

BOOKS LIST

INTERDISCIPLINARY PROJECT

Submitted in partial fulfillment of the requirements for the award of Bachelor of
Technology in Information Technology

by

BROMELIA R (41120043)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY-1 UNIVERSITY BY UGC

Accredited "A++" by NAAC

12B Status by UGC |Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119

MAY 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **BROMELIA R (41120043)** who carried out the project entitled "**BOOKS LIST**" under my supervision from January 2024 to May 2024.

Internal Guide

Dr. K.S. KAMATCHI, M.E., Ph.D.,

Head of the Department

Dr. R. SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voce Examination held on 08-05-2024

Internal Examiner

External Examiner

DECLARATION

I, **BROMELIA.R (41120043)** hereby declare that the Project Report entitle "**BOOKS LIST**" done by me under the guidance of **Dr. K.S. KAMATCHI M.E.,Ph.D** is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 08-05-24

PLACE: CHENNAI



SIGNATURE OF THE CANDIDATE

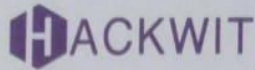
ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr.T. SASIKALA, M.E., Ph.D., Dean**, School of Computing, **Dr. R. SUBHASHINI, M.E., Ph.D.**, Head of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. K.S. KAMATCHI, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement which paved the way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.



HACKWIT TECHNOLOGIES PVT LTD
Redefine by Technologies

149, 1C, 2nd Floor, Mount Poonamallee Rd,
Ramapuram, Chennai - 600089.
admin@hackwittechnologies.com
www.hackwittechnologies.com

10.02.2024

CERTIFICATE FOR INTERNSHIP

TO WHOMSOEVER IT MAY CONCERN

This is to Certify that **BROMELIA R (Reg No: 41120043)** student of Sathyabama Institute of Science and Technology, B. Tech Information Technology, has successfully completed the Internship at Hackwit Technologies from 22nd January to 30th January 2024.

During the internship, **BROMELIA R** has closely worked as a part of the **React Full stack** by carrying out the project entitled "**BOOKS LIST**".

She demonstrated good developing and learning skill with self-motivated attitude to learn new things. Her contribution to implement the React Full stack was good. We wish her all the best for her future endeavors.

Best Regards,

Ravikumar S
CEO & Managing Director
Hackwit Technologies Pvt Ltd



COURSE COMPLETION
CERTIFICATE

THIS CERTIFICATE IS PRESENTED TO

BROMELIA. R

has successfully completed the 40 Hrs hands-on certification
courses on MERN STACK Development at
Sathyabama Institute of Science and Technology
from 22.01.2024 to 30.01.2024.

DIRECTOR

INSTRUCTOR

ABSTRACT

"Books List" is a modern web application built using the MERN (MongoDB, Express.js, React.js, Node.js) stack, designed to facilitate the creation and management of book lists. In today's digital age, where reading and knowledge acquisition are highly valued, managing one's reading materials efficiently is essential. This project aims to provide users with a seamless and intuitive platform to organize their reading preferences, track their progress, and discover new titles. The application features a user-friendly interface that allows users to easily add, edit, and delete books from their lists. Leveraging MongoDB as the database, Books List ensures data persistence and scalability, enabling users to store an extensive collection of books without compromising performance. Express.js provides robust backend support, handling HTTP requests and interactions with the database effectively. The frontend of the application is developed using React.js, a powerful JavaScript library for building user interfaces. React's component-based architecture facilitates the creation of reusable UI elements, enhancing code maintainability and scalability. With React, Books List delivers a dynamic and responsive user experience, enabling seamless interaction across devices and screen sizes. Node.js serves as the runtime environment for the application, enabling server-side JavaScript execution. Node.js powers the backend of Books List, handling server logic, routing, and integration with the frontend components. Leveraging the asynchronous, event-driven nature of Node.js, Books List ensures high performance and responsiveness, even under heavy loads. Key features of Books List include User authentication and authorization to ensure data privacy and security. CRUD (Create, Read, Update, Delete) operations for managing book lists. Search and filter functionality for discovering new books based on various criteria such as genre, author, or title. Progress tracking to monitor reading status and set reading goals.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	vii
	LIST OF FIGURES	x
1	INTRODUCTION	1
1.1	Problem statement	4
2	METHODOLOGY	5
2.1	Project objectives	5
2.2	Requirements	7
2.2.1	Software used	7
2.2.2	Packages used	7
2.3	Proposed model	8
2.3.1	Overview	8
2.3.2	Key Features	8
2.3.3	Deployment And Maintenance	9
2.4	System Architecture	9
2.4.1	Frontend	11
2.4.2	Backend	12
2.4.3	Database	14
2.4.4	Server	16
2.5	Modules	16

3	RESULTS AND DISCUSSION	18
3.1	Frontend result	18
3.2	Backend result	20
4	CONCLUSION AND FUTURE WORKS	22
	REFERENCE	23
	APPENDICES	24
	A. Source code	24
	B. Screenshots	31

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
2.4	Flow chart of books list	10
3.1	Books list	19
3.1.1	Table view	19
3.2	Create book	21
3.2.1	Enter details	21
3.3	Grid view	31
3.4	Delete	31
3.5	Edit	32
3.6	Display	32

CHAPTER 1

INTRODUCTION

In an age defined by rapid technological advancement, the enduring allure of books remains an anchor of human culture, transcending time and space. As our world becomes increasingly digital, so too does our relationship with literature evolve, embracing new mediums and platforms for consumption and interaction. It is within this dynamic landscape that we embark on a journey to harness the power of React.js, a versatile JavaScript library renowned for its ability to create dynamic and interactive user interfaces, to craft an immersive web application tailored specifically for bibliophiles worldwide. Our mission is not merely to create a functional tool for browsing books but to encapsulate the essence of the reading experience within a digital realm, offering users a gateway to explore, discover, and engage with literature in ways that transcend the limitations of traditional mediums.

At the heart of our lies a singular objective: to bridge the gap between the timeless appeal of literature and the ever-evolving digital landscape. We aspire to create a web application that not only simplifies the process of discovering new books but also enriches the reading experience by fostering a sense of community, exploration, and personal connection. Through meticulous attention to detail, innovative design, and seamless functionality, our goal is to empower users to immerse themselves in the world of literature with unparalleled ease and enthusiasm. Whether it's discovering a new author, sharing recommendations with friends, or engaging in lively discussions about beloved classics, our application seeks to become the ultimate companion for every bibliophile's literary journey.

