**Project Title:** **BookNest: Where Stories Nestle**

**1. INTRODUCTION**

• **Project Title**: BookNest – A Role-Based Online Bookstore Platform  
• **Team Members**:

  • **Team Member**: A. Mahesh Babu  
  • **Team Member**: B. Jaswanth  
  • **Team Member**: D. Dinesh Eswar Reddy  
  • **Team Member**: G. Sai Deepak Reddy

**2. PROJECT OVERVIEW**

• **Purpose**:  
The purpose of the *BookNest* project is to develop a role-based online bookstore platform that facilitates seamless interaction between book readers, independent sellers, and administrators. The system aims to provide a secure, user-friendly, and scalable solution for browsing, selling, and managing books online.

• **Features**:  
• User registration and secure login with JWT authentication  
• Role-based access for Users, Sellers, and Admins  
• Book listing with image, genre, author, and price  
• Dynamic search and filter functionality  
• Cart management and order placement  
• Seller dashboard for inventory and order management  
• Admin panel for system-wide monitoring and moderation  
• Feedback and rating system for purchased books  
• Responsive frontend for web access across devices

**3. ARCHITECTURE**

• **Frontend**:  
The frontend of BookNest is built using **React.js**, providing a component-based structure for reusability and maintainability. It uses **React Router** for client-side routing and ensures responsiveness with **Bootstrap**. The user interface dynamically renders content based on user roles (User, Seller, Admin) and communicates with backend APIs through secure HTTP requests using **Axios**.

• **Backend**:  
The backend is developed using **Node.js** and **Express.js**, structured in a modular format following **MVC architecture**. It provides RESTful APIs for user authentication, book management, cart operations, order processing, and admin controls. JWT is used for session management, and middleware is implemented for role-based access control.

• **Database**:  
The application uses **MongoDB Atlas** as the database. Collections are structured as follows:  
  • users – stores user details and roles  
  • books – contains book listings with metadata and image URLs  
  • orders – records placed orders with book and user references  
  • feedback – stores user reviews and ratings  
  • sellers – holds seller profiles and listed inventory  
Database interactions are handled using **Mongoose** for schema definitions and queries.

**4. SETUP INSTRUCTIONS**

• **Prerequisites**:  
Before setting up the BookNest project, ensure the following software is installed on your system:  
  • [Node.js](https://nodejs.org) (v16 or later)  
  • [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) or a local MongoDB instance  
  • Git  
  • A code editor (e.g., VS Code)  
  • Internet connection for dependency downloads

• **Installation Steps**:

1. **Clone the Repository**  
     bash   git clone https://github.com/your-username/BookNest.git   cd BookNest
2. **Install Server Dependencies**  
     bash   cd server   npm install
3. **Install Client Dependencies**  
     bash   cd ../client   npm install
4. **Configure Environment Variables**  
   Create a .env file inside the server directory and add the following:  
   PORT=5000   MONGO\_URI=your\_mongodb\_connection\_string
5. **Run the Application**  
   Start the backend:  
     bash   cd server   npm start     
   Start the frontend:  
     bash   cd ../client   npm start
6. **Access the Application**  
   Visit: http://localhost:3000 in your browser to view the frontend.
7. **5. FOLDER STRUCTURE**
8. • **Client** (React Frontend):  
   The frontend is built using **React.js**, and the codebase is organized to support scalability and maintainability.
   1. BookNest-Frontend/
   2. ├── public/
   3. │ └── index.html
   4. ├── src/
   5. │ ├── components/
   6. │ │ ├── Navbar.jsx
   7. │ │ ├── Footer.jsx
   8. │ │ └── BookCard.jsx
   9. │ ├── pages/
   10. │ │ ├── HomePage.jsx
   11. │ │ ├── LoginPage.jsx
   12. │ │ ├── SignupPage.jsx
   13. │ │ ├── CartPage.jsx
   14. │ │ ├── SellerDashboard.jsx
   15. │ │ ├── AdminDashboard.jsx
   16. │ │ └── OrdersPage.jsx
   17. │ ├── utils/
   18. │ │ └── auth.js
   19. │ ├── App.jsx
   20. │ ├── index.js
   21. │ └── styles/
   22. │ └── Login.css
   23. ├── .env
   24. ├── package.json
   25. └── README.md
9. • **Server** (Node.js Backend):  
   The backend is developed using **Node.js** and **Express.js**, structured following MVC principles.
   1. BookNest-Backend/
   2. ├── controllers/
   3. │ ├── authController.js
   4. │ ├── orderController.js
   5. │ ├── sellerController.js
   6. │ └── adminController.js
   7. ├── models/
   8. │ ├── user.js
   9. │ ├── book.js
   10. │ └── order.js
   11. ├── routes/
   12. │ ├── auth.js
   13. │ ├── orders.js
   14. │ ├── seller.js
   15. │ └── admin.js
   16. ├── config/
   17. │ └── db.js
   18. ├── middleware/
   19. │ └── authMiddleware.js
   20. ├── server.js
   21. ├── package.json
   22. └── .env

This folder structure ensures separation of concerns, ease of navigation, and maintainability during collaborative development.

