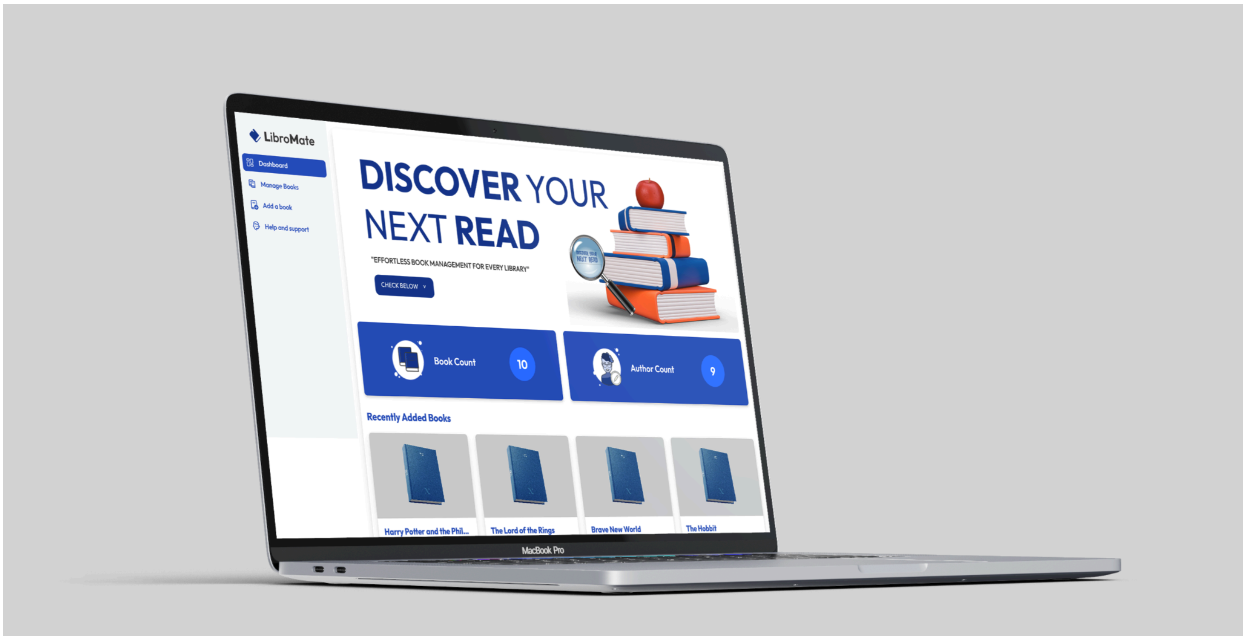
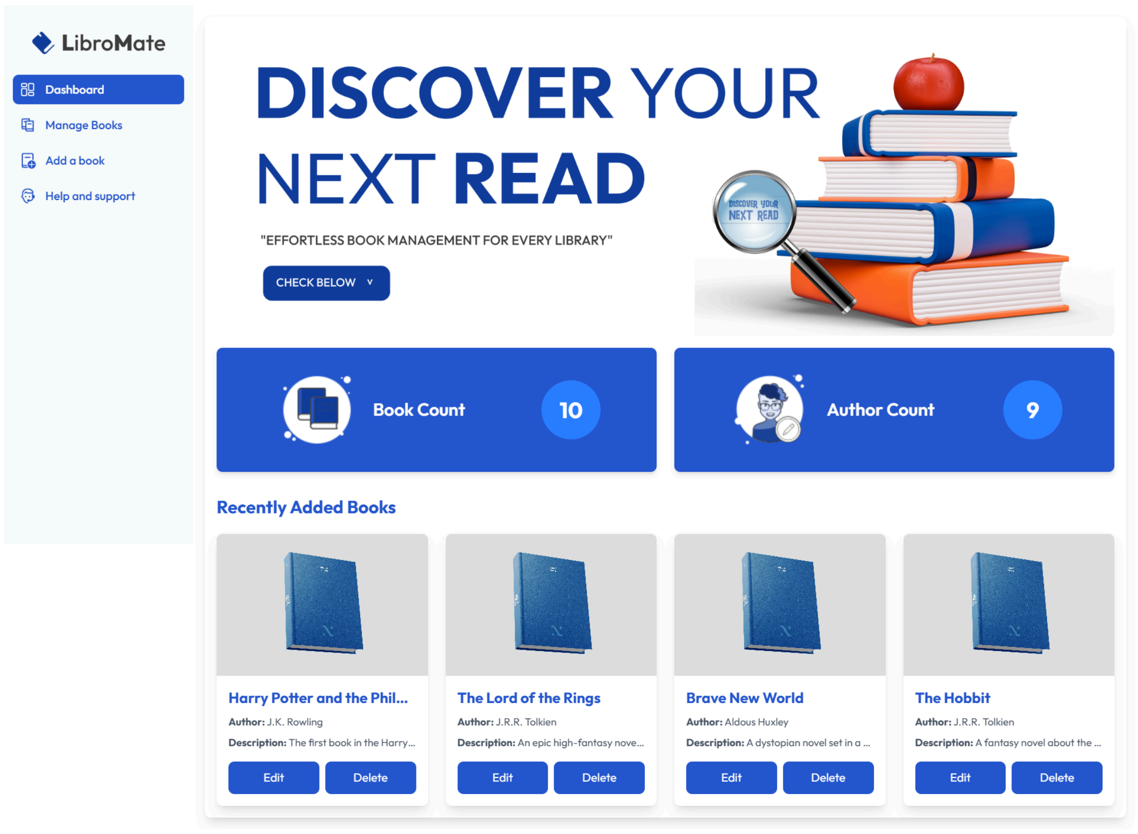
Library Management System

