Full Stack Development with MERN

Project Documentation format

1. Introduction

• **Project Title:** [Your Project Title]

• **Team Members:** List team members and their roles.

2. Project Overview

• **Purpose:** Briefly describe the purpose and goals of the project. • **Features:** Highlight key features and functionalities.

3. Architecture

• **Frontend:** Describe the frontend architecture using React.

• **Backend:** Outline the backend architecture using Node.js and Express.js. • **Database:** Detail the database schema and interactions with MongoDB.

4. Setup Instructions

• **Prerequisites:** List software dependencies (e.g., Node.js, MongoDB). • **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

5. Folder Structure

• **Client:** Describe the structure of the React frontend.

• **Server:** Explain the organization of the Node.js backend.

6. Running the Application

• Provide commands to start the frontend and backend servers locally.

o **Frontend:** npm start in the client directory.

o **Backend:** npm start in the server directory.

7. API Documentation

• Document all endpoints exposed by the backend.

• Include request methods, parameters, and example responses.

8. Authentication

• Explain how authentication and authorization are handled in the project

• Include details about tokens, sessions, or any other methods used.

9. User Interface

• Provide screenshots or GIFs showcasing different UI features.

10. Testing

• Describe the testing strategy and tools used.

11. Screenshots or Demo

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

12. Known Issues

• Document any known bugs or issues that users or developers should be aware of.

13. Future Enhancements

• Outline potential future features or improvements that could be made to the project.

BookNest: Where Stories Nestle

1. Introduction

Project Title: BookNest: Where Stories Nestle

Team Details:

Team ID: LTVIP2026TMIDS81807

Team Size: 4

Team Leader: Guttula Venkata Sivaram

Team member: Shubham Kumar

Team member: Pooja Sree Talliboyina

Team member: Basudora Jagadeeswara Rao

2. Project Overview:

Purpose of the Project

BookNest is an online bookstore developed using the MERN stack. The purpose of this project is to provide a centralized digital platform where customers can browse and purchase books, sellers can manage inventory, and administrators can oversee platform activities

Goals of the Project

1. To create an efficient digital online bookstore system for browsing and purchasing books.
2. To implement secure authentication and authorization mechanisms for users, sellers, and administrators.
3. o provide role-based dashboards for Admin, Seller, and Customer.
4. To develop a scalable architecture using cloud database services for managing books and orders.
5. To enhance online book shopping experience through a user-friendly design.

Key Features and Functionalities:

Customer Features

• User Registration and Login with JWT authentication.

• Browse available books and view book details.

• Add books to cart and place orders.

• View order history and notifications.

Seller Features

• Manage book inventory and listings.
• Update profile details.
• View and process customer orders.

Admin Features

• Approve seller registrations.
• Monitor users and sellers.
• Manage account status and platform governance.

Technical Features

• RESTful API Architecture.
• MongoDB Atlas Cloud Database.
• Role-Based Access Control.
• Modular MVC Backend Structure.

3. Architecture

I. Web Application (Frontend Layer):

The frontend layer of DocSpot is developed using React.js, which provides a component-based architecture. This layer serves as the primary interface where users interact with the system.

Key frontend responsibilities include:

- Rendering dashboards for different roles (Customer, Seller, and Admin.
- Managing routing using React Router.
- Handling API communication using Axios.
- Providing responsive UI using Material UI, Ant Design, Bootstrap, and MDB React UI Kit.

