

Smart Bridge Internship

Full Stack Developer(Mern Stack)

Project Title:

DocSpot - Seamless Appointment Booking for Health

Team ID : LTVIP2025TMID20459

Team Members:

Team Leader : Modugu Vidya

Team member : Meer Apsarunnisa

Team member : Medepalli Sruthi

Team member : Merugumala Sobha Vinay Babu

Introduction

The healthcare industry is undergoing a digital transformation, with increasing emphasis on **enhancing accessibility, convenience, and efficiency** through technology. This shift has amplified the need for robust online systems that simplify healthcare services such as finding doctors, booking appointments, and managing patient records.

The **Book a Doctor App** is designed to meet this demand. It provides a **comprehensive, user-friendly web platform** that seamlessly connects patients with healthcare providers. By streamlining the appointment booking process and improving communication among patients, doctors, and administrators, the application reduces the friction typically associated with managing medical visits.

This project is built using the **MERN stack**—**MongoDB, Express.js, React.js, and Node.js**—demonstrating a practical and scalable approach to full-stack web development in the healthcare domain.

Project Overview

The **Book a Doctor App** is a comprehensive web-based solution that empowers users to **search, schedule, and manage medical appointments** online with ease. The platform is structured around three core user roles, each with dedicated functionality:

- **Patients** can register, log in, browse doctors by specialization or location, book appointments, upload relevant documents (e.g., prescriptions), and track their upcoming and past consultations.
- **Doctors** can apply to join the platform, manage their profiles and availability, respond to appointment requests, and update appointment statuses and medical records.
- **Administrators** oversee the platform by verifying doctor registrations, managing users, monitoring platform activity, and resolving any disputes or issues.

Key Features:

- Real-time booking with live doctor availability
- Secure authentication using **JWT (JSON Web Tokens)**
- **Role-based access control** to ensure appropriate permissions
- Custom dashboards tailored for patients, doctors, and administrators
- Notifications for appointment confirmations and status updates
- **Responsive React.js design** that delivers a smooth experience across devices

The application's architecture follows a modular and component-based structure, aligning functionality with user roles. The frontend, built with React.js, offers a dynamic single-page experience. On the backend, Node.js and Express.js handle server-side operations, while MongoDB serves as the flexible, document-oriented database.

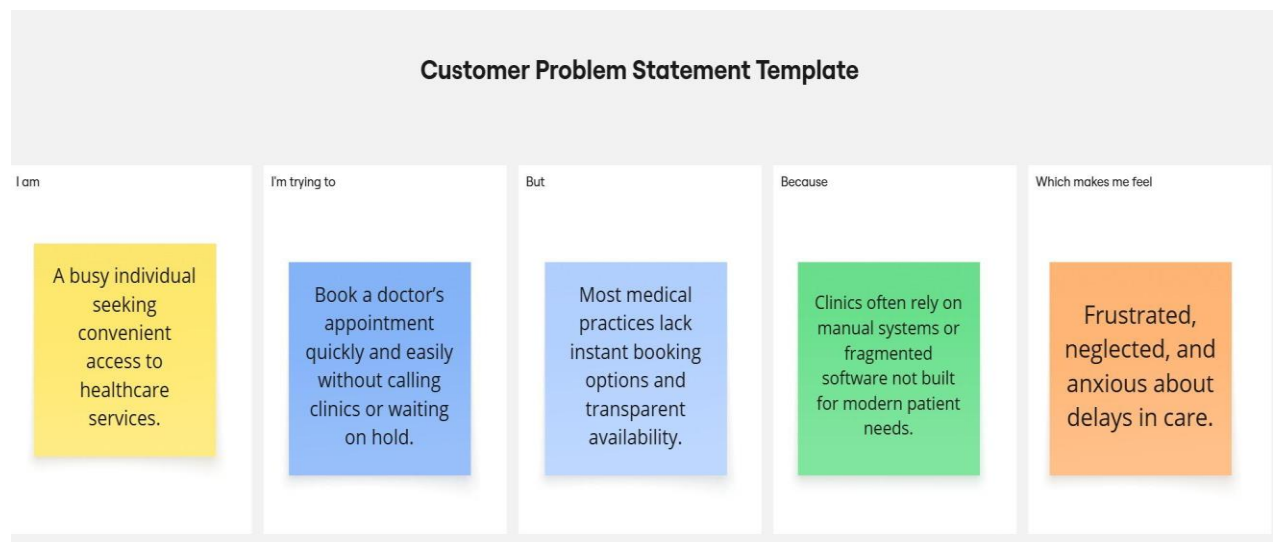
Purpose

The main **purpose** of this project is to **simplify and digitalize the process of booking medical appointments**, especially in a world where timely healthcare access is critical. Traditionally, patients need to call hospitals or clinics, wait in queues, or face scheduling conflicts. This application removes those obstacles by offering an efficient and secure alternative.

Additional goals include:

- ❑ **Improving patient-doctor communication:** The app provides tools for both parties to stay informed about appointment status, updates, and follow-ups.
- ❑ **Showcasing modern web development skills:** It demonstrates how the MERN stack can be used to build a fully functional, scalable, and maintainable full-stack application.
- ❑ **Enhancing healthcare management:** By storing and managing user data, documents, and booking history, the system supports better organization for both patients and providers.
- ❑ **Creating a secure and user-friendly experience:** Security features like JWT-based login and role-based access ensure data privacy, while the responsive UI improves accessibility.

Ideation Phase



Problem Statement:

1. Patients face difficulty booking timely and convenient medical appointments.
2. Existing systems often lack real-time availability, streamlined communication, and a user-friendly experience.

