

# Project Report

## Doctor Appointment Booking System

<b>Team ID</b>	LTVIP2025TMID58986
<b>Team Leader</b>	R Varshitha
<b>Team Member 1</b>	Sai Keerthana E
<b>Team Member 2</b>	Sai Harshith
<b>Team Member 3</b>	S Baba Fakruddin

### INTRODUCTION:

In today's fast-paced world, timely and convenient access to healthcare services is crucial. The Book a Doctor App is a comprehensive and user-centric healthcare booking system developed to simplify and enhance the process of connecting patients with qualified medical professionals. This platform bridges the gap between healthcare providers and patients by offering a digital solution for appointment scheduling, medical document management, and doctor-patient interaction.

The application enables patients to search for doctors based on parameters such as specialization, location, and availability, thereby ensuring quick access to the right healthcare professional. Once a suitable doctor is identified, users can effortlessly book appointments, view their schedules, and receive real-time notifications and reminders, reducing delays and missed consultations.

Doctors benefit from a dedicated dashboard that allows them to efficiently manage appointments, access and update patient records, and communicate with patients through secure channels. In parallel, administrators have access to tools for overseeing system operations, ensuring compliance with policies, managing user roles, and handling disputes or complaints, ensuring the smooth functioning of the platform.

The system is built using a robust client-server architecture, with a React.js-based frontend designed using modern UI libraries like Bootstrap and Material UI to offer an intuitive and responsive user experience. On the backend, Node.js with Express.js ensures scalable and secure REST API management, while MongoDB is used for efficient NoSQL data storage and retrieval, especially suited for handling dynamic healthcare data.

Overall, the Book a Doctor App addresses the growing need for digital healthcare platforms by offering a secure, reliable, and user-friendly solution. It caters to all stakeholders involved—patients, doctors, and administrators—by enhancing accessibility, efficiency, and communication within the healthcare ecosystem.

### KEY FEATURES

## 1. Patient Registration & Profile Creation

- **Secure Sign-Up:** Email and password-based authentication for user security.
- **Profile Creation:** Stores personal and medical details securely for streamlined future bookings.

## 2. Doctor Browsing & Filtering

- **Advanced Search Filters:** Search doctors by specialty, location, and real-time availability.
- **Live Availability:** Ensures patients only see and select available time slots, reducing conflicts.

## 3. Appointment Booking & Management

- **Intuitive Booking Interface:** Enables patients to choose date/time and upload documents (e.g., prescriptions, medical records).
- **Automated Notifications:** Email/SMS confirmations and reminders to reduce no-shows.

## 4. Doctor's Dashboard

- **Schedule Management:** View and manage bookings, availability, and appointment statuses (e.g., scheduled, completed).
- **Patient Record Access:** Secure access to medical records with options to add visit notes and follow-up instructions.

## 5. Admin Controls & Approvals

- **Doctor Verification:** Admins review and approve doctor registrations, ensuring authenticity.
- **System Oversight:** Manage users, monitor platform health, resolve disputes, and enforce compliance.

## DESCRIPTION

The **Book a Doctor App** is a full-stack healthcare appointment platform designed to offer a **streamlined experience** for both patients and healthcare providers. The app bridges the gap between users and doctors by simplifying appointment scheduling and document sharing through a secure and modern digital interface.

Patients benefit from:

- **Secure account creation** and **profile management**
- Easy **doctor search** and **appointment booking**
- **Document uploads** for records and prescriptions
- **Automated reminders** to prevent missed appointments

Doctors receive:

- A **dashboard** to manage schedules and patient information
- Tools for **updating medical records**, prescriptions, and follow-up instructions

Admins ensure:

- **Doctor verifications**, system integrity, and dispute resolution
- Monitoring of all activities for **compliance** and **platform governance**

## Tech Stack

- **Frontend:** React.js with **Bootstrap** and **Material UI**
- **Backend:** Node.js and **Express.js**
- **Database:** MongoDB (NoSQL)
- **Other Tools:**
  - **Axios** for API communication
  - **Moment.js** for date/time handling
  - **bcrypt** for password hashing and security

The platform offers **a responsive and user-friendly interface**, secure data handling, and role-based access control—meeting the growing demand for **digitized, accessible healthcare solutions**.

## SCENARIO-BASED CASE STUDY

### 1. User Registration

*John*, seeking a routine health check-up, registers through the app using his email and password. Upon successful sign-up, he logs in and begins exploring the available services.

### 2. Browsing Doctors

John is presented with a list of verified doctors. He filters the list by choosing “Family Physician,” sets his location, and views available appointment slots.

### 3. Booking an Appointment

John selects *Dr. Smith* and proceeds to book an appointment. He chooses a time slot, uploads his recent medical reports, and submits the form. He immediately receives a confirmation message indicating the booking is in process.

### 4. Appointment Confirmation

Dr. Smith checks his dashboard, reviews John’s request, and confirms the appointment. John receives both an email and SMS with the finalized appointment details.

### 5. Appointment Management

As the appointment date approaches, John logs in to reschedule due to a conflict. The app allows him to cancel and rebook without hassle. He can also contact Dr. Smith directly via secure messaging.

### 6. Admin Approval (Background Process)

Meanwhile, the admin reviews Dr. Smith’s submitted credentials and documents during his initial registration. After verification, Dr. Smith’s profile is approved and made visible to patients.

### 7. Platform Governance

The admin monitors all platform activities, addresses any disputes or technical issues, ensures compliance with privacy laws, and implements updates as needed.

### 8. Doctor’s Appointment Management

Dr. Smith logs in to view his daily schedule. He prepares for each consultation, updates the appointment status post-visit, and records necessary medical details securely.

## 9. Appointment Consultation

John visits Dr. Smith's clinic at the appointed time. The doctor performs the check-up, provides medical advice, and recommends a follow-up if necessary.

## 10. Post-Appointment Follow-Up

After the visit, Dr. Smith updates John's medical records on the app. John receives a digital visit summary with treatment notes and prescriptions, accessible through his dashboard.

## TECHNICAL ARCHITECTURE

The **Book a Doctor App** is built on a **modern client-server architecture** designed to ensure scalability, performance, and security. The application leverages a combination of frontend and backend technologies, database management systems, and middleware tools to offer a seamless and efficient healthcare booking experience.

### Frontend (Client-Side)

- **Technologies Used:** React.js, Bootstrap, Material UI
- **Features:**
  - Provides a **responsive and intuitive UI** for patients, doctors, and administrators.
  - Uses **Axios** for efficient and asynchronous **API communication** with the backend.
  - Incorporates real-time UI feedback and dynamic form handling to enhance usability.

### Backend (Server-Side)

- **Technology Stack:** Node.js with Express.js
- **Responsibilities:**
  - Handles **business logic, API endpoints, and request/response cycles.**
  - Processes appointment bookings, doctor approvals, user roles, and document uploads.
  - Implements **Role-Based Access Control (RBAC)** for secured role-specific features and route protection.

### Database Layer

- **Database Used:** MongoDB (NoSQL)
- **Storage Functions:**
  - Efficiently stores and retrieves data for **user profiles, doctor records, appointments, and medical documents.**
  - Provides **scalability** and flexibility to support growing datasets without performance degradation.

### Authentication & Security

- **JWT (JSON Web Tokens):** Used for **secure session management** and token-based authentication across the app.
- **bcrypt:** Ensures **secure password hashing**, preventing plain-text storage and improving resistance to brute-force attacks.

- **RBAC (Role-Based Access Control):** Enforces access privileges based on user roles (Patient, Doctor, Admin), ensuring data security and privacy.

