Team ID: LTVIP2025TMID46216

FULL STACK DEVELOPMENT WITH MERN

1.Introduction:

Project Title: Docspot : Seamless Appointment booking for health

Team Members:

- U Lavanya – Team Leader
- U Mounika – Team Member
- Rachamadugu Bhavyasree – Team Member

2. Project Overview:

Purpose:

The Book a Doctor App is an innovative healthcare booking platform designed to streamline the process of connecting patients with healthcare providers. This system enables users to easily find, schedule, and manage medical appointments, all within a user-friendly interface. By offering functionalities like doctor browsing, appointment scheduling, and secure document uploading, the app caters to the needs of patients, doctors, and administrators alike.

Description:

The Book a Doctor App is a user-centric platform designed to make healthcare appointment booking easy and efficient. The app connects patients and healthcare providers through a streamlined digital interface, allowing users to search, filter, and book appointments based on specialty, location, and real-time availability.

For patients, the app offers secure registration, profile creation, and document upload, with automated notifications and reminders to ensure no missed appointments. Doctors benefit from a dedicated dashboard where they can manage availability, confirm bookings, view patient records, and provide post-visit summaries. An admin interface allows for doctor registration approvals, system monitoring, and compliance management, ensuring a smooth, reliable experience.

Built using Bootstrap and Material UI for a modern frontend, the app also uses Axios for seamless backend communication, with Express.js and MongoDB handling server logic and data storage. Moment.js supports precise scheduling, and security libraries like bcrypt ensure secure handling of user data.

With features that enhance accessibility, communication, and efficiency, the Book a Doctor App supports the growing demand for accessible healthcare options, providing patients with convenient, reliable access to healthcare while helping providers manage their schedules effectively.

For doctors and clinics, DocSpot provides a dashboard to manage appointments, view patient details, and reduce scheduling conflicts, improving operational efficiency. The application is built using HTML, CSS, JavaScript, Bootstrap, React.js, Node.js, and MongoDB, ensuring a responsive user interface and a scalable backend to handle user data and appointment records reliably.
away, and the literary world is at your fingertips. It's time to open the door to a future where the love for books meets the convenience of modern technology.

Features:

PATIENT REGISTRATION & PROFILE CREATION:

- SECURE SIGN-UP using email and password authentication.
- PROFILE CREATION that securely stores personal and medical information for future appointments

DOCTOR BROWSING & FILTERING:

- ALLOWS USERS TO SEARCH AND FILTER doctors based on specialty, location, and real-time availability.
- LIVE AVAILABILITY UPDATES ensure patients select only available time slots, minimising scheduling conflicts.

APPOINTMENT BOOKING & MANAGEMENT:

- USER-FRIENDLY BOOKING INTERFACE where patients choose appointment dates, times, and upload relevant documents (e.g., medical records).
- AUTOMATED CONFIRMATION MESSAGES AND REMINDERS via email or SMS help reduce missed appointments.

DOCTOR'S DASHBOARD:

- DOCTORS CAN MANAGE AVAILABILITY, VIEW BOOKINGS, AND UPDATE appointment statuses (e.g., confirmed, completed).
- SECURE ACCESS TO PATIENT RECORDS with options to add visit summaries, follow-up notes, and medical recommendations.

ADMIN CONTROLS & APPROVAL:

- APPROVE DOCTOR REGISTRATIONS, ensuring that only verified healthcare professionals are listed.
- PLATFORM OVERSIGHT, including user management, policy enforcement, and dispute resolution for a smooth user experience.

Scenario Based Case Study:

Aman is a software engineer who recently moved to a new city for work. Due to his busy schedule, he often postpones his health check-ups and finds it difficult to visit clinics to book appointments in person. Aman wants a solution that allows him to discover doctors nearby, view their profiles, check availability, and book appointments conveniently without disrupting his work routine.

User Registration and Authentication:
Allow users to register securely with their email and phone number, set a password, log in, and authenticate their identity to access the DocSpot platform.

Doctor Listings:
Display a comprehensive list of available doctors with details such as name, specialization, location, experience, consultation fees, and availability status.

Doctor Selection:
Provide users with options to select doctors based on filters like specialization, location, consultation fees, ratings, and available slots.

Appointment Booking Process:
Allow users to select preferred doctors, view their available time slots, and book appointments seamlessly. Upon booking, the system generates an appointment entry, updates the doctor's availability, and sends confirmation notifications to the user.