**6. RUNNING THE APPLICATION**

To run the **BookNest** application locally, follow the steps below after completing the setup and installation:

• **Frontend**  
Navigate to the client directory and run the following command to start the React frontend server:

sql

Copy code

npm start

This will launch the frontend on http://localhost:3000/ by default.

• **Backend**  
Navigate to the server directory and run the following command to start the Node.js backend server:

sql

Copy code

npm start

This will start the Express server, typically on http://localhost:5000/, and enable API communication with the frontend.

Ensure both servers are running simultaneously in separate terminal windows for the application to work correctly.

**7. API DOCUMENTATION**

The BookNest backend provides a set of RESTful API endpoints to support various functionalities for users, sellers, and administrators. The API is built using **Express.js** and communicates in **JSON** format.

**1. User Authentication**

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

json

Copy code

{

"name": "John Doe",

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

"password": "password123",

"role": "user"

}

**Response:**

json

Copy code

{

"message": "User registered successfully",

"token": "JWT\_TOKEN"

}

**POST** /api/auth/login  
• Authenticates a user and returns a token.  
**Request Body:**

json

Copy code

{

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

"password": "password123"

}

**Response:**

json

Copy code

{

"message": "Login successful",

"token": "JWT\_TOKEN"

}

**2. Book Management**

**GET** /api/books  
• Fetches all available books.  
**Response:**

json

Copy code

[

{

"\_id": "book123",

"title": "The Alchemist",

"author": "Paulo Coelho",

"genre": "Fiction",

"price": 299,

"image": "https://..."

}

]

**POST** /api/books *(Seller only)*  
• Adds a new book.  
**Request Body:**

json

Copy code

{

"title": "New Book",

"author": "Author Name",

"genre": "Drama",

"price": 499,

"image": "https://image-url.com"

}

**DELETE** /api/books/:id *(Seller/Admin)*  
• Deletes a specific book by ID.

**3. Cart & Orders**

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

json

Copy code

{

"userId": "user123",

"bookId": "book123",

"quantity": 1

}

**POST** /api/orders  
• Places an order from the cart.  
**Request Body:**

json

Copy code

{

"userId": "user123",

"cart": [

{

"bookId": "book123",

"quantity": 2

}

]

}

**4. Feedback**

**POST** /api/feedback  
• Submits feedback on a purchased book.  
**Request Body:**

json

Copy code

{

"userId": "user123",

"bookId": "book123",

"rating": 4,

"comment": "Great book!"

}

**GET** /api/feedback/:bookId  
• Fetches all feedback for a book.

**5. Admin Actions**

**GET** /api/admin/users  
• Retrieves all registered users. *(Admin only)*

**DELETE** /api/admin/user/:id  
• Deletes a user account. *(Admin only)*

**GET** /api/admin/orders  
• Views all orders across the system. *(Admin only)*

**8. AUTHENTICATION**

The **BookNest** application uses **JWT (JSON Web Tokens)** to implement secure user authentication and authorization across all roles — User, Seller, and Admin.

• **Authentication Workflow**  
Upon successful login or registration, the server generates a **JWT token** signed using a secret key. This token contains encoded information such as the user's ID and role.

* The token is returned to the client and stored securely (usually in localStorage).
* All subsequent requests to protected endpoints must include this token in the Authorization header using the Bearer schema.

**Example:**

makefile

Copy code

Authorization: Bearer <JWT\_TOKEN>

• **Authorization**  
The system uses **middleware functions** to verify the token and decode the user's role before granting access to specific routes.