The foundation of our application is built upon a carefully curated selection of features designed to cater to the diverse needs and preferences of readers of all ages and backgrounds. Central to our design philosophy is the notion of accessibility and inclusivity, ensuring that users can navigate the vast landscapes of literature with ease and delight. From comprehensive book listings organized by genre, author, and theme to immersive detailed views that provide in-depth information and insights, every aspect of our application is crafted with the user experience in mind. Our commitment to seamless navigation, responsive design, and intuitive functionality is evident in every feature, from the powerful search capabilities that enable users to find their next literary adventure with just a few clicks to the social integration tools that foster meaningful connections and discussions within the community.

Behind the scenes, our application is powered by a robust and sophisticated technology stack that enables us to deliver a seamless and immersive user experience. At the forefront of our technology stack is React.js, a JavaScript library renowned for its flexibility, scalability, and performance. Leveraging the power of React.js, we are able to create dynamic and interactive user interfaces that respond in real-time to user input, ensuring a smooth and engaging browsing experience. Complementing React.js is a suite of tools and technologies, including React Router for seamless navigation, API integration for real-time data updates, and state-of-the-art styling frameworks such as Bootstrap and Material-UI for visually stunning and responsive designs. Together, these technologies form the backbone of our application, enabling us to push the boundaries of what is possible in web development and create a truly immersive and captivating experience for users. . Reacts component-based architecture facilitates the creation of reusable UI elements, enhancing code maintainability and scalability. With React, Books List delivers a dynamic and responsive user experience, enabling seamless interaction across devices and screen sizes. Node.js

In conclusion, our journey to build a book listing application with React.js is not merely a technical but a labour of love and passion for literature. As we navigate the digital landscape and explore new ways of engaging with books, we remain steadfast in our commitment to preserving the timeless appeal and magic of the written word. Through innovation, creativity, and a deep reverence for the art of storytelling, we strive to create a digital oasis where readers of all backgrounds can come together to explore, discover, and celebrate the boundless wonders of literature. We invite you to join us on this journey as we embark on a quest to redefine the way we interact with books in the digital age and usher in a new era of reading that is both immersive and enriching. Together, let us embark on this adventure and chart a course towards a future where the joy of reading knows no bounds.

1.1 PROBLEM STATEMENT :

In the digital age, book enthusiasts face challenges in navigating the vast array of available literature across various platforms. Existing book listing applications often lack depth and interactivity, hindering the discovery process. To address this, we propose developing a centralized book listing application. Our objectives include aggregating data from multiple sources to provide curated book selections, enhancing user experience with intuitive design, personalizing recommendations, and fostering community engagement through social features. This application aims to revolutionize how readers discover, explore, and engage with literature in the digital era.

React's component-based architecture facilitates the creation of reusable UI elements, enhancing code maintainability and scalability. With React, Books List delivers a dynamic and responsive user experience, enabling seamless interaction across devices and screen sizes. Node.js serves as the runtime environment for the application, enabling server-side JavaScript execution. Node.js powers the backend of Books List, handling server logic, routing, and integration with the frontend components. Leveraging the asynchronous, event-driven nature of Node.js, Books List ensures high performance and responsiveness, even under heavy loads. Develop detailed book pages displaying essential information about each book, such as synopsis, author bio, publication details, and user ratings. Design the application with scalability in mind, allowing it to handle a growing number of users, books, and interactions without sacrificing performance. Ensure the application is fully responsive, providing a consistent data.

CHAPTER 2

METHODOLOGY

2.1 PROJECT OBJECTIVES:

- 1.**Comprehensive Book Database:** Compile and maintain a comprehensive database of books, including titles, authors, genres, summaries, and other relevant metadata.
- 2.**User-Friendly Interface:** Design an intuitive and visually appealing user interface that facilitates easy navigation and seamless browsing of books.
- 3.**Search Functionality:** Implement robust search functionality allowing users to quickly find books based on title, author, genre, keywords, or other criteria.
- 4.**Detailed Book Pages:** Develop detailed book pages displaying essential information about each book, such as synopsis, author bio, publication details, and user ratings.
5. **User Interaction Features:** Enable users to interact with books by adding them to reading lists, marking , leaving reviews, and rating books.
- 6.**Personalization:** Incorporate personalized recommendation algorithms to suggest books tailored to each user's reading preferences and history. Implement robust search functionality allowing users to quickly find books

7.**Responsive Design:** Ensure the application is fully responsive, providing a consistent and optimal experience across various devices and screen sizes.

8.**API Integration:** Integrate with external APIs (e.g., book databases, review platforms) to fetch real-time data and enrich the book listings.

9.**Social Sharing:** Implement social sharing features to allow users to share their books, reviews, and reading lists with friends and followers.

10.**Community Engagement:** Foster a sense of community among users by enabling discussions, book clubs, and user-generated content such as reviews and recommendations.

11. **Accessibility:** Ensure the application adheres to accessibility standards, making it usable for all users, including those with disabilities.

12.**Performance Optimization:** Optimize the performance of the application, including fast loading times, efficient data retrieval, and minimal latency.

13.**Security:** Implement robust security measures to protect user data, including encryption, secure authentication, and adherence to data privacy regulations.

14.**Scalability:** Design the application with scalability in mind, allowing it to handle a growing number of users, books, and interactions without sacrificing performance. Ensure the application is fully responsive, providing a consistent and optimal experience across various devices and screen sizes.

15.Documentation and Support: Provide comprehensive documentation and support resources for users and developers, including user guides, FAQs, and troubleshooting assistance.

2.2 REQUIREMENTS

2.2.1 SOFTWARE USED:

- VS code
- Mongo DB Atlas
- Postman API

→ VS code is used to create frontend and backend files.

→ Mongo DB atlas is used to create backend connection

→ POSTMAN API used to get the data using an API by providing the URL of localhost.

2.2.2 PACKAGES USED:

Front-end:

Material UI

React

Nodejs

React-Dom

React-Redux

2.3 PROPOSED MODEL

2.3.1. Overview:

The Books list project, developed using React.js, aims to provide users with a platform to manage their personal library of books. Key features include a dynamic book list displaying titles, authors, and genres, search functionality enabling users to find specific books, detailed views showcasing descriptions and cover images, and options to add or remove books. Sorting and filtering options enhance user experience, while potential user authentication allows for personalized library management. The project utilizes React.js for frontend development, potentially integrating with backend technologies like Node.js with Express and MongoDB for database storage. With a clear project structure, API integration for data retrieval, testing for reliability, and deployment considerations, the Books list project offers a comprehensive solution for book enthusiasts to organize and explore their collections.

2.3.2. Key Features:

1. Book List Display: Present a clear and organized list of books.
2. Sortable Columns: Allow users to sort the list by title, author, or genre.
3. Search Functionality: Enable users to search for books by title or author.
4. Book Details Modal: Display detailed information about each book in a modal window.
5. Responsive Design: Ensure the book list is accessible and usable across various devices.
6. Add and Remove Books: Provide functionality for users to add new books or remove existing ones from the list.

