

1.Introduction

Project Title: Booknest – Where Stories Nestle

Team Members:

Team ID: LTVIP2025TMID55365

Team Leader: Pattapu Chohitha

Team member: Pantula Vindhya Kanaka Sai Pavani (Full Stack Developer)

Team member: Pamarthi Sri Lakshmi

Team member: Panguluri Vishnu Vardhan

About the Project:

Booknest is a full-stack web application built using the MERN stack (MongoDB, Express.js, React.js, Node.js). It serves as an online bookstore platform where users can conveniently browse, wishlist, and order books based on their interests. Admin users can manage the platform's content, orders, and user base using a dedicated administrative dashboard.

The project was undertaken as a solo development initiative, with the intent of showcasing complete ownership of the software development lifecycle (SDLC). From ideation to deployment, every aspect—be it the frontend user interface or the backend API logic—was conceptualized, designed, and implemented independently.

Objective:

The primary goal of this project is to build a real-world, scalable e-commerce platform tailored to the domain of book selling. The solution is designed to offer both:

- A smooth user experience for customers who wish to buy books online.
- A robust admin panel for managing users, orders, and inventory.

2. Project Overview

Purpose:

The purpose of the Booknest project is to design and implement a fully functional, user-friendly, and responsive online bookstore platform that leverages the power of the MERN stack (MongoDB, Express.js, React.js, Node.js). In today's fast-paced world, people increasingly prefer to shop online due to convenience and time efficiency. Booknest aims to meet this demand specifically in the book retail domain.

The system enables users to browse a variety of books, filter based on genre or author, manage a wishlist and shopping cart, and place orders for physical books. In parallel, it provides an admin interface that allows administrators to manage the entire inventory, update order statuses, upload book images, and oversee user accounts and order history.

This project simulates a real-world online bookstore similar to Amazon Books or Flipkart Books, but designed from scratch as a solo development project, giving full control over the frontend, backend, database modeling, and authentication.

Goals:

The key goals of the Booknest project are:

- To develop a secure, scalable, and intuitive platform for book enthusiasts.
- To implement user and admin functionalities with a clear role distinction.
- To demonstrate a full-stack developer's ability to build and document an end-to-end application.
- To provide a responsive UI that adapts across devices using modern CSS frameworks.

Features:

The following are the key features and modules implemented in Booknест:

User Authentication & Authorization:

- User registration via email and password.
- Login functionality with JWT-based session tokens.
- Role-based access control (Admin/User).
- Secure password hashing with bcrypt.js.

Book Management:

- Admin can add, edit, and delete books.
- Upload book images using file input and store via Cloudinary or local file system.
- List of books displayed to users with search and filter options (genre/author).

Wishlist System:

- Users can add or remove books from their wishlist.
- Wishlist icon (heart) toggles dynamically and reflects in the user's dashboard.

Cart and Order Placement:

- Add books to cart with quantity control.
- Place orders by filling delivery address.
- View order confirmation and order history.

Admin Panel:

Admin Dashboard with access to manage:

- Books
- Users
- Orders

Update order status (e.g., Processing, Shipped, Delivered)

User Dashboard:

- Displays user-specific wishlist and past orders.
- Allows profile updates (name, email, password).

Image Upload:

- Book image upload during book addition.
- Preview of cover image during listing.

Responsive UI

- Built with Tailwind CSS for rapid UI development.
- Fully responsive across mobile, tablet, and desktop screens.

3. Architecture

The BookNest application is built on a modern 3-tier architecture using the MERN technology stack: MongoDB (Database), Express.js (Backend), React.js (Frontend), and Node.js (Runtime Environment). This modular structure allows for easy maintenance, scalability, and separation of concerns between the user interface, server-side logic, and data storage.

Frontend Architecture – React.js:

The frontend is built using React.js, a component-based JavaScript library designed for creating dynamic, responsive user interfaces.

Structure:

- The codebase is structured around reusable components.
- Pages like Home, Books, Login, Register, Wishlist, Cart, and Dashboard are organized in a clear folder hierarchy inside /client.
- Routing is handled using React Router DOM, allowing smooth navigation and protected routes based on user authentication and roles.
- Global layout components such as the navigation bar and footer are shared across all pages via a main Layout.jsx wrapper.
- Conditional rendering is used to show or hide links based on whether the user is logged in and their role (admin/user).
- State management is handled using useState, useEffect, and context where needed, e.g., wishlist or cart badge updates.

Styling:

- Tailwind CSS is used to style all pages and components.
- The UI is fully responsive and adapts to screen sizes, supporting mobile-first design.

Backend Architecture – Node.js & Express.js:

The backend, built with Node.js and Express.js, serves as the API layer that handles all client-server interactions.

Key Structure:

- Routes are modularized by feature: /routes/bookRoutes.js, /routes/userRoutes.js, /routes/orderRoutes.js, etc.
- Each route file connects to its respective controller that contains business logic, such as addBook, getOrders, or updateProfile.
- Middleware is used to protect routes (authMiddleware.js) and check admin privileges.
- The server reads environment variables (JWT_SECRET, MONGODB_URI) via the dotenv package.
- Image handling is done using Multer, allowing admins to upload cover images for books.