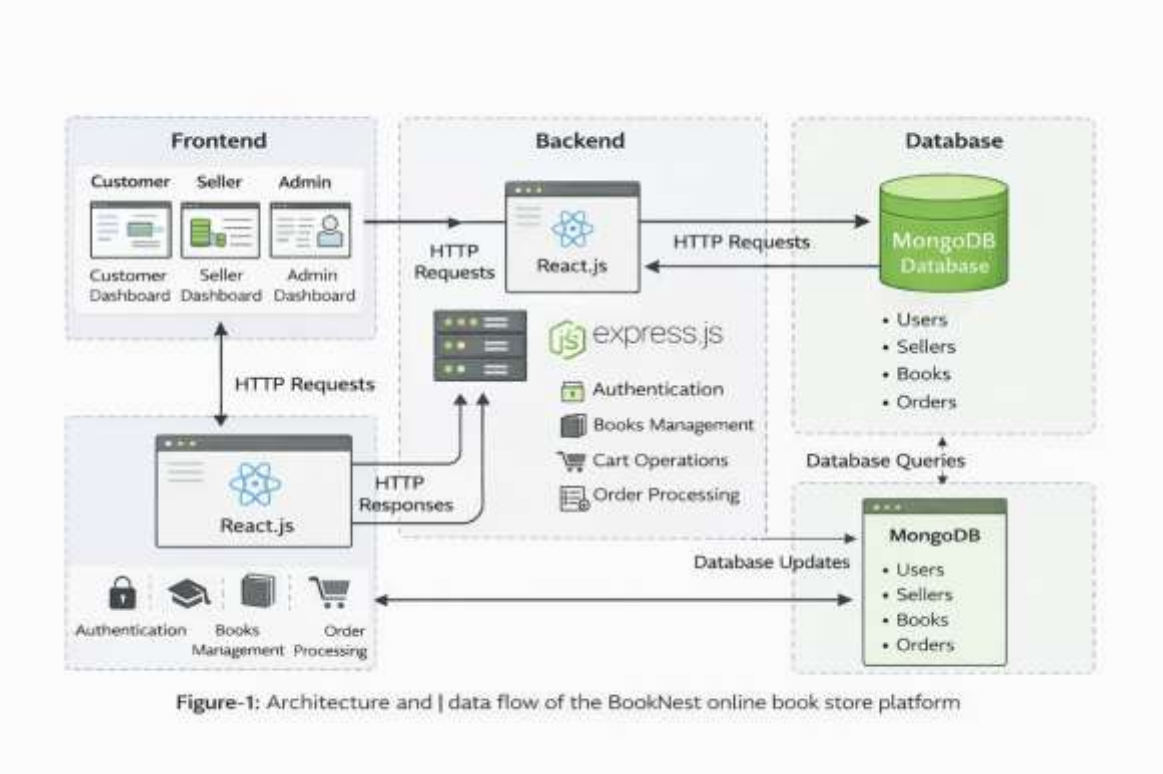


Fig-1: Main Architecture

II. Backend Services (Logic Layer)

The backend is built using Node.js and Express.js following an MVC architecture.

Components:

- **Routes:** Define API endpoints.
- **Controllers:** Contain business logic for book orders, authentication, inventory management, and admin actions.
- **Middleware:** Handle JWT authentication and role verification.
- **Schemas:** Represent database models using Mongoose.

This layered structure ensures maintainability, scalability, and secure request processing.

III. Database Layer (MongoDB Atlas)

The database layer uses MongoDB Atlas, a cloud-hosted NoSQL database.

Collections include:

• Users

• Sellers

• Books

• Orders

• ReviewsMongoDB's flexible schema allows dynamic data storage and efficient querying.

4. Setup Instructions

Prerequisites:

To develop a full-stack Ecommerce App for Furniture Tool using React js, Node.js, Express js and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

• Download: https://nodejs.org/en/download/

• Installation instructions: https://nodejs.org/en/download/package-manager/

MongoDB: MongoDB: Set up a MongoDB database to store users, sellers, books, and order information. Install MongoDB locally or use a cloud-based MongoDB service.

• Download: https://www.mongodb.com/try/download/community

• Installation instructions: https://docs.mongodb.com/manual/installation/

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

• Installation: Open your command prompt or terminal and run the following command: npm install express

React js: React is a JavaScript library for building client-side applications.

And Creating Single Page Web-Application

Installation Overview

1. Clone the project repository.
2. Install frontend and backend dependencies.
3. Configure environment variables including database URL and JWT secret key.
4. Run both frontend and backend servers simultaneously.

Getting Started

Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

Quick Start

npm create vite@latest

cd my-app

npm install

npm run dev

If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app or yarn global remove create-react-app to ensure that npx always uses the latest version.

Create a new React project:

• Choose or create a directory where you want to set up your React project.

• Open your terminal or command prompt.

• Navigate to the selected directory using the cd command.

• Create a new React project by running the following command: npx create-react-app your-app-name. Wait for the project to be created:

• This command will generate the basic project structure and install the necessary dependencies

Navigate into the project directory:

• After the project creation is complete, navigate into the project directory by running the following command: cd your-app-name

Start the development server:

• To launch the development server and see your React app in the browser, run the following command: npm run dev

• The npm start will compile your app and start the development server.

• Open your web browser and navigate to https://localhost:5173 to see your React app.

You have successfully set up React on your machine and created a new React project. You can now start building your app by modifying the generated project files in the src directory.

Please note that these instructions provide a basic setup for React. You can explore more advanced configurations and features by referring to the official React documentation: https://react.dev/

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Library: Utilize React to build the user-facing part of the application, including book listings, cart functionality, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

Git: Download and installation instructions can be found at: https://git-scm.com/downloads

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code.

• Visual Studio Code: Download from https://code.visualstudio.com/download

5. Folder Structure

The project is divided into two main modules:

Client Module:

Contains React components, routing configurations, and UI-related files.

Backend Module

Includes configuration files, controllers, middleware logic, API routes, database schemas, and server initialization scripts.

This structured organization enhances code readability and promotes modular development.

