**Full Stack Development with MERN**

**Project Documentation**

**PROJECT:**

**ONLINE BOOK STORE APPLICATION**

**Team Members**

**ABISHEK S (311521104003)**

**HARSHAVARDHAN G (311521104016)**

**SASHONG S (311521104051)**

**DHANUSH S (311521104304)**

**1. Introduction**

* **Project Title:** Book Store
* **Team Members:**

S Abishek (311521104003) – Front-End Development  
G Harshavardhan (311521104016) – Database Set Up and Connectivity  
S Sashong (311521104051) – Back-End Development  
S Dhanush (311521104304) – Project Configuration and Integration

**2. Project Overview**

The Book-Store Application is a cutting-edge e-commerce platform designed to cater to the evolving needs of modern book lovers. Built using the powerful MERN stack (MongoDB, Express.js, React, and Node.js), this application combines robust backend functionality with a seamless and visually engaging user interface.

It bridges the gap between traditional bookstores and digital convenience, offering a platform where users can effortlessly discover, explore, and purchase books from an extensive catalog. With features like secure authentication, detailed book listings, personalized recommendations, and streamlined order management, the application creates an immersive experience for readers.

The project also empowers administrators with comprehensive tools to manage book inventory, monitor orders, and oversee user accounts, ensuring the platform runs smoothly and efficiently. Designed with scalability, security, and usability in mind, the Book-Store Application transforms how literature enthusiasts connect with their favorite books while embracing the possibilities of modern technology.

**Purpose:**

The purpose of the Book-Store Application is to redefine how book enthusiasts discover, explore, and purchase literature by providing a seamless, secure, and engaging digital platform. The application caters to the needs of readers and administrators, enabling efficient book management and a personalized shopping experience.

* **Enhancing Customer Experience**: Users can browse an extensive collection of books, filter selections based on preferences, and manage their profiles and orders through an intuitive interface.
* **Admin Control and Management**: Administrators are empowered with robust tools to manage book inventory, track orders, and maintain user data for efficient platform operation.
* **Promoting Secure Transactions**: Leveraging advanced security measures, the platform ensures safe user authentication, data protection, and order processing.
* **Book Discovery and Customization:** Readers can explore books by genre, author, and ratings, and make informed purchasing decisions with detailed descriptions and reviews.
* **Fostering Convenience with Cart Management:** Users can easily add books to their cart, manage quantities, and save preferred items for later purchases, ensuring a streamlined shopping journey.

**Features:**

1. **User Authentication and Management**

* Secure JWT-based authentication for reliable access control.
* Password hashing with bcrypt ensures data protection.
* Session handling for smooth multi-tab browsing and token expiry for security.
* "Forgot Password" functionality to assist users without delays.

2. **Customer Dashboard and Profile Management**

* Users can view and update profile details, track active, completed, and canceled orders.
* Access to cart and wishlist for managing books of interest and purchases efficiently.

3. **Book Listings and Filters**

* Comprehensive catalog of books displayed with details such as title, author, genre, description, price, and ratings.
* Advanced filtering by genre, author, and popularity, with the ability to clear filters for tailored browsing.

4. **Book Browsing and Search Functionality**

* Intuitive search bar for locating books by keywords related to titles, genres, or authors.
* Enhanced navigation ensures users can quickly find their desired books.

5. **Book Details and Cart Management**

* Detailed book pages showcasing title, description, price, ratings, and availability.
* Options to "Buy Now" or "Add to Cart" for a seamless purchasing process.
* Cart displays selected items, quantities, and total price, with options to remove or proceed to checkout.

6. **Wishlist Functionality**

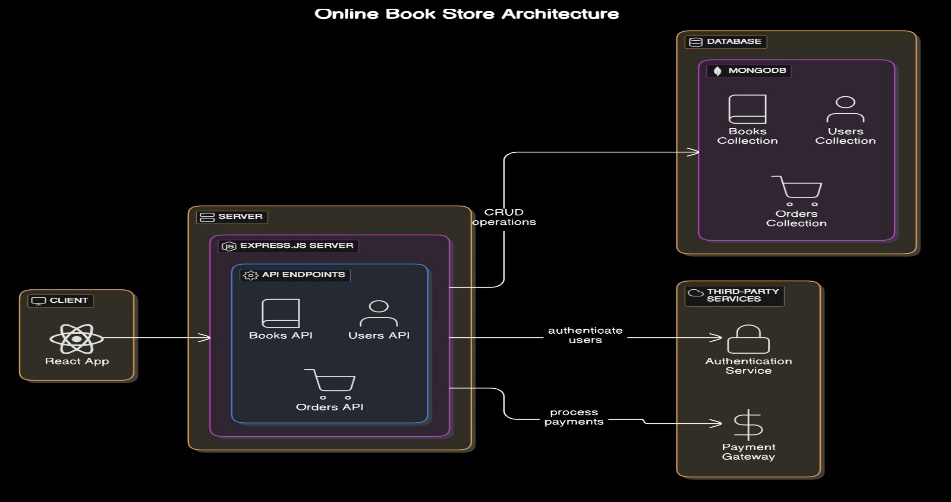
* Users can save books for future purchases and easily move them to the cart when ready.

7. **Order Management**

* Customers can view detailed order statuses (placed, in-transit, delivered) and order history.
* Information includes payment method, order date, and expected delivery timeline.

8. **Admin Dashboard and Functions**

* + Centralized hub for managing books, users, and orders.
  + Add, update, or delete book listings with ease.
  + Monitor and filter orders based on their status for better tracking and management

**3. Architecture**

*Figure 1.Architecture*

**Frontend**The frontend is structured as a single-page application (SPA) using React. This architecture aims to deliver an interactive user experience, where various components dynamically load and update content without needing full page reloads.

* **Component Structure:**

Functional Components: All React components are structured as functional components for better performance and use of React hooks.

* **State Management:**

Context API, combined with custom hooks and reducers, is used for managing global state (e.g., user authentication, cart data, and filters). Session storage is used to store JWT tokens and user data to manage sessions across different tabs. Tokens are verified before making requests.

* **Routing:**

React Router: react-router-dom is used for client-side routing, with different routes for each main page and protected routes for user-specific and admin-specific pages.

* **API Communication:**

Axios is used for HTTP requests to communicate with the backend API. The JWT token is included in the header of each request to ensure secure access to protected routes. Centralized error handling with feedback for users, such as login failure, invalid inputs, or out-of-stock items.

* **Styling and UI:**

Styled components or inline styles are used for modular and scalable CSS. CSS media queries are used to ensure responsiveness across devices, particularly for product lists and the cart.

**Backend:**The backend is built with Node.js and Express.js, providing a RESTful API for the frontend and managing user authentication, product data, and order processing.

* **Server Structure:**

The main application is built using Express which initializes all middleware, routes, and database connections. Environment Configuration is done using dotenv and is used to manage environment variables like database URI, JWT secret, and server port.

* **API Design:**
  + RESTful Endpoints: CRUD operations for Users, Products, Orders, and Cart items.
  + Route Modules: Routes are modularized for each entity (e.g., authRoutes, productRoutes, orderRoutes, userRoutes) for better organization.
  + Middlewares:
  + JWT Middleware: Validates tokens for protected routes to ensure only authenticated users can access restricted endpoints. Centralized error handling to manage all types of server-side errors. express-validator is used for validating user inputs, such as registration and login fields.
* **Authentication and Authorization:**
  + JSON Web Tokens are used for secure access, generated on login and verified on protected routes. bcrypt is used to hash user passwords before storing them in the database.
  + Role-Based Access Control functions are used to control access for Customer and Admin roles, allowing only admins to access certain endpoints.
* **Session and Token Handling:**

Tokens have an expiry time, after which users need to log in again. The token, user ID, and user type are stored in the frontend session storage for identifying the logged-in user across pages.

**Database:**The database uses MongoDB with a schema defined through Mongoose. Collections are created to manage Users, Products, Orders, and Cart items, ensuring data integrity and efficient querying.

**Database Structure:**

**i) User Schema**

* **Fields:**
  + **username:** String, unique identifier for the user.
  + **password:** String, hashed for secure storage.
  + **email:** String, unique email address of the user.
  + **usertype:** Enum, defines roles like customer or admin.
* **Indexes**:
  + Unique indexes on username and email to prevent duplicates.
* **Relationships**:
  + userId is referenced in the Orders and Wishlist collections.

**ii) Book Schema**

* **Fields**:
  + **title:** String, name of the book.
  + **author:** String, name of the author.
  + **genre:** String, genre of the book.
  + **description:** String, a brief summary of the book.
  + **price:** Number, cost of the book.
  + **availability:** Boolean, stock status of the book.
  + **rating:** Number, average user rating of the book.
  + **mainImg:** String, URL for the primary book image.
  + **carousel**: Array of Strings, URLs for additional images.
* **Indexes**:
  + Indexed by title and genre for efficient searching and filtering.
* **Relationships**:
  + bookId is referenced in the Cart and Orders collections.

**iii) Order Schema**

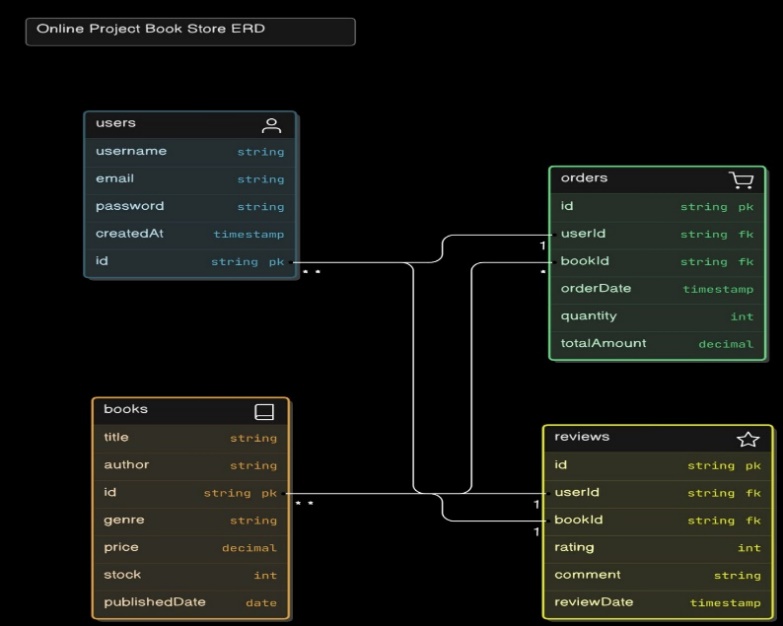
* **Fields**:
  + **userId:** Reference to the User collection.
  + **books:** Array, contains details of books in the order.
  + **orderDate:** Date, when the order was placed.
  + **deliveryDate:** Date, estimated delivery time.
  + **paymentMethod:** String,method used for payment.
  + **orderStatus:** Enum, indicates order state (e.g., placed, in-transit, delivered, or canceled).
* **Relationships**:
  + userId references the User collection.
  + Each book in the books array contains a reference to the Book collection.

**iv) Cart Schema**

* **Fields**:
  + **userId:** Reference to the User collection.
  + **bookId:** Reference to the Book collection.
  + **quantity:** Number, quantity of the selected book.
  + **price:** Number, price of the book.
* **Relationships**:
  + Each cart item references userId and bookId.
* **Constraints**:
  + Ensure uniqueness of the userId and bookId combination to avoid duplicate cart entries.

**v) Wishlist Schema**

* **Fields**:
  + **userId:** Reference to the User collection.
  + **bookId:** Reference to the Book collection.
* **Relationships**:
  + userId and bookId act as foreign keys referencing the User and Book collections.
* **Constraints**:
  + Ensure uniqueness of the userId and bookId combination for each wishlist entry.

****

*Figure 2.ERD Diagram*

**4. Setup Instructions**To set up the Book Store application, follow these steps:

**1. Install Prerequisites:**

* **Node.js (Version 20.x):** Download and install from Node.js official website, which includes npm (Node Package Manager).
* **MongoDB:** Set up MongoDB using MongoDB Compass for a local instance or create a MongoDB Atlas account for a cloud-based setup.
* **Git:** Install Git from Git downloads for version control.
* **npm -** Node Package Manager (comes with Node.js).
* **Code Editor -** Visual Studio Code or another preferred IDE.

