Library Management System

Name: Malmi Wimalaweera

Contents

Introduction

This report provides a detailed overview of the Library Management System (LMS), developed as a RESTful API with a React-based frontend. The project basically manages books in a library and allows basic CRUD operations on book data: Create, Read, Update, Delete.

The system is developed using ASP.NET Core for the backend and React.js for the frontend. SQLite for the database.

Project Objectives

The overall purpose of the project is to:

Developed a full backend API to manage books.

Designing responsive and interactive front-end for end-user interaction.

Provides functionalities for viewing, adding, updating and deleting books.

Ensuring smooth integration between frontend and backend.

Technologies Used

Backend – ASP.NET Core

Frontend – React.js with typescript

Database – SQLite database

API Testing – Postman

Http Client – Axios

IDE – Visual Studio Code

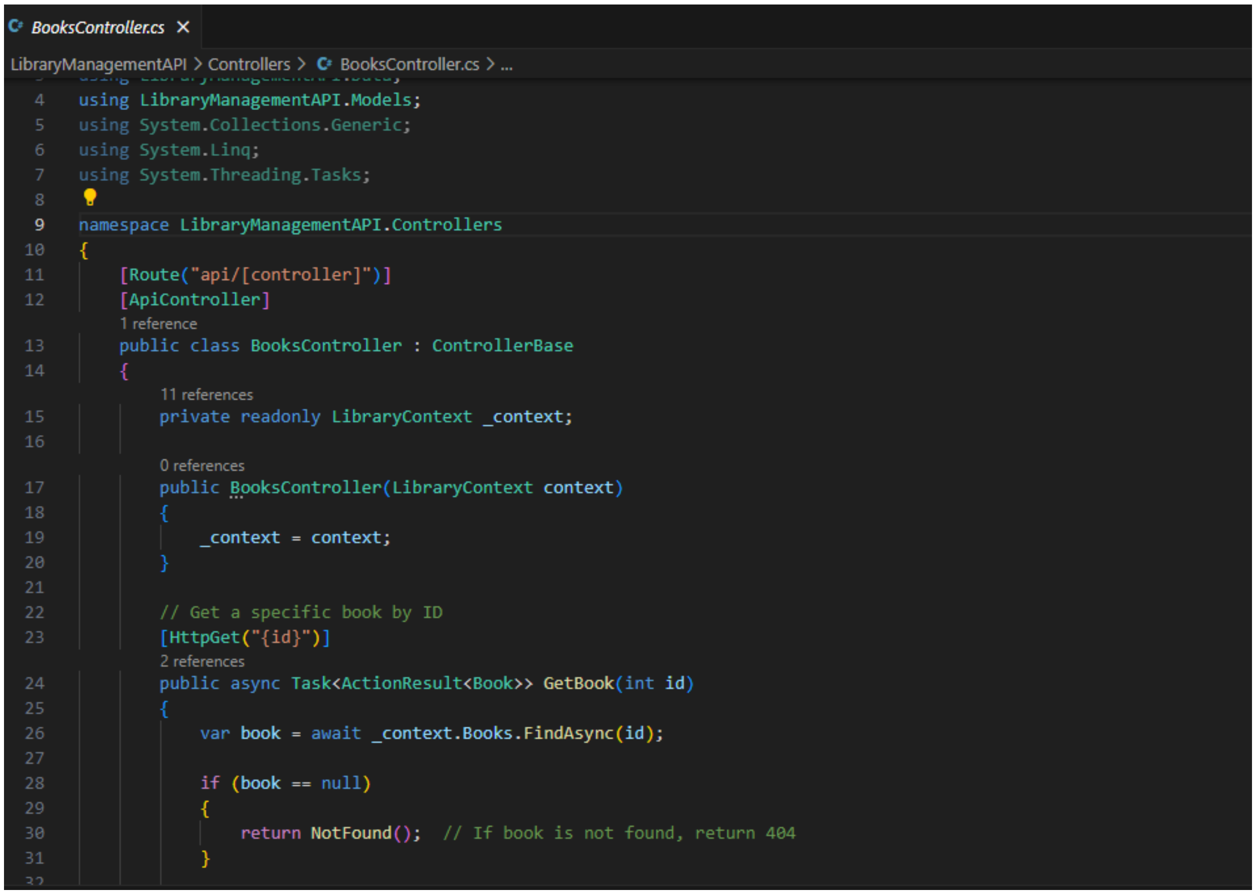
Backend Implementation

The backend was built using ASP.NET Core. The following endpoints were implemented.