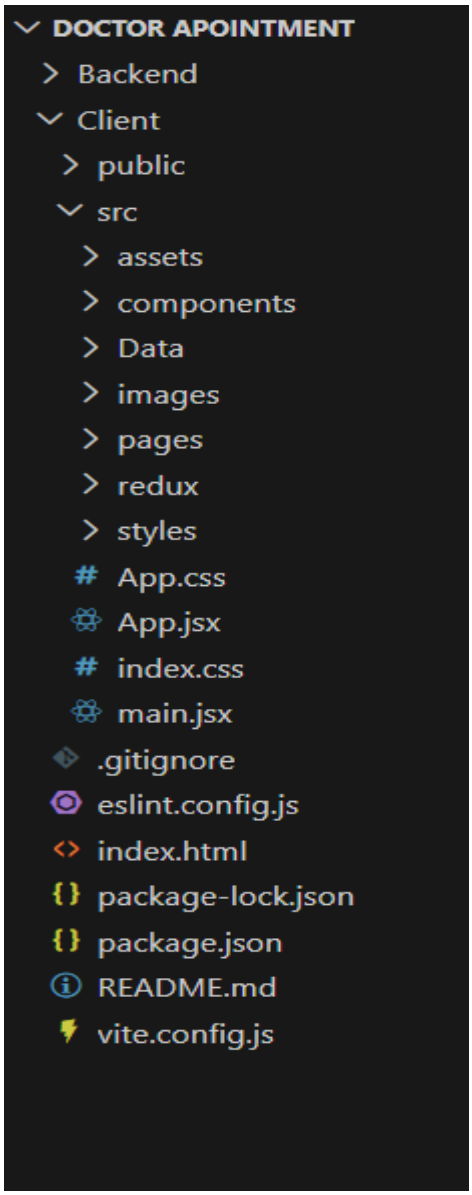


Fig-2: Client Structure

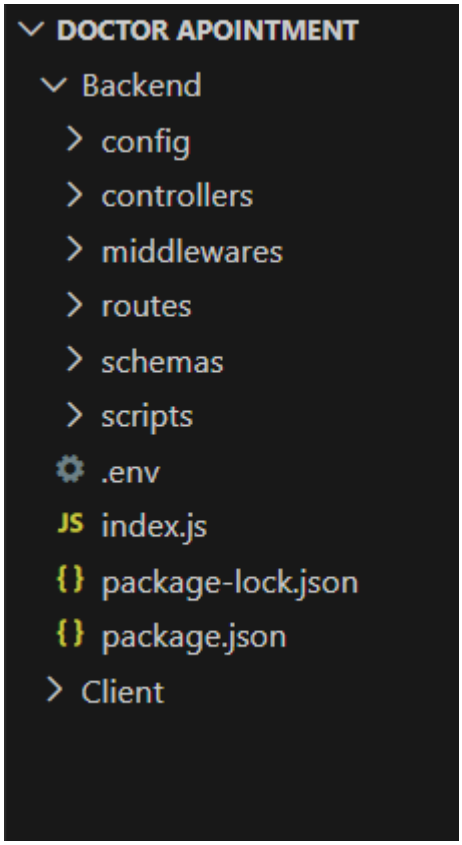


Fig-3: Backend Structure

Frontend (React Client) – Structure

The frontend of the **BookNest** application is developed using React.js and follows a modular component-based architecture that promotes reusability and maintainability. The client side is responsible for rendering user interfaces, managing user interactions, and communicating with backend APIs through HTTP requests.

The React client is organized into structured folders such as pages, components, redux store, routes, and services. The pages directory contains role-based screens including **customer dashboard, seller dashboard, admin panel, authentication pages, book listings, cart,**

and order management interfaces. Each page represents a functional module that interacts with backend services.

Reusable UI elements such as navigation bars, loaders, protected routes, and layout wrappers are stored within the components folder. These shared components ensure consistent UI design across different modules of the application.

Global application state is managed using Redux Toolkit, which allows centralized data handling and efficient state synchronization across components. React Hooks such as useState, useEffect, and useSelector are used to manage dynamic rendering, lifecycle events, and API responses.

Backend (Node.js Server) – Structure:

The backend of **BookNest** is built using Node.js and Express.js following a structured MVC (Model-View-Controller) architecture. This design pattern separates business logic, routing, and database models into independent layers, improving scalability and maintainability of the application.

The backend directory is organized into folders such as config, controllers, middlewares, routes, schemas, and scripts.

The schemas folder defines MongoDB data models using Mongoose. It includes schema definitions for **Users, Sellers, Books, Orders, and Reviews**, which represent the core entities of the system. These schemas enforce data structure and enable efficient database interactions.

The routes folder manages API endpoint definitions. Each route file corresponds to a specific module such as **userRoutes, sellerRoutes, adminRoutes, bookRoutes, and orderRoutes**. Routes receive incoming HTTP requests and delegate processing to appropriate controllers.

Controllers contain the main business logic of the application. They handle user authentication, **book browsing and ordering workflows, seller management processes, profile updates, and order status management**. Controllers act as an intermediary between routes and database models, ensuring modular design.

The middlewares folder contains authentication and authorization logic. JWT middleware verifies user identity, while role-based middleware restricts access to specific APIs based on user roles such as **admin, seller, or customer**. Bcrypt hashing is used to securely store passwords within the database.

The config folder manages database connection settings and environment configurations, enabling secure integration with MongoDB Atlas cloud database.

RESTful API principles are followed throughout the backend, allowing standardized communication between frontend and server. Express.js handles request routing, middleware execution, and response management, ensuring efficient server performance.

This backend architecture enhances security, improves maintainability, and allows future scalability such as microservices integration or cloud deployment.

6. Running the Application

To run the application successfully without any errors , we need to run the both backend and frontend at the same time , both should be in the running state simultaneously. To run these both, we need the commands, the commands are :

For Backend : First navigate into the folder using the command

“cd folder name(backend) “

Next, run the command

“npm start “.

For Frontend: First navigate into the folder using the command

“cd folder name(frontend) “

Next, run the command

“npm run dev “

7. API Documentation

The BookNest backend exposes multiple RESTful API endpoints that handle authentication, seller management, order processing, notifications

, and administrative control. All APIs follow standard HTTP methods such as GET and POST, and responses are returned in JSON format.

Protected routes require JWT authentication middleware to verify user identity and enforce role-based authorization.

1. Authentication & User APIs

1.1 Register User

Endpoint:

POST /api/v1/user/register

Description:

Creates a new user account in the system.

Request Body:


```
{
  "name": "John Doe",
  "email": "john@gmail.com",
  "password": "123456"
}
```


Response:


```
{
  "message": "User registered successfully"
}
```

1.2 Login User

Endpoint:

POST /api/v1/user/login

Description:

Authenticates user credentials and generates a JWT token.

