

# **Smart Bridge Internship**

## **Full Stack Developer(Mern Stack)**

Team ID: LTVIP2025TMID53034

Team Members:

Team Leader: Inkollu Loukika

Team member: Gunde Nithin Roy

Team member: Guntupalli Srivalli

Team member: Guntupalli Venkata Sal Jyoshnavi

### **Introduction**

The rise of digital platforms has transformed how books are sold and bought online. However, challenges still exist for both book sellers and readers in discovering relevant listings, ensuring transaction safety, and managing inventories or orders. BookNest is a user-centric e-commerce platform built to bridge this gap by simplifying book buying, selling, and browsing.

Built using the MERN Stack—MongoDB, Express.js, React.js, Node.js—BookNest delivers a dynamic and scalable web experience for independent sellers and avid readers alike.

### **Project Overview**

- BookNest is a multi-role e-commerce platform tailored to buying and selling books online. With dedicated features for buyers, sellers, and administrators, the platform supports the full book commerce lifecycle from listing to delivery.

### **User Roles and Key Functions**

**Sellers:** Upload book listings (title, author, price, cover), manage inventory, and view order statuses.

**Admins:** Approve seller accounts, monitor site activity, manage disputes and control listings.

### **Core Features**

Real-time book browsing and search filters (genre, author, price)

Buyer & seller registration/login using JWT authentication

Role-based dashboards for each user category

Order placement, cart, and cancellation features

Admin approval of seller accounts

Notifications for order updates

Responsive UI for desktop/mobile/tablet

### **Purpose**

The goal of BookNest is to empower local sellers and offer readers an organized, secure, and seamless book shopping experience.

#### **Additional goals include:**

Streamline book buying with filters and instant checkout

Provide user authentication and secure transactions

Enhance discoverability of books for buyers

Manage book inventories and orders for sellers

Equip admins with tools for oversight and moderation

### **Ideation Phase**



## Problem Statement:

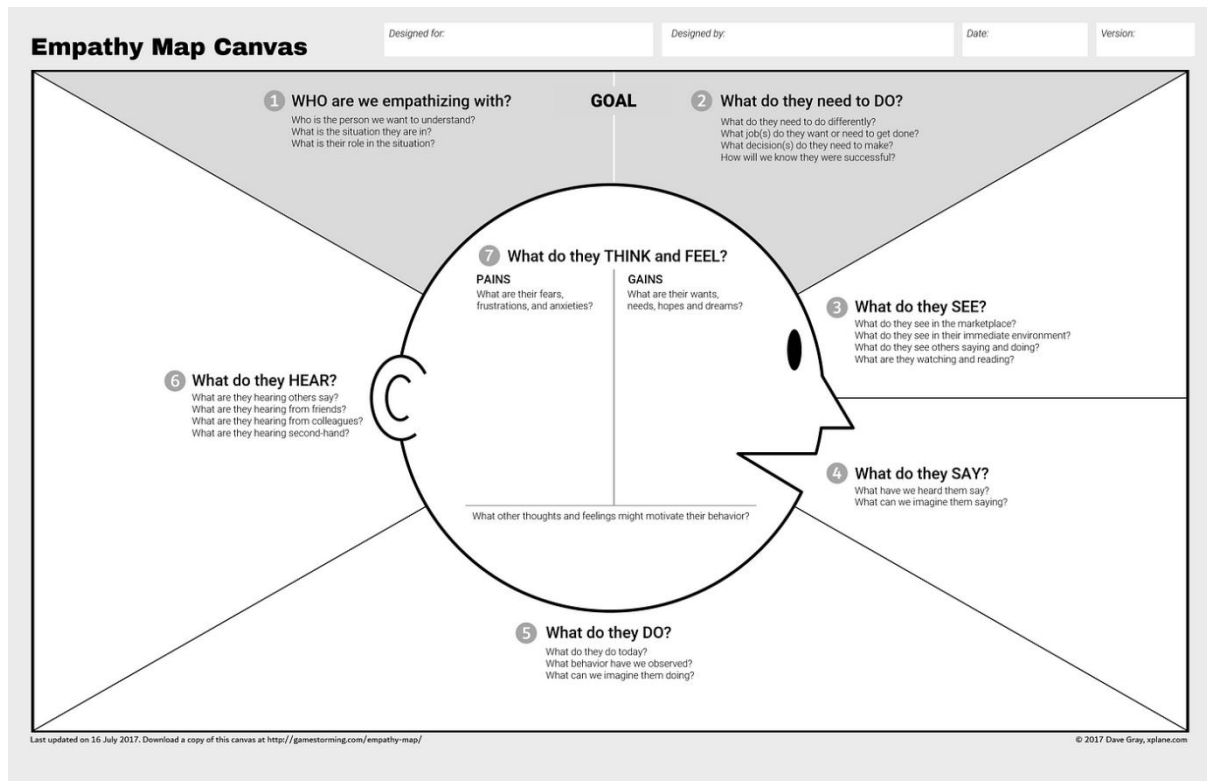
1. Book lovers find it hard to discover niche or second-hand books in crowded marketplaces.
2. Independent sellers lack a dedicated platform for organized book listings.
3. Manual or disorganized order management leads to delayed deliveries and bad experiences.

## Proposed Solution:

1. Develop a dedicated book-selling platform using the MERN stack.
2. Features for:
3. Buyers: Register, browse, search, order, track, and cancel.
4. Sellers: Upload books, manage listings, check orders.
5. Admins: Approve sellers, manage users, resolve issues.
6. Add real-time UI updates, role-based dashboards, and responsive design.