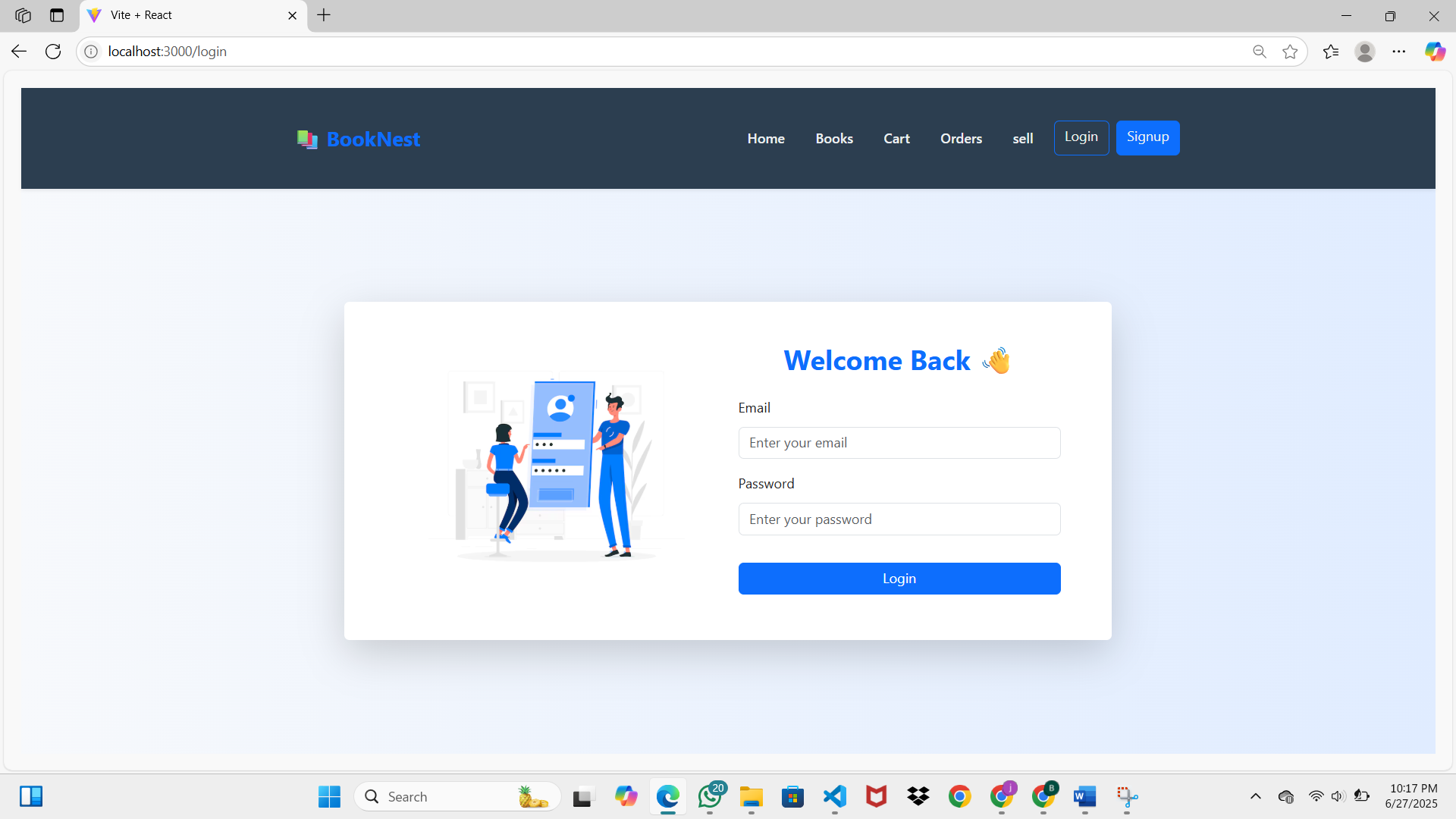
* **Users** can access personal orders, browse books, and provide feedback.
* **Sellers** can add, update, or delete books and manage orders.
* **Admins** can manage users, sellers, and the book inventory.