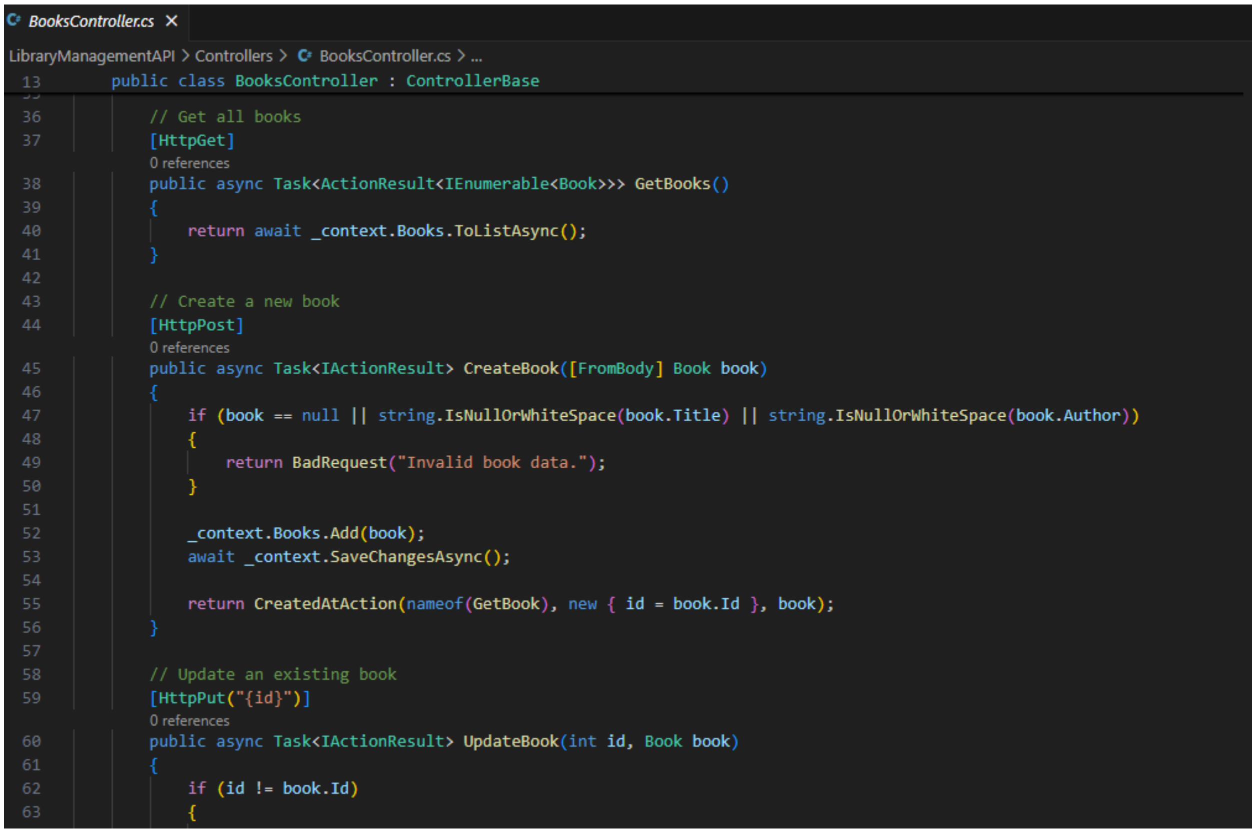
HTTP Method	Endpoint	Description
GET	/api/books	Retrieve all books

POST	/api/books	Add a new book
PUT	/api/books/{id}	Update an existing book
DELETE	/api/books/{id}	Delete an existing book

BookController.cs


BooksController.cs
LibraryManagementAPI > Controllers > BooksController.cs > ...

```
using LibraryManagementAPI.Models;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace LibraryManagementAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class BooksController : ControllerBase
    {
        11 references
        private readonly LibraryContext _context;

        0 references
        public BooksController(LibraryContext context)
        {
            _context = context;
        }

        // Get a specific book by ID
        [HttpGet("{id}")]
        2 references
        public async Task<ActionResult<Book>> GetBook(int id)
        {
            var book = await _context.Books.FindAsync(id);

            if (book == null)
            {
                return NotFound();  // If book is not found, return 404
            }
```



BooksController.cs
LibraryManagementAPI > Controllers > BooksController.cs > ...

```csharp
    public class BooksController : ControllerBase

        // Get all books
        [HttpGet]
        0 references
        public async Task<ActionResult<IEnumerable<Book>>> GetBooks()
        {
            return await _context.Books.ToListAsync();
        }

        // Create a new book
        [HttpPost]
        0 references
        public async Task<IActionResult> CreateBook([FromBody] Book book)
        {
            if (book == null || string.IsNullOrWhiteSpace(book.Title) || string.IsNullOrWhiteSpace(book.Author))
            {
                return BadRequest("Invalid book data.");
            }

            _context.Books.Add(book);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(GetBook), new { id = book.Id }, book);
        }

        // Update an existing book
        [HttpPut("{id}")]
        0 references
        public async Task<IActionResult> UpdateBook(int id, Book book)
        {
            if (id != book.Id)
            {
```