## Empathy Map Canvas

This empathy map is based on the primary user: **the customer:**



## Brainstorming & Idea Prioritization

### Objective:

The objective of this phase was to explore, design, and prioritize innovative ideas to build a seamless, user-friendly online bookstore platform. Through collaborative brainstorming and strategic planning, the team focused on addressing common issues in book purchasing and inventory management while creating a scalable digital foundation for future e-commerce expansion.

### Key Activities:

#### **Step 1: Team Collaboration and Problem Statement Selection**

##### Team Collaboration and Problem Statement Selection

The team assembled on [Insert Start Date] to analyze gaps in traditional and online book-selling platforms.

**Through research and shared user experiences, the final problem statement was defined:**

"Traditional bookstore systems lack real-time inventory, user convenience, and personalized experience, resulting in poor engagement and missed sales opportunities."

## **Step 2: Brainstorming, Idea Listing, and Grouping**

**All team members contributed ideas openly, which were then categorized into three core focus areas:**

### **User-Centered Features:**

Role-based registration/login (Admin, Seller, Buyer)

Book browsing with filters (category, price, author, rating)

Add to wishlist/cart functionality

Order history and tracking

Reviews and ratings for books

### **Seller-Centered Features:**

Add/update/remove books with images and details

View order requests and manage stock

Access analytics for top-selling books and customer trends

### **Admin-Centered Features:**

Approve or reject seller registrations

Manage all users and books

Monitor transactions and user feedback

Generate platform-wide reports

## **Step 3: Idea Prioritization**

The team used an impact–effort matrix to identify features for the Minimum Viable Product (MVP).

Priority was given to high-impact, low-effort functionalities to ensure a quick and effective launch.

### **Selected for MVP (Phase-1):**

Role-based registration/login (Admin, Seller, Buyer)

- Book listing with category filters
- Add to cart and checkout
- Admin approval of seller accounts
- Order tracking for users and sellers
- Dashboards for all user

### **Deferred for Future Phases:**

Recommendation system based on purchase history

- Real-time chat with sellers
- AI-based personalized book suggestions
- Loyalty points and discounts
- Return/refund management system

## **Customer Journey Map**

**Purpose:** To visualize and improve the user experience when browsing, reserving, and borrowing books online, whether from a public library, campus collection, or community-sharing system.

### **Key Components:**

#### **1. User Actions:**

- Registering and logging into the platform.
- Browsing books by category, author, availability.
- Reserving or borrowing books online.
- Viewing borrowing history and due dates.
- Returning books and giving ratings/reviews.

#### **2. User Thoughts and Emotions:**

- **Before:** Frustration with unavailable books or unclear availability.
- **During:** Joy in discovering and reserving desired titles quickly.
- **After:** Satisfaction or dissatisfaction based on timely returns and reminders.

#### **3. Touchpoints:**

- Home page, category filters, book detail pages.
- Reservation form and confirmation pages.
- Email/SMS alerts for due dates and reminders.
- User dashboard for tracking status and history.

#### **4. Pain Points and Opportunities:**

- Difficulty finding books → Implement smart search & filters.
- Overdue penalties → Enable timely reminders.

- Limited availability → Waitlist/reservation feature.

**Insight:** Mapping this journey highlights where users get stuck and helps streamline the borrowing/reservation process with personalization, reminders, and better navigation.

## Solution Requirements

### 1. Functional Requirements:

These define what the system should do.

- User Registration/Login with role-based access (Reader, Admin).
- Browse/search books with filters (genre, availability, title, author).
- Book reservation system with calendar-based pickup/return slot.
- Reader dashboard to view current and past borrows, due dates.
- Admin dashboard to manage books, returns, and users.
- Notification system for reminders and overdue alerts.
- Feedback system for book reviews and ratings.

### 2. Non-Functional Requirements:

- Responsive interface (mobile/tablet/desktop).
- Optimized search and minimal API response times.
- Secure data handling and role-based access control.
- Easy maintainability and scalability.

### 3. System Requirements:

- **Frontend:** React.js with Bootstrap/Tailwind.
- **Backend:** Node.js + Express.js.
- **Database:** MongoDB with Mongoose.
- **Authentication:** JWT + bcrypt.js.
- **Hosting:** Render or Vercel with GitHub Actions CI/CD.
- **Notifications:** NodeMailer or Twilio.

## Data Flow :

A **Data Flow Diagram (DFD)** illustrates how data moves within the Freelance Finder platform. It captures how users (freelancers and clients) interact with the system, how information flows between different components, and where the data is stored.

### As a User...

- I can log in securely and access my dashboard.
- I can browse or search for books by different criteria.
- I can reserve books and view my borrowing queue.

- I get reminders before due dates to avoid penalties.
- I can give feedback and rate books I've read.

### **As an Admin...**

- I can add, edit, or delete book records.
- I can approve/track user reservations and returns.
- I can manage late returns and generate reports.
- I can view usage analytics and system logs

## **SETUP INSTRUCTIONS:**

### **1. JWT-Based Authentication**

- Uses JSON Web Tokens (JWT) for secure login sessions.
- Tokens are stored securely (typically in HTTP-only cookies or local storage).
- Tokens are validated for every protected route to prevent unauthorized access.

### **2. Role-Based Access Control (RBAC)**

Access to routes and features is restricted based on user roles: Patients can book/view appointments.

Doctors can manage appointments.

Admins can manage users and system data.

Unauthorized role attempts are blocked at the backend.

### **3. Password Hashing**

- User passwords are hashed using bcrypt before storing in the database.
- Even if the database is compromised, passwords remain unreadable.

### **4. Input Validation and Sanitization**

- All user input is validated using libraries like express-validator or custom middleware.
- Protects against SQL injection (though MongoDB is NoSQL) and XSS attacks.

