Library Management System
Assignment (Report)

1. Introduction

The Library Management System is a full-stack web application built to streamline the management of books in a library. It allows users to efficiently create, view, update, and delete book records. The system is developed using modern best practices, featuring a well-structured backend API and a responsive, user-friendly frontend interface.

Tech Stack

Layers	Tools and Technologies
Backend	C# / .NET
Frontend	Typescript / React
Database	SQLite
Tools	Postman / Git / Vs Code

Database Design

Field	Type	Description
Id	Int (PK)	Unique Identifier
Title	String	Book Title
Author	String	Book Author
ISBN	Int	Book ISBN
Category	String	Book Category

2. Backend

The backend is implemented using **ASP.NET Core Web API** with **Entity Framework Core** and **SQLite** for persistent storage.

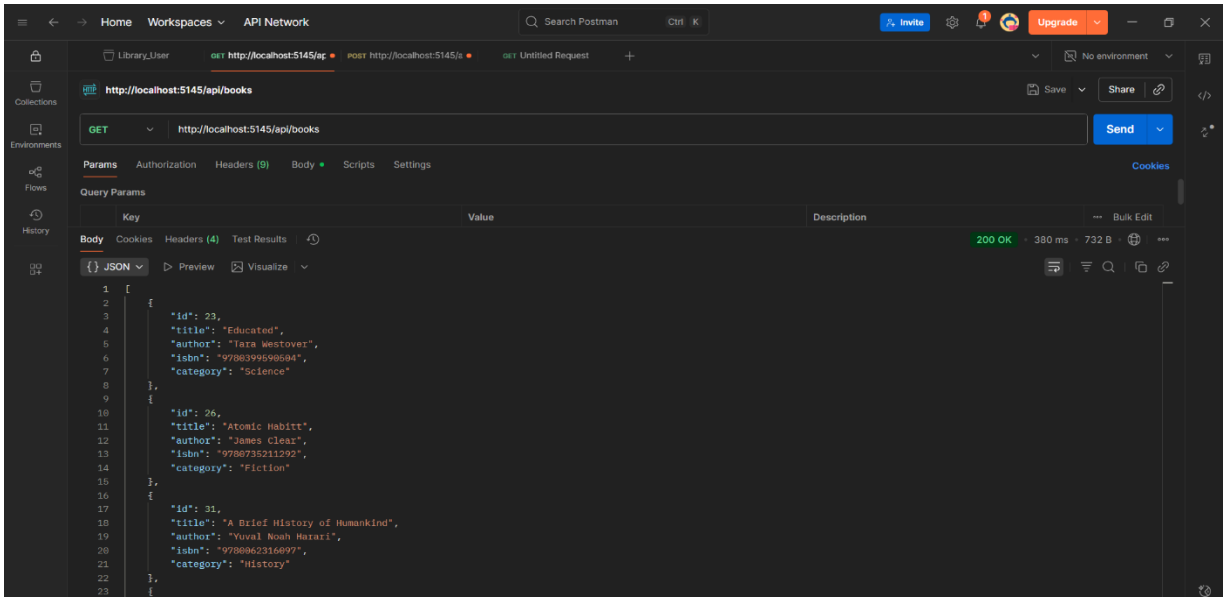
Key points:

- Implements **CRUD operations** for products.
- Exposes RESTful endpoints for frontend consumption.
- **Data validation** and **error handling** are included to ensure data integrity.
- **DbContext** manages database access and migrations.

Backend Structure

```
LibraryBackend/
├── Controllers/
│   └── BooksController.cs
├── Data/
│   └── LibraryContext.cs
├── DTOs/
│   ├── BookCreateDto.cs
│   ├── BookReadDto.cs
│   └── BookUpdateDto.cs
├── Models/
│   └── Book.cs
├── Services/
│   ├── BookService.cs
│   └── IBookService.cs
├── Migrations/
├── Library.db
└── Program.cs
```

API Testing



Endpoints

Method	Endpoint	Description
GET	/api/books	Get all books
GET	/api/books/{id}	Get a book by id
POST	/api/books	Create a new book
PUT	/api/books{id}	Update a existing book
DELETE	/api/books{id}	Delete a book

3. Frontend

The frontend is built with **React + TypeScript**:

- Uses **Vite** for fast development.
- Implements **React Router** for navigation.
- **Components** handle views such as BookList, BookForm
- **Axios** is used to call backend API endpoints.
- **Tailwind CSS** provides responsive UI design.
- **Reat Hot Toast** to show the nortification

Frontend Structure

```
LibraryFrontend/
├── node_modules/
├── public/
├── src/
│   ├── api/
│   ├── assets/
│   ├── components/
│   ├── pages/
│   ├── App.css
│   ├── App.tsx
│   ├── index.css
│   ├── main.tsx
│   └── types.ts
├── .gitignore
```

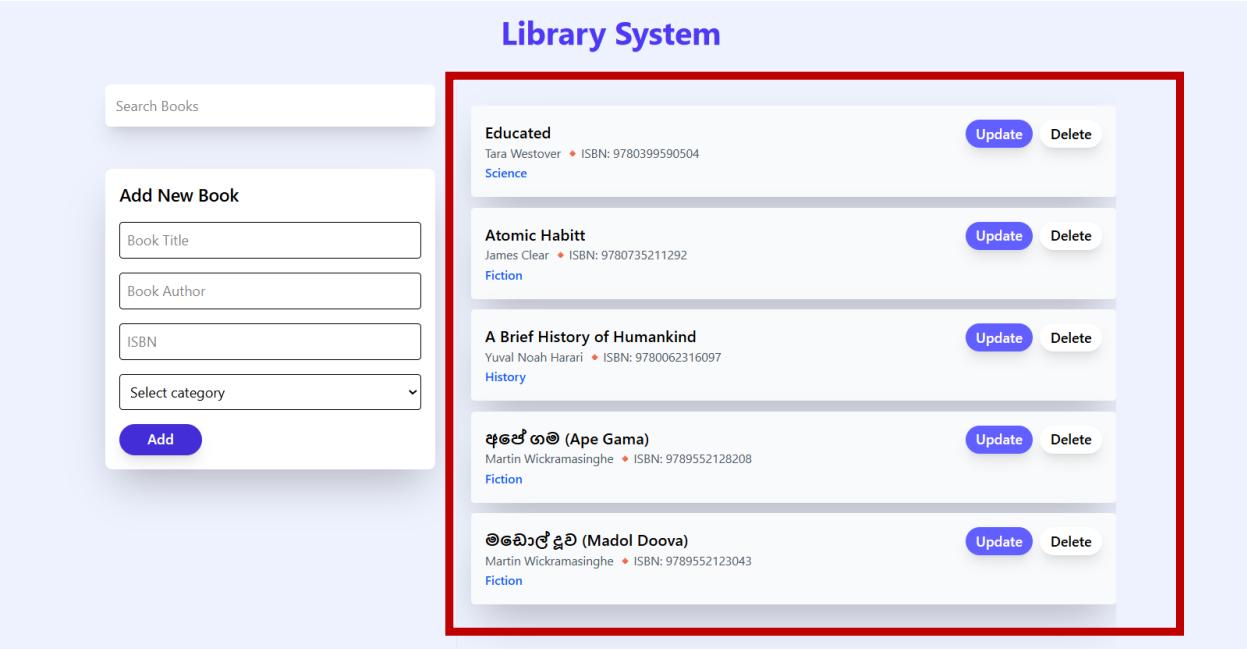
4. Development Steps

1. Initialize backend with ASP.NET Core Web API.

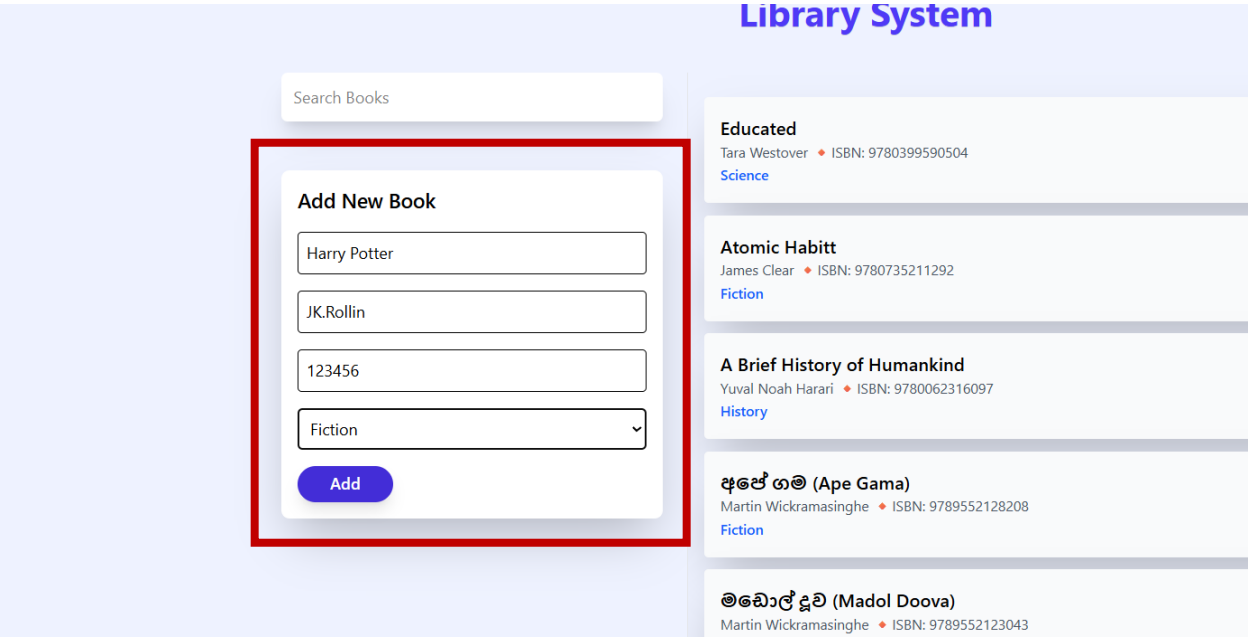
2. Configure EF Core and SQLite.
3. Implement Models, Controllers, Repositories, Services.
4. Create migrations and update database.
5. Initialize frontend with React + TypeScript.
6. Implement API service layer (bookApi.ts) and connect to backend.
7. Build UI components and pages.
8. Add routing and client-side validation.
9. Test CRUD operations manually via Postman