Development Report



Name: Malinga Premalal
Date: 25/02/2025
Internship Program: Software Engineering Internship
Company Name: Expernetic LLC

Pages Include:

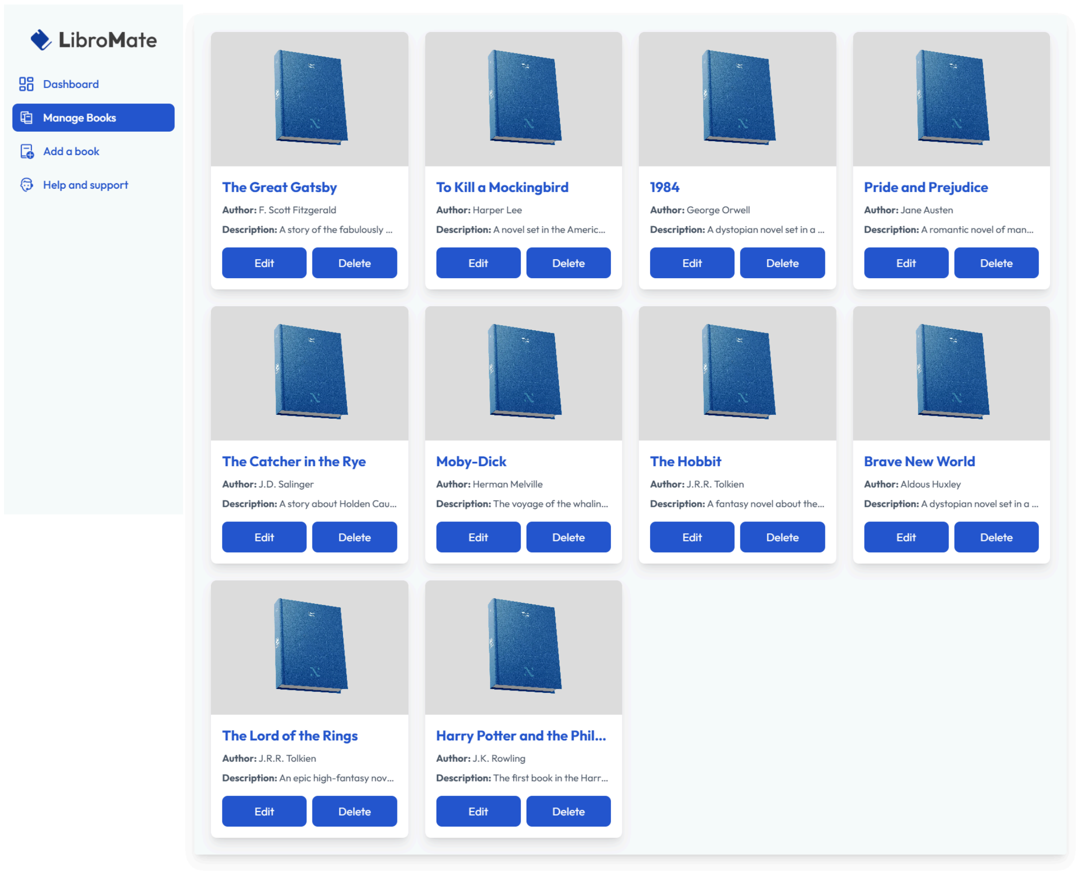


Harry Potter and the Phil...
Author: J.K. Rowling
Description: The first book in the Harry...
Edit Delete

The Lord of the Rings
Author: J.R.R. Tolkien
Description: An epic high-fantasy nove...
Edit Delete

Brave New World
Author: Aldous Huxley
Description: A dystopian novel set in a ...
Edit Delete

The Hobbit
Author: J.R.R. Tolkien
Description: A fantasy novel about the ...
Edit Delete



LibroMate

- Dashboard
- Manage Books
- Add a book
- Help and support

The Great Gatsby
Author: F. Scott Fitzgerald
Description: A story of the fabulously ...
Edit Delete

To Kill a Mockingbird
Author: Harper Lee
Description: A novel set in the Americ...
Edit Delete

1984
Author: George Orwell
Description: A dystopian novel set in a ...
Edit Delete

Pride and Prejudice
Author: Jane Austen
Description: A romantic novel of man...
Edit Delete

The Catcher in the Rye
Author: J.D. Salinger
Description: A story about Holden Cau...
Edit Delete

