

# **Book Store Web Application using MERN Stack**

## **Smart Internz(Naan Mudhalvan)**

*Project report submitted in fulfilment for the requirement of the degree of*

### **BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY**

by

<i>MUKESH D</i>	<i>211121205018</i>
<i>PRANESH R K</i>	<i>211121205021</i>
<i>SHAMGANESH M</i>	<i>211121205025</i>
<i>THULASIRAJAN T</i>	<i>211121205028</i>



**MADHA ENGINEERING COLLEGE**  
KUNDRATHUR, CHENNAI - 600069

NOV 2024

## **ABSTRACT**

Building a bookstore web application using the MERN stack involves leveraging MongoDB, Express.js, React, and Node.js to create a robust, scalable, and user-friendly platform. MongoDB serves as the database, offering a flexible schema design to store extensive details about books, users and orders. Express.js handles server-side logic, RESTful API creation, and middleware integration, efficiently managing user authentication, authorization, and file uploads. React powers the frontend, providing a dynamic, responsive, and interactive user interface, while utilizing state management tools like Redux or Context API to handle complex application states. Node.js facilitates server-side execution of JavaScript, managing asynchronous operations and ensuring high performance. Key features of the application include secure user authentication and authorization, a comprehensive book catalog with search and filter options, a smooth shopping cart and checkout system, user reviews and ratings, and an admin dashboard for managing users, orders, and inventory. The MERN stack's architecture supports scalability and flexibility, ensuring that the application can grow with changing requirements. Its combination of fast server-side processing and responsive frontend experience enhances user engagement and satisfaction. Comprehensive security measures further protect user data and maintain the integrity of the application, making it a reliable platform for online book buying.

## TABLE OF CONTENTS

Chapter No.	Topic	Page No.
	<b>Abstract</b>	<b>ii</b>
	<b>List of Figures</b>	<b>v</b>
<b>1</b>	<b>Introduction</b>	<b>1</b>
	1.1 Components of the MERN Stack in the Bookstore Application	
<b>2</b>	<b>Pre-Requirement</b>	<b>3</b>
	2.1 Tools and Technologies	
<b>3</b>	<b>Project Overview</b>	<b>4</b>
	3.1 Purpose	
	3.2 Features and Functionalities	
	3.3 Development Workflow	
<b>4</b>	<b>Architecture</b>	<b>5</b>
<b>5</b>	<b>Setup Instruction</b>	<b>6</b>
	5.1 Set Up Your Development Environment:	
	5.2 Initialize Your Project	
	5.3 Set Up the Backend with Express and Node.js	
	5.4 Implement API Endpoints	
<b>6</b>	<b>Configure environment variables</b>	<b>7</b>
	6.1 Frontend	
	6.2 Backend	
	6.3 Implementation	
<b>7</b>	<b>Folder Structure</b>	<b>8</b>
<b>8</b>	<b>Running the Application</b>	<b>9</b>
<b>9</b>	<b>Extension &amp; Packages</b>	<b>10</b>
	9.1 Backend Packages	
	9.2 Frontend Packages	
<b>10</b>	<b>API Documentation</b>	<b>47</b>
	10.1 User Management Endpoints	
	10.2 Order Management Endpoints	

<b>11</b>	<b>Authentication</b>	<b>49</b>
	11.1 User Registration	
	11.2 User Login	
	11.3 Get User Profile	
	11.4 Admin API Steps	
<b>12</b>	<b>Authentication Mechanism</b>	<b>50</b>
<b>13</b>	<b>Authorization</b>	<b>51</b>
	13.1 Key Components	
	13.2 Implementation Overview	
<b>14</b>	<b>Demo Link</b>	<b>52</b>
<b>15</b>	<b>User Interface and Screenshot</b>	<b>52</b>
<b>16</b>	<b>Testing Strategy</b>	<b>59</b>
<b>17</b>	<b>Known Issues</b>	<b>60</b>
<b>18</b>	<b>Future Enhancements</b>	<b>61</b>
<b>19</b>	<b>Conclusion</b>	<b>62</b>
<b>20</b>	<b>Reference</b>	<b>63</b>

## LIST OF FIGURES

<b>Fig</b>	<b>Title of the Figures</b>	<b>Page No.</b>
Fig 4.1	Architecture	5
Fig 7.1	Folder Structure	8
Fig 7.2	Frontend	8
Fig 7.3	Backend	8
Fig 8.1	Frontend Application running	9
Fig 8.2	Backend Application running	9

# 1.INTRODUCTION

In today's digital age, online bookstores have revolutionized the way we discover, purchase, and interact with books. The MERN stack, consisting of MongoDB, Express.js, React.js, and Node.js, is a popular choice for building full-stack web applications, making it an ideal choice for developing a dynamic and scalable online bookstore. This Bookstore application, built using the MERN stack, aims to provide users with a seamless experience to browse, search, and purchase books while also offering features for administrators to manage inventory, users, and orders.

MongoDB serves as the database, providing a flexible, document-oriented data model that simplifies the storage of complex data structures. This is particularly useful for handling the diverse data involved in a bookstore application, such as book details, user profiles, reviews, and purchase histories. MongoDB's scalability and performance capabilities ensure that the application can handle a large volume of data and high traffic loads efficiently.

Express.js, a minimalist web framework for Node.js, facilitates the creation of robust and scalable server-side applications. It simplifies the management of routes, middleware, and server configurations, allowing developers to focus on implementing core functionalities such as user authentication, authorization, and CRUD operations for books and orders. Express.js also seamlessly integrates with MongoDB, enabling efficient data handling and storage.

React, a popular front-end library, is used to build the user interface of the bookstore application. React's component-based architecture allows developers to create reusable UI components, ensuring a consistent look and feel across the application. React's efficient rendering system, powered by the virtual DOM, enhances the user experience by providing fast and responsive interactions. Additionally, state management tools like Redux or the Context API can be employed to handle the application's state, ensuring data consistency and facilitating complex interactions.

Node.js, the runtime environment that executes JavaScript on the server-side, provides the backbone for the MERN stack. It is known for its event-driven, non-blocking I/O model, which makes it ideal for building scalable network applications. Node.js ensures that the server can handle multiple simultaneous requests without compromising performance, which is crucial for an online bookstore that may experience high traffic, especially during peak times.

## **1.1 Components of the MERN Stack in the Bookstore Application:**

### **MongoDB (Database):**

- MongoDB serves as the database where all book details, user information, and transactions are stored. MongoDB's document-based structure allows for flexible data storage, which is essential for storing various book attributes like titles, authors, genres, prices, and availability.
- Additionally, it supports storing user data (like their cart details, order history, and preferences) in a manner that can be easily queried and updated.

### **Express.js (Backend Framework):**

- Express.js is used to build the backend server. It handles the communication between the front-end and the database. Express.js enables the API endpoints for user authentication, book management, and order processing.
- It helps route different HTTP requests (GET, POST, PUT, DELETE) and ensures data is served to the front-end efficiently.

### **React.js (Frontend Library):**

- React.js powers the frontend of the bookstore application, offering a dynamic and responsive user interface. Users can browse books, filter results by categories or authors, add books to the shopping cart, and proceed with checkout.
- With state management libraries like Redux (or React's built-in context API), React ensures the state (e.g., user authentication, cart items) is consistent across different components.

### **Node.js (Server-side JavaScript Runtime):**

- Node.js runs the backend server. As a JavaScript runtime, it allows the application to use JavaScript for both server-side and client-side code, providing consistency across the full stack.
- Node.js is known for its high performance and scalability, which is important when handling multiple user requests simultaneously in an e-commerce application.

## 2.PRE-REQUIREMENT

- **HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.
- **Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.
- **Front-end Library:** Utilize React to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.
- **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.
- **Git:** Download and installation instructions can be found at: <https://git-scm.com/downloads>
- **Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.
  - Visual Studio Code: Download from <https://code.visualstudio.com/download>
  - Sublime Text: Download from <https://www.sublimetext.com/download>
  - WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

### 2.1 Tools and Technologies:

1. **Node.js and npm:** Ensure Node.js and npm (Node Package Manager) are installed on your machine.
2. **MongoDB:** Install MongoDB and set up a local or cloud-based database.
3. **Text Editor/IDE:** Use a code editor like Visual Studio Code, which offers excellent support for JavaScript and React development.
4. **Version Control System:** Set up Git for version control and create a GitHub repository for your project.
5. **Postman or Insomnia:** Use API testing tools like Postman or Insomnia to test your backend endpoints.
6. **Browser Development Tools:** Familiarize yourself with browser developer tools for debugging and testing your frontend.



## **3.PROJECT OVERVIEW**

### **3.1 Purpose:**

A bookstore web application provides an online platform for users to browse, purchase, and read books conveniently. It enhances the book-buying experience with user-friendly navigation, reviews, and recommendations. Additionally, it helps store owners manage inventory and track sales efficiently.

### **3.2 Features and Functionalities:**

#### **1. User Authentication and Authorization:**

- Secure registration and login processes.
- Role-based access control (Admin, Seller, User) to restrict access to specific features.

#### **2. Book Catalog Management:**

- Features to add, update, delete, and view books.
- Advanced search and filter options for users to find books easily.

#### **3. Shopping Cart and Checkout:**

- Users can add books to their cart, manage the cart, and proceed to secure checkout.
- Integration with payment gateways for processing transactions.

#### **4. Order Management:**

- Users can view and track their orders.
- Sellers can manage orders related to their products.

### **3.3 Development Workflow:**

#### **1. Frontend Development:**

- Use React to build the user interface.
- Implement state management with Redux or Context API.
- Design responsive UI components to ensure a seamless user experience across devices.

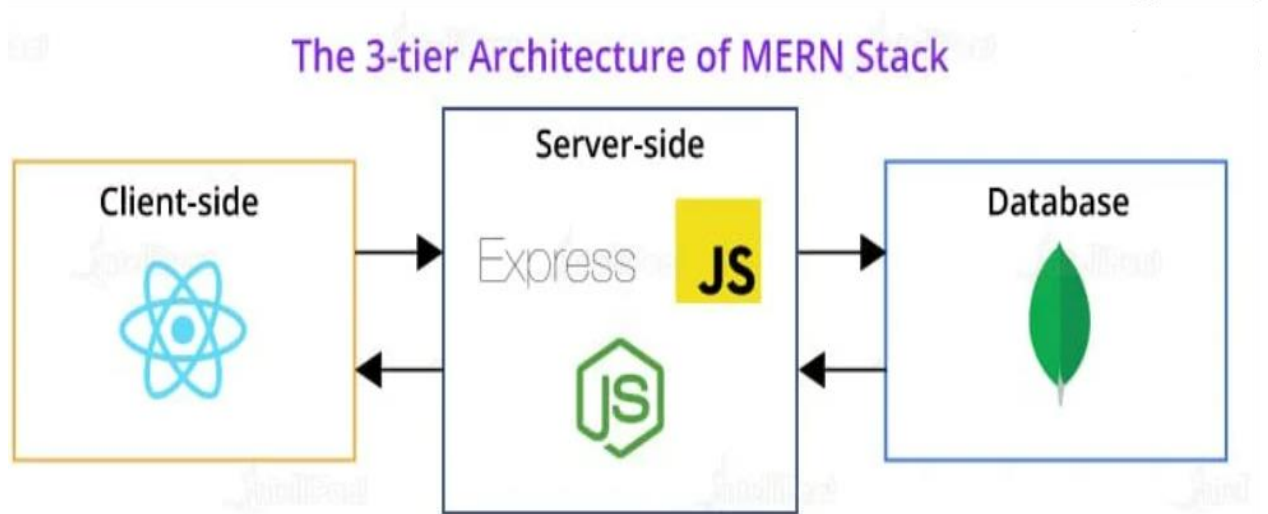
#### **2. Backend Development:**

- Set up the Express.js server to handle API requests.
- Connect the server to MongoDB for data storage.

#### **3. Integration and Testing:**

- Integrate frontend and backend components.
- Perform unit, integration, and end-to-end testing to ensure functionality and reliability.

## 4.ARCHITECTURE



**Fig 4.1 Architecture**

➤ **Frontend:**

The frontend is built using React.js, providing a responsive and dynamic user interface. React components manage the different pages and functions, allowing users to add Wishlist, order, and view dashboard. React, a popular front-end library, is used to build the user interface of the bookstore application.

➤ **Backend:**

Node.js and Express.js power the backend, handling user requests, authentication, Manages business logic, user authentication, and other backend services., and other business logic. The backend follows a RESTful API approach to communicate with the frontend. Express.js also seamlessly integrates with MongoDB, enabling efficient data handling and storage.

➤ **Database:**

MongoDB is used as the database, Stores user information, book details, orders, Wishlist etc. Additionally, it supports storing user data (like their cart details, order history, and preferences) in a manner that can be easily queried and updated. MongoDB's scalability and performance capabilities ensure that the application can handle a large volume of data and high traffic loads efficiently.

## 5.SETUP INSTRUCTION

### 5.1 Set Up Your Development Environment:

1. **Install Node.js and npm:** Ensure you have Node.js and npm (Node Package Manager) installed on your machine.
2. **Install MongoDB:** Set up a local MongoDB database or use a cloud-based service like MongoDB Atlas.

### 5.2 Initialize Your Project:

1. **Create a New Project Directory:** `mkdir bookstore && cd bookstore`
2. **Initialize npm:** Run `npm init -y` to create a package.json file.

### 5.3 Set Up the Backend with Express and Node.js:

1. **Install Backend Dependencies:**
  - Express.js: `npm install express`
  - Mongoose: `npm install mongoose`
  - Cors: `npm install cors`
  - JWT: `npm install jsonwebtoken`
  - bcrypt: `npm install bcryptjs`
  - dotenv: `npm install dotenv`
  - Multer: `npm install multer`
2. **Create Server File:** Create a `server.js` file and set up a basic Express server.
3. **Connect to MongoDB:** Use Mongoose to connect your application to MongoDB.

### 5.4 Implement API Endpoints:

1. **Create Routes Directory:** `mkdir routes`
2. **Define API Routes:** Implement routes for user authentication, book management, order management, etc.
3. **Set Up Middleware:** Implement middleware for authentication and authorization.

## 6.CONFIGURE ENVIRONMENT VARIABLES

- Create a `.env` file in the backend directory with required keys (e.g., database URI, JWT secret).
- Create a file named `.env` in the root directory of your project. This file will store your environment variables. Here, you can define variables like `PORT`, `MONGODB_URI`, `JWT_SECRET`, and any other sensitive or configuration-specific information.

### 6.1 Frontend:

- ``src/``: Contains reusable UI components.
- ``src``: Houses different pages (User, Seller, Admin, Components).
- ``src/app.js``: Manages API calls and handles communication with the backend.

### 6.2 Backend:

- ``db/User``: Holds logic for handling various API requests and response for user components.
- ``db/Seller``: Holds logic for handling various API requests and response for Seller components.
- ``db/Admin``: Holds logic for handling various API requests and response for Admin components.
- ``backend/jsfiles``: Contains Mongoose schemas and models for MongoDB collections.

### 6.3 Implementation:

#### Create a `.env` File:

- Store your environment variables (e.g., database connection string, JWT secret key, server port) in this file.

#### Install `dotenv` Package:

- This package loads environment variables from the `.env` file into the `process.env` object.