Security:

- Passwords are hashed using bcrypt.js
- Authentication is handled via JWT (JSON Web Tokens) and stored in localStorage.
- Admin access is protected and can only be granted via role checks.

Database Architecture – MongoDB:

MongoDB (cloud-hosted using MongoDB Atlas) is used to store and manage all persistent data. It follows a NoSQL document model, using collections to represent different data entities.

Collections:

- users: Stores user credentials, roles, and profile details.
- books: Stores book metadata like title, author, genre, price, description, and image URL.
- orders: Tracks orders placed by users with references to user and book IDs, delivery address, and status.
- wishlist: Stores a list of book IDs each user has added to their wishlist.

Relationships:

- Orders reference both userId and a list of bookIds using ObjectId references.
- Population (.populate()) is used in queries to resolve references when needed, such as showing user info with their orders or displaying book details in a wishlist.

Indexing & Performance:

- MongoDB's natural indexing supports fast reads for book filters (genre, author).
- Aggregation pipelines can be added for advanced search, analytics, or recommendations in future versions.

4. Setup Instructions

This section provides the complete environment setup guide to run the **Booknest** project locally. It covers required tools, cloning the codebase, installing dependencies, setting environment variables, and running both client and server.

Prerequisites

Before installing and running the Booknest project, ensure that the following software and tools are installed on your system:

Tool	Purpose	Version (Recommended)
Node.js	JavaScript runtime environment	v18+
npm	Node package manager (comes with Node.js)	v9+
MongoDB Atlas	Cloud-hosted NoSQL database	Free Cluster
Git	Version control for cloning the repository	Any stable version
Code Editor	Recommended: VS Code	Optional
Cloudinary	(Optional) for storing book cover images	Free Tier
Postman	(Optional) for testing backend APIs	Optional

Installation Guide:

Step 1: Clone the Project Repository

Bash (commands):

- `git clone https://github.com/your-username/booknest.git`
- `cd booknest`

The repository typically has two main folders:

- `/client` – the React frontend
- `/server` – the Node.js + Express backend

Step 2: Install Dependencies

Install client-side (frontend) packages:

Bash (commands):

- `cd client`
- `npm install`

Install server-side (backend) packages:

Bash (commands):

- `cd ../server`
- `npm install`

This will install all required packages from package.json including:

Backend: express, mongoose, bcrypt, jsonwebtoken, cors, multer, dotenv

Frontend: react-router-dom, axios, tailwindcss, @heroicons/react

Step 3: Set Up Environment Variables

Inside the /server folder, create a .env file:

Ini (file information):

- `MONGO_URI=your_mongodb_atlas_connection_string`
- `JWT_SECRET=your_secret_key`
- `PORT=5000`
- Replace `your_mongodb_atlas_connection_string` with the URI from MongoDB Atlas.
- Ensure the .env file is listed in .gitignore to keep secrets private.

Step 4: Initialize the Database (Optional)

Ensure your MongoDB cluster is running. Once you start the backend server, initial requests will auto-connect and populate collections like users, books, and orders.

Step 5: Run the Application

Start the backend server:

Bash (commands):

- `cd server`
- `npm start`

Start the frontend React app:

bash

- `cd ../client`
- `npm start`

Once started:

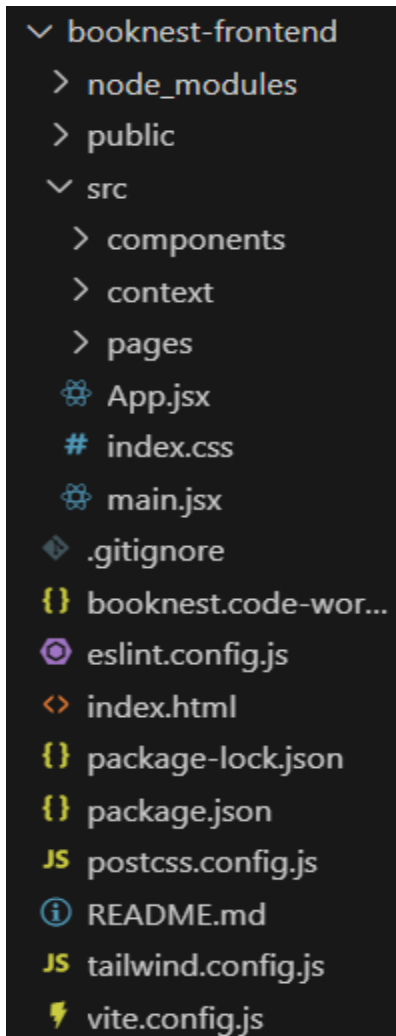
- **Frontend runs on:** `http://localhost:3000`
- **Backend runs on:** `http://localhost:5000`

You can now register, login, browse books, add to wishlist/cart, place orders, and test the admin panel. If you're using Postman, test APIs directly by sending requests to <http://localhost:5000/api/...>

5. Folder Structure

The Booknest project is divided into two main directories: /booknest-frontend for the frontend and /booknest-backend for the backend. This separation ensures modularity, clarity, and easier deployment in both local and production environments.