**2. Download Project Files:**

Place all project files in a dedicated project directory on the local machine.

**3. Install Dependencies:**

Open a terminal, navigate to the project directory, and install the required dependencies:

*npm install*

**4. Set Up Environment Variables:**

In the project root, create a .env file to store environment-specific variables such as database connection strings and JWT secrets. Contents in the .env file is as follows:

PORT=3001

MONGO\_URI=<mongodb-connection-string>

JWT\_SECRET=<jwt-secret>

**5. Run the Application:**

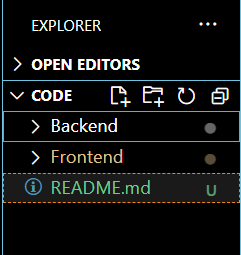
Start the application in development mode:

*npm start*

Now the application can be accessed locally at http://localhost:3000.

**5. Folder Structure**

The Book Store application follows a well-organized folder structure for both the **Frontend** (React) and **Backend** (Node.js) to ensure clarity, scalability, and maintainability of the project.

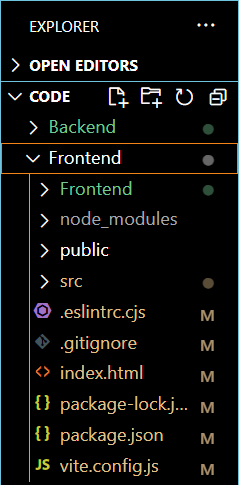


*Figure 3.Overall Folder Structure*

**Client: React Frontend Structure**

The frontend of Book Store is built using **React**, and its folder structure is organized as follows:

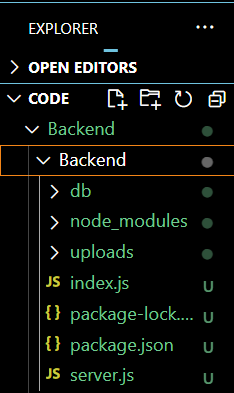
* **public/**:
  + Contains static assets like the index.html file, favicon, and other public files that are served directly to the browser.
* **src/**:
  + This is where all the source code of the React application resides.
  + **components/**: Contains reusable components such as the header, footer, product card, cart items, etc. These components are used throughout the application.
  + **pages/**: Contains React components that represent different pages in the application, such as Home, ProductDetails, Cart, Profile, etc.
  + **context/**: Handles the global state management for the app using React Context API. This includes user authentication and cart state management.
  + **styles/**: Contains custom CSS files to style the components and pages, ensuring the application's design is modular and maintainable.
  + **images/**: Stores image assets like product images, and icons used throughout the application, centralizing all visual elements for easy management.
  + **App.js**: The root component where routing and major layout structures are defined.
  + **index.js**: Entry point of the application where ReactDOM renders the app into the DOM.
* **package.json**:
  + This file manages project dependencies, scripts, and other configurations for the frontend application.



*Figure 3.1 Frontend Structure*

**Server: Node.js Backend Structure**

The backend of Book Store is built using **Node.js** and follows a simple but effective folder structure for API development:



*Figure 3.2.Backend Structure*

* **models/**:
  + Contains the Mongoose models for interacting with the MongoDB database. For example, User.js, Product.js, Order.js, etc., define the schema for respective entities in the database.
* **routes/**:
  + Houses route files where the API endpoints are defined. For example, userRoutes.js handles user authentication routes, productRoutes.js manages product-related endpoints, and orderRoutes.js handles order processing.
* **middleware/**:
  + Includes middleware functions used in the application. For example, authMiddleware.js verifies the JWT token for protected routes, and errorHandler.js handles errors globally throughout the backend.
* **index.js**:
  + The entry point of the backend application where the Express server is set up, middleware is applied, and routes are initialized.
* **package.json**:
  + This file manages backend dependencies and includes scripts for running the server.

**6. Running the Application**

**Frontend**

1. **Navigate to the Frontend Directory**
   * Open a terminal window on your system.
   * Use the cd (change directory) command to navigate to the frontend folder, which contains the frontend code for the application:

Code

cd C:\Users\BookStore\frontend

1. **Install Frontend Dependencies**
   * Before starting the frontend server, ensure all required dependencies are installed. If this is your first time running the project or if dependencies have changed, use the following command:

Code

npm install

* + This will install all necessary packages listed in the package.json file, including React and other dependencies.

1. **Start the Frontend Server**
   * After installing the dependencies, start the React development server using:

Code

npm start

* + This will compile the React application and start a local development server.

1. **Access the Application**
   * Once the server is running, open a web browser and navigate to:

Code

http://localhost:3000

* + The frontend application will be accessible at this URL, displaying the BookStore user interface.

**Backend**

1. **Navigate to the Backend Directory**
   * Open a terminal window.
   * Use the cd command to navigate to the backend folder, where the backend code resides:

Code

cd C:\Users\BookStore\backend

1. **Install Backend Dependencies**
   * If this is the first time running the backend server, install the required dependencies by running:

Code

npm install

* + This will install necessary backend packages, such as Express, Mongoose, and others listed in the package.json file.

1. **Start the Backend Server**
   * Once the dependencies are installed, start the backend server using:

Code

node index.js

* + This will initialize the backend server and connect it to the database.

1. **Access the Backend**
   * The backend server will now be running and accessible via:

Code

http://localhost:3001

* + This URL handles all API requests, supporting user authentication, book inventory management, and order processing.

**7. API Documentation  
1. Authentication**

* **POST /api/user/register**  
  **Description**: Registers a new user.  
  **Request Body**:

Code

{

"username": "john\_doe",

"email": "john.doe@example.com",

"password": "password123",

"usertype": "customer"

}

**Response**:

* + **200 OK**: User registration successful with user details.
  + **400 Bad Request**: Invalid or missing parameters.
* **POST /api/user/login**  
  **Description**: Logs in a user and generates a JWT token for authorization.  
  **Request Body**:

Code

{

"email": "john.doe@example.com",

"password": "password123"

}

**Response**:

* + **200 OK**: Returns a JWT token.
  + **401 Unauthorized**: Invalid credentials.
* **POST /api/forgot/forgot-password**  
  **Description**: Sends a password reset link to the user's email.  
  **Request Body**:

Code

{

"email": "john.doe@example.com"

}

**Response**:

* + **200 OK**: Password reset link sent successfully.
  + **400 Bad Request**: Email not found in the system.

**2. User Management**

* **GET /api/users**  
  **Description**: Retrieves a list of all users in the system.  
  **Response**:

Code

[

{

"userId": "1234567890",

"username": "john\_doe",

"email": "john.doe@example.com",

"usertype": "customer"

},

{

"userId": "0987654321",

"username": "admin\_user",

"email": "admin@example.com",

"usertype": "admin"

}

]

**3. Book Management**

* **GET /api/books/fetch-books**  
  **Description**: Retrieves a list of all books with optional filtering by genre, price range, and author.  
  **Query Parameters**: genre, priceMin, priceMax, author  
  **Response**:

Code

[

{

"title": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"genre": "Fiction",

"description": "A classic novel set in the 1920s.",

"mainImg": "<main-img-url>",

"price": 500,

"rating": 4.8

}

]

* **GET /api/books/fetch-books/**

**Description**: Retrieves details of a specific book by its ID.  
**Path Parameter**: id  
**Response**:

Code

{

"title": "To Kill a Mockingbird",

"author": "Harper Lee",

"genre": "Classic",

"description": "A novel about racial injustice in the Deep South.",

"mainImg": "<main-img-url>",

"price": 450,

"rating": 4.9

}

* **POST /api/books**  
  **Description**: Adds a new book.  
  **Request Body**:

Code

{

"title": "1984",

"author": "George Orwell",

"genre": "Dystopian",

"description": "A novel about totalitarianism.",

"mainImg": "<main-img-url>",

"price": 600,

"rating": 4.7

}

**Response**:

* + **200 OK**: Book added successfully.
* **PUT /api/books/update-book/**

**Description**: Updates details of an existing book.  
**Request Body**:

Code

{

"title": "1984",

"author": "George Orwell",

"genre": "Dystopian",

"description": "A revised description of the book.",

"mainImg": "<main-img-url>",

"price": 650,

"rating": 4.7

}

**Response**:

* + **200 OK**: Book updated successfully.

**4. Cart Management**

* **POST /api/cart/add-to-cart**  
  **Description**: Adds a book to the user's cart.  
  **Request Body**:

Code

{

"userId": "1234567890",

"bookId": "0987654321",

"quantity": 1

}

**Response**:

* + **200 OK**: Book added to cart.
* **GET /api/cart/fetch-cart/**

**Description**: Retrieves all books in the user's cart.  
**Path Parameter**: userId  
**Response**:

Code

[

{

"bookId": "0987654321",

"title": "The Great Gatsby",

"quantity": 1,

"price": 500

}

]

* **PUT /api/cart/update-quantity**  
  **Description**: Updates the quantity of a book in the cart.  
  **Request Body**:

Code

{

"id": "1234567890",

"quantity": 2

}

**Response**:

* + **200 OK**: Cart updated successfully.

**5. Order Management**

* **POST /api/order/place-order**  
  **Description**: Places an order for books in the cart.  
  **Request Body**:

Code

{

"userId": "1234567890",

"cartItems": [

{ "bookId": "0987654321", "quantity": 1 }

],

"address": "123 Main Street",

"pincode": "400001",

"paymentMethod": "Credit Card"

}

**Response**:

* + **200 OK**: Order placed successfully.
* **GET /api/order/fetch-orders/**

**Description**: Retrieves all orders for a specific user.  
**Path Parameter**: userId  
**Response**:

Code

[

{

"orderId": "111222333",

"books": [

{ "title": "1984", "quantity": 1, "price": 600 }

],

"status": "Shipped"

}

]

**6. Wishlist Management**

* **POST /api/wishlist/add**  
  **Description**: Adds a book to the user's wishlist.  
  **Request Body**:

Code

{

"userId": "1234567890",

"bookId": "0987654321"

}

**Response**:

* + **200 OK**: Book added to wishlist.
* **GET /api/wishlist/**

**Description**: Retrieves all books in the user's wishlist.  
**Path Parameter**: userId  
**Response**:

Code

[

{

"bookId": "0987654321",

"title": "The Great Gatsby",

"price": 500

}

]

* **POST /api/wishlist/remove**  
  **Description**: Removes a book from the wishlist.  
  **Request Body**:

Code

{

"userId": "1234567890",

"bookId": "0987654321"

}

**Response**:

* + **200 OK**: Book removed from wishlist.

**7. Search**

* **GET /api/search**  
  **Description**: Searches for books by title, author, or genre.  
  **Query Parameters**: query  
  **Response**:

Code

[

{

"bookId": "0987654321",

"title": "1984",

"author": "George Orwell",

"price": 600

}

]  
**8. Authentication**

The authentication and authorization process in the Book Store application is designed to securely verify user identity and manage access control using JSON Web Tokens (JWT), session storage, and password hashing.

**1. User Authentication with JWT Tokens**

* Upon successful login or registration, a **JSON Web Token (JWT)** is generated and returned to the client. This token is encoded with user-specific details (e.g., user ID and role) and a secret key to ensure authenticity.
* The token is stored in the client’s session storage and included in headers for all protected requests, allowing the server to verify user identity and permissions.

**2. Token-Based Authorization**

* **JWT tokens** are validated on each request to ensure only authenticated users can access restricted features. For example:
  + Customers can access their profiles, wishlists, and purchase histories.
  + Admin users can access the Admin Dashboard to manage inventory, orders, and users.
* The token contains encoded user information (like user ID and role), enabling **role-based authorization** to enforce access control across different parts of the application.

**3. Password Hashing**

* Passwords are securely hashed using **bcrypt** before being stored in the database, protecting user credentials from potential breaches.
* During login, the submitted password is hashed and verified against the stored hash to authenticate users securely.

**4. Session Management**

* The application uses **session storage** to manage active user sessions across browser tabs, ensuring users stay logged in until token expiry. Key user details like user ID, name, email, and role (e.g., customer or admin) are stored in session storage to provide a seamless experience during the session.
* Token expiry is managed to ensure that sessions automatically log out after a certain period, enhancing security and preventing unauthorized access.