### **5. Protected API Routes**



- Sensitive API endpoints (e.g., /book-appointment, /admin/approve-doctor) are protected using JWT middleware.
- Only authenticated users with valid tokens can access them.

## 6. CORS and CSRF Protection

- Configured CORS policy to allow only trusted domains to access the API.
- Optional: Use CSRF protection tokens if storing JWTs in cookies.

## 7. Rate Limiting and Brute Force Protection

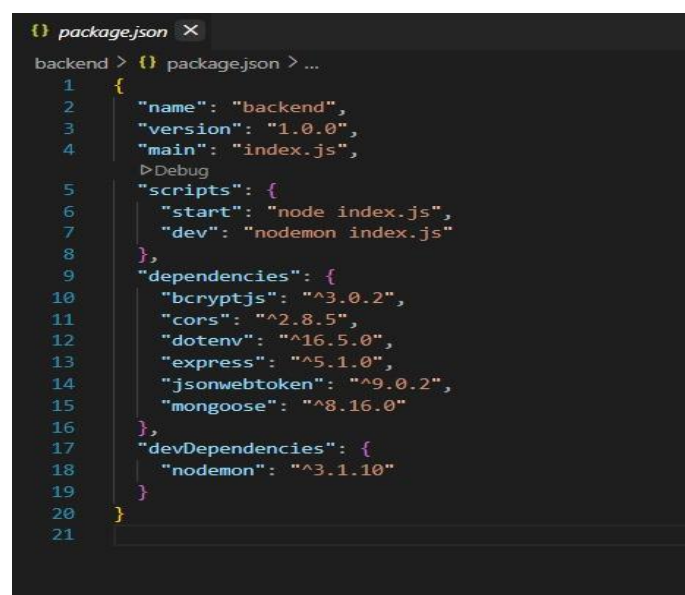
- Implements rate limiting middleware (e.g., express-rate-limit) to prevent brute force login attempts.
- Can be extended to include account lockout after repeated failed logins.

## 8. Logging and Monitoring

- Logs suspicious activities (like repeated failed logins or unauthorized access attempts).
- Admin panel can monitor user activities and detect anomalies.

## 9. HTTPS Deployment (Production)

- In production, the app should be served over HTTPS to encrypt data in transit.
- 



```
package.json X
backend > {} package.json > ...
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "main": "index.js",
5    "scripts": {
6      "start": "node index.js",
7      "dev": "nodemon index.js"
8    },
9    "dependencies": {
10     "bcryptjs": "^3.0.2",
11     "cors": "^2.8.5",
12     "dotenv": "^16.5.0",
13     "express": "^5.1.0",
14     "jsonwebtoken": "^9.0.2",
15     "mongoose": "^8.16.0"
16   },
17   "devDependencies": {
18     "nodemon": "^3.1.10"
19   }
20 }
21
```

```
{} package.json X
frontend > {} package.json > ...
1  {}
2  "name": "frontend",
3  "version": "0.1.0",
4  "private": true,
5  "dependencies": {
6    "@emotion/react": "^11.14.0",
7    "@emotion/styled": "^11.14.0",
8    "@mui/material": "^7.1.2",
9    "@testing-library/dom": "^10.4.0",
10   "@testing-library/jest-dom": "^6.6.3",
11   "@testing-library/react": "^16.3.0",
12   "@testing-library/user-event": "^13.5.0",
13   "axios": "^1.10.0",
14   "bootstrap": "^5.3.7",
15   "react": "^19.1.0",
16   "react-bootstrap": "^2.10.10",
17   "react-dom": "^19.1.0",
18   "react-icons": "^5.5.0",
19   "react-router-dom": "^7.6.2",
20   "react-scripts": "5.0.1",
21   "web-vitals": "^2.1.4"
22 },
23   > Debug
24   "scripts": {
25     "start": "react-scripts start",
26     "build": "react-scripts build",
27     "test": "react-scripts test",
28     "eject": "react-scripts eject"
29   },
30   "eslintConfig": {
31     "extends": [
32       "react-app",
33       "react-app/jest"
34     ]
35   },
36   "browserslist": {
37     "production": [
38       ">0.2%",
39       "not dead",
40       "not op_mini all"
41     ],
42     "development": [
43       "last 1 chrome version",
44       "last 1 firefox version",
45       "last 1 safari version"
46     ]
47   }
48 }
```

## PRE-REQUISITES:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, and React.js:

- **Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server side.

Download: <https://nodejs.org/en/download/>

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

### **npm init**

- **Express.js:**

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

### **npm install express**

- **MongoDB:**

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

Download: <https://www.mongodb.com/try/download/community>

Installation instructions: <https://docs.mongodb.com/manual/installation/>

- **Moment.js:**

Moment.js is a JavaScript package that makes it simple to parse, validate, manipulate, and display date/time in JavaScript. Moment.js allows you to display dates in a human-readable format based on your location. Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://momentjs.com/>

- **React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

- **Antd:**

Ant Design is a React.js UI library that contains easy-to-use components that are useful for building interactive user interfaces. It is very easy to use as well as integrate. It is one of the smart options to design web applications using react.

Follow the installation guide: <https://ant.design/docs/react/introduce>

- **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. To Connect the Database with Node JS go through the below provided link:

<https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

- **Front-end Framework:** Utilize Reactjs to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard.

For making better UI we have also used some libraries like material UI and bootstrap.