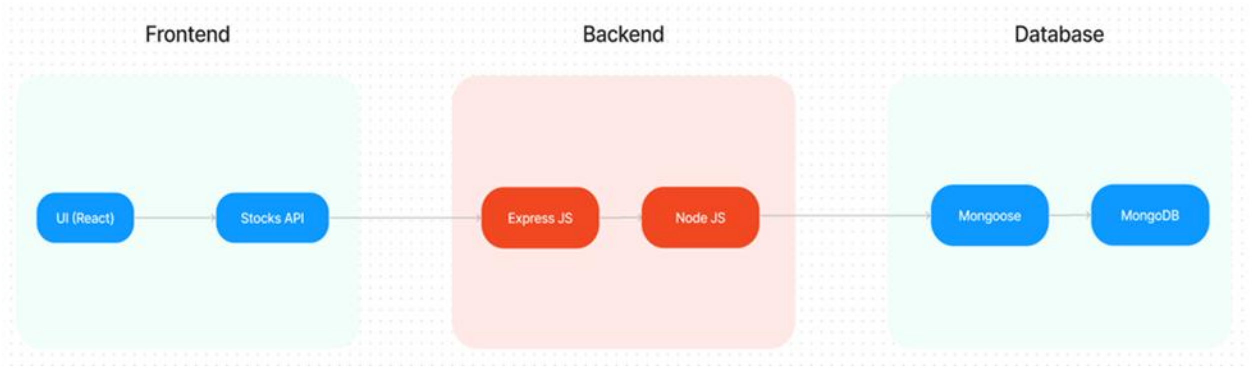
Appointment Confirmation:
Provide users with a confirmation page or notification containing appointment details, including doctor information, consultation time, fees, and a unique appointment ID.

Appointment History:
Allow users to view their past and upcoming appointments, track appointment statuses, reschedule if needed, and add reviews for doctors after their consultation.

3.Architecture:

The Book a Doctor App features a modern technical architecture based on a client-server model. The frontend utilises Bootstrap and Material UI for a responsive user interface, with Axios handling seamless API communication. The backend is powered by Express.js, offering robust server-side logic, while MongoDB provides scalable data storage for user profiles, appointments, and doctor information. Authentication is secured using JWT for session management and bcrypt for password hashing. Moment.js manages date and time functionalities, ensuring accurate appointment scheduling.

Team ID: LTVIP2025TMID46216



Frontend:

- **Bootstrap and Material UI:** Provide a responsive and modern UI that adapts to various devices, ensuring a user-friendly experience.
- **Axios:** A promise-based HTTP client for making requests to the backend, ensuring smooth data communication between the frontend and server.

Backend:

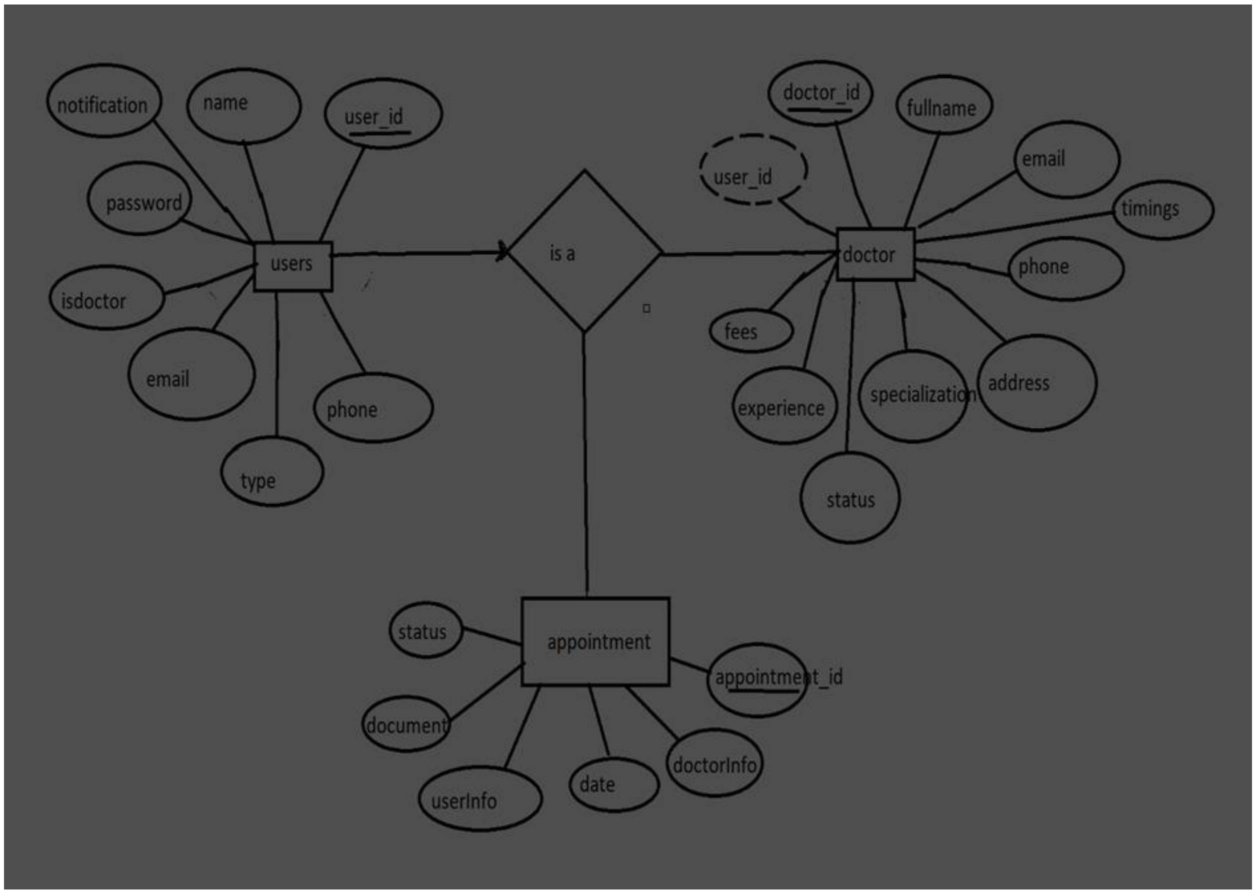
- - Built with Node.js and Express.js
- - RESTful API architecture
- - Middleware for error handling and authentication

