

BOOK STORE – MERN FULL STACK WEB APPLICATION DOCUMENTATION

INTRODUCTION

- Project Title: Book Store
- Team Members: Individual project (single member)

PROJECT OVERVIEW

Purpose

The **Book Store** is a MERN full stack web application designed to provide users with a smooth and user-friendly online platform to browse, purchase, and manage books. It allows customers to explore various book categories, view detailed book information, place orders securely, and track their purchases efficiently.

Features

1. **Comprehensive Product Catalog:**

The Book Store offers a wide collection of books including fiction, non-fiction, academic, and competitive exam books with details such as author name, price, description, ratings, and availability.

2. **Buy Now Button:**

Each book includes a “**Buy Now**” button that allows users to quickly proceed with the purchase.

3. Order Details Page:

After clicking **Buy Now**, users are redirected to the order details page where they can enter delivery address, payment details, and confirm the selected book.

4. Secure & Efficient Checkout:

The application ensures secure handling of user data and provides a fast and reliable checkout process.

5. Order Confirmation & Review:

After placing an order, users receive confirmation and can view complete order details including book information, payment status, and delivery address.

ARCHITECTURE

- Frontend (React JS)

The frontend is developed using **React JS** and includes components for **user authentication, book listings, cart management, user profile, and admin dashboard**. **React Router** is used for navigation and **Context API** is used for state management.

- Backend (Node.js + Express.js)

The backend is built using **Node.js and Express.js** and provides **RESTful APIs** for managing **users, books, cart, and orders**. It handles authentication, business logic, and admin operations.

- Database (MongoDB)

MongoDB is used to store application data, including **Users, Books, Cart, and Orders** collections. **Mongoose** is used for schema definition and database interactions.

SETUP INSTRUCTIONS

Prerequisites

- Node.js
- MongoDB (Local or MongoDB Atlas)
- Git

Installation

1. Clone the repository

```
git clone [your_repo_link]
```

2. Install backend dependencies

```
cd server
```

```
npm install
```

3. Install frontend dependencies

```
cd ../client
```

```
npm install
```

4. Create .env file in the /server folder

```
PORT=4000
```

```
MONGO_URI=your_connection_string
```

```
JWT_SECRET=your_secret_key
```

FOLDER STRUCTURE

Client (React Frontend)

client/

- └─ public/ # Static assets
- └─ src/ # Main source code
 - └─ components/ # Reusable UI components
 - └─ context/ # Global state management
 - └─ image/ # Images and icons
 - └─ pages/ # Application pages/screens
 - └─ styles/ # CSS files (App.css, etc.)
 - └─ App.js # Root component
 - └─ index.js # Entry point
 - └─ index.css # Global styles
 - └─ App.test.js # App tests
 - └─ reportWebVitals.js # Performance metrics
 - └─ setupTests.js # Test configuration
- └─ .gitignore
- └─ package-lock.json
- └─ README.md

Server (node.js + express.js)

```
server/  
| package.json  
| package-lock.json  
| index.js  
| server.js  
| .env  
|  
├-- controllers/  
├-- models/  
├-- routes/  
├-- middleware/  
└ config/
```

RUNNING THE APPLICATION

Frontend

- cd client
- npm start

Backend

- cd server
- npm start

API DOCUMENTATION

User APIs

- **POST /api/auth/register** – Register a new user
- **POST /api/auth/login** – User login
- **GET /api/auth/profile** – Get logged-in user profile (*protected*)

Product APIs

- **GET /api/books** – Get all books
- **GET /api/books/:id** – Get a single book by ID
- **POST /api/books** – Add a book (*seller only*)
- **PUT /api/books/:id** – Update a book (*seller only*)
- **DELETE /api/books/:id** – Delete a book (*seller only*)

Cart APIs

- **POST /api/cart/add** – Add a book to cart
- **GET /api/cart/:userId** – Get user cart
- **DELETE /api/cart/:itemId** – Remove a book from cart

Order APIs

- **POST /api/orders** – Place a new order
- **GET /api/orders/:userId** – Get orders of a user
- **GET /api/orders/details/:orderId** – Get order details

Admin APIs

- **POST /api/admin/login** – Admin login
- **GET /api/admin/users** – Get all users
- **GET /api/admin/orders** – Get all orders

AUTHENTICATION

- **JWT Generation:**

When a user logs in, the backend verifies the credentials and generates a JWT token.