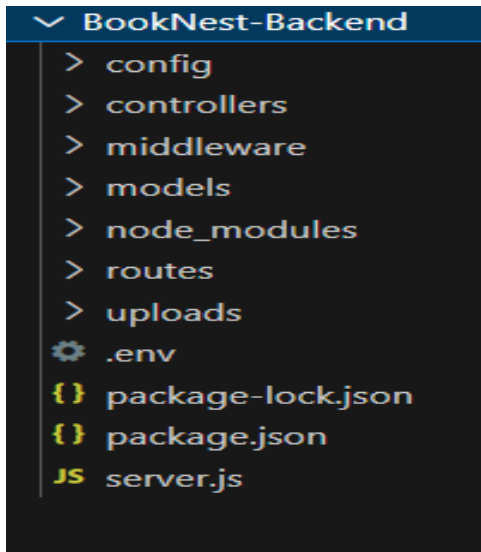
Client Folder Structure (React Frontend):



Description:

Folder/File	Purpose
booknest-frontend/	Contains index.html, favicon, and static assets
public/	Optional folder to hold images, icons, or static files
components/	Reusable UI components (e.g., BookCard, ProtectedRoute, Filters, Navbar)
pages/	Main screens like Home, Books, Login, Register, Wishlist, etc.
App.jsx	Main app logic and React Router setup
Layout.jsx	Shared layout (Navbar + page content wrapper)
index.js / main.jsx	React DOM rendering and root setup
tailwind.config.js	TailwindCSS configuration
package.json	Lists React dependencies and frontend scripts

Server Folder Structure (Node.js + Express Backend):



Description:

Folder/File	Purpose
controllers/	Contains logic for handling requests (e.g., bookController.js, userController.js)
middleware/	Contains middleware like authMiddleware.js, errorHandler.js, adminCheck.js
models/	Mongoose schemas for collections: User.js, Book.js, Order.js, etc.
routes/	API route definitions grouped by functionality: userRoutes.js, bookRoutes.js
uploads/	Local file storage folder for book images (if Cloudinary is not used)
config/	(Optional) MongoDB or Cloudinary connection configurations
.env	Environment variables (JWT_SECRET, MongoDB URI, etc.)
server.js	Entry point: initializes Express app, middleware, routes, DB connection
package.json	Backend dependencies and scripts

6. Running The Application

Once all the dependencies are installed and the environment variables are configured, you are ready to run the Booknest application locally on your machine.

This section outlines the necessary terminal commands and expected output during startup for both the React frontend and the Node.js backend.

Starting the Frontend (React Booknest-frontend):

Navigate to the client folder:

Bash (commands):

- `cd client`

Start the development server:

Bash (commands):

- `npm start`

Output:

Upon successful start, you'll see something like:

Compiled successfully!

You can now view BookNest in the browser.

Local: <http://localhost:3000>

- This command launches the React development server on <http://localhost:3000>
- Any changes made to components will auto-refresh (hot reloading).
- React Router handles routing to pages like `/login`, `/books`, `/wishlist`, `/admin`, etc.

Starting the Backend (Node + Express Server):

Navigate to the server folder:

Bash (commands):

- `cd ../server`

Start the Express server:

Bash (commands):

- `npm start`

(Or use nodemon for development if installed: `npx nodemon server.js`)

Output:

If the .env file is correctly set up, you'll see:

Connected to MongoDB...

Server running on `http://localhost:5000`

- This runs your Express API backend on `http://localhost:5000`
- The server exposes REST endpoints like `/api/books`, `/api/orders`, `/api/users`, etc.

Workflow:

- React app fetches data using Axios and calls backend endpoints at `http://localhost:5000/api/...`
- The backend connects to MongoDB Atlas for storing and retrieving user, book, wishlist, and order data.
- All API routes are protected using JWT-based middleware.

Summary of Commands:

Task	Command	Directory
Install Client	npm install	/booknest-frontend
Run Frontend	npm start	/booknest-frontend
Install Server	npm install	/BookNest-Backend
Run Backend	npm start	/ BookNest-Backend

After both servers are running, you can open your browser and access:

- Frontend UI: <http://localhost:3000>
- Backend API: <http://localhost:5000/api>

Everything is now connected, and the application is fully functional for use and testing.

7. API Documentation

The Booknest backend is built using Express.js and exposes a RESTful API interface to handle all business logic. The APIs are categorized by modules: Authentication, Books, Orders, Wishlist, Profile, and Admin.

Each API follows standard REST conventions using HTTP methods such as GET, POST, PUT, and DELETE.