Request Body:


```
{
  "email": "john@gmail.com",
  "password": "123456"
}
```


Response:


```
{
  "success": true,
  "token": "jwt_token_here",
  "user": {
    "_id": "12345",
    "email": "john@gmail.com"
  }
```

}

1.3 Get User Data (Protected)

Endpoint:

POST /api/v1/user/getUserData

Description:

Returns authenticated user profile details.

Authorization Required: JWT Token

1.4 Apply as Seller

Endpoint:

POST /api/v1/user/apply-seller

Description:

Allows a user to submit a seller application.

1.5 Notifications APIs

Get All Notifications

POST /api/v1/user/get-all-notification

Delete All Notifications

POST /api/v1/user/delete-all-notification

Description:

Fetches or clears user notification history.

---

2. Seller APIs

2.1 Get Doctor Info

Endpoint:

POST /api/v1/ /seller/getSellerInfo

Description:

Fetches profile details of logged-in Sellers.

2.2 Update Doctor Profile

Endpoint:

POST /api/v1/seller/updateProfile

Description:

Updates seller profile details such as store name, address, and contact information..

2.3 Get Doctor By ID

Endpoint:

POST /api/v1/seller/getSellerById

Description:

Returns seller information using unique ID.

---

3. Order APIs

3.1 Book Appointment

Place Order

Endpoint:

POST /api/v1/order/place-order

Description:

Creates a new order**.**

Request Body**:**

```
{
  "bookId": "book123",
  "quantity": 1
}
```

Response:

```
{
  "message": "Order placed successfully"
}
```

3.2 Check Doctor Availability

Endpoint:

POST /api/v1/order/check-availability

Description:

Checks available time slots before booking.

3.3 Get User Appointments

Endpoint:

GET /api/v1/order/user-orders

Description:

Returns appointment history for logged-in patient.

3.4 Get seller orders

Endpoint:

GET /api/v1/order/seller-orders

Description:

Returns appointment list for doctor dashboard.

3.5 Update Order Status

Endpoint:

POST /api/v1/order/update-status

Description:

Allows seller to accept or reject customer orders.

---

4. Admin APIs

4.1 Get All Users

Endpoint:

GET /api/v1/admin/getAllUsers

Description:

Fetches all registered users for administrative monitoring.

4.2 Get All Sellers

Endpoint:

GET /api/v1/admin/getAllsellers

Description:

Returns seller list including approval status.

4.3 Change Seller Account Status

Endpoint:

POST /api/v1/admin/changeAccountStatus

Description:

Admin approves or rejects seller applications.

4.4 Change User Status

Endpoint:

POST /api/v1/admin/change-user-status

Description:

Updates user account status within the system.

Authentication Requirement

All protected endpoints require a valid JWT token in request headers:

Authorization: Bearer <token>

JWT middleware verifies identity and enforces role-based access for Admin, Seller, and User dashboards.

8. Authentication

Authentication and Authorization in BookNest

The BookNest application implements a secure authentication and authorization mechanism using JSON Web Tokens (JWT) along with role-based access control. The system ensures that only authenticated users can access protected resources such as appointment booking, doctor dashboards, and administrative controls.

Authentication verifies the identity of users, while authorization determines the level of access based on user roles such as Admin, Seller, and Customer. This layered security approach protects user data and prevents unauthorized actions within the platform..

Authentication Process

Authentication in BookNest follows a token-based architecture. When a user logs in using valid credentials, the backend server verifies the user's email and password stored in the MongoDB Atlas database. Passwords are encrypted using bcrypt hashing to ensure secure storage.

After successful verification, the server generates a JSON Web Token (JWT) that includes essential user information such as the user ID and role. This token is digitally signed using a secret key and returned to the frontend application.

The frontend stores the token securely and attaches it to future API requests that require authentication.

Example header format:

Authorization: Bearer <JWT_Token>

Before processing any protected request, the backend authentication middleware validates the token to ensure it is legitimate and has not expired.

JWT (JSON Web Token) Usage

The JWT used in DocSpot contains encoded payload information including:

- User ID
- User Role (admin, Seller, user)
- Token expiration timestamp

Using JWT enables:

- Stateless authentication without maintaining server sessions.
- Secure request verification without repeated database queries.
- Efficient communication between frontend and backend services.

Each protected API endpoint uses authentication middleware that performs the following steps:

1. Extracts the token from request headers.
2. Verifies the token using a secret key.
3. Decodes the payload data.
4. Attaches authenticated user information to the request object.

If the token is invalid, missing, or expired, the server denies access and returns an unauthorized response.

Authorization (Role-Based Access Control)

BookNest implements Role-Based Access Control (RBAC) to restrict access to sensitive operations based on user roles.

Every user account contains a role field that determines permissions:

- **Customer:** Can browse books, place orders, and manage personal order history.
- **Seller:** Can manage book inventory, process customer orders, and update profile details.
- **Admin:** Can approve seller applications, monitor users, and control system operations.

Authorization middleware checks the role of the authenticated user before allowing access to certain routes.

Examples include:

- Only Admin users can approve doctor accounts or manage system users.
- Only Doctors can update appointment status.
- Only authenticated users can access appointment booking features.

If a user attempts to access a restricted route without proper permissions, the system returns an authorization error.

Session Handling

The BookNest platform uses stateless authentication through JWT tokens instead of traditional server-side sessions. This design improves scalability and performance by eliminating the need to store session data on the server.