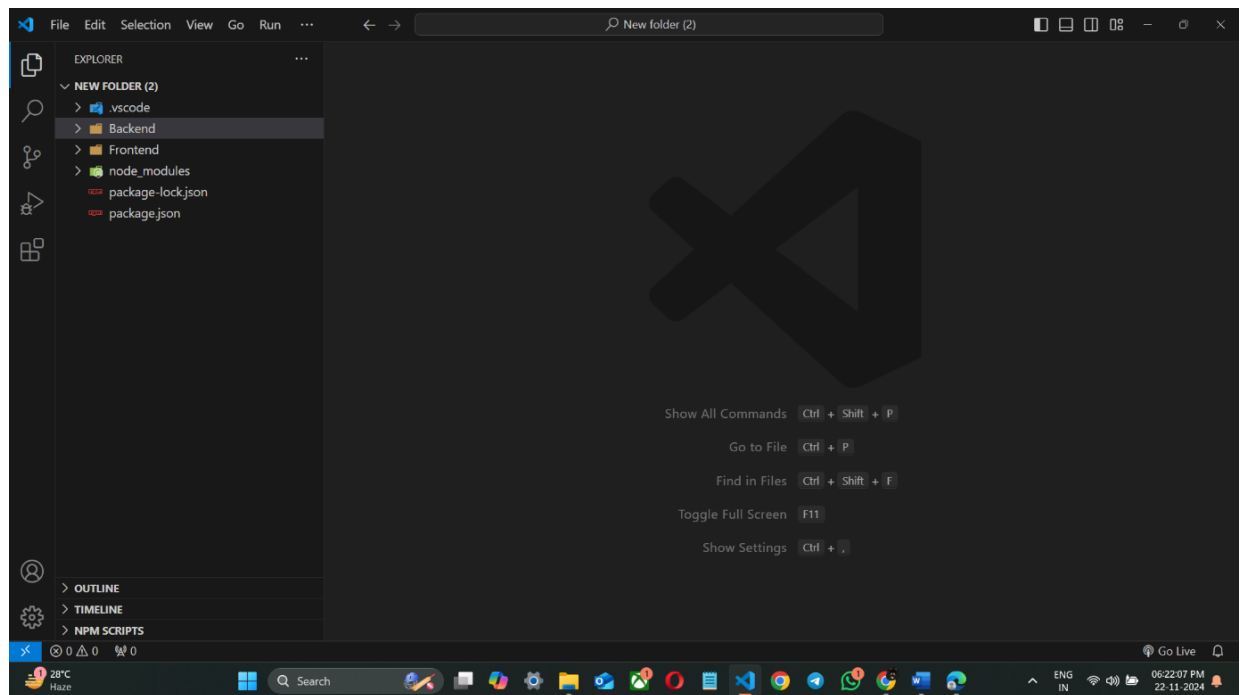
#### Load Environment Variables:

- At the start of your server application, configure it to load the environment variables from the `.env` file.

#### Access Environment Variables:

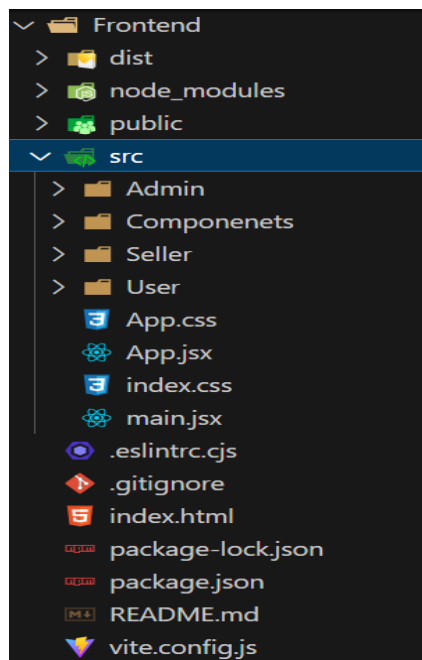
- Use `process.env` to access these variables in your codebase wherever necessary.

## 7.FOLDER STRUCTURE



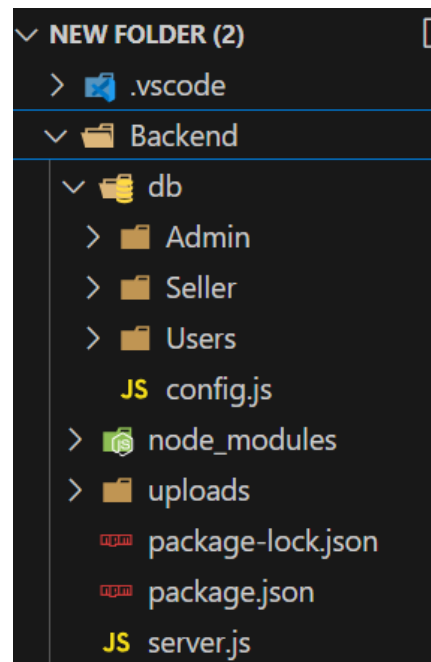
*Fig 7.1 Folder Structure*

### Frontend



*Fig 7.2 Frontend*

### Backend



*Fig 7.3 Backend*

## 8.RUNNING THE APPLICATION

### ➤ Frontend:

- To navigate to frontend use the command ``cd frontend`` in terminal.
- Navigate to the frontend directory and run ``npm run dev`` to start the React application.

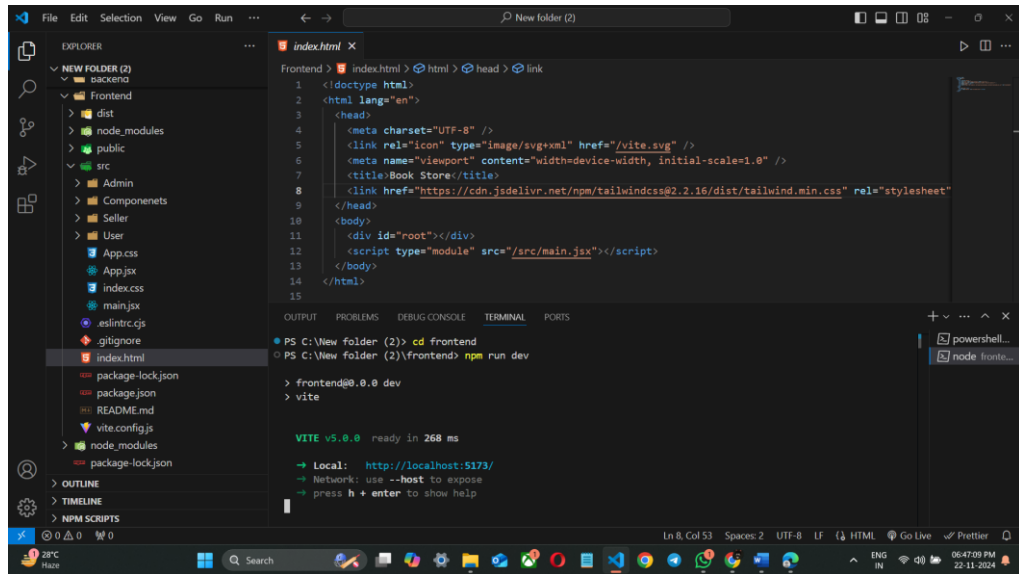


Fig 8.1 Frontend Application running

### ➤ Backend:

- To navigate to backend use the command ``cd backend`` in terminal.
- Navigate to the backend directory and run ``npm start`` to start the Express.js backend.

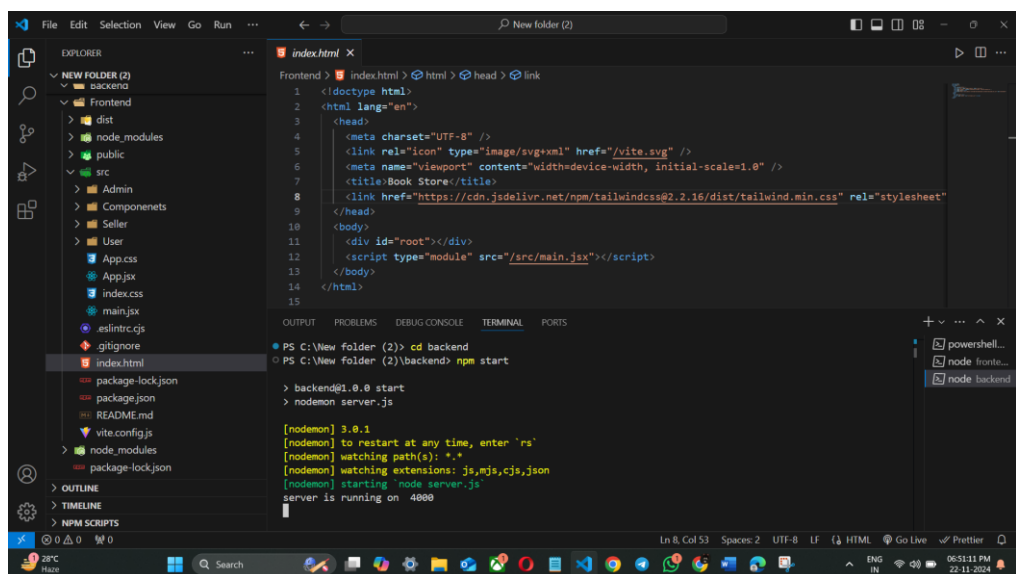


Fig 8.2 Backend Application running

## 9.EXTENSION & PACKAGES

To build a bookstore web application using the MERN stack, you'll need several essential packages and extensions.

For the frontend, use React to build the user interface, React Router for navigation, and Redux for state management. Axios will help with making HTTP requests, while Material-UI provides pre-built UI components. For form handling and validation, Formik and Yup are useful.

For the backend, you'll need Express for the server, Mongoose to connect to MongoDB, and Cors to handle cross-origin requests. JWT is essential for authentication, and bcrypt for password hashing. Use Multer for file uploads and dotenv for managing environment variables.

Additional tools include Nodemon for auto-restarting the server during development, ESLint and Prettier for code quality and formatting, Jest for unit testing, and Docker for containerization. IDE extensions like ESLint, Prettier, GitLens, and Path Intellisense in VSCode can enhance your development experience.

### 9.1 Backend Packages:

#### Package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^8.0.1",
    "multer": "^1.4.5-lts.1",
    "nodeman": "^1.1.2",
    "nodemon": "^3.0.1"
  }
}
```

- This `package.json` file defines the configuration for a Node.js backend project. The project, named "backend," is at version 1.0.0. It specifies the main entry point as `index.js` and includes a script to start the server using `nodemon`, a tool that automatically restarts the server upon changes in the code.
- Dependencies include `cors` for enabling cross-origin resource sharing, `express` for building the web server, `mongoose` for interacting with MongoDB, and `multer` for handling file uploads. Additionally, it lists `nodemon` and `nodeman` (likely a typo or misconfigured dependency) for development purposes.
- The `license` is set to ISC, indicating the open-source license under which the project is distributed. This setup ensures that all necessary packages are installed and the server runs efficiently with automatic restarts during development.

## Package-lock-json

The package-lock.json file is automatically generated when you run npm install in a Node.js project. This file ensures that the exact versions of dependencies are installed, maintaining consistency across different environments. Here's a brief overview of its primary functions:

```
{
  "name": "backend",
  "version": "1.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "backend",
      "version": "1.0.0",
      "license": "ISC",
      "dependencies": {
        "cors": "^2.8.5",
        "express": "^4.18.2",
        "mongoose": "^8.0.1",
        "multer": "^1.4.5-lts.1",
        "nodeman": "^1.1.2",
        "nodemon": "^3.0.1"
      }
    },
    "node_modules/@mongodb-js/saslprep": {
```



```

    "version": "1.1.1", i: integrity": "sha512-
t7c5K033joZZMspnHg/gWPE4kandgc20xE74aY0tGKfgB9VPuVJPix0H6fhmm2erj5PBJ21mqc
x34lpIGtUCsQ==",
    "dependencies": {
      "sparse-bitfield": "^3.0.3"
    }
  },
  "node_modules/@types/node": {
    "version": "20.9.1",
    "resolved": "https://registry.npmjs.org/@types/node/-/node-
20.9.1.tgz",
    "integrity": "sha512-
HhmZzh5LSJNS508jQKpJ/3Zcrr1G6L70hpGqMIAoM9YVD0YBRNWysfwcXq8VnSjlNpCpgLzMXd
iPo+dxcvSmiA==",
    "dependencies": {
      "undici-types": "~5.26.4"
    }
  },
  "node_modules/@types/webidl-conversions": {
    "version": "7.0.3",
    "resolved": "https://registry.npmjs.org/@types/webidl-conversions/-
/webidl-conversions-7.0.3.tgz",
    "integrity": "sha512-
CiJJvcRtIgzadHCYXw7dqEnMNRjhGZlYK05Mj90yktqV8uVT8fD2BF0B7S1uwBE3Kj2Z+4UyPm
Fw/Ixgw/LALA==",
  },
  "node_modules/@types/whatwg-url": {
    "version": "8.2.2",
    "resolved": "https://registry.npmjs.org/@types/whatwg-url/-/whatwg-
url-8.2.2.tgz",
    "integrity": "sha512-
FtQu10RWgn3D9U4aazdwIE2yzphmTJREDqNdODHrbrZmmMqI0vMheC/6NE/J1Yveaj8H+ela+Y
wWTjq5PGmuhA==",
    "dependencies": {
      "@types/node": "*",
      "@types/webidl-conversions": "*"
    }
  },
  "node_modules/abbrev": {
    "version": "1.1.1",
    "resolved": "https://registry.npmjs.org/abbrev/-/abbrev-1.1.1.tgz",
    "integrity": "sha512-
nne9/IiQ/hzIhY6pdDnbBtz7DjPTKrY00P/zvPSm5pOFkl6xuGrGnXn/VtTNNfNtAfZ9/1Rteh
kszu9qcTii0Q=="
  },
  "node_modules/accepts": {
    "version": "1.3.8",

```

```

    "resolved": "https://registry.npmjs.org/accepts/-/accepts-
1.3.8.tgz",
    "integrity": "sha512-
PYAthTa2m2VKxuvSD3DPC/Gy+U+sOA1LAuT8mkmRuvw+NACSaeXEQ+NhcVF7rONl6qcaxV3Uue
mwawk+7+SJLw==",
    "dependencies": {
      "mime-types": "~2.1.34",
      "negotiator": "0.6.3"
    },
    "engines": {
      "node": ">= 0.6"
    }
  },
  "node_modules/anymatch": {
    "version": "3.1.3",
    "resolved": "https://registry.npmjs.org/anymatch/-/anymatch-
3.1.3.tgz",
    "integrity": "sha512-
KMReFUr0B4t+D+OBkjr3KYqvocp2XaSz055UcB6mgQMd3KbcE+mWTyvVV7D/zsdEbNnV6acZUu
tkiHQXvTr1Rw==",
    "dependencies": {
      "normalize-path": "^3.0.0",
      "picomatch": "^2.0.4"
    },
    "engines": {
      "node": ">= 8"
    }
  },
  "node_modules/append-field": {
    "version": "1.0.0",
    "resolved": "https://registry.npmjs.org/append-field/-/append-field-
1.0.0.tgz",
    "integrity": "sha512-
klpgFSWLW1ZEs8svjfb7g4qWY0YS5imI82dTg+QahUvJ8YqAY0P10Uk8tTyh9ZGuYEZEMaeJYC
F5BFuX552hsw=="
  },
  "node_modules/array-flatten": {
    "version": "1.1.1",
    "resolved": "https://registry.npmjs.org/array-flatten/-/array-
flatten-1.1.1.tgz",
    "integrity": "sha512-
PCVAQswWemu6UdxsDFFX/+gVeYqKAod3D3UVm91jHwynguOwAvYPhx8nNlM++NqRcK6CxxpUaf
jmhIdKiHibqg=="
  },
  "node_modules/balanced-match": {
    "version": "1.0.2",
    "resolved": "https://registry.npmjs.org/balanced-match/-/balanced-
match-1.0.2.tgz",

```

