Library Management System

Project Type	Full Stack Web Application
Backend	C# .NET with SQLite Database
Frontend	React with TypeScript
Database	SQLite with Entity Framework
Development Duration	1 Week
Date	2025.12.19

1. Project Overview

1.1 Project Objectives

- Develop a RESTful API backend using C# .NET framework.
- Implement SQLite database with Entity Framework ORM.
- Create a responsive frontend using React and TypeScript.
- Implement complete CRUD operations for book management.
- Ensure proper error handling and data validation.
- Follow software development best practices and design patterns.
- Provide comprehensive code documentation.

1.2 Core Features

- Create Book Records: - Add new books with title, author, and description.
- View Book Records: - Display all books in an organized, card-based layout.
- Update Book Records: - Edit existing book information with validation. Delete Book Records: - Remove books with a confirmation dialog.
- Real-time Updates: - Immediate UI updates after operations.
- Error Handling: - Comprehensive error messages and user feedback.
- Responsive Design: - Mobile-friendly interface with modern styling.

2. Backend Implementation

2.1 Architecture Overview

The backend follows a layered architecture pattern with clear separation of concerns.

- **Controllers Layer**: - Handles HTTP requests and responses. (BooksController)
- **Data Layer**: - Database context and entity configurations. (LibraryDbContext)
- **Models Layer**: - Entity definitions and business objects. (Book model)
- **DTOs Layer**: - Data Transfer Objects for API communication.

2.2 Database Model

The Book entity includes the following properties.

Property	Type	Description
Id	Int	Primary Key
Title	String	Book Title
Author	String	Book Author
Description	String	Book Description
CreatedAt	DateTime	Creation timestamp
UpdatedAt	DateTime	Last update timestamp

2.3 RESTful API Endpoints

Method	End Point	Description
Get	/api/books	Retrieve all books
Get	/api/books/{id}	Retrieve book by ID
Post	/api/books	Create new book
Put	/api/books/{id}	Update book
Delete	/api/books/{id}	Delete book

2.4 Key Implementation Features

- **Data Validation**: - Comprehensive input validation using Data Annotations.
- **Error Handling**: - Try-catch blocks with appropriate HTTP status codes.
- **CORS Configuration**: - Properly configured to allow frontend communication.
- **Async Operations**: - All database operations use async/await for better performance.

- **DTO Pattern**: - Separate DTOs for Create, Update, and Response operations.
- **Entity Framework**: - Code-first approach with automatic migrations.

3. Frontend Implementation

3.1 Component Architecture

The frontend follows a component-based architecture with React best practices.

- **App Component**: Main container managing application state and API calls.
- **BookList Component**: Displays books in a responsive card grid layout.
- **BookForm Component**: Reusable form for creating and editing books.
- **Book Service**: Centralized API communication layer using Axios.
- **Type Definitions**: TypeScript interfaces ensuring type safety.

3.2 State Management

The application uses React hooks for efficient state management.

- **useState**: Managing books array, loading states, form visibility, and messages.
- **useEffect**: Fetching data on component mount.
- **Local State**: Form inputs and validation errors in BookForm component.

3.3 User Interface Design

The UI emphasises usability and modern design principles:

- **Responsive Layout**: Mobile-first design that adapts to all screen sizes.
- **Card-Based Design**: Books displayed in clean, organized cards with hover effects.
- **Empty States**: Friendly messages when no books exist.
- **Form Validation**: Real-time validation with clear error messages.
- **User Feedback**: Success and error alerts with auto-dismiss.
- **Confirmation Dialogs**: Delete confirmation to prevent accidental data loss.
- **Icons**: React Icons library for consistent, professional icons.

4. Challenges Faced and Solutions

4.1 CORS Configuration

Challenge: Frontend could not connect to the backend due to CORS restrictions.
Solution: Configured CORS in Program.cs to allow requests from React development servers (localhost:3000 and localhost:5173).

4.2 Asynchronous Operations

Challenge: Handling API calls and keeping the UI in sync with database changes.
Solution: Used async/await for API calls and added loading indicators and error handling for better user feedback.

4.3 State Management

Challenge: Managing multiple states such as book data, loading status, and form modes.
Solution: Centralized state in the main component and passed data using props and callback functions.

4.4 Form Validation

Challenge: Maintaining consistent validation on both frontend and backend.
Solution: Applied Data Annotations on the backend and matching validation logic on the frontend.

5. Key Insights and Learning Outcomes

5.1 Technical Skills Developed

- **Backend Development:** Learned C# .NET, Entity Framework, and how to build RESTful APIs with proper error handling.
- **Frontend Development:** Improved React skills using TypeScript, hooks (useState, useEffect), and component-based design.
- **Database Management:** Gained experience with SQLite, Entity Framework Code-First approach, and data modeling.
- **API Design:** Understood REST principles, HTTP methods, status codes, and API structure.
- **Type Safety:** Learned how TypeScript helps prevent errors early and improves code maintainability.

6.Future Enhancement Possibilities

- User authentication and authorization system
- Search and filter functionality for books
- Sorting options (by title, author, date added)
- Pagination for large book collections

7. Deployment and Setup Instructions

7.1 Requirements

• .NET 6.0 SDK or later

• Node.js 16.x or later

• npm or yarn package manager

• Git for version control

• Visual Studio Code

7.2 Backend Setup

1. Clone the repository from Git

2. Navigate to the backend directory

3. Run 'dotnet restore' to install dependencies

4. Run 'dotnet build' to compile the project

5. Run 'dotnet run' to start the backend server

6. Backend will be available at http://localhost:5238

7.3 Frontend Setup

1. Navigate to the frontend directory

2. Run 'npm install' to install dependencies

3. Run 'npm run dev' to start the development server

4. Frontend will be available at http://localhost:5173

5. Access the application in a web browser