Install Dependencies:

- Navigate into the cloned repository directory:

```
cd book-a-doctor
```

- Install the required dependencies by running the following commands:

```
cd frontend
```

```
npm install
```

```
cd ../backend
```

```
npm install
```

Start the Development Server:

- To start the development server, execute the following command:

```
npm start
```

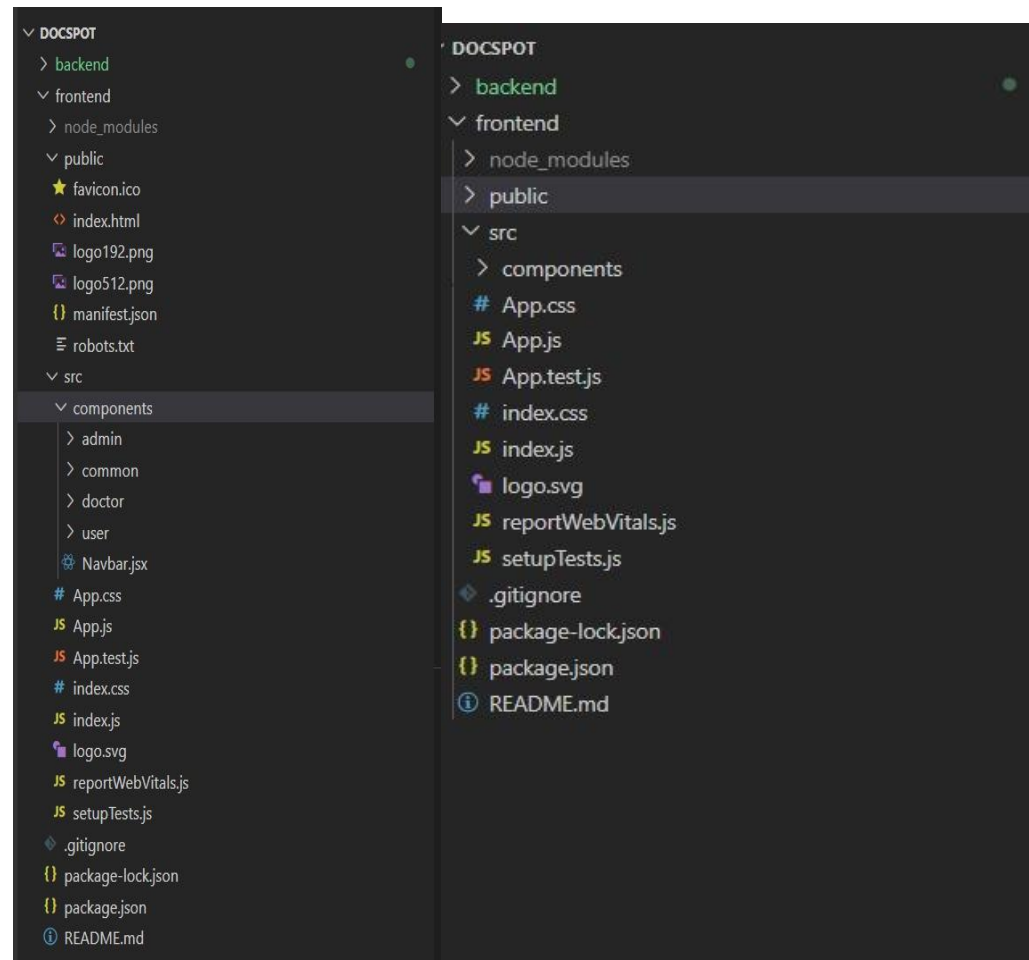
- The book a doctor app will be accessible at <http://localhost:3000>

You have successfully installed and set up the online complaint registration and management app on your local machine. You can now proceed with further customization, development, and testing.

## FOLDER STRUCTURE:

Here's a typical and clean folder structure for a "Book a Doctor Using MERN" application, organized into frontend (client) and backend (server) folders:

### Project Folder Structure



### Client(Frontend):

Here's a detailed breakdown of the client/ folder in your "Book a Doctor Using MERN" app — which houses the React frontend.

Here's a clear and complete explanation of the **structure of the React frontend** in your **"Book a Doctor Using MERN"** project.

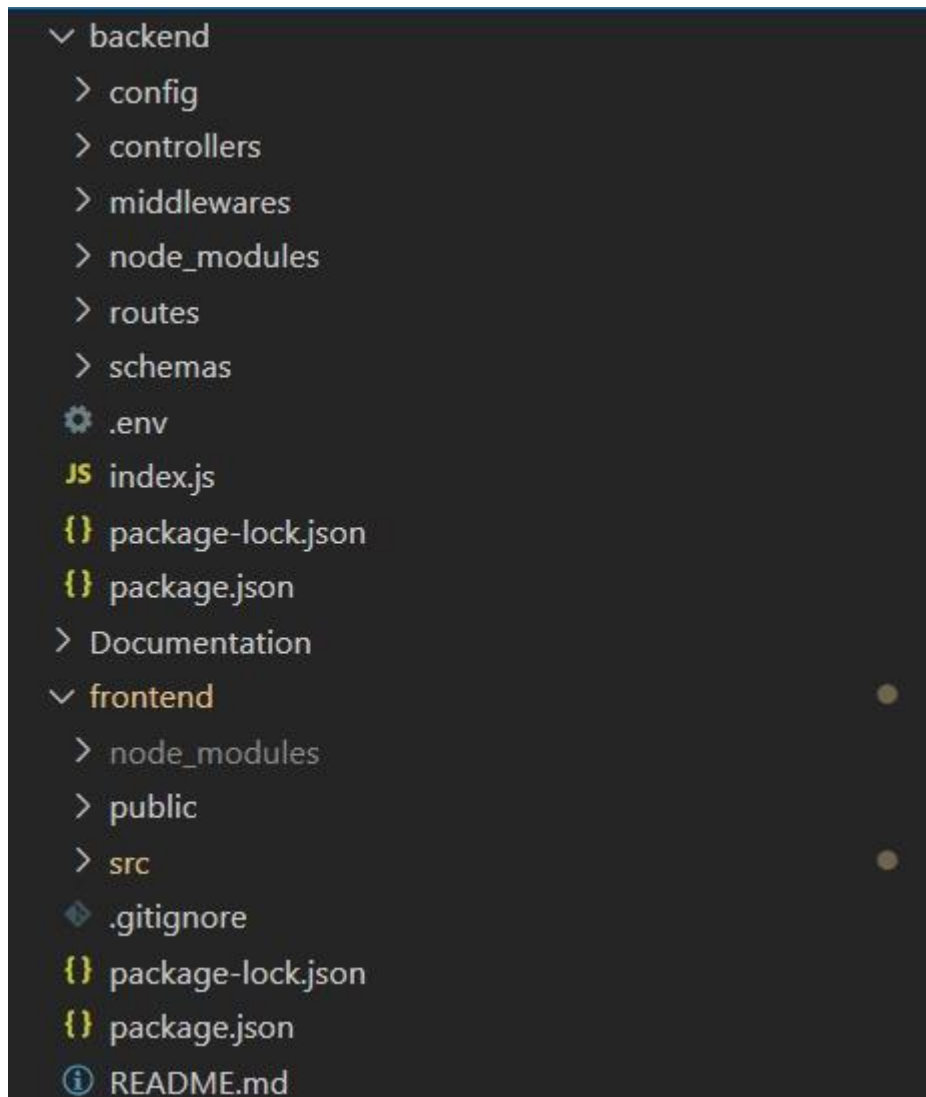
**React Frontend Structure Overview** The React frontend is organized in a modular, scalable way to support different user roles (Patient, Doctor, Admin) and features like authentication, appointment booking, and dashboards