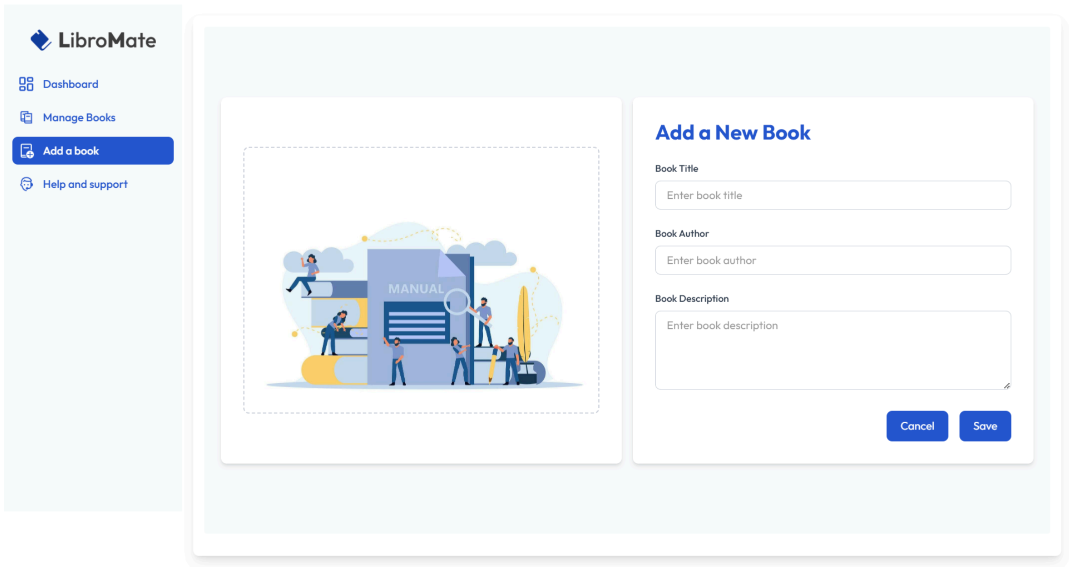
Moby-Dick
Author: Herman Melville
Description: The voyage of the whalin...
Edit Delete

The Hobbit
Author: J.R.R. Tolkien
Description: A fantasy novel about the...
Edit Delete

Brave New World
Author: Aldous Huxley
Description: A dystopian novel set in a ...
Edit Delete

The Lord of the Rings
Author: J.R.R. Tolkien
Description: An epic high-fantasy nov...
Edit Delete

Harry Potter and the Phil...
Author: J.K. Rowling
Description: The first book in the Harr...
Edit Delete