### Time & Scheduling Management

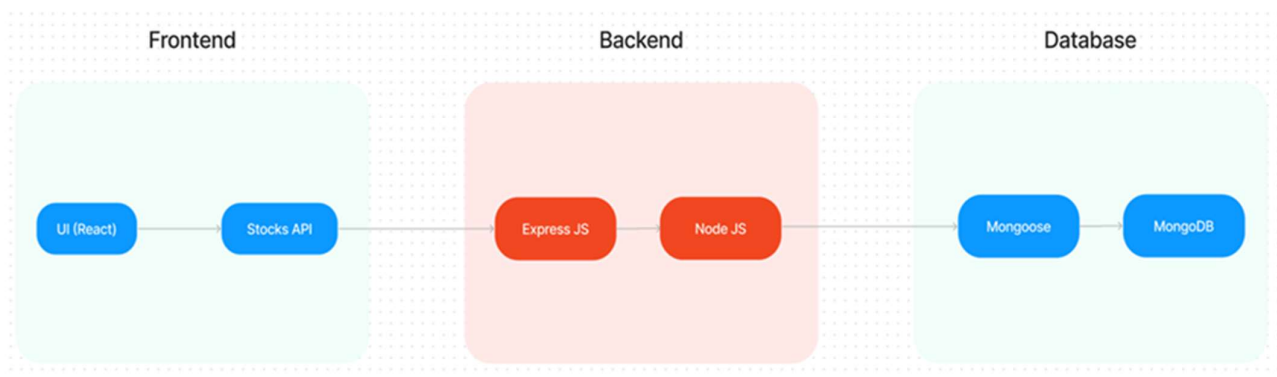
- **Moment.js:** Handles **date and time functionalities**, enabling precise and localized appointment scheduling, time validation, and formatting.

### Admin Dashboard Functions

- Approves and manages **doctor registrations**.
- Oversees overall **platform governance** and **policy compliance**.
- Resolves **disputes** and manages system configurations to ensure a consistent and secure experience for all users.

### Performance Optimization

- **Caching Techniques:** Applied to reduce database load and improve response times for frequently accessed resources.
- **Load Balancing (optional deployment level):** Can be implemented using tools like NGINX or a cloud-based service to distribute user requests across multiple servers and enhance performance during high traffic periods.



## TECHNOLOGY STACK OVERVIEW

The **Book a Doctor App** is developed using a modern full-stack web architecture that ensures **scalability, security, and responsiveness**. The system integrates powerful frontend frameworks with a robust backend, secure authentication, and performance-optimized technologies to deliver a seamless healthcare appointment experience.

### Frontend Technologies

- **Bootstrap & Material UI**

Provide a **responsive**, mobile-friendly, and visually modern user interface. They ensure consistent design elements and adaptive layouts across various devices.

- **Axios**

A **promise-based HTTP client** used to perform API calls between frontend and backend.  
Enables **asynchronous communication** for real-time data fetching and submission

## **Backend Framework**

- **Express.js (Node.js Framework)**

A **lightweight and scalable backend framework** responsible for handling:

- API endpoints
- HTTP requests and responses
- Routing and middleware functions

Ensures fast server-side execution and **clean code structure**.

## **Database and Authentication**

- **MongoDB**

A **NoSQL, document-oriented database** that stores:

- Patient profiles
- Doctor records
- Appointment details

It supports **scalable and high-performance** data operations with flexible schemas.

- **JWT (JSON Web Tokens)**

Used for **stateless user authentication**. Ensures that authenticated users remain logged in via token verification without relying on session storage.

- **bcrypt**

A cryptographic library for **password hashing**, protecting user credentials from exposure or unauthorized access in case of data breaches.

## **Admin Panel & Governance**

- **Admin Interface**

Enables platform administrators to:

- Approve or reject doctor registrations
- Monitor platform performance
- Manage users and system settings

- **Role-Based Access Control (RBAC)**

Ensures **data isolation and secure feature access** based on user roles (Patient, Doctor, Admin), enhancing **privacy and governance**.

## **Scalability and Performance**

- **MongoDB Horizontal Scaling**

Supports data sharding and cluster distribution to handle **growing user traffic** and **large datasets** efficiently.

- **Load Balancing**

Distributes incoming traffic across multiple server instances, **preventing overload** and improving **application uptime** during peak usage.

- **Caching Mechanisms**

Temporarily stores **frequently accessed data**, reducing repeated database queries and **enhancing response speed**.

## **Time Management and Scheduling**

- **Moment.js**

Handles:

- Date/time formatting
- Time zone conversions
- Appointment validation and precision scheduling

Ensures **accuracy and consistency** across all calendar-based operations.

## **Security Features**

- **HTTPS with SSL/TLS**

Ensures that all client-server communications are **encrypted**, preventing data interception or tampering.

- **Data Encryption**

All sensitive data—such as medical documents and personal details—is **encrypted at rest and in transit**, ensuring **regulatory compliance** (e.g., HIPAA/GDPR).

## **Notifications and Reminders**

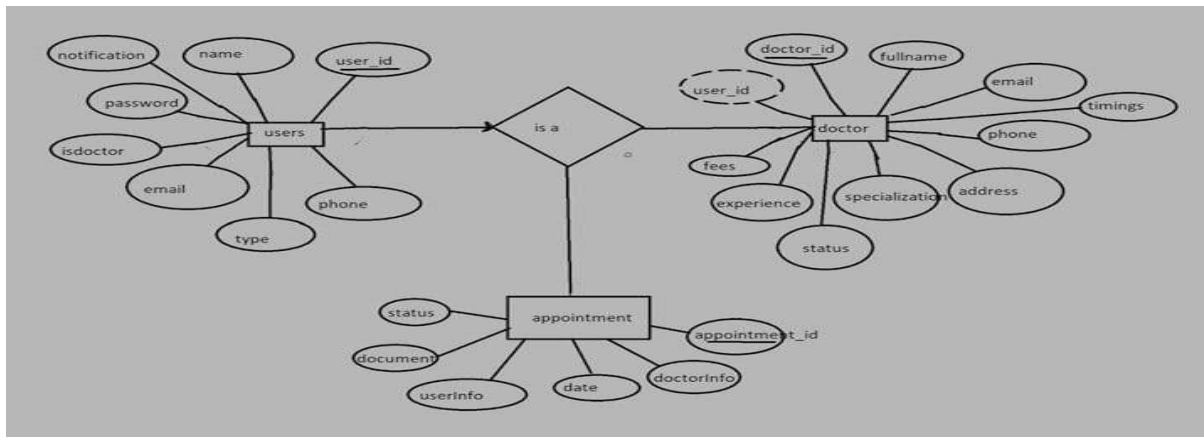
- **Email/SMS Integration**

Provides automated communication for:

- Appointment confirmations
- Reminders and alerts
- Cancellations and rescheduling notifications

Keeps both patients and doctors informed and reduces no-shows

## ER DIAGRAM :



## Entity-Relationship (ER) :

The **ER diagram** represents three main entities: **Users**, **Doctors**, and **Appointments**, along with their attributes and relationships.

1. The **Users** collection includes:
  - **\_id** (Primary Key)
  - name
  - email
  - notification
  - password
  - isDoctor (true if the user is a doctor; false otherwise)
  - type (admin/patient/doctor)
  - phone
2. The isDoctor field helps differentiate between:
  - Doctors (isDoctor: true)
  - Patients or Admins (isDoctor: false)
3. The **Doctors** collection contains doctor-specific information:
  - **\_id** (Primary Key)
  - userID (Foreign Key referencing Users.\_id)
  - fullName
  - email
  - timings
  - phone
  - address
  - specialisation
  - status (e.g., pending, approved)
  - experience
  - fees



4. The `userID` field in the `Doctors` collection links each doctor to their corresponding user account in the `Users` collection.
5. The **Appointments** collection includes details of appointments:
  - `_id` (Primary Key)
  - `doctorInfo` (Foreign Key referencing `Doctors._id`)
  - `userInfo` (Foreign Key referencing `Users._id`)
  - `date` (appointment date/time)
  - `document` (optional file such as medical records)
  - `status` (e.g., pending, confirmed, cancelled)
6. This collection manages the relationship between doctors and users (patients) for each appointment.
7. **Relationships among entities:**
  - One **User** can be linked to one **Doctor** (one-to-one)
  - One **User** (patient) can have **multiple Appointments** (one-to-many)
  - One **Doctor** can handle **multiple Appointments** (one-to-many)
8. **Foreign Key Mapping:**
  - `Doctors.userID` → `Users._id`
  - `Appointments.userInfo` → `Users._id`
  - `Appointments.doctorInfo` → `Doctors._id`
9. This ER structure enables the app to efficiently manage patient-doctor interactions, bookings, and role-based data access

## PRE REQUISITES :

### NODE.JS AND NPM:

- Node.js is a JavaScript runtime that allows you to run JavaScript code on the server-side. It provides a scalable platform for network applications.
- npm (Node Package Manager) is required to install libraries and manage dependencies.
- Download Node.js: [Node.js Download](#)
- Installation instructions: [Installation Guide](#)
- Run `npm init` to set up the project and create a `package.json` file.