**9.UserInterface**The User Interface (UI) of the **BookStore application** is designed to provide an intuitive and engaging experience for both customers and administrators. Built using React, the UI ensures responsiveness, smooth interactions, and a user-friendly environment for browsing, purchasing, and managing books. It is structured to guide users effortlessly from exploring collections to completing purchases while equipping admins with tools to manage the platform effectively.

**1. Customer Experience**

The **BookStore UI** for customers is designed for simplicity and seamless navigation.

* The **homepage** features a clean layout with a carousel banner showcasing featured books, a search bar, and book cards displaying essential details like titles, authors, genres, and prices.
* Customers can filter books by genre, author, or price range. Each book card links to a **detailed page** where users can read a synopsis, view reviews, and add the book to their cart.
* The **wishlist feature** allows customers to save their favorite books for future purchases.
* The **cart page** enables users to modify quantities, remove books, and proceed to checkout.
* The **profile page** lets users manage personal information, view and track order statuses, and access their wishlist and cart for a seamless shopping experience.

**2. Admin Interface**

The **Admin Dashboard** provides a comprehensive interface for managing the platform's content and users.

* Admins can access sections for managing books (add, update, delete), monitoring and processing orders (e.g., Placed, In-transit, Delivered, Canceled), and managing customer information.
* A detailed form allows admins to add new books, including fields for titles, authors, genres, prices, and cover images.
* The dashboard also includes tools for updating inventory and handling order statuses, ensuring admins maintain full control over operations while streamlining workflows.

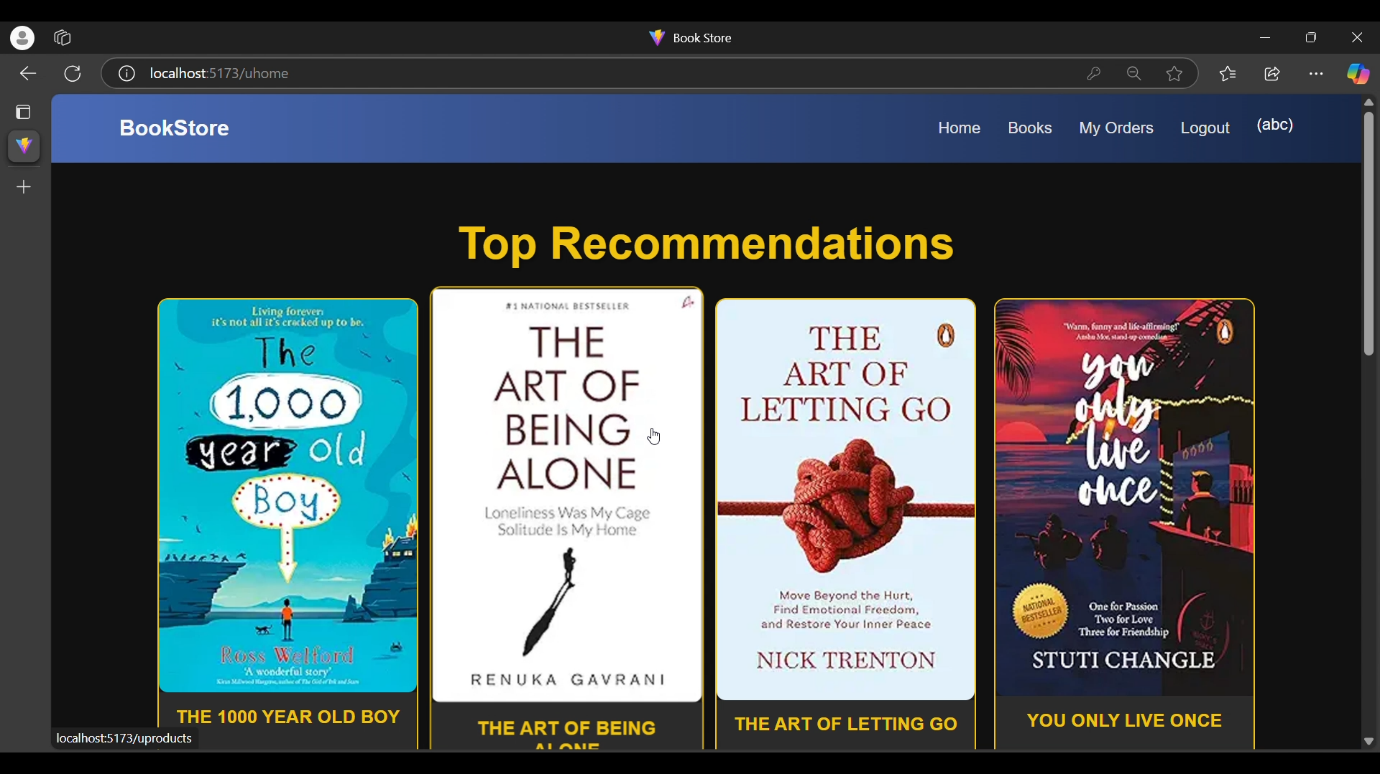
**3. Responsiveness and Aesthetics**

The UI is fully responsive, adapting seamlessly to various screen sizes, from desktops to mobile devices.

* The design leverages **Bootstrap** and custom CSS to provide a consistent and polished appearance across all devices, ensuring a smooth user experience whether accessed on a large monitor or a small mobile screen.
* The aesthetic emphasizes modern, clean visuals with a focus on readability and ease of navigation, making the platform both functional and visually appealing.

This design ensures that the **BookStore application** is user-centric, offering an enjoyable browsing and purchasing experience for customers and an efficient management system for administrators.

**Screenshots:**

****

**10. Testing**The testing strategy for the Book Store application focuses on ensuring the functionality, security, and performance of the application through API testing using Postman. Postman is utilized for testing the backend API endpoints and verifying the responses to ensure they meet expected behaviours.

1. **API Testing with Postman**:

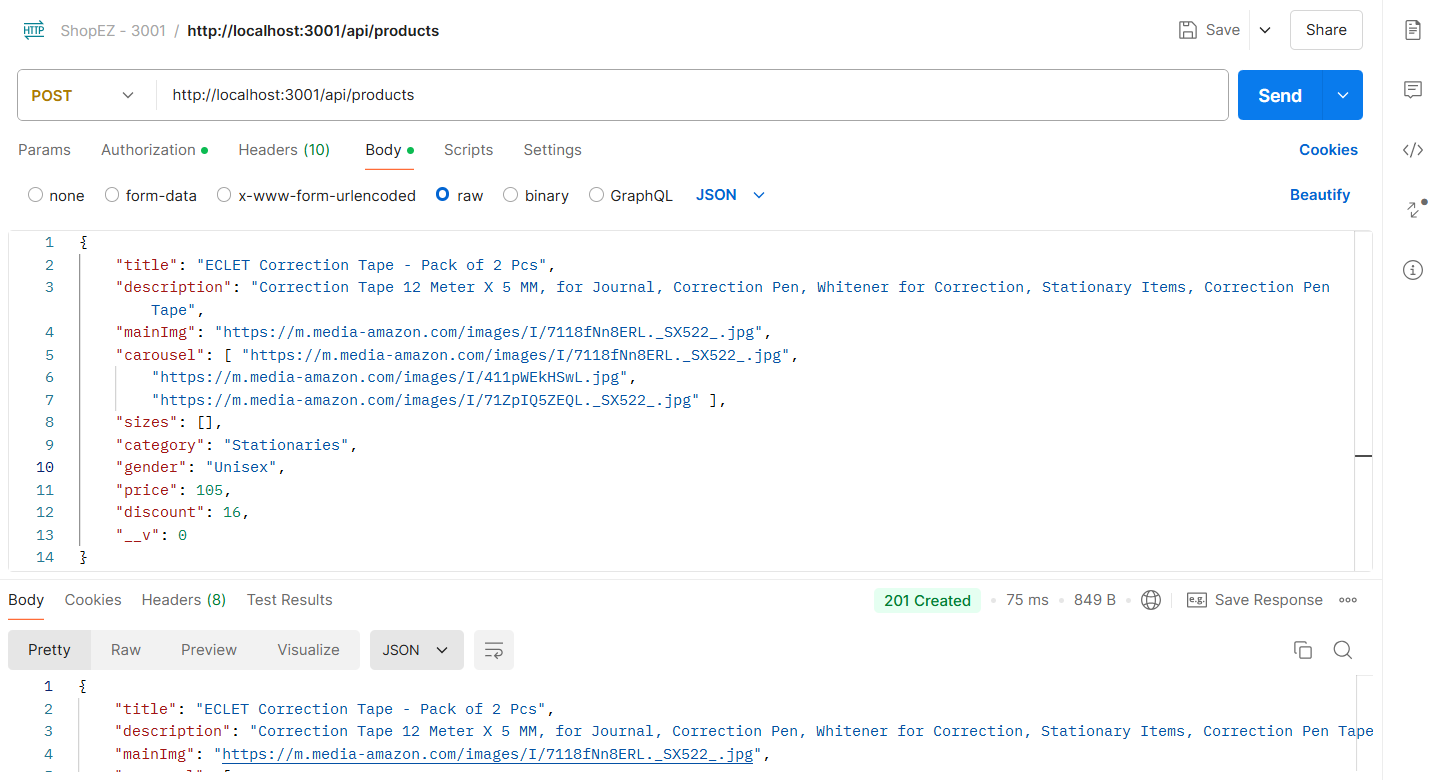
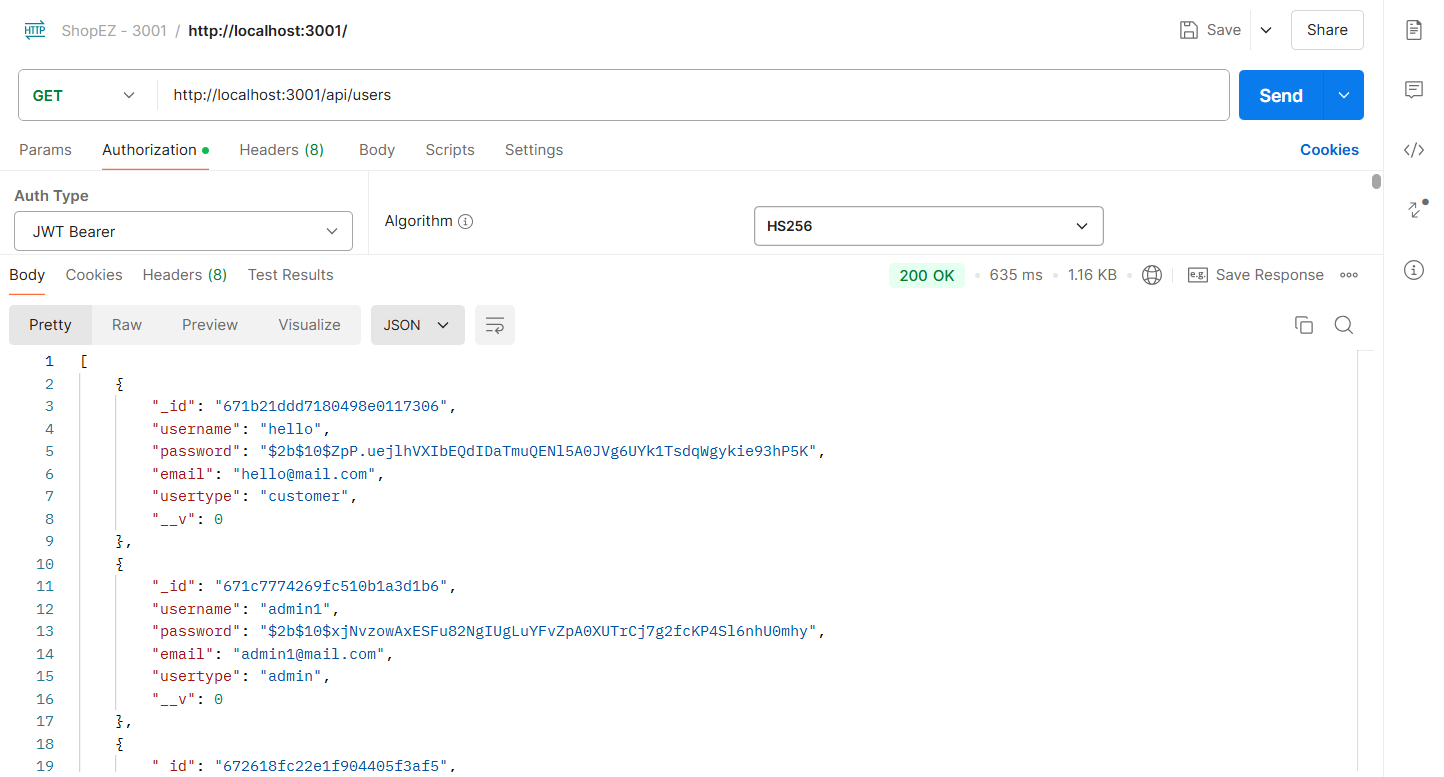
* Postman is used to test all API endpoints, including user authentication, product management, order processing, and cart functionalities. Each endpoint is tested by sending various requests (GET, POST, PUT, DELETE) with different parameters to verify the accuracy and correctness of the responses.
* Authentication endpoints are tested by sending login credentials, checking for correct token generation, and validating token-based access control.
* Product-related endpoints (such as adding, updating, and retrieving products) are tested to ensure the correct flow of data and proper error handling.
* Order and cart management endpoints are also tested to ensure that actions such as adding items to the cart, placing orders, and updating order status are functioning correctly.