LibroMate

- Dashboard
- Manage Books
- Add a book
- Help and support

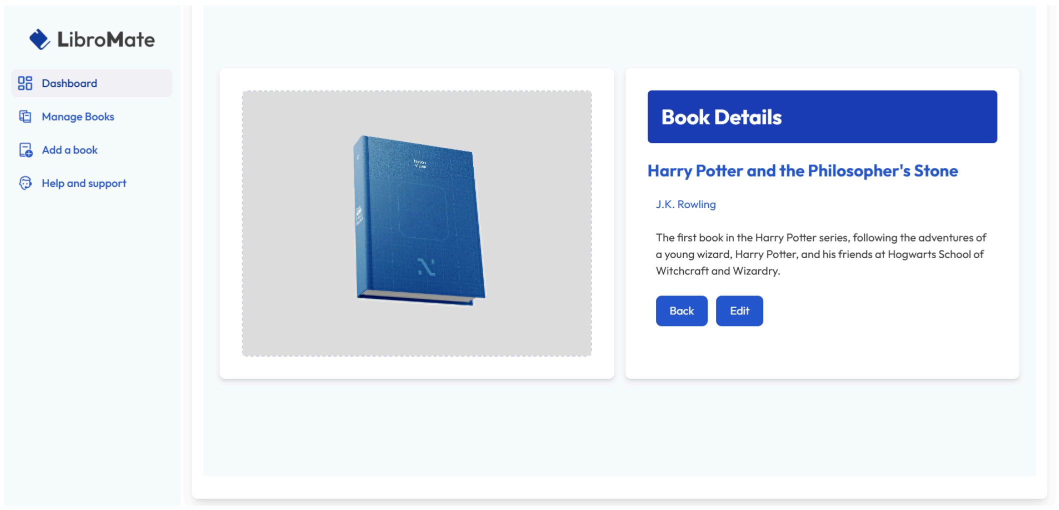
Add a New Book

Book Title
Enter book title

Book Author
Enter book author

Book Description
Enter book description

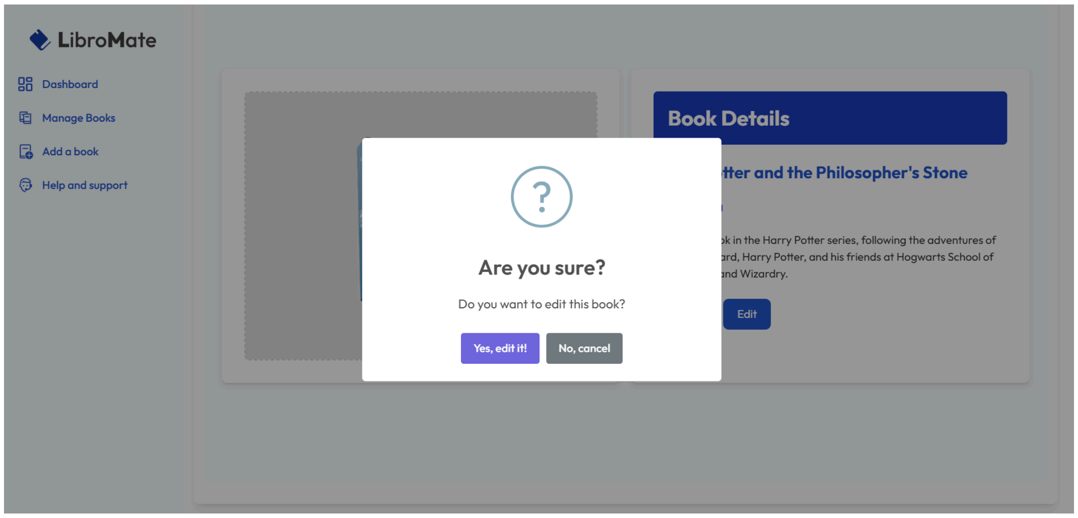
Cancel Save



Harry Potter and the Philosopher's Stone

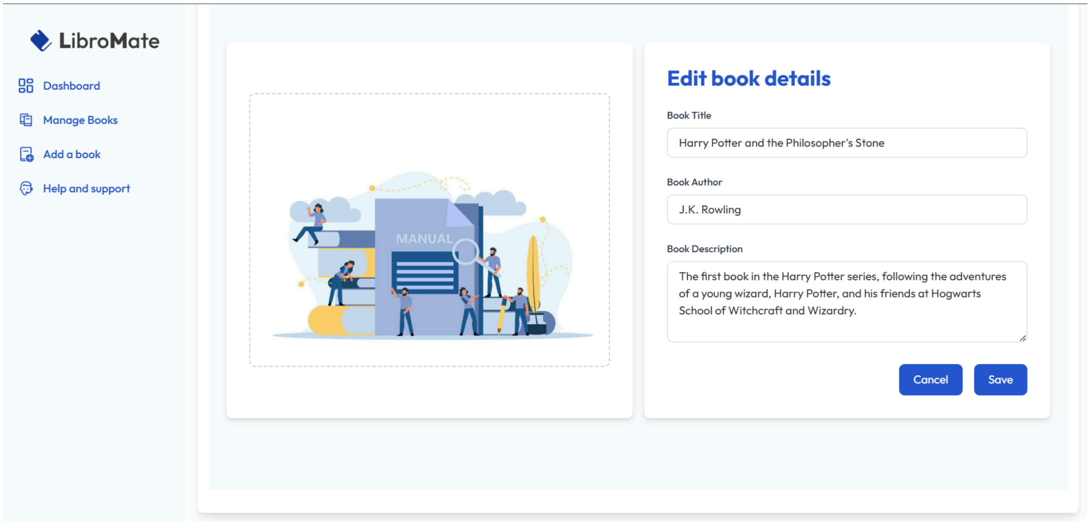
J.K. Rowling

The first book in the Harry Potter series, following the adventures of a young wizard, Harry Potter, and his friends at Hogwarts School of Witchcraft and Wizardry.



Are you sure?

Do you want to edit this book?

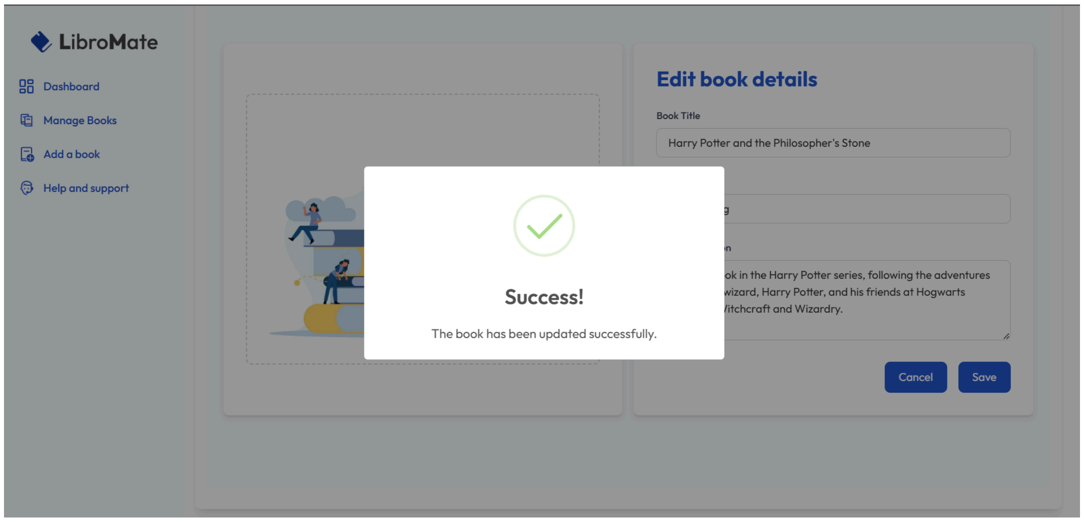


LibroMate

- Dashboard
- Manage Books
- Add a book
- Help and support

Edit book details

Book Title	Harry Potter and the Philosopher's Stone
Book Author	J.K. Rowling
Book Description	The first book in the Harry Potter series, following the adventures of a young wizard, Harry Potter, and his friends at Hogwarts School of Witchcraft and Wizardry.