Authentication Routes:

Method	Endpoint	Description	Payload/Params	Response Example
POST	/api/register	Register new user	{ name, email, password }	{ token, user }
POST	/api/login	Login existing user	{ email, password }	{ token, user }

Book Routes:

Method	Endpoint	Description	Payload/Params	Response Example
GET	/api/books	Fetch all books	Optional: genre, author	[{ _id, title, author, ... }]
POST	/api/books	Add a new book (admin only)	multipart/form-data with fields	{ message: "Book added" }
PUT	/api/books/:id	Update a book (admin only)	Updated book fields	{ message: "Book updated" }
DELETE	/api/books/:id	Delete a book (admin only)	—	{ message: "Book deleted" }

Order Routes:

Method	Endpoint	Description	Payload/Params	Response Example
POST	/api/orders	Place a new order	{ books: [], address: "" }	{ message: "Order placed" }
GET	/api/orders	Get user's order history	JWT token in headers	[{ orderId, status, books }]
PUT	/api/orders/:id	Update order status (admin only)	{ status: "Shipped" }	{ message: "Order updated" }

Wishlist Routes:

Method	Endpoint	Description	Payload/Params	Response Example
POST	/api/wishlist/:id	Toggle wishlist item (add/remove)	Book ID as param	{ message: "Added to wishlist" }
GET	/api/wishlist	Get all wishlist books for user	JWT token in headers	[{ bookId, title, image }]

User Profile Routes:

Method	Endpoint	Description	Payload/Params	Response Example
GET	/api/users/profile	Get current user profile	JWT token	{ name, email, role }
PUT	/api/users/profile	Update user profile	{ name, email, password }	{ message: "Profile updated" }

Admin Routes:

Method	Endpoint	Description	Access
GET	/api/admin/users	View all users	Admin Only
GET	/api/admin/orders	View all orders	Admin Only
DELETE	/api/admin/users/:id	Delete a user	Admin Only

Authentication using Thunder Client:

All protected routes require a JWT token to be sent via headers:

- Authorization: Bearer <token>

Sample Response Format:

```
{"message": "Order placed successfully",  
  "order": { "_id": "6653b81f2b9f21e10f4c9d43",  
    "user": "6649bb8f2aa...",  
    "books": [ ... ],  
    "status": "Processing",  
    "address": "Hyderabad" }}
```

8. Authentication

Authentication and authorization are essential aspects of the Booknest application. They ensure that users can securely access their accounts and that only authorized users (like admins) can perform sensitive operations like managing books or updating order statuses.

The project uses a token-based authentication system with JWT (JSON Web Tokens) and enforces role-based authorization for protected routes.

Authentication Flow

1. User Registration (/api/register)

- Users register by providing their name, email, and password.
- The password is hashed using bcrypt.js before storing in the MongoDB database.
- After registration, the server generates a JWT token and sends it back to the client.

2. User Login (/api/login)

On login, the server:

- Finds the user by email
- Verifies the password using bcrypt.compare()
- If valid, issues a JWT token signed using a secret (JWT_SECRET) from the .env file

The token is returned and stored in localStorage on the client side.

JWT Token Details

- Payload includes: userId, email, and role (either "user" or "admin")
- Signed using: jsonwebtoken package
- Stored securely on the frontend using localStorage
- Expiration time: can be configured (e.g., 1 day)
- Example Token Payload:

```
{
  "userId": "664a1b...",
  "email": "leelu@example.com",
  "role": "admin",
  "iat": 1710000000,
  "exp": 1710086400
}
```

Protected Routes & Middleware

The backend uses middleware to secure routes:

/middleware/authMiddleware.js:

```
const jwt = require("jsonwebtoken");

const protect = (req, res, next) => {

  const token = req.headers.authorization?.split(" ")[1];

  if (!token) return res.status(401).json({ message: "Unauthorized" });

  try {

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    req.user = decoded;

    next();

  } catch {

    return res.status(403).json({ message: "Invalid Token" });

  }

};
```

This middleware:

- Extracts token from the request header, verifies it
- Attaches the decoded user to req.user for use in controllers

Authorization by Role

Some routes are **restricted to admins only** using an additional check:

```
const adminOnly = (req, res, next) => {  
  
  if (req.user.role !== "admin") {  
  
    return res.status(403).json({ message: "Access denied" });  
  
  }  
  
  next();  
  
};
```

Used in admin-specific routes like:

- /api/admin/books
- /api/admin/orders
- /api/admin/users

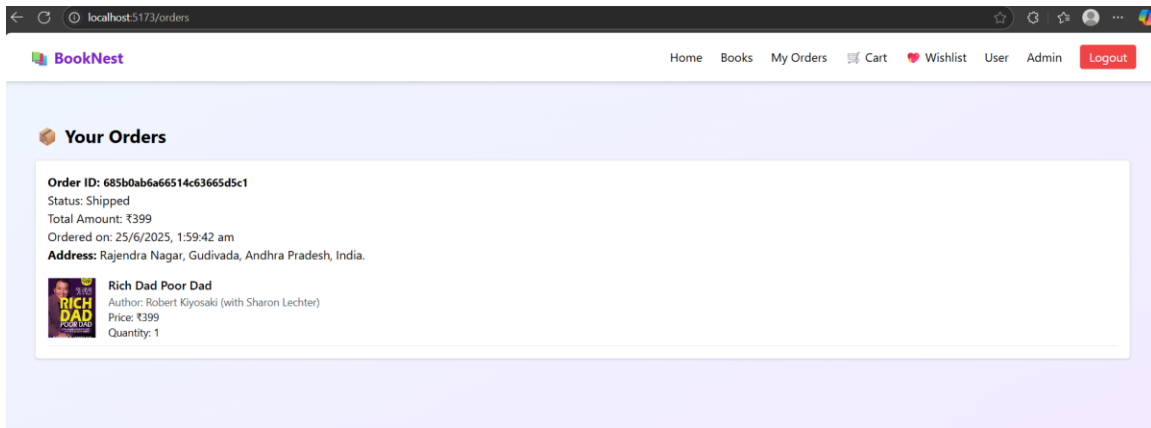
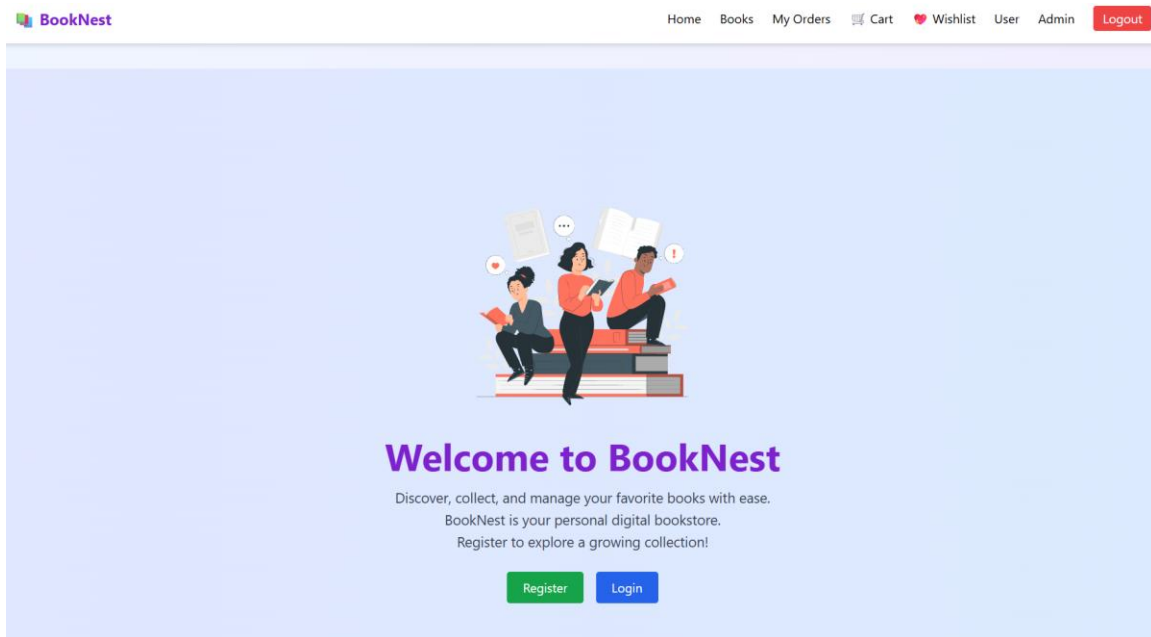
Session Management

- Booknest is a stateless app (no server-side sessions)
- All user identity and session info is maintained via JWT tokens
- Tokens are validated on every request using the middleware, ensuring security

Aspect	Implementation
Auth Type	Token-based (JWT)
Storage (Frontend)	localStorage
Password Protection	bcrypt.js
Route Protection	Express Middleware
Admin Control	Role-based access with middleware
Expiry Handling	Configurable in JWT options

9. User Interface

The BookNest user interface (UI) is designed to be modern, clean, and fully responsive. It was built using React.js and styled with Tailwind CSS, ensuring fast load times, consistent visual hierarchy, and an accessible layout across all devices (desktop, tablet, mobile).



10. Testing

Thorough testing was carried out at both the frontend and backend levels to ensure the stability, correctness, and security of the Booknest application. Although automated testing was not implemented, manual testing strategies were followed rigorously to verify all critical workflows and components.

Testing Strategy

The testing strategy used in Booknest followed these four main layers:

Layer	Focus
Unit Testing	Testing individual components like forms, buttons, filters
Integration Testing	Ensuring smooth data flow between frontend & backend (e.g., order placement)
End-to-End Testing (E2E)	Simulating user journeys such as register → login → browse → add to cart → place order
Role-based Testing	Testing both user and admin roles for protected route access and permissions

Frontend Testing (Manual)

- **Form Validations:** Login/Register forms were tested for empty fields, invalid inputs, and error messages.
- **Component Rendering:** All pages were tested to render correctly across devices using browser developer tools.
- **Protected Routes:** Verified that non-authenticated users are redirected to the login page when trying to access /dashboard or /admin.
- **UI Responsiveness:** Manually tested the layout in mobile, tablet, and desktop views.