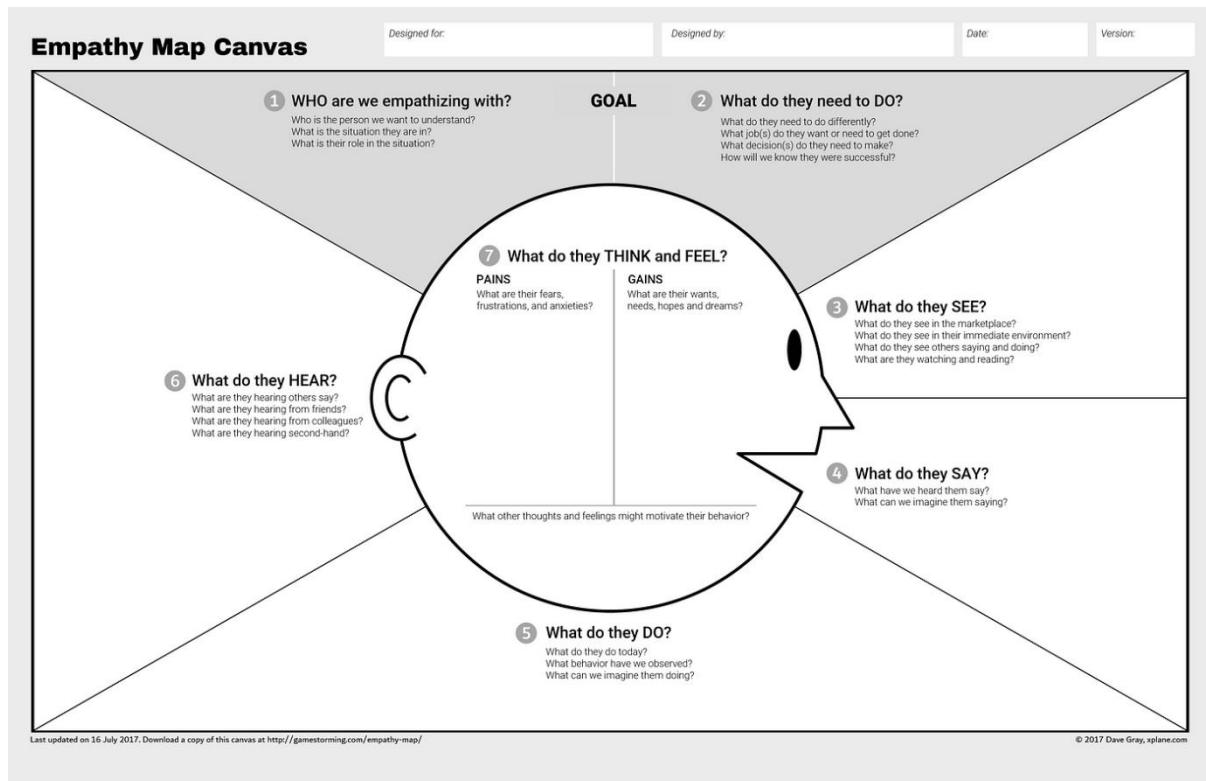
3. There's a disconnect between patients and verified healthcare professionals, leading to inefficiencies and frustration.

Proposed Solution:

1. Develop a **MERN stack web application** as a centralized healthcare appointment system.
2. Provide **patients** with an intuitive interface to:
 - Register and log in securely.
 - Search for doctors based on specialty, location, and availability.
 - Book, reschedule, or cancel appointments in real time.
3. Enable **doctors** to:
 - Manage their profiles and availability.
 - View and update their appointment calendar.
 - Communicate with patients if necessary.
4. Empower **administrators** to:
 - Oversee all user activity.
 - Manage platform content (specialties, locations, user verifications).
 - Ensure data integrity and system security.
5. Integrate **notification and reminder systems** (email/SMS) to reduce no-shows and enhance communication.
6. Ensure **responsive design** for accessibility across devices (desktop, tablet, mobile).
7. Implement **role-based authentication** and **data protection measures** to safeguard user information.

Empathy Map Canvas

This empathy map is based on the primary user: **the patient**.



Brainstorming & Idea Prioritization

Objective:

The objective of this phase was to explore, evaluate, and prioritize innovative ideas to create a seamless, user-friendly digital platform for booking healthcare appointments. Through collaborative ideation and strategic selection, the team focused on identifying features that would immediately resolve real-world scheduling issues while laying a scalable foundation for future healthcare tech enhancements.

Key Activities:

Step 1: Team Collaboration and Problem Statement Selection

- The team convened on 31 January 2025 to discuss gaps in healthcare scheduling systems.
- Through research and shared experiences, the team finalized the problem: *"Manual booking systems in healthcare cause delays, scheduling errors, and communication breakdowns for patients, doctors, and administrators."*

Step 2: Brainstorming, Idea Listing, and Grouping

- All team members contributed ideas openly without judgment.

- Suggestions were grouped into categories:

Patient-Centered Ideas:

- Register/login with role selection (patient, doctor)
- Browse/filter doctors based on specialty, time, and location
- Upload reports/prescriptions during appointment booking
- View appointment history and current status
- Receive confirmation and reminders via alerts/modals

Doctor-Centered Ideas:

- Dashboard showing upcoming appointments
- Ability to confirm/reschedule/cancel bookings
- Access to uploaded patient documents
- Add consultation notes and visit summaries

Admin-Centered Ideas:

- Vet and approve new doctor registrations
- View platform analytics and booking activity
- Manage user access and handle complaints

Stretch/Phase-2 Ideas:

- Real-time chat between patients and doctors
- Video consultation module
- Patient reviews and ratings
- AI-based slot recommendation

Step 3: Idea Prioritization

- The team used an impact–effort scale to select MVP features.
- Priority was given to high-impact, low-to-medium effort features:

Selected for MVP (Phase-1):

- Patient and doctor registration/login
- Doctor directory with filters
- Appointment booking with file uploads
- Admin approval of doctors
- Appointment status updates (Pending, Confirmed, Completed, Cancelled)
- Role-based dashboards for patient, doctor, and admin

Deferred for Future Phases:

- Real-time messaging
- Teleconsultation
- AI-enhanced scheduling
- Feedback/review system

Customer Journey Map

Purpose: To capture and visualize the complete user experience across all stages of interacting with the healthcare appointment system. This helps in identifying user needs, expectations, emotional responses, and areas for improvement.

