Library Management System – Project Report

Candidate Name: Yashith Chandeepa
Email: yashithc.dev@gmail.com
Date: February 16, 2026

---

1. Project Overview

This project presents the development of a Library Management System built as part of the Software Engineering Internship Assignment for Expernetic LLC. The primary objective of this assignment was to design and implement a full-stack web application that allows users to perform CRUD operations on book records using a C# .NET backend and a React + TypeScript frontend.

The system enables users to create, view, update, and delete books with a clean, modern interface. The backend exposes RESTful API endpoints, while the frontend consumes these APIs to provide a seamless and responsive user experience.

The project was developed independently and adheres strictly to the scope defined in the assignment brief.

---

2. Backend Implementation (C# .NET)

Technology Stack

- Framework: ASP.NET Core 8 Web API
- Database: SQLite
- ORM: Entity Framework Core
- API Documentation: Swagger (OpenAPI)

Architecture Overview

The backend follows a clean and structured architecture:

- Model layer for defining data structures
- DbContext for database interaction
- Controller layer for handling HTTP requests
- DTO usage for request validation and safe data transfer

Key Features

RESTful Endpoints

The following RESTful endpoints were implemented:

- GET /api/books
  Retrieves all books from the database.
- GET /api/books/{id}
  Retrieves a specific book by ID.
- POST /api/books
  Creates a new book record.
- PUT /api/books/{id}
  Updates an existing book record.
- DELETE /api/books/{id}
  Deletes a book record.

Each endpoint returns appropriate HTTP status codes such as 200 OK, 201 Created, 400 Bad Request, and 404 Not Found.

Data Modeling

The core entity is the Book model with the following attributes:

- Id (Primary Key)
- Title (Required)
- Author (Required)
- Description (Optional)
- CreatedAt (Timestamp for record tracking)

Entity Framework Core is used to map the Book model to a SQLite database table. Migrations were used to generate and update the database schema.

Error Handling

Validation is implemented both through model attributes and controller-level checks. Required fields are validated before data is saved to the database.

Global exception handling ensures that unhandled errors return structured responses instead of exposing internal stack traces. Meaningful error messages are returned to the frontend for user feedback.

API Documentation

Swagger was integrated to provide interactive API documentation. This allows developers to test endpoints directly in the browser and understand request and response formats clearly.

---

3. Frontend Implementation (React & TypeScript)

Technology Stack

- Framework: React 18
- Language: TypeScript
- Styling: Tailwind CSS (Dark Navy Theme)
- Routing: React Router
- State Management: React Hooks

UI Architecture

The frontend is organized using a modular component structure:

- Pages folder for route-based views
- Components folder for reusable UI components
- API service layer for backend communication
- Type definitions for strong typing

Key UI Components

BookList

The BookList component:

- Fetches books from the backend API
- Displays books in a responsive card/table layout
- Handles delete operations
- Displays loading and error states
- Includes an empty state UI when no books exist

BookForm

The BookForm component:

- Used for both adding and editing books
- Detects mode based on route parameters
- Includes controlled form inputs

- Performs basic validation
- Displays backend error messages
- Redirects on successful submission

Navigation

The application includes a modern dark navy header with navigation links for:

- Home
- Add Book

The layout is responsive and optimized for desktop and tablet views.

---

4. Design Decisions

Use of Tailwind CSS

Tailwind CSS was selected for styling because:

- It enables rapid UI development.
- It ensures consistent spacing and layout.
- It simplifies responsive design.
- It avoids large custom CSS files.
- It supports modern design patterns efficiently.

The application uses a dark navy theme with blue accent colors to provide a professional and modern appearance.

Use of TypeScript

TypeScript was chosen to:

- Ensure type safety.
- Prevent runtime errors.
- Improve maintainability.
- Enhance code readability.
- Provide better developer tooling and IntelliSense.

RESTful Design Principles

The API strictly follows REST conventions. Clear route naming, proper HTTP verbs, and status codes were used to maintain standard API behavior.

---

5. Challenges & Solutions

Challenge 1: Frontend and Backend Integration

Issue:
Initially, the frontend could not communicate with the backend due to CORS restrictions.

Solution:
CORS configuration was added in the ASP.NET Core application to allow requests from the frontend development server. Additionally, API base URLs were aligned correctly.

Challenge 2: Form Validation and State Management

Issue:
Managing form state for both Add and Edit modes while keeping the component reusable required careful state handling.

Solution:
React Hooks were used to manage form state dynamically. The same BookForm component handles both modes by checking route parameters. Validation is enforced both in the frontend and backend to ensure data integrity.

Challenge 3: Error Handling

Issue:
Ensuring consistent error handling across both layers required proper coordination.

Solution:
Standardized error responses were implemented in the backend. The frontend captures these errors and displays user-friendly messages in a styled error banner.

---

6. Key Insights & Reflections

This project strengthened my understanding of full-stack development using .NET and React.

Key learnings include:

- Proper use of Entity Framework Core for data persistence.
- Importance of RESTful API design.
- Effective integration between frontend and backend layers.
- Managing form state in React using controlled components.
- Applying clean code principles in both frontend and backend.
- Structuring projects for maintainability and clarity.

Developing this system independently reinforced problem-solving skills, debugging practices, and structured development workflows.

---

7. How to Run the Application

Backend:

1. Navigate to the backend folder.
2. Run database migrations:
   dotnet ef database update
3. Start the API:
   dotnet run
4. Access Swagger at:
   https://localhost:xxxx/swagger

Frontend:

1. Navigate to the frontend folder.
2. Install dependencies:
   npm install
3. Start development server:
   npm run dev
4. Access the application at:
   http://localhost:5173

Ensure the backend server is running before using the frontend.

Conclusion

The Library Management System successfully fulfills the requirements of the Software Engineering Internship Assignment. It demonstrates the implementation of a clean, well-structured full-stack application using ASP.NET Core, SQLite, React, TypeScript, and Tailwind CSS.

The project reflects strong understanding of RESTful API development, database integration, frontend component architecture, validation strategies, and modern UI design practices.