Because each request carries authentication information within the token, the backend can validate requests independently without maintaining session state.

Security Measures Implemented

Several security practices are implemented to protect the system:

- Passwords are hashed using bcrypt before storing in MongoDB Atlas.
- JWT tokens are signed with a secure secret key.
- Protected routes validate tokens through authentication middleware.
- Role-based authorization prevents unauthorized access to sensitive endpoints.
- Token expiration mechanisms automatically invalidate inactive sessions.

These security measures ensure that BookNest maintains data confidentiality, integrity, and controlled access throughout the application lifecycle.

9. User Interface:

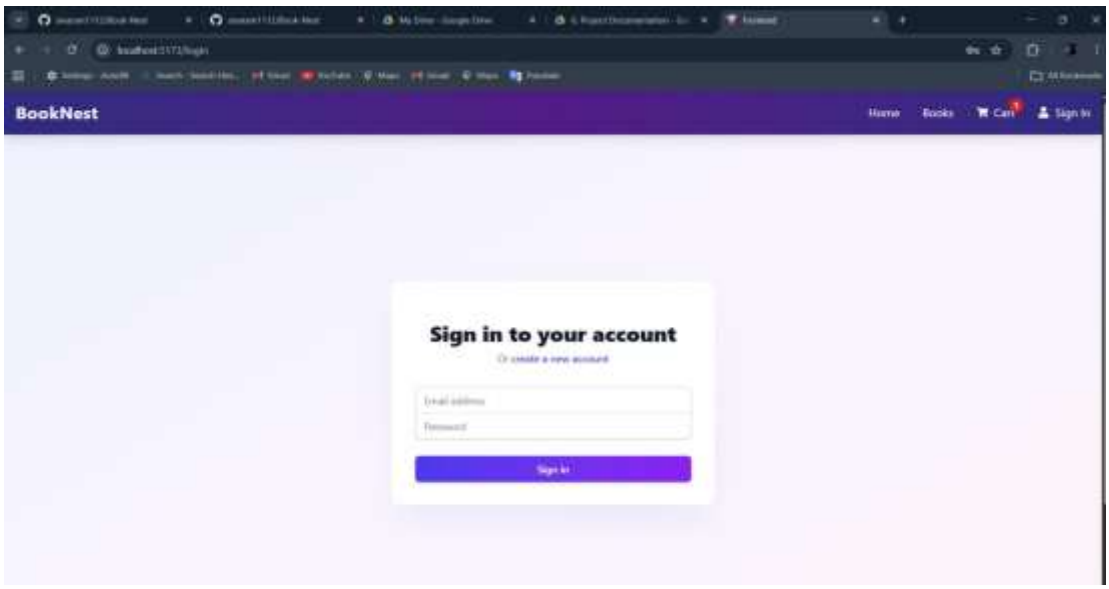


Fig-4:login page

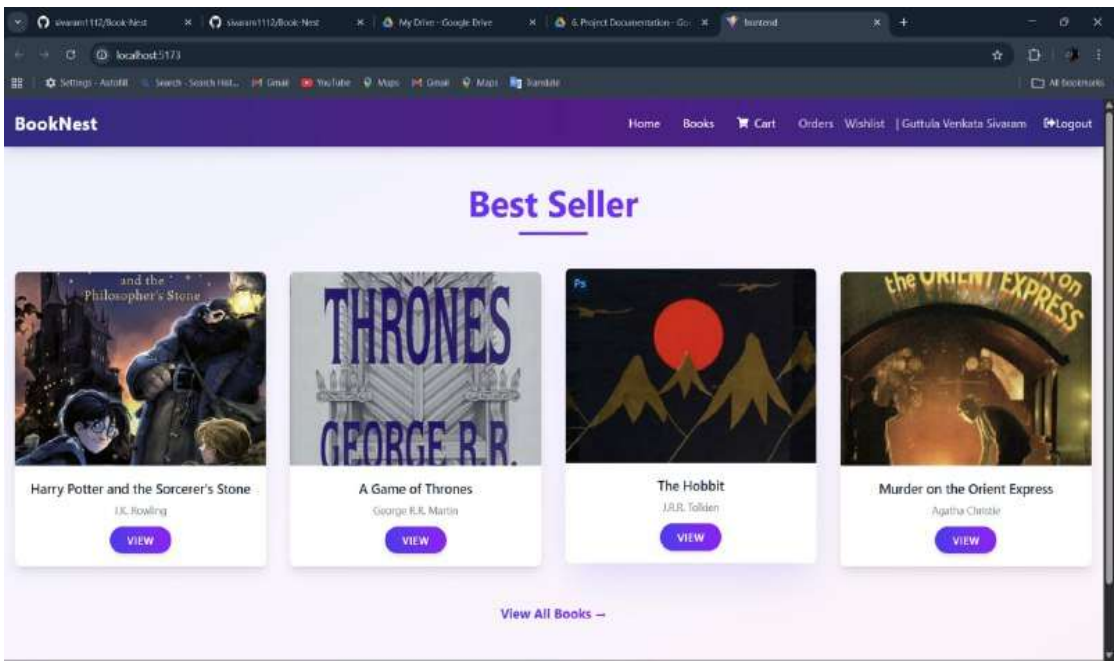


Fig-5: User Dashboard

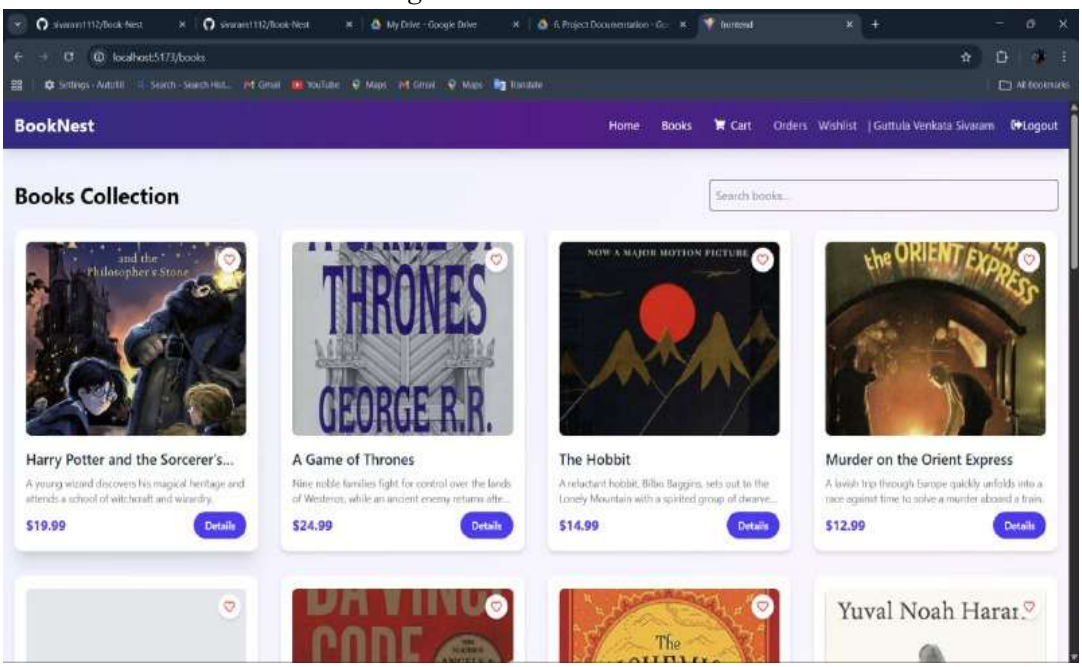


Fig-6: User Book collection Display

10. Testing

Testing Strategy and Tools Used

Testing in the ShopSmart project was performed to ensure that the application functions correctly, securely, and efficiently across both frontend and backend components. The testing strategy included manual testing, functional testing, API testing, and validation testing.

1. Manual Testing

Manual testing was performed throughout development to verify the correctness of features. Each module, such as authentication, product management, cart operations, and order processing, was tested individually. Test cases were created to check valid inputs, invalid inputs, and edge cases.

Examples:

- Testing login with correct and incorrect credentials.
- Checking product search and category filtering.
- Verifying order status updates by admin.
- Ensuring cart updates correctly when items are added or removed.