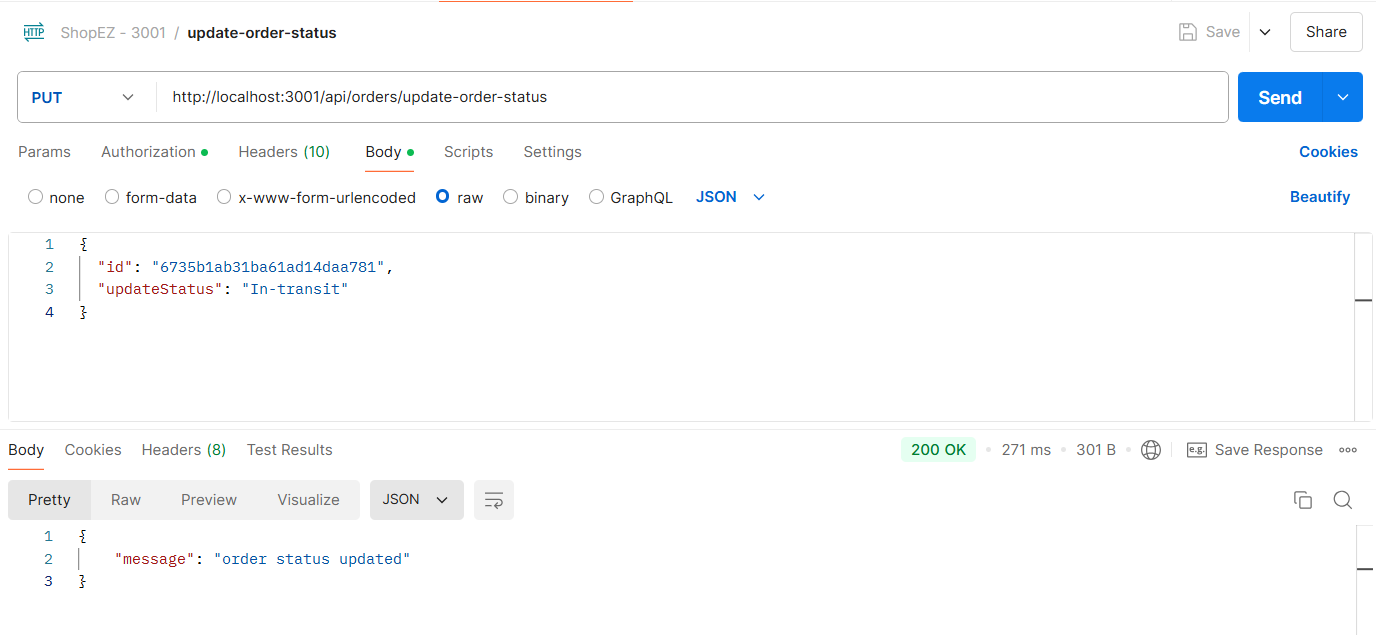
1. **Error Handling and Validation**:

* Responses are tested for proper status codes (200, 400, 401, 404, etc.), ensuring that the system handles errors correctly and returns appropriate messages when invalid data is sent or when users try to access restricted resources without proper authorization.
* Test cases are designed to cover both positive and negative scenarios, such as valid inputs and edge cases like missing or incorrect parameters.

1. **Test Automation and Repeatability**:

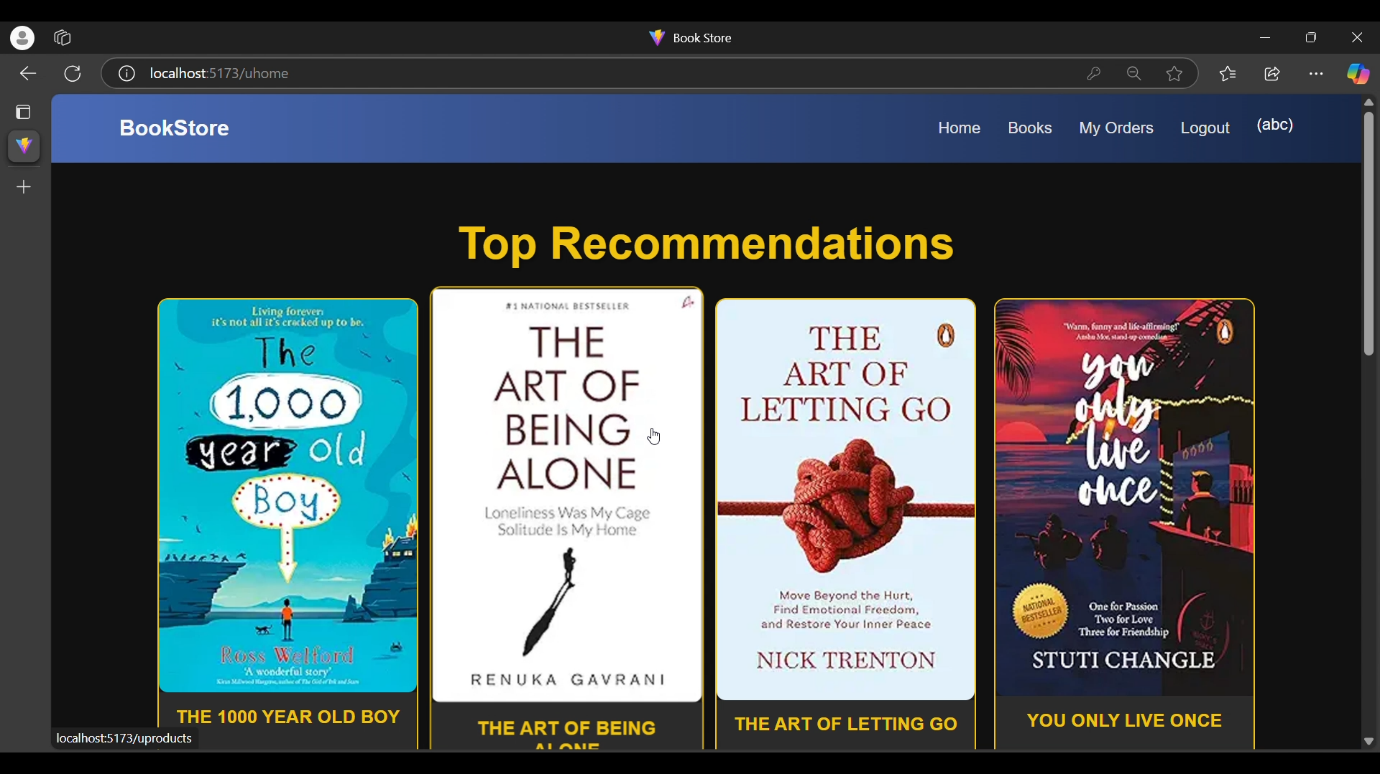
* Postman collections are used to automate and group tests, allowing for easy repeatability of test cases.
* This ensures consistency when validating changes made to the application.