```

    "integrity": "sha512-
3oSeU00TMV67hN1AmbXsK4yaqU7tjiH1lbXRdZ0pH0KW9+CeX4bRAaX0Anxt0tx2MrpRpWwQaPw
lIiSEJhYU5Pw==",
  },
  "node_modules/binary-extensions": {
    "version": "2.2.0",
    "resolved": "https://registry.npmjs.org/binary-extensions/-/binary-
extensions-2.2.0.tgz",
    "integrity": "sha512-
jDctJ/IVQbZoJykoeHbhXp0lNBqGNcwXJKJog42E5HDPuWQTSdjCHdihjj0DlnheQ7b1bT6dH0
afNAiS8ooQKA==",
    "engines": {
      "node": ">=8"
    }
  },
  "node_modules/body-parser": {
    "version": "1.20.1",
    "resolved": "https://registry.npmjs.org/body-parser/-/body-parser-
1.20.1.tgz",
    "integrity": "sha512-
jWi7abTbYwajOytWCQc37VulmWiRae5RyTpaCyDcS5/lMdtwSz5lOpDE67srw/HYe35f1z3fDQ
w+3txg7gNtWw==",
    "dependencies": {
      "bytes": "3.1.2",
      "content-type": "~1.0.4",
      "debug": "2.6.9",
      "depd": "2.0.0",
      "destroy": "1.2.0",
      "http-errors": "2.0.0",
      "iconv-lite": "0.4.24",
      "on-finished": "2.4.1",
      "qs": "6.11.0",
      "raw-body": "2.5.1",
      "type-is": "~1.6.18",
      "unpipe": "1.0.0"
    },
    "engines": {
      "node": ">= 0.8",
      "npm": "1.2.8000 || >= 1.4.16"
    }
  },
  "node_modules/brace-expansion": {
    "version": "1.1.11",
    "resolved": "https://registry.npmjs.org/brace-expansion/-/brace-
expansion-1.1.11.tgz",
    "integrity": "sha512-
iCuPHDFgrHX7H2vEI/5xpz07zSHB00TpuqghmYtVmMO6518mCuRMOYf1dEB10g187ufozdaHg
WKcYFb61qGiA==",

```

```

    "dependencies": {
      "balanced-match": "^1.0.0",
      "concat-map": "0.0.1"
    }
  },
  "node_modules/braces": {
    "version": "3.0.2",
    "resolved": "https://registry.npmjs.org/braces/-/braces-3.0.2.tgz",
    "integrity": "sha512-b8um+L1RzM3WDSzvhm6gIz1yfTbBt6YTlcEKAvsmqCZZFw46z6261Vj9j1yEPW33H5H+1BQpZM
P1k8l+78Ha0A==",
    "dependencies": {
      "fill-range": "^7.0.1"
    },
    "engines": {
      "node": ">=8"
    }
  },
  "node_modules/bson": {
    "version": "6.2.0",
    "resolved": "https://registry.npmjs.org/bson/-/bson-6.2.0.tgz",
    "integrity": "sha512-ID1cI+7bazPDyL9wYy9GaQ8gEEohWvcU1/Yf0dIdutJxnmInEEyCsb4awy/OiBfall7zBA179P
ahi3vCdFze3Q==",
    "engines": {
      "node": ">=16.20.1"
    }
  },
  "node_modules/buffer-from": {
    "version": "1.1.2",
    "resolved": "https://registry.npmjs.org/buffer-from/-/buffer-from-1.1.2.tgz",
    "integrity": "sha512-E+XQCRwSbaaiChtv6k6Dwgc+bx+Bs6vuKJHH15kox/BaKbhiXzqQ0wK4c022yElGp20CmjwVhT
3HmxgyPGnJfQ==",
    "dependencies": {
      "node_modules/busboy": {
        "version": "1.6.0",
        "resolved": "https://registry.npmjs.org/busboy/-/busboy-1.6.0.tgz",
        "integrity": "sha512-8SFQbg/0hQ9xy3UNTB0YEnsNBbWfhf7Rtnzpl7TkBiTBRfrQ9Fxcnz7VJsleJpyp6rVLvXiuOR
qjlHi5q+PYuA==",
        "dependencies": {
          "streamsearch": "^1.1.0"
        },
        "engines": {
          "node": ">=10.16.0"
        }
      }
    }
  }
}

```

```

    },
    "node_modules/bytes": {
      "version": "3.1.2",
      "resolved": "https://registry.npmjs.org/bytes/-/bytes-3.1.2.tgz",
      "integrity": "sha512-1Fb59351c1b517110814421b87889088",
      "engines": {
        "node": ">= 0.8"
      }
    },
    "node_modules/call-bind": {
      "version": "1.0.5",
      "resolved": "https://registry.npmjs.org/call-bind/-/call-bind-1.0.5.tgz",
      "integrity": "sha512-C3nQxfFZxFRVoJoGKKI8y3MOEo129NQ+FgQ08iye+Mk4zNZZGdjfs06bVTr+DBSlA66Q2VEcMki/cUCP4SercQ==",
      "dependencies": {
        "function-bind": "^1.1.2",
        "get-intrinsic": "^1.2.1",
        "set-function-length": "^1.1.1"
      },
      "funding": {
        "url": "https://github.com/sponsors/ljharb"
      }
    },
    "node_modules/chokidar": {
      "version": "3.5.3",
      "resolved": "https://registry.npmjs.org/chokidar/-/chokidar-3.5.3.tgz",
      "integrity": "sha512-Dr3sfKRP6oTcjf2JmUmFJfeVMvXBdegxB0iVQ5eb2V10uFJUCAS80ByZdVAyVb8xXNz3GjjTgj9kLWSZTqE6kw==",
      "funding": [
        {
          "type": "individual",
          "url": "https://paulmillr.com/funding/"
        }
      ],
      "dependencies": {
        "anymatch": "~3.1.2",
        "braces": "~3.0.2",
        "glob-parent": "~5.1.2",
        "is-binary-path": "~2.1.0",
        "is-glob": "~4.0.1",
        "normalize-path": "~3.0.0",
        "readdirp": "~3.6.0"
      }
    }
  }

```

```

    },
    "engines": {
      "node": ">= 8.10.0"
    },
    "dependencies": {
      "buffer-from": "^1.0.0",
      "inherits": "^2.0.3",
      "readable-stream": "^2.2.2",
      "typedarray": "^0.0.6"
    }
  },
  "dependencies": {
    "accepts": "~1.3.8",
    "array-flatten": "1.1.1",
    "body-parser": "1.20.1",
    "content-disposition": "0.5.4",
    "content-type": "~1.0.4",
    "cookie": "0.5.0",
    "cookie-signature": "1.0.6",
    "debug": "2.6.9",
    "depd": "2.0.0",
    "encodeurl": "~1.0.2",
    "escape-html": "~1.0.3",
    "etag": "~1.8.1",
    "finalhandler": "1.2.0",
    "fresh": "0.5.2",
    "http-errors": "2.0.0",
    "merge-descriptors": "1.0.1",
    "methods": "~1.1.2",
    "on-finished": "2.4.1",
    "parseurl": "~1.3.3",
    "path-to-regexp": "0.1.7",
    "proxy-addr": "~2.0.7",
    "qs": "6.11.0",
    "range-parser": "~1.2.1",
    "safe-buffer": "5.2.1",
    "send": "0.18.0",
    "serve-static": "1.15.0",
    "setprototypeof": "1.2.0",
    "statuses": "2.0.1",
    "type-is": "~1.6.18",
    "utils-merge": "1.0.1",
    "vary": "~1.1.2"
  },
  "engines": {
    "node": ">= 0.10.0"
  }
}

```

## Node modules

In a Node.js project, the node\_modules directory is crucial because it contains all the dependencies (packages) your project requires to run. These dependencies are specified in the package.json file, and when you run npm install, Node.js downloads and installs these dependencies into the node\_modules folder.

## Server.js

```
const express = require('express')

const PORT = 4000;
const cors = require('cors')
require('./db/config')
const multer = require('multer'); // Import multer
const Admin = require('./db/Admin/Admin')
const users = require('./db/Users/userschema')
const seller = require('./db/Seller/Sellers')
const items = require('./db/Seller/Additem')
const myorders = require('./db/Users/myorders')
const WishlistItem = require('./db/Users/Wishlist')

const app = express()

app.use(express.json())

app.use(cors({
  origin: ["http://localhost:5174"],
  methods: ["POST", "GET", "DELETE", "PUT"],
  credentials: true
}));

const storage = multer.diskStorage({
  destination: 'uploads', // The directory where uploaded files will be
  // stored
  filename: function (req, file, callback) {
    callback(null, Date.now() + '-' + file.originalname); // Set the
    // file name
  },
});

const upload = multer({ storage });
app.use('/uploads', express.static('uploads'));

// Admin //

// Login
app.post('/alogin', (req, resp) => {
  const { email, password } = req.body;
```

```

Admin.findOne({ email: email })
  .then(user => {
    if (user) {
      if (user.password === password) {
        return resp.json({ Status: "Success", user: {
id:user.id,name: user.name, email: user.email } })
      } else {
        resp.json("login fail")
      }
    } else {
      resp.json("no user")
    }
  })
})

// Register Api
app.post('/signup', (req, resp) => {
  const { name, email, password } = req.body;
  Admin.findOne({ email: email })
    .then(use => {
      if (use) {
        resp.json("Already have an account")
      } else {
        Admin.create({ email: email, name: name, password:
password })
          .then(result => resp.json(" Account Created"))
          .catch(err => resp.json(err))
      }
    }).catch(err => resp.json("failed "))
})

app.get('/users', (req, res) => {
  users.find()
    .then((user) => {
      res.status(200).json(user)
    })
    .catch(() => {
      res.sendStatus(500)
    })
})

app.delete('/userdelete/:id', (req, res) => {
  const { id } = req.params
  users.findByIdAndDelete(id)
    .then(() => {
      res.sendStatus(200);
    })
    .catch((error) => {
      res.status(500).json({ error: 'Internal server error' });
    })
})

```



```

    });
  })
  app.delete('/userorderdelete/:id', async (req, res) => {
    const { id } = req.params;
    try {
      await myorders.findByIdAndDelete(id);
      res.sendStatus(200);
    } catch (error) {
      res.status(500).json({ error: 'Internal server error' });
    }
  });
  app.delete('/useritemdelete/:id', async (req, res) => {
    const { id } = req.params;
    try {
      await items.findByIdAndDelete(id);
      res.sendStatus(200);
    } catch (error) {
      res.status(500).json({ error: 'Internal server error' });
    }
  });
  app.get('/sellers', (req, res) => {
    seller.find()
      .then((seller) => {
        res.status(200).json(seller)
      })
      .catch(() => {
        res.sendStatus(500)
      })
  })
  app.delete('/sellerdelete/:id', (req, res) => {
    const { id } = req.params
    seller.findByIdAndDelete(id)
      .then(() => {
        res.sendStatus(200);
      })
      .catch((error) => {
        res.status(500).json({ error: 'Internal server error' });
      });
  })
  app.get('/orders', (req, res) => {
    myorders.find()
      .then((orders) => {
        res.status(200).json(orders)
      })
      .catch(() => {
        res.sendStatus(500)
      })
  })

```

```

    })
  });

// Seller //

// login api
app.post('/slogin', (req, resp) => {
  const { email, password } = req.body;
  seller.findOne({ email: email })
    .then(user => {
      if (user) {
        if (user.password === password) {
          return resp.json({ Status: "Success", user: { id:
user.id, name: user.name, email: user.email } })
        } else {
          resp.json("login fail")
        }
      } else {
        resp.json("no user")
      }
    })
  })

// Register Api
app.post('/ssignup', (req, resp) => {
  const { name, email, password } = req.body;
  seller.findOne({ email: email })
    .then(use => {
      if (use) {
        resp.json("Already have an account")
      } else {
        seller.create({ email: email, name: name, password:
password })
          .then(result => resp.json(" Account Created"))
          .catch(err => resp.json(err))
      }
    })
  }).catch(err => resp.json("failed "))
})

// addBook
app.post('/items', upload.single('itemImage'), async (req, res) => {
  const { title, author, genre, description, price, userId, userName } =
req.body;
  const itemImage = req.file.path; // The path to the uploaded image

  try {
    const item = new items({ itemImage, title, author, genre,
description, price, userId, userName });
    await item.save();
    res.status(201).json(item);
  }
});

```

```

    } catch (err) {
      res.status(400).json({ error: 'Failed to create item' });
    }
  });
//getbooks
app.get('/getitem/:userId', async (req, res) => {
  const userId = req.params.userId;
  try {
    const tasks = await items.find({ userId }).sort('position');
    res.json(tasks);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch tasks' });
  }
});
//delete book
app.delete('/itemdelete/:id', (req, res) => {
  const { id } = req.params;
  items.findByIdAndDelete(id)
    .then(() => {
      res.sendStatus(200);
    })
    .catch((error) => {
      res.status(500).json({ error: 'Internal server error' });
    });
});
//getorders
app.get('/getsellerorders/:userId', async (req, res) => {
  const sellerId = req.params.userId;
  try {
    const tasks = await myorders.find({ sellerId }).sort('position');
    res.json(tasks);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch tasks' });
  }
});

// users //

// login
app.post('/login', (req, res) => {
  const { email, password } = req.body;
  users.findOne({ email: email })
    .then(user => {
      if (user) {
        if (user.password === password) {
          return res.json({ Status: "Success", user: { id:
user.id, name: user.name, email: user.email } });
        }
      }
    })

```

```

        else {
            res.json("Invalid Password")
        }
    }
    else {
        res.json("User not found")
    }
})
})

app.post('/signup', (req, resp) => {
    const { name, email, password } = req.body;
    users.findOne({ email: email })
        .then(use => {
            if (use) {
                resp.json("Already have an account")
            } else {
                users.create({ email: email, name: name, password:
password })
                    .then(result => resp.json(" Account Created"))
                    .catch(err => resp.json(err))
            }
        })
        .catch(err => resp.json("failed "))
})

app.get('/item', async (req, res) => {
    try {
        const images = await items.find();
        res.json(images);
    } catch (error) {
        console.error(error);
        res.status(500).send('Server Error');
    }
});

// Single item
app.get('/item/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const item = await items.findById({ _id: id });
        res.json(item);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
});

app.post('/userorder', async (req, res) => {

```

```

    const { flatno, city, state, pincode, totalamount, seller, sellerId,
BookingDate, description, Delivery, userId, userName: String, booktitle,
bookauthor, bookgenre, itemImage } = req.body;

    try {
      const order = new myorders({ flatno, city, state, pincode,
totalamount, seller, sellerId, BookingDate, description, userId, Delivery,
userName: String, booktitle, bookauthor, bookgenre, itemImage });
      await order.save();
      res.status(201).json(order);
    } catch (err) {
      res.status(400).json({ error: 'Failed to create policy' });
    }
  });

app.get('/getorders/:userId', async (req, res) => {
  const userId = req.params.userId;
  try {
    const tasks = await myorders.find({ userId }).sort('position');
    res.json(tasks);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch tasks' });
  }
});

app.get('/wishlist', async (req, res) => {
  try {
    const wishlistItems = await WishlistItem.find();
    res.json(wishlistItems);
  } catch (error) {
    console.error(error);
    res.status(500).send('Server Error');
  }
});

app.get('/wishlist/:userId', async (req, res) => {
  const userId = req.params.userId;
  try {
    const tasks = await WishlistItem.find({ userId
}).sort('position');
    res.json(tasks);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch tasks' });
  }
});

app.post('/wishlist/add', async (req, res) => {
  const { itemId, title,itemImage,userId,userName } = req.body;

```

```

    try {
      // Check if the item is already in the wishlist

      const existingItem = await WishlistItem.findOne({ itemId });
      if (existingItem) {
        return res.status(400).json({ msg: 'Item already in wishlist'
});
      }
      // Create a new wishlist item
      const newItem = new WishlistItem({
itemId,title,itemImage,userId,userName});
      await newItem.save();

      res.json(newItem);
    } catch (error) {
      console.error(error);
      res.status(500).send('Server Error');
    }
  });
  app.post('/wishlist/remove', async (req, res) => {
    const { itemId } = req.body;

    try {
      // Find and remove the item from the wishlist
      await WishlistItem.findOneAndDelete({ itemId });

      res.json({ msg: 'Item removed from wishlist' });
    } catch (error) {
      console.error(error);
      res.status(500).send('Server Error');
    }
  });
  app.listen(PORT, () => {
    console.log(`server is running on  ${PORT}`)
  });

```

## Admin.js

```

const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  name:String,
  email: String,
  password: String,
  userId:{
    type:mongoose.Schema.Types.ObjectId,
    ref:"admin",

```