- **node\_modules/** – Auto-generated folder with all installed packages.
- **public/** – Static files like index.html.
- **src/** – Main folder for all your code.

**Inside src/components/:**

**admin/ – Admin pages:**

- AdminAppointments.jsx – Manage appointments
- AdminDoctors.jsx – Manage doctorsAdminHome.jsx – Admin dashboard



**Server(Backend):**

Here's a clear explanation of the **organization of the Node.js backend** in your "**Book a Doctor Using MERN**" project. This structure is modular, scalable, and follows common best practices for building RESTful APIs with Express and MongoDB.

Node.js Backend Structure

## Detailed Folder Breakdown

### config/

- Stores configuration files.
- db.js: Connects to MongoDB using Mongoose.
- jwt.js: JWT secret and token generation logic.

### controllers/

- Handles the logic for each route.

Each file maps to a feature: authController.js: Handles login, registration.

doctorController.js: Manages doctor profiles and availability.

appointmentController.js: Booking and appointment logic.

Each file maps to a feature: authController.js: Handles login, registration.

doctorController.js: Manages doctor profiles and availability.

appointmentController.js: Booking and appointment logic.

### middleware/

Functions that run before route handlers. authMiddleware.js: Verifies JWT tokens.

roleMiddleware.js: Restricts access based on user role (admin, doctor, patient).

errorHandler.js: Global error handling middleware.

### models/

Mongoose schemas for MongoDB collections: User.js: Common user data.

Doctor.js: Doctor-specific fields (specialization, availability).

Appointment.js: Appointment data (time, patient, doctor, status).

### routes/

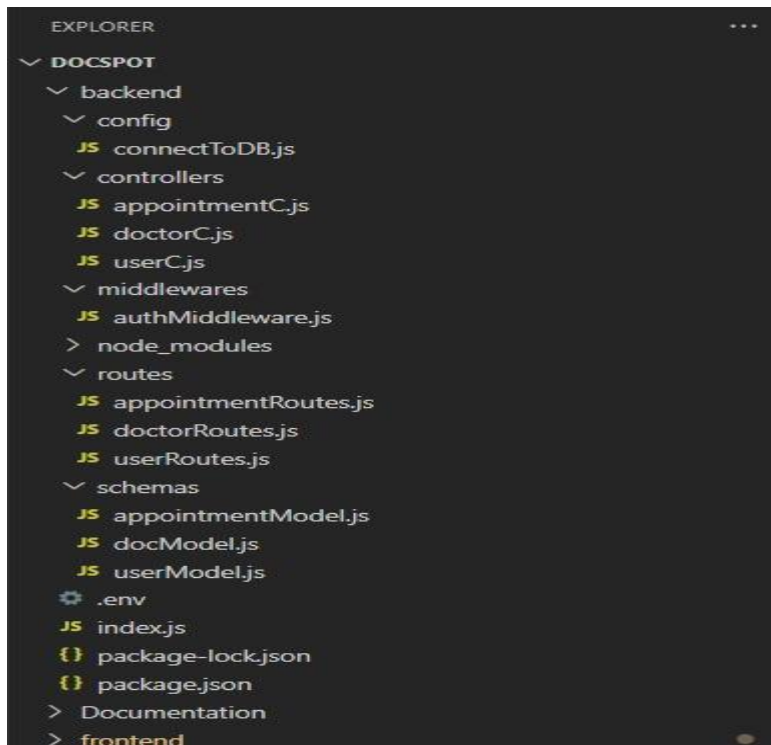
- Defines Express route endpoints.
- Imports relevant controllers and middleware.

Examples: authRoutes.js: /api/auth/login, /register

doctorRoutes.js: /api/doctor/update-profile

patientRoutes.js: /api/patient/book-appointment

adminRoutes.js: /api/admin/get-all-users



## API DOCUMENTATION:

Here's a well-organized **API Documentation** for your "**Book a Doctor Using MERN**" application's backend. It includes **HTTP methods**, **endpoints**, **parameters**, and **example responses** for major features.

