**Full Stack Development with MERN**

**Project Title** - **Online Book Store**

Team Members

Samyuktha J – Frontend (Team Leader)

Ashwini E – Database

Janani VV – Backend

Lavanya V – Backend

**1. Project Overview**

**Purpose**

The primary goals are to provide users with an intuitive platform to browse, search, and purchase books while streamlining inventory management and order processing for bookstore administrators. The project serves as a modern, digital alternative to traditional bookstores, bringing convenience and accessibility to book enthusiasts.

**Features**

1. **Personalized Recommendations**: A recommendation engine that suggests books to users based on their browsing history, purchases, and ratings, enhancing the user experience.

2.**Admin Dashboard**: Admins can manage and monitor books, users, and orders. This includes adding new books, updating inventory, and viewing user activities and order histories.

3. **Mobile Responsiveness**: The platform is designed to be fully responsive, ensuring that users can browse and shop for books seamlessly on various devices, including smartphones and tablets.

**2. Architecture**

**Frontend - React**

Framework: Built with React for a dynamic and responsive user interface.

Reusable Components:

Navbar: For app-wide navigation links.

BookList: Displays available books with filtering and sorting options.

BookDetail: Shows detailed information about each book, including reviews..

State Management: Uses Context API or Redux to manage user and cart state across the app.

Routing: React Router enables seamless navigation between pages (Home, Book Details, Cart, Profile, etc.).

API Integration: Axios or Fetch API handles communication with backend endpoints.

CSS Frameworks: Bootstrap or Tailwind CSS for responsive and mobile-friendly design.

Custom Hooks: Reusable logic for data fetching and authentication, ensuring modularity.

**Backend - Node.js & Express**

RESTful API: Provides structured endpoints for managing core resources:

/api/books for book data

/api/users for user management

/api/orders for cart and order processing

Security:

JWT Authentication: Ensures secure access to protected routes.

Middleware: Custom middleware for error handling and validation (e.g., user input).

Environment Configuration: Sensitive data (e.g., database URI, API keys) stored in environment variables

**Database - MongoDB**

Database Structure:

Users Collection: Contains usernames, emails, hashed passwords, and order history.

Books Collection: Stores book titles, authors, prices, descriptions, and inventory.

Orders Collection: Holds user orders, items, total cost, and order statuses.

Reviews Collection: Links users to book reviews with ratings and comments.

Mongoose ORM:

Defines schemas for each collection, enforces data types and relationships.

Relationships:

ObjectIds create references between collections (e.g., linking reviews to books, orders to users).

Data Retrieval:

Efficient querying for complex data (e.g., fetching all reviews for a book or all orders for a user).

**3.Setup Instructions**

**Prerequisites**  
 Make sure to have the following software installed:

* **Node.js**
* **MongoDB**
* **Git**

**Installation**

1. **Clone the Repository**
2. **Install Dependencies**
   * **Backend**

cd backend

npm install

* + **Frontend**

cd /frontend

npm install

1. **Dependencies**

Front end

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"

Backend

"dependencies": {

 "bcryptjs": "^2.4.3"

"cors": "^2.8.5",

"express": "^4.18.2",

"jsonwebtoken": "^9.0.2"

"mongo": "^0.1.0",

"mongod": "^2.0.0",

"mongodb": "^6.10.0",

"mongoose": "^8.0.1",

"multer": "^1.4.5-lts.1",

"nodemon": "^3.0.1"

1. **Environment Variables**  
   In the **backend** folder, create a .env file with:

MONGO\_URI=your-mongodb-url

JWT\_SECRET=your-jwt-secret

1. **Run the Application**  
   Start both the backend and frontend servers

cd backend && npm run dev

cd /frontend && npm run dev

**4.Folder Structure**

**Client - React Frontend**

**src**: Contains the main React application code.

**components**: Reusable components, such as Navbar, BookList, BookDetail, and ShoppingCart.

**pages**: Different pages of the app (e.g., HomePage, BookPage, CartPage, ProfilePage.

**context**: State management using context providers for global state (e.g., AuthContext, CartContext.

**Hooks**: Custom React hooks for reusable logic (e.g., useFetchBooks, useAuth.

**services**: API calls using Axios or Fetch for interacting with backend endpoints.

**App.js**: Main application component that sets up routing with React Router.

**index.js**: Entry point for rendering the React app.

**Server (Node.js Backend)**

**controllers**: Handles the core logic for each route (e.g., bookController, userController, selllerController.

**models**: Mongoose models defining schemas for collections like User, Book and Order.

**Routes**: Defines RESTful endpoints for resources (e.g., bookRoutes, userRoutes, sellerRoutes.

**middleware**: Custom middleware for tasks like authentication (JWT), error handling, and validation.

**config**: Configuration files, including database connection and environment variables.

**server.js:** Main server file to initialize Express, set up middleware, and start the server.

**5.Running the Application**

To start the application locally, use the following commands:

* **Frontend**  
  Navigate to the frontend directory and start the React app:

cd client

npm run dev

* **Backend**  
  Navigate to the backend directory and start the server:

cd server

npm run dev

6.API documentation

**Admin Endpoints (Located in backend/admin)**

**1.Admin Login**

Endpoint: /admin/login

Method: POST

Description: Authenticates an admin user.

Request Parameters:

username (string, required)

password (string, required)

Example Response:

{

"message": "Admin login successful",

"admin": {

"id": "1",

"username": "admin123"}

}

**Seller Endpoints (Located in backend/seller)**

**1. Add Book**

Endpoint: /seller/addbook

Method: POST

Description: Adds a book for sale by a seller.

Request Parameters:

title, author, price, description, etc.

Example Response:

{"message": "Book added successfully",

"book": {

"id": "101",

"title": "Sample Book" }

**2. View My Products**

Endpoint: /seller/myproducts

Method: GET

Description: Retrieves all books listed by the seller.

Example Response:

[

{

"id": "101",

"title": "Sample Book",

"price": 15.99

}

]

**User Endpoints (Located in backend/user)**

**1.User Sign-up**

Endpoint: /user/signup

Method: POST

Description: Registers a new user.

Request Parameters:

username, email, password, etc.

Example Response:

{

"message": "User registered successfully",

"user": {

"id": "201",

"username": "user123"

}

}

**2. View Wishlist**

Endpoint: /user/wishlist

Method: GET

Description: Retrieves the wishlist for the user.

Example Response:

[

{

"bookId": "105",

"title": "Wishlist Book",

"author": "Author Name"

}

]

7.Authentication

In this project, **authentication** and **authorization** ensure that only registered users can access certain features (like placing orders or leaving reviews), while admins have permissions to manage books and orders.

**1.JWT (JSON Web Tokens) Authentication**

* + The project uses **JWT tokens** to securely authenticate users. When a user logs in, the backend generates a JWT and sends it back to the client.
  + This token is stored securely on the client side (in localStorage or sessionStorage) and is included in the headers of future requests to access protected resources.
  + JWT tokens have a limited validity period for security. When expired, the user needs to log in again to obtain a new token.

**2.Authorization**

* + **Role-Based Access**: The backend checks user roles (regular user vs. admin) to restrict access to certain actions:
    - **Users** can browse books, add reviews, and place orders.
    - **Admins** can add, update, or delete book entries and manage orders.
  + **Middleware**: Middleware functions in the backend validate the JWT on each request to ensure that only authenticated users can access protected routes (e.g., order history, checkout).