```

    }
  })

module.exports =mongoose.model('admin',UserSchema)

```

## Seller.js

```

const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  author: {
    type: String,
    required: true,
  },
  genre: {
    type: String,
    required: true,
  },
  // itemType:String,
  itemImage:String,
  description:String,
  price:String,
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  userName:String,
})

module.exports =mongoose.model('books',bookSchema)


const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({
  name:String,
  email: String,
  password: String,
  userId:{
    type:mongoose.Schema.Types.ObjectId,
    ref:"vendor",
  }
})

module.exports =mongoose.model('Seller',UserSchema)

```

## MyOrder.js

```
const mongoose = require('mongoose');

const bookschema = new mongoose.Schema({
  flatno:String,
  pincode:String,
  city:String,
  state:String,
  totalamount:String,
  seller:String,
  sellerId:String,
  booktitle:String,
  bookauthor:String,
  bookgenre:String,
  itemImage:String,
  description:String,
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  userName:String,
  BookingDate: {
    type: String, // Store dates as strings
    default: () => new Date().toLocaleDateString('hi-IN') // Set the
default value to the current date in "MM/DD/YYYY" format
  },
  Delivery: {
    type: String, // Store dates as strings
    default: () => {
      // Set the default value to the current date + 7 days in
"MM/DD/YYYY" format
      const currentDate = new Date();
      currentDate.setDate(currentDate.getDate() + 7); // Add 7 days
      const day = currentDate.getDate();
      const month = currentDate.getMonth() + 1; // Month is zero-based,
so add 1
      const year = currentDate.getFullYear();
      // Format the date in "MM/DD/YYYY" format
      const formattedDate = `${month}/${day}/${year}`;
      return formattedDate;
    }
  }
})

module.exports =mongoose.model('myorders',bookschema)
```



## Wishlist.js

```
const mongoose = require('mongoose');

const wishlistItemSchema = new mongoose.Schema({
  itemId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  userName:String,
  itemImage:String,
  title:String,
});

module.exports = mongoose.model('WishlistItem', wishlistItemSchema);
```

## MySchema.js

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  name:String,
  email: String,
  password: String,
  userId:{
    type:mongoose.Schema.Types.ObjectId,
    ref:"user",

  }
})

module.exports =mongoose.model('users',UserSchema)
```

## Explanation:

The server runs on port 4000 and employs middleware like `cors` for cross-origin requests and `express.json` for handling JSON payloads. It also configures `multer` for file uploads, storing images in the `uploads` directory. For user authentication, the app provides endpoints for admin and seller login and registration, checking credentials against MongoDB records. It includes routes for managing users, sellers, orders, and items, enabling CRUD operations like fetching lists, adding new entries, and deleting records. Each route handles specific tasks, such as logging in an admin, registering a seller, or uploading book details and images. The code ensures seamless integration with the MongoDB database for data storage and retrieval, aiming to provide a robust backend infrastructure for the bookstore application. Enhancements for better security, such as

password hashing and improved error handling, are advisable to ensure scalability and reliability.

## 9.2 Frontend Packages:

### Package.json

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "axios": "^1.6.2",
    "bootstrap": "^5.3.2",
    "react": "^18.2.0",
    "react-bootstrap": "^2.9.1",
    "react-dom": "^18.2.0",
    "react-icons": "^4.12.0",
    "react-router-dom": "^6.19.0",
    "recharts": "^2.10.1",
    "tailwindcss": "^3.3.5"
  },
  "devDependencies": {
    "@types/react": "^18.2.37",
    "@types/react-dom": "^18.2.15",
    "@vitejs/plugin-react": "^4.2.0",
    "eslint": "^8.53.0",
    "eslint-plugin-react": "^7.33.2",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.4",
    "vite": "^5.0.0"
  }
}
```

```
{
  "dependencies": {
    "axios": "^1.6.2",
    "bootstrap": "^5.3.2",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^8.0.1",
    "multer": "^1.4.5-lts.1",
```

```

    "nodeman": "^1.1.2",
    "nodemon": "^3.0.1",
    "react": "^18.3.1",
    "recharts": "^2.10.1",
    "tailwindcss": "^3.3.5"
  }
}

```

## Package-lock.json

The package-lock.json file in a frontend project serves a similar purpose to that in a backend project. It ensures that the exact versions of the installed dependencies are preserved, providing consistency across different environments. Here's a breakdown of its key functions and what a typical package-lock.json might look like in a frontend project:

```

{
  "name": "frontend",
  "version": "0.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "frontend",
      "version": "0.0.0",
      "dependencies": {
        "axios": "^1.6.2",
        "bootstrap": "^5.3.2",
        "react": "^18.2.0",
        "react-bootstrap": "^2.9.1",
        "react-dom": "^18.2.0",
        "react-icons": "^4.12.0",
        "react-router-dom": "^6.19.0",
        "recharts": "^2.10.1",
        "tailwindcss": "^3.3.5"
      },
      "devDependencies": {
        "@types/react": "^18.2.37",
        "@types/react-dom": "^18.2.15",
        "@vitejs/plugin-react": "^4.2.0",
        "eslint": "^8.53.0",
        "eslint-plugin-react": "^7.33.2",
        "eslint-plugin-react-hooks": "^4.6.0",
        "eslint-plugin-react-refresh": "^0.4.4",
        "vite": "^5.0.0"
      }
    }
  },
}

```

```

    "node_modules/@aashutoshrathi/word-wrap": {
      "version": "1.2.6",
      "resolved": "https://registry.npmjs.org/@aashutoshrathi/word-wrap/-
/word-wrap-1.2.6.tgz",
      "integrity": "sha512-1Yjs2SvM8Tf1ER/OD3c0jhWWOZb58A2t7wpE2S9XfBYTiI1+XFhQG2bjy4Pu1I+EA1CNUzRDYD
dFwFYUKvXcIA==",
      "dev": true,
      "engines": {
        "node": ">=0.10.0"
      }
    },
    "node_modules/@alloc/quick-lru": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@alloc/quick-lru/-/quick-
lru-5.2.0.tgz",
      "integrity": "sha512-UrcABB+4bUrFABwbluTIBErXwvbsU/V7TZWfmbgJfbkwiBuziS9gxdODUyuiecfGQ85jg1MW6
juS3+z5TsKLw==",
      "engines": {
        "node": ">=10"
      },
      "funding": {
        "url": "https://github.com/sponsors/sindresorhus"
      }
    },
    "node_modules/@ampproject/remapping": {
      "version": "2.2.1",
      "resolved": "https://registry.npmjs.org/@ampproject/remapping/-
/remapping-2.2.1.tgz",
      "integrity": "sha512-lFMjJTrFL3j7L9yBxwYfCq2k6qqwHyzuU1/XBnif78PWTJYyL/dfowQHWE3sp6U6ZzqWiiIZnp
TM096zhkjwTg==",
      "dev": true,
      "dependencies": {
        "@jridgewell/gen-mapping": "^0.3.0",
        "@jridgewell/trace-mapping": "^0.3.9"
      },
      "engines": {
        "node": ">=6.0.0"
      }
    },
    "node_modules/@babel/code-frame": {
      "version": "7.22.13",
      "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-
frame-7.22.13.tgz",

```

```

    "integrity": "sha512-
XktuhWlJ5g+3TJXc5upd9Ks1HutSArik6jf2eAjYFYIOF4ej3RN+184cZbzDvbPnuTJJIUhPKKJ
E3cIsYTiAT3w==",
    "dev": true,
    "dependencies": {
      "@babel/highlight": "^7.22.13",
      "chalk": "^2.4.2"
    },
    "engines": {
      "node": ">=6.9.0"
    }
  },
  "node_modules/@babel/compat-data": {
    "version": "7.23.3",
    "resolved": "https://registry.npmjs.org/@babel/compat-data/-/compat-
data-7.23.3.tgz",
    "integrity": "sha512-
BmR4bWbDIOFJmJ9z2cZ8Gmm2MXgEDgjdWgpKmKWUt54UGFJdlj31ECTbaDvCG/qVdG3AQ1SfpZ
Es01lUFbzLOQ==",
    "dev": true,
    "engines": {
      "node": ">=6.9.0"
    }
  },
  "node_modules/@babel/core": {
    "version": "7.23.3",
    "resolved": "https://registry.npmjs.org/@babel/core/-/core-
7.23.3.tgz",
    "integrity": "sha512-
Jg+msLuNuCJDyBvFv5+OKOUjWMZgd85bKjbICd3zWrKAo+bJ49HJufi7CQE0q0uR8NGyO6xkCA
CScNqyjHSZew==",
    "dev": true,
    "dependencies": {
      "@ampproject/remapping": "^2.2.0",
      "@babel/code-frame": "^7.22.13",
      "@babel/generator": "^7.23.3",
      "@babel/helper-compilation-targets": "^7.22.15",
      "@babel/helper-module-transforms": "^7.23.3",
      "@babel/helpers": "^7.23.2",
      "@babel/parser": "^7.23.3",
      "@babel/template": "^7.22.15",
      "@babel/traverse": "^7.23.3",
      "@babel/types": "^7.23.3",
      "convert-source-map": "^2.0.0",
      "debug": "^4.1.0",
      "gensync": "^1.0.0-beta.2",
      "json5": "^2.2.3",
      "semver": "^6.3.1"
    }
  }

```

```

    },
    "engines": {
      "node": ">=6.9.0"
    },
    "funding": {
      "type": "opencollective",
      "url": "https://opencollective.com/babel"
    }
  },
  "node_modules/@babel/generator": {
    "version": "7.23.3",
    "resolved": "https://registry.npmjs.org/@babel/generator/-
/generator-7.23.3.tgz",
    "integrity": "sha512-
keeZWAV4LU3tW0qRi19HRpabC/ilM0HRBBzf9/k8FFiG4KVpiv0FIy4hHfLffQZNhziCTPTmd5
9zoyv6DNISzg==",
    "dev": true,
    "dependencies": {
      "@babel/types": "^7.23.3",
      "@jridgewell/gen-mapping": "^0.3.2",
      "@jridgewell/trace-mapping": "^0.3.17",
      "jsesc": "^2.5.1"
    },
    "engines": {
      "node": ">=6.9.0"
    }
  },
},

```

## Index.html

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
  />
    <title>Book Store</title>
    <link
href="https://cdn.jsdelivrivr.net/npm/tailwindcss@2.2.16/dist/tailwind.min.cs
s" rel="stylesheet">
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>

```

## Main.jsx

```
import React from 'react'

import ReactDOM from 'react-dom/client'
import App from './App.jsx'
// import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

## App.jsx

```
import 'bootstrap/dist/css/bootstrap.min.css';

import Login from './User/Login'
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import Signup from './User/Signup';
import Unavbar from './User/Unavbar';
import Addbook from './Seller/Addbook';
import Item from './Seller/Book';
import Myproducts from './Seller/Myproducts';
import Slogin from './Seller/Slogin';
import Book from './Seller/Book';
import Orders from './Seller/Orders';
import Products from './User/Products';
import Uitem from './User/Uitem';
import Myorders from './User/Myorders';
import Uhome from './User/Uhome';
import OrderItem from './User/OrderItem';
import Shome from './Seller/Shome';
import Ssignup from './Seller/Ssignup';
import Home from './Componenets/Home';
import Alogin from './Admin/Alogin';
import Asignup from './Admin/Asignup';
import Ahome from './Admin/Ahome';
import Users from './Admin/Users';
import Vendors from './Admin/Seller';
import Seller from './Admin/Seller';
import Wishlist from './User/Wishlist';
// import './App.css'
function App() {
  return (
    <div>
      <BrowserRouter>
        <Routes>
```

```

    <Route path="/" element={<Home/>} />

    {/* Admin */}
    <Route path="/login" element={<Alogin/>} />
    <Route path="/signup" element={<Asignup/>} />
    <Route path="/ahome" element={<Ahome/>} />
    <Route path="/users" element={<Users/>} />
    <Route path="/sellers" element={<Seller/>} />

    {/* seller */}
    <Route path="/slogin" element={<Slogin/>} />
    <Route path="/ssignup" element={<Ssignup/>} />
    <Route path="/shome" element={<Shome/>} />
    <Route path="/myproducts" element={<Myproducts/>} />
    <Route path="/addbook" element={<Addbook/>} />
    <Route path="/book/:id" element={<Book/>} />
    <Route path="/orders" element={<Orders/>} />

    {/* user */}
    <Route path="/login" element={<Login/>}/>
    <Route path="/signup" element={<Signup/>} />
    <Route path="/nav" element={<Unavbar/>}/>
    <Route path="/uhome" element={<Uhome/>} />
    <Route path="/uproducts" element={<Products/>} />
    <Route path="/uitem/:id" element={<Uitem/>} />
    <Route path="/orderitem/:id" element={<OrderItem/>} />
    <Route path="/myorders" element={<Myorders />} />
    <Route path="/wishlist" element={<Wishlist />} />

    </Routes>
  </BrowserRouter>
</div>
)
}

export default App

```

## Seller.jsx

```

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Button, Table, Card } from 'react-bootstrap';
import { FaTrash, FaEdit } from 'react-icons/fa';
import { Link } from 'react-router-dom';
import Anavbar from './Anavbar';

```



```

const Seller = () => {
  const [userbookings, setUserbookings] = useState([]);
  const [users, setUsers] = useState([]);

  const [showDetails, setShowDetails] = useState(false);

  const toggleDetails = () => {
    setShowDetails(!showDetails);
  };
  const [showDetail, setShowDetail] = useState(false);

  const toggleDetail = () => {
    setShowDetail(!showDetail);
  };

  useEffect(() => {
    axios.get(`http://localhost:4000/sellers`)
      .then((response) => {
        setUsers(response.data);
        // setLoading(false);
      })
      .catch((error) => {
        // setError('Failed to fetch projects.');
        // setLoading(false);
      });
  }, []);

  const deleteData = (taskId) => {
    axios.delete(`http://localhost:4000/sellerdelete/${taskId}`);
    window.location.assign('/vendors');
    alert('User is deleted');
  };

  const deleteitem = (taskId) => {
    axios.delete(`http://localhost:4000/useritemdelete/${taskId}`);
    window.location.assign('/users');
    alert('deleted');
  };

  const fetchUserBikeData = (userId) => {
    axios.get(`http://localhost:4000/getitem/${userId}`)
      .then((response) => {
        setUserbookings(response.data);
        toggleDetails(); // Show Plan Details when data is fetched
      })
      .catch((error) => {
        console.error('Error fetching user bike data:', error);
      });
  };

  const calculateStatus = (Delivery) => {

```

```

const currentDate = new Date();
const formattedDeliveryDate = new Date(Delivery);

if (formattedDeliveryDate >= currentDate) {
  return "ontheway";
} else {
  return "delivered";
}
};

return (
  <div>
    <Anavbar/>
    <br />
    <h1 className='text-center'>Vendors</h1> <br />
    <div style={{display:"flex",justifyContent:"center"}}>
      <Table striped bordered hover variant="dark" style={{width:"70%"}}>
        <thead>
          <tr>
            <th>sl/no</th>
            <th>UserId</th>
            <th>User name</th>
            <th>Email</th>
            <th>Operation</th>
          </tr>
        </thead>
        <tbody>
          {users.map((item, index) => (
            <tr key={item._id}>
              <td>{index + 1}</td>
              <td>{item._id}</td>
              <td>{item.name}</td>
              <td>{item.email}</td>
              <td>
                <button style={{ border: 'none', background: 'none' }}>
                  <Link to={`/useredit/${item._id}`} style={{ color: 'blue',
textDecoration: 'none' }}>
                    <FaEdit />
                  </Link>
                </button>
                <button onClick={() => deleteData(item._id)} style={{
border: 'none', color: 'red', background: 'none' }}>
                  <FaTrash />
                </button>{' '}
                <Button onClick={() => fetchUserBikeData(item._id)} style={{
marginBottom: '12px' }}>
                  view
                </Button>
              </td>
            </tr>
          ))}
        </tbody>
      </Table>
    </div>
  </div>
);

```

