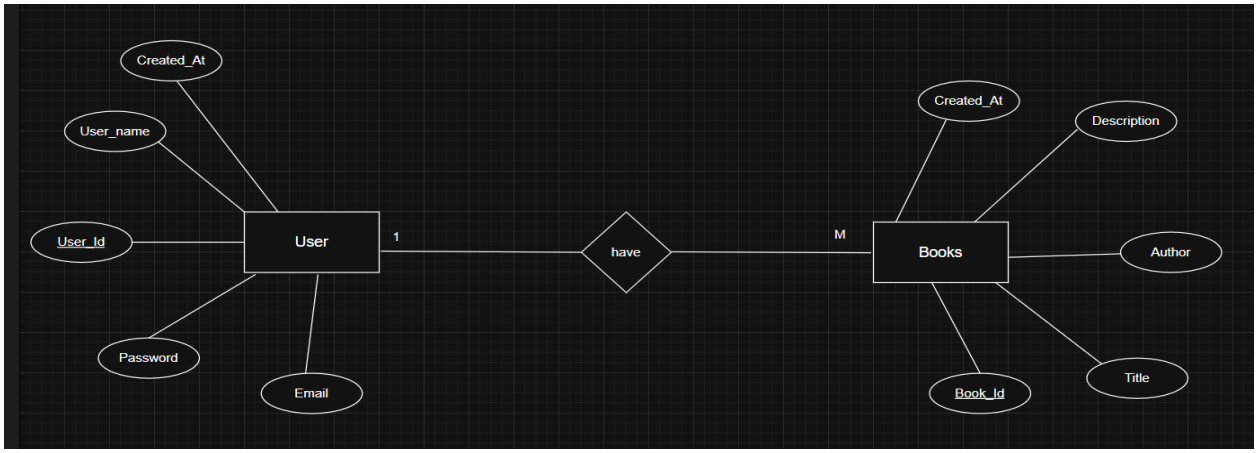

Figure 6.1

Description:

- **User** table stores account details.
- **Book** table stores book information and links to the User.
- Relationship: One User can have many Books.

User	Books
User_Id **(PK)**	Book_id **(PK)**
User_name	Title
Password	Author
Email	Description
Created_At	Created At
	User_Id **(FK)**