**3.Session Management**

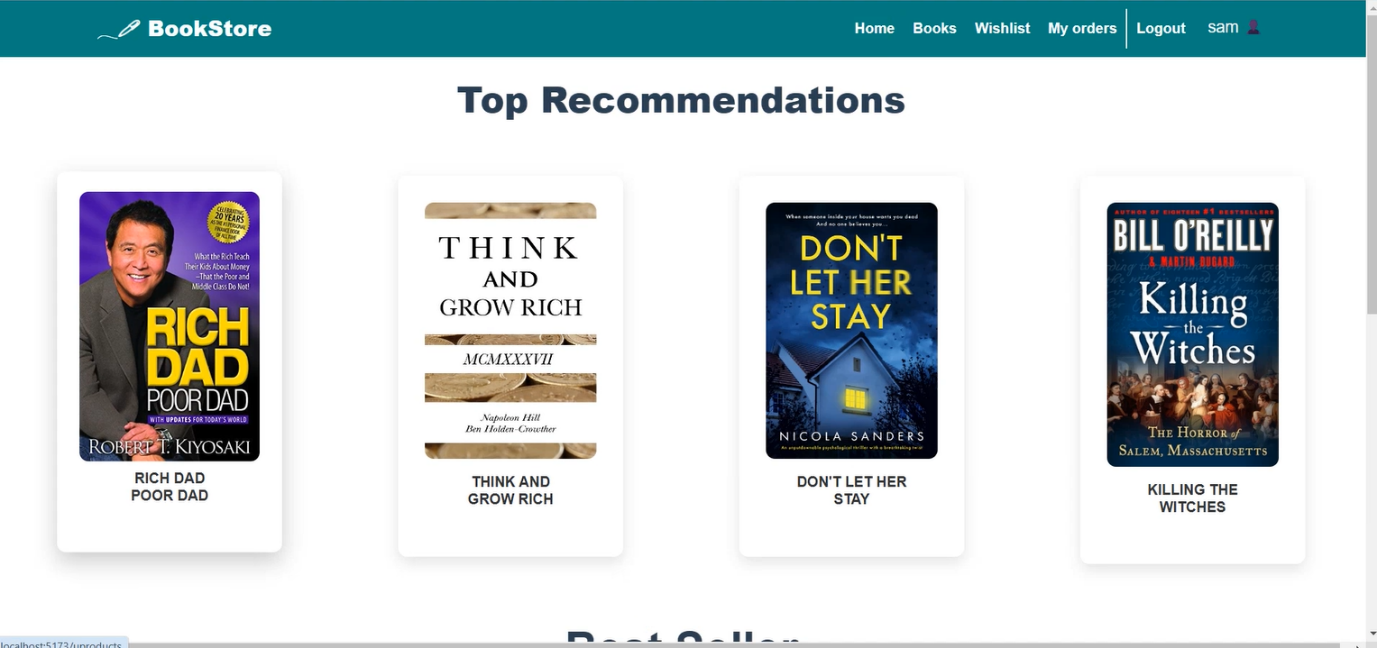
* + JWT-based authentication does not require server-side session storage, making the app scalable. Each request includes the JWT, which is verified by the server without the need to maintain sessions.