LibroMate

- Dashboard
- Manage Books
- Add a book
- Help and support

Edit book details

Book Title

Harry Potter and the Philosopher's Stone

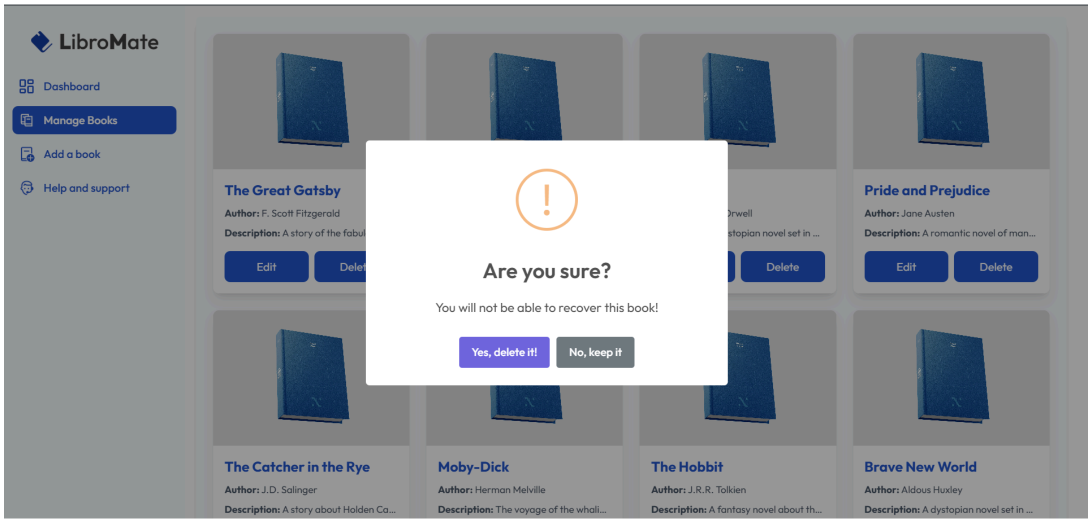
g

n

ok in the Harry Potter series, following the adventures wizard, Harry Potter, and his friends at Hogwarts Vitchcraft and Wizardry.

Success!

The book has been updated successfully.



Are you sure?

You will not be able to recover this book!

1.Introduction

Project Overview

LibroMate is a Library Management System designed to streamline the management of books in a library. This system provides an efficient and user-friendly interface that allows users to view available books, add new books, edit existing book details, and delete books as needed.

Purpose of the System

The primary goal of LibroMate is to enhance the efficiency of book management within a library by providing a simple and structured approach to handling book records.

Objectives

The key objectives of LibroMate are:

- To allow users to create new book records with essential details.
- To provide a structured way for users to view all existing books.
- To enable users to update book details when necessary.
- To offer a functionality to delete book records that are no longer required.
- To ensure a smooth and efficient user experience through an intuitive interface.

Technologies Used

LibroMate is developed using the following technologies:

- **Backend**: ASP.NET Web API – for handling API requests and database operations.
- **Database**: SQLite – for storing and managing book records.
- **Frontend**: React with TypeScript – for building a responsive and dynamic user interface.

This combination of technologies ensures that the system is scalable, efficient, and easy to maintain.