```

<div style={{ display: 'flex' }}>
  {showDetails && (
    <div className="fixed top-0 left-0 w-screen h-screen
flex items-center justify-center z-50" >
      <div className="bg-gray-900 bg-opacity-50 absolute
inset-0"></div>
      <div className="bg-white p-4 rounded-lg z-10 relative"
style={{ maxHeight: "80vh", overflowY: "scroll" }}>
        {/* Rest of your content */}
        <p className="text-sm text-gray-600">
          <div className="container mx-auto mt-8"
style={{width:"1350px"}}>
            <h1 className='text-center text-blue-300'>Vendor
Products</h1>
            {userbookings.map((item) => {
              const status =
calculateStatus(item.Delivery);
              return (
                <Card
                  key={item._id}
                  style={{
                    width: '90%',
                    marginLeft: '65px',
                    backgroundColor: '#fff',
                    boxShadow: '0 2px 4px rgba(0, 0, 0,
0.1)',
                    borderRadius: '8px',
                    padding: '15px',
                    margin: '10px 0',
                  }}
                >
                  <div style={{ display: 'flex',
justifyContent: 'space-around' }}>
                    <div>
                      <img
src={`http://localhost:4000/${item?.itemImage}`} alt={` ${item.itemtype}
Image`} style={{ height: "80px",width:"120px" }} /> <br/>
                    </div>
                    <div>
                      <p>Product Name:</p>
                      <p>{item.itemtype}-{item._id.slice(3,
7)}</p>
                    </div>
                    <div>
                      <p>Orderid:</p>
                      <p>{item._id.slice(0,10)}</p>
                    </div>
                  </div>
                </Card>
              )
            })}
          </div>
        </div>
      </div>
    )
  )
</div>

```

```

        <p>Warranty</p>
        <p>1 year</p>
      </div>
      <div>
        <p>Price</p>
        <p>{item.price}</p>
      </div>
      <button onClick={() =>
deleteitem(item._id)} style={{ border: 'none', color: 'red', background:
'none' }}>
        <FaTrash />
      </button>
    </div>
  </Card>
);
}}
</div>
</p>
<Button onClick={toggleDetails} className="mt-4">
  Close
</Button>
</div>
</div>
))}
</div>
</td>
</tr>
))}
</tbody>
</Table>
</div>
</div>
)
}

export default Seller

```

## User.jsx

```

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Button, Table, Card } from 'react-bootstrap';
import { FaTrash, FaEdit } from 'react-icons/fa';
import { Link } from 'react-router-dom';
import Anavbar from './Anavbar';

const Users = () => {

```

```

const [userbookings, setUserbookings] = useState([]);
const [users, setUsers] = useState([]);

const [showDetails, setShowDetails] = useState(false);

const toggleDetails = () => {
  setShowDetails(!showDetails);
};
const [showDetail, setShowDetail] = useState(false);

const toggleDetail = () => {
  setShowDetail(!showDetail);
};

useEffect(() => {
  axios.get(`http://localhost:4000/users`)
    .then((response) => {
      setUsers(response.data);
      // setLoading(false);
    })
    .catch((error) => {
      // setError('Failed to fetch projects. ');
      // setLoading(false);
    });
}, []);

const deleteData = (taskId) => {
  axios.delete(`http://localhost:4000/userdelete/${taskId}`);
  window.location.assign('/users');
  alert('User is deleted');
};

const deleteorder = (taskId) => {
  axios.delete(`http://localhost:4000/userorderdelete/${taskId}`);
  window.location.assign('/users');
  alert('deleted');
};

const fetchUserBikeData = (userId) => {

  axios.get(`http://localhost:4000/getorders/${userId}`)

  .then((response) => {
    setUserbookings(response.data);
    toggleDetails(); // Show Plan Details when data is fetched
  })
  .catch((error) => {
    console.error('Error fetching user bike data:', error);
  });
};

```

```

const calculateStatus = (Delivery) => {
  const currentDate = new Date();
  const formattedDeliveryDate = new Date(Delivery);

  if (formattedDeliveryDate >= currentDate) {
    return "ontheway";
  } else {
    return "delivered";
  }
};

return (
  <div>
    <Navbar/>
    <br />
    <h1 className='text-center'>Users</h1> <br />
    <div style={{display:"flex",justifyContent:"center"}}>
      <Table striped bordered hover variant="dark" style={{width:"70%"}}>
        <thead>
          <tr>
            <th>sl/no</th>
            <th>UserId</th>
            <th>User name</th>
            <th>Email</th>
            <th>Operation</th>
          </tr>
        </thead>
        <tbody>
          {users.map((item, index) => (
            <tr key={item._id}>
              <td>{index + 1}</td>
              <td>{item._id}</td>
              <td>{item.name}</td>
              <td>{item.email}</td>
              <td>
                <button style={{ border: 'none', background: 'none' }}>
                  <Link to={` /useredit/${item._id}`} style={{ color: 'blue',
textDecoration: 'none' }}>
                    <FaEdit />
                  </Link>
                </button>
                <button onClick={() => deleteData(item._id)} style={{
border: 'none', color: 'red', background: 'none' }}>
                  <FaTrash />
                </button>{' '}
                <Button onClick={() => fetchUserBikeData(item._id)} style={{
marginBottom: '12px' }}>
                  view
                </Button>
              </td>
            </tr>
          ))}
        </tbody>
      </Table>
    </div>
  </div>
);

```

```

<div style={{ display: 'flex' }}>
  {showDetails && (
    <div className="fixed top-0 left-0 w-screen h-screen
flex items-center justify-center z-50" >
      <div className="bg-gray-900 bg-opacity-50 absolute
inset-0"></div>
      <div className="bg-white p-4 rounded-lg z-10 relative"
style={{ maxHeight: "80vh", overflowY: "scroll" }}>
        {/* Rest of your content */}
        <p className="text-sm text-gray-600">
          <div className="container mx-auto mt-8"
style={{width:"1350px"}}>
            <h1 className='text-center text-blue-300'>User
Orders</h1>
            {userbookings.map((item) => {
              const status =
calculateStatus(item.Delivery);
              return (
                <Card
                  key={item._id}
                  style={{
                    width: '90%',
                    marginLeft: '65px',
                    backgroundColor: '#fff',
                    boxShadow: '0 2px 4px rgba(0, 0, 0,
0.1)',
                    borderRadius: '8px',
                    padding: '15px',
                    margin: '10px 0',
                  }}
                >
                  <div style={{ display: 'flex',
justifyContent: 'space-around' }}>
                    <div>
                      <img
src={`http://localhost:4000/${item?.itemImage}`} alt={` ${item.itemtype}
Image`} style={{ height: "80px" }} />
                    </div>
                    <div>
                      <p>Product Name:</p>
                      <p>{item.itemname}-{item._id.slice(3,
7)}</p>
                    </div>
                    <div>
                      <p>Orderid:</p>
                      <p>{item._id.slice(0,10)}</p>
                    </div>
                  </div>
                </div>
              )
            })}

```

```

        <p>Address:</p>
        {item.flatno},<br
/>{item.city},{item.pincode}),<br />{item.state}.
    </div>
    <div>
        <p>Buyer</p>
        <p>{item.userName}</p>
    </div>
    <div>
        <p>Seller</p>
        <p>{item.seller}</p>
    </div>
    <div>
        <p>BookingDate</p>
        <p>{item.BookingDate}</p>
    </div>
    <div>
        <p>Delivery By</p>
        <p>{item.Delivery}</p>
    </div>
    <div>
        <p>Warranty</p>
        <p>1 year</p>
    </div>
    <div>
        <p>Price</p>
        <p>{item.totalamount}</p>
    </div>
    <div>
        <p>Status</p>
        <p>{status}</p>
    </div>
    <button onClick={() =>
deleteorder(item._id)} style={{ border: 'none', color: 'red', background:
'none' }}>
        <FaTrash />
    </button>
    </div>
</Card>
);
}}
</div>
</p>
<Button onClick={toggleDetails} className="mt-4">
    Close
</Button>
</div>
</div>

```



```

        ))
      </div>
    </td>
  </tr>
</tbody>
</Table>
</div>
</div>
)
}
export default Users

```

## Home.jsx

```

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { BarChart, Bar, XAxis, YAxis, Tooltip, Legend } from 'recharts';
import { Card } from 'react-bootstrap';
import { Link } from 'react-router-dom';
import Anavbar from './Anavbar';

function Ahome() {
  const [users, setUsers] = useState([]);
  const [vendors, setVendors] = useState([]);
  const [items, setItems] = useState([]);
  const [orders, setOrders] = useState([]);

  useEffect(() => {
    // Fetch user data
    axios.get(`http://localhost:4000/users`)
      .then((response) => {
        setUsers(response.data);
      })
      .catch((error) => {
        console.error('Error fetching users: ', error);
      });

    // Fetch vendors data
    axios.get(`http://localhost:4000/sellers`)
      .then((response) => {
        setVendors(response.data);
      })
      .catch((error) => {
        console.error('Error fetching bookings: ', error);
      });

    // Fetch items data

```

```

    axios.get(`http://localhost:4000/item`)
    .then((response) => {
        setItems(response.data);
    })
    .catch((error) => {
        console.error('Error fetching bookings: ', error);
    });

    // Fetch orders data
    axios.get(`http://localhost:4000/orders`)
    .then((response) => {
        setOrders(response.data);
    })
    .catch((error) => {
        console.error('Error fetching bookings: ', error);
    });
}, []);

const colors = ['#2B124C', '#AE4451', '#F39F5A', 'orange'];

// Calculate the number of users and bookings
const totalUsers = users.length;
const totalvendors = vendors.length;
const totalItems = items.length;
const totalOrders = orders.length;

// Define data for the bar chart
const data = [
    { name: 'Users', value: totalUsers, fill: '#2B124C' },
    { name: 'vendors', value: totalvendors, fill: 'cyan' },
    { name: 'Items', value: totalItems, fill: 'blue' },
    { name: 'Orders', value: totalOrders, fill: 'orange' },
];
return (
    <div>
        <Anavbar/>
        <h3 className="text-center" style={{color:""}}>DashBoard</h3>
        <Card body style={{ background: "white", width: "80%", marginLeft:
"10%", marginTop: "20px", height: "580px" }}>
            <div className="flex justify-around items-center p-4">
                <Link to="/users" style={{textDecoration:"none"}}>
                    <div className="w-64 h-32 bg-red-500 rounded-lg shadow-md flex
flex-col justify-center items-center text-xl font-bold text-gray-800 text-
center">
                        USERS <br /> <br />{totalUsers}
                    </div>
                </Link>
                <Link to="/vendors" style={{textDecoration:"none"}}>

```

```

        <div className="w-64 h-32 bg-blue-500 rounded-lg shadow-md flex
flex-col justify-center items-center text-xl font-bold text-gray-800 text-
center">
            Vendors <br /> <br /> {totalvendors}
        </div>
    </Link>
    <Link to="/vendors" style={{textDecoration:"none"}}>
        <div className="w-64 h-32 bg-green-500 rounded-lg shadow-md flex
flex-col justify-center items-center text-xl font-bold text-gray-800 text-
center">
            Items <br /> <br />{totalItems}
        </div>
    </Link>
    <Link to="/users" style={{textDecoration:"none"}}>
        <div className="w-64 h-32 bg-yellow-500 rounded-lg shadow-md
flex flex-col justify-center items-center text-xl font-bold text-gray-800
text-center">
            Total Orders <br /> <br />{totalOrders}
        </div>
    </Link>
</div>
<br/>
<br/>
<br/>
<div style={{paddingLeft:"350px"}}>
<BarChart width={400} height={300} data={data} >
    <XAxis dataKey="name" />
    <YAxis />
    <Tooltip />
    <Legend />
    <Bar dataKey="value" fill="#8884d8" barSize={50} />