Database:

- - MongoDB for data persistence
- **User Interface:**
  The user interface serves as the platform where patients can search for doctors, view profiles, check availability, and book appointments. It is designed to be intuitive and user-friendly, enabling easy navigation, filtering of doctors by specialization and location, and seamless appointment booking from any device.
- **Web Server:**
  The web server hosts the user interface of the DocSpot application, serving dynamic web pages to users and ensuring a responsive, smooth browsing and booking experience. It handles rendering patient dashboards, doctor dashboards, appointment pages, and profile management views.
- **API Gateway:**
  The API gateway acts as the central entry point for client requests, directing them to the relevant backend services. It handles requests such as fetching doctor listings, booking appointments, retrieving user appointment history, and managing user profiles, ensuring efficient routing and load management.

- **Authentication Service:**
  The authentication service manages user authentication and authorization, ensuring secure access to the DocSpot platform. It handles user registration, login, session management, and protects sensitive user data during the appointment booking process.
- **Database:**
  The database stores persistent data related to the DocSpot application, including doctor details (name, specialization, location, availability), patient profiles, appointment records, and consultation history. It ensures reliable data management and quick retrieval to support real-time appointment booking and tracking.

ER-Diagram:



User-Appointment Relationship:
• Type: One-to-Many (1:M). A user can book multiple appointments, but each appointment is tied to one user.
• Implementation: Keep the UserID foreign key in the Appointment table to track user appointment history.

Doctor-Appointment Relationship:
• Type: One-to-Many (1:M). A doctor can have multiple appointments, but each appointment is with one doctor.
• Implementation: Keep the DoctorID foreign key in the Appointment table to track doctor appointments.

Doctor-Specialization Relationship:
• Type: Many-to-Many (M:M). A doctor can have multiple specializations, and a specialization can belong to multiple doctors.
• Implementation: Introduce an intermediate entity, "DoctorSpecialization", with foreign keys to both Doctor and Specialization tables.

User-Review Relationship:
• Type: One-to-Many (1:M). A user can write multiple reviews, but each review is written by one user.
• Implementation: Keep the UserID foreign key in the Review table to track user reviews.

Doctor-Review Relationship:
• Type: One-to-Many (1:M). A doctor can receive multiple reviews, but each review is about one doctor.
• Implementation: Keep the DoctorID foreign key in the Review table to track doctor reviews.

User-Payment Relationship:
• Type: One-to-Many (1:M). A user can make multiple payments, but each payment belongs to one user.
• Implementation: Keep the UserID foreign key in the Payment table to track user payments.

Appointment-Payment Relationship:
• Type: One-to-One (1:1) or One-to-Many (1:M). Each appointment can have one payment record, or optionally allow multiple payments for partial payments.
• Implementation: Keep the AppointmentID foreign key in the Payment table to link payments to appointments.

Hospital-Doctor Relationship:
• Type: One-to-Many (1:M). A hospital can have multiple doctors, but each doctor is affiliated with one hospital.
• Implementation: Keep the HospitalID foreign key in the Doctor table to track doctor affiliation.