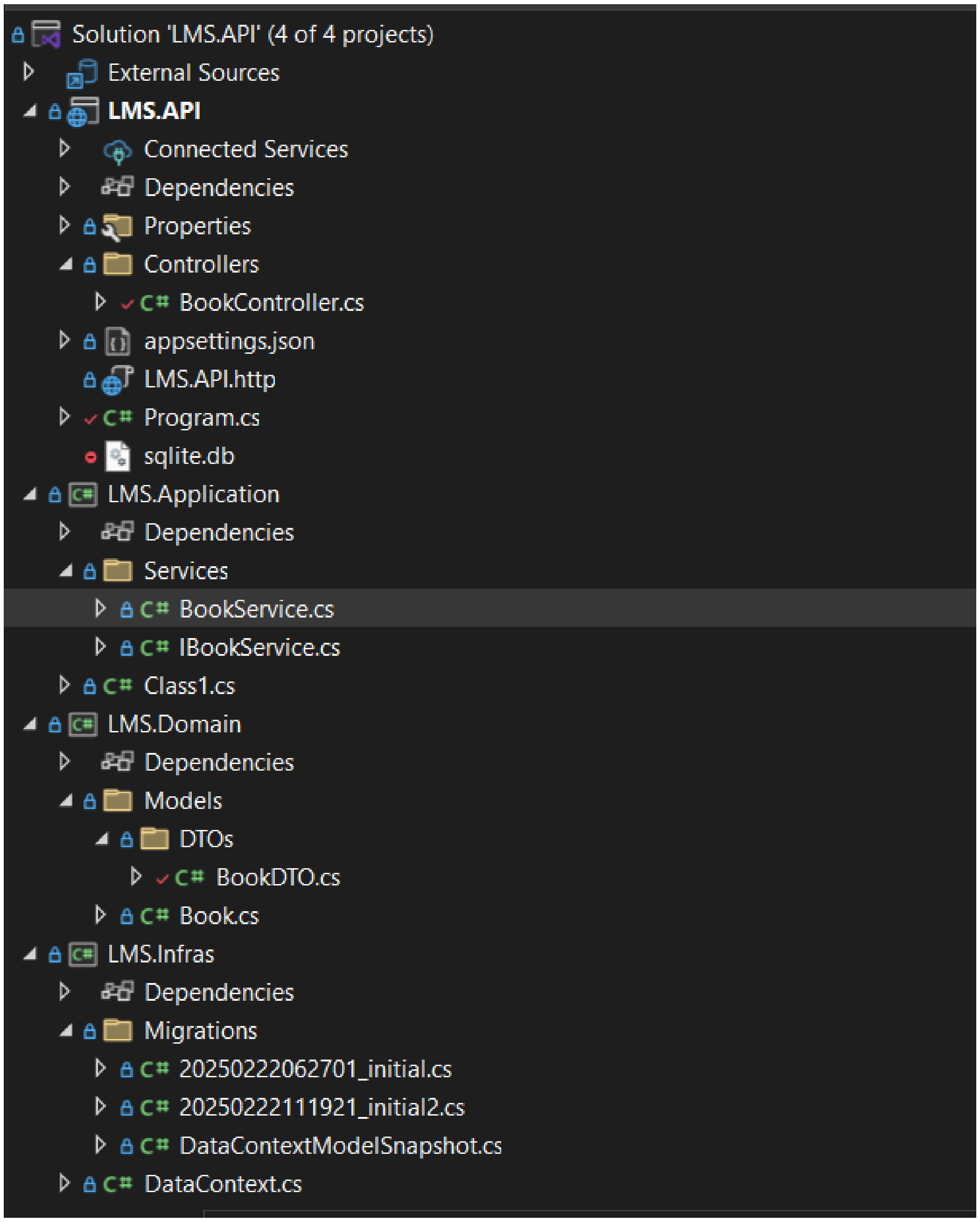
---

2. Backend Architecture & Development Approach

The backend of **LibroMate** is developed using **ASP.NET Web API** and follows a structured, modular approach to ensure **scalability, maintainability, and separation of concerns**. Key design principles include:

- **DTO (Data Transfer Object) Usage** – DTOs are used to **prevent direct exposure of database models**, ensuring better data encapsulation and flexibility.
- **Layered Architecture** – The backend is structured into:
    - **Controllers –** Handle API requests and responses.
    - **Service Layer** – Contains **business logic** and interacts with the database.
    - **Repository Layer** – Manages **direct database operations** via Entity Framework.
- **Dependency Injection (DI)** – The **Service Layer** is injected into the API controllers through **interfaces**, ensuring a loosely coupled and testable architecture.
- **Exception Handling** – Error handling is **implemented at both the controller and service layers** to provide robust and meaningful error responses.

A **screenshot of the backend folder structure** will be attached to illustrate this architecture.



```
Solution 'LMS.API' (4 of 4 projects)
  External Sources
  LMS.API
    Connected Services
    Dependencies
    Properties
    Controllers
      BookController.cs
    appsettings.json
    LMS.API.http
    Program.cs
    sqlite.db
  LMS.Application
    Dependencies
    Services
      BookService.cs
      IBookService.cs
    Class1.cs
  LMS.Domain
    Dependencies
    Models
      DTOs
        BookDTO.cs
      Book.cs
  LMS.Infras
    Dependencies
    Migrations
      20250222062701_initial.cs
      20250222111921_initial2.cs
      DataContextModelSnapshot.cs
    DataContext.cs
```