</BarChart>
</div>
</Card>

</div>
);
}
export default Ahome;

```

## Node Modules

In a frontend project, the `node_modules` directory is just as essential as in a backend project, although it serves a slightly different purpose. It houses all the dependencies that the frontend application requires to function. These dependencies are typically installed using a package manager like npm or Yarn, as specified in the `package.json` file.

## 10.API DOCUMENTATION

### Add a Book:

- **Method:** POST /books
- **Description:** Adds a new book to the inventory.

### View All Books:

- **Method:** GET /books
- **Description:** Retrieves a list of all books in the inventory.

### Update a Book:

- **Method:** PUT /books/{id}
- **Description:** Updates the details of a specific book.

### Delete a Book:

- **Method:** DELETE /books/{id}
- **Description:** Removes a book from the inventory.

## 10.1 User Management Endpoints

### 1. Register a User:

- **Method:** POST /users/register
- **Description:** Registers a new user.

### 2. Login a User:

- **Method:** POST /users/login
- **Description:** Authenticates a user and generates a token.

### 3. Get User Details:

- **Method:** GET /users/{id}
- **Description:** Retrieves details of a specific user by ID.

#### 4. Update User Details:

- **Method:** PUT /users/{id}
- **Description:** Updates details of a specific user.

#### 5. Delete a User:

- **Method:** DELETE /users/{id}
- **Description:** Removes a user from the system.

### 10.2 Order Management Endpoints

#### 1. Create an Order:

- **Method:** POST /orders
- **Description:** Creates a new order for a user.

#### 2. View All Orders:

- **Method:** GET /orders
- **Description:** Retrieves a list of all orders.

#### 3. View a Single Order:

- **Method:** GET /orders/{id}
- **Description:** Retrieves details of a specific order by its ID.

#### 4. Update an Order:

- **Method:** PUT /orders/{id}
- **Description:** Updates the details of a specific order.

#### 5. Delete an Order:

- **Method:** DELETE /orders/{id}
- **Description:** Removes an order from the system.

## **11.AUTHENTICATION**

### **11.1 User Registration:**

- User sends a request to register with their details (name, email, password, etc.).
- The system validates the input data.
- If validation passes, a new user account is created.
- The system responds with a success message and user ID.

### **11.2 User Login:**

- User sends login credentials (email and password).
- The system verifies the credentials.
- If valid, a token (JWT or similar) is generated and returned.
- User uses the token for authenticated requests.

### **11.3 Get User Profile:**

- User sends a request with the authentication token.
- The system verifies the token and retrieves user details.
- The system returns the user's profile information.

### **11.4 Admin API Steps:**

#### **1.Admin Login:**

- Admin provides credentials to log in.
- The system verifies the credentials.
- If valid, a token is issued for admin-level access.

#### **2. View All Users:**

- Admin sends a request with their authentication token.
- The system verifies the token and fetches all user records.
- The system returns the list of users.

#### **3. Delete a User:**

- Admin sends a request to delete a specific user, providing the user ID.
- The system verifies the admin token and checks if the user exists.
- If valid, the user is deleted, and the system responds with a confirmation.

## 12. AUTHENTICATION MECHANISM

Authentication is managed using JSON Web Tokens (JWT). When a user logs in, a JWT is generated and sent to the frontend to be stored locally (typically in cookies or local Storage).

### User Registration:

- **Endpoint:** `POST /register`
- **Process:** Users provide their details (username, email, password) to create an account.
- **Data Handling:** Passwords are hashed before being stored in the database to enhance security.

### User Login:

- **Endpoint:** `POST /login`
- **Process:** Users submit their email and password.
- **Verification:** The server verifies the email exists and the password matches the hashed version in the database.
- **Token Generation:** On successful login, a JWT (JSON Web Token) is generated and sent back to the user.

### Token-Based Authentication:

- **JWT:** The token includes user information and a secret key, ensuring secure verification.
- **Storage:** The JWT is stored in the client's local storage or cookies.
- **Authorization Header:** For subsequent requests, the JWT is included in the `Authorization` header as `Bearer <token>`.

### Middleware Validation:

- **Token Validation:** Middleware intercepts requests to protected routes, verifying the token's validity.
- **Access Control:** If the token is valid, the request proceeds; otherwise, access is denied.

### Database Setup:

- Use MongoDB to create a database for storing user credentials and related information.
- Design a schema that includes fields like username, email, and password using Mongoose.

## 13.AUTHORIZATION

Authorization is the process that determines what actions a user can perform and what resources they can access within the application. It's different from authentication, which is about verifying the user's identity.

### 13.1 Key Components:

#### 1. User Roles:

- **Admin:** Has full control over the application, including managing users, books, orders, and system settings.
- **Seller:** Can manage their own inventory, add new books, and handle orders related to their products.
- **User (Customer):** Can browse books, add items to the cart, make purchases, view order history, and leave reviews and ratings.

#### 2. Permissions:

- Each role is assigned specific permissions that define what they can do within the application. For example:
  - **Admin:** Can add, edit, or delete any book, user, or order.
  - **Seller:** Can add, edit, or delete only their books and manage orders related to their products.
  - **User:** Can browse books, make purchases, and manage their own orders and reviews.

### 13.2 Implementation Overview:

#### 1. Defining Roles and Permissions:

- Clearly outline the different roles (Admin, Seller, User) and their respective permissions.