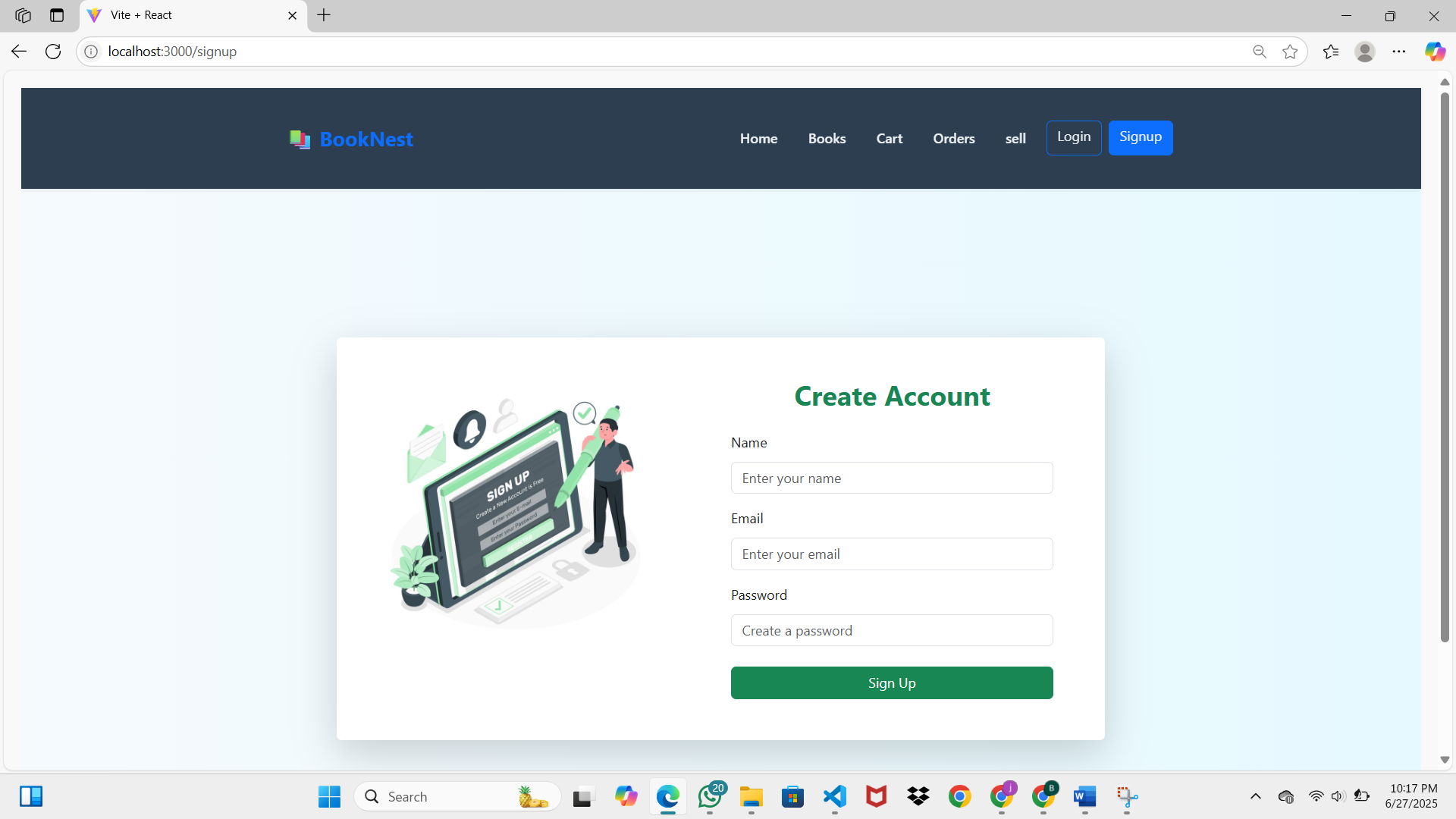
Unauthorized access attempts return a 403 Forbidden response.