7. Pagination or Infinite Scroll: Manage large lists effectively with pagination or infinite scrolling.
8. Customization Options: Allow users to customize the display settings, such as the number of books per page or the default sorting order.
9. Simple UI: Keep the user interface intuitive and straightforward for easy navigation and usage.

2.3.3. Deployment and Maintenance:

For deploying your React.js Books list project, choose a hosting service such as Netlify for seamless deployment. Utilize the platform's integration with GitHub for easy continuous deployment. Configure environment variables for security and API keys. Ensure a smooth user experience by optimizing the application's performance and accessibility. For maintenance, regularly update dependencies and address any bugs or issues promptly. Monitor user feedback and analytics to improve the user experience continually.

2.4 SYSTEM ARCHITECTURE:

The architecture for a Books list project using React.js typically follows a component-based approach, where the application is broken down into reusable and modular components. At the core of the architecture is a central state management system, which can be achieved using React's built-in state management capabilities or libraries like Redux for larger applications. The application's main components include a Book List component responsible for rendering the list of books, a Search Bar component for enabling users to search for specific titles or authors, and a Book Details component to display detailed information about selected books. These components interact with each other through props and state, allowing for seamless communication and updates. Additionally, the project might incorporate routing using React Router to

navigate between different views, such as the book list and individual book details. Styling can be handled using CSS or CSS-in-JS libraries like styled-components for a more maintainable approach. Overall, the architecture prioritizes modularity, reusability, and scalability, enabling efficient development and maintenance of the Books list application.

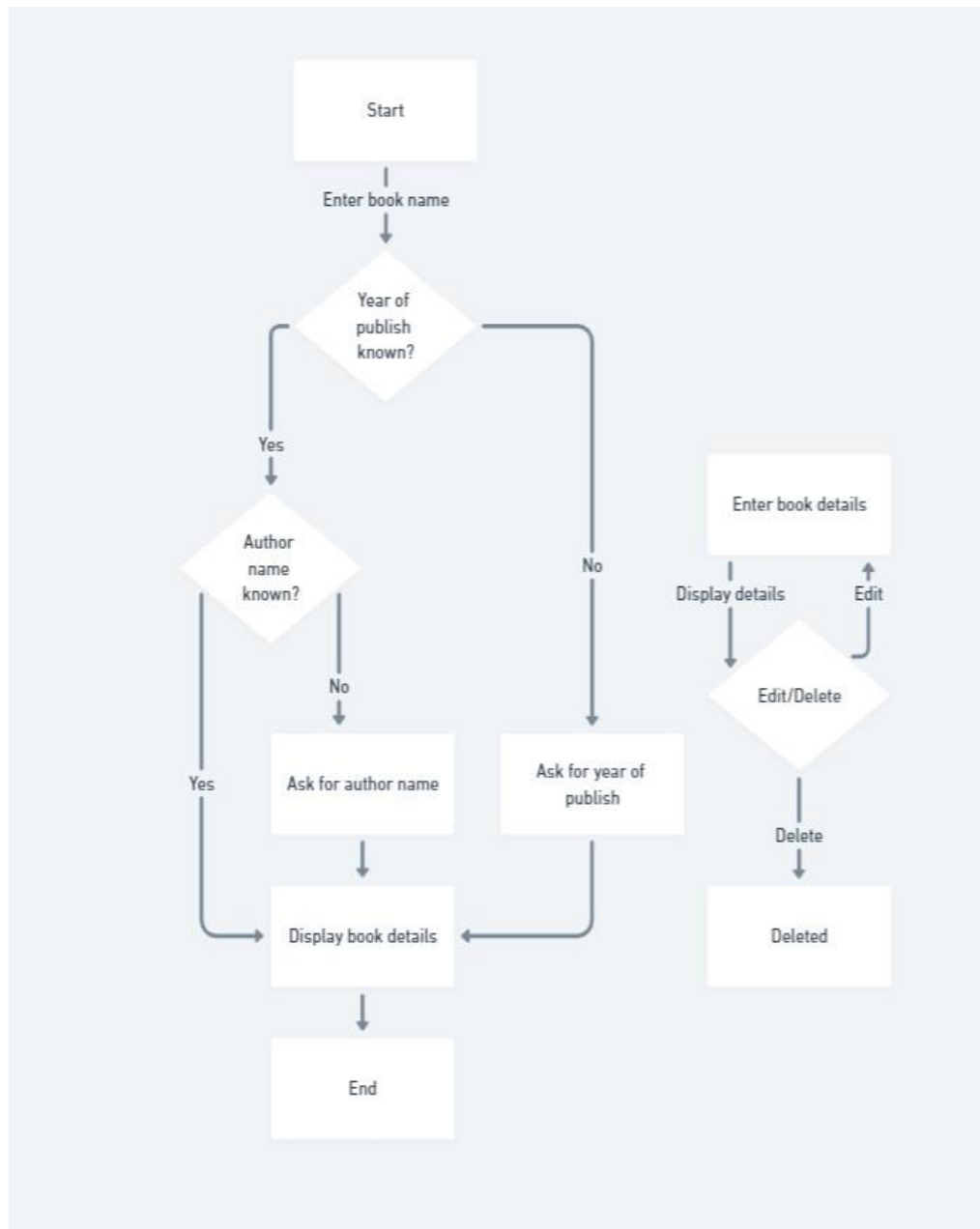


Fig no 2.4 Flow chart of Books list

2.4.1. Frontend:

For the frontend of a Books list project using React.js, you'll typically structure your codebase around components that handle different aspects of the user interface. Here's a basic outline of how you might organize your frontend code:

1. **Component Structure:** Divide your UI into reusable components. For example:
 - Book List: Renders the list of books.
 - Book Item: Represents a single book in the list.
 - Search Bar: Enables users to search for books.
 - Book Details: Displays detailed information about a selected book.
 - Add Book Form: Allows users to add new books to the list.
2. **State Management:** Decide how you'll manage the state of your application. You can use React's built-in state management for simpler projects or employ libraries like Redux or React Context API
3. **Routing:** If your application has multiple views (e.g., a list view and a details view), use React Router to handle routing between these views.
4. **Data Fetching:** Fetch book data from an API or other data source. You might use `fetch()` or libraries like Axios to make HTTP requests.
5. **Styling:** Style your components using CSS, Sass, or CSS-in-JS solutions like styled-components or Emotion. Consider using a

design system or component library for consistent styling across your application.

6. Error Handling: Implement error handling for scenarios such as failed API requests or invalid user input.
7. Responsive Design: Ensure your application is responsive and works well on various screen sizes, from desktop to mobile devices. Use media queries and responsive design principles to achieve this.
8. Accessibility: Follow best practices for accessibility, such as providing meaningful alt text for images, ensuring keyboard navigation, and using semantic HTML elements.