- Create a mapping of roles to their allowed actions, ensuring that each role can only access what they are permitted to.

#### 2. Assigning Roles to Users:

- When a user registers or is created, assign them a role based on their function within the application.
- Store this role information securely in the user's profile.



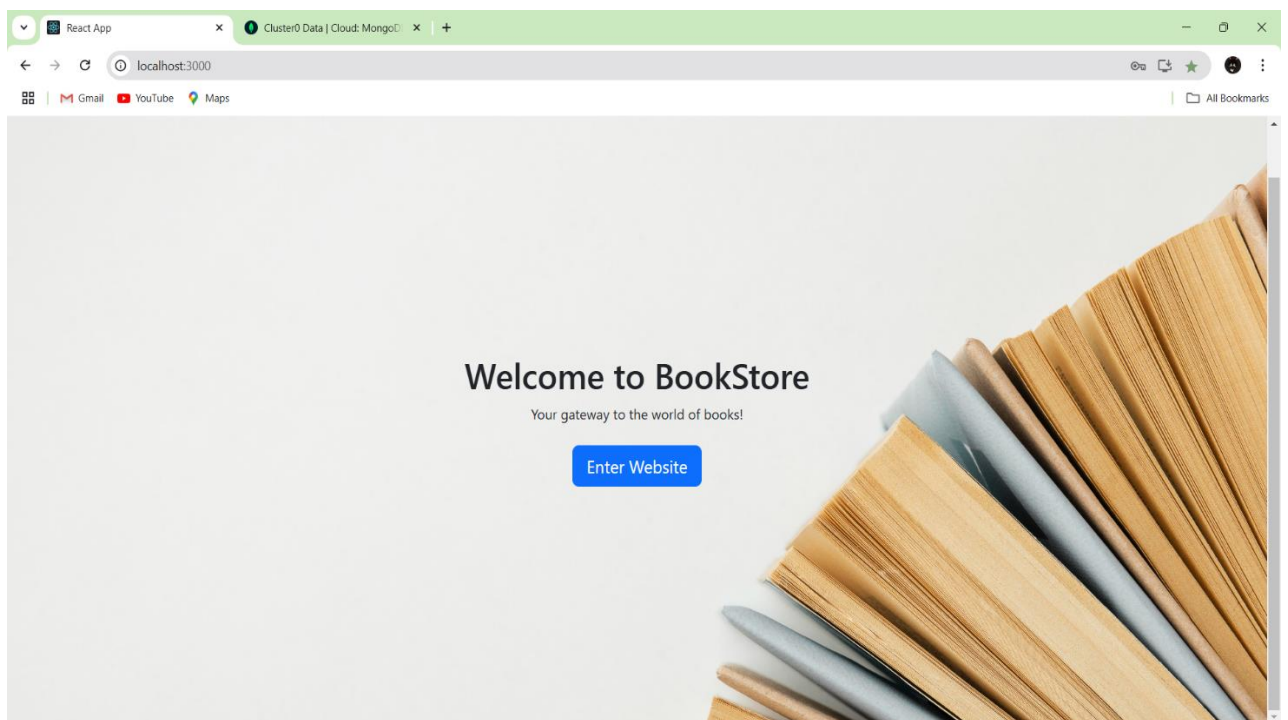
## 14.DEMO LINK

Drive Link: [https://drive.google.com/file/d/1oDREAM5tpKWl0ZLR5zfl\\_ATXQ-4jBMiF/view?usp=sharing](https://drive.google.com/file/d/1oDREAM5tpKWl0ZLR5zfl_ATXQ-4jBMiF/view?usp=sharing)

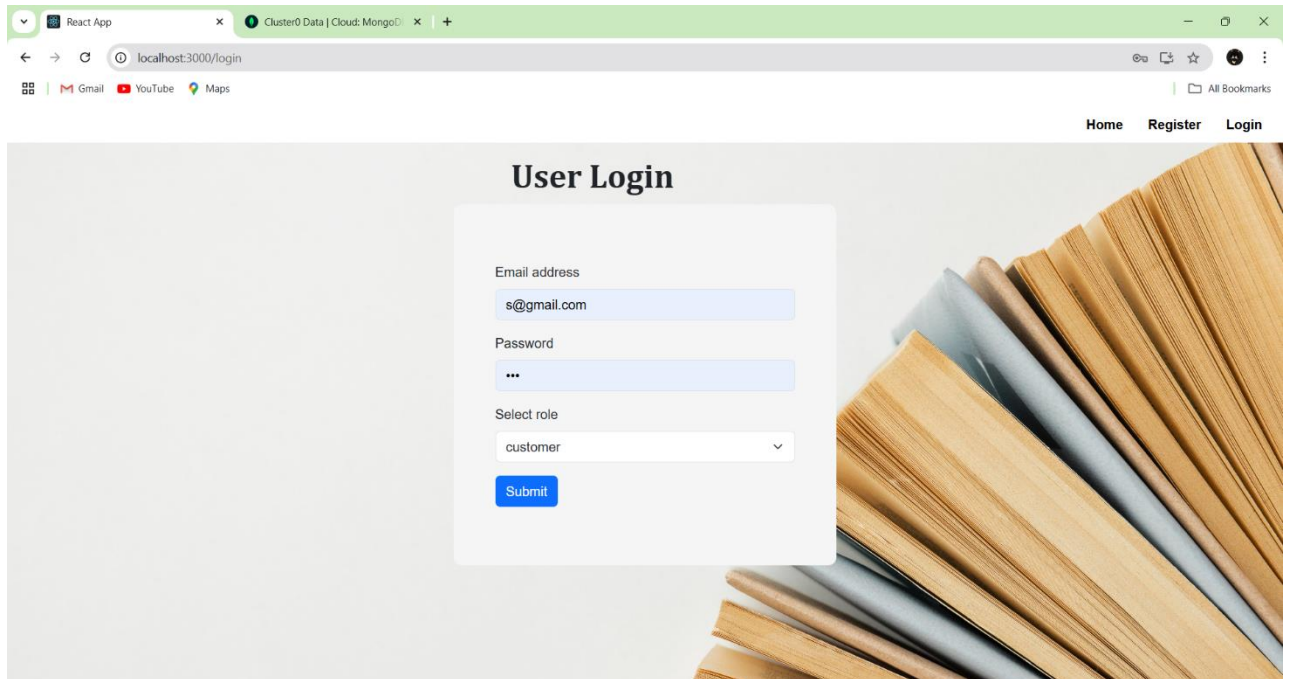
GitHub Link : <https://github.com/Thulasirajan03/Book Store>

## 15.USER INTERFACE AND SCREENSHOT

### Landing Page



## Login Page



The screenshot shows a web browser window with the title "React App" and the address bar displaying "localhost:3000/login". The browser's tab bar shows "Cluster0 Data | Cloud: Mongo". The page features a navigation bar with links for "Home", "Register", and "Login". The main content area is titled "User Login" and contains a form with the following fields: "Email address" (with the value "s@gmail.com"), "Password" (with masked characters "..."), and "Select role" (a dropdown menu with "customer" selected). A blue "Submit" button is located at the bottom of the form. The background of the page is a light gray with a stack of books on the right side.

React App Cluster0 Data | Cloud: Mongo

localhost:3000/login

Home Register Login

### User Login

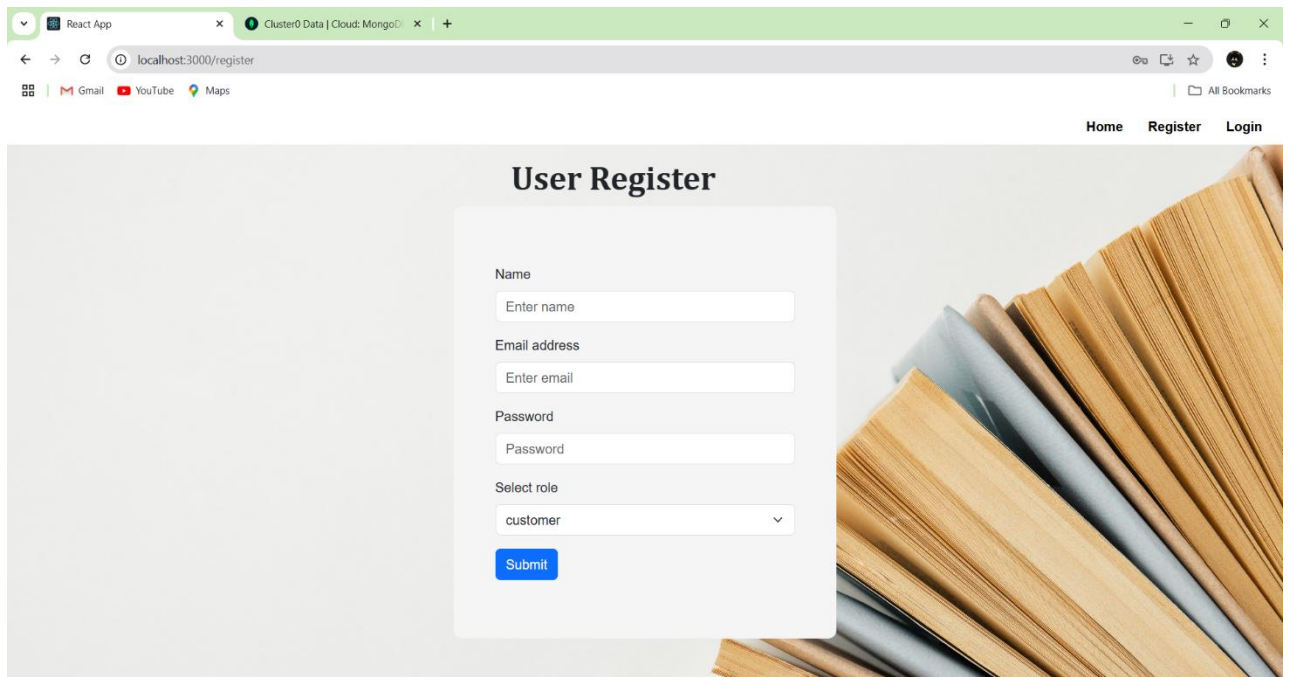
Email address  
s@gmail.com

Password  
...

Select role  
customer

Submit

## Register Page



The screenshot shows a web browser window with the title "React App" and the address bar displaying "localhost:3000/register". The browser's tab bar shows "Cluster0 Data | Cloud: Mongo". The page features a navigation bar with links for "Home", "Register", and "Login". The main content area is titled "User Register" and contains a form with the following fields: "Name" (with the placeholder "Enter name"), "Email address" (with the placeholder "Enter email"), "Password" (with the placeholder "Password"), and "Select role" (a dropdown menu with "customer" selected). A blue "Submit" button is located at the bottom of the form. The background of the page is a light gray with a stack of books on the right side.

React App Cluster0 Data | Cloud: Mongo

localhost:3000/register

Home Register Login

### User Register

Name  
Enter name

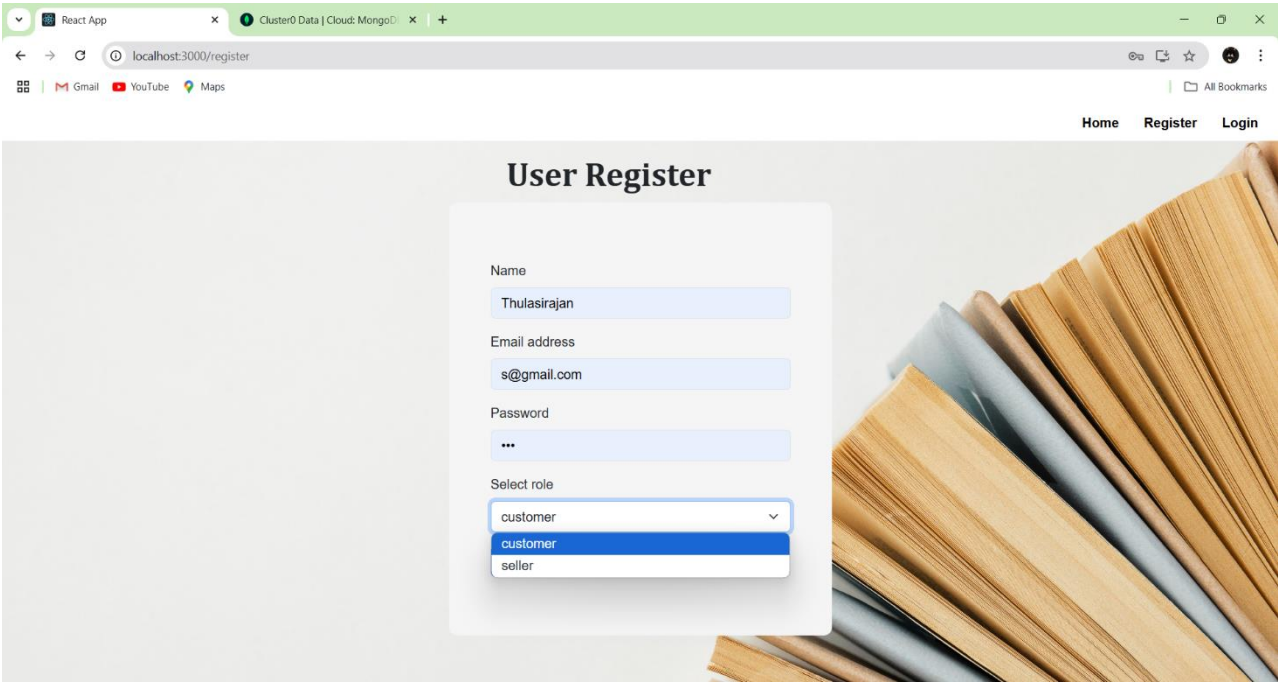
Email address  
Enter email

Password  
Password

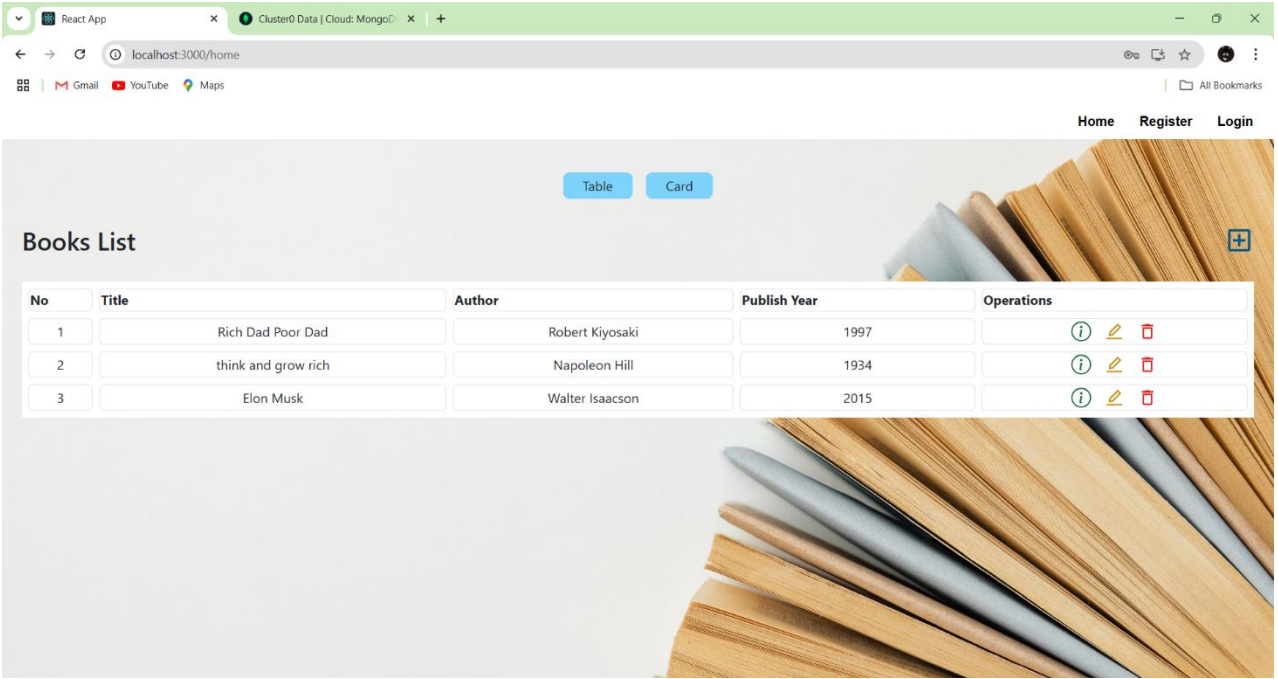
Select role  
customer

Submit

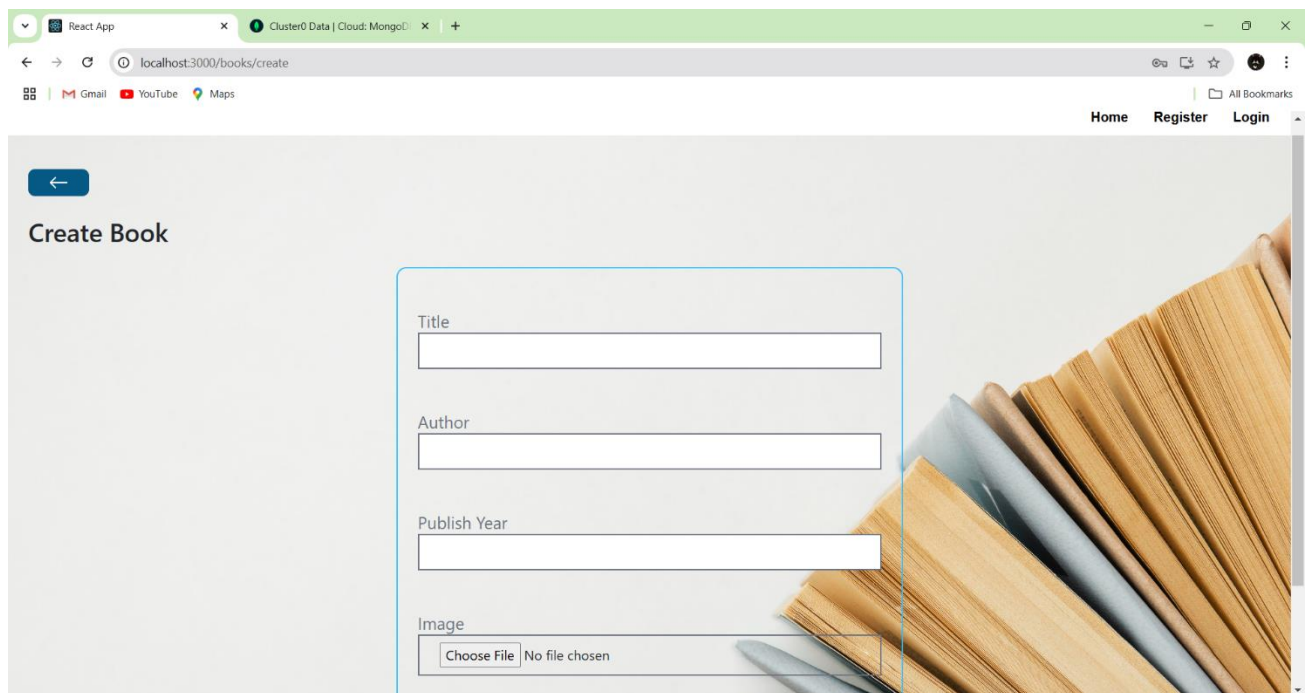
# Register Page with Customer & Seller



# Seller Page with Book List



## Add Book

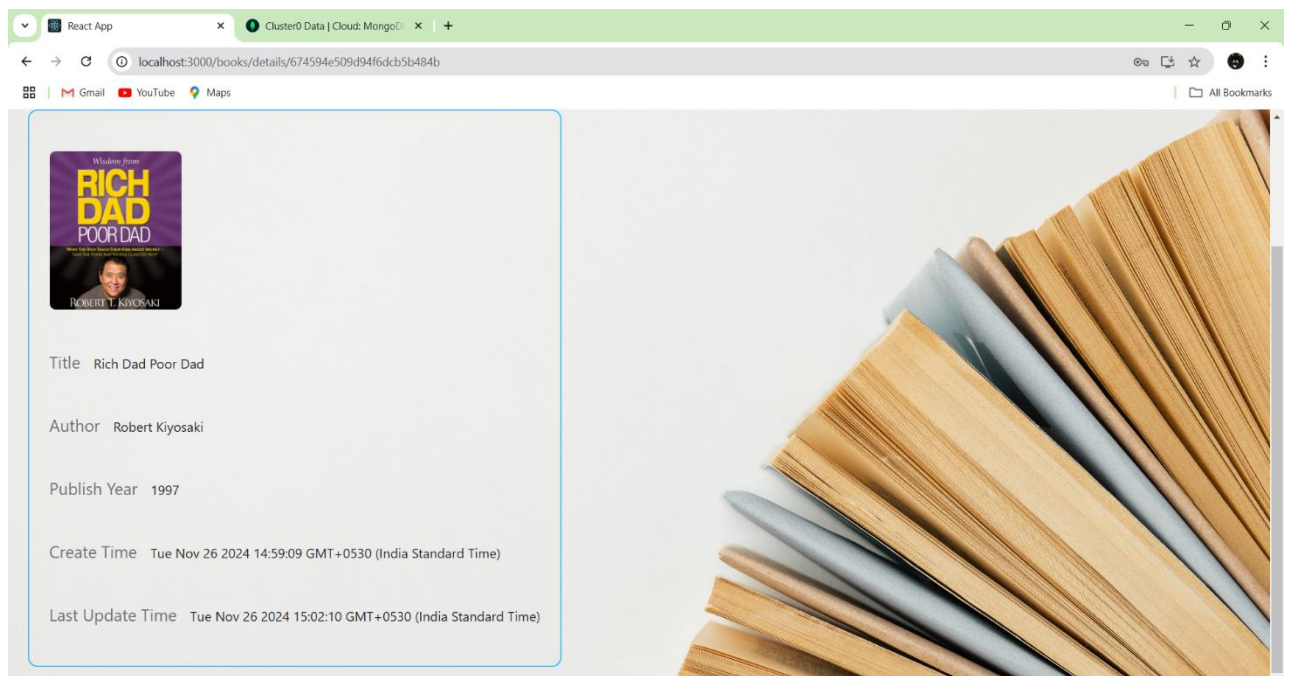


The screenshot shows a web browser window with the address bar displaying 'localhost:3000/books/create'. The page has a navigation bar with 'Home', 'Register', and 'Login' links. The main content area is titled 'Create Book' and contains a form with the following fields:

- Title:
- Author:
- Publish Year:
- Image:  No file chosen

The background of the page features a stack of books.

## Book Information



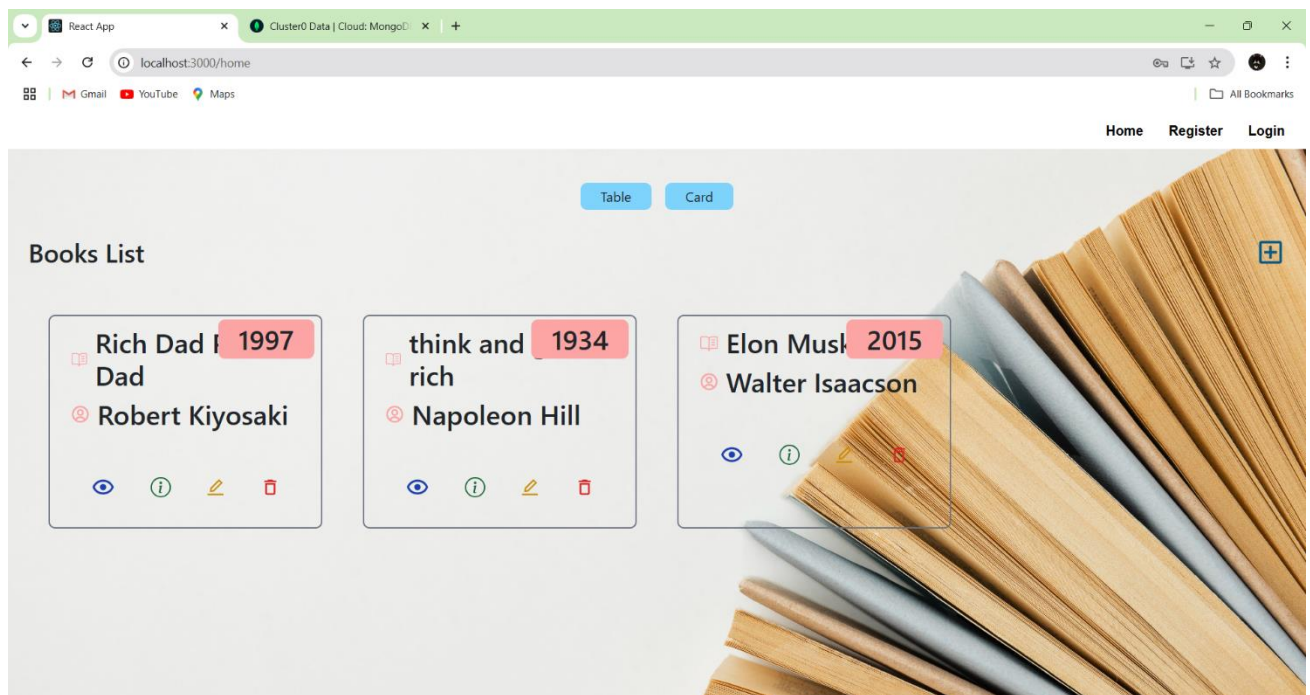
The screenshot shows a web browser window with the address bar displaying 'localhost:3000/books/details/674594e509d94f6dcb5b484b'. The page displays the details for the book 'Rich Dad Poor Dad' by Robert Kiyosaki. The information is presented in a list format:

- Title: Rich Dad Poor Dad
- Author: Robert Kiyosaki
- Publish Year: 1997
- Create Time: Tue Nov 26 2024 14:59:09 GMT+0530 (India Standard Time)
- Last Update Time: Tue Nov 26 2024 15:02:10 GMT+0530 (India Standard Time)

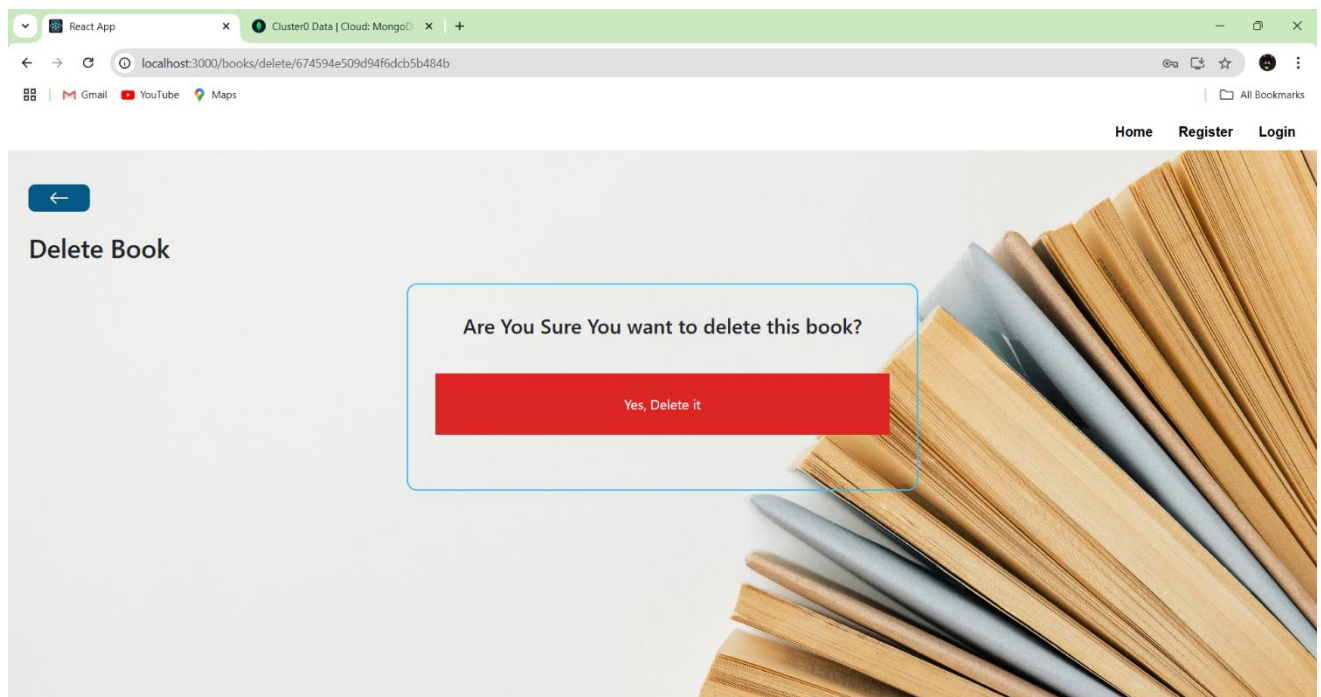
A book cover image for 'Rich Dad Poor Dad' is displayed at the top left of the information section. The background of the page features a stack of books.



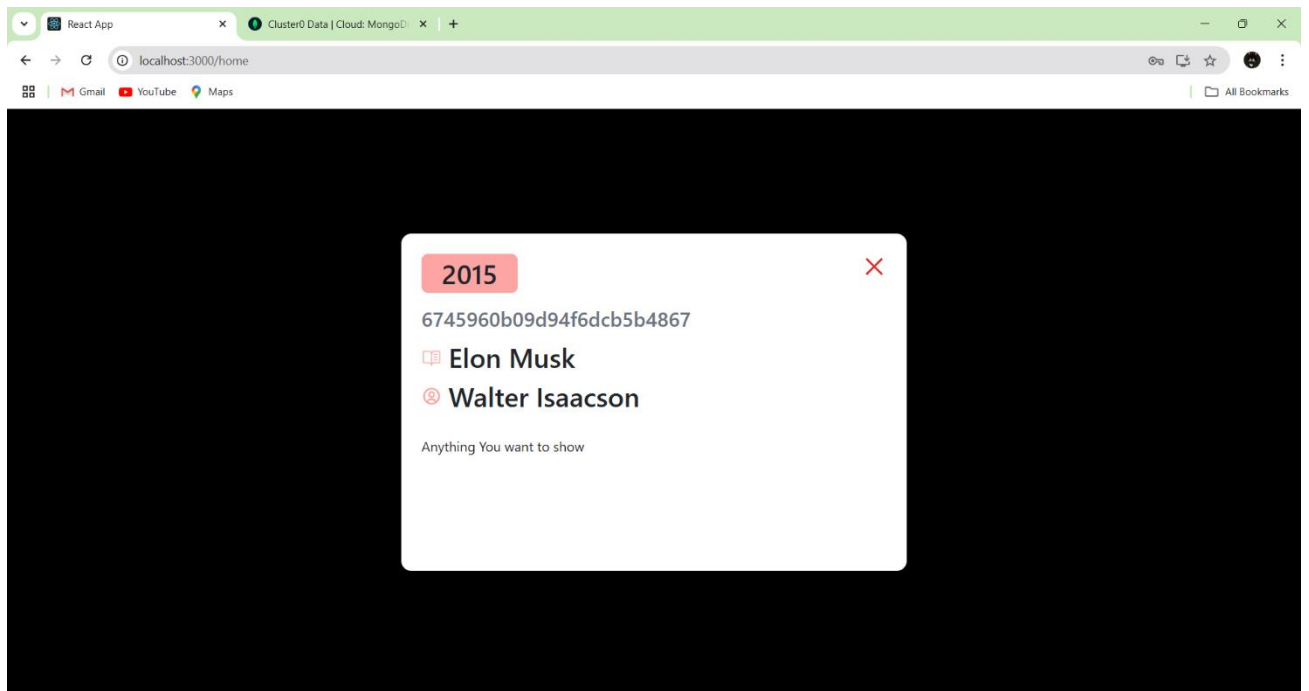
## Book cart



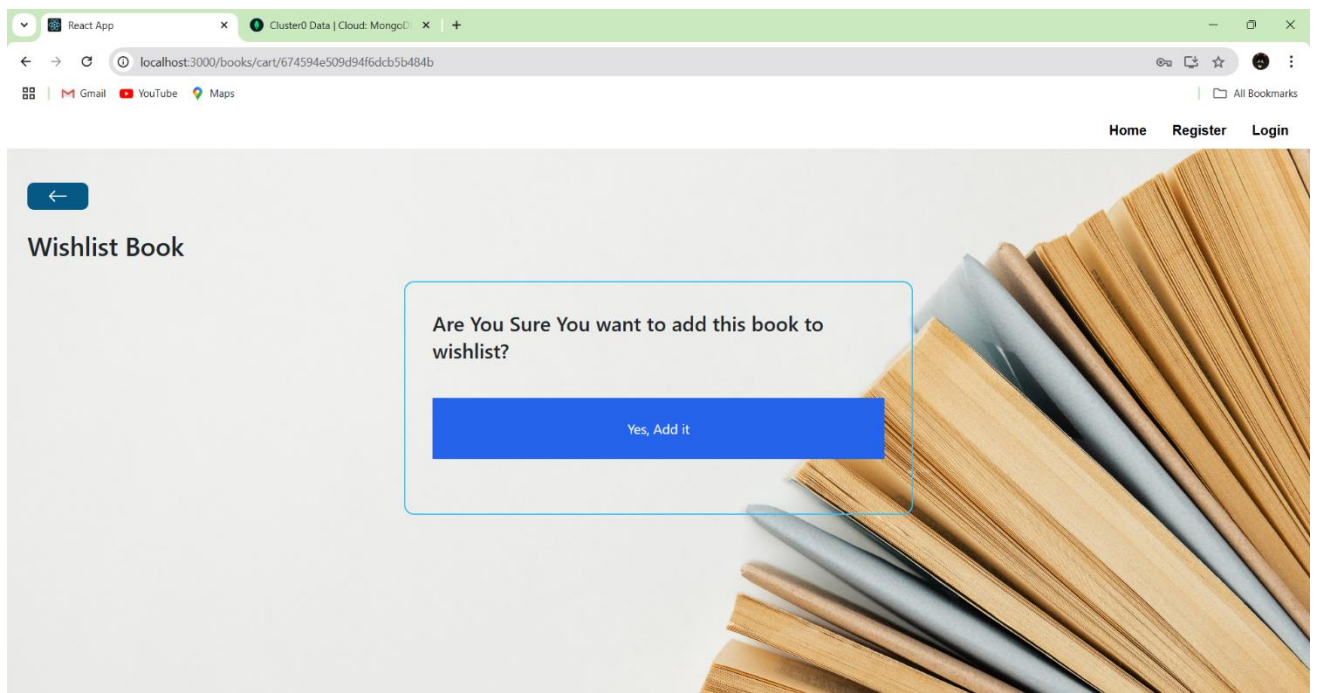
## Delete Book



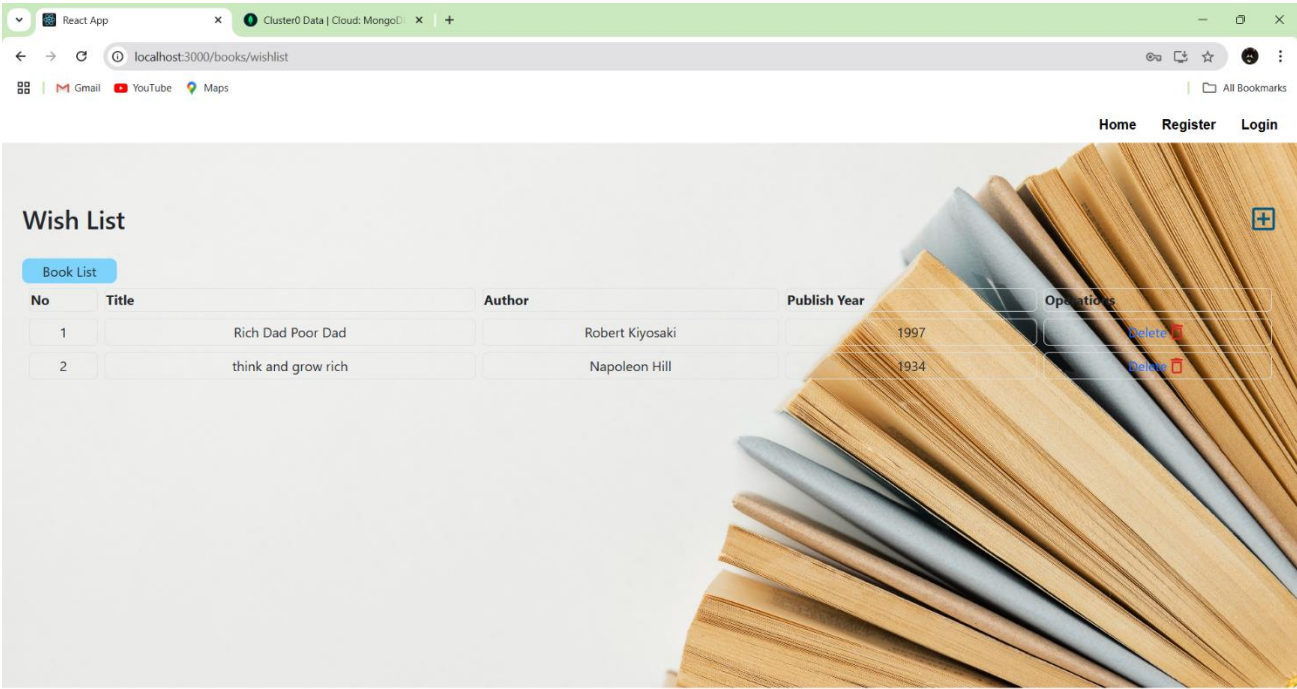
## Book Brief



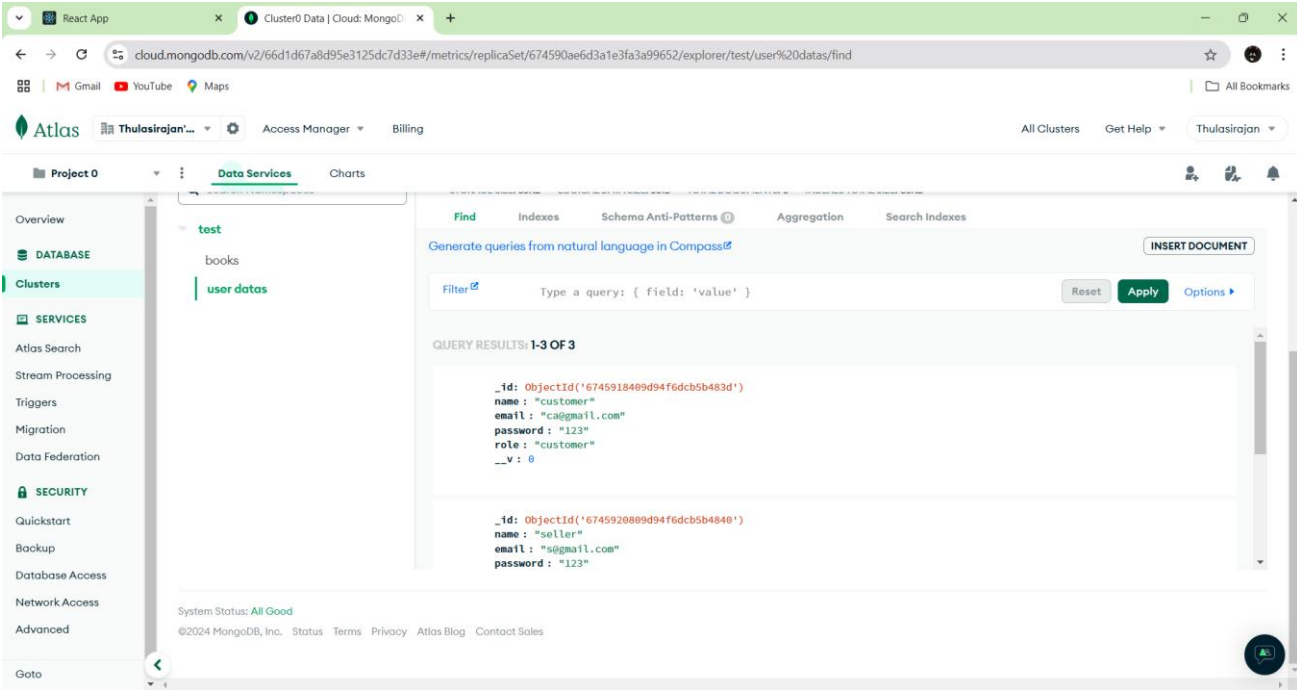
## Book Wishlist



# Wishlist Table



# Database Information



## 16. TESTING STRATEGY

### 1. Unit Testing:

- **Purpose:** Verify that individual components and functions work as expected.
- **Scope:** Test functions, methods, classes, and components in isolation.
- **Tools:** Jest, Mocha, Chai (for JavaScript testing).

### 2. Integration Testing:

- **Purpose:** Ensure that different components and services work together seamlessly.
- **Scope:** Test interactions between modules, such as the frontend and backend, or database connections.
- **Tools:** Supertest (for API testing), Postman (for manual API testing).

### 3. End-to-End (E2E) Testing:

- **Purpose:** Simulate real user scenarios to validate the entire application flow.
- **Scope:** Test the application from the user's perspective, covering complete workflows.
- **Tools:** Cypress, Selenium, Puppeteer.

### 4. Performance Testing:

- **Purpose:** Assess the application's responsiveness, speed, and stability under load.
- **Scope:** Test how the application performs under different conditions, such as high traffic or large data volume.
- **Tools:** Apache JMeter, LoadRunner.

### 5. Security Testing:

- **Purpose:** Identify vulnerabilities and ensure the application is secure from threats.
- **Scope:** Test for common security issues like SQL injection, XSS, CSRF, and data breaches.
- **Tools:** OWASP ZAP, Burp Suite.



## 17.KNOWN ISSUES

- **Slow Loading Times:** Slow server response times or inefficient code can lead to delays in page loading, affecting user experience.
- **Security Vulnerabilities:** Inadequate security measures can expose the application to risks such as data breaches and unauthorized access.
- **Inventory Management:** Keeping the inventory updated in real-time can be challenging, leading to discrepancies between actual stock and displayed stock.
- **Payment Processing Issues:** Integration with payment gateways can sometimes result in failed transactions or delays in payment processing.