Doctor-Availability Relationship:
• Type: One-to-Many (1:M). A doctor can have multiple availability slots, but each slot belongs to one doctor.
• Implementation: Keep the DoctorID foreign key in the Availability table to track doctor availability.

Additional Relationships:
• User-Doctor Favorite Relationship: Many-to-Many (M:M). A user can bookmark many doctors, and a doctor can be bookmarked by many users. (Similar to Doctor-Specialization, use an intermediate "UserFavorites" table)
• User-MedicalRecords Relationship: One-to-Many (1:M). A user can have multiple medical records uploaded, but each record belongs to one user. (Keep the UserID foreign key in the MedicalRecord table)
• User-Notification Relationship: One-to-Many (1:M). A user can receive multiple notifications, but each notification is tied to one user. (Keep the UserID foreign key in the Notification table)
• Doctor-SlotBooking Relationship: One-to-Many (1:M). A doctor can have multiple slot bookings, but each booking is tied to one doctor. (Keep the DoctorID foreign key in the SlotBooking table)

4.Setup Instructions:

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.

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:8001 or the specified port.

ACCESS THE APPLICATION:

- After running the servers, access the Doctor Appointment Webpage in your browser at http://localhost:3000 for the frontend interface and http://localhost:8001 for backend API services.

• Open your terminal or command prompt.

• Navigate to the selected directory using the cd command.

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

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

Navigate into the project directory:

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

Start the development server:

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

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

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

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

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

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

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

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

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

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

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

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

• **Sublime Text:** Download from https://www.sublimetext.com/download

• **WebStorm:** Download from https://www.jetbrains.com/webstorm/download

Roles and Responsibility

User:

• Registration: Users are responsible for registering an account on the DocSpot app by providing essential details such as name, email, password, and contact information.

• Profile Management: Users can manage their profiles, updating details like name, email, contact number, and password.

• Doctor Browsing: Users can browse available doctors, explore different specializations, and search for doctors by name or specialization.

• Appointment Booking: Users can book appointments with doctors by selecting available slots, specifying concerns, and confirming bookings.

• Payment: Users can make secure payments for booked appointments using the integrated payment system.

• Feedback: Users can provide reviews and ratings for doctors after consultations on the DocSpot platform.

• Logout: Lastly, users can logout from the DocSpot app securely**.**

Doctor:

• Registration: Doctors register an account on the DocSpot app by providing necessary details such as name, email, specialization, qualifications, and password.

• Profile Management: Doctors can manage their profiles, updating information like specialization, contact details, qualifications, and profile picture.

• Availability Management: Doctors can manage their availability slots, updating available days and times for consultations.

• Appointment Management: Doctors can view, accept, or manage booked appointments and track patient concerns for consultations.

• Feedback Review: Doctors can view feedback and ratings provided by patients to improve services and maintain patient trust.

• Logout: Finally, doctors can logout from the DocSpot app securely.

Admin:

• System Management: Admins have full control over the DocSpot system, overseeing configurations, functionalities, and platform security.

• User Management: Admins can manage user accounts, including viewing, updating, or deleting user profiles when necessary.

• Doctor Management: Admins can manage doctor accounts, including approving doctor registrations, updating profiles, and monitoring doctor activity on the platform.

• Appointment Oversight: Admins can monitor and manage appointments across the platform to ensure smooth functioning and resolve disputes if necessary.

• Feedback Management: Admins can oversee patient feedback and doctor ratings to maintain quality of service on the platform.

• Logout: Finally, admins can logout from the DocSpot app securely.

User Flow:

Patient (User):

• Registration/Login → Profile Management → Browse Doctors/Search → Book Appointment → Payment → Feedback/Review → Logout