Key Components:

- 1. User Actions:**
 - Registering and logging into the platform.
 - Searching for doctors based on specialty, location, or availability.
 - Booking, rescheduling, or canceling appointments.
 - Receiving notifications and reminders.
 - Providing feedback post-consultation.
- 2. User Thoughts and Emotions:**
 - *Before:* Anxious about delays and doctor availability.
 - *During:* Relief or confidence when finding a match quickly.
 - *After:* Satisfaction (or dissatisfaction) depending on ease of use and service quality.
- 3. Touchpoints:**
 - Website pages (home, search, doctor profiles, booking form).
 - Email/SMS notifications.
 - Chat support or FAQs.
 - Appointment calendar and dashboard.
- 4. Pain Points and Opportunities:**
 - Confusion during registration → Improve onboarding UX.
 - Long search times or irrelevant results → Optimize search filters.
 - Missed or forgotten appointments → Add real-time reminders.
 - Lack of post-visit feedback → Enable user reviews and satisfaction ratings.

Insight: Mapping this journey allows for **user-centered design decisions**, paving the way for a smoother experience, better engagement, and improved retention.

Solution Requirements

1. Functional Requirements:

These define what the system should do.

- **User Registration and Login:** Secure authentication with role-based access (patient, doctor, admin).
- **Doctor Search and Filter:** Search doctors by specialization, availability, location, and consultation mode.
- **Appointment Booking:** Real-time slot selection, with options to reschedule or cancel.
- **Doctor Dashboard:** View upcoming appointments, manage availability, edit profile.
- **Admin Panel:** Manage users, verify doctor credentials, monitor appointments, and control platform content.
- **Notification System:** Send confirmations, reminders, and updates via email/SMS.
- **Feedback Module:** Allow patients to rate and review doctors post-consultation.
- **Responsive UI:** Fully functional on desktop, tablet, and mobile devices.

2. Non-Functional Requirements:

These describe the system's operational attributes.

- **Performance:** Fast response times and minimal downtime.
- **Scalability:** Capable of handling growth in users, doctors, and appointment volume.
- **Security:** Data encryption, secure APIs, and protection against common vulnerabilities (e.g., SQL injection, XSS).
- **Usability:** Clean, intuitive interface for all user roles.
- **Maintainability:** Modular codebase for easier updates and debugging.

3. System Requirements:

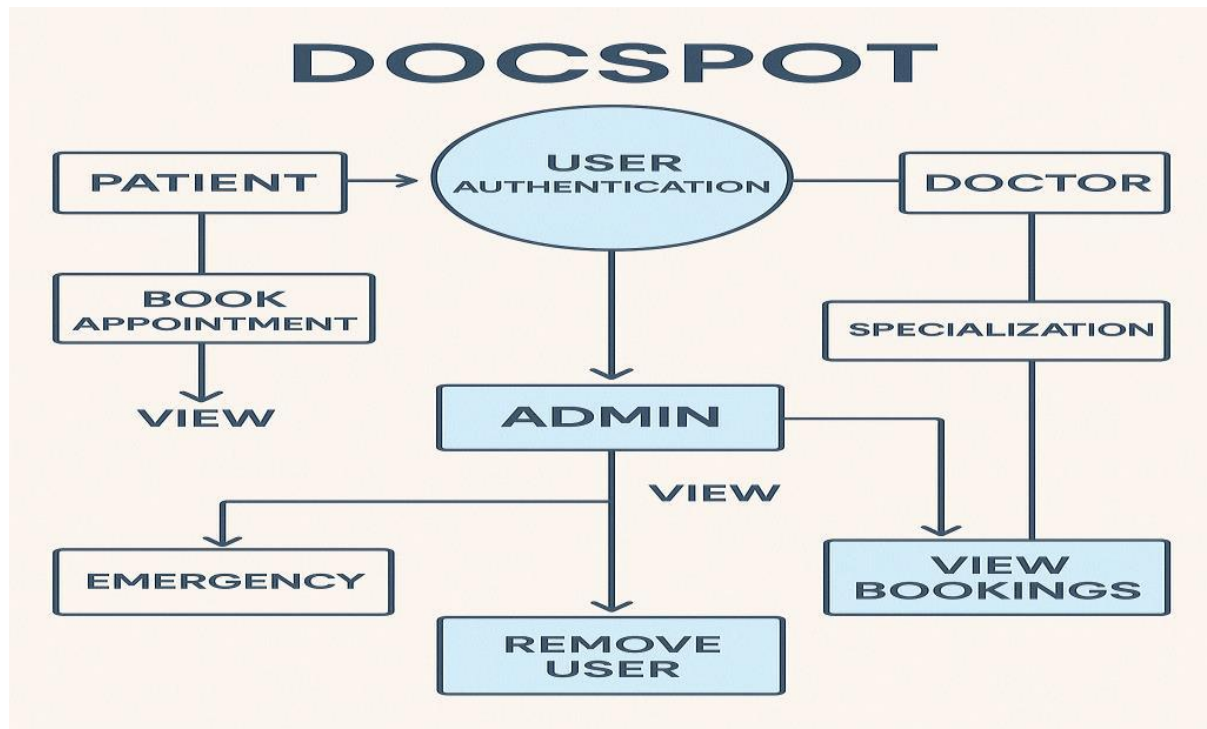
These outline the technical stack and environment setup.