- **User Experience Problems:** Poorly designed user interfaces or navigation can make it difficult for users to find and purchase books.
- **Database Performance:** As your bookstore grows, managing large datasets and ensuring efficient queries can become challenging. Proper indexing and optimizing queries are essential to maintain performance.
- **Authentication and Security:** Implementing secure authentication and protecting sensitive data (like user credentials and payment information) is crucial. Using JWT for authentication and ensuring HTTPS is used for data transmission are good practices.
- **Frontend-Backend Integration:** Ensuring smooth communication between the frontend (React.js) and backend (Node.js/Express.js) can sometimes be tricky, especially when handling asynchronous operations and managing state.
- **Scalability:** Designing the application to handle increased traffic and data is important. Using load balancing, caching, and considering cloud services can help in scaling the application.
- **User Experience (UX):** Creating a user-friendly interface that is intuitive and responsive across different devices can be challenging. Regular user testing and feedback are essential to improve UX.
- **Error Handling:** Properly handling errors and providing meaningful error messages to users can improve the overall user experience and help in debugging issues.
- **Deployment and Maintenance:** Setting up continuous integration/continuous deployment (CI/CD) pipelines and maintaining the application can be time-consuming. Using tools like Docker for containerization and deployment platforms like Heroku or AWS can simplify this process.

## 18.FUTURE ENHANCEMENTS

- **Advanced Search and Filtering:** Implement more sophisticated search capabilities, allowing users to filter books by various criteria such as genre, author, price range, publication date, and ratings. This can help users find exactly what they are looking for with ease.
- **Personalized Recommendations:** Develop a recommendation engine that suggests books based on user preferences, reading history, and purchase patterns. Machine learning algorithms can be employed to analyze user behaviour and provide tailored book recommendations.
- **Social Features:** Integrate social features like book clubs, discussion forums, and the ability for users to share their reading lists and reviews on social media platforms. This can foster a sense of community and increase user engagement.
- **Mobile App Development:** Create a mobile application for your bookstore to provide a seamless and optimized experience for users on the go. React Native can be used to build cross-platform mobile apps.
- **Subscription Service:** Offer a subscription-based model where users can subscribe to receive a certain number of books or access exclusive content each month. This can provide a steady revenue stream and enhance user loyalty.
- **Enhanced Payment Options:** Integrate multiple payment gateways to offer users a variety of payment options, including digital wallets, credit/debit cards, and bank transfers. This can improve the checkout experience and cater to a wider audience.
- **Voice Search and Assistants:** Implement voice search capabilities and integrate with virtual assistants like Amazon Alexa or Google Assistant. This can make the user experience more interactive and accessible.
- **Analytics and Insights:** Develop an analytics dashboard for administrators to track user behaviour, sales trends, and other key metrics. This can help in making informed business decisions and optimizing the bookstore's performance.
- **Localization and Multilingual Support:** Expand your reach by adding support for multiple languages and localizing content to cater to users from different regions.
- **Sustainability Initiatives:** Introduce features that promote sustainability, such as a section for second-hand books, book donations, or eco-friendly packaging options.

## 19.CONCLUSION

In conclusion, building a bookstore using the MERN stack (MongoDB, Express.js, React.js, and Node.js) offers a comprehensive solution for creating a robust, scalable, and user-friendly online platform. By leveraging MongoDB for efficient data management, Express.js for streamlined server-side operations, React.js for a dynamic and responsive user interface, and Node.js for handling asynchronous tasks, you can develop a high-performance application.

Throughout the development process, it's crucial to address common challenges such as database performance, authentication and security, frontend-backend integration, scalability, user experience, error handling, deployment, and testing. Implementing best practices and using appropriate tools and libraries can help mitigate these issues and enhance the overall functionality of your bookstore.

Future enhancements, such as advanced search and filtering, personalized recommendations, social features, mobile app development, and enhanced payment options, can further enrich the user experience and expand the capabilities of your application. By continuously iterating and improving your bookstore, you can provide a seamless and enjoyable experience for your users, ensuring the success and growth of your online bookstore.

## 20.REFERENCE

### **React.js Documentation. (2024)**

- React – A JavaScript library for building user interfaces. Retrieved from <https://reactjs.org/>

### **MongoDB Documentation. (2024)**

- MongoDB: The Database for Modern Applications. Retrieved from <https://www.mongodb.com/>

### **Node.js Documentation. (2024)**

- Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Retrieved from <https://nodejs.org/>

### **Libraries and Frameworks:**

#### **Node.js. (2024)**

- Node.js Documentation. Retrieved from <https://nodejs.org/>

#### **Express.js. (2024)**

- Express - Fast, unopinionated, minimalist web framework for Node.js. Retrieved from <https://expressjs.com/>
- JWT.io. (2024). JWT: JSON Web Token. Retrieved from <https://jwt.io/>

#### **Bootstrap. (2024)**

- Bootstrap: The most popular HTML, CSS, and JS library in the world. Retrieved from <https://getbootstrap.com>