2. Functional Testing:

Functional testing was conducted to ensure that all system functionalities behave according to the requirements. Each feature was tested from a user perspective to confirm that expected outputs were generated for given inputs.

Modules tested include:

- User Registration and Login
- Book CRUD operations
- Cart management
- Checkout and order placement
- Admin dashboard controls
- PDF report generation

3. API Testing

Backend APIs were tested using tools such as:

- Postman (for testing REST API endpoints)
- Browser developer tools (Network tab)

Each API endpoint was tested with:

- Valid request data
- Missing fields
- Unauthorized access attempts
- Invalid tokens

This ensured proper error handling and secure API behavior.

4. Authentication and Authorization Testing

JWT token verification was tested by:

- Accessing protected routes without a token.
- Using expired or invalid tokens.
- Attempting admin operations with normal user accounts.

The system correctly restricted unauthorized access, confirming proper role-based control.

5. UI Testing

Frontend UI testing was performed to ensure:

- Proper rendering of components.
- Responsive layout behavior.
- Correct display of product images and stock status.
- Real-time search suggestions.
- Error message display for failed operations.

6. Error Handling Testing

The system was tested for:

- Server errors
- Database connection failures
- Invalid inputs
- Network interruptions

Proper error messages were displayed to the user without crashing the application.

Tools Used for Testing

- Postman – for API endpoint testing
- Browser Developer Tools – for debugging frontend
- MongoDB Compass – for verifying database records
- Console Logging – for debugging application flow