5. Application View

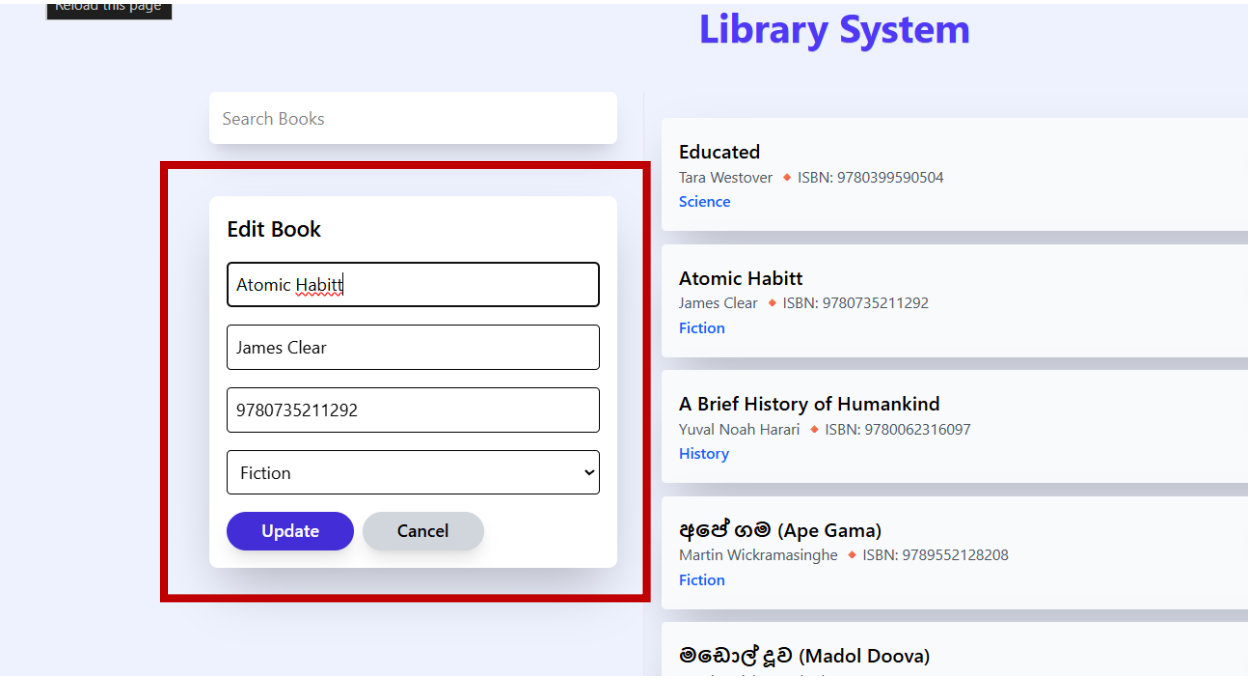
- Product List Page view



- Add Book Form view



- Update Book Form view



- Intro Page

- SQLite Database Table View

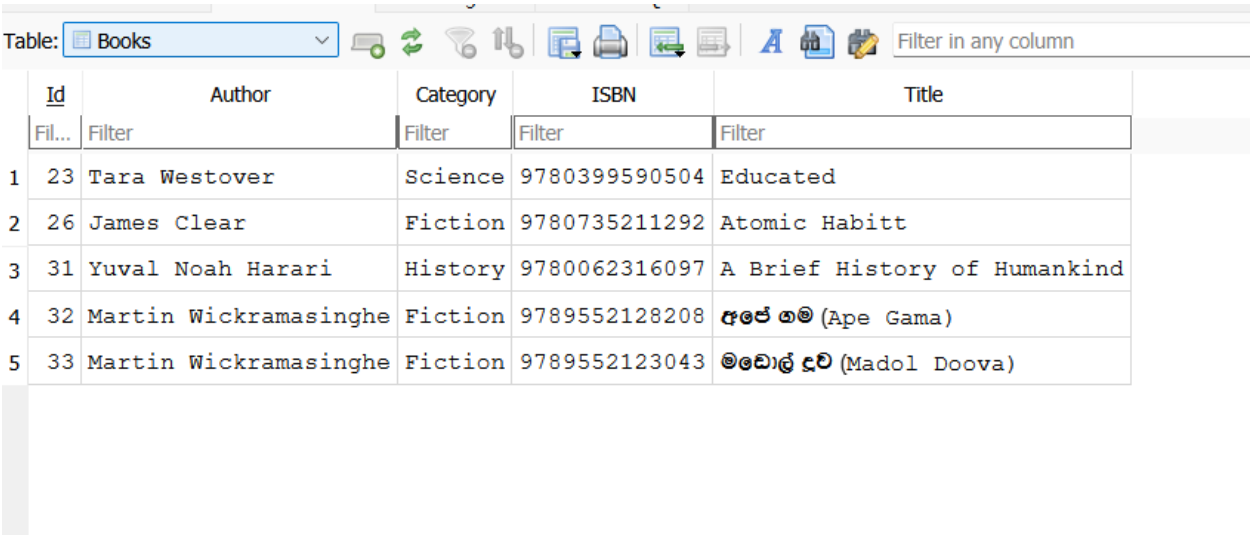


Table: Books

Filter in any column

	Id	Author	Category	ISBN	Title
	Fil...	Filter	Filter	Filter	Filter
1	23	Tara Westover	Science	9780399590504	Educated
2	26	James Clear	Fiction	9780735211292	Atomic Habitt
3	31	Yuval Noah Harari	History	9780062316097	A Brief History of Humankind
4	32	Martin Wickramasinghe	Fiction	9789552128208	අඹේ ගම (Ape Gama)
5	33	Martin Wickramasinghe	Fiction	9789552123043	මඩොල් දූව (Madol Doova)

6. Challenges & Solutions

Challenge	Solution
CORS errors during API calls	Configured CORS policy in backend
Type mismatches between frontend & backend	Created TypeScript interfaces for DTOs
Validation of forms	Implemented client-side and server-side validation
SQLite migration issues	Verified EF Core configuration and ran migrations carefully

7. Insights / Learning

- Understanding full-stack integration and data flow.
- Learning EF Core for ORM and database management.
- Enhancing React + TypeScript skills, including state management.

8. Conclusion

This project successfully demonstrates a **full-stack CRUD application** with seamless integration between **React frontend** and **ASP.NET Core backend**. All features are functional, validated, and tested. Future improvements can include authentication, pagination, and deployment in cloud environments.