Tools used:

- Browser DevTools (Chrome, Edge)
- React Developer Tools (Extension)

Backend API Testing (Manual using Postman)

All backend API endpoints were tested using Postman to verify:

- Correct response structure
- Status codes (200, 201, 400, 401, 403, 404)
- Protected route behavior with and without JWT
- Admin-only route access

Sample Tests Conducted:

- POST /api/register with invalid email → returns 400 Bad Request
- GET /api/books without token → returns list of books
- POST /api/orders without token → returns 401 Unauthorized
- PUT /api/orders/:id as admin → status updated successfully

Role-Based Testing:

Scenario	Expected Result
User accesses admin route	Access denied (403)
Admin accesses all user routes	Full access granted
Book edit by non-admin	Access denied
JWT tampering or expiration	Invalid token error (handled gracefully)

Cross-Browser Testing

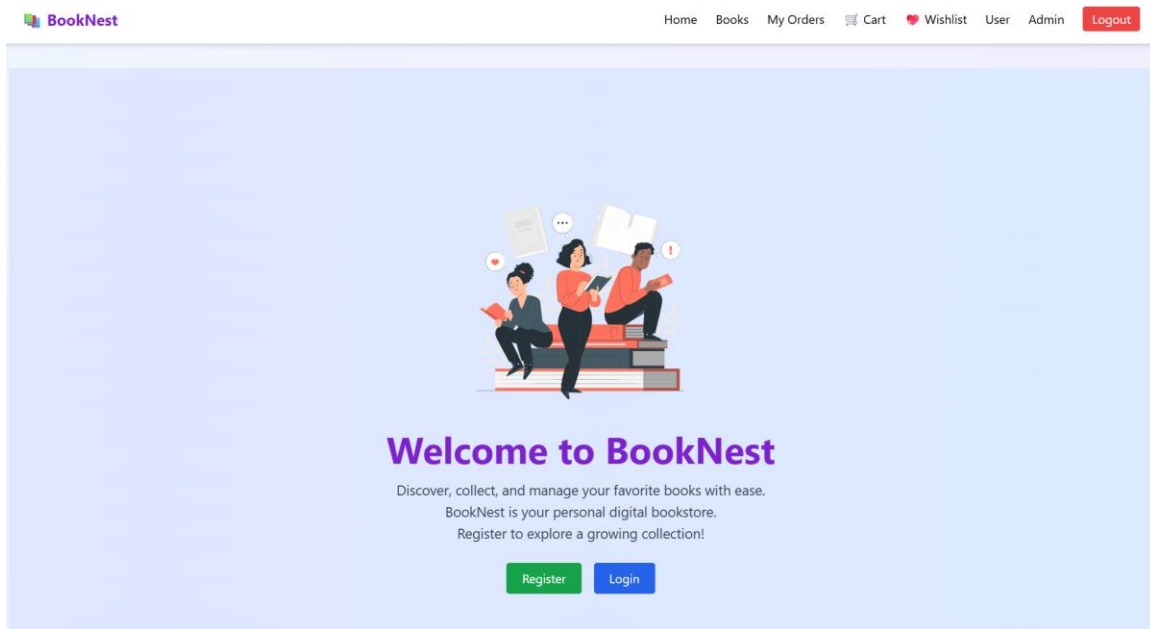
The app was tested on:

- Google Chrome
- Microsoft Edge
- Firefox

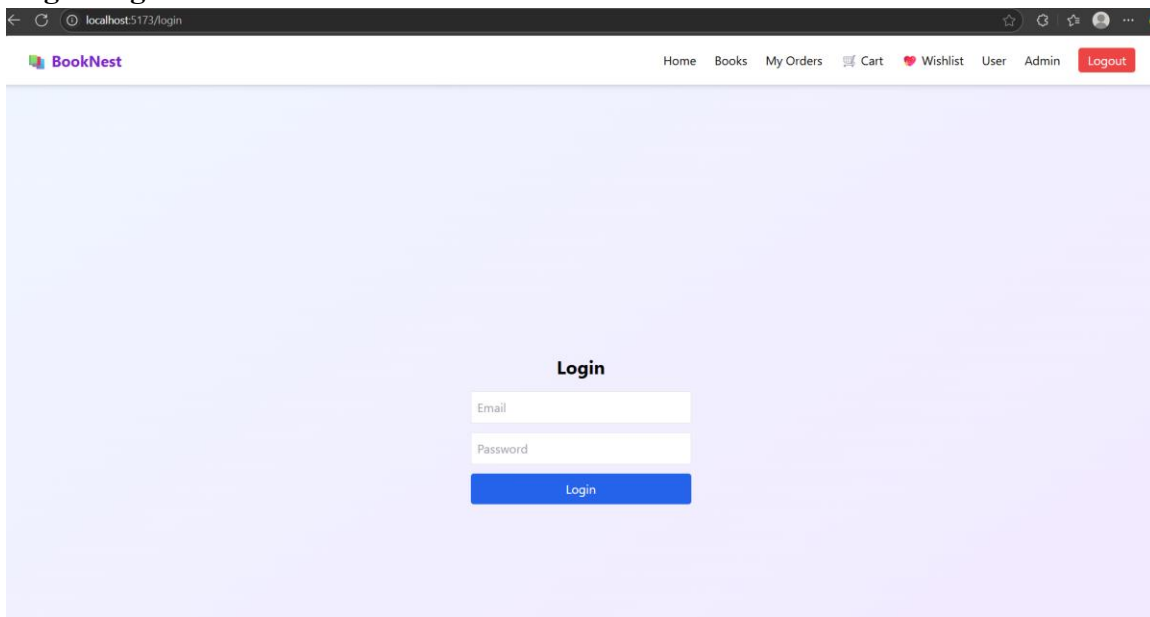
All displayed consistent behavior and layout rendering.