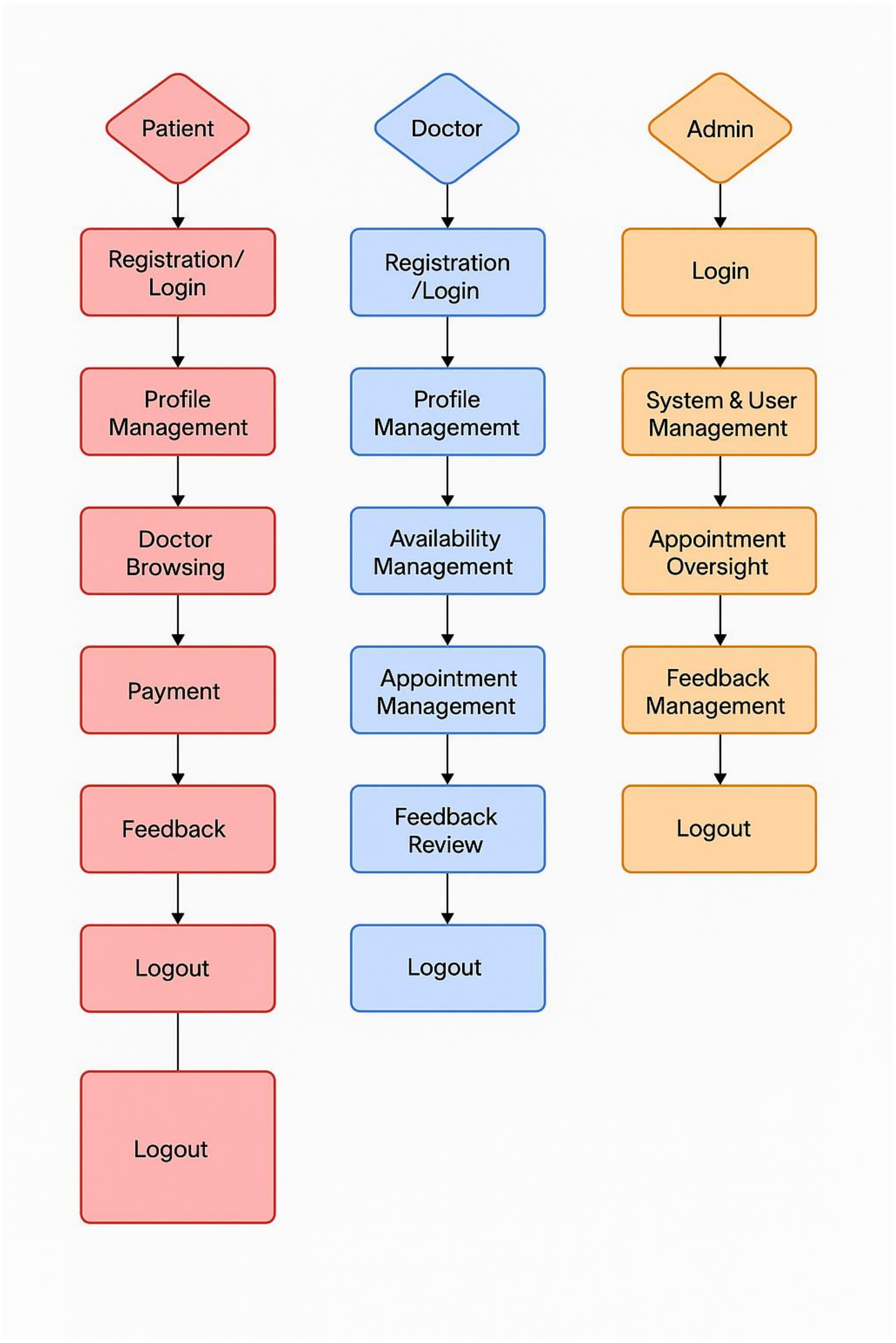
Doctor:

• Registration/Login → Profile Management → Manage Availability → View/Manage Appointments → Review Patient Feedback → Logout

Admin:

• Login → System & User Management → Doctor Management → Appointment Oversight → Feedback Management → Logout

Team ID: LTVIP2025TMID46216



Start:

Users open the DocSpot app to seamlessly book health appointments with doctors.

Home Page:

Users land on the home page, which provides an overview of DocSpot's offerings, such as featured doctors, top specialties, and benefits of using DocSpot. From here, they can navigate to various sections of the app.

Access Profile:

Users have the option to access their profiles, allowing them to view or update personal information, medical preferences, and view their appointment history.

Find Doctors:

After accessing their profiles, users proceed to browse and search for doctors. The app presents a list of available doctors along with details such as name, specialization, location, ratings, and availability.

Book Appointment:

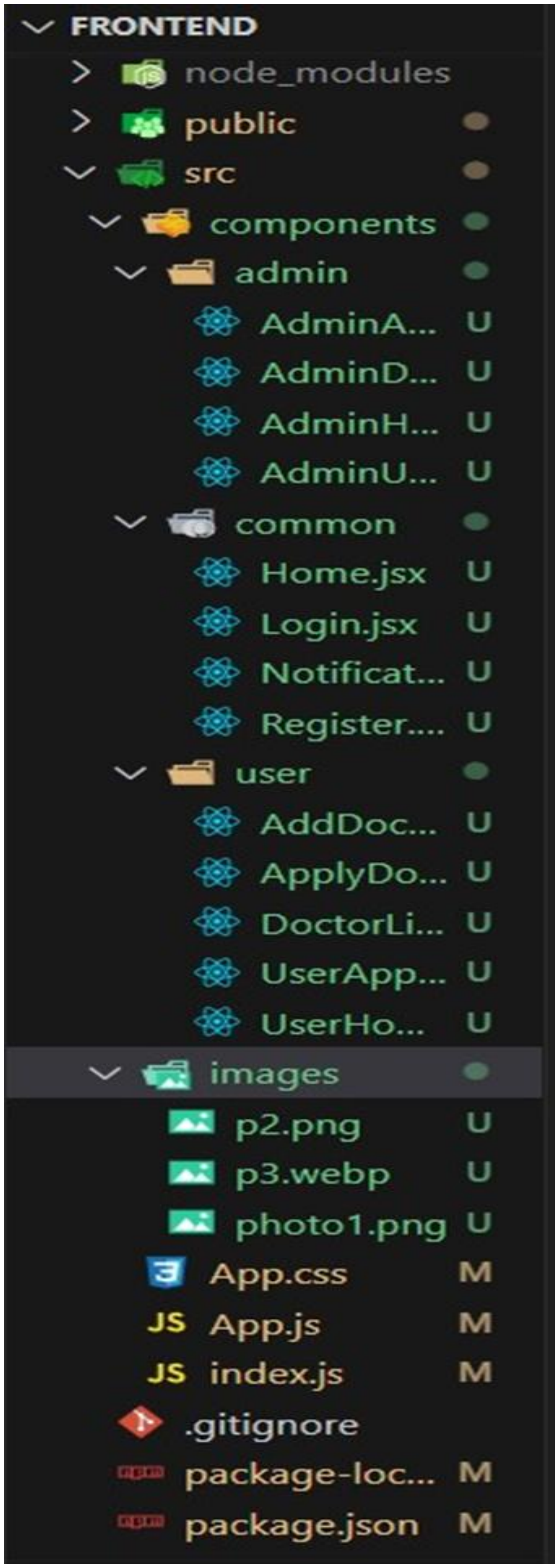
Users navigate through the available doctor options and select a doctor to book an appointment. They specify the date, time slot, and reason for visit. They can also choose the appointment mode (in-person or online consultation).

View Appointments:

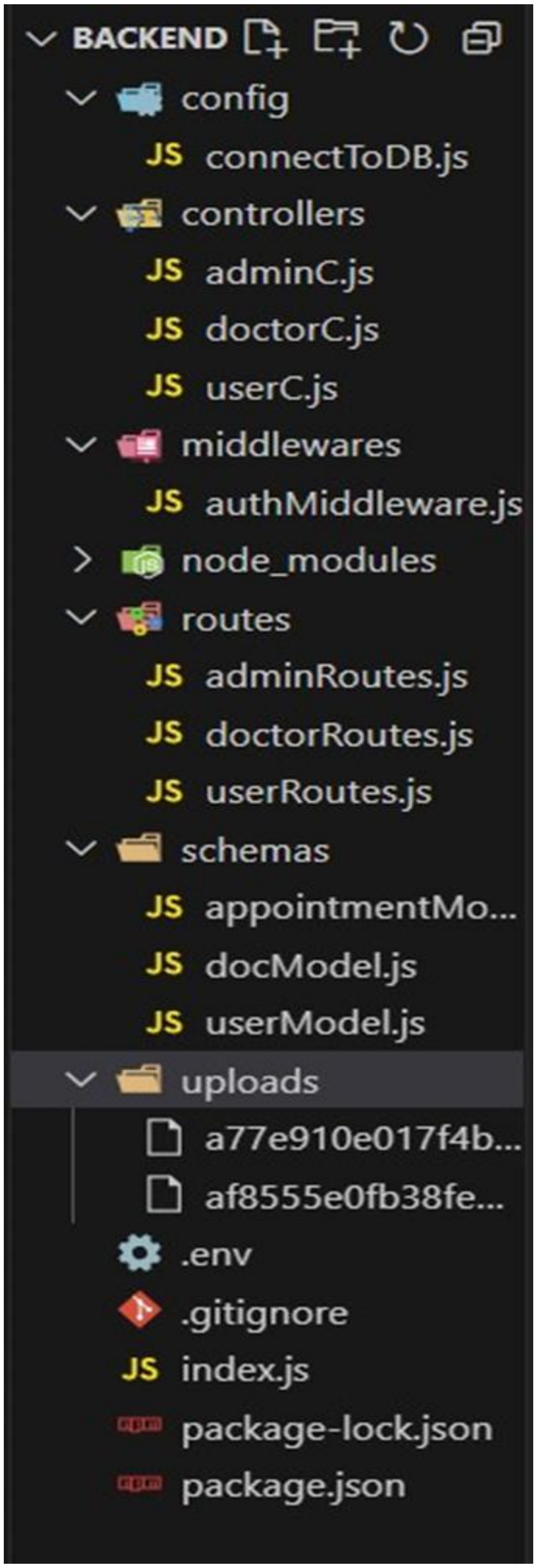
Users have the option to view their current and past appointments. This section provides details about the booked doctor, appointment status, consultation mode, and payment history.