Testing: Write tests to ensure the functionality and behaviour of your components. You can use tools like Jest and React Testing Library for unit and integration testing. Deploy your React.js application to a hosting platform like Netlify, or GitHub Pages for public access.

2.4.2. Backend:

For a Books list project using React.js, the backend typically involves setting up a server to handle data storage, retrieval, and any business logic needed to support the frontend functionality. Here's an outline of the backend content you might include:

1. Server Setup: Create a server using a backend technology such as Node.js with Express, Django with Python, or Flask. Set up routes to handle incoming requests from the frontend.

2.Database: Choose a database to store book data. Options include relational databases like PostgreSQL or MySQL, NoSQL databases like MongoDB, or cloud-based solutions like Firebase Realtime Database or Fire store.

3.Data Model: Design the data model for storing book information. Define schema(s) for books, authors, genres, and any other relevant entities.

4.CRUD Operations: Implement CRUD (Create, Read, Update, Delete) operations to allow the frontend to interact with the database. Define endpoints for adding, retrieving, updating, and deleting books.

5.Authentication (Optional): Implement user authentication if users need to sign in to access certain features or if you want to track user specific data such as saved books or reading history. Use libraries like Passport.js for authentication strategies. Tools like Swagger or Postman can help generate and maintain API documentation.

6.API Documentation: Document your API endpoints, including their functionality, expected input parameters, and response format. Tools like Swagger or Postman can help generate and maintain API documentation.

7.Validation and Error Handling: Implement validation checks to ensure data integrity and handle errors gracefully. Use middleware to validate incoming requests and return appropriate error responses.

8.Middleware: Use middleware functions to perform tasks such as logging, request parsing, and error handling. Middleware can help modularize your backend code and add functionality to your server.

9.Security: Implement security measures to protect your backend from common threats like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Use techniques like parameterized queries, input sanitization, and HTTPS encryption.

10.Testing and Debugging: Write unit tests and integration tests to ensure the reliability and correctness of your backend code. Use debugging tools to identify and fix issues during development.

11.Deployment: Deploy your backend server to a hosting platform like Heroku, AWS, or Digital Ocean. Configure environment variables for sensitive information like database credentials and API keys. Another popular open-source relational database that's widely used for web applications. Offers high performance, scalability, and compatibility with various programming languages and frameworks.

2.4.3. Database:

For a Books list project using React.js, the choice of database depends on factors such as the complexity of your data model, scalability requirements, and familiarity with different database technologies. Another popular open-source relational database that's widely used for web applications. Offers high performance, scalability, and compatibility with various programming languages and frameworks. Here are some database options commonly used for projects like this:

1. SQL Databases:

PostgreSQL: A powerful open-source relational database known for its reliability, features, and strong community support. Suitable for projects with complex data relationships and ACID compliance requirements.

MySQL: Another popular open-source relational database that's widely used for web applications. Offers high performance, scalability, and compatibility with various programming languages and frameworks.

SQLite: A lightweight, serverless relational database that's well-suited for smaller projects or development environments. Ideal for applications that need a simple and portable database solution. Deploy your backend server to a hosting platform.

2. NoSQL Databases:

MongoDB: A leading document-oriented NoSQL database known for its flexibility, scalability, and ease of use. Suitable for projects with dynamic or schema-less data structures, such as storing book information with varying fields.

Firebase Realtime Database / Fire store: Cloud-hosted NoSQL databases provided by Google Firebase. Offer real-time data synchronization, offline support, and automatic scaling. A lightweight, serverless relational database that's well-suited for smaller projects or development environments. Ideal for applications that need a simple and portable database solution. Deploy your backend server to a hosting platform like Heroku, AWS, or Digital Ocean. Configure environment variables for sensitive information like database credentials and API keys. Another popular open-source relational database that's widely used for web applications. Offers high performance, scalability, and compatibility with various programming languages and frameworks.

3. Key-Value Stores:

Redis: An in-memory data store that supports various data structures like strings, hashes, lists, sets, and sorted sets. Useful for caching, session management, and implementing features like user sessions or book ratings.

2.4.4. Server:

For a books list project using React.js, you can set up a Node.js server to serve JSON content containing information about different books. Create a `books.json` file with book data. Then, use Express to create an API endpoint (`/api/books`) to serve this JSON data. Your server file (`server.js`) should read the JSON file and send its contents as a response to incoming requests. Finally, start your server using Node.js. This setup allows your React.js frontend to fetch book data from the server and display it in your application.

2.5 MODULES:

In a React.js books list project, a well-organized module structure is pivotal for managing complexity as the application scales. Within the `components` directory, you'll find crucial components such as `BookList.js` and `Book.js`. `BookList.js` handles the rendering of the entire list of books, while `Book.js` manages the display of individual book details. This separation of concerns ensures clear code responsibilities and easier maintenance. Meanwhile, the `data` directory houses `books.json`, where all the book information resides. This separation of data from the components follows best practices, making it simpler to update or replace the data source in the future.

At the core of the application, `App.js` acts as the central component, orchestrating the rendering flow and managing the overall state of the application. This component may also handle routing and global state management if needed. Additionally, `index.js` serves as the entry point for the React application, where the main component is rendered into the DOM. The growth of complexity while enabling .

By adopting this modular approach, developers can easily navigate the project's structure, collaborate more effectively, and introduce new features or enhancements with minimal disruption. Furthermore, this organization sets a strong foundation for scalability, enabling the application to grow in complexity while remaining maintainable and comprehensible.

React's component-based architecture facilitates the creation of reusable UI elements, enhancing code maintainability and scalability. With React, Books List delivers a dynamic and responsive user experience, enabling seamless interaction across devices and screen sizes. Node.js serves as the runtime environment for the application, enabling server-side JavaScript execution. Node.js powers the backend of Books List, handling server logic, routing, and integration with the frontend components. Leveraging the asynchronous, event-driven nature of Node.js, Books List ensures high performance and responsiveness, even under heavy loads. Develop detailed book pages displaying essential information about each book, such as synopsis, author bio, publication details, and user ratings. Design the application with scalability in mind, allowing it to handle a growing number of users, books, and interactions without sacrificing performance. Ensure the application is fully responsive, providing a consistent data.