### EXPRESS.JS:

- Express.js is a web application framework for Node.js that helps you build APIs and web applications with features like routing and middleware.
- Install Express.js to manage backend routing and API endpoints.
- Install Express:

- Run npm install express

## **MONGODB:**

- MongoDB is a NoSQL database that stores data in a JSON-like format, making it suitable for storing data like user profiles, doctor details, and appointments.
- Set up a MongoDB database for your application to store data.
- Download MongoDB: [MongoDB Download](#)
- Installation instructions: [MongoDB Installation Guide](#)

## **MOMENT.JS:**

- Moment.js is a JavaScript package for handling date and time operations, allowing easy manipulation and formatting.
- Install Moment.js for managing date-related tasks, such as appointment scheduling.
- Moment.js Website: [Moment.js Documentation](#)

## **REACT.JS:**

- React.js is a popular JavaScript library for building interactive and reusable user interfaces. It enables the development of dynamic web applications.
- Install React.js to build the frontend for your application.
- React.js Documentation: [Create a New React App](#)

## **ANTD (ANT DESIGN):**

- Ant Design is a UI library for React.js, providing a set of reusable components to create user-friendly and visually appealing interfaces.
- Install Ant Design for UI components such as forms, tables, and modals.
- Ant Design Documentation: [Ant Design React](#)

## **HTML, CSS, AND JAVASCRIPT:**

- Basic knowledge of HTML, CSS, and JavaScript is essential to structure, style, and add interactivity to the user interface.

## **DATABASE CONNECTIVITY (MONGOOSE):**

- Use Mongoose, an Object-Document Mapping (ODM) library, to connect your Node.js backend to MongoDB for managing CRUD operations.
- Learn Database Connectivity: Node.js + Mongoose + MongoDB

## **FRONT-END FRAMEWORKS AND LIBRARIES:**

- React.js will handle the client-side interface for managing doctor bookings, viewing appointment statuses, and providing an admin dashboard.
- You may use Material UI and Bootstrap to enhance the look and feel of the application.

## **SETUP AND INSTALLATION INSTRUCTIONS :**

### **CLONE THE PROJECT REPOSITORY:**

- Download the project files from GitHub or clone the repository using Git.

### **INSTALL DEPENDENCIES:**

- Navigate to the frontend and backend directories and install all required dependencies for both parts of the application.
- Frontend:
  - Navigate to the frontend directory and run npm install.
- Backend:
  - Navigate to the backend directory and run npm install.

### **START THE DEVELOPMENT SERVER:**

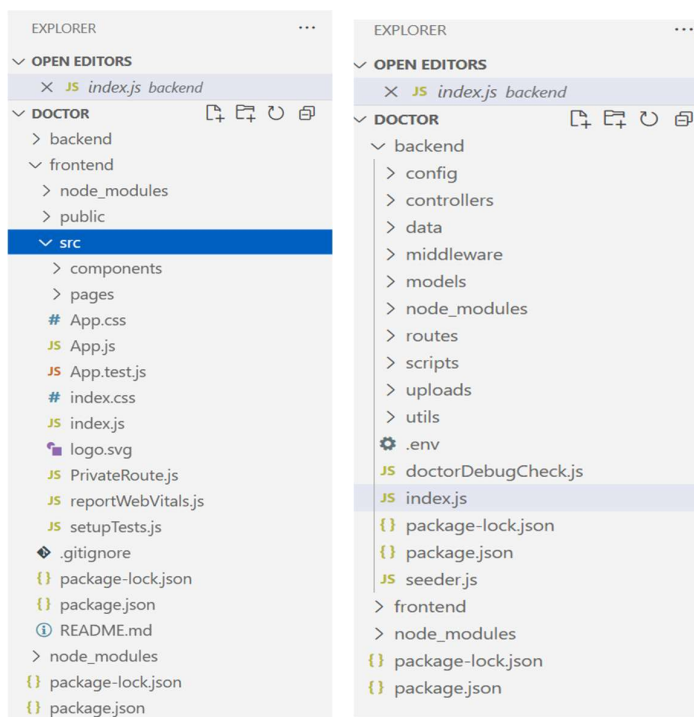
- After installing the dependencies, start the development server for both frontend and backend.
- Frontend will run on <http://localhost:3000>.
- Backend will run on <http://localhost:5000> or the specified port.

### **ACCESS THE APPLICATION:**

- Once both the frontend and backend servers are running successfully:
  - Frontend Interface: Open your browser and navigate to <http://localhost:3000> to access the user-facing Doctor Appointment Webpage.

- Backend API Services: Access the backend API endpoints via <http://localhost:5000> to interact with the server-side functionalities
- Ensure that MongoDB is running locally or connected via a cloud service (like MongoDB Atlas) before starting the backend server, as the application depends on the database for storing and retrieving user, doctor, and appointment information.
- Make sure to configure environment variables (such as MONGO\_URI, PORT, JWT\_SECRET, etc.) in a .env file in the backend directory to ensure smooth connection with the database and secure authentication processes

## PROJECT STRUCTURE :



The project is structured to include both Frontend and Backend components, each responsible for distinct tasks in the development of the Doctor Appointment Webpage. Below is the flow, describing the role and responsibilities of the users, the admin, and the doctor within the system.

## FRONTEND PART :

- The frontend is responsible for creating and rendering the user interface that customers, doctors, and admins interact with. It consists of the following files and folders:
- REACT COMPONENTS – Each component is designed for user interactions such as displaying doctors, booking appointments, and viewing notifications.

- ROUTING – Handles navigation between pages like customer dashboard, booking form, history, etc.
- STATE MANAGEMENT – Keeps track of the logged-in user's session, doctor details, and appointment statuses.
- STYLING – Uses CSS and UI libraries (e.g., Ant Design) to style the components.

## **BACKEND PART :**

The backend handles the server-side operations, including user authentication, data handling, and API responses. It contains the following files and folders:

- API ENDPOINTS – Defines the routes for handling customer, doctor, and admin functionalities, such as booking appointments, updating statuses, etc.
- DATABASE MODELS – Defines schemas for Users, Doctors, and Appointments using MongoDB and Mongoose.
- AUTHENTICATION & AUTHORIZATION – Manages login, registration, and access control for different user roles.
- NOTIFICATION SYSTEM – Sends notifications to users about appointment status updates.
- ADMIN FUNCTIONS – Admin-related routes for managing users, doctors, and appointment approvals.

## **APPLICATION FLOW:**

This project has three main types of users: Customer, Doctor, and Admin. Each has specific roles and responsibilities that are defined by the API endpoints.

### **CUSTOMER/ORDINARY USER:**

- CREATE AN ACCOUNT & LOGIN – Customers can register and log in using their email and password.
- VIEW DOCTORS – After logging in, customers will see a list of available doctors in their dashboard.
- BOOK APPOINTMENT – Customers can select a doctor and book an appointment by filling out a form with the appointment date and required documents.
- VIEW APPOINTMENT STATUS – Customers can track the status of their appointments (approved, pending, cancelled) and receive notifications when the appointment is scheduled.
- CANCEL BOOKING – Customers can cancel their appointments from their booking history page and change the status of the booking if needed.