### **API Documentation – Book a Doctor Using MERN**

**Base URL:** <http://localhost:5002/api>

## AUTHENTICATION APIS:

### 1. Register User

- POST /auth/register
- Body:

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



- Response:

```
{  
  
  "success": true,  
  
  "message": "User registered successfully",  
  
  "token": "JWT_TOKEN"  
  
}
```

## 2. Login User

- POST /auth/login
- Body:

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

- Response:

```
{  
  
  "success": true,  
  
  "message": "Login successful",  
  
  "token": "JWT_TOKEN"  
  
}
```

User APIs

## 3. Get User Profile

- GET /user/profile
- Headers: Authorization: Bearer JWT\_TOKEN
- Response:

```
{  
  
  "name": "John Doe",  
  
}
```

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

```
"role": "patient"
```

```
}
```

Doctor APIs

#### **4. Apply as Doctor**

- POST /doctor/apply
- Headers: Authorization: Bearer JWT\_TOKEN
- Body:

```
{
```

```
"specialization": "Cardiology",
```

```
"experience": "5 years",
```

```
"availableTimes": ["09:00", "17:00"]
```

```
}
```

- Response:

```
{
```

```
"success": true,
```

```
"message": "Doctor application submitted"
```

```
}
```

#### **5. Get Doctor Info**

- GET /doctor/:id
- Params: id – Doctor ID
- Response:

```
{
```

```
"name": "Dr. Smith",
```

```
"specialization": "Dermatology",  
  
"availableTimes": ["09:00", "17:00"]  
  
}
```

## 6. Book Appointment

- POST /appointments/book
- Headers: Authorization: Bearer JWT\_TOKEN
- Body:

```
{  
  
"doctorId": "doctor123",  
  
"date": "2025-07-01",  
  
"time": "10:00"  
  
}
```

- Response:

```
{  
  
"success": true,  
  
"message": "Appointment booked successfully"  
  
}
```

## 7. Get User Appointments

- GET /appointments/user
- Headers: Authorization: Bearer JWT\_TOKEN
- Response:

```
[  
  
{  
  
"doctor": "Dr. Smith",  
  
"date": "2025-07-01",  
  
"time": "10:00",
```

```
"status": "pending"
```

```
}
```

## 8. Get Doctor Appointments

- GET /appointments/doctor
- Headers: Authorization: Bearer JWT\_TOKEN
- Response

```
{
```

```
"patient": "John Doe",
```

```
"date": "2025-07-01",
```

```
"time": "10:00",
```

```
"status": "confirmed"
```

```
}
```

Admin APIs

## 9. Get All Users


- GET /admin/users
- Headers: Authorization: Bearer JWT\_TOKEN (Admin only)

## 10. Approve Doctor

- **POST** /admin/approve-doctor/:doctorId

**Authorization:**

**Headers: Authorization: Bearer JWT\_TOKEN (Admin only)**

-  Authentication & Authorization
- The application uses JWT (JSON Web Tokens) for stateless authentication and role-based access control for authorization.

Authentication Flow

- User Registration/Login
- When a user registers or logs in via /auth/register or /auth/login, the server:
- Validates credentials.
- Generates a JWT token signed with a secret key.
- Returns the token to the client.

- Token Structure
- The JWT contains user data:
- {
- "id": "user\_id",
- "role": "patient"
- }
- It is signed using a secret defined in .env:
- JWT\_SECRET=your\_jwt\_secret\_key
- Token Storage
- On the frontend, the token is usually stored in:
- localStorage (or sessionStorage) or as an HTTP-only cookie (for extra security in production).

## Technology Stack:

S.No	Component	Technology Used
1	User Interface	React.js, Bootstrap, Tailwind CSS
2	Routing	React Router DOM
3	API Calls	Axios
4	Auth & Role Management	JWT, bcrypt.js
5	Backend	Node.js, Express.js
6	Database	MongoDB, Mongoose
7	Notifications	NodeMailer / Cron jobs

## Components & Technologies – BookNest

S.No	Component	Description	Technology Used
1	User Interface	Web-based responsive UI for patients, doctors, and admins	React.js, HTML, CSS, JavaScript, Bootstrap, Tailwind CSS
2	Routing & Navigation	Handles client-side page switching and protected routes	React Router DOM, Context API
3	API Communication	Facilitates frontend-backend interaction	Axios (HTTP client)
4	Authentication System	Secure login and role management	JSON Web Tokens (JWT), bcrypt.js
5	Appointment Logic	Booking, status management, and slot scheduling	Node.js, Express.js

6	Admin Panel	Doctor approval, user management, and appointment oversight	React.js (admin views), Express.js (backend logic)
---	-------------	---	--

## Problem–Solution Fit Overview

Parameter	Description
Problem Statement	Users struggle with finding available books and face issues with returns.
Idea / Solution	A centralized platform for browsing, reserving, and managing book lending.
Novel Features	Smart search, waitlist handling, rating system, email reminders.
Social Impact	Encourages reading, improves library access, reduces missed returns.
Revenue Model	Freemium library integrations, subscription for advanced features.
Scalability	Modular design for multi-branch library support and multi-language support.