- **Frontend:** React.js with Material UI or Bootstrap.
- **Backend:** Node.js with Express.js..
- **Database:** MongoDB with Mongoose for schema management.
- **Hosting:** Cloud deployment (e.g., Render, Vercel, or AWS).
- **Authentication:** JWT-based login with token expiration.

Data Flow Diagrams:

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.

- Register and log in securely so I can access my appointments.
- Search for doctors based on specialty, location, and availability to find the right match for my needs.
- Book an appointment by choosing a convenient time and uploading my medical reports.
- View my booking history and track the status of upcoming appointments.
- Reschedule or cancel appointments if my plans change.
- Receive reminders for my upcoming visits via SMS or email.
- Leave a review after my consultation to share my experience.



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

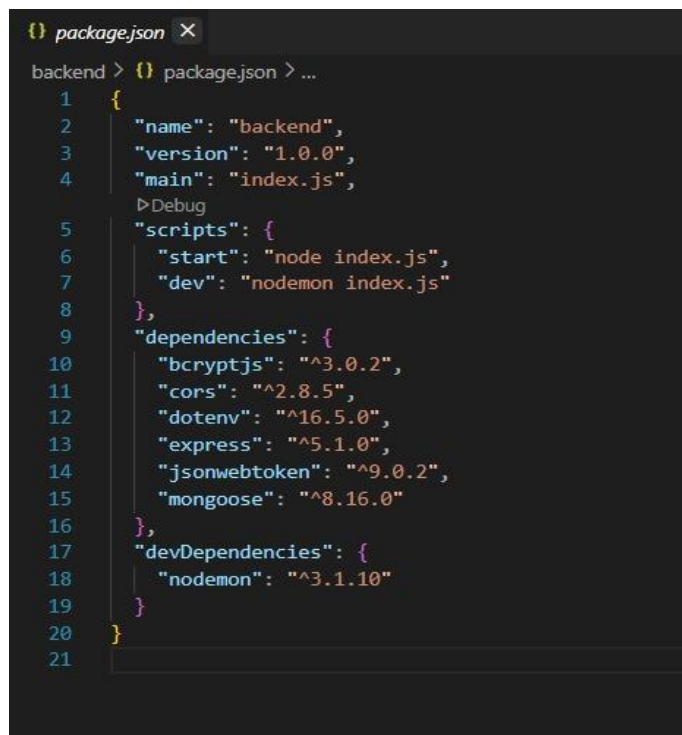
- Implements rate limiting middleware (e.g., express-rate-limit) to prevent brute force login attempts.
- Can be extended to include account lockout after repeated failed logins.