- **Token Storage:**

The JWT token is stored on the frontend using **localStorage** or **sessionStorage**.

- **Protected Routes:**

For every protected API request, the token is included in the Authorization header as:

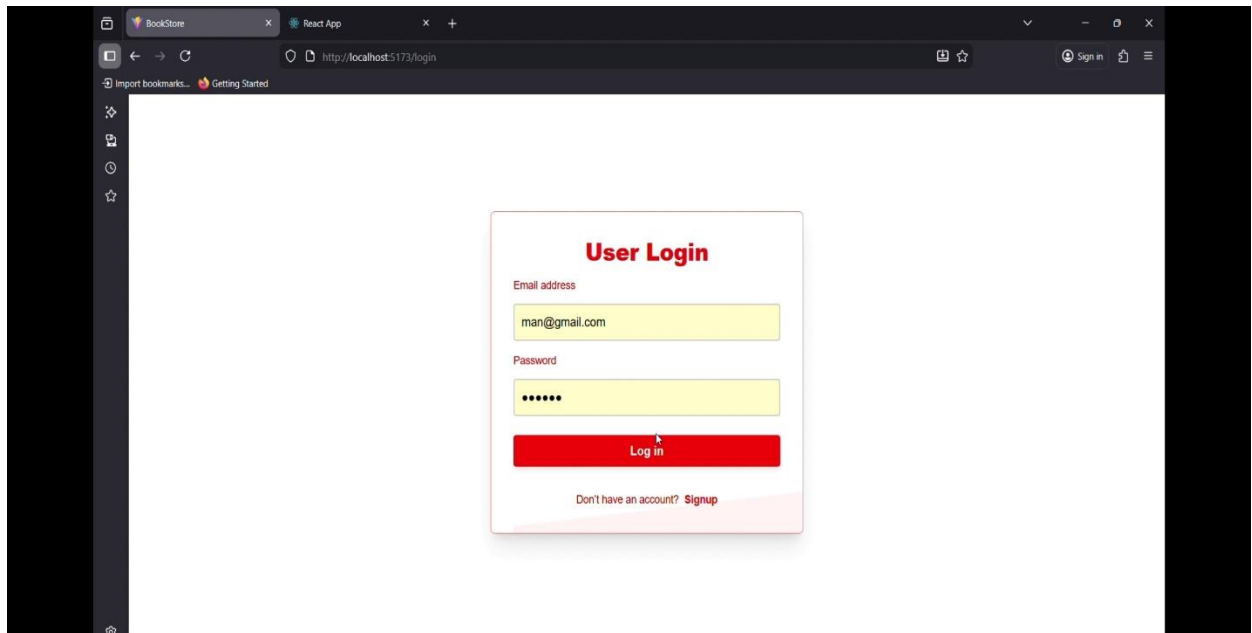
Authorization: Bearer <token>

- **Authorization:**

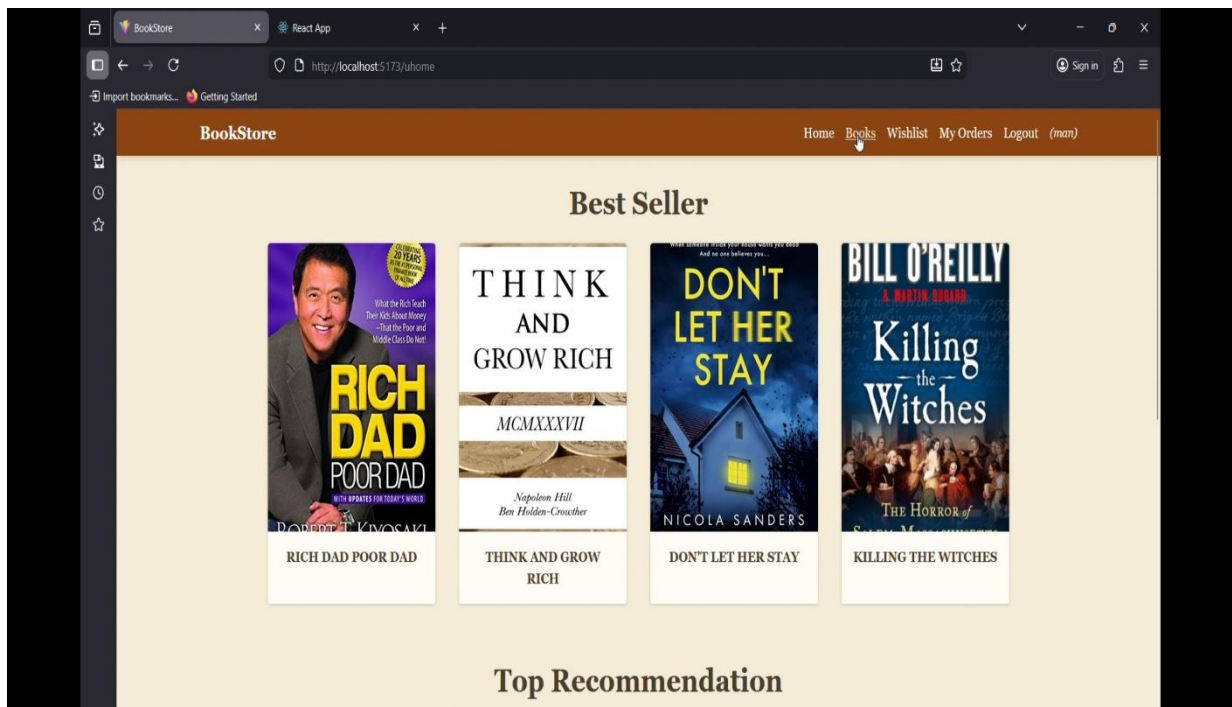
The backend validates the token before granting access to restricted routes such as user profile, cart, orders, and admin operations.