---

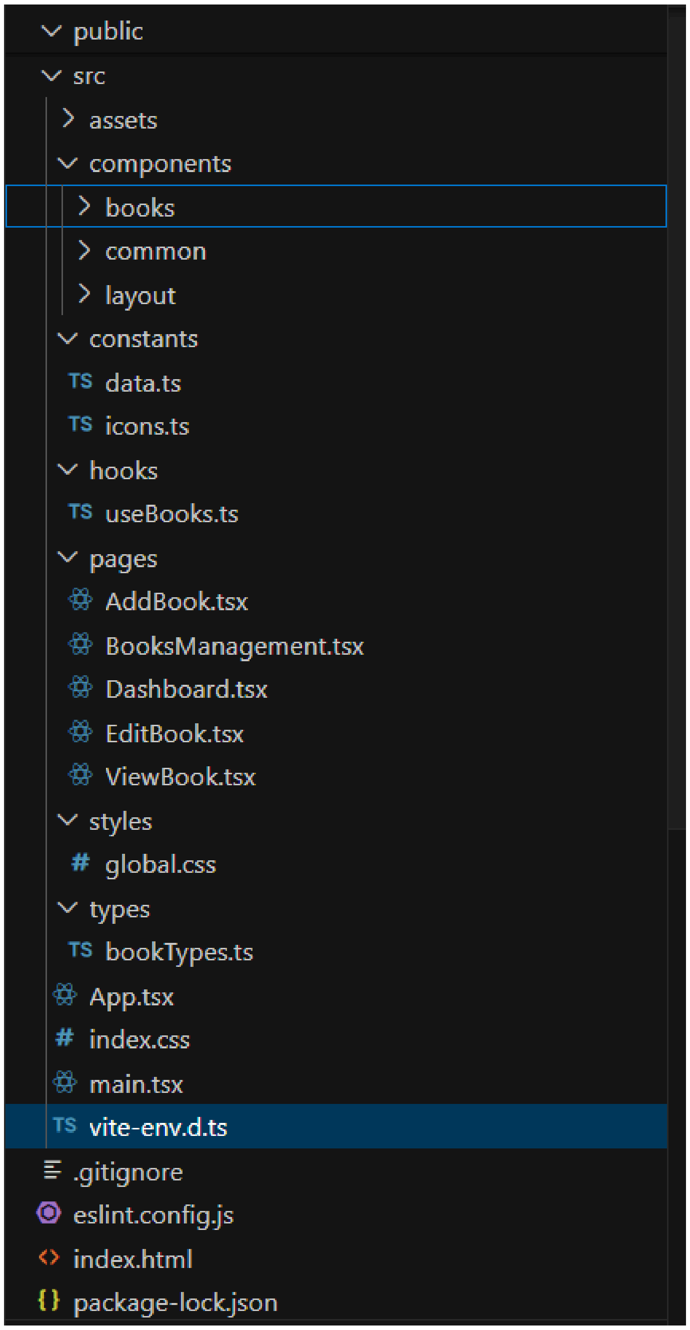
3. Frontend Development

User Interface Design & Features

The frontend of **LibroMate** is designed to be **clean, responsive, and user-friendly**. The interface ensures a smooth user experience by providing an intuitive layout and easy navigation for managing books. The key design principles followed include:

- **Responsive Design** – The UI adapts seamlessly to different screen sizes, ensuring usability on both desktops and mobile devices.
- **Scalability & Reusability** – The application is structured to promote **code reuse and maintainability** by separating **state management** and using **reusable components**.
- **Enhanced User Experience** – Implemented **clear and informative alerts** using the sweetalert2 library for user feedback.
- **Error Handling** – Handled gracefully with **user-friendly messages and validation prompts**.

A **screenshot of the frontend folder structure** will be attached to illustrate this architecture.



```
public
src
  assets
  components
    books
    common
    layout
  constants
    data.ts
    icons.ts
  hooks
    useBooks.ts
  pages
    AddBook.tsx
    BooksManagement.tsx
    Dashboard.tsx
    EditBook.tsx
    ViewBook.tsx
  styles
    global.css
  types
    bookTypes.ts
  App.tsx
  index.css
  main.tsx
  vite-env.d.ts
.gitignore
eslint.config.js
index.html
package-lock.json
```

---

4. Challenges Faced & Solutions

During the development of **LibroMate**, I encountered several technical challenges that required me to **learn new concepts, refine my skills, and apply best practices** to build an efficient and maintainable system. Below are the key challenges I faced and the solutions I implemented.

Backend Challenges & Solutions

1. Working with SQLite in ASP.NET

Challenge:

- I had limited experience integrating **SQLite** with **ASP.NET** and configuring it with **Entity Framework (EF Core)**.
- Understanding how to define models, apply migrations, and interact with the database was initially challenging.

Solution:

- I researched **Entity Framework Core** and **SQLite-specific configurations** to understand how to properly set up the database.
- Used **EF Core migrations** to manage database schema changes efficiently.
- Implemented proper **database connection handling** to prevent performance bottlenecks.

Key Learning:

- Gained hands-on experience working with **SQLite in ASP.NET**, including setting up migrations and database interactions.

---

2. Implementing a Layered Backend Architecture

Challenge:

- I needed to structure the **backend in a clean, modular, and scalable way** by separating concerns properly.
- Initially, I was unsure how to **effectively organize controllers, services, and data layers** in ASP.NET.

Solution:

- Followed **best practices for layered architecture**:
  - **Controllers** only handle HTTP requests and delegate logic to services.
  - **Services** contain the business logic and interact with the database.

    - **DTOs (Data Transfer Objects)** are used to **prevent direct model exposure**, ensuring better data encapsulation and security.
- Applied **Dependency Injection (DI)** to inject services into controllers, promoting maintainability and flexibility.

Key Learning:

- **Refreshed my knowledge** on designing **a layered architecture in ASP.NET**.
- Learned how **DTOs** improve security and prevent **tight coupling between the database and API responses**.
- Gained **better understanding of Dependency Injection** to keep code modular and testable.