7. System Implementation

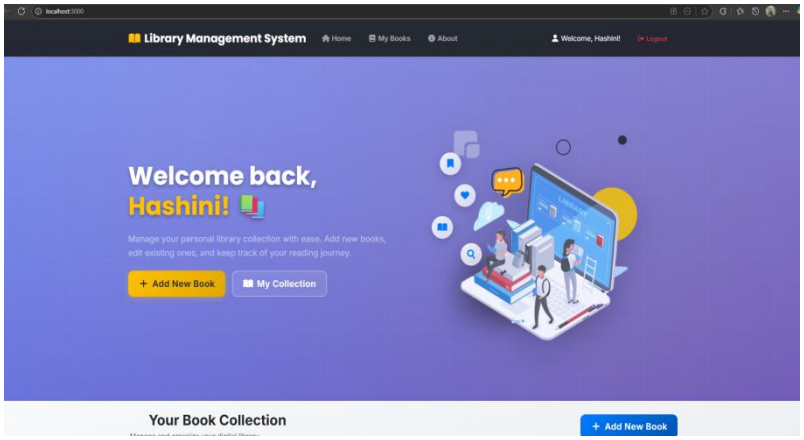

Figure 7.1

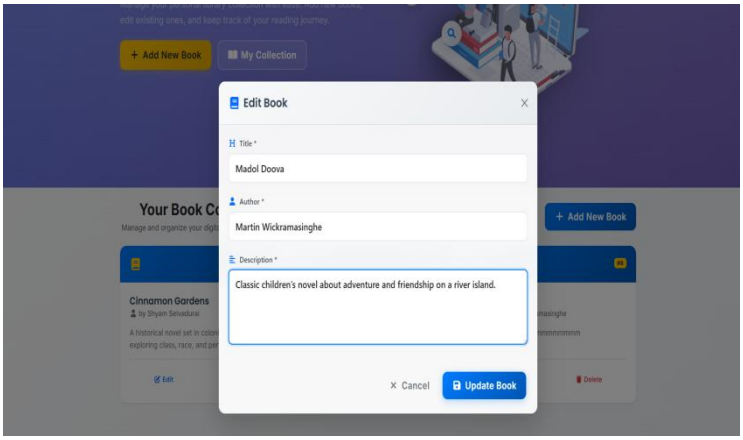

Figure 7.2

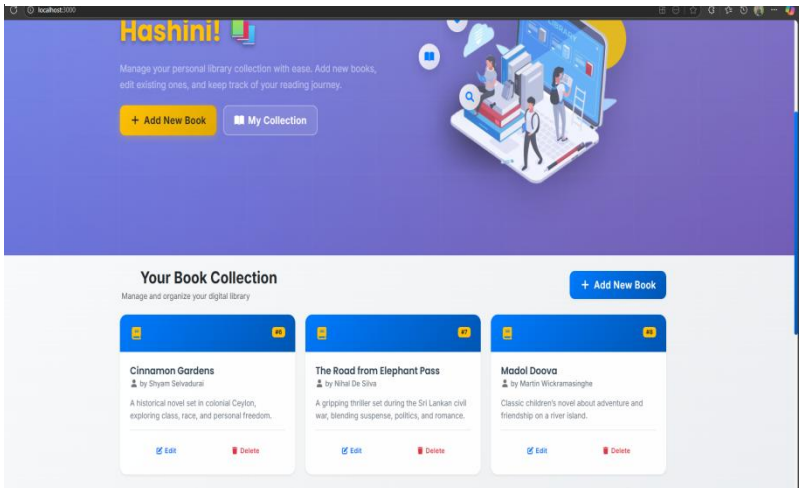

Figure 7.3

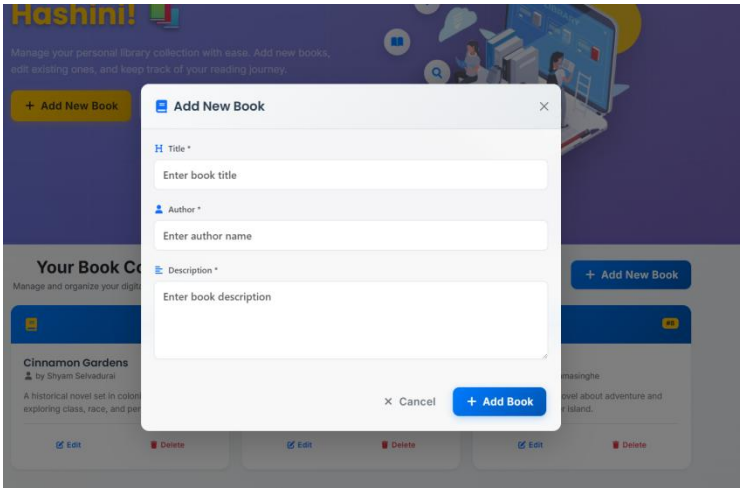

Figure 7.4

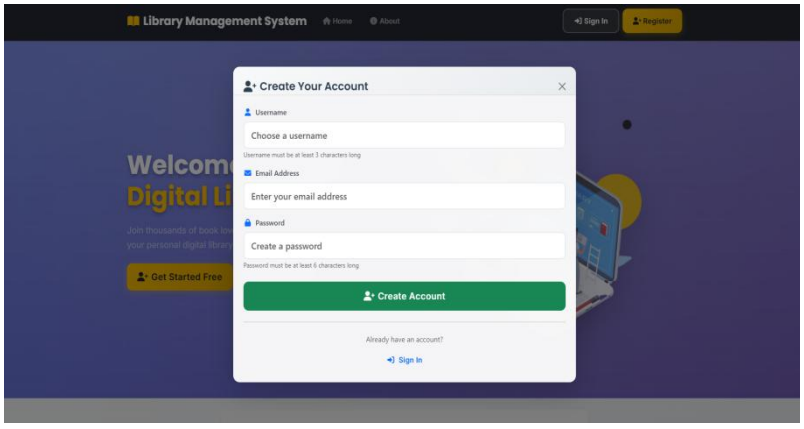

Figure 7.5

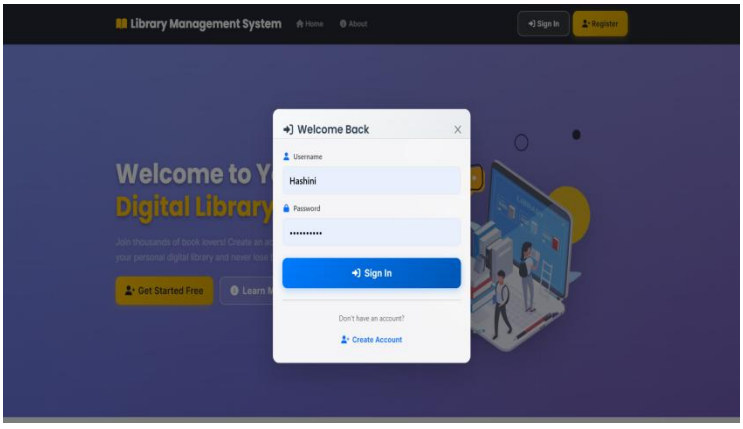

Figure 7.6

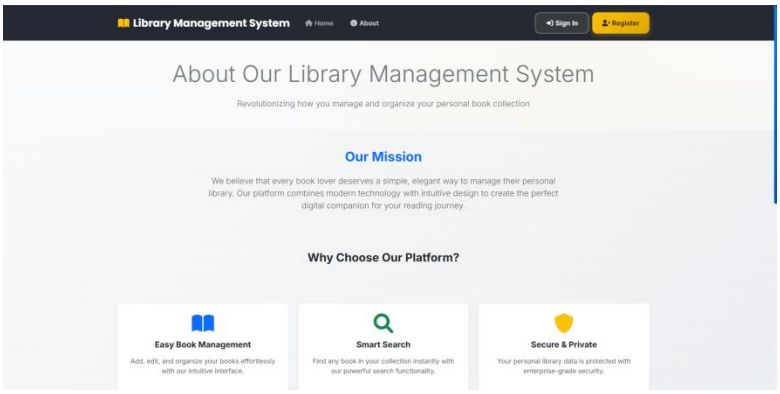

Figure 7.7

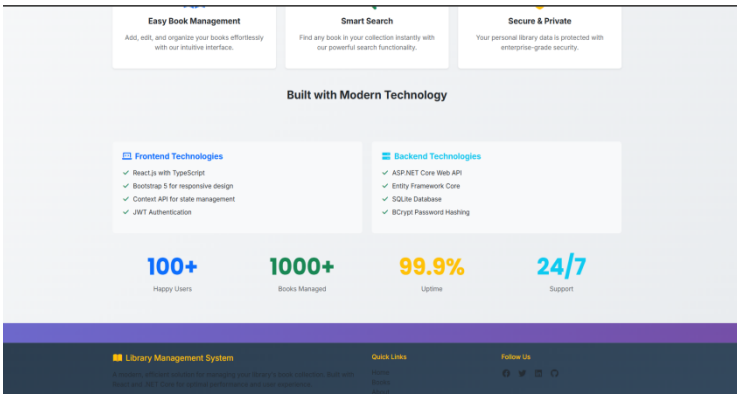

Figure 7.8

8. Authentication & Security

- **Passwords:** Stored with BCrypt hashing (no plain text).
- **Authentication:** JWT tokens with expiry time.
- **Authorization:** Only logged-in users can manage books.
- **CORS:** Configured to allow only trusted frontend URLs.

9. Additional Features

- Responsive design with Bootstrap.
- Confirmation pop-ups before deleting a book.
- About page and professional footer.
- Loading indicators for better user experience.

10. Challenges Faced

- **JWT Setup:** Solved by adding backend middleware and Axios interceptors in frontend.
- **State Management:** Fixed by using React Context API.
- **CORS Errors:** Solved by proper backend configuration.
- **Database Relationships:** Configured one-to-many in Entity Framework.

11. Key Learnings

- How to connect a .NET backend to a React frontend.
- How to design and use REST APIs.
- How to make a secure authentication system.
- How to keep code clean and maintainable.

12. Testing & Deployment

Testing:

- Checked registration, login, and logout.
- Tested all CRUD book operations.
- Verified that unauthorized access is blocked.

Deployment Prep:

- Separate settings for development and production.
- Used EF Core migrations for database changes.
- Optimized frontend build for faster load times.

13. Future Improvements

- Add book categories and search feature.
- Add dark mode.
- Make a mobile app version.
- Add API documentation with Swagger.

14. Conclusion

The Library Management System makes it easy to add, view, update, and delete books in a simple and secure way. It uses modern tools like ASP.NET Core, SQLite, and React to give a smooth user experience. The system is reliable, easy to use, and can be improved in the future with more features like advanced search or mobile support.

Repository: https://github.com/SandaminiObadage/Library_MS.git
Contact: hashiniobadage6030@gmail.com