11. Screenshots or Demo

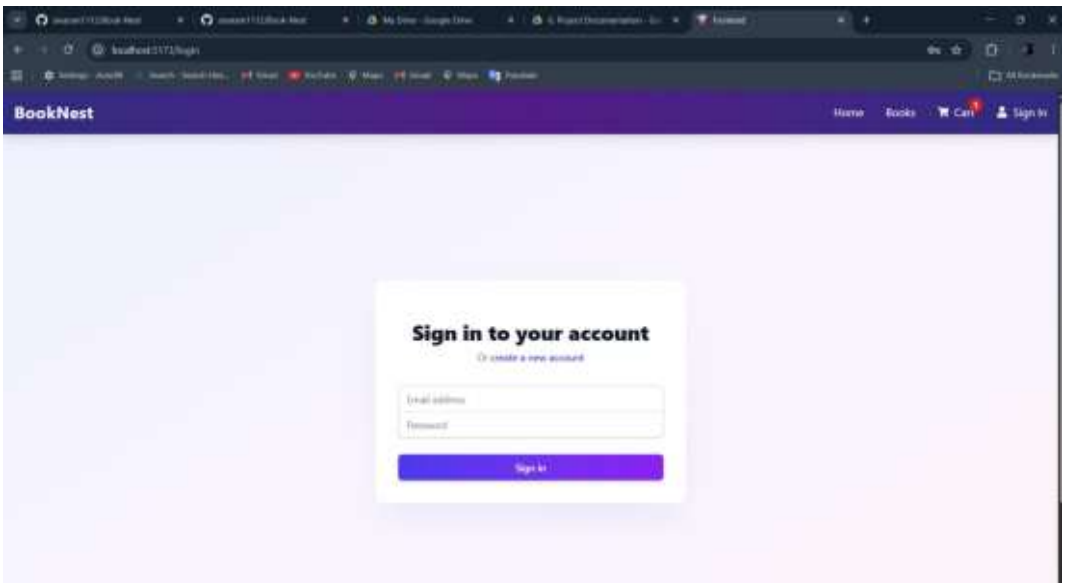


Figure-7:Login page with user credentils

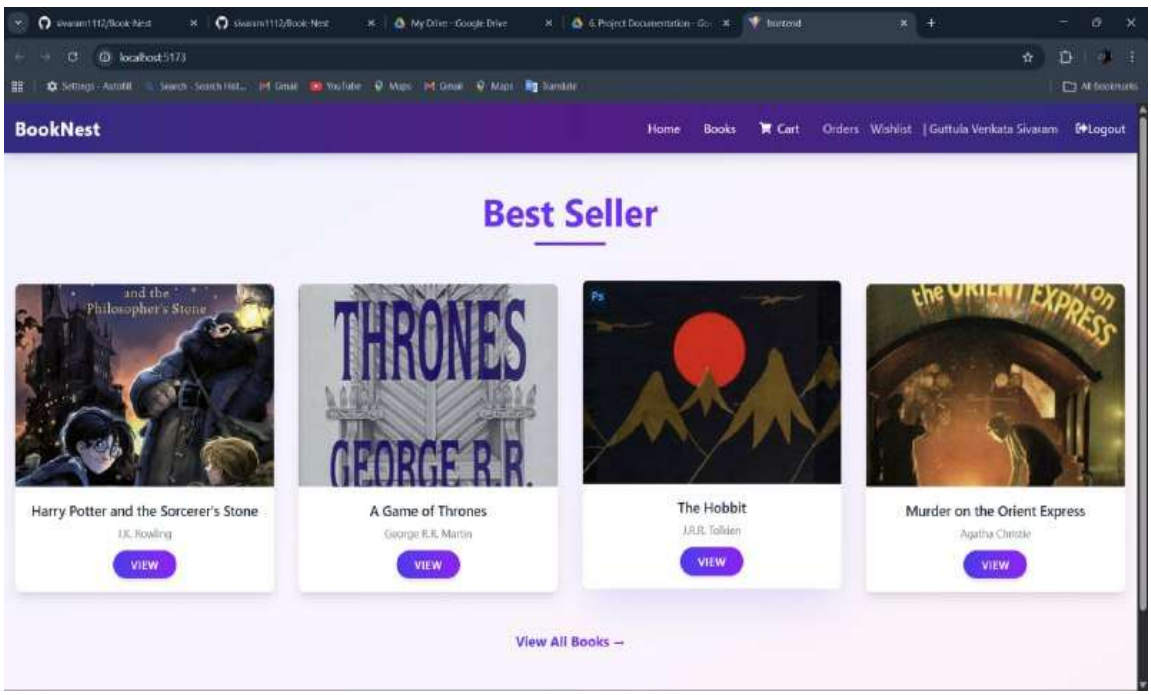


Figure-8: user Dashboard

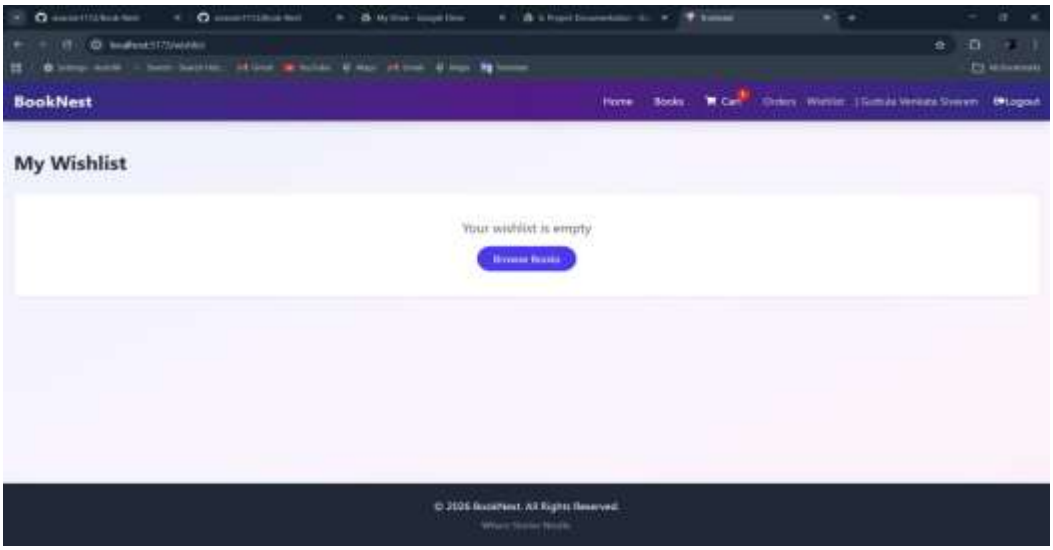
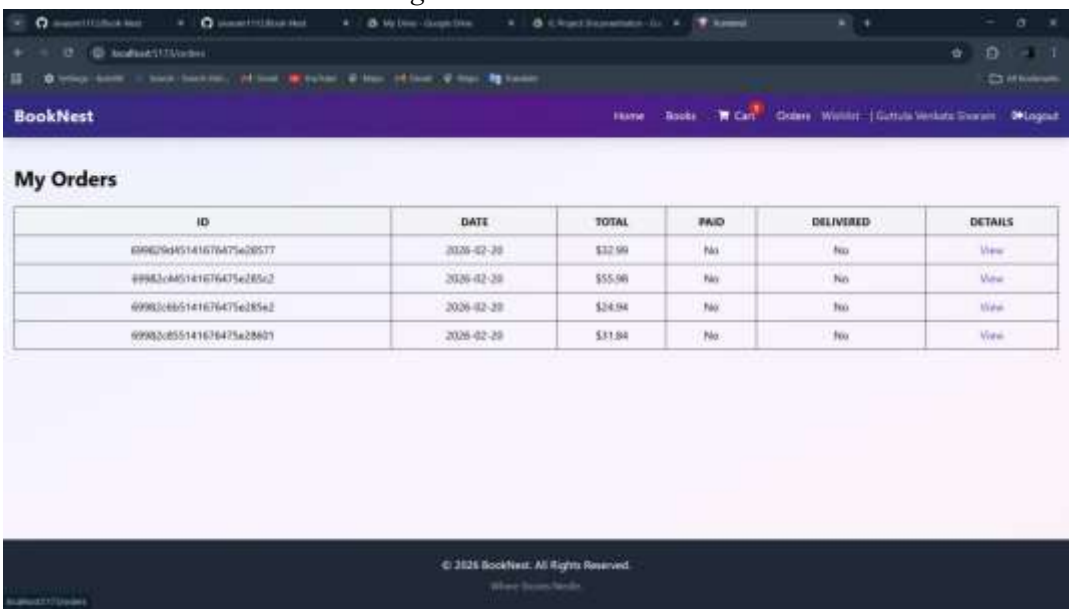


Figure-9: User collection



F[g-10:Order History

12. Known Issues

Although the BookNest application successfully implements the core functionalities of an online bookstore platform, there are certain limitations and known issues that may affect usability and scalability.

1. No Real-Time Appointment Updates

Currently, appointment status updates are reflected only after page refresh. The system does not include real-time notification mechanisms such as WebSockets or push services.

Impact:

Users may not immediately see updates when a seller processes or updates an order status.

2. JWT Token Storage in Local Storage

Authentication tokens are stored in browser local storage for maintaining login sessions.

Impact:

While suitable for development and academic projects, storing tokens in local storage may expose potential security risks such as XSS attacks in production environments.

3. Limited Mobile Responsiveness

The user interface is optimized mainly for desktop usage. Some layouts may not fully adapt to smaller mobile screens.

Impact:

Mobile users may experience minor UI alignment issues.

4. Image Upload Performance

Uploading medical documents during appointment booking depends on network speed and file size.

Impact:

Large files may take longer to upload, affecting user experience.

5. No Payment Integration

The current system focuses only on appointment scheduling and does not include online payment or billing features.

Impact:

The application is suitable for demonstration and learning purposes but requires payment gateway integration for real-world deployment.

13. Future Enhancements

Although the DocSpot platform successfully implements the core functionality of an online doctor appointment booking system, several enhancements can be introduced in future versions to improve usability, security, scalability, and overall healthcare experience.