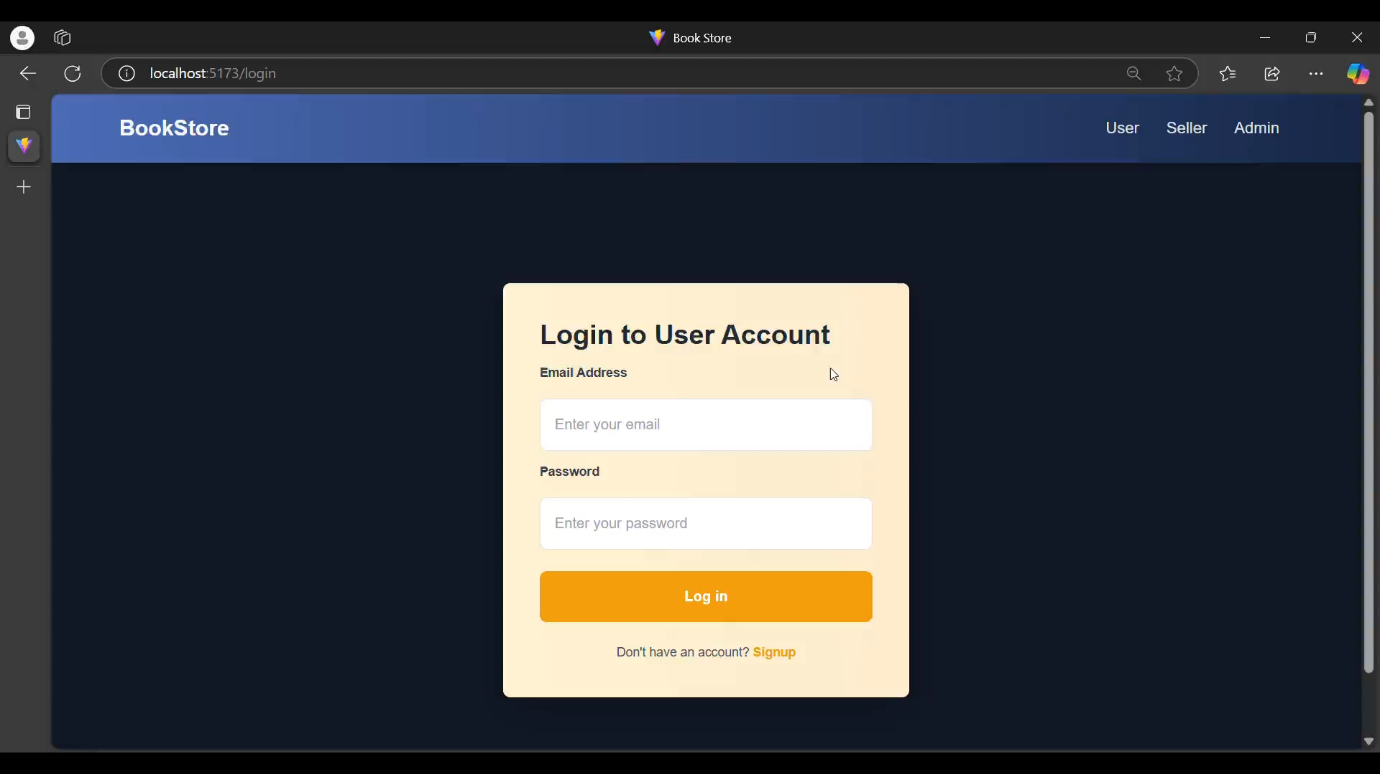


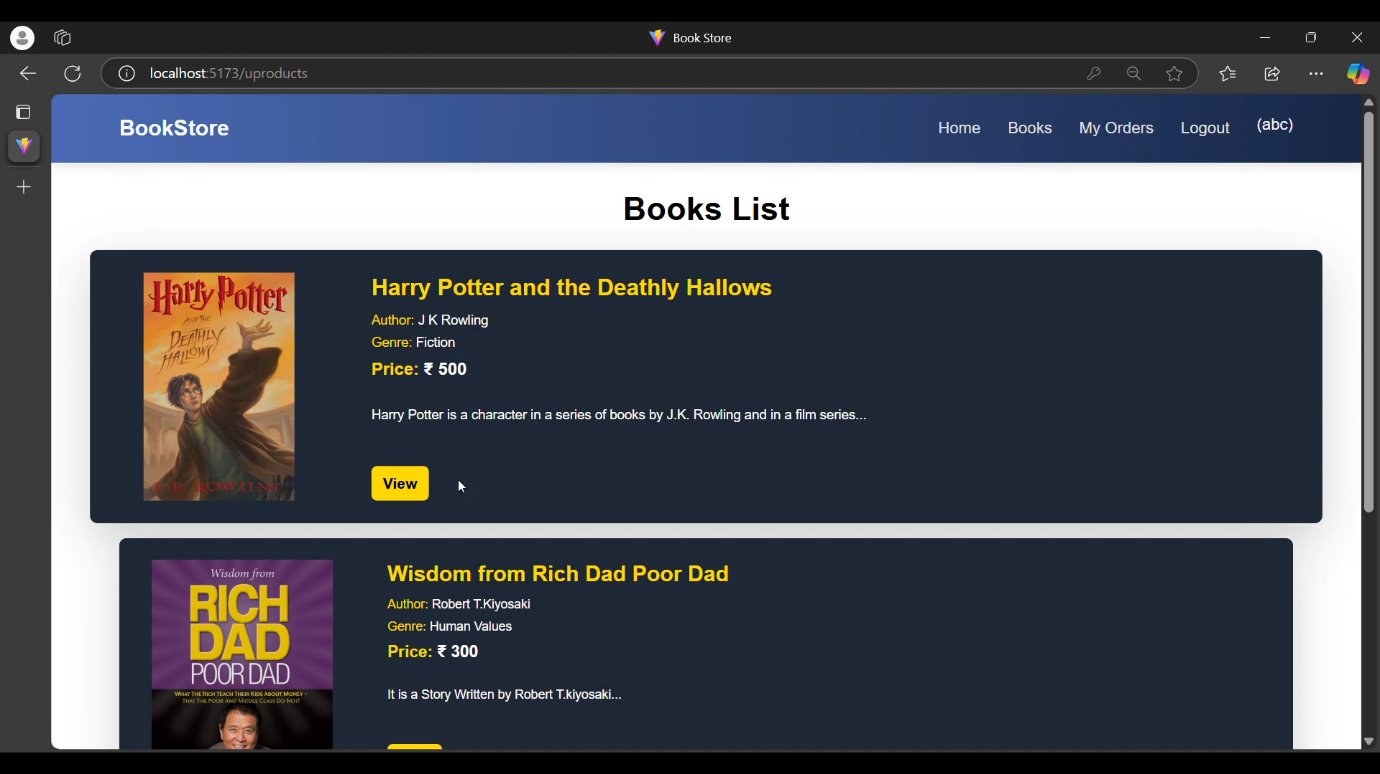


**11. Screenshots or Demo**

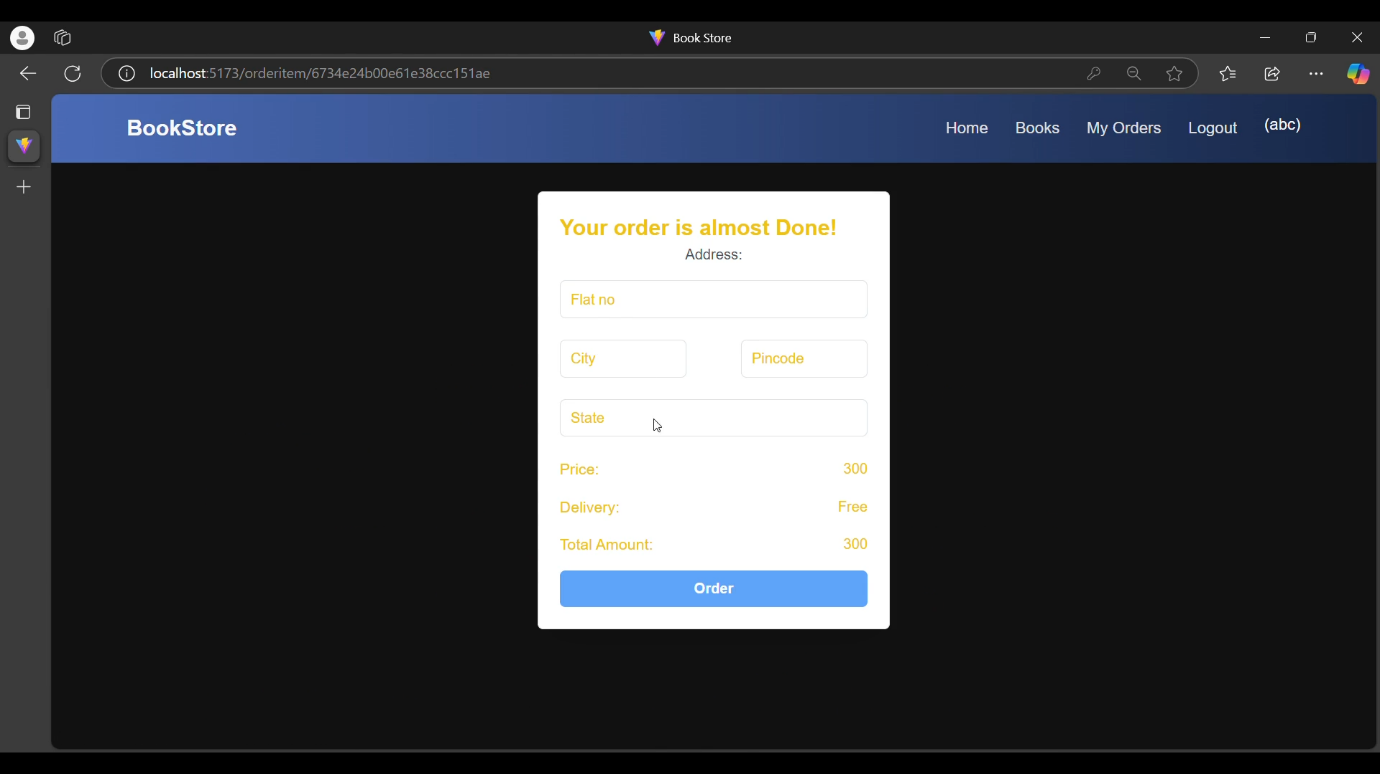
* Provide screenshots or a link to a demo to showcase the application.