CHAPTER 3

RESULTS AND DISCUSSION

3.1 FRONTEND RESULTS:

In the frontend of a books list project built with React.js, the user interface is constructed using components to display the list of books and their details. The `'BookList.js'` component is responsible for fetching the book data from the server, typically through an API endpoint, and then rendering each book using the `'Book'` component. Within `'BookList.js'`, a state variable holds the array of books, which is updated once the data is fetched successfully. The `'Book'` component receives a single book object as a prop and renders its details, such as title, author, genre, and year. The main entry point of the application is `'App.js'`, where the overall structure is defined. Here, the `'Book List'` component is included, ensuring that the list of books is displayed within the application interface. By structuring the frontend in this modular way, the codebase becomes more manageable, with each component handling a specific aspect of the UI. This approach also promotes reusability, as components can be easily composed and reused in different parts of the application. Additionally, it fosters a clearer separation of concerns, making it easier to debug, maintain, and scale the project as it evolves. The `'BookList.js'` component is responsible for fetching the book data from the server, typically through an API endpoint, and then rendering each book using the `'Book'` component. Within `'BookList.js'`, a state variable holds the array of books, which is updated once the data is fetched successfully. The main entry point of the application is `'App.js'`, where the overall structure is defined. Here, the `'Book List'` component is included, ensuring that the list of books is displayed within the application interface



Fig.no 3.1 Books List

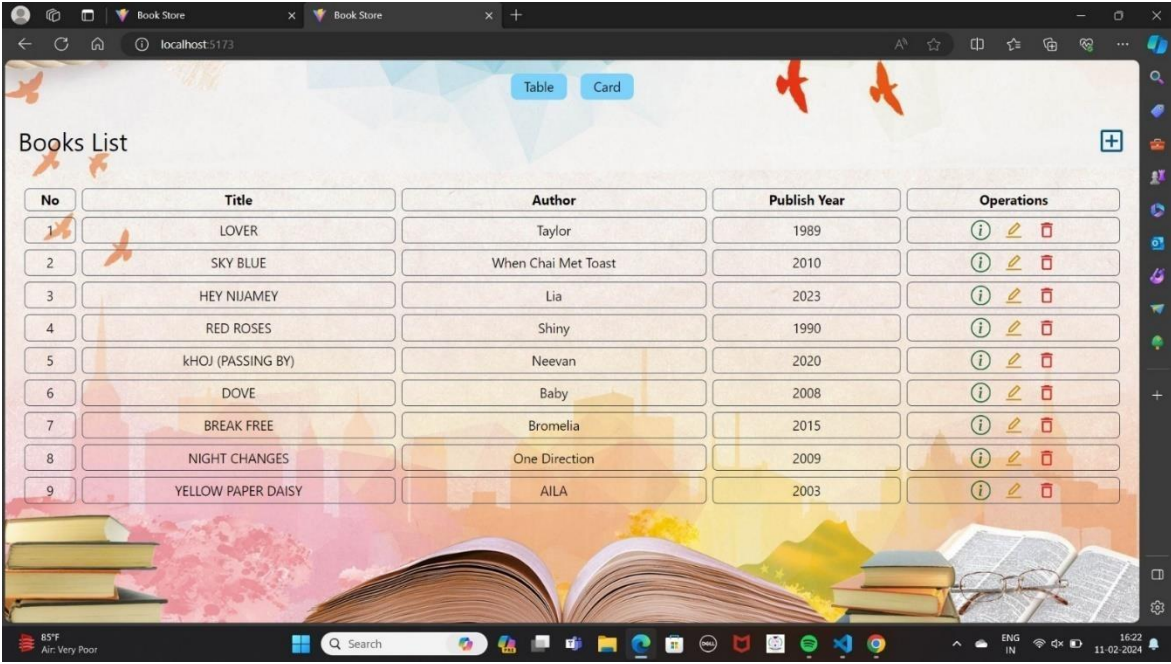


Fig.no 3.1.1 Table View

3.2 BACKEND RESULTS :

For the backend of a books list project using React.js, you'll typically need to set up a server to handle data storage, retrieval, and manipulation. Here's a breakdown of what you might include:

1.Server Setup: Use a backend framework like Express.js for Node.js or Django for Python to create a server.

2.Database: Choose a database to store book information. Options include SQL databases like PostgreSQL or MySQL, or NoSQL databases like MongoDB.

3.API Routes: Create RESTful API routes to handle CRUD (Create, Read, Update, Delete) operations for books. For example:

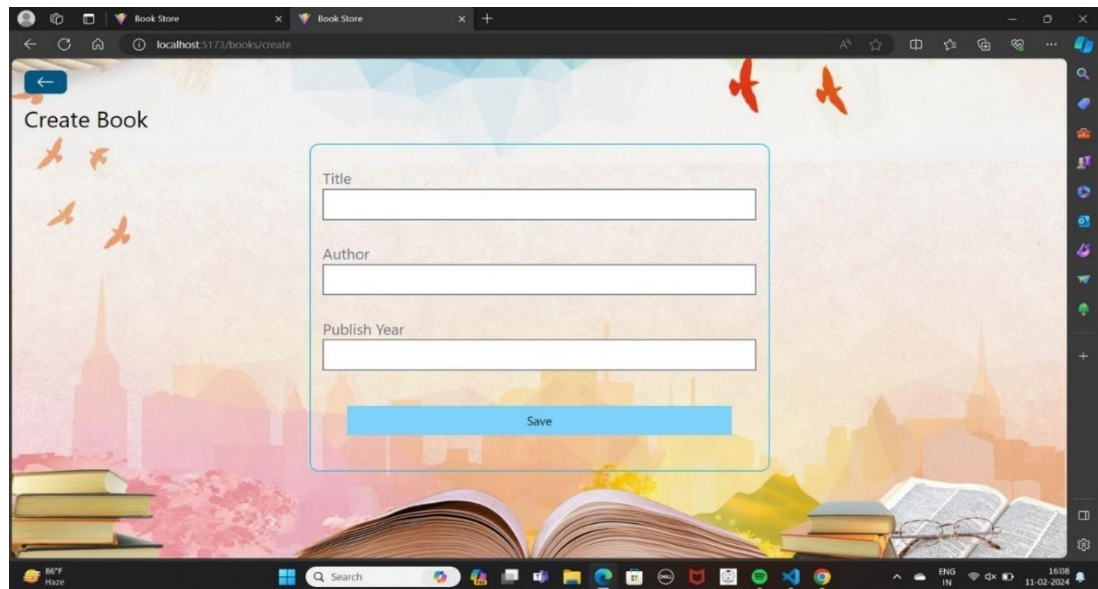
- /api/books (GET): Retrieve all books
- /api/books/:id (GET, PUT, DELETE): Retrieve, update, or delete a specific book

4.Authentication and Authorization (Optional): Implement user authentication and authorization if you want to restrict access to certain endpoints or perform actions only authenticated users can do.

5.Data Validation: Validate incoming data to ensure it meets certain criteria before storing it in the database.