## ADMIN:

- MONITOR ALL OPERATIONS – The admin oversees the platform, including the management of users, doctors, and appointments.
- APPROVE DOCTOR APPLICATIONS – Admins can review and approve doctor applications, making them available in the app.
- MANAGE POLICIES – Admin enforces platform policies, terms of service, and privacy regulations.
- USER MANAGEMENT – Admins can manage the profiles of customers and doctors, monitor their actions, and maintain a secure environment.

## DOCTOR:

- ACCOUNT APPROVAL – Doctors must receive approval from the admin before they can use the platform.
- MANAGE APPOINTMENTS – Once registered, doctors can manage the appointments they receive from customers, including confirming, rescheduling, or rejecting them.
- APPOINTMENT NOTIFICATIONS – Doctors are notified when new appointments are booked and when customers update their appointment details.

## API DOCUMENTATION:

Endpoint	Method	Description	Request Body/Params	Response
/api/user/register	POST	Register a new user (Patient/Doctor/Admin)	{ name, email, password, isDoctor }	201 Created, user details or error
/api/user/login	POST	Login user	{ email, password }	200 OK, JWT token or error
/api/doctor/apply	POST	Doctor applies for verification	{ userId, fullName, specialization, experience, fees, timings }	200 OK, message
/api/doctor/getAll	GET	Get list of all approved doctors	Auth Header (JWT Token)	200 OK, doctor list

/api/appointments/book	POST	Book an appointment	{ doctorId, userId, date, document }	201 Created, appointment ID
/api/appointments/user	GET	View appointments by a patient	Auth Header, user ID in token	200 OK, list of appointments
/api/appointments/doctor	GET	View appointments by a doctor	Auth Header, doctor ID in token	200 OK, appointment details
/api/appointments/update	PUT	Update appointment status (e.g., approve/cancel)	{ appointmentId, status }	200 OK, updated appointment
/api/admin/approveDoctor	PUT	Admin approves doctor application	{ doctorId }	200 OK, approval confirmation
/api/admin/users	GET	Admin gets list of all users	Auth Header (admin only)	200 OK, user data

## SETUP & CONFIGURATION :

Setting up the Doctor Appointment Webpage involves configuring both the Frontend (React.js) and Backend (Node.js, Express.js, MongoDB) to ensure the application runs smoothly. Below are the steps to set up and configure the environment for your project.

## FRONTEND CONFIGURATION :

### INSTALLATION :

#### Clone the Repository:

- Clone the project from GitHub to your local machine:
- bash
- Copy code
- git clone <your-repository-url>
- Replace <your-repository-url> with the URL of your project repository.

#### Navigate to Frontend Directory:

- After cloning, navigate to the frontend folder where the React.js app is located:
- bash
- Copy code
- cd book-a-doctor/frontend

## **Install Dependencies:**

- Use npm (Node Package Manager) to install the necessary dependencies:
- bash
- Copy code
- npm install
- This will install all the required libraries, such as React, React Router, Ant Design, and others.

## **Run the React Development Server:**

- To start the frontend server and run the React application:
- bash
- Copy code
- npm start
- The application will be available at <http://localhost:3000> in your browser.

## **BACKEND CONFIGURATION :**

### **INSTALLATION :**

#### **Navigate to Backend Directory:**

- Move to the backend folder of your project:
- bash
- Copy code
- cd book-a-doctor/backend
- Install Dependencies:
- Install the necessary backend dependencies using npm:
- bash
- Copy code
- npm install
- This will install all required libraries for Node.js and Express.js, such as express, mongoose, bcryptjs, jsonwebtoken, etc.

#### **Configure MongoDB :**

- Ensure you have MongoDB installed on your local machine or use a cloud-based MongoDB service like MongoDB Atlas.



- In the backend, open the configuration file (e.g., config/db.js or .env) and configure the connection URL for MongoDB:
- bash
- Copy code
- MONGO\_URI=mongodb://localhost:27017/doctor\_appointment
- If you are using MongoDB Atlas, replace the localhost part with your MongoDB Atlas connection string.

### **Set Up Environment Variables:**

- Create a .env file in the backend directory to store environment-specific variables such as:
- bash
- Copy code
- PORT=5000
- JWT\_SECRET=your\_jwt\_secret
- Make sure to replace your\_jwt\_secret with a strong secret key for JWT authentication.

### **Run the Backend Server:**

- Start the backend server by running:
- bash
- Copy code
- npm start
- The backend server will be running at <http://localhost:5000>.

### **DATABASE CONFIGURATION (MONGODB) :**

#### **Install MongoDB (Local Installation):**

- If you are using a local MongoDB instance, download and install it from the official MongoDB website: Download MongoDB

#### **Set Up MongoDB Database:**

- After installation, start the MongoDB service:
- bash
- Copy code
- mongod
- This will run MongoDB on the default port 27017.

## MongoDB Atlas (Cloud-based MongoDB):

- If you prefer to use MongoDB Atlas, create an account on MongoDB Atlas and create a cluster.
- Once the cluster is set up, get the connection string and replace it in the backend's .env file:
- bash
- Copy code
- MONGO\_URI=<your-mongodb-atlas-connection-string>

## FINAL CONFIGURATION & RUNNING THE APP

### Run Both Servers:

- The React frontend and Node.js backend servers should run simultaneously for the app to function correctly.
- You can open two terminal windows or use tools concurrently to run both servers together.
- To install concurrently, run:
- bash
- Copy code
- npm install concurrently --save-dev
- 

In your package.json file, add a script to run both servers:

json

```
"scripts": {  
  "start": "concurrently \"npm run server\" \"npm run client\"",  
  "server": "node backend/server.js",  
  "client": "npm start --prefix frontend"  
}
```

### Start Both Servers:

- Now you can run the following command in the root directory to start both the backend and frontend:
- bash
- Copy code
- npm start
- The backend will be available at <http://localhost:5000>, and the frontend at <http://localhost:3000>.

## **VERIFYING THE APP :**

### **Check Frontend:**

- Open your browser and go to `http://localhost:3000`. The React.js application should load with the list of doctors, booking forms, and status updates.

### **Check Backend:**

- Open Postman or any API testing tool to verify if the backend endpoints are working correctly, such as user login, doctor registration, and appointment creation.

## **ADDITIONAL SETUP :**

- Version Control:
- If you haven't already done so, initialise a Git repository in the root of your project:
- `bash`
- `git init`

### **Add your project files and commit them:**

- `bash`
- `git add .`
- `git commit -m "Initial commit"`
- Deployment (Optional):
- If you want to deploy your app, consider using cloud platforms like Heroku, AWS, or Vercel for frontend hosting and backend API deployment.

## **FOLDER SETUP :**

The folder structure for your Doctor Appointment Webpage project will include separate folders for the frontend and backend components to keep the code organised and modular. Here's how to set it up:

## **PROJECT ROOT STRUCTURE :**

### **Create the Main Folders:**

- In your project's root directory, create two main folders: frontend and backend.
- plaintext
- project-root/

```
├── frontend/  
└── backend/
```

## **BACKEND SETUP :**

- Install Necessary Packages in the Backend Folder:
- Navigate to the backend folder and install the following essential packages:
- plaintext
- backend/
  - ├── config/
  - ├── controllers/
  - ├── models/
  - ├── routes/
  - ├── middleware/
  - ├── uploads/
  - ├── server.js
  - └── .env

## **Packages to Install :**

- cors: To enable cross-origin requests.
- bcryptjs: For securely hashing user passwords.
- express: A lightweight framework to handle server-side routing and API management.
- dotenv: For loading environment variables.
- mongoose: To connect and interact with MongoDB.
- multer: To handle file uploads.
- nodemon: A utility to auto-restart the server upon code changes (for development).
- jsonwebtoken: To manage secure, stateless user authentication.
- Installation Commands

## **Run these commands in the backend folder:**

- 
- bash
- Copy code
- npm init -y
- npm install cors bcryptjs express dotenv mongoose multer jsonwebtoken
- npm install --save-dev nodemon