## Proposed Solution – DocSpot App

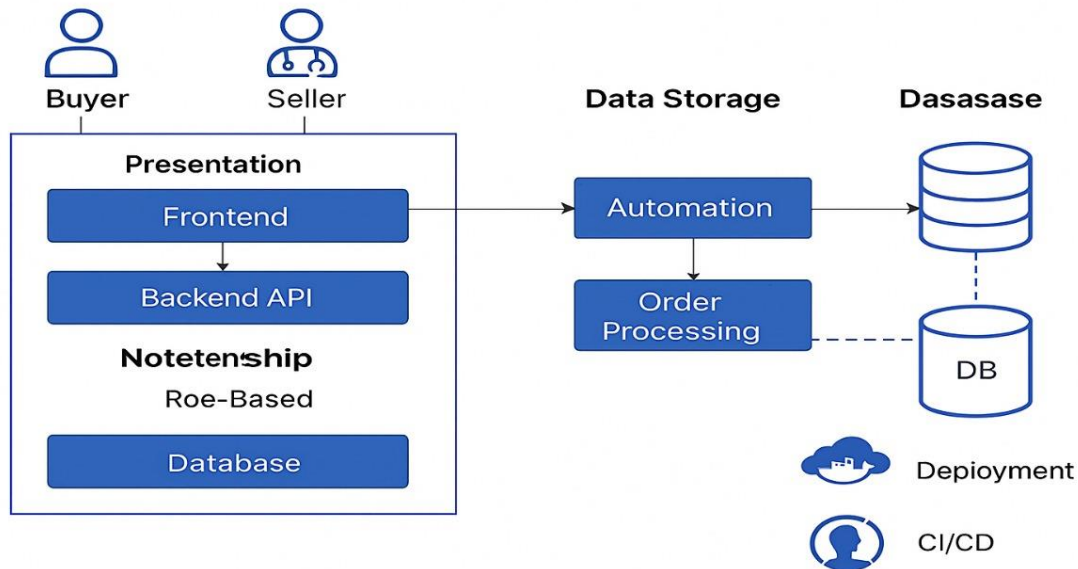
S. No.	Parameter	Description
1	<b>Problem Statement</b>	Book readers and library-goers struggle to find, reserve, or track books efficiently. Admins face issues with managing book inventory, user access, and overdue tracking.
2	<b>Idea / Solution</b>	A full-stack MERN web app for real-time book reservation, user-role access (readers, librarians, admins), return reminders, inventory management, and analytics.
3	<b>Novel Features</b>	Real-time book availability, multi-role system, return date reminders, overdue penalties, review system, digital borrowing for eBooks.
4	<b>Social Impact</b>	Encourages reading habits, digitizes library access, improves resource sharing in underserved or rural areas.
5	<b>Revenue Model</b>	Freemium for users, subscriptions for libraries, pay-per-book for rare content, affiliate eBook sales.
6	<b>Scalability</b>	Supports multilingual content, responsive mobile-first design, microservice-ready backend for future modules (e.g., eBook store, community).

# Solution Architecture

Architecture Principles: Modularity, Security, Scalability, Maintainability

## Solution Architecture

Booknest: Sealmest: Where Stories Nestle



- **Client-Server Model:**
  - **Frontend (Client):** React-based responsive UI for searching, booking, reviewing books, and profile management.
  - **Backend (Server):** Node.js with Express for routing, authentication, business logic, and handling API calls.
- **Component Interaction:**
  - REST APIs connect frontend to backend.
  - Real-time reminders via WebSockets or third-party APIs (email/SMS).
  - Admin portal interacts with book database, inventory, and users.
- **Data Storage:**
  - MongoDB stores user data, book records, reservations, penalties, reviews.
  - Encryption used for sensitive data (e.g., credentials, payment details).
- **Extensibility:**
  - Clearly defined services (user, book, booking, review, notifications).
  - Clean code structure to allow team scaling and feature expansion.

## PROJECT PLANNING & SCHEDULING:

### Project Planning

- **Scope Definition:** Book discovery, reservations, reviews, and library admin dashboard.
- **Task Breakdown:**
  - Module 1: User Authentication

- Module 2: Book Search & Reservation
  - Module 3: Admin & Inventory Management
  - Module 4: Notifications & Penalties
- **Resource Allocation:** MERN stack developers, QA testers, UI/UX designer.
- **Timeline Estimation:** 4–6 weeks for MVP.
- **Milestones:**
  - Week 1: UI & Backend Setup
  - Week 2: Auth Complete
  - Week 3: Book Flow Complete
  - Week 4: Admin Dashboard
  - Week 5: Testing & Polishing
- **Risks & Mitigation:**
  - Book data volume → Use pagination
  - UI delay → Ready-made UI kits (e.g., Material UI)
- **Communication Plan:**
  - Weekly Scrum
  - Jira/Trello for tracking
  - Slack/Email for async communication

## FUNCTIONAL AND PERFORMANCE TESTING:

### Performance Testing

#### Performance Testing Goals

- Ensure users can browse and reserve books without delay.
- Optimize for < 1.5s response for book search.
- Handle concurrent users during peak hours.

#### Testing Types

- **Load Testing:** 1000+ concurrent users reserving books.
- **Stress Testing:** Simulate mass returns & overdue generation.
- **Scalability Testing:** Add 50,000 books and test pagination, search filters.
- **Endurance Testing:** Run system continuously for 48 hours to detect memory leaks or failures.