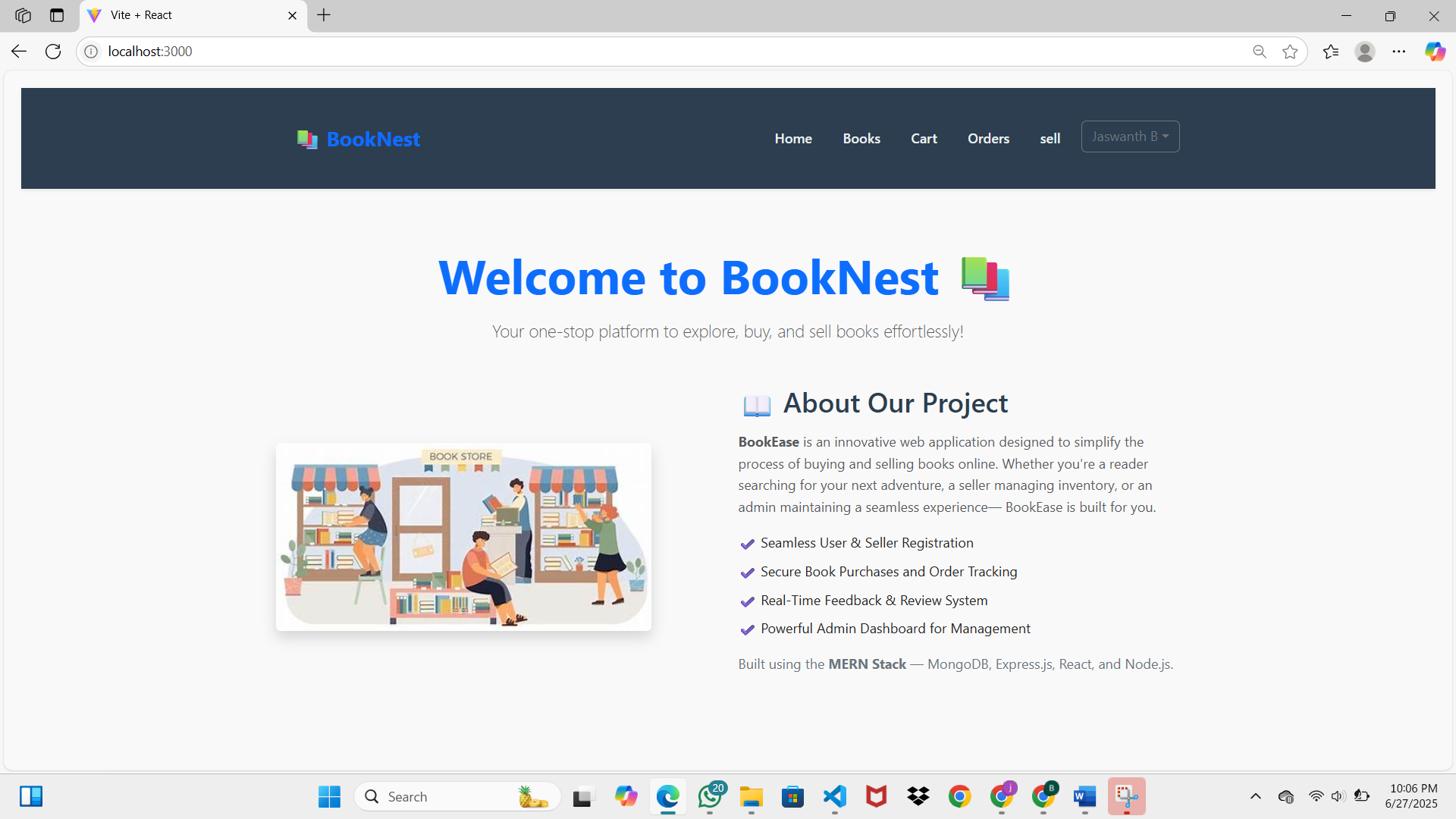
• **Security Measures**

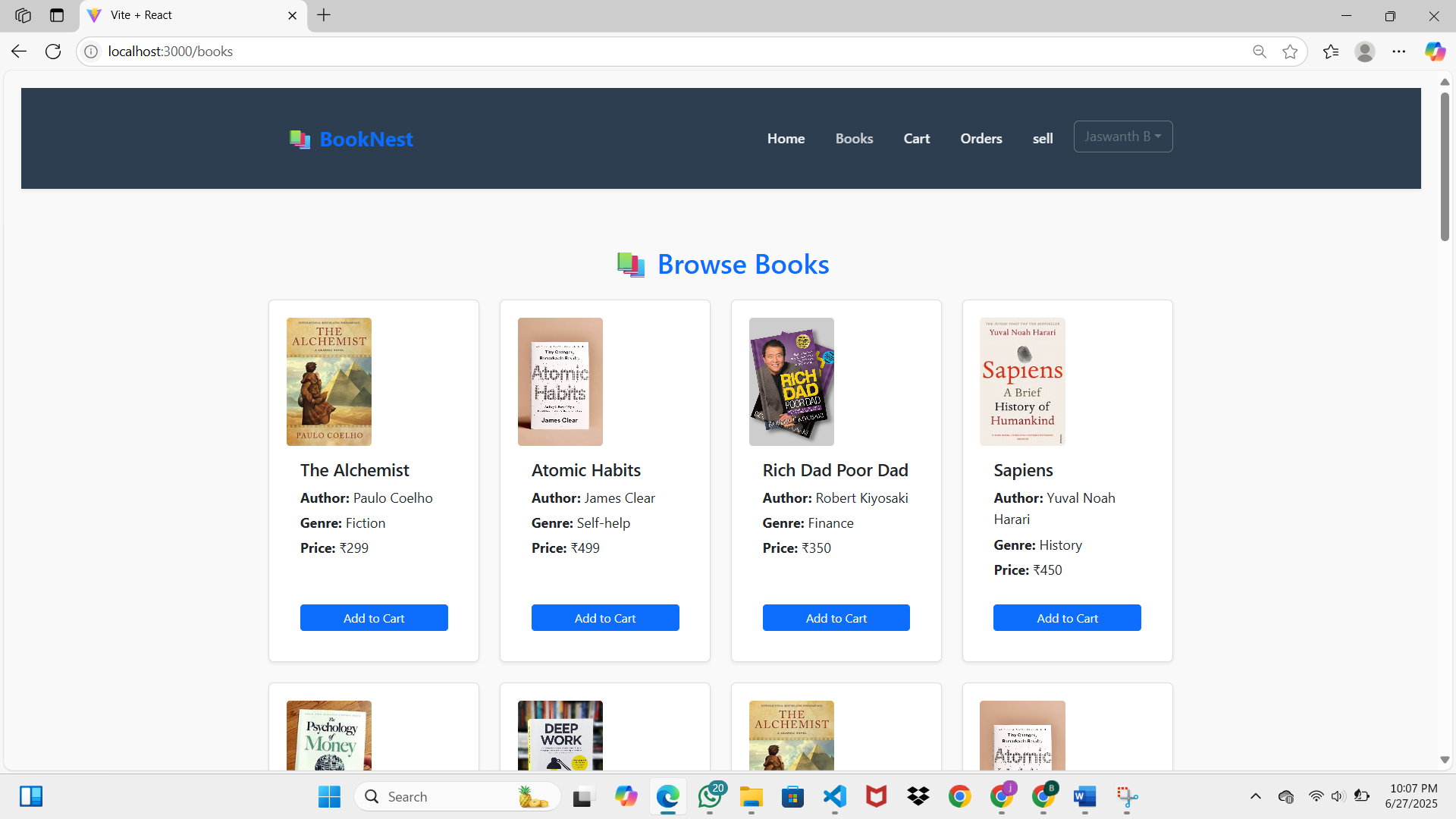
* Passwords are hashed using **bcrypt** before storage.
* Tokens are time-bound and expire after a specified duration (e.g., 1 hour).
* Sensitive routes are protected using custom **role-based access control (RBAC)** middleware.