## FRONTEND SETUP :

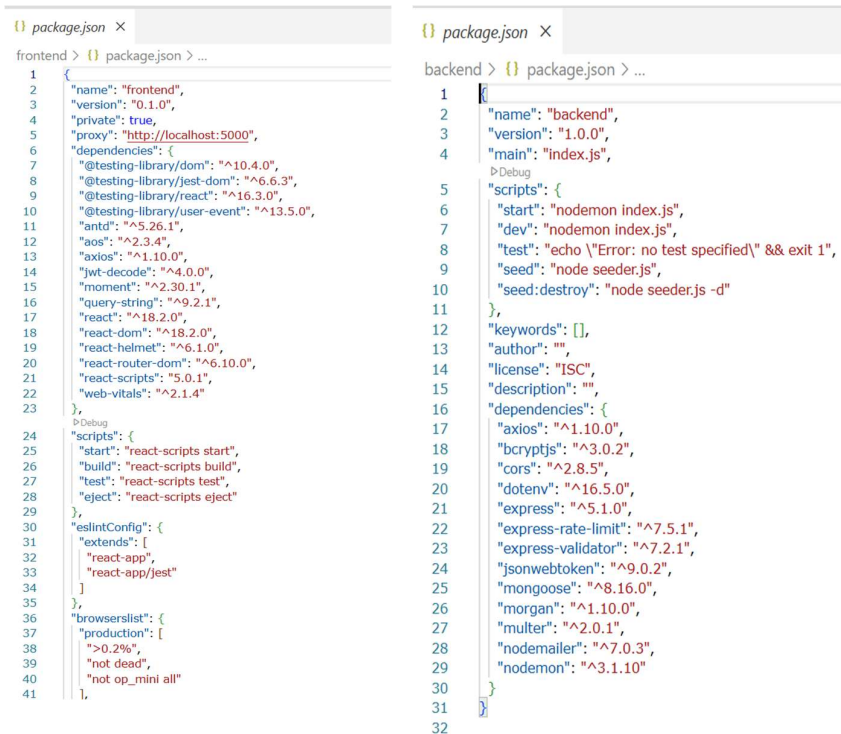
- React Project Initialization:
- 
- Navigate to the frontend folder and initialise a new React application:
- plaintext
- Copy code
- frontend/
  - ├─ public/
  - ├─ src/
    - ├─ components/
    - ├─ pages/
    - ├─ services/
    - └─ App.js
  - └─ .env
  - └─ package.json
- Setting Up the Frontend Project
- In the frontend folder, run the following commands to set up and install any initial dependencies:
- bash.

## MILESTONE 1: PROJECT SETUP AND CONFIGURATION :

### Project Structure and Environment Setup

- Setting up a structured environment is crucial for building a scalable and maintainable application.
- By organizing the project into separate folders for frontend and backend, the application becomes easier to manage, debug, and deploy.
- This modular approach ensures:
  - Clear separation of concerns between client-side and server-side code.
  - Easier collaboration in teams working on different layers.
  - Simplified dependency management and environment configuration.
- **This modular architecture offers several benefits:**
- Clear separation of concerns between UI (frontend) and business logic/data handling (backend).
- Simplified debugging and testing, as issues can be isolated within their respective layers.
- Improved team collaboration, with frontend and backend developers working independently without overlap.

- Easier dependency management, allowing each part to maintain its own node\_modules and configuration files.



The image shows two side-by-side screenshots of code editors displaying package.json files. The left editor shows the 'frontend' package.json, which includes dependencies for testing libraries, axios, jwt-decode, moment, query-string, react, react-dom, react-helmet, react-router-dom, react-scripts, and web-vitals. It also includes scripts for start, build, test, and eject, an ESLint configuration, and a browserslist. The right editor shows the 'backend' package.json, which includes dependencies for axios, bcryptjs, cors, dotenv, express, express-rate-limit, express-validator, jsonwebtoken, mongoose, morgan, multer, nodemailer, and nodemon. It includes a script for start and keywords, author, license, and description.

```

frontend > {} package.json > ...
1 {
2   "name": "frontend",
3   "version": "0.1.0",
4   "private": true,
5   "proxy": "http://localhost:5000",
6   "dependencies": {
7     "@testing-library/dom": "^10.4.0",
8     "@testing-library/jest-dom": "^6.6.3",
9     "@testing-library/react": "^16.3.0",
10    "@testing-library/user-event": "^13.5.0",
11    "antd": "^5.26.1",
12    "aos": "^2.3.4",
13    "axios": "^1.10.0",
14    "jwt-decode": "^4.0.0",
15    "moment": "^2.30.1",
16    "query-string": "^9.2.1",
17    "react": "^18.2.0",
18    "react-dom": "^18.2.0",
19    "react-helmet": "^6.1.0",
20    "react-router-dom": "^6.10.0",
21    "react-scripts": "5.0.1",
22    "web-vitals": "^2.1.4"
23  },
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  },
43  "devDependencies": {}
44}

backend > {} package.json > ...
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "start": "nodemon index.js",
7     "dev": "nodemon index.js",
8     "test": "echo \"Error: no test specified\" && exit 1",
9     "seed": "node seeder.js",
10    "seed:destroy": "node seeder.js -d"
11  },
12  "keywords": [],
13  "author": "",
14  "license": "ISC",
15  "description": "",
16  "dependencies": {
17    "axios": "^1.10.0",
18    "bcryptjs": "^3.0.2",
19    "cors": "^2.8.5",
20    "dotenv": "^16.5.0",
21    "express": "^5.1.0",
22    "express-rate-limit": "^7.5.1",
23    "express-validator": "^7.2.1",
24    "jsonwebtoken": "^9.0.2",
25    "mongoose": "^8.16.0",
26    "morgan": "^1.10.0",
27    "multer": "^2.0.1",
28    "nodemailer": "^7.0.3",
29    "nodemon": "^3.1.10"
30  },
31  "devDependencies": {}
32}

```

## PROJECT FOLDERS:

- Frontend Folder: Contains all code related to the user interface, written in JavaScript using frameworks and libraries like React, Material UI, and Bootstrap. This setup helps maintain a clear boundary between UI logic and server logic.
- Backend Folder: Manages the server, API routes, and database interactions, typically handled through Node.js and Express.js. Using separate folders enables a modular structure, allowing changes in one area without affecting the other.

## LIBRARY AND TOOL INSTALLATION:

### Backend Libraries:

- Node.js: Provides a runtime environment to run JavaScript code on the server side.
- MongoDB: A NoSQL database, perfect for flexible and schema-less data storage, ideal for applications needing frequent updates and various data types.
- Bcrypt: Encrypts passwords for secure authentication, helping protect user data from potential breaches.
- Body-parser: Parses incoming request bodies, making it easy to access data in various formats like JSON.
- Frontend Libraries:

- **React.js:** Manages component-based UI creation, providing the flexibility to build reusable UI components.
- **Material UI & Bootstrap:** Provides styling frameworks, ensuring a consistent, responsive, and visually appealing design.
- **Axios:** Facilitates easy HTTP requests, allowing the frontend to communicate with the backend effectively.

## **MILESTONE 2: BACKEND DEVELOPMENT :**

The backend forms the core logic and data management for the project. A well-structured backend ensures efficient data handling, security, and scalability.

### **EXPRESS.JS SERVER SETUP:**

- **Express Server:** Acts as a hub for all requests and responses, routing them to appropriate endpoints. It's Essential for managing incoming requests from the frontend, processing them, and sending responses back.
- **Middleware Configuration:** Middleware like `body-parser` parses JSON data in requests, while `cors` enables cross-origin communication between the frontend and backend. Middleware makes it easy to add additional functionality, like error handling or data validation, without interfering with core application logic.

### **API ROUTE DEFINITION:**

- **Route Organization:** Organizing routes by functionality (e.g., authentication, appointments, complaints) keeps the codebase readable and easy to maintain. For instance, all authentication-related routes can reside in an `auth.js` file, ensuring each file has a single purpose.
- **Express Route Handlers:** Route handlers manage the flow of data between client and server, such as fetching, creating, updating, and deleting records.