USER INTERFACE

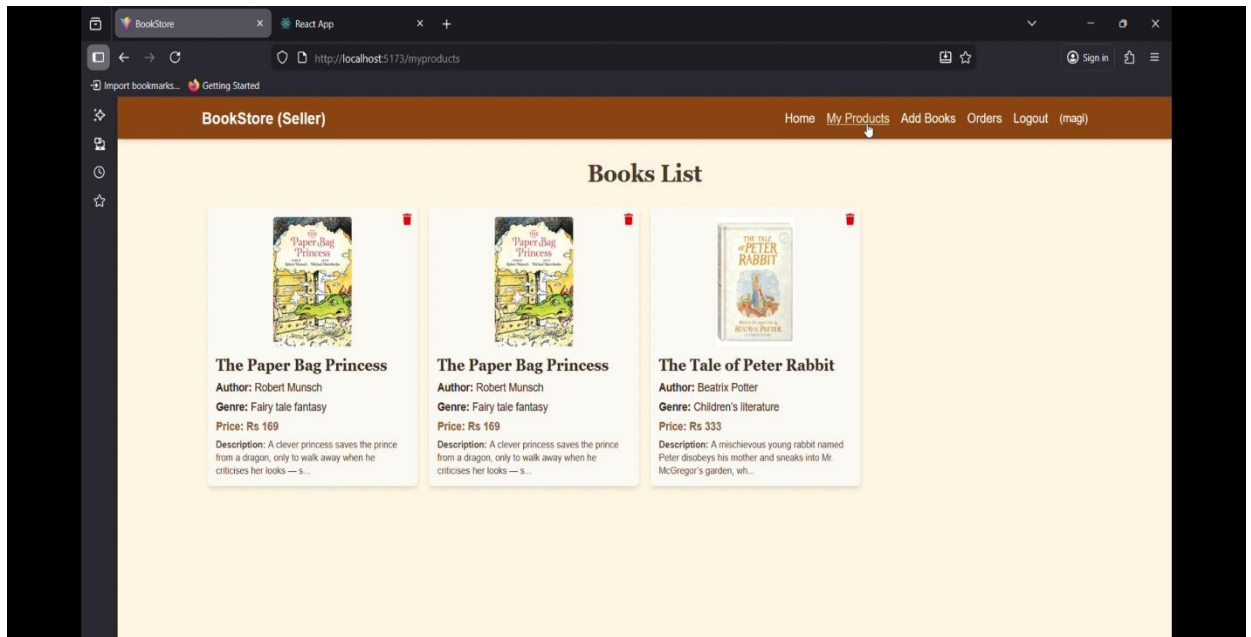
Signup Page



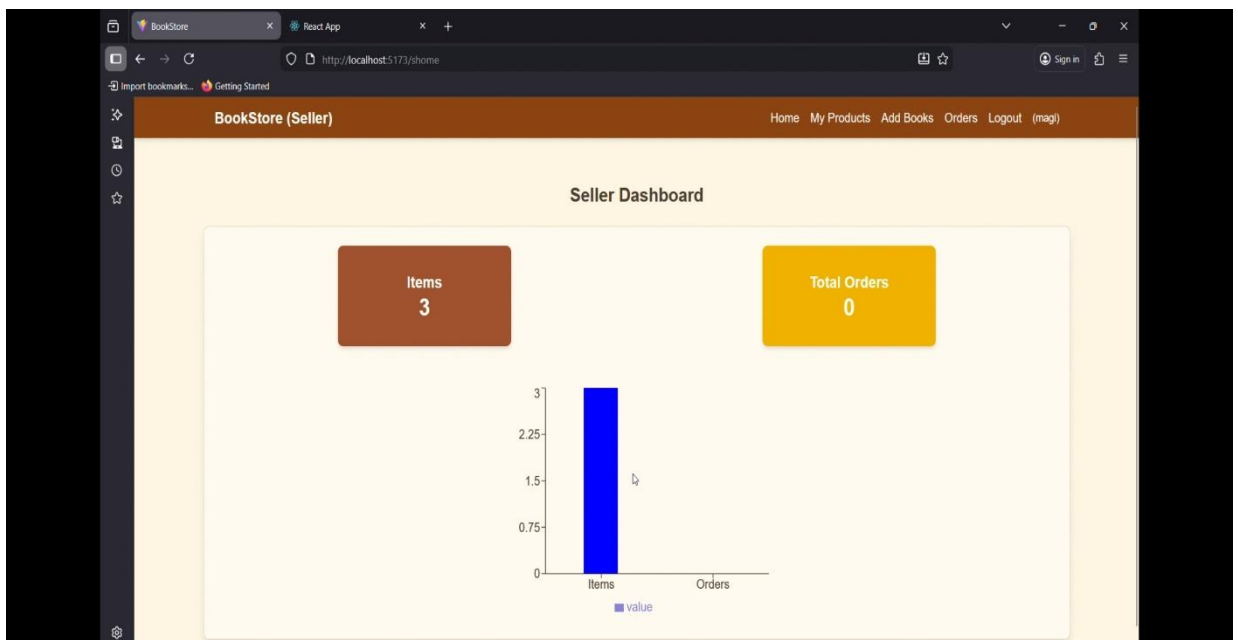
Books Listing Page



Products Details Page



Seller Dashboard



TESTING

Testing Strategy

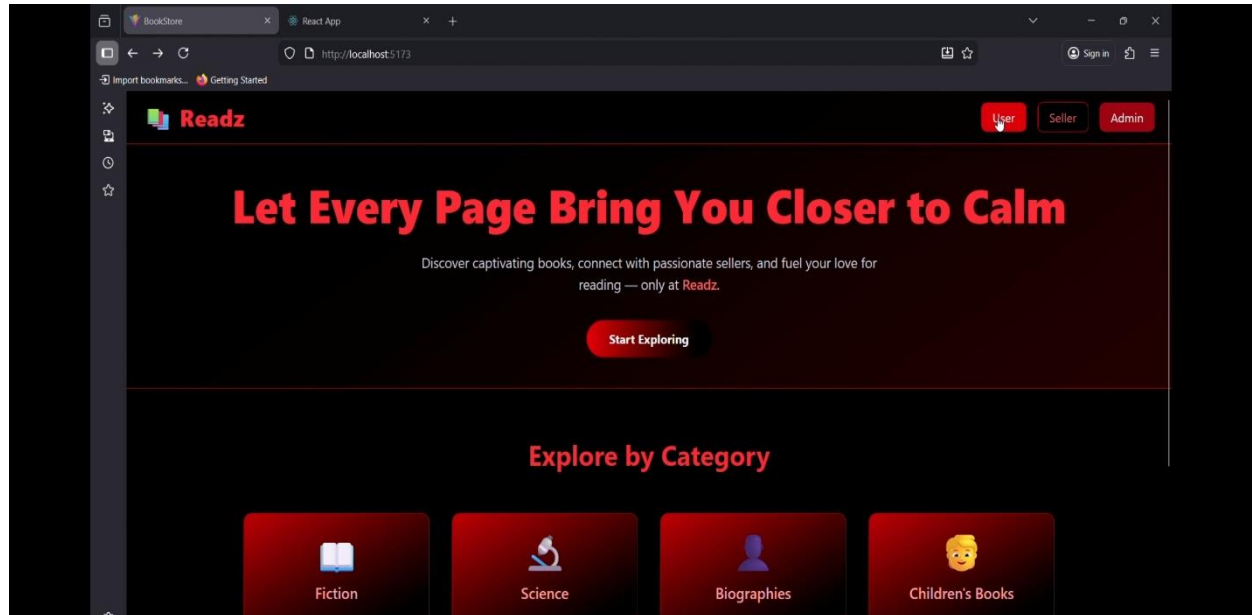
- **UI Testing:**
Each page (Home, Books, Cart, Profile, Seller & Admin Dashboard) was manually tested to verify layout, navigation, and user interactions.
- **API Testing:**
All backend API endpoints were tested using **Postman** to validate request methods, parameters, responses, and error handling.