### Agile Sprint Backlog

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Sprint-1	User Authentication	USN-1	As a user, I can sign up and log in.	3	High
		USN-2	As a user, I can reset my password securely.	2	Medium
Sprint-2	Book Discovery & Reservation	USN-3	As a reader, I can search/filter books by genre, author, or title.	2	High
		USN-4	As a reader, I can reserve an available book.	3	High



Sprint-3	Inventory & Admin Panel	USN-5	As an admin, I can add/edit/delete books and set availability.	3	High
		USN-6	As an admin, I can view overdue books and fine users.	2	High
Sprint-4	Notifications & Reviews	USN-7	As a user, I receive reminders for book return deadlines.	2	Medium
		USN-8	As a user, I can rate/review books I've read.	2	High
		USN-9	As a user, I can download eBooks (if available).	2	Medium

## RESULTS:

### Output Screenshots

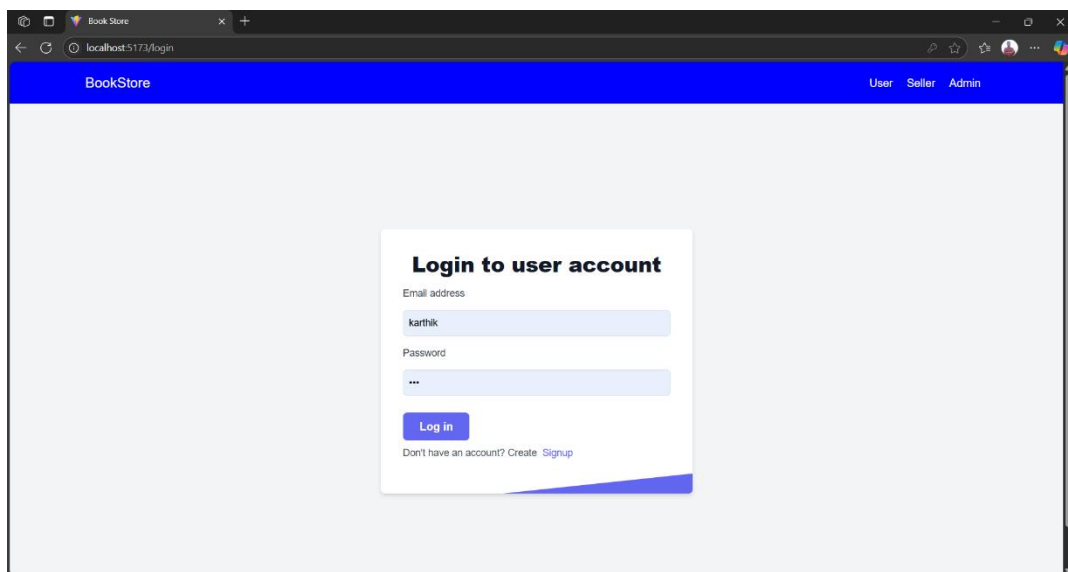
#### Links :

[https://docs.google.com/document/d/1eM6wTVcuQzAgSisKO7fi8Wf3tyAnT698/edit?usp=drive\\_link&oid=103014717462452408435&rtpof=true&sd=true](https://docs.google.com/document/d/1eM6wTVcuQzAgSisKO7fi8Wf3tyAnT698/edit?usp=drive_link&oid=103014717462452408435&rtpof=true&sd=true)

### LANDING PAGE :



## LOGIN PAGE FOR USER:



## LOGIN SUCCESS

## LOGIN PAGE FOR SELLER:

The screenshot shows a web browser window with the address bar displaying 'localhost:5173/login'. The page has a blue header with the text 'BookStore' and navigation links for 'User', 'Seller', and 'Admin'. The main content area features a white login form titled 'Login to Seller account'. The form includes an 'Email address' field with the value 'karthik', a 'Password' field with three dots indicating it is masked, and a blue 'Log in' button. Below the button, there is a link that says 'Don't have an account? Create Signup'.

## REGISTER FOR USER

The screenshot shows a white signup form titled 'Signup' centered on a light gray background. The form contains three input fields: 'Name', 'Email address', and 'Password', each with a placeholder text of the same name. Below these fields is a blue 'Signup' button. At the bottom of the form, there is a link that says 'Already have an account Login'.

## REGISTER FOR SELLER

## Seller Registration

Name

Email address

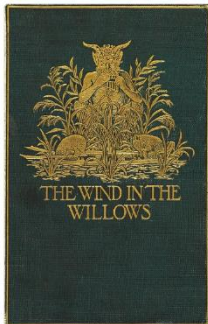
Password

Signup

Already have an account [Login](#)

**BACK TO DASHBOARD**

### Top Recommendation



THE WIND IN WILLOWS



ONE DARK WINDOW



SORCERY OF THORNS





ENCHANTMENT OF RAVENS

localhost:5173/upproducts

[Contact us](#)

**APPLY NOW**

### My Orders

	ProductName:	Orderid:	Address:	Seller	BookingDate	Delivery By	Price	Status
	-d6ae	685d6ae877	925, gudivada,(333), ap.	karthik	26/6/2025	7/3/2025	\$20099	ontheway
	ProductName:	Orderid:	Address:	Seller	BookingDate	Delivery By	Price	Status
	-d6bd	685d6bd577	234, vij,(234567), ap.	srh	26/6/2025	7/3/2025	\$90099	ontheway

[Contact us](#)

## ORDER APPROVED

## CONCLUSION:

**BookNest** demonstrates a robust, scalable library/bookstore booking and management system using the MERN stack. It supports a real-world use case with multiple user roles and features like book reservation, overdue alerts, and inventory control. Security is enforced using JWT and RBAC. With future upgrades like multilingual support and mobile versions, BookNest can serve institutions globally. This project serves as both a **learning tool** and a **practical application**, showcasing how to design and develop a complete web-based system using the MERN stack.

## FUTURE SCOPE:

- Mobile apps for Android and iOS.
- AI-based book recommendations.
- QR code-based in-library borrowing.
- Support for library branches and inter-branch loans.
- Book clubs and reader communities.
- Offline sync and reading history analytics.

## APPENDIX:

☐ **Source Code:** <https://github.com/Karthik-Immaneni-7/booknest>

☐ **Project Demo Link:**

[https://drive.google.com/file/d/1apn4nEoCPD9EFaQpzlbCqIxseluSbr6M/view?usp=drive\\_link](https://drive.google.com/file/d/1apn4nEoCPD9EFaQpzlbCqIxseluSbr6M/view?usp=drive_link)