### **DATA MODELS (SCHEMAS) WITH MONGOOSE:**

- **User Schema:** Defines structure for user data and includes fields for personal information and role-based access (e.g., doctor, customer, admin). Using a schema ensures consistent data storage.
- **Appointment and Complaint Models:** Models like Appointment and Complaint manage complex data interactions, including relationships between users and appointments, allowing efficient querying.

- **CRUD Operations:** CRUD functionalities (Create, Read, Update, Delete) provide a standard interface for handling data operations. They simplify data manipulation in a structured, predictable way.

### **USER AUTHENTICATION:**

- **JWT Authentication:** JSON Web Tokens securely handle session management, ensuring that only verified users access protected routes. JWT tokens are embedded in request headers, verifying each request without storing session data on the server.

### **ADMIN AND TRANSACTION HANDLING:**

- **Admin Privileges:** Administrators oversee user registrations, approve appointments, and manage doctor applications. Admin-specific routes ensure that these actions are isolated and secure.
- **Transaction Management:** This functionality allows customers to interact with appointments, booking history, and cancellations in real-time.

### **ERROR HANDLING:**

- **Middleware for Error Handling:** Error-handling middleware catches issues and sends meaningful error responses with HTTP status codes (like 404 for Not Found or 500 for Server Error), enabling better debugging and user feedback.

- 

### **MILESTONE 3: DATABASE DEVELOPMENT**

- Using MongoDB as the database provides a flexible, schema-less structure, perfect for handling different types of user and appointment data.

### **SCHEMAS FOR DATABASE COLLECTIONS:**

- **User Schema:** Defines fields for user information like name, email, password, and userType. This schema allows fine-grained control over user data and easy retrieval of information.
- **Complaint and Assigned Complaint Schemas:** These schemas manage complaint data, with fields linking complaints to users and statuses. They allow efficient tracking of complaints and status updates by linking agents to users.
- **Chat Window Schema:** This schema organises messages between users and agents, storing them by complaint ID for a streamlined user-agent communication flow.



## **DATABASE COLLECTIONS IN MONGODB:**

- 
- MongoDB collections, such as users, complaints, and messages, provide a structured, NoSQL approach to data management, making it easy to scale as data grows.
- MILESTONE 4: FRONTEND DEVELOPMENT
- Frontend development focuses on creating an interactive, intuitive user experience through a React-based user interface.

## **REACT APPLICATION SETUP:**

- 
- Folder Structure and Libraries: Setting up the initial React app structure and libraries ensures a smooth development workflow. By organising files into components, services, and pages, the project becomes easy to navigate and maintain.
- UI Component Libraries: Material UI and Bootstrap offer pre-built components, enabling rapid UI development and consistent design across all screens.

## **UI COMPONENTS FOR REUSABILITY:**

- Reusable Components: Each UI element, like forms, dashboards, and buttons, is designed as a reusable component. This modularity allows efficient reuse across the app, reducing development time and ensuring consistency.
- Styling and Layout: Styling and layout components maintain a cohesive look and feel, contributing to the user experience with clean, intuitive visuals.

## **FRONTEND LOGIC IMPLEMENTATION:**

- API Integration: Axios is used to make API calls to the backend, connecting UI components with data from the server.
- Data Binding and State Management: React's state management binds data to the UI, automatically updating it as the user interacts with the app.

## **MILESTONE 5: PROJECT IMPLEMENTATION AND TESTING:**

- After completing development, running a final set of tests is crucial for identifying any bugs or issues.

## **VERIFY FUNCTIONALITY:**

Running the entire application ensures that each part (frontend, backend, database) works cohesively.

Testing various user flows (e.g., booking, cancelling, updating appointments) helps confirm that all processes are functioning as intended.

### **USER INTERFACE ELEMENTS:**

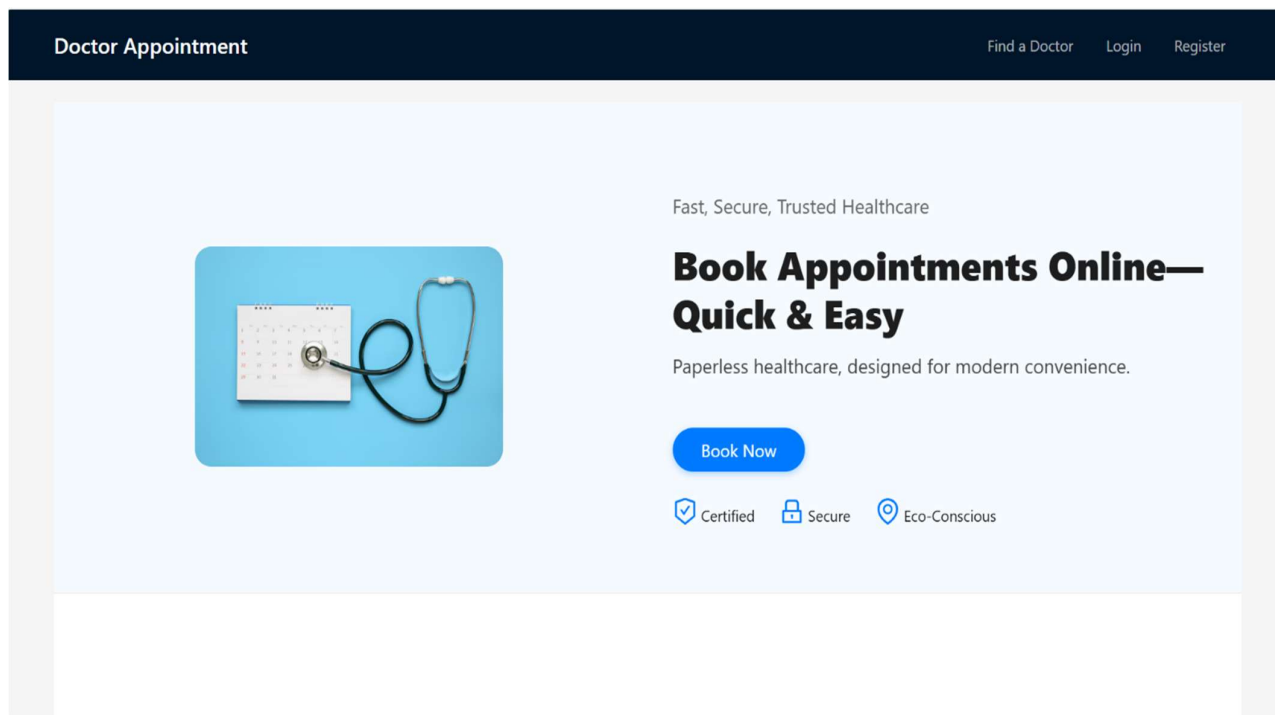
Testing the UI includes verifying the look and feel of each page—landing, login, registration, and dashboards for different user types.

Ensuring responsive design and usability across devices and screen sizes is also essential.