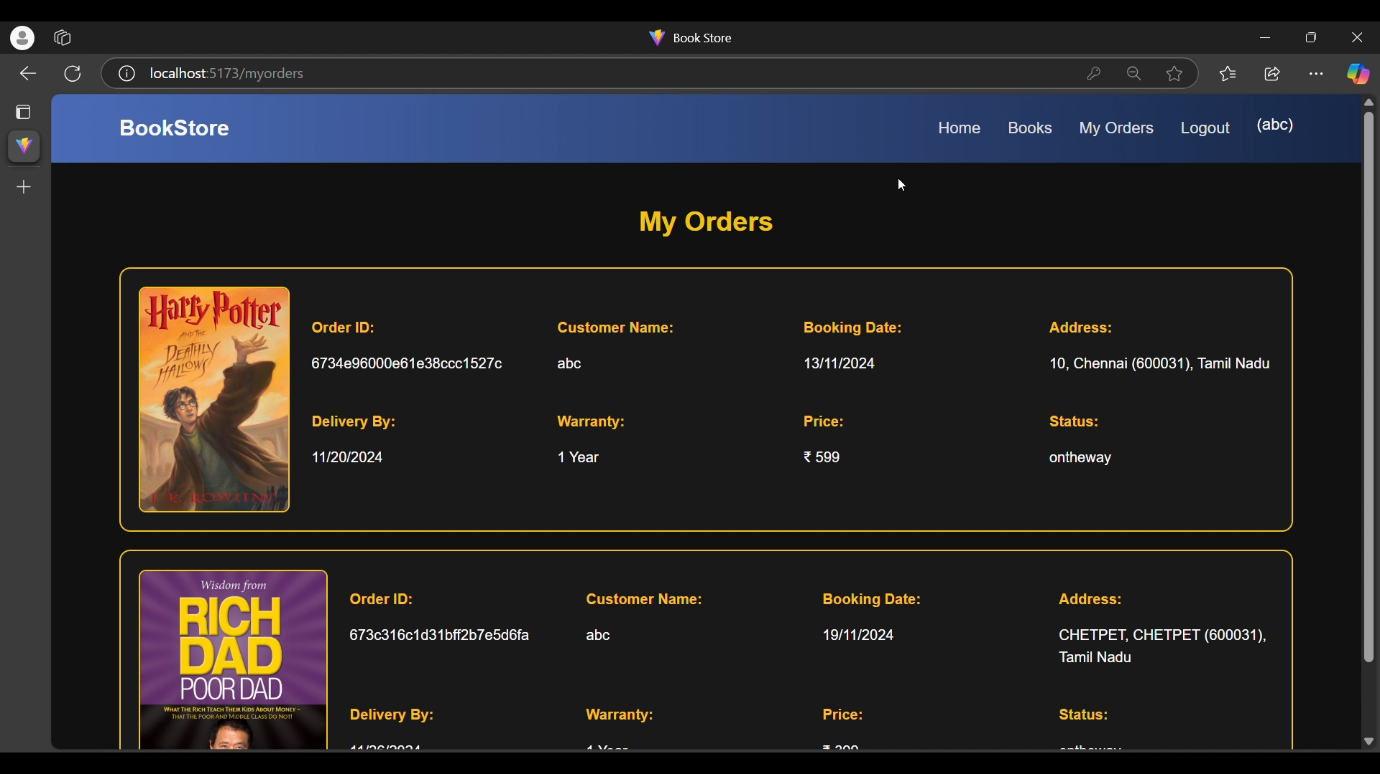
**Home Page**

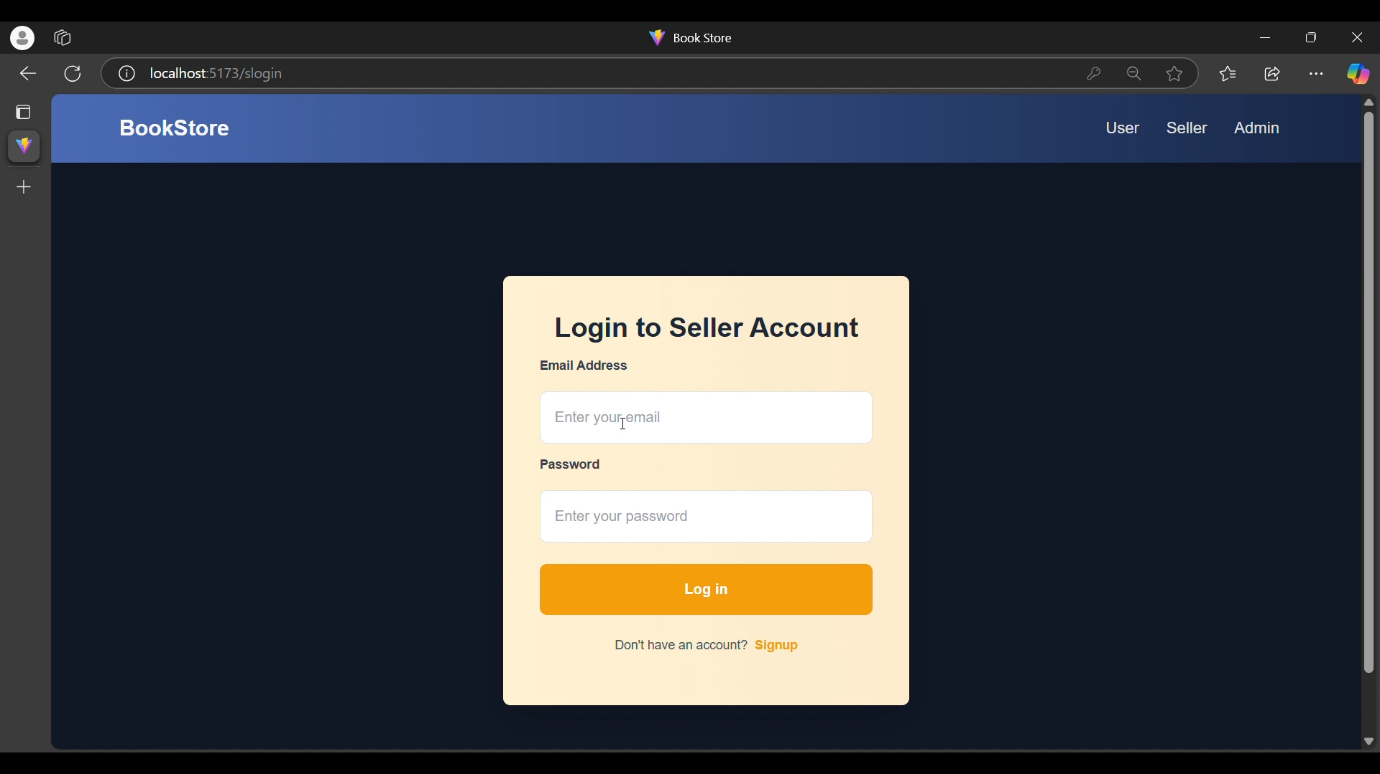
**Authentication - User Login**

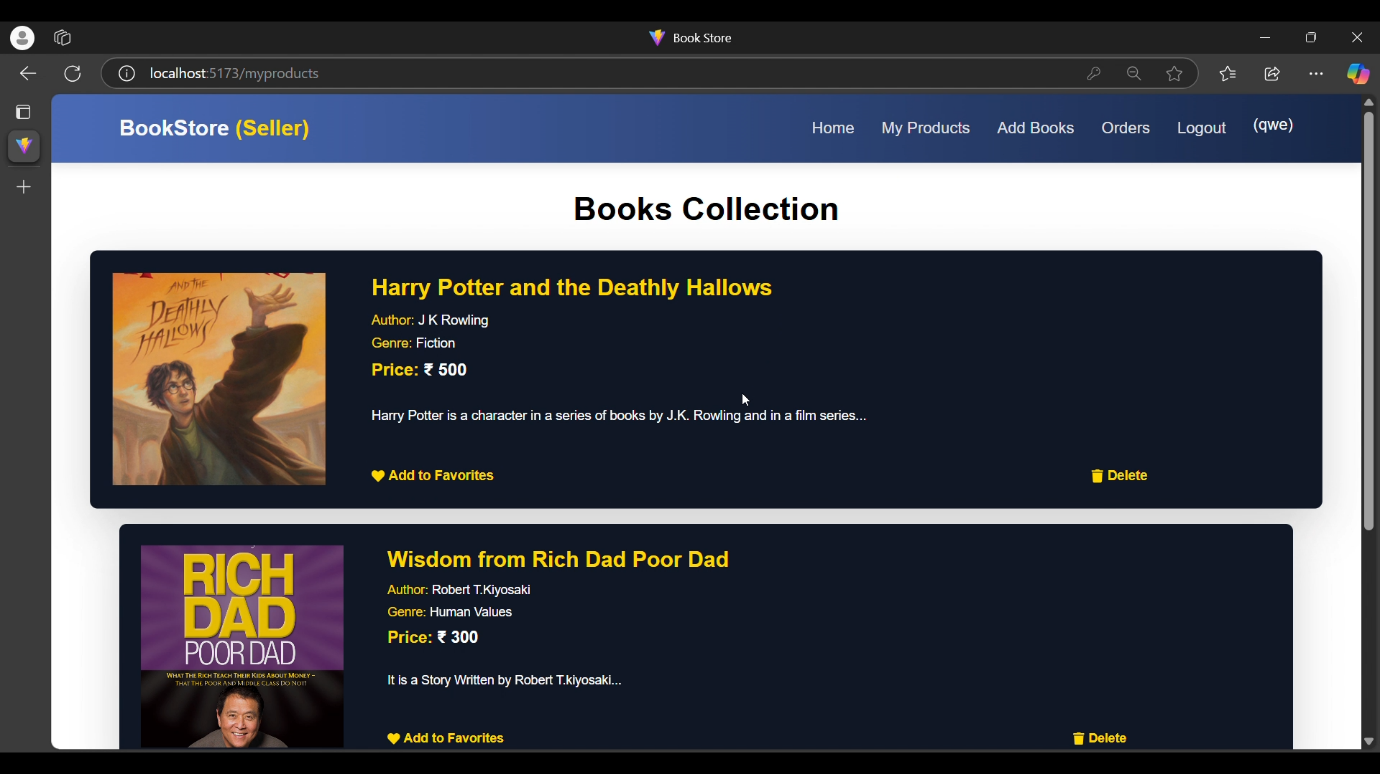
**Book List**

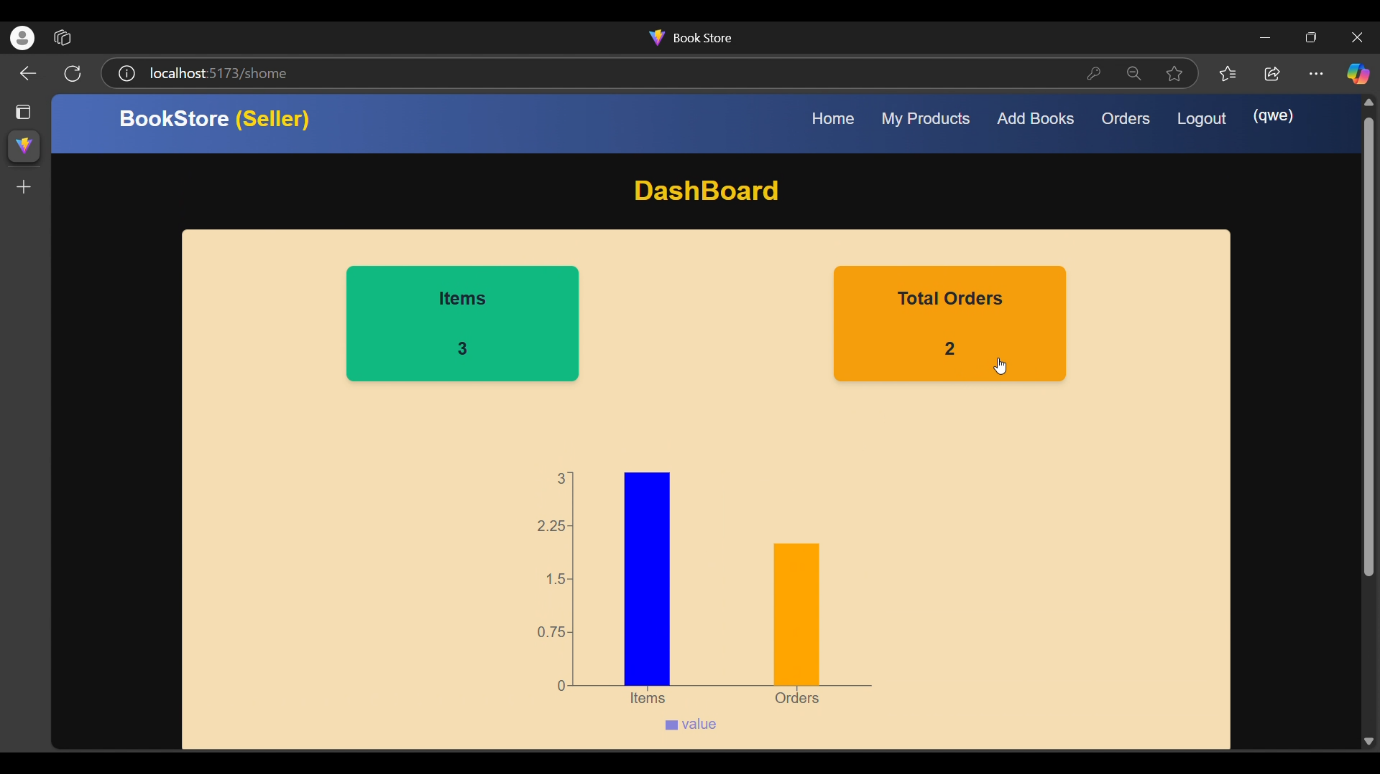
**Particular Book** **Details**

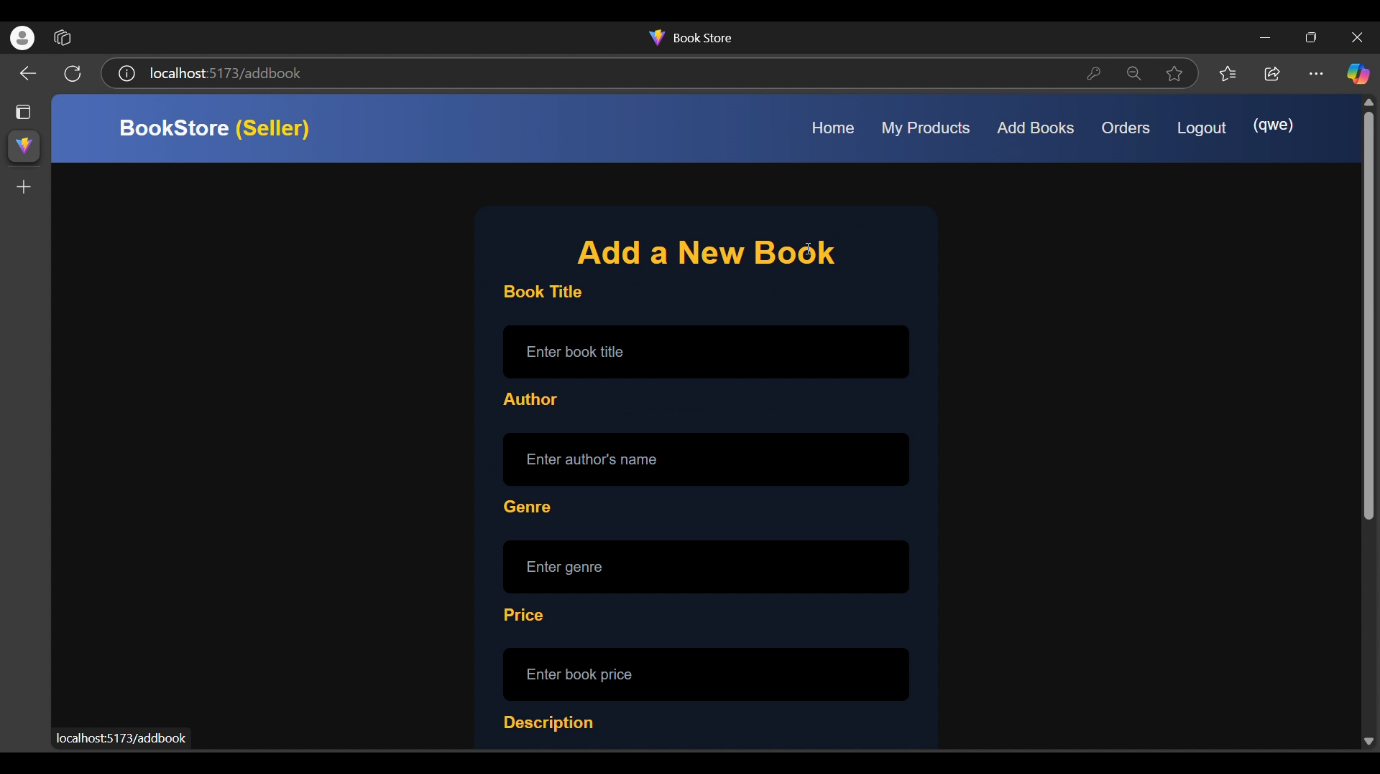
**Book Order Details**

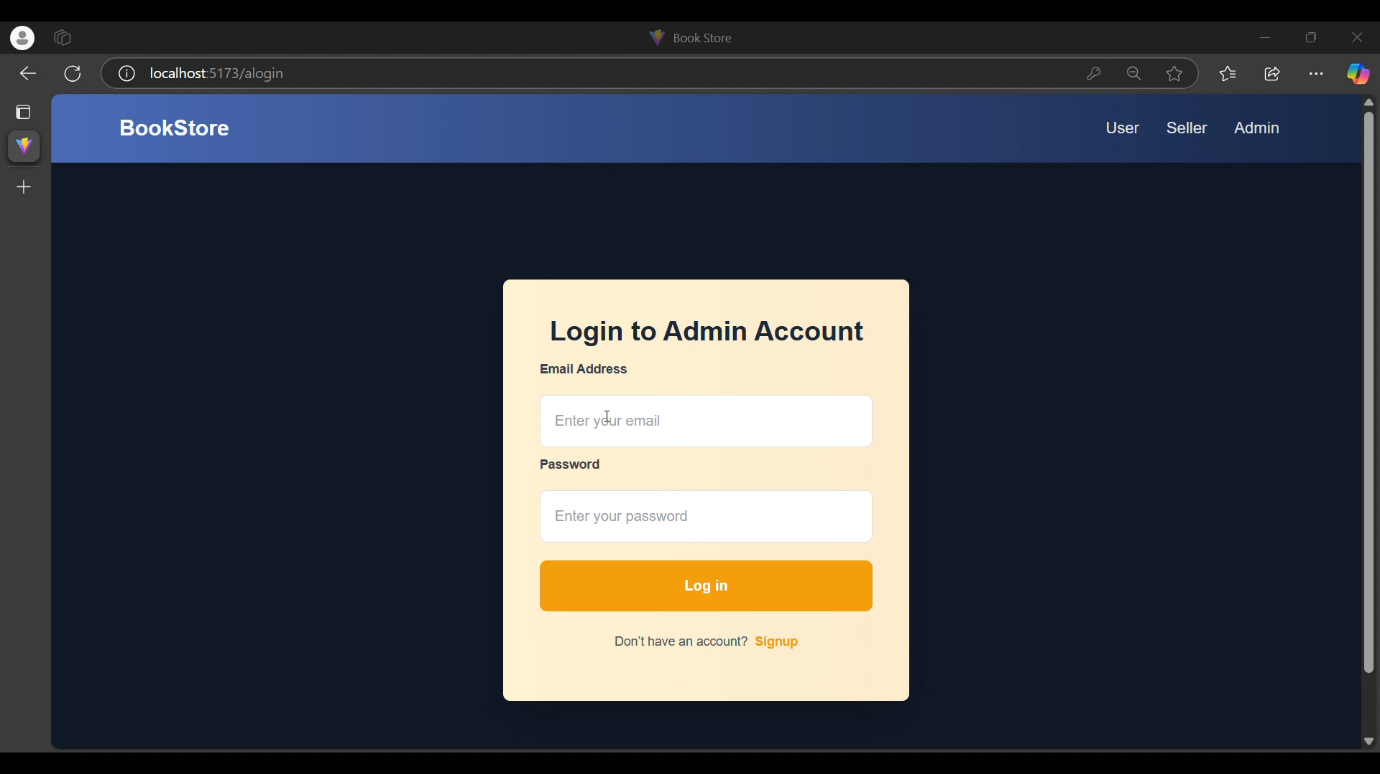
**My Orders**

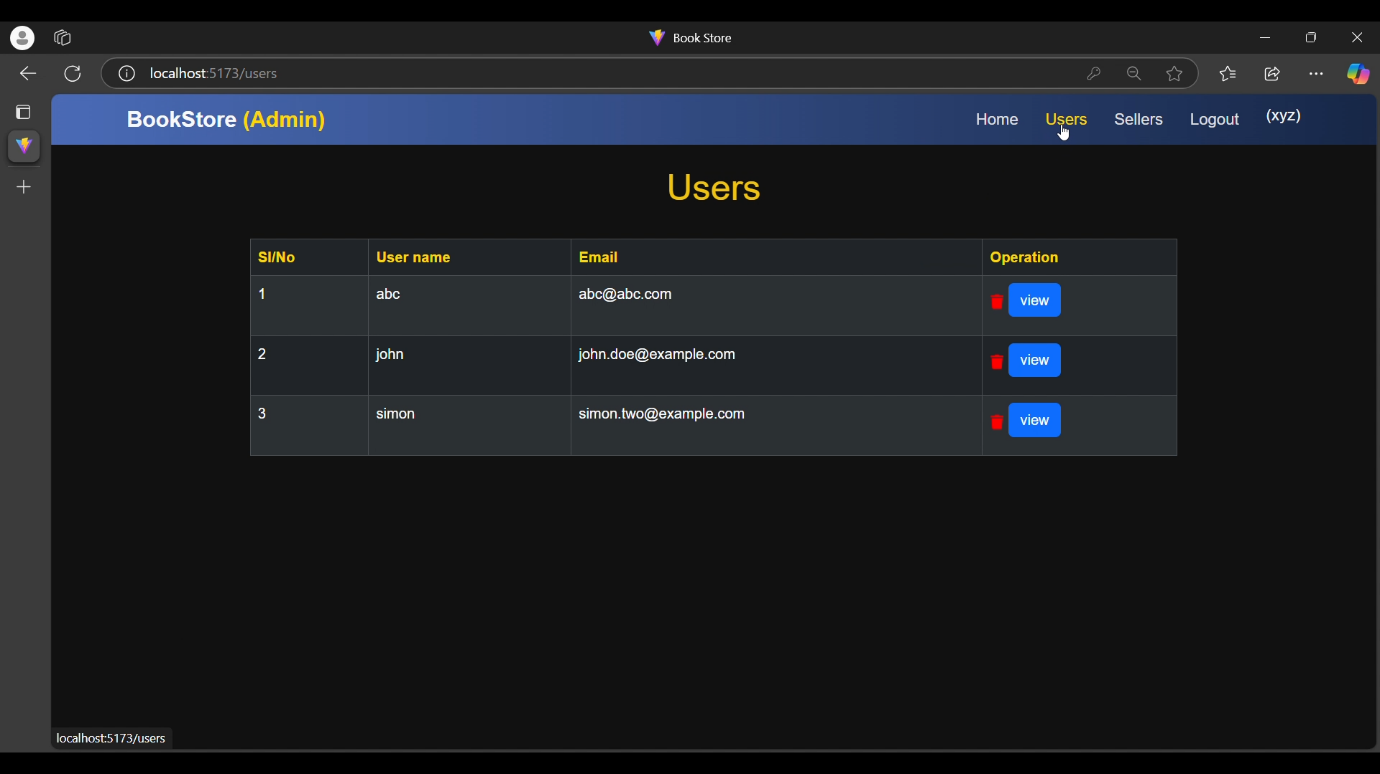
**Seller Login**

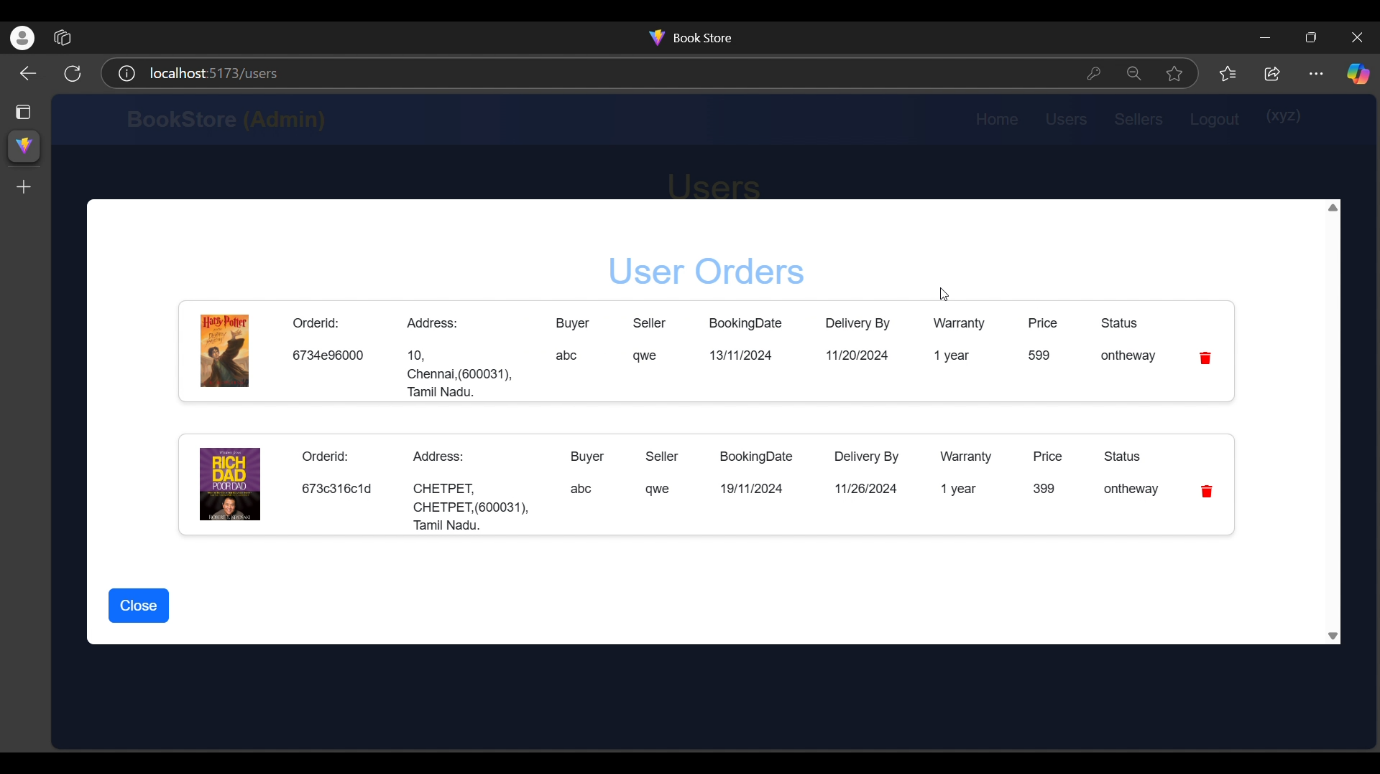
**Seller Book Collections**

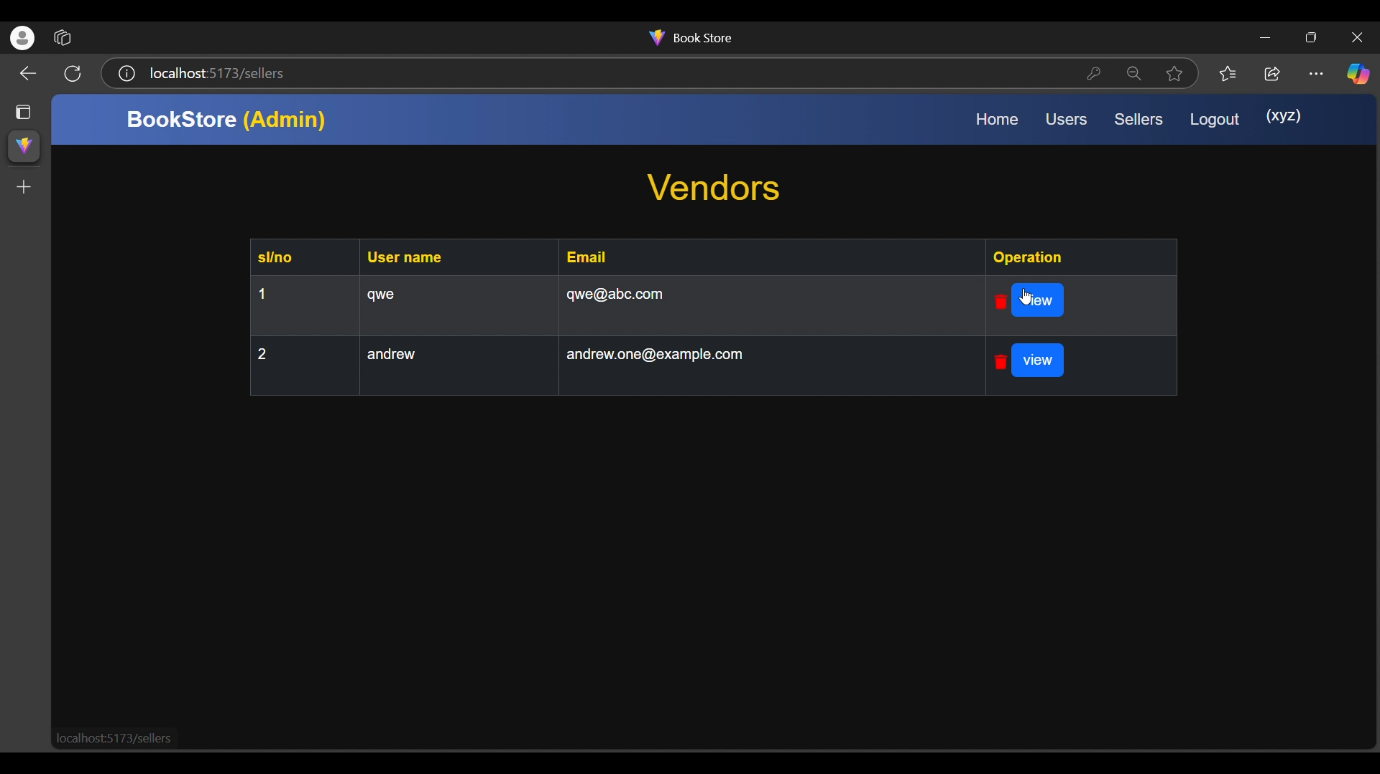
**Seller Dashboard**

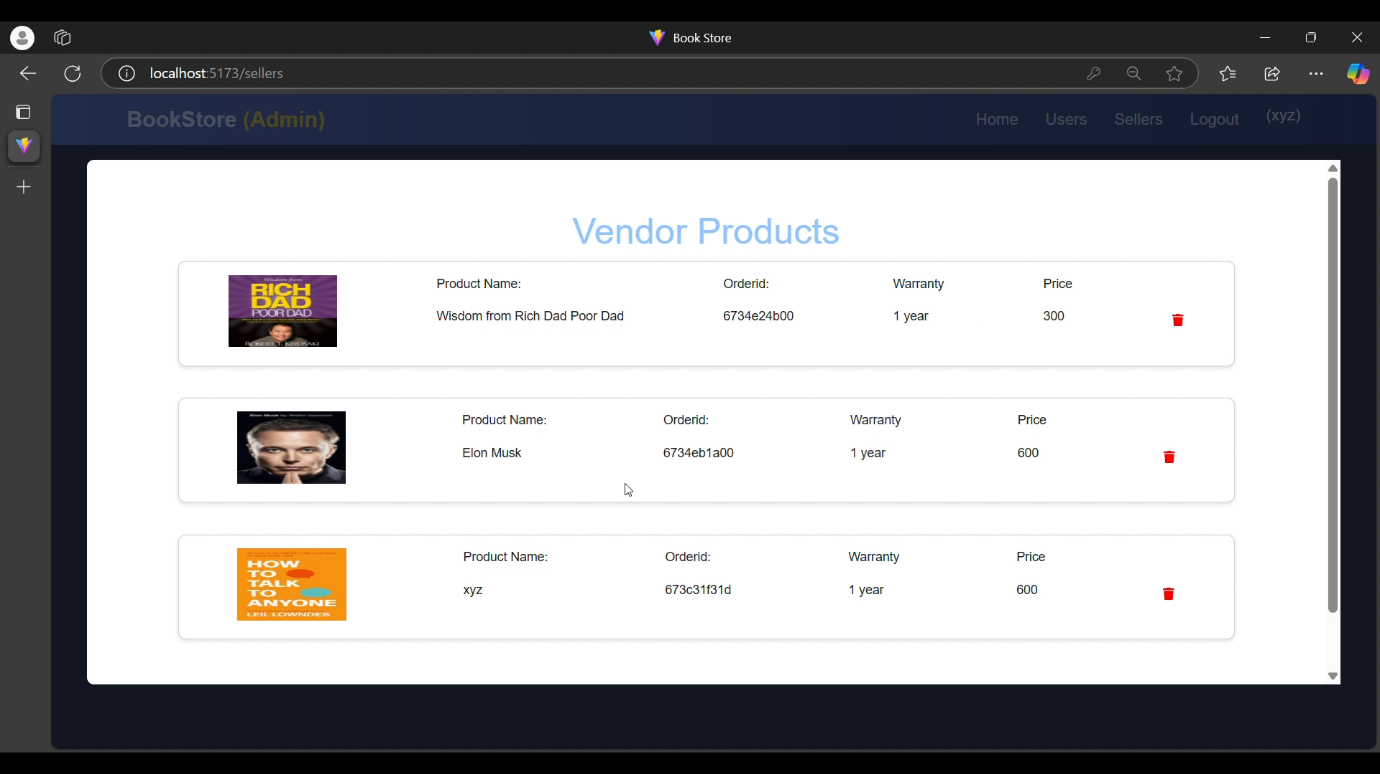
**Adding New Book**

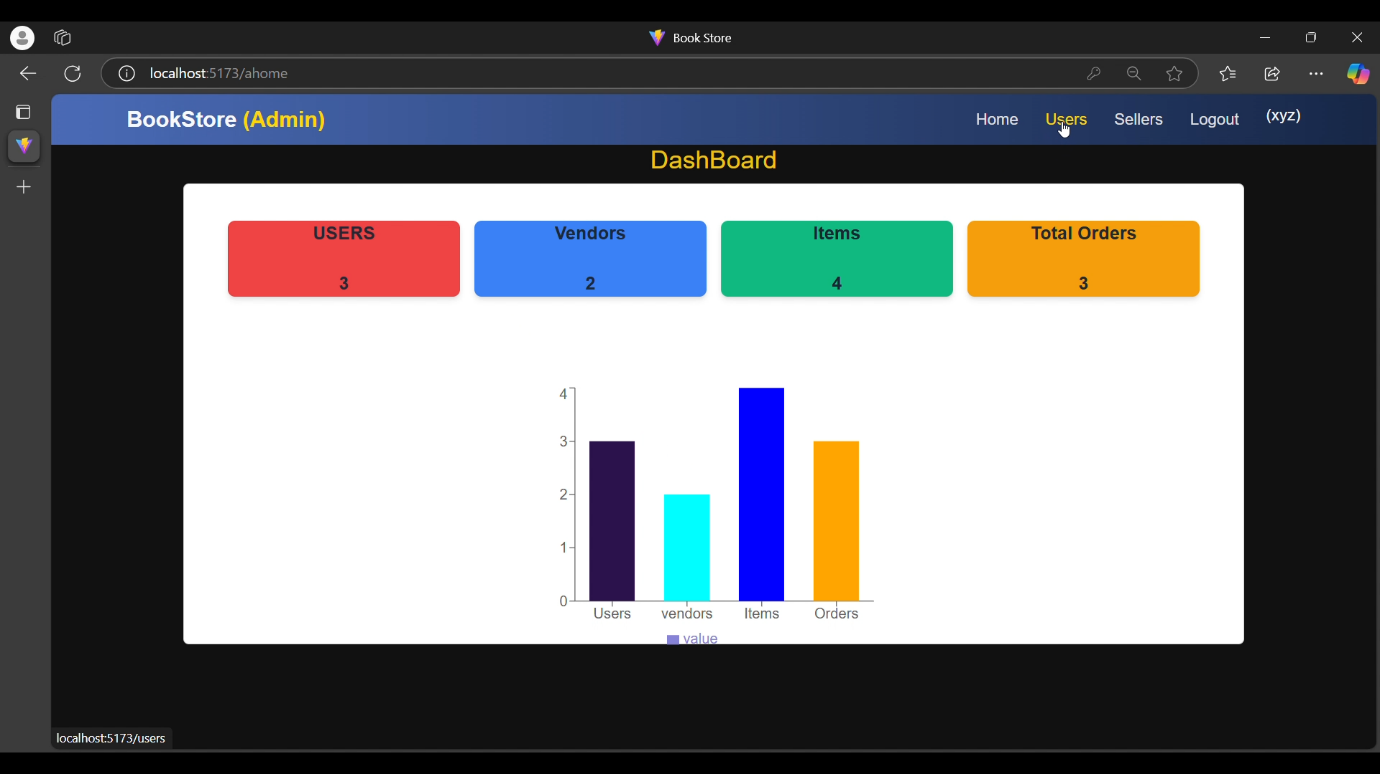
**Admin Login**

**User Details**

****

****

****

**Admin Dashboad**

**Demo Video Link:** [**Book\_Store.mp4**](https://github.com/Sashong17/Naan_Mudhalvan_MERN_Book_Store/blob/master/Book_Store.mp4)

**12. Known Issues**1. **Account Recovery Process**

* **Description:** The account recovery process does not currently include a verification step to confirm the validity of the email address before proceeding with recovery instructions.
* **Impact:** If a user mistypes their email address, they will not receive any feedback, potentially causing confusion as the recovery link may not reach them.

2. **Cross-Browser Compatibility**

* **Description:** Certain features, like the book preview carousel and zoom functionality, may not display or work properly in all browsers, particularly older versions of Internet Explorer or Safari.
* **Impact:** Users on specific browsers may encounter layout or functionality issues, leading to a suboptimal browsing experience.

3. **Mobile Responsiveness**

* **Description:** Some pages, such as the **Admin Dashboard** and the **All Books** view, are not fully responsive on smaller screens or devices.