Frontend Implementation

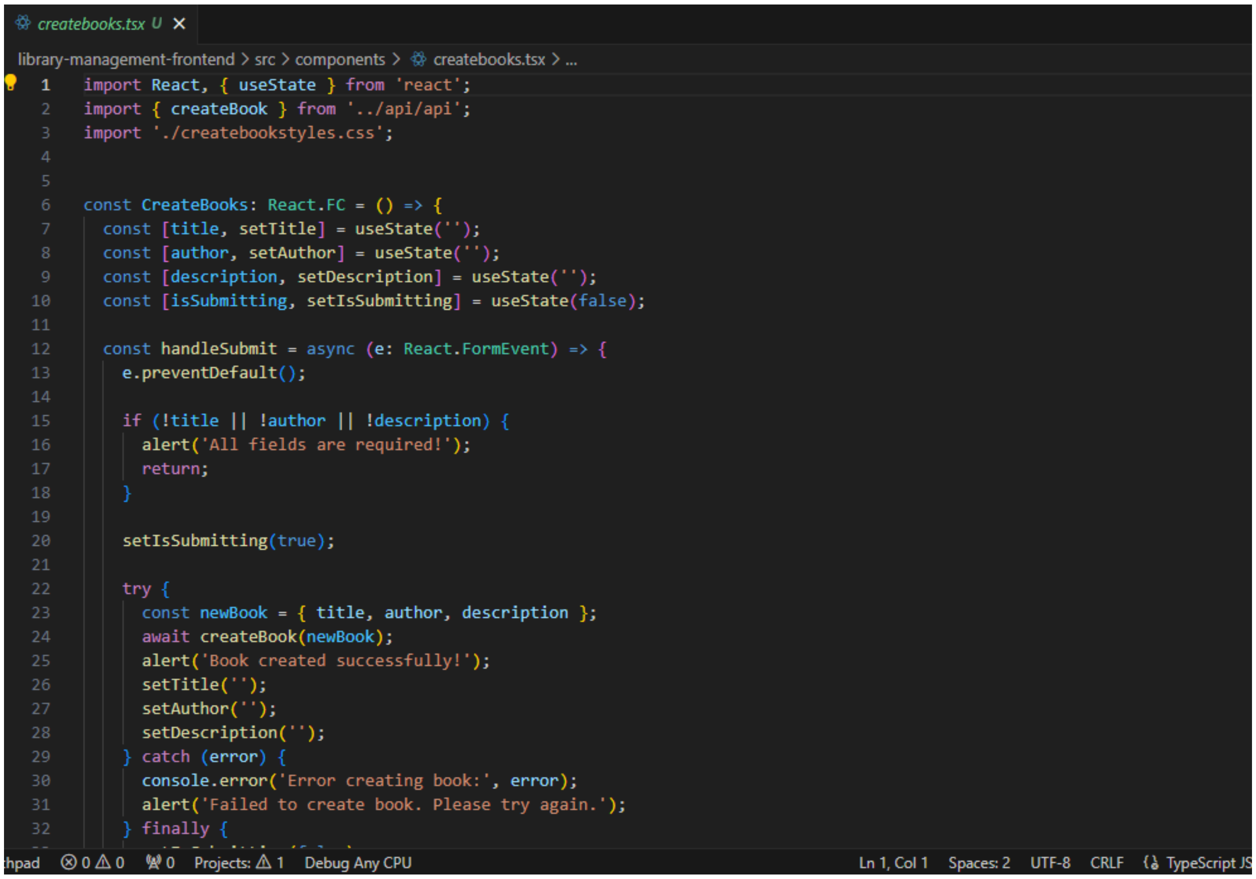
The frontend was built using React.js with TypeScript. The application allows users to:

1. Add a new book.
2. View all books.
3. Edit book details.
4. Delete books.

The components folder modularizes the functionality for CRUD operations:

createbooks.tsx

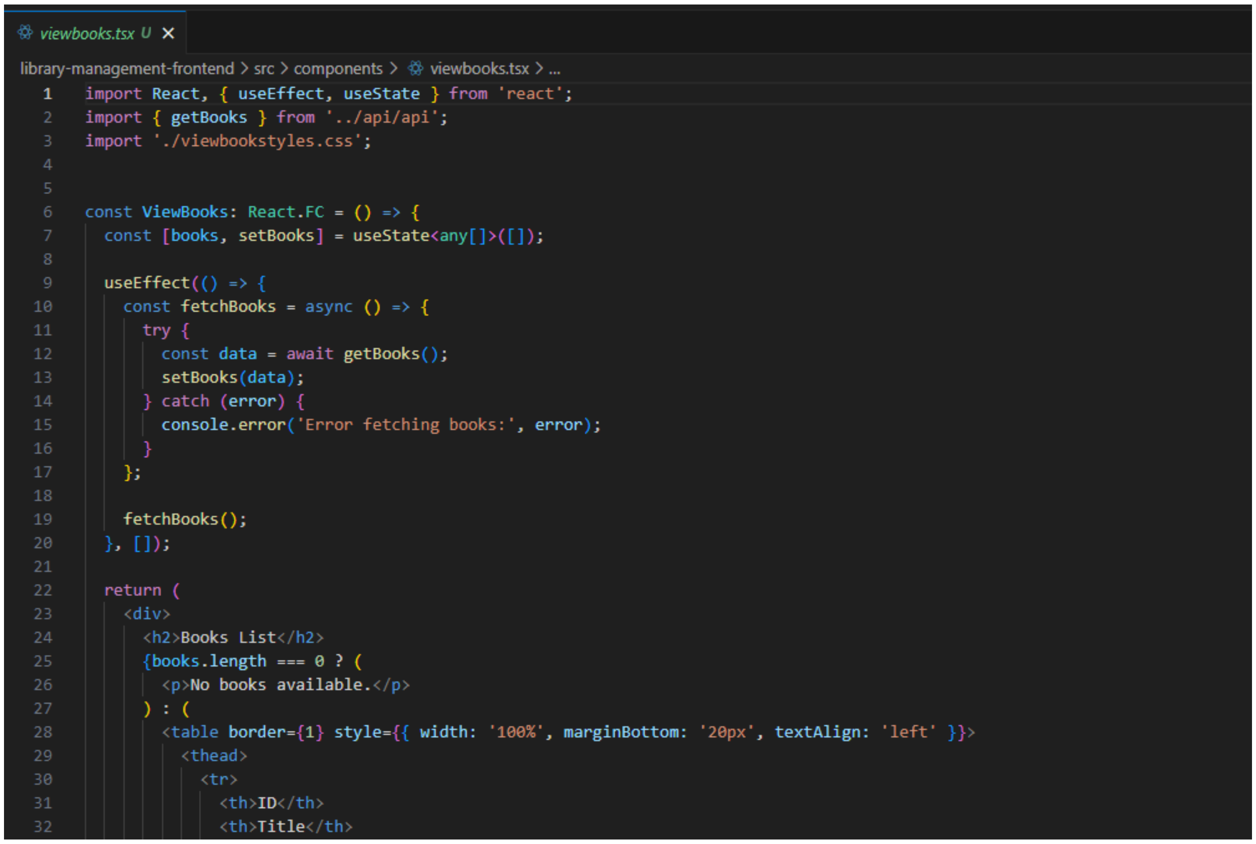
This is likely a form component to add new books to the library. Contains form fields for entering details like title, author, description. Sends POST requests via api.ts to add new books.


createbooks.tsx U
library-management-frontend > src > components > createbooks.tsx > ...
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF TypeScript JS
Debug Any CPU

```
import React, { useState } from 'react';
import { createBook } from '../api/api';
import './createbookstyles.css';


const CreateBooks: React.FC = () => {
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [description, setDescription] = useState('');
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    if (!title || !author || !description) {
      alert('All fields are required!');
      return;
    }

    setIsSubmitting(true);

    try {
      const newBook = { title, author, description };
      await createBook(newBook);
      alert('Book created successfully!');
      setTitle('');
      setAuthor('');
      setDescription('');
    } catch (error) {
      console.error('Error creating book:', error);
      alert('Failed to create book. Please try again.');
    } finally {
```

