Shelf Wise – Library Management System

Internship Assignment

Submitted By: S.M.M. Shafran

Date of Submission: 10/08/2025

Table of Contents

Introduction

This project was developed as part of the **Software Engineering Internship assignment** with the objective of demonstrating the ability to design, implement, and document a complete software solution. The assignment required the creation of a simple **Library Management System** that enables users to efficiently manage book records through an intuitive and responsive interface. The system's core functionality is centered on the four essential CRUD operations — **Create**, **Read**, **Update**, and **Delete** — allowing users to add new books, view existing records, make modifications, and remove entries as necessary.

The backend was developed using **C# .NET** in combination with **SQLite** and **Entity Framework**, ensuring robust data management, scalability for future improvements, and simplified database operations. The frontend was implemented using **React** with **TypeScript**, providing a modern, maintainable, and user-friendly interface that communicates seamlessly with the backend via RESTful API endpoints.

This project not only fulfills the functional requirements outlined in the assignment but also serves as an opportunity to apply best practices in software development, such as proper error handling, input validation, and clean code structure. While the primary focus was on CRUD operations, optional features such as user authentication were considered to enhance usability and security

.

Backend Implementation

Technologies used

Language & Framework	C# .NET
Database	SQLite
ORM	Entity Framework Core
Architecture	RESTful API
Tools	Visual Studio / Visual Studio Code, Postman for API testing

Database Design

Table: Book

Id	Primary key, int
Title	String
Author	String
Book Description	String
Author Description	String
Book Cover Image	Byte

Table: User

Id	Primary Key, int
First Name	String
Last Name	String
Email	String
Password	String
Age	Int

API Endpoints

API Endpoints for Book

POST /api/book	Create a new book record
GET /api/book	Retrieve all books
GET /api/book/{id}	Retrieve a specific book by ID
PUT /api/books/{id}	Update an existing book
DELETE /api/books/{id}	Delete a book

API Endpoint for user

POST /api/user/register	Register new user
POST /api/user/login	Login existing user

User Authentication

- Implemented **user registration** and **login** functionality using C# .NET.
- Passwords securely hashed before storage.
- JWT (JSON Web Token) used for authentication:
    - Token generated on successful login.
    - Token stored on the client side for session persistence.

Frontend Implementation

Technologies Used

Language & Framework	React with TypeScript
UI Libraries	Tailwind CSS
Routing	React Router
HTTP Client	Axios

Pages and components

Pages

- Home page: Displays all books with options to view.
- Add Book Page: Form to create new book
- Book Details Page: View Complete book details with update and delete options.

Components

- Card component: Displays individual book details with Book cover image, Book title, and Author. A button with the option for navigate to book details page.
- Button Component: A reusable clickable element for triggering actions consistently across the app.
- Navbar Component: Provides navigation links for moving between different sections of the application.

CRUD Integration

- Axios handles API requests to the backend.
- Form validation prevents empty or invalid inputs.

Integration Between Backend & Frontend

- Backend API runs on **http://localhost:[5173]**.
- Frontend communicates with backend via Axios requests.
- CORS policy configured in backend to allow frontend access.

Challenges Faced and Solutions

CORS Errors:
Initially, I encountered Cross-Origin Resource Sharing (CORS) errors when trying to connect the frontend with the backend API. This was resolved by properly configuring CORS policies in the backend to allow requests from the frontend's origin.

Database Migrations:
Managing database schema changes with SQLite was challenging at first. I learned to effectively use Entity Framework migrations to keep the database schema up-to-date without losing data, which streamlined the development process.

Form Validation:
Ensuring that user input was valid and secure required implementing validation on both the frontend and backend. This dual-layer validation helped maintain data consistency and improved overall application reliability.

Using TypeScript:
For the first time, I worked extensively with TypeScript, which presented a learning curve. Understanding static typing, interfaces, and type safety was challenging but ultimately improved code quality and reduced runtime errors. Adopting TypeScript helped me write more maintainable and robust code.

Additional Features Implemented

- Implemented user authentication (login, register).
- Improved UI styling for better user experience using Tailwind CSS.

Insights and Learning

- Gained hands-on experience in integrating React with a .NET backend.
- Learned database management using SQLite and Entity Framework.
- Improved understanding of RESTful API design and HTTP methods.
- Enhanced skills in error handling and data validation.

Application Video Demonstration

Google Drive Link for the video:https://drive.google.com/file/d/1lenI9MQhQi0s0eaW5Fm2-NmnSHUEDLGd/view?usp=drive_link

Instructions to Run the Project Locally

Clone the Repository

git clone https://github.com/ShafranSheikh/ShelfWise---Library-Management-System.git

Backend Setup (Visual Studio – Package Manager Console)

1. Open the **backend** folder in **Visual Studio**.
2. Open **Package Manager Console** (Tools → NuGet Package Manager → Package Manager Console).
3. Ensure the **Default Project** in the Package Manager Console is set to your backend project.
4. Update the database:
5. Update-Database
6. Run the backend API by pressing **F5** or **Ctrl + F5** in Visual Studio.
7. The API will be available at:
    - http://localhost:5008

Frontend Setup (Visual Studio Code)

1. Open the frontend folder in Visual Studio Code.
2. Install dependencies:
    — npm install
3. Start the development server:
    — npm run dev
4. The frontend will be available at:
    — http://localhost:5173

Conclusion

This Library Management System project successfully demonstrates the ability to develop a complete software application using modern technologies. The combination of C# .NET for backend, SQLite for data storage, and React with TypeScript for frontend allowed for a robust, efficient, and user-friendly solution. This project provided valuable experience in full-stack development, problem-solving, and integrating different technologies into a cohesive application.