* **Impact:** Mobile users may face issues with misaligned pages, broken layouts, or UI components when accessing the app on smaller screens.

4. **Authentication Token Expiry Handling**

* **Description:** The system inconsistently handles token expiry, which can result in users remaining logged in after their session expires or requiring a manual refresh to reauthenticate.
* **Impact:** Users may experience irregularities in session management, such as unexpected logouts or challenges accessing features post-token expiry.

5. **Admin Book Update Issues**

* **Description:** When admins update book details (e.g., title, price, description), the changes are not always immediately reflected in the Customer view.
* **Impact:** Admins may find that updates are delayed or not visible right away, causing potential confusion during inventory management.

These issues are being actively worked on and will be addressed in future updates to improve the overall user experience and functionality.

**13. Future Enhancements**

1. **Marketplace Integration**
   * Future versions of BookStore could include third-party seller integration, allowing external vendors to list their books. This would expand the platform’s catalog, offering customers a wider variety of titles and genres.
2. **AI-Driven Personalization**
   * Incorporating AI to provide personalized book recommendations based on user reading preferences, browsing history, and purchase patterns could enhance customer satisfaction and drive sales.
3. **Augmented Reality (AR) Features**
   * Integrating AR technology could allow users to preview books virtually, such as flipping through sample pages or viewing bookshelves with selected titles, enhancing the browsing experience.
4. **Subscription-Based Models**