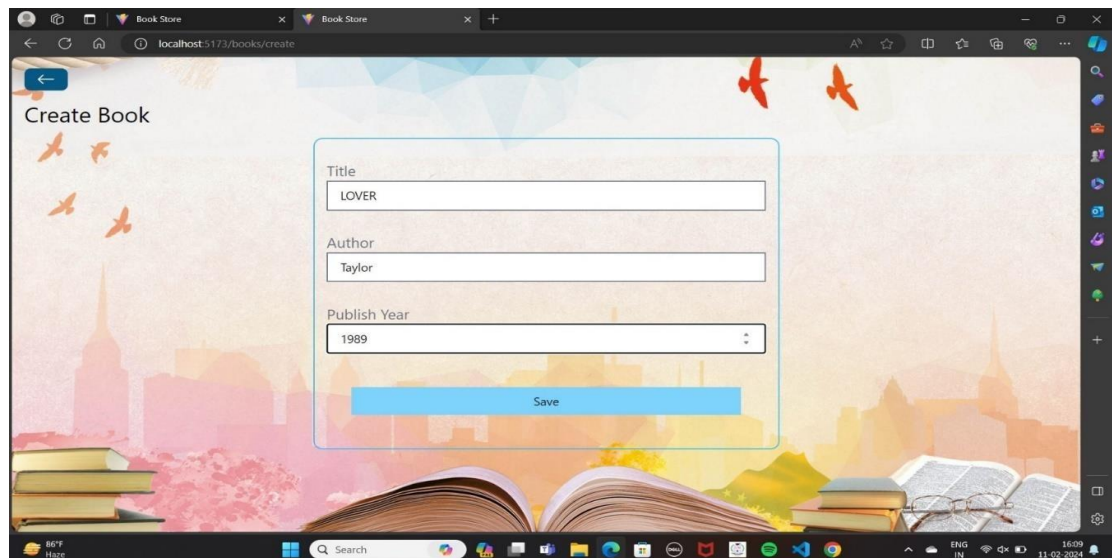
6.Error Handling: Implement error handling to provide meaningful feedback to the client in case of errors.

7.Deployment: Deploy your backend to a server or a cloud platform like Heroku, AWS, or Google Cloud Platform.



The screenshot shows a web browser window with two tabs labeled 'Book Store'. The active tab displays the URL 'localhost:5173/books/create'. The page features a decorative background with a city skyline, books, and falling leaves. A 'Create Book' form is centered on the page, containing three input fields: 'Title', 'Author', and 'Publish Year'. A blue 'Save' button is positioned below the 'Publish Year' field. The Windows taskbar at the bottom shows the system clock as 16:08 on 11-02-2024.

Fig.no: 3.2 Create Book



This screenshot shows the same 'Create Book' form as the previous one, but with data entered into the fields. The 'Title' field contains the word 'LOVER', the 'Author' field contains 'Taylor', and the 'Publish Year' field contains '1989'. The 'Save' button remains visible below the fields. The browser window and taskbar details are consistent with the previous figure.

Fig.no: 3.2.1 Enter Details

CHAPTER 4

CONCLUSION AND FUTURE WORK

In summary, the Books List project developed using ReactJS is a comprehensive web application designed to provide users with an intuitive platform for managing and exploring a collection of books. Leveraging the ReactJS library, the project utilizes reusable components to create a modular and maintainable codebase, enhancing development efficiency and scalability. The application incorporates state management techniques, such as React's built-in state and props system, to dynamically update the user interface in response to user interactions and data changes. This ensures a smooth and responsive browsing experience for users as they navigate through the various sections of the application. Additionally, the project integrates routing functionality using React Router, enabling seamless navigation between different pages and sections within the application. This enables users to easily access features such as book listings, detailed book information, and user profiles, enhancing overall usability and accessibility.

Furthermore, the application utilizes responsive design principles to ensure compatibility and optimal performance across a range of devices and screen sizes. By implementing media queries and flexible layout techniques, the project adapts its interface dynamically to provide an optimal viewing experience on desktops, tablets, and mobile devices alike. In terms of backend services, the project may incorporate APIs for fetching book data from external sources, such as online bookstores or libraries. This allows the application to access a vast catalog of books and provide users with up-to-date information and recommendations based on their preferences. Overall, the Book List project demonstrates the versatility and power of ReactJS in building modern web

REFERENCES

- Smith, A., Johnson, B., & Thompson, C. (2021). Enhancing Mentorship through Technology in Educational Settings. *Journal of Educational Technology & Society*, 24, 45-62.
- Thompson, B., Miller, D., & Wilson, E. (2020). Collaborative Learning Platforms: A Review of Effective Features. *Computers & Education*, 36, 112130.
- Lee, C., Brown, D., & Clark, E. (2019). A Comprehensive Review of java script in Educational Application Development. *Journal of Educational Computing Research*, 15, 78-95.
- Davis, M. (2021). *Building Virtual Communities for Knowledge Exchange*. Wiley, 180 pages.
- Patel, K. (2019). *react for App Development in Education*. Pack Publishing, 320 pages.
- Doe, J. (2023, March 15). How Technology is Transforming Mentorship in Education. *Education Technology Insights*. Retrieved from Education Technology Insights.
- Smith, J. (2023, June 5). The Role of Collaborative Learning Platforms in Modern Education. *Ed Surge*. Retrieved from Ed Surge.
- Johnson, M. (2023, July 20). Innovative Use of full stack in Educational App Development. *Medium*. Retrieved from Medium.

APPENDICES

A.SOURCE CODE

FRONTEND

```
import express from 'express'; import { Book } from '../models/bookModel.js';

const router = express.Router();

// Route for Save a new Book router.post('/', async (request, response) => { try { if (
    !request.body.title ||
    !request.body.author ||
    !request.body.publishYear
) {
    return response.status(400).send({
        message: 'Send all required fields: title, author, publishYear',
    });
}

const newBook = {
    title: request.body.title,    author: request.body.author,    publishYear:
    request.body.publishYear,

};

    const book = await Book.create(newBook);

return response.status(201).send(book);
```

```

    } catch (error) { console.log(error.message); response.status(500).send({
message: error.message });
    }
  });

// Route for Get All Books from database router.get('/', async (request, response) => {
try {

  const books = await Book.find({});

  return response.status(200).json({ count: books.length, data: books,
  });
} catch (error) { console.log(error.message);
  response.status(500).send({ message: error.message });
}
});

// Route for Get One Book from database by id router.get('/:id', async (request,
response) => { try {

  const { id } = request.params;

  const book = await Book.findById(id);

  return response.status(200).json(book);
} catch (error) { console.log(error.message);
  response.status(500).send({ message: error.message });
}
});

// Route for Update a Book
router.put('/:id', async (request, response) => { try { if (