5.Folder Structure:

Team ID: LTVIP2025TMID46216



```
FRONTEND
  > node_modules
  > public                ●
  v src                   ●
    v components          ●
      v admin             ●
          AdminA...       U
          AdminD...       U
          AdminH...       U
          AdminU...       U
      v common            ●
          Home.jsx        U
          Login.jsx       U
          Notificat...    U
          Register....    U
      v user              ●
          AddDoc...       U
          ApplyDo...      U
          DoctorLi...     U
          UserApp...      U
          UserHo...       U
    v images              ●
        p2.png            U
        p3.webp           U
        photo1.png        U
      App.css             M
      App.js              M
      index.js            M
    .gitignore
    package-loc...        M
    package.json          M
```



```
BACKEND
  v config
      connectToDB.js
  v controllers
      adminC.js
      doctorC.js
      userC.js
  v middlewares
      authMiddleware.js
  > node_modules
  v routes
      adminRoutes.js
      doctorRoutes.js
      userRoutes.js
  v schemas
      appointmentMo...
      docModel.js
      userModel.js
  v uploads
      a77e910e017f4b...
      af8555e0fb38fe...
    .env
    .gitignore
    index.js
    package-lock.json
    package.json
```

6.Running the Application

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:

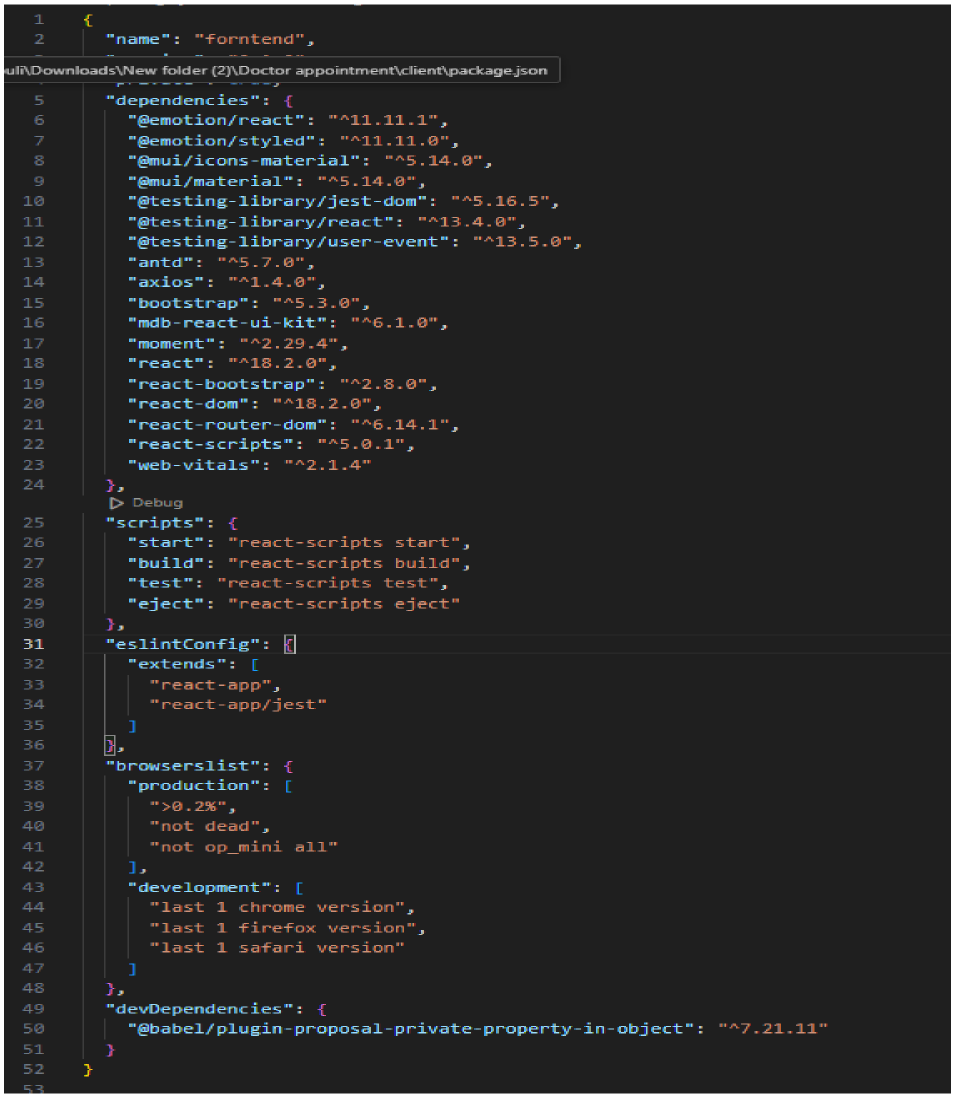
json

Copy code

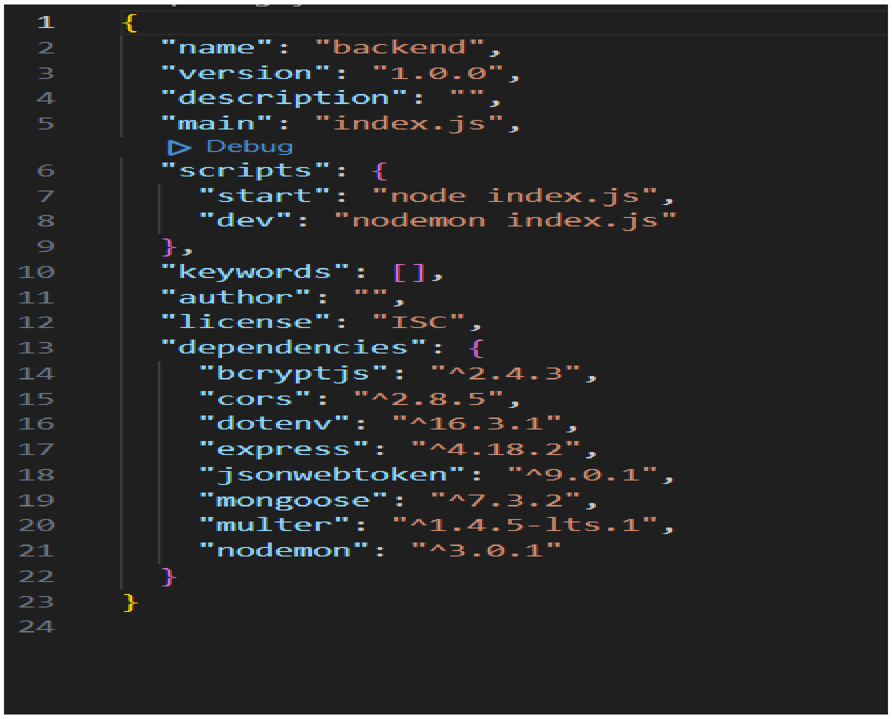
```
"scripts": {
  "start": "concurrently \"npm run server\" \"npm run client\"",
  "server": "node backend/server.js",
  "client": "npm start --prefix frontend"
}
```

MILESTONE 1: PROJECT SETUP AND CONFIGURATION :

- Setting up a structured environment is crucial for any application. By organising the project into separate folders for the frontend and backend, we ensure that the codebase is manageable and scalable.