viewbooks.tsx

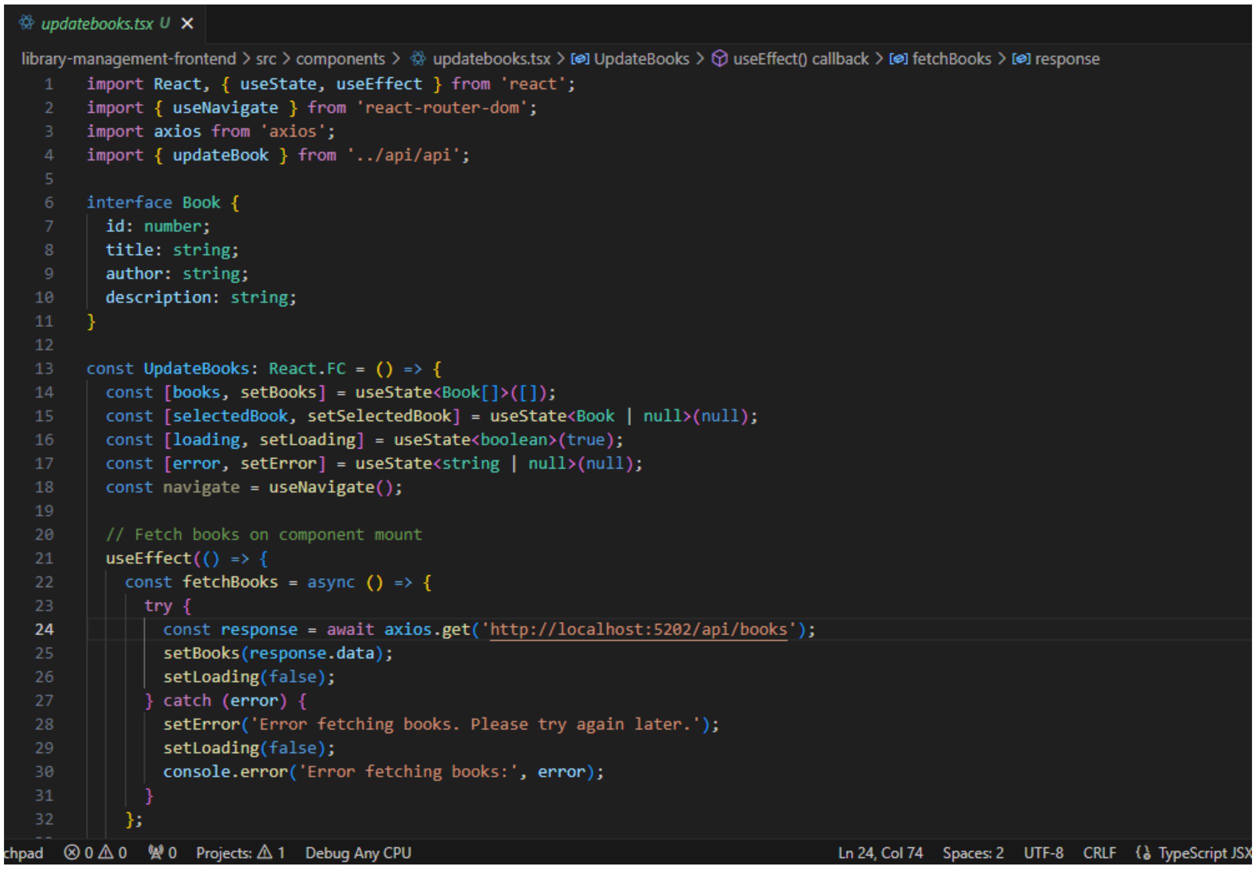
This component displays a list of books in a table or grid format. Fetches book data from the API using GET requests and renders it in a structured layout.


viewbooks.tsx U ×
library-management-frontend > src > components > viewbooks.tsx > ...

```tsx
import React, { useEffect, useState } from 'react';
import { getBooks } from '../api/api';
import './viewbookstyles.css';


const ViewBooks: React.FC = () => {
  const [books, setBooks] = useState<any[]>([]);

  useEffect(() => {
    const fetchBooks = async () => {
      try {
        const data = await getBooks();
        setBooks(data);
      } catch (error) {
        console.error('Error fetching books:', error);
      }
    };

    fetchBooks();
  }, []);

  return (
    <div>
      <h2>Books List</h2>
      {books.length === 0 ? (
        <p>No books available.</p>
      ) : (
        <table border={1} style={{ width: '100%', marginBottom: '20px', textAlign: 'left' }}>
          <thead>
            <tr>
              <th>ID</th>
              <th>Title</th>
```