Tools Used

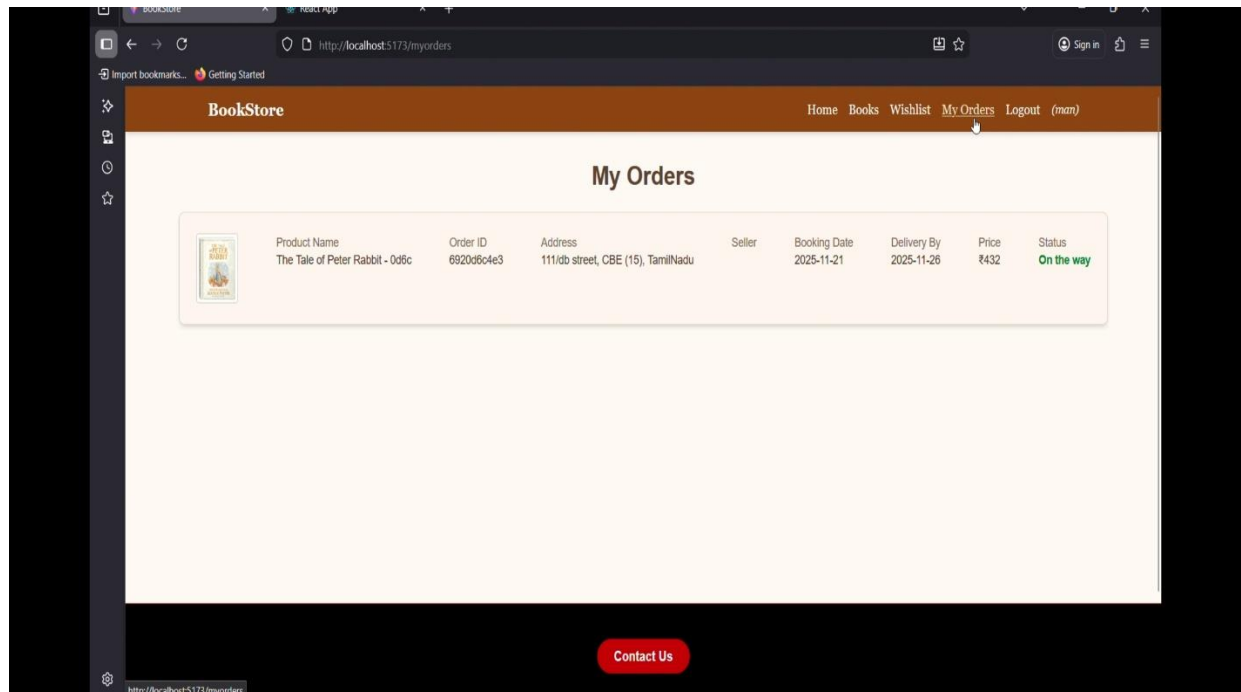
- **Manual UI Testing** (Browser-based testing)
- **Postman** for API endpoint testing
- **Console Logs & Debugging Tools** for resolving errors

SCREENSHOTS OR DEMO

Home Page



Order Confirmation Page



KNOWN ISSUES

- Some bookstore pages are not fully mobile responsive.
- Bookstore API responses may be slow at times.
- Error messages are not always displayed clearly.
- Duplicate book orders or cart entries may occur if buttons are clicked repeatedly.
- Images may load slowly on weak internet.

FUTURE ENHANCEMENTS

- Add advanced search and filters for better user experience.
- Implement payment gateway integration for online payments.
- Add an admin analytics dashboard with charts and reports.
- Improve mobile responsiveness and overall UI design.
- Enable email/SMS notifications for orders and updates.
- Add wishlist functionality, book reviews, and user ratings.
- Implement role-based access control for enhanced security.