11. Screenshots / Demo

Home Page:



Login Page:



Registration Page:

localhost:5173/register

BookNest

Home Books My Orders Cart Wishlist User Admin Logout

Register

Name

Email

Password


Books Page:

BookNest

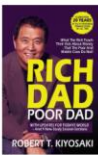
Home Books My Orders Cart Wishlist User Admin Logout

Available Books


All Genres All Authors Reset Filters




The Little Prince
A cosmic journey revealing life's unseen truths through a child's eyes.
Author: Antoine de Saint-Exupéry
Genre: Fantasy, Children's Fiction, Philosophical Allegory
Price: ₹599
[Add to Cart](#)




Rich Dad Poor Dad
"Rich Dad Poor Dad" contrasts the financial philosophies of two fathers to underscore the importance of financial literacy, teaching readers to acquire income-generating assets rather than liabilities to achieve wealth.
Author: Robert Kiyosaki (with Sharon Lechter)
Genre: Personal Finance, Non-Fiction, Self-Help, Business, Investment
Price: ₹399
[Add to Cart](#)




The Silent Patient
A psychological thriller where a woman's act of violence against her husband and her refusal to speak leads to a gripping mystery.
Author: Alex Michaelides
Genre: Thriller
Price: ₹399
[Add to Cart](#)



Atomic Habits





The Alchemist




Stephen Hawking

Cart Page:


HomeBooksMy OrdersCart1WishlistUserAdminLogout

Your Cart



The Little Prince
₹599 × 1
Antoine de Saint-Exupéry

Remove


Delivery Address:


Enter your delivery address

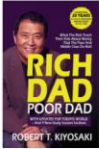
Total: ₹599

✓ Place Order

Wishlist Page:

HomeBooksMy OrdersCart1WishlistUserAdminLogout

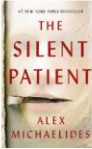
Your Wishlist



Rich Dad Poor Dad
Robert Kiyosaki (with Sharon Lechter)
₹399

Move to Cart

Remove



The Silent Patient
Alex Michaelides
₹399

Move to Cart

Remove

Admin Dashboard:

HomeBooksMy OrdersCart1WishlistUserAdminLogout

User Dashboard

Name: abc

Email: abc@gmail.com

Role: user

Edit Profile

Your Orders Page:

BookNest

HomeBooksMy OrdersCart1WishlistUserAdminLogout

Your Orders

Order ID: 685b0ab6a66514c63665d5c1

Status: Shipped

Total Amount: ₹399

Ordered on: 25/6/2025, 1:59:42 am

Address: Rajendra Nagar, Gudivada, Andhra Pradesh, India.

RICH DAD POOR DAD

Rich Dad Poor Dad

Author: Robert Kiyosaki (with Sharon Lechter)

Price: ₹399

Quantity: 1

Admin Dashboard:

BookNest

HomeBooksMy OrdersCartWishlistUserAdminLogout

Admin Dashboard

Total Users

3

Total Books

10

Total Orders

1

Admin Viewing all the Users:

BookNest

HomeBooksMy OrdersCartWishlistUserAdminLogout

All Users

Name	Email	Role	Created
Leelu	leelu@example.com	user	20/6/2025
Admin	admin@example.com	admin	21/6/2025
abc	abc@gmail.com	user	22/6/2025

Admin can perform CRUD operations on books:

BookNest

[Home](#)[Books](#)[My Orders](#)

Cart

Wishlist

User

Admin

Logout

Manage Books

Add Book

Title	Author	Genre	Price	Actions	
The Little Prince	Antoine de Saint-Exupéry	Fantasy, Children's Fiction, Philosophical Allegory	₹599	Edit	Delete
Rich Dad Poor Dad	Robert Kiyosaki (with Sharon Lechter)	Personal Finance, Non-Fiction, Self-Help, Business, Investment	₹399	Edit	Delete
The Silent Patient	Alex Michaelides	Thriller	₹399	Edit	Delete
Atomic Habits	James Clear	Self-Help	₹499	Edit	Delete
The Alchemist	Paulo Coelho	Fiction	₹350	Edit	Delete
A Brief History of Time	Stephen Hawking	Science	₹599	Edit	Delete
Wings of Fire	A.P.J. Abdul Kalam	Autobiography	₹299	Edit	Delete
The Hobbit	J.R.R. Tolkien	Fantasy	₹450	Edit	Delete
The Subtle Art of Not Giving a Fck*	Mark Manson	Self-Help	₹399	Edit	Delete
Harry Potter and the Sorcerer's Stone	J.K. Rowling	Fantasy	₹550	Edit	Delete

© 2025 BookNest. All rights reserved.