updatebooks.tsx

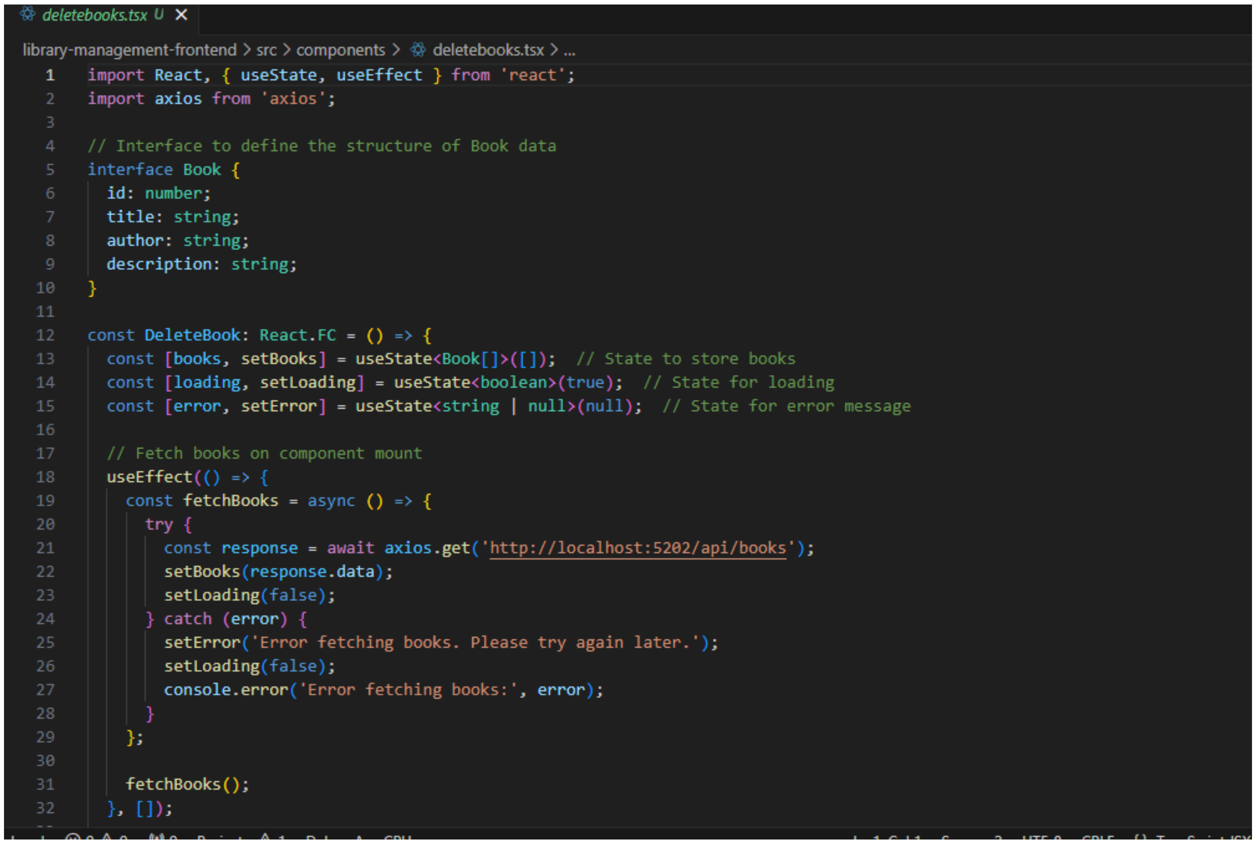
This component provides a form to edit or update existing books in the system. Retrieves current book details using GET and allows modifications before sending a PUT request.


updatebooks.tsx U
library-management-frontend > src > components > updatebooks.tsx > UpdateBooks > useEffect() callback > fetchBooks > response
chpad ⊗ 0 ⚠ 0 0 Projects: ⚠ 1 Debug Any CPU Ln 24, Col 74 Spaces: 2 UTF-8 CRLF TypeScript JSX

```tsx
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import { updateBook } from '../api/api';

interface Book {
  id: number;
  title: string;
  author: string;
  description: string;
}

const UpdateBooks: React.FC = () => {
  const [books, setBooks] = useState<Book[]>([]);
  const [selectedBook, setSelectedBook] = useState<Book | null>(null);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);
  const navigate = useNavigate();

  // Fetch books on component mount
  useEffect(() => {
    const fetchBooks = async () => {
      try {
        const response = await axios.get('http://localhost:5202/api/books');
        setBooks(response.data);
        setLoading(false);
      } catch (error) {
        setError('Error fetching books. Please try again later.');
        setLoading(false);
        console.error('Error fetching books:', error);
      }
    };
```

deletebooks.tsx

Manages the functionality for **deleting books** from the system. Calls the API to remove a book by its ID and updates the UI accordingly.