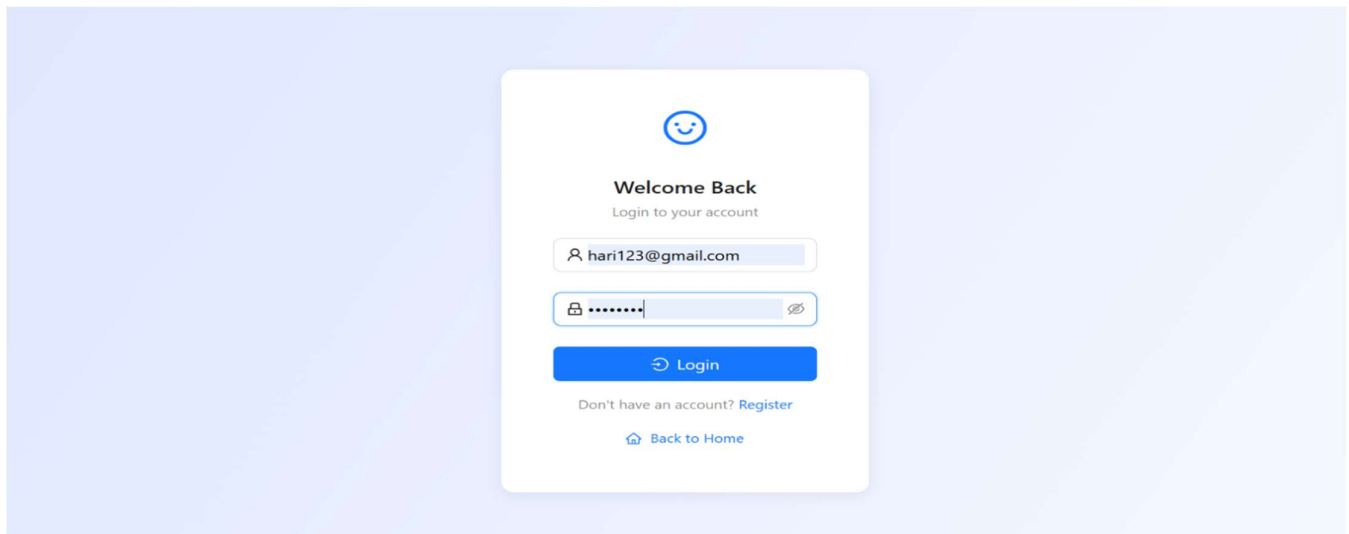
### **FINAL DEPLOYMENT:**

Once testing is complete, the application can be deployed to a production server, making it accessible to end-users.

### **LANDING PAGE :**



## LOGIN PAGE :



A login page with a light blue background. In the center is a white card with a blue smiley face icon at the top. Below the icon, the text "Welcome Back" is displayed in bold, followed by "Login to your account" in a smaller font. There are two input fields: the first contains the email "hari123@gmail.com" and has a magnifying glass icon on the left; the second contains a masked password "\*\*\*\*\*" and has a lock icon on the left and an eye icon on the right. Below these fields is a blue "Login" button with a circular arrow icon. At the bottom of the card, there is a link "Don't have an account? Register" and a link "Back to Home" with a house icon.

Welcome Back  
Login to your account

hari123@gmail.com

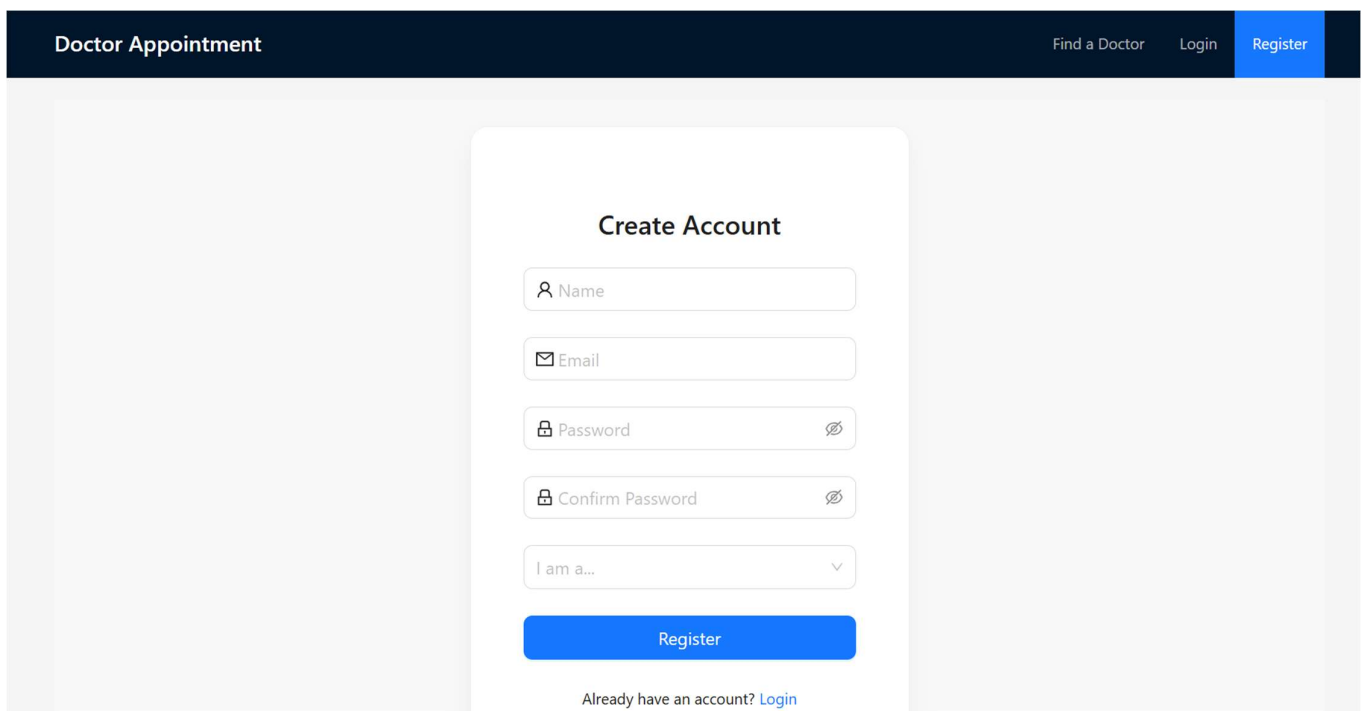
\*\*\*\*\*

Login

Don't have an account? [Register](#)

[Back to Home](#)

## REGISTRATION PAGE :



A registration page with a dark blue header. The header contains the text "Doctor Appointment" on the left and "Find a Doctor", "Login", and "Register" on the right, with "Register" highlighted in blue. The main content area is light gray. In the center is a white card with the title "Create Account". Below the title are five input fields: "Name" (with a person icon), "Email" (with an envelope icon), "Password" (with a lock icon and an eye icon), "Confirm Password" (with a lock icon and an eye icon), and "I am a..." (with a dropdown arrow). Below these fields is a blue "Register" button. At the bottom of the card, there is a link "Already have an account? Login".

Doctor Appointment

Find a Doctor Login Register

Create Account

Name

Email

Password

Confirm Password

I am a...


Register

Already have an account? [Login](#)

## USER PAGE :

Doctor Appointment

Find a Doctor

 2

My Dashboard


Logout

# Find Your Perfect Doctor

Search from our network of qualified professionals

Filter by specialty


Rating

 Search by Experience

## ADMIN PAGE :

Doctor Appointment

Find a Doctor





My Dashboard


Logout


# Admin Dashboard

Manage doctors, users, and appointments with ease


Total Users  
 34

Doctors  
 28

Pending Approvals  
 6



Appointments  
 11

## APPLY AS DOCTOR :



### Welcome Back

Login to your account

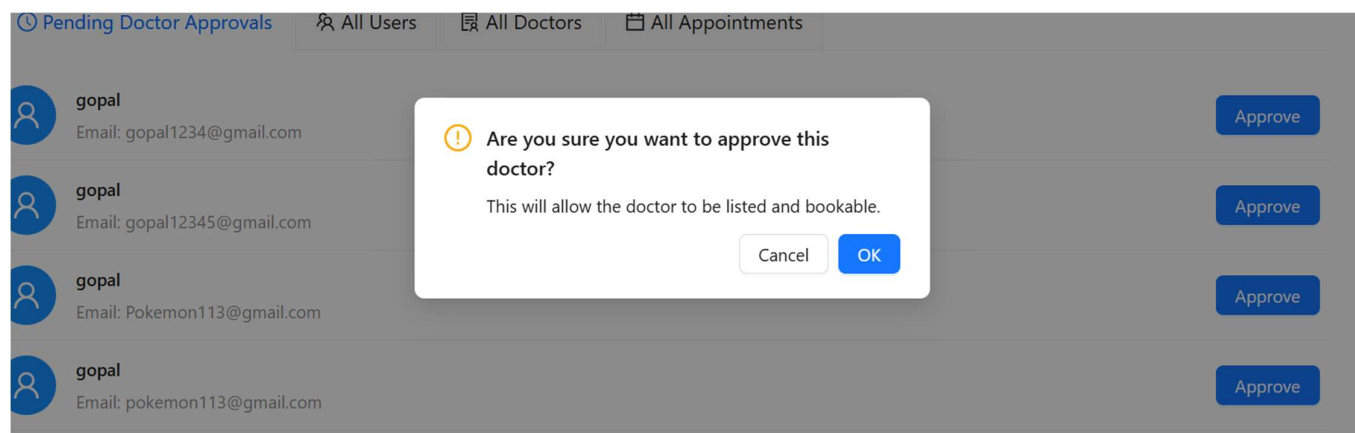
 Pending Approval 

Your account is awaiting admin approval.