1. Digital Reading and E-Book Support

Future versions of BookNest can include e-book downloads or online reading functionality, allowing users to access purchased books digitally without physical deliver

2. Improved User Interface and Experience

The current design focuses mainly on functionality. Future updates may introduce better mobile responsiveness, smoother animations, and accessibility enhancements using advanced UI frameworks to provide a more engaging shopping experience

3. Social Login Integration

Adding OAuth-based login options such as Google authentication can simplify the registration process and improve user convenience while maintaining secure access control.

4. Real-Time Notifications System

Implementing real-time notifications using technologies such as WebSockets or Firebase Cloud Messaging would allow users to instantly receive updates regarding appointment confirmations, cancellations, and doctor responses.

5. Online Payment Integration

Future enhancements may include integration of secure digital payment systems such as UPI, debit/credit cards, or online wallets. This feature would enable patients to pay consultation fees directly through the platform, making the system more production-ready.

6. Advanced Security Improvements

Instead of storing JWT tokens in local storage, future versions may adopt HTTP-only cookies and stronger security practices to reduce vulnerabilities such as cross-site scripting attacks.

.

7. Performance Optimization

- System performance can be further improved through:
- Database indexing for faster book and order queries.
- Backend caching techniques to reduce repeated API calls.
- Lazy loading of frontend components to improve initial page load time.
- Pagination for large book listings and order records..

8. Mobile Application Development

A dedicated mobile application using React Native or Flutter can extend BookNest's accessibility to Android and iOS users, enabling appointment booking and doctor communication on mobile devices.

Project Video Demo

The Demo of the Application is available at:
https://drive.google.com/file/d/1zdVhn28XzizmyGoCftTwDqmyjpDuQjOF/view?usp=sharing