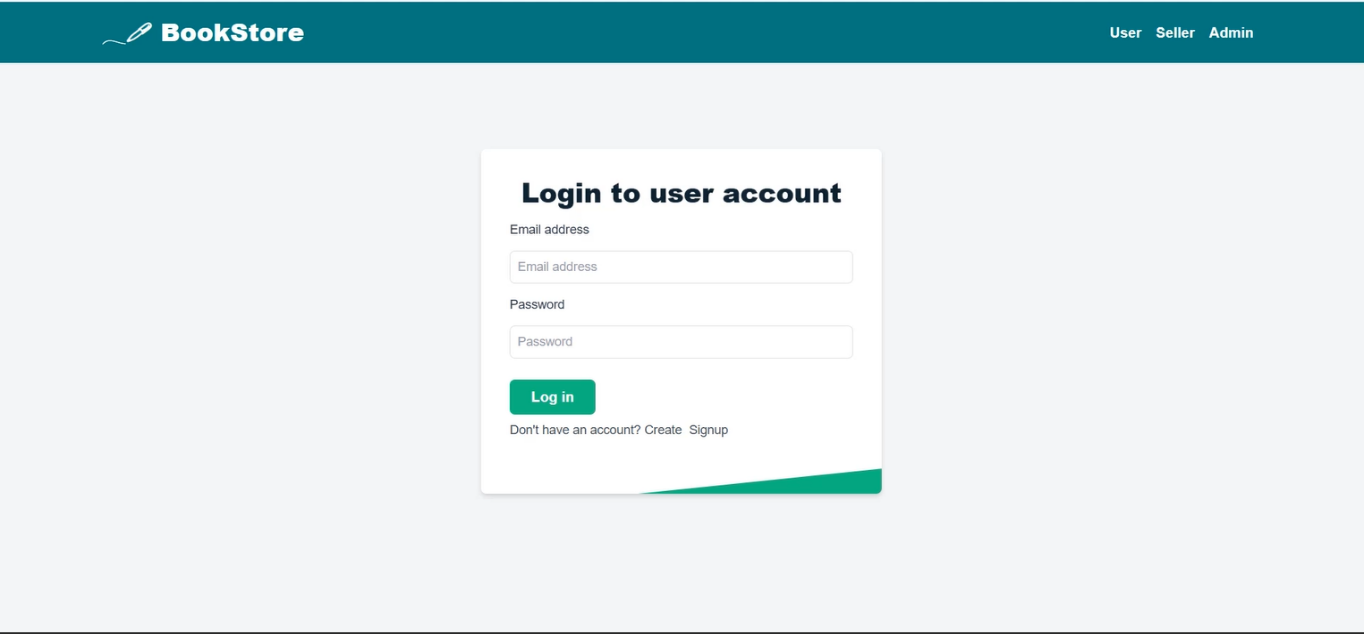
**4.Security Measures**

* + **Password Hashing**: User passwords are securely hashed before storage in the database, preventing unauthorized access in case of data leaks.
  + **Environment Variables**: Sensitive keys, like the JWT secret and database URI, are stored in environment variables to avoid exposing them in the codebase.