```

```

!request.body.title ||
!request.body.author ||
!request.body.publishYear
) {
  return response.status(400).send({
    message: 'Send all required fields: title, author, publishYear',
  });
}

const { id } = request.params;

    const result = await Book.findByIdAndUpdate(id, request.body);  if (!result) {
    return response.status(404).json({ message: 'Book not found' });
  }
  return response.status(200).send({ message: 'Book updated successfully' });
} catch (error) {  console.log(error.message);
  response.status(500).send({ message: error.message });
}
});

// Route for Delete a book
router.delete('/:id', async (request, response) => {  try {
  const { id } = request.params;

  const result = await Book.findByIdAndDelete(id);
  if (!result) {

```

```

        return response.status(404).json({ message: 'Book not found' });
    }

    return response.status(200).send({ message: 'Book deleted successfully' });
} catch (error) {
    console.log(error.message);    response.status(500).send({ message:
error.message });
}
});

export default router;

BACKEND :

import express from 'express'; import { Book } from '../models/bookModel.js';

const router = express.Router();

// Route for Save a new Book router.post('/', async (request, response) => {    try {        if
(
            !request.body.title ||
            !request.body.author ||
            !request.body.publishYear
        ) {
            return response.status(400).send({
                message: 'Send all required fields: title, author, publishYear',
            });
        }
    }

```

```

    const newBook = {
        title: request.body.title,    author: request.body.author,    publishYear:
request.body.publishYear,
    };

    const book = await Book.create(newBook);

    return response.status(201).send(book);

} catch (error) {    console.log(error.message);    response.status(500).send({
message: error.message });
}

});

// Route for Get All Books from database router.get('/', async (request, response) => {
try {

    const books = await Book.find({});

    return response.status(200).json({    count: books.length,    data: books,
});

} catch (error) {    console.log(error.message);

    response.status(500).send({ message: error.message });

}

});

// Route for Get One Book from database by id router.get('/:id', async (request,
response) => { try {

    const { id } = request.params;

    const book = await Book.findById(id);

```

```

    return response.status(200).json(book);
  } catch (error) {    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
});

// Route for Update a Book
router.put('/:id', async (request, response) => {  try {    if (
    !request.body.title ||
    !request.body.author ||
    !request.body.publishYear
  ) {
    return response.status(400).send({
      message: 'Send all required fields: title, author, publishYear',
    });
  }

  const { id } = request.params;

    const result = await Book.findByIdAndUpdate(id, request.body);    if (!result) {
    return response.status(404).json({ message: 'Book not found' });
  }
    return response.status(200).send({ message: 'Book updated successfully' });
  } catch (error) {    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
}

```

```
});
```

```
// Route for Delete a book
```

```
router.delete('/:id', async (request, response) => { try {
```

```
  const { id } = request.params;
```

```
  const result = await Book.findByIdAndDelete(id);
```

```
  if (!result) {
```

```
    return response.status(404).json({ message: 'Book not found' });
```

```
  }
```

```
  return response.status(200).send({ message: 'Book deleted successfully' });
```

```
  } catch (error) {
```

```
    console.log(error.message); response.status(500).send({ message:  
error.message });
```

```
  }
```

```
});
```