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    >Debug
6    "scripts": {
7      "start": "node index.js",
8      "dev": "nodemon index.js"
9    },
10   "dependencies": {
11     "bcryptjs": "^3.0.2",
12     "cors": "^2.8.5",
13     "dotenv": "^16.5.0",
14     "express": "^5.1.0",
15     "jsonwebtoken": "^9.0.2",
16     "mongoose": "^8.16.0"
17   },
18   "devDependencies": {
19     "nodemon": "^3.1.10"
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   "scripts": {
24     "start": "react-scripts start",
25     "build": "react-scripts build",
26     "test": "react-scripts test",
27     "eject": "react-scripts eject"
28   },
29   "eslintConfig": {
30     "extends": [
31       "react-app",
32       "react-app/jest"
33     ]
34   },
35   "browserslist": {
36     "production": [
37       ">0.2%",
38       "not dead",
39       "not op_mini all"
40     ],
41     "development": [
42       "last 1 chrome version",
43       "last 1 firefox version",
44       "last 1 safari version"
45     ]
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
```

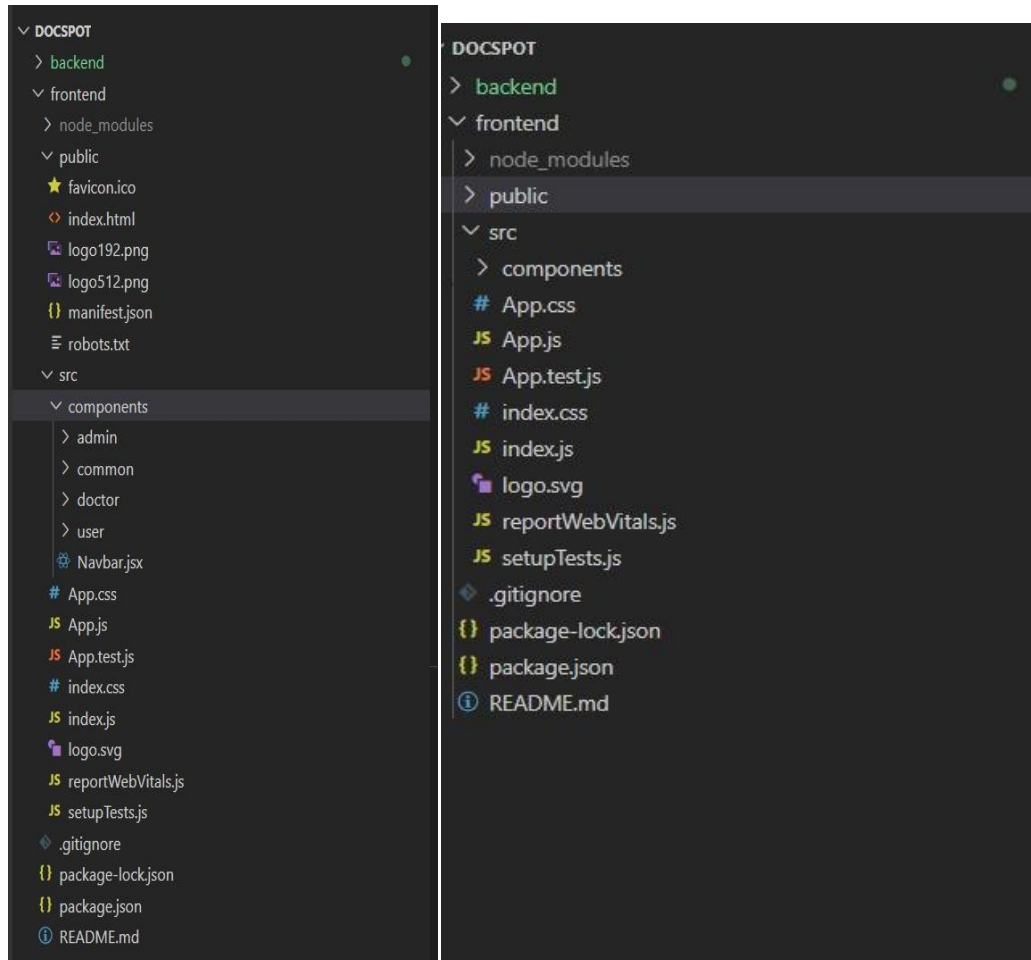
- The book a doctor app will be accessible at <http://localhost:3000>

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.

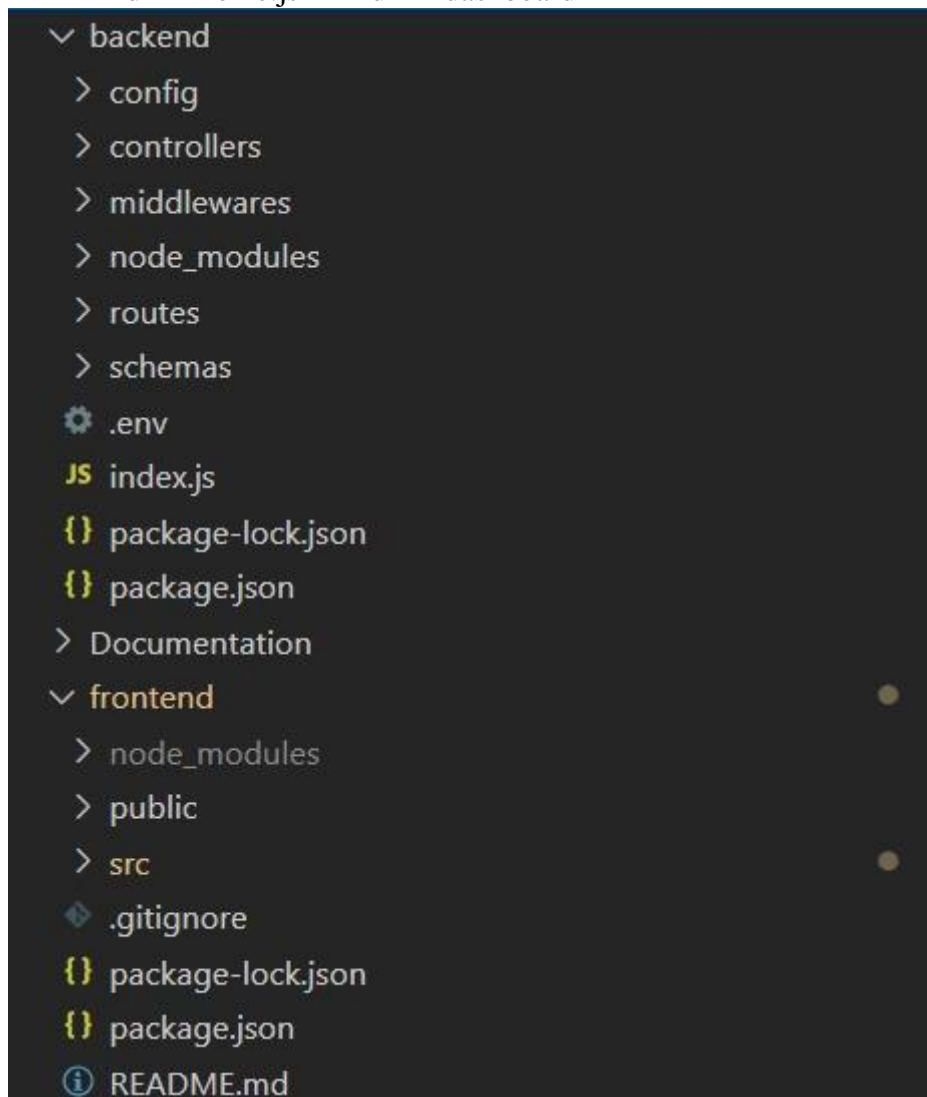
- **node_modules/** – Auto-generated folder with all installed packages.
- **public/** – Static files like index.html.
- **src/** – Main folder for all your code.

Inside src/components/:

admin/ – Admin pages:

- AdminAppointments.jsx – Manage appointments

- AdminDoctors.jsx – Manage doctors
- AdminHome.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/

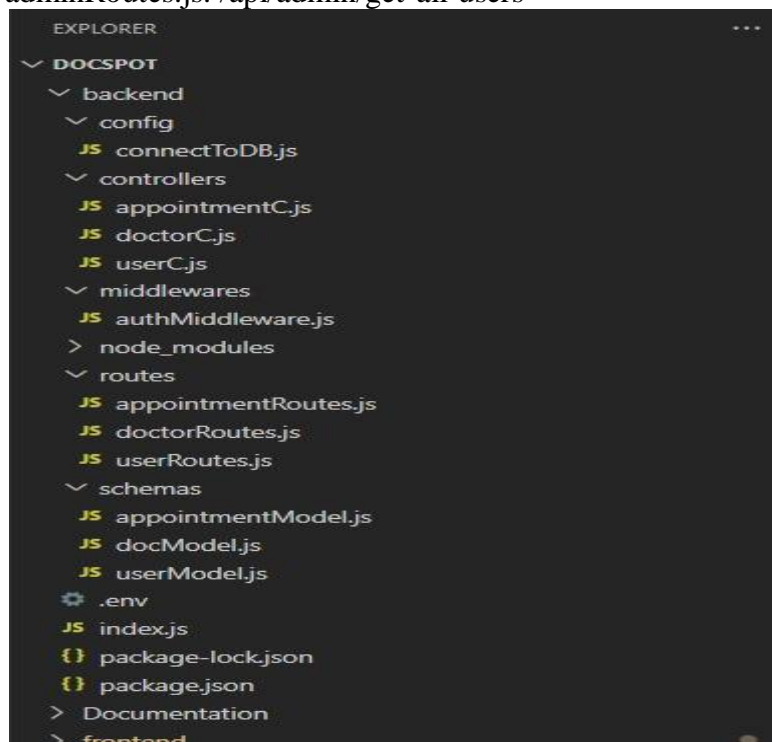
- Defines Express route endpoints.
- Imports relevant controllers and middleware.

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).

USER INTERFACE:

Here's an overview of the User Interface (UI) for the "Book a Doctor Using MERN" application, including the layout and key pages for each user role:

The application uses a modern, responsive design built with React.js and styled with a CSS framework (like Bootstrap, Tailwind CSS, or Material UI). The UI is intuitive and clean, offering distinct experiences for Patients, Doctors, and Admin

 User Roles & Dashboards Role	Features Shown in UI
Patient	Book doctor, view appointments, manage profile
Doctor	View/manage appointments, update schedule
Admin	Approve doctors, manage users, system analytics

Main UI Screens

1. Authentication Pages

- Login Email + Password fields
- Role selection (optional dropdown or auto-detected)
- Register Name, Email, Password, Confirm Password
- Role (Patient or Doctor)
-

2. Home Page (Public)

- Brief info about the service
- Login/Register CTA buttons
- Optional: Testimonials, features section

3. Patient Dashboard

- Sidebar navigation (Appointments, Profile, Logout)
- Book Appointment Page
- Dropdown or card view of doctors
- Date/time picker
- My Appointments Table or card list showing booked appointments with status

4. Doctor Dashboard

- Sidebar navigation (Appointments, Profile, Schedule)
- Manage Appointments View list of patient bookings
- Accept/Reject with status update
- Availability Settings Time slots configuration
-

5. Admin Panel

- Users Management List of registered users
- Doctor Applications Approve or reject pending doctor applications
- System Overview Stats: total users, active doctors, total appointments
-

6. Notifications

In-app or toast notifications for: Appointment status updates

Doctor approvals

Errors/success messages

TESTING:

- Goal of Testing
- To ensure that all components of the system — from backend APIs to frontend UI — behave as expected, are secure, and remain stable as features are added or changed.

- **Testing Strategy**

The project uses a layered testing strategy: Layer	Type of Testing	Description