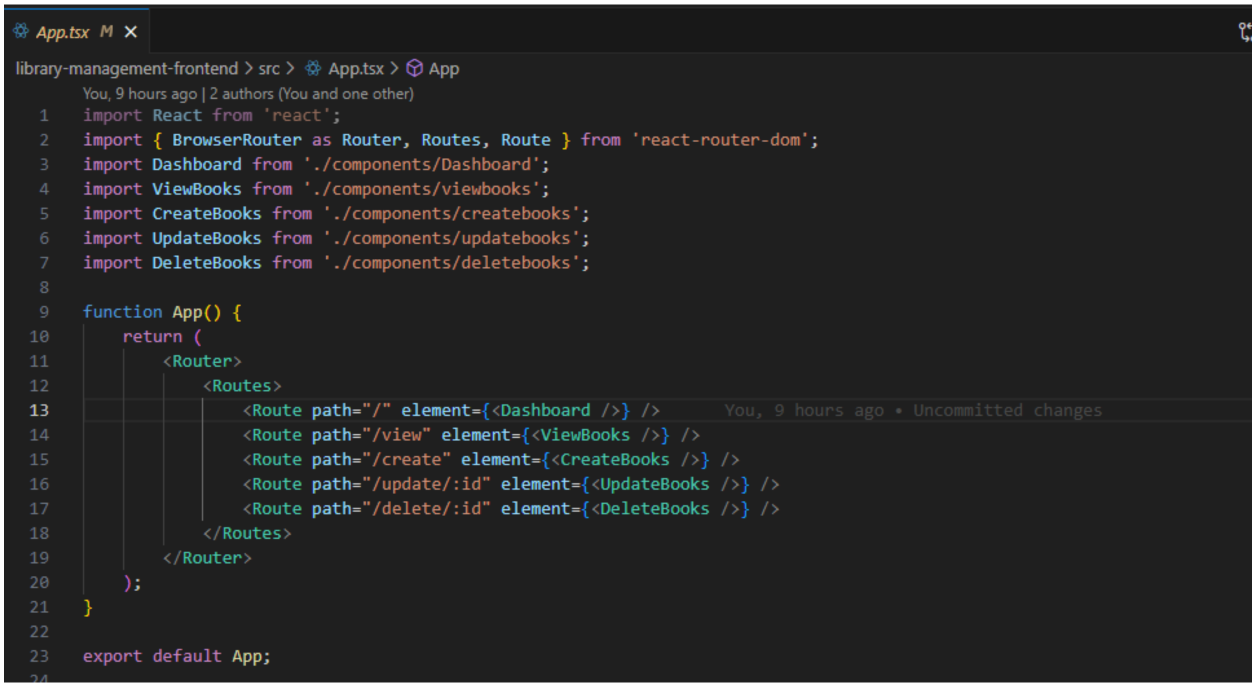

deletebooks.tsx U

library-management-frontend > src > components > deletebooks.tsx > ...

```tsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';

// Interface to define the structure of Book data
interface Book {
  id: number;
  title: string;
  author: string;
  description: string;
}

const DeleteBook: React.FC = () => {
  const [books, setBooks] = useState<Book[]>([]);  // State to store books
  const [loading, setLoading] = useState<boolean>(true);  // State for loading
  const [error, setError] = useState<string | null>(null);  // State for error message

  // Fetch books on component mount
  useEffect(() => {
    const fetchBooks = async () => {
      try {
        const response = await axios.get('http://localhost:5202/api/books');
        setBooks(response.data);
        setLoading(false);
      } catch (error) {
        setError('Error fetching books. Please try again later.');
        setLoading(false);
        console.error('Error fetching books:', error);
      }
    };

    fetchBooks();
  }, []);
```