Frontend Challenges & Solutions

1. Structuring Reusable Components

Challenge:

- Initially, it was difficult to decide how to **separate components and functions** for better reusability.
- I started by writing larger components, which made the code less readable and harder to maintain.

Solution:

- Refactored the frontend by **breaking down large components into smaller, reusable components**.
- Used **props** and **state management**, ensuring smooth data flow between components.

Key Learning:

- Improved my ability to **structure React components** in a scalable way.
- Learned how **separating components** enhances **code readability and maintainability**.

---

2. Sidebar Component Structure Issue

Challenge:

- Initially, I used a simple <a> tag for sidebar navigation.

- However, I later realized that this approach caused **full page reloads**, breaking the **single-page application (SPA) behavior** of React.

Solution:

- Replaced <a> tags with **React Router's <Link> component**, which allows seamless navigation **without refreshing the page**.
- Ensured that the sidebar works properly within React's component-based structure.

Key Learning:

- Understood the importance of using **React Router's <Link> component** for proper **client-side navigation** in React applications.

Deployment & Setup Instructions

This section provides step-by-step instructions on how to set up and run the **LibroMate** application locally, including backend and frontend setup, required dependencies, and necessary configurations.

Backend Setup (ASP.NET Web API & SQLite)

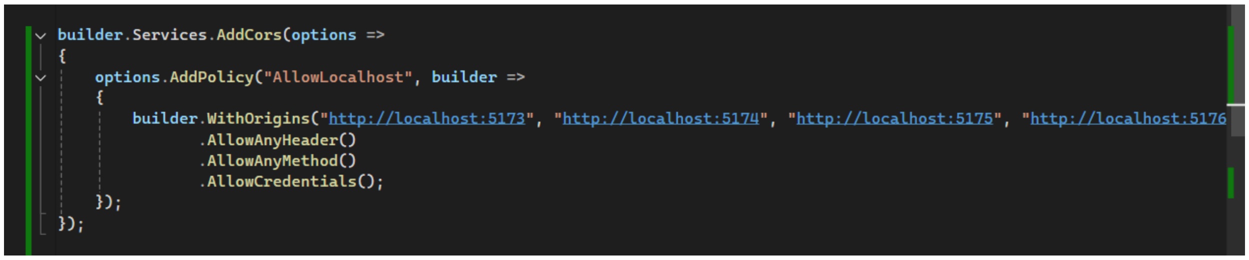
To set up and run the backend, follow these steps:

Step 1: Open the Project in Visual Studio

- Open **Visual Studio** and load the backend solution.
- Ensure all dependencies are installed (NuGet will handle missing packages).

Step 2: Configure CORS (Cross-Origin Resource Sharing)

Since the frontend runs on a different port, you need to **enable CORS** in the backend by adding the following configuration in Program.cs



```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowLocalhost", builder =>
    {
        builder.WithOrigins("http://localhost:5173", "http://localhost:5174", "http://localhost:5175", "http://localhost:5176
                .AllowAnyHeader()
                .AllowAnyMethod()
                .AllowCredentials();
    });
});
```

💡 **Make sure to update the allowed origins** to match the port where your frontend is running.

Step 3: Run the Backend API

- Click the **Start** button in Visual Studio.
- The API will start running, and **Swagger** (API documentation) will open in your browser.

Frontend Setup (React + TypeScript + Tailwind CSS)

To set up and run the frontend, follow these steps:

Step 1: Install Dependencies

- Open a terminal inside the frontend project directory.
- Run the following command to install all required dependencies

```
npm install
```

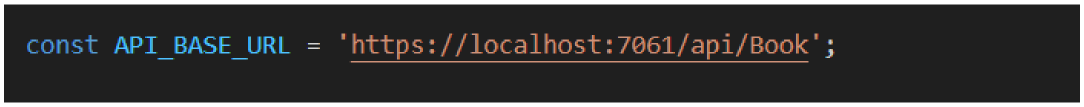
Step 2: Start the Frontend Application

- After installing dependencies, start the development server by running:

```
npm run dev
```

Configuring API Base URL

To ensure the frontend communicates with the correct backend API, update the **API base URL** in your frontend project:



```
const API_BASE_URL = 'https://localhost:7061/api/Book';
```

💡 **Important:**

- Make sure the port number (7061) matches your **actual backend API port** in **Visual Studio**.
- If the backend runs on a different port, update it accordingly.

Conclusion

The development of **LibroMate** has been a successful learning experience, integrating **ASP.NET Web API, SQLite, React, TypeScript, and Tailwind CSS** to build a functional **Library Management System**. The project follows a **structured and scalable architecture**, ensuring smooth communication between the **backend and frontend** while maintaining a clean and responsive user interface.

Project Repository Links

- **Backend Repository:** https://github.com/N-Malinga/LMSbackend
- **Frontend Repository:** https://github.com/N-Malinga/LMSFrontend

This marks the completion of **LibroMate**, a simple yet efficient Library Management System. 🚀