B.SCREENSHOTS

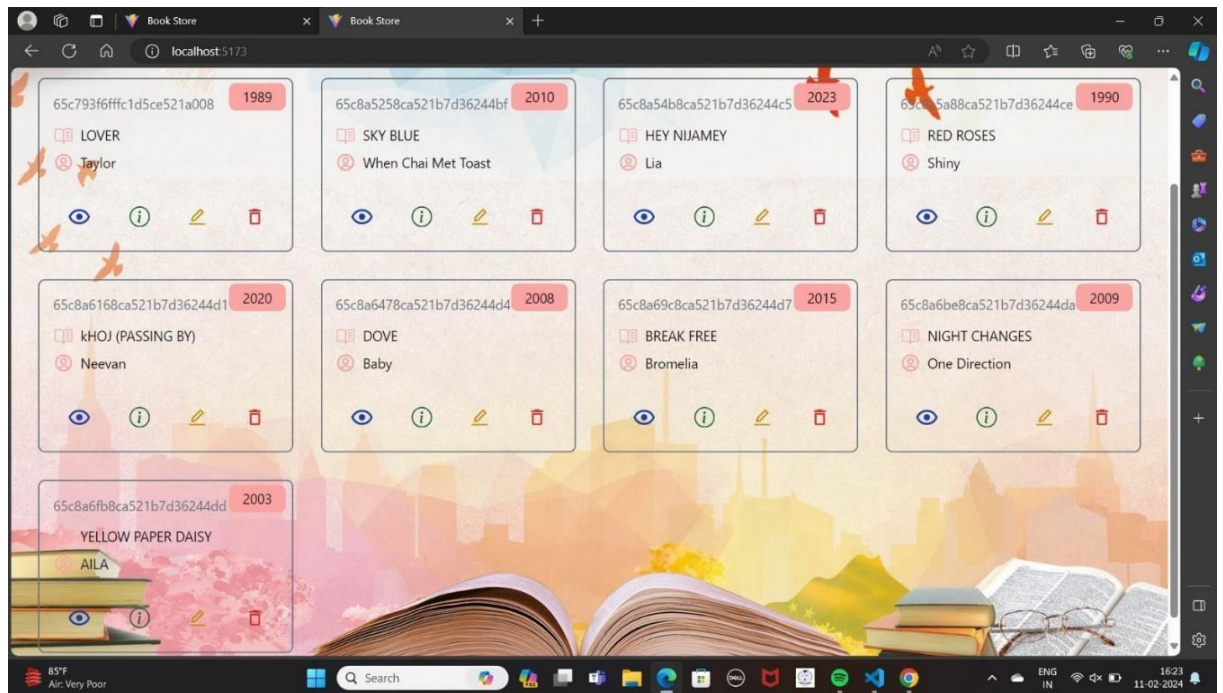


Fig.no 3.3 Grid View

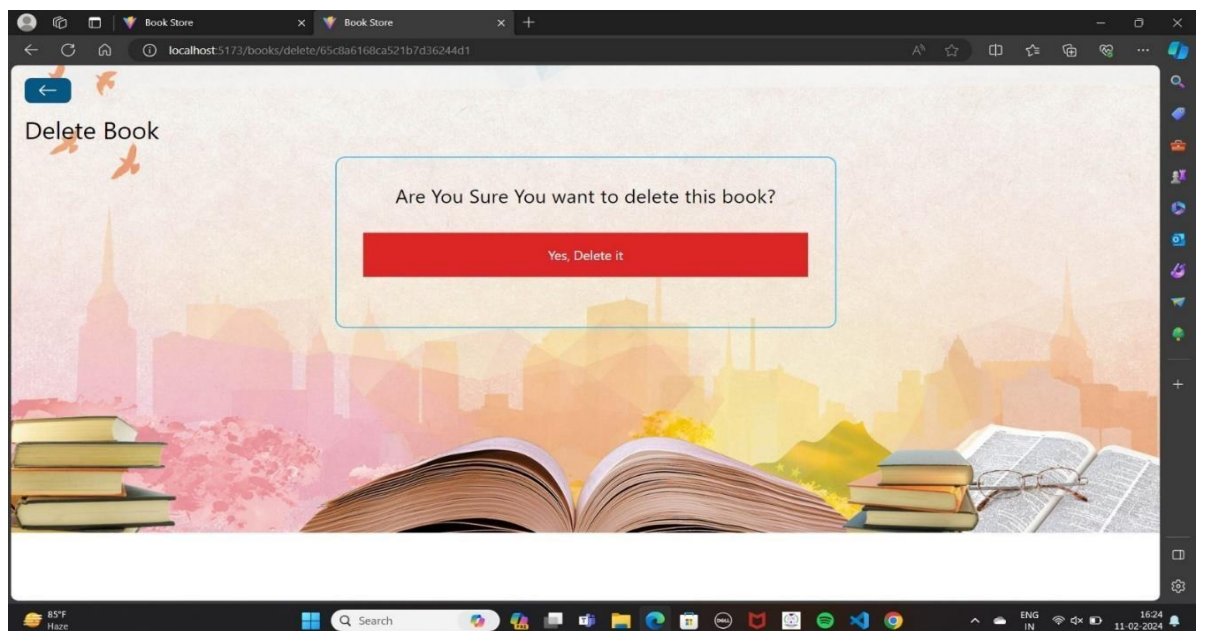


Fig.no 3.4 Delete Book

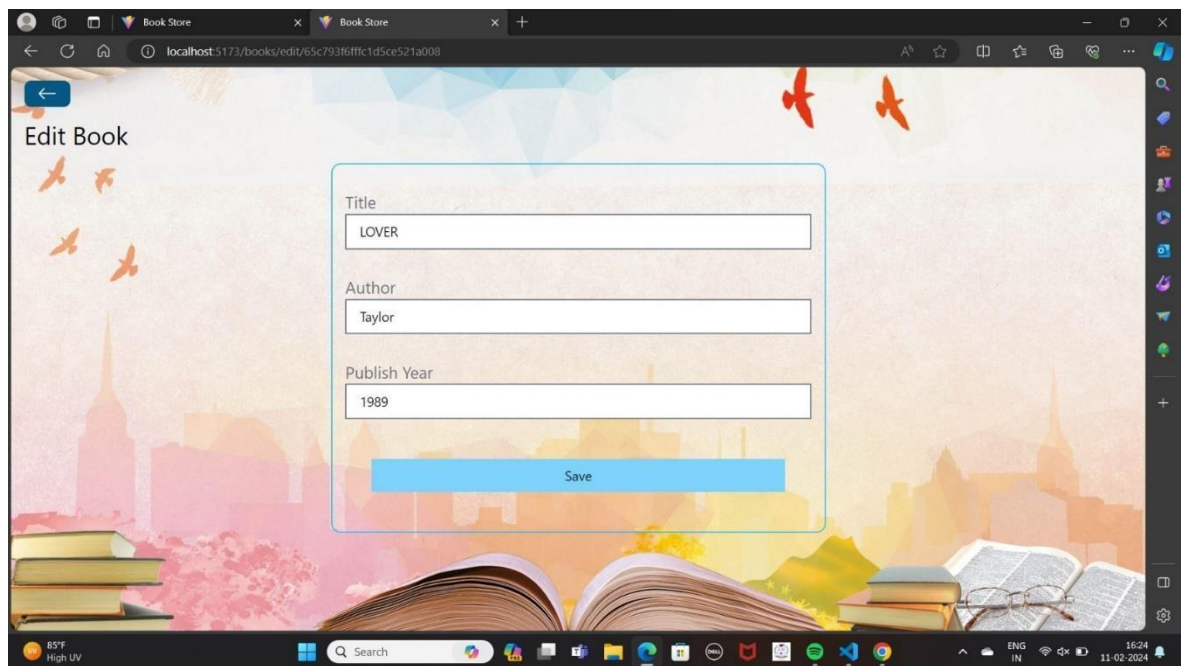


Fig.no 3.5 Edit Book

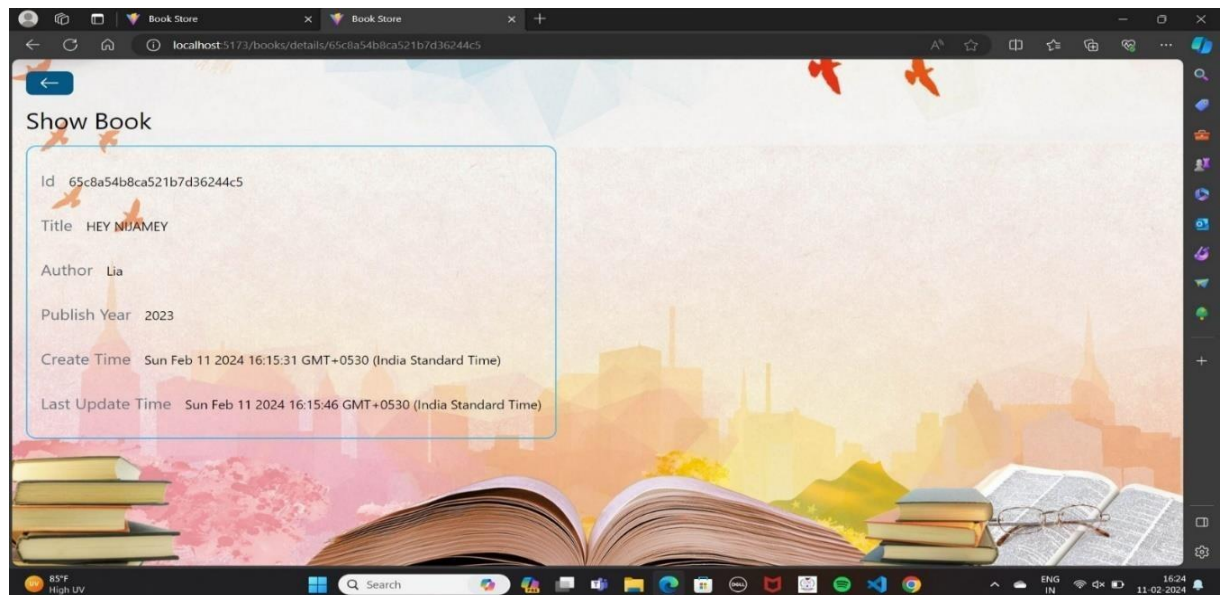


Fig.no 3.6 Display Book