App.tsx

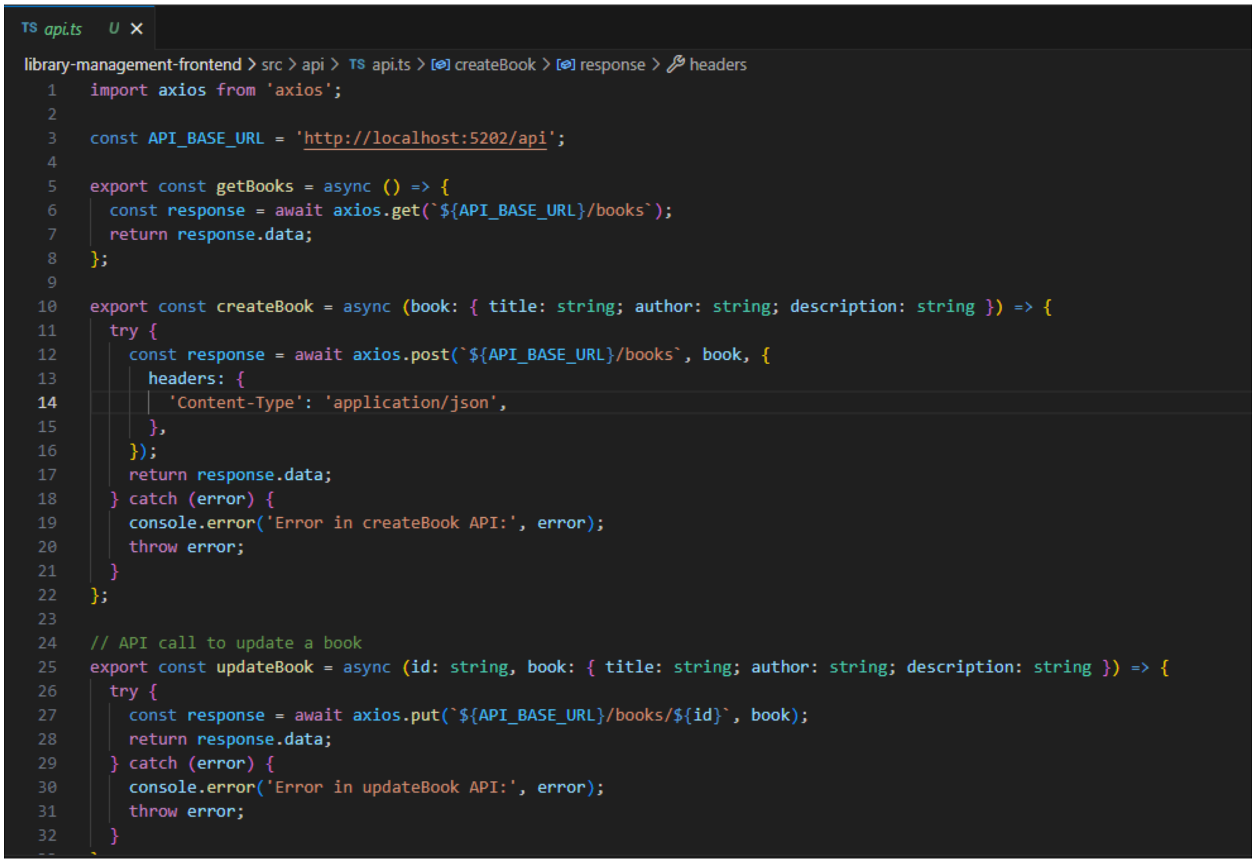
This is the main React component that serves as the entry point of the application. This includes routing to navigate between pages like "create", "view", "update", and "delete".


App.tsx M ×
library-management-frontend > src > App.tsx > App
You, 9 hours ago | 2 authors (You and one other)

```tsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Dashboard from './components/Dashboard';
import ViewBooks from './components/viewbooks';
import CreateBooks from './components/createbooks';
import UpdateBooks from './components/updatebooks';
import DeleteBooks from './components/deletebooks';

function App() {
    return (
        <Router>
            <Routes>
                <Route path="/" element={<Dashboard />} />        You, 9 hours ago • Uncommitted changes
                <Route path="/view" element={<ViewBooks />} />
                <Route path="/create" element={<CreateBooks />} />
                <Route path="/update/:id" element={<UpdateBooks />} />
                <Route path="/delete/:id" element={<DeleteBooks />} />
            </Routes>
        </Router>
    );
}

export default App;
```

API Integration:

The api.ts file centralizes all API calls, making it easy to manage and reuse API endpoints.


api.ts U ×
library-management-frontend > src > api > api.ts > createBook > response > headers

```
import axios from 'axios';

const API_BASE_URL = 'http://localhost:5202/api';

export const getBooks = async () => {
  const response = await axios.get(`${API_BASE_URL}/books`);
  return response.data;
};

export const createBook = async (book: { title: string; author: string; description: string }) => {
  try {
    const response = await axios.post(`${API_BASE_URL}/books`, book, {
      headers: {
        'Content-Type': 'application/json',
      },
    });
    return response.data;
  } catch (error) {
    console.error('Error in createBook API:', error);
    throw error;
  }
};

// API call to update a book
export const updateBook = async (id: string, book: { title: string; author: string; description: string }) => {
  try {
    const response = await axios.put(`${API_BASE_URL}/books/${id}`, book);
    return response.data;
  } catch (error) {
    console.error('Error in updateBook API:', error);
    throw error;
  }
```

Challenges Faced

There were some of the challenges during the development:

Cross Origin Resource Sharing Issues: Initial API calls from React were blocked because of CORS policy.

Data Binding Issues: Fields in the React form were not mapped correctly.

Additional Features

Error Handling: Relevant messages are displayed to the user if a book cannot be added or updated.

Form validation: Ensures no field is left empty.

Response Messages: Dynamic success and error messages on the frontend.

Conclusion

Library Management System achieved its goals, the creation of a backend API with the user-friendly React frontend for managing book records easily while SQL Server does its duty to ensure the integrity of data.

Some future enhancements would include advanced search features, pagination, and even user authentication to secure the system.