```
{
  "name": "forntend",
uli\Downloads\New folder (2)\Doctor appointment\client\package.json
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.14.0",
    "@mui/material": "^5.14.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "antd": "^5.7.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.3.0",
    "mdb-react-ui-kit": "^6.1.0",
    "moment": "^2.29.4",
    "react": "^18.2.0",
    "react-bootstrap": "^2.8.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.14.1",
    "react-scripts": "^5.0.1",
    "web-vitals": "^2.1.4"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@babel/plugin-proposal-private-property-in-object": "^7.21.11"
  }
}
```



```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.1",
    "mongoose": "^7.3.2",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.0.1"
  }
}
```

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.

7.API Documentation:

User Authentication

POST /api/users/register
Request Body:
{ name, email, password }
Response: 201 Created

POST /api/users/login
Request Body:
{ email, password }
Response: 200 OK, JWT token and user info

Doctors

GET /api/doctors
Response: 200 OK, JSON list of doctors

GET /api/doctors/:id
Response: 200 OK, JSON of doctor details

Appointments

POST /api/appointments
Request Body:
{ userId, doctorId, date, time, symptoms }
Response: 201 Created

GET /api/appointments/user/:userId
Response: 200 OK, JSON list of user's appointments

GET /api/appointments/doctor/:doctorId
Response: 200 OK, JSON list of doctor's appointments

Payments

POST /api/payments
Request Body:
{ appointmentId, userId, amount, paymentMethod }
Response: 201 Created

GET /api/payments/user/:userId
Response: 200 OK, JSON list of user's payments

Reviews

POST /api/reviews
Request Body:
{ userId, doctorId, rating, comment }
Response: 201 Created

GET /api/reviews/doctor/:doctorId
Response: 200 