Backend	Unit & Integration Tests	Test individual functions & API endpoints
Frontend	Component Tests	Test UI components independently
Full Stack	End-to-End (E2E) Tests	Simulate user interactions across the stack

Testing Tools Used

Backend (Node.js + Express)

Jest: JavaScript testing framework for unit and integration tests.

Supertest: For making HTTP assertions (API testing).

- MongoDB Memory Server: To run integration tests without affecting real database.

Example Tests:

- Test /auth/login with valid/invalid credentials.
- Ensure /appointments/book requires a valid JWT.

Frontend (React)

- React Testing Library: For testing React components by simulating real user behavior.
- Jest: Also used for frontend unit testing.
- Mock Service Worker (MSW) (optional): For mocking backend APIs in frontend tests.

Example Tests:

- Render Login form and simulate user input.
- Ensure DoctorCard displays doctor details correctly.

End-to-End (Optional)

Cypress or Playwright: For simulating user flows like: Register → Login → Book

Appointment → Logout

Admin approving a doctor

CI/CD Integration (Optional)

- Tests can be integrated into GitHub Actions or another CI pipeline to run on every push or pull request.

Testing Best Practices Followed

- Use test data and mocks to isolate tests.
- Keep tests independent and repeatable.
- Aim for high coverage on critical routes and logic (e.g., authentication, booking).

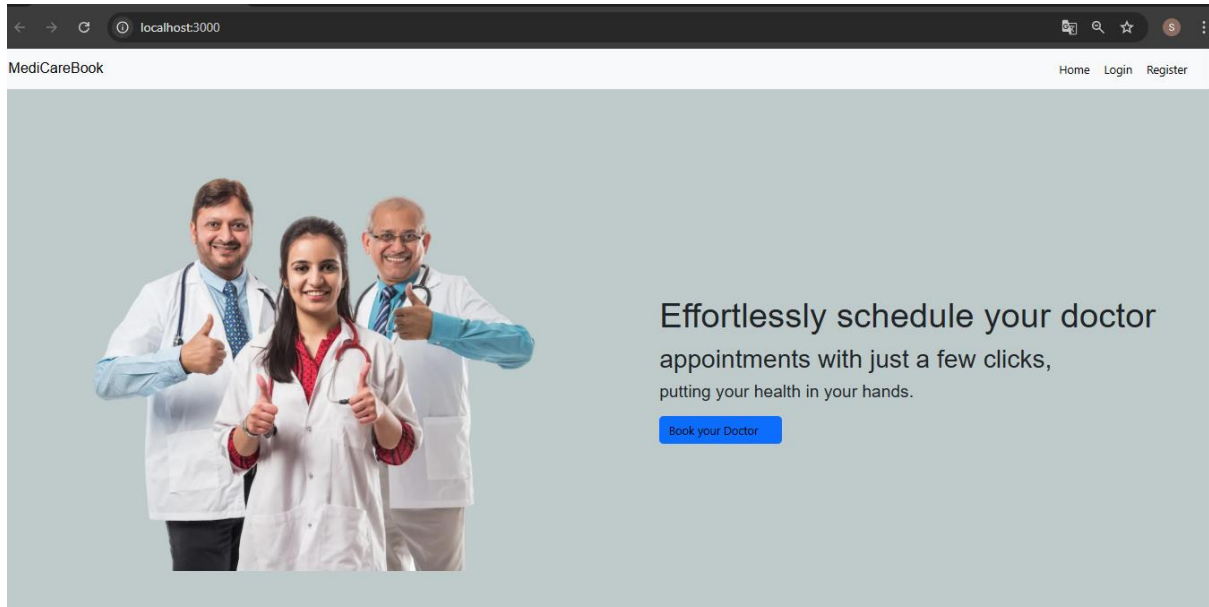
RESULTS:

Output Screenshots

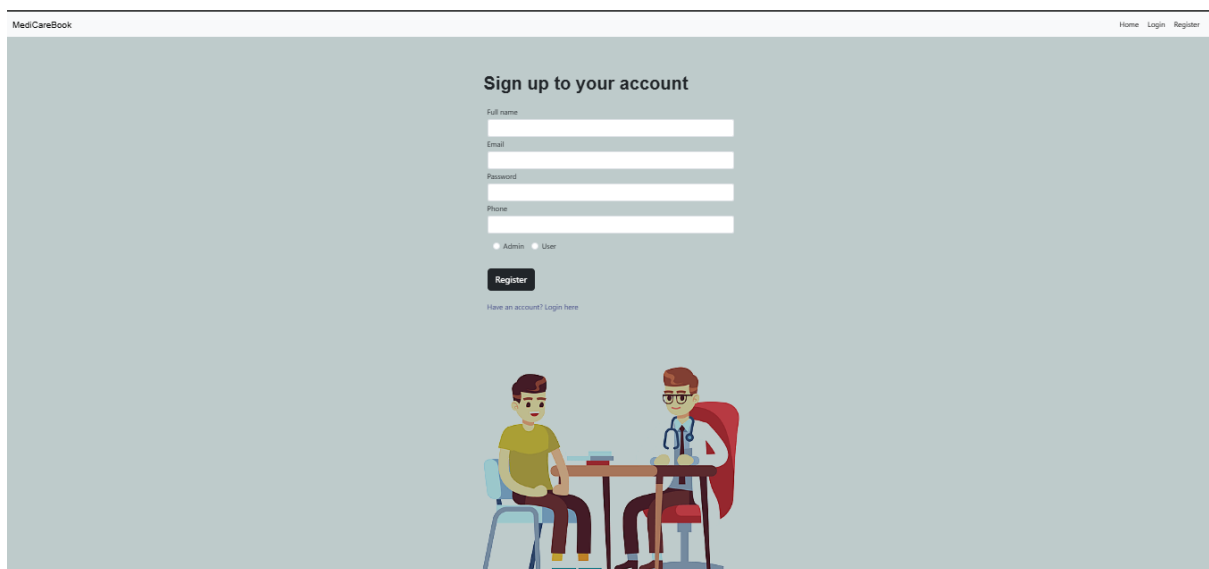
Links :

<https://drive.google.com/drive/folders/1rMk96q7NqaCxtTmlzXph26vli0J8q82l>

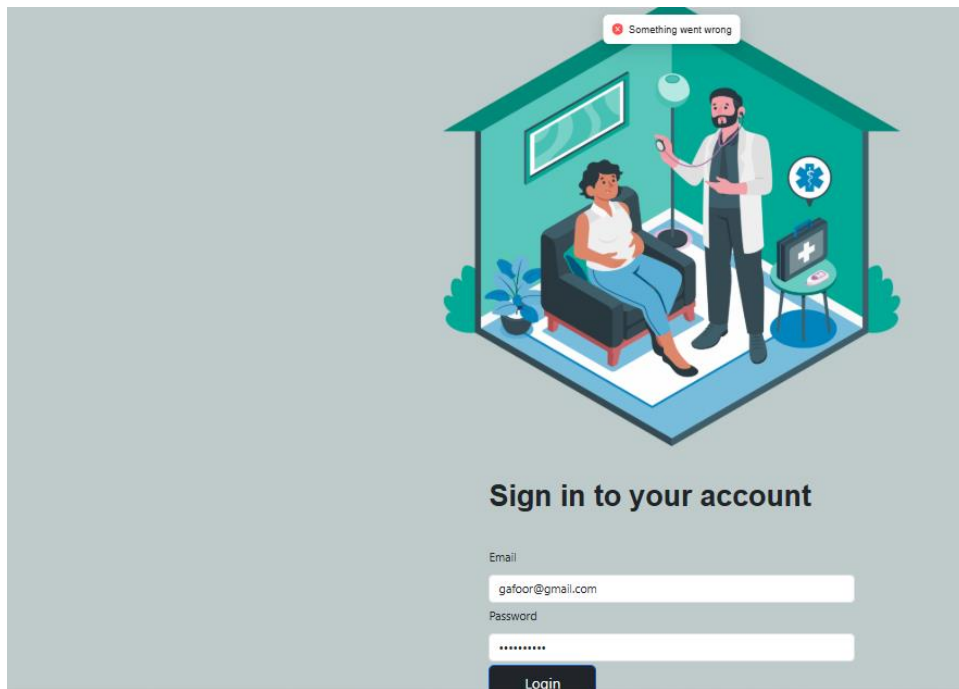
LANDING PAGE :



REGISTRATION:



LOGIN PAGE :



REGISTER FOR APPOINTMENT

Book Appointment with
Doctor

Your Name:

Your Age:

Select Slot:

Upload Documents:
 No file chosen

BOOK APPOINTMENT

Book Appointment with Doctor

Your Name:

Your Age:

Select Slot:

2025-06-29

▼