   * Offering subscription services for book clubs, monthly book deliveries, or access to exclusive content could create consistent revenue streams and boost customer loyalty.
5. **Global Expansion**
   * The platform could broaden its reach by supporting multiple languages and currencies, making it more accessible to international readers and publishers.
6. **Advanced Analytics for Admins**
   * Providing detailed analytics and insights for admins—such as reader preferences, demand forecasting for popular titles, and sales trends—could improve inventory management and strategic decision-making.
7. **Sustainability Initiatives**
   * Introducing eco-friendly book options, such as recycled paper editions or digital formats, and promoting sustainable practices like book donations or exchanges could appeal to environmentally conscious customers, positioning BookStore as a socially responsible brand.

**CONCLUSION:**

The development of an online book store using the MERN stack (MongoDB, Express.js, React, and Node.js) highlights the effectiveness of modern web technologies in creating robust and dynamic applications. This project successfully demonstrates the integration of a scalable database, a responsive front-end, and a secure back-end to provide a seamless user experience. By incorporating features like browsing, searching, cart management, and secure checkout, the platform meets the core requirements of an e-commerce system. Additionally, the modular design of the MERN stack allows for future enhancements, such as personalized recommendations or advanced analytics. Overall, this project showcases the MERN stack's potential to deliver a comprehensive and efficient solution for online book sales.

**GITHUB LINK:** [**BOOK STORE**](https://github.com/Sashong17/Naan_Mudhalvan_MERN_Book_Store)