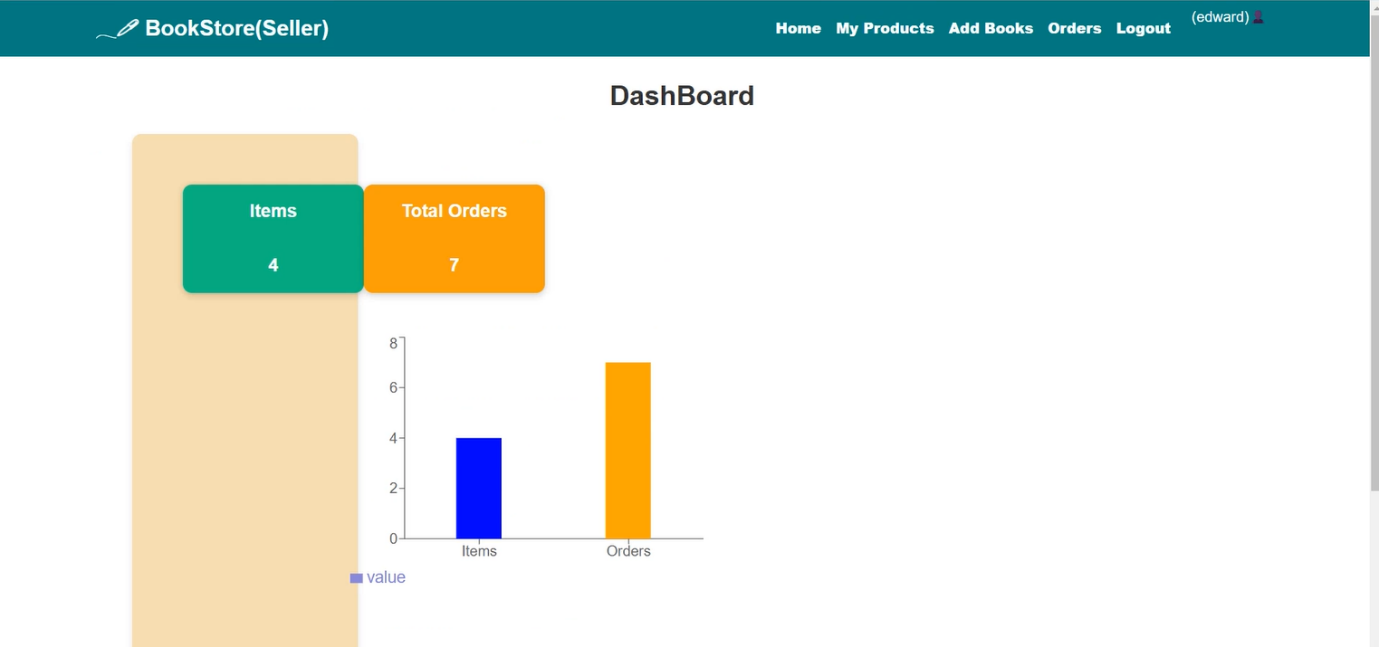
**8. User Interface**

The overall UI adopts a minimalist approach, reducing clutter and focusing on essential content. This not only improves usability but also ensures a faster, more enjoyable browsing experience.





The interface is designed with simplicity and usability in mind, ensuring that users can easily navigate through the bookstore without confusion. The **Navbar** offers clear categories for browsing books (e.g., genres, authors), user profile management, and shopping cart. It helps users quickly find what they're looking for.



**9.Testing**

**1.Testing Strategy**

**Unit Testing**: Focuses on individual components, functions, and routes to ensure they work as expected. Unit tests cover frontend components (e.g., `Navbar`, `BookList`) and backend API endpoints (e.g., `/api/books`, `/api/orders`).

**Integration Testing**: Tests the interaction between frontend and backend, including API calls and data flow. Integration tests validate that users can browse books, add items to the cart, and complete checkout smoothly.

**2. Tools Used**

**Jest**: A JavaScript testing framework used primarily for unit and integration testing of frontend and backend code.

**React Testing Library**: Used with Jest to test React components, checking that UI elements render and function correctly.

**10. Demo**

Complete demonstration of our online book store project -

https://drive.google.com/file/d/1jjH\_NHskrOxWP9iNGKa8fV7XiyIlT0va/view?usp=drivesdk

**11.Known Issues**

**1.Limited Error Handling on Checkout**

During the checkout process, errors from the payment gateway (e.g., declined card) may not be fully detailed for the user. Enhanced error messaging and handling are planned for the next release.

**2.Mobile Responsiveness**

While the app is largely responsive, some components, like the shopping cart and order history pages, may not display optimally on smaller screens. Further CSS adjustments are scheduled to improve mobile compatibility.

**3.Browser Compatibility**

Certain UI elements may not display correctly in older versions of Internet Explorer. It is recommended to use a modern browser (Chrome, Firefox, or Edge) for the best experience.

**12. Future Enhancements**

**1. Advanced Search and Filtering**

Implement a more comprehensive search functionality, allowing users to filter by genre, author, price range, and rating, making it easier to find books that match specific criteria.

**2. Personalized Recommendations**

Integrate a recommendation engine to provide personalized book suggestions based on user preferences, reading history, and popular trends, enhancing user engagement.

**3. Multi-Language Support**

Enable multi-language support to make the bookstore accessible to a global audience, allowing users to switch between languages for a more inclusive experience.

**4. Enhanced Security Measures**

Add features like two-factor authentication (2FA) for user accounts and fraud detection on payments to improve overall security for users and the platform.