Upload Documents:

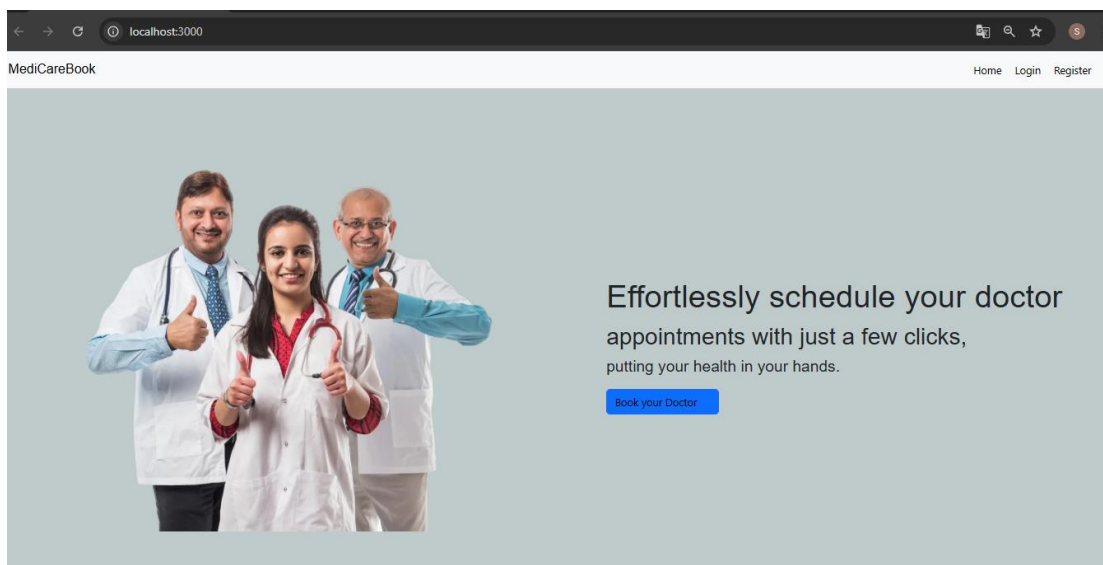
Choose files

No file chosen

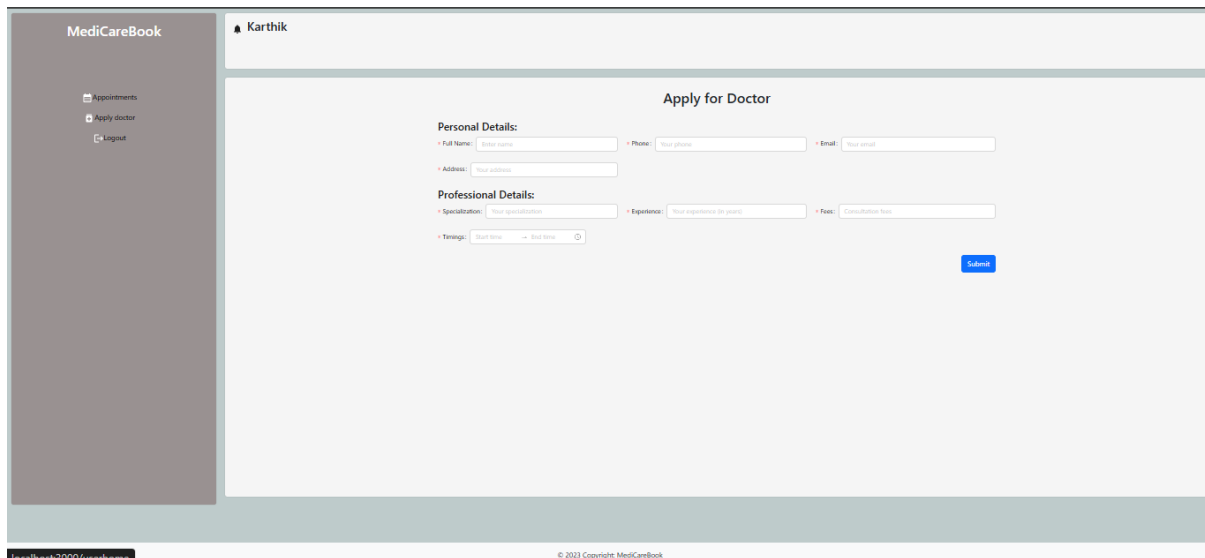
Book Appointment

☒ Appointment booked!

BACK TO DASHBOARD



APPLY NOW



CONCLUSION:

The "**Book a Doctor Using MERN**" application demonstrates a full-stack implementation of a real-world healthcare booking system. Built using the **MERN stack** (MongoDB, Express.js, React.js, and Node.js), it offers a modern, scalable solution for managing doctor appointments with distinct user roles including **patients**, **doctors**, and **administrators**. Through well-structured **APIs**, a responsive **user interface**, and robust **authentication and authorization**, the system allows users to register, book appointments, manage schedules, and administer the platform effectively. Security measures like **JWT-based authentication** and **role-based access control** ensure safe and reliable interactions.

While there are a few known limitations (e.g., time zone handling, pagination), the system provides a solid foundation that can be extended and enhanced with additional features such as notifications, analytics, and real-time chat between patients and doctors.

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:

Potential areas for future enhancement include:

- ☐ Integration of **video consultations** for telemedicine.
- ☐ Implementation of **AI-driven doctor recommendations** based on patient history.
- ☐ Advanced **analytics dashboard** for doctors and admins to track appointment trends.
- ☐ Mobile app versions for iOS and Android to widen accessibility.
- ☐ Integration with **electronic health records (EHR)** systems for automatic updates.
- ☐ Multilingual support to reach a broader user base.

APPENDIX:

- ☐ **Source Code:** <https://github.com/Vinaymerugumala/code>
- ☐ **Project Demo Link:** <https://drive.google.com/file/d/1CZZVFcpIw96s-zSTn20ECLh1h-ulC6zC/view?usp=sharing>