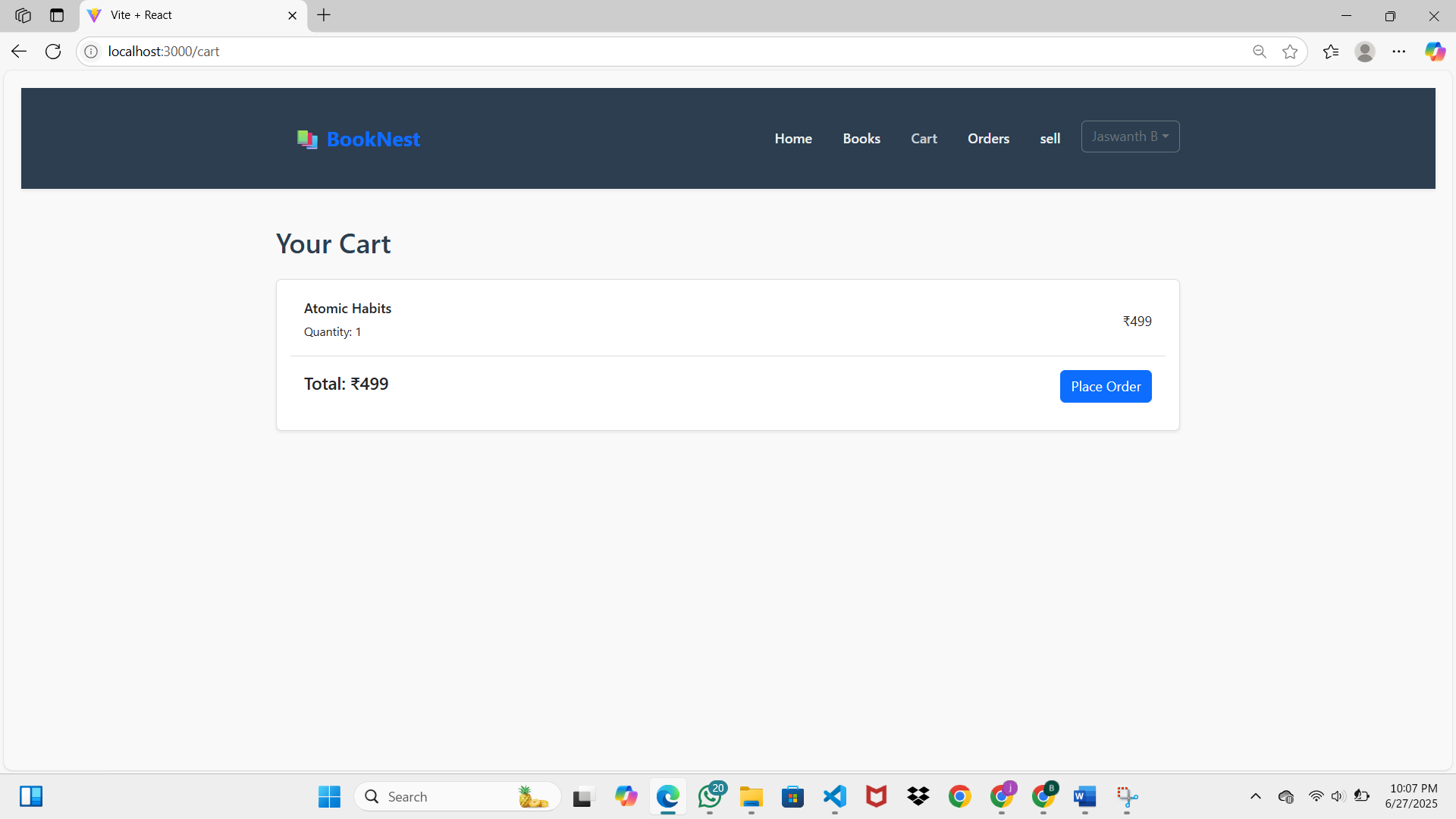
**9. User Interface**

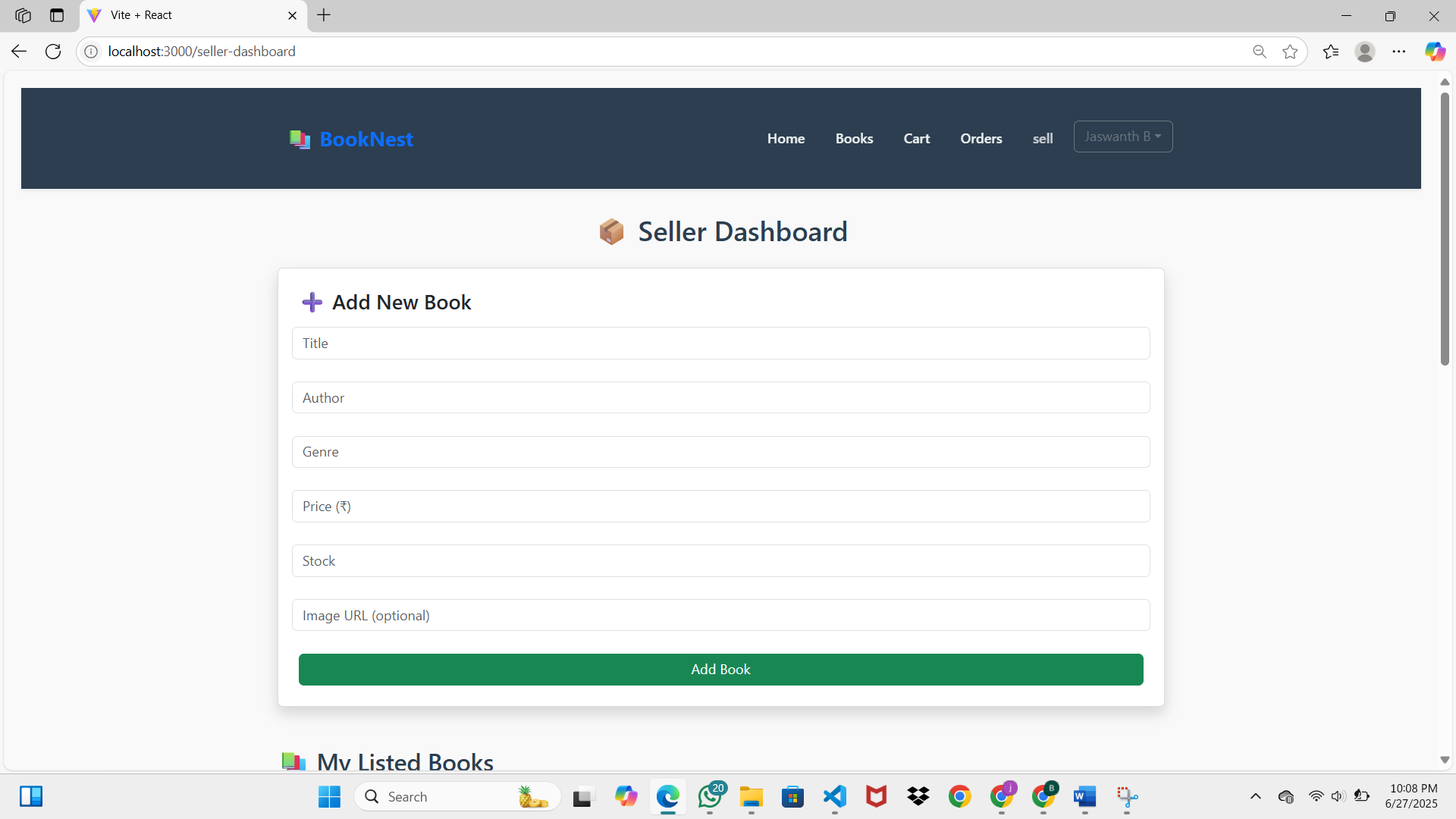












**10. TESTING**

A structured testing strategy was followed to ensure the correctness, reliability, and performance of the **BookNest** platform across all modules and user roles.

• **Testing Strategy**  
The testing approach included **manual testing**, **API testing**, and basic **performance analysis**. Testing was performed in stages—unit testing, integration testing, and end-to-end testing.

* **Unit Testing**: Individual components and functions (e.g., form handlers, API services) were manually verified.
* **Integration Testing**: Frontend and backend integration was tested by interacting with real API responses using tools like **Postman**.
* **End-to-End Testing**: Full user flows (e.g., registration → login → add to cart → checkout) were executed from the browser.
* **Role-Based Testing**: Verified proper access and redirection for User, Seller, and Admin roles.
* **Negative Testing**: Checked for incorrect login credentials, missing input data, and invalid API requests.

• **Tools Used**

* **Postman** – To test and validate backend API endpoints (CRUD operations, protected routes).
* **Browser DevTools** – For debugging, inspecting console/network activity, and mobile responsiveness.
* **Lighthouse** – To measure frontend performance, accessibility, SEO, and best practices.
* **MongoDB Compass** – For inspecting and verifying database records after operations.
* **Console & Logs** – Used during development to track bugs and verify business logic.

The system passed all major test cases with successful API responses, correct UI rendering, and secure session handling. Further automation can be considered in future phases using tools like Jest, Cypress, or Mocha.

**12. KNOWN ISSUES**

The current version of the **BookNest** platform is stable for core operations; however, a few known limitations and bugs have been identified during testing and development. These issues are planned for resolution in future updates.

• **Responsive Layout on Small Screens**  
Some UI elements in the seller and admin dashboards may misalign or overflow on smaller mobile devices.

• **Missing Error Messages on Forms**  
In certain cases, form validations (e.g., login or book listing) do not show user-friendly error messages for empty or invalid inputs.

• **Basic Feedback Validation**  
Users can currently submit empty or repetitive reviews without validation. A content filter and duplicate check should be implemented.

• **Static Search Filtering**  
Search functionality does not support combined filters (e.g., genre + author). Results are based on one field at a time.

• **No Persistent Cart Storage**  
Cart items are stored only during the session. On refresh or logout, cart data is lost unless a backend persistence layer is added.

• **Limited Analytics in Admin Panel**  
The admin dashboard lacks graphical representation or export options for user/book/order statistics.

These issues do not affect the core functionalities like registration, book browsing, ordering, or feedback but are important for improving the overall user experience in upcoming versions.

**13. FUTURE ENHANCEMENTS**

Several enhancements have been identified to improve the functionality, user experience, and scalability of the **BookNest** platform. These can be implemented in future iterations:

• **Integrated Payment Gateway**  
Implement secure payment options using services like Razorpay, Stripe, or PayPal for real-time transactions.

• **Mobile Application Support**  
Develop dedicated mobile apps for Android and iOS using React Native or Flutter for better accessibility.

• **AI-Powered Recommendation System**  
Introduce machine learning algorithms to suggest books based on user preferences and purchase history.

• **Real-Time Chat & Support**  
Add live chat functionality between users and sellers for quick communication and support.

• **Advanced Admin Analytics**  
Include dashboards with data visualizations (charts, graphs) to monitor user engagement, sales, and trends.

• **Order Tracking System**  
Allow users to track the status of their orders in real time with expected delivery timelines.

• **Push Notifications & Email Alerts**  
Enable alerts for new orders, order status updates, promotions, and reviews via email or browser notifications.

• **Multi-Language Support**  
Support localization and translation features to make the platform accessible to users in different regions.

• **Wishlist and Save for Later**  
Allow users to mark books for future reference without immediate purchase.

These enhancements aim to elevate BookNest into a full-featured, modern e-commerce platform tailored for books and literary products.