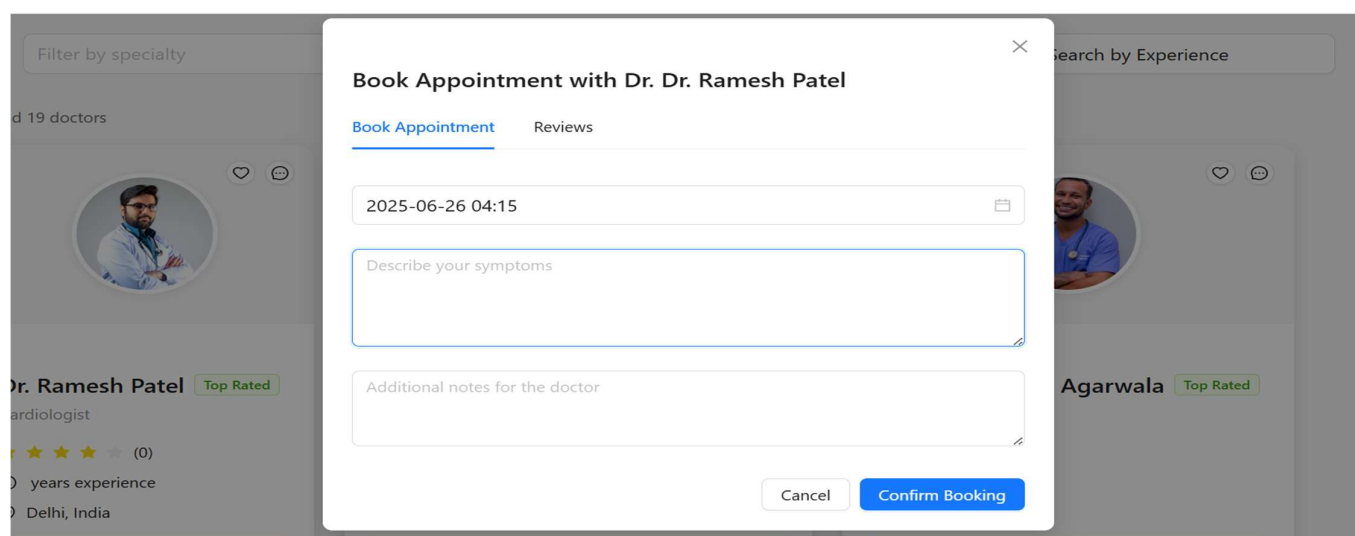
Don't have an account? [Register](#)

[Back to Home](#)

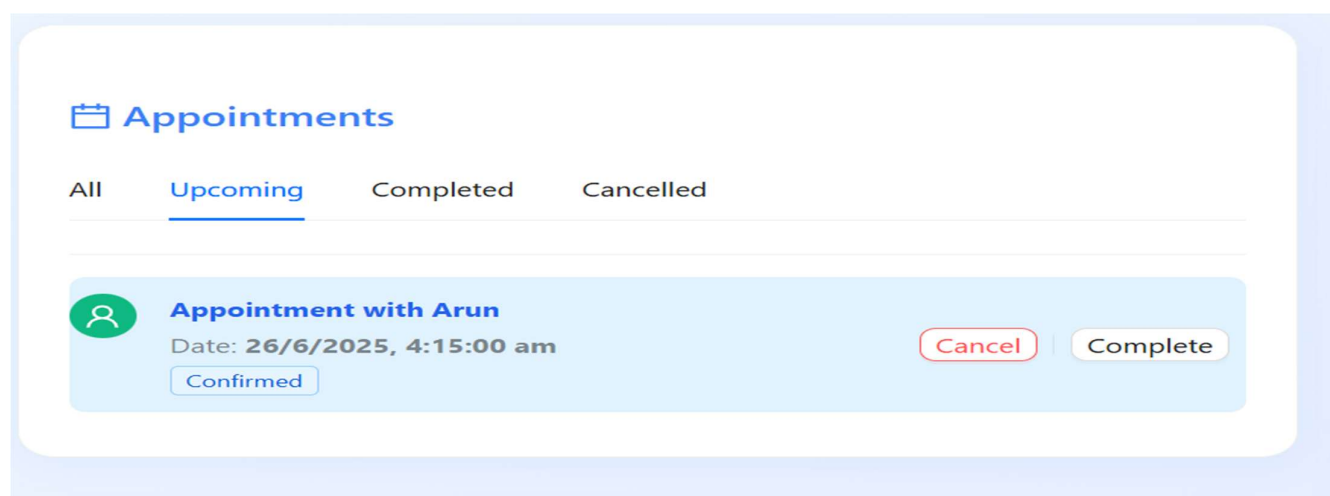
## ADMIN APPROVE DOCTOR :



## BOOK DOCTOR :



## DOCTOR APPROVE USER APPOINTMENT :



ALL HISTORY :

🕒 Pending Doctor Approvals

👤 All Users

👨 All Doctors

All Appointments

User	Doctor	Date	Status
raju	Dr. Ramesh Patel	24/6/2025, 5:00:00 am	Completed
raju	Dr. Anjali Reddy	27/6/2025, 5:00:00 am	Completed
raju	Dr. Vikas Malhotra	27/6/2025, 5:00:00 am	Confirmed
ShivaReddy	Dr. Ramesh Patel	24/6/2025, 4:00:00 am	Completed
raju	Aruna	25/6/2025, 4:00:00 am	Completed
raju	Harshith	24/6/2025, 4:00:00 am	Completed
raju	Dr. Ramesh Patel	25/6/2025, 4:00:00 am	Completed
HarikaKomal	Mahika	24/6/2025, 12:00:00 am	Completed

KNOWN ISSUES:

- No real-time chat system between patient and doctor (currently relies on appointments).
- SMS notifications may be delayed if third-party services fail.
- Doctor verification process is manual and lacks document OCR or automation.
- Image/document upload limits might be restrictive for large files (>5MB).
- No search history caching — filtering doctors may refetch from server each time.
- Lack of role-based dashboard customization — same layout used for all user types.
- Appointment clashes may occur if multiple patients book same slot simultaneously (no concurrency check).
- Time zone support is limited — scheduling across regions can lead to mismatches.
- No audit logging or activity tracking for admin actions (security concern).
- Currently no offline access or Progressive Web App (PWA) support.

FUTURE ENHANCEMENTS:

1. Live chat or video consultation integration (e.g., using WebRTC).
2. Payment gateway integration for online consultation fees.
3. Doctor ratings and reviews from patients.
4. Mobile App version using React Native for iOS and Android.
5. Multilingual support for wider accessibility.
6. Appointment analytics dashboard for doctors/admins.
7. Smart appointment reminders via WhatsApp and Push Notifications.
8. Integration with wearable health devices for real-time health updates.
9. AI-based doctor recommendation system based on symptoms/history.

## **Conclusion**

This project presents the development of a fully functional and scalable Doctor Appointment Booking System, designed to streamline the process of scheduling and managing appointments across three distinct user roles: customers (patients), doctors, and administrators.

Each development phase—environment setup, backend development, database integration, frontend design, and final implementation—was approached methodically to ensure modularity, clarity, and extensibility.

The final system delivers the following key capabilities:

- Secure user authentication and role-based access.
- Efficient doctor-patient appointment management.
- Admin-level monitoring and control functionalities.

By combining technologies like Node.js, Express, MongoDB, React.js, and Ant Design, the application achieves both performance and usability. The modular structure also ensures future enhancements can be incorporated with minimal disruption, making the system well-suited for real-world deployment.