Admin updating the order status:

BookNest

[Home](#)[Books](#)[My Orders](#)

Cart

Wishlist

User

Admin

Logout

All Orders

Order ID	User	Books	Total Price	Address	Date	Status
685b0ab6a66514c63665d5c1	abc	<div>Rich Dad Poor Dad x 1</div>	₹399	Rajendra Nagar, Gudivada, Andhra Pradesh, India.	25/6/2025, 1:59:42 am	<div>Shipped</div> <div>Processing</div> <div>Shipped</div> <div>Delivered</div> <div>Cancelled</div>

12. Known Issues

While the BookNest application is fully functional and tested across multiple workflows, there are a few known issues and limitations that are either minor in nature or planned for resolution in future development phases. These issues do not significantly impact the user experience but are documented for transparency and improvement tracking.

1. Wishlist Sync Delay on Refresh

- **Issue:** When a user toggles a book in the wishlist, the UI updates instantly. However, upon a full page refresh, the changes may take a few seconds to reflect due to async fetch delays.
- **Cause:** Wishlist fetch occurs after initial page render and depends on token authentication and data propagation.
- **Workaround:** User can wait a few seconds or manually navigate to the wishlist page to confirm the update.

2. Image Upload Reliability

- **Issue:** Occasionally, image uploads for book covers fail when the internet connection is weak or slow.
- **Cause:** Book cover upload is handled using multipart/form-data and saved either locally or to Cloudinary. Network latency causes the form submission to timeout.
- **Workaround:** Retrying the upload resolves the issue. Improving error handling and upload feedback will enhance user experience.

3. No Refresh Token Mechanism

- **Issue:** JWT tokens are short-lived and stored in localStorage, with no token refresh system.
- **Impact:** Users may be logged out after token expiration and have to log in again manually.
- **Recommendation:** Implement refresh token and token rotation to improve session longevity.

4. No Payment Integration

- **Issue:** Orders are placed without any payment gateway integration.
- **Cause:** The current version assumes cash-on-delivery or offline payment.
- **Planned Fix:** Integrate Razorpay or Stripe to handle secure online transactions.

5. No Automated Test Coverage

- **Issue:** All testing is manual; no unit/integration tests are written using Jest, Mocha, or Cypress.
- **Impact:** Limits test automation and may affect regression testing as the app scales.
- **Recommendation:** Add automated testing in the next phase with CI/CD pipeline.

6. Admin Role Cannot Be Assigned from UI

- **Issue:** Admin users must be assigned via direct database update (e.g., MongoDB Compass or shell).
- **Risk:** If no admin exists, user management becomes inaccessible.
- **Planned Improvement:** Add a UI toggle for role assignment (admin/user) accessible by super admins.

7. Deployment Not Configured

- **Issue:** App is currently set up for local development only.
- **Impact:** No public access unless manually deployed on platforms like Vercel (client) and Render (server).
- **Next Step:** Create deployment scripts and host the application for live usage.

13. Future Enhancements

1. Payment Gateway Integration

- **Why:** Currently, orders are placed without any payment mechanism.
- **Enhancement:** Integrate Razorpay or Stripe to allow secure online payments via cards, UPI, or wallets.
- **Impact:** Makes the application production-ready for real commerce.

2. Token Refresh Mechanism

- **Why:** Users must re-login when their token expires.
- **Enhancement:** Implement refresh tokens stored in httpOnly cookies to automatically renew sessions.
- **Impact:** Improves session persistence and user convenience without compromising security.

3. Automated Testing Framework

- **Why:** Manual testing limits reliability and scalability.
- **Enhancement:** Use Jest, React Testing Library, or Cypress for unit, integration, and E2E testing.
- **Impact:** Ensures robust development with test-driven deployment and fewer regressions.

4. Book Reviews & Ratings

- **Why:** Users currently can't provide feedback or see public opinions.
- **Enhancement:** Allow users to leave reviews and rate books (1–5 stars).
- **Impact:** Boosts engagement, builds trust, and helps users make informed decisions.

5. Cloud Deployment (Frontend + Backend)

- **Why:** App is accessible only in a local environment.
- **Enhancement:** Deploy React frontend to Vercel and Express backend to Render/Heroku.
- **Impact:** Makes the platform accessible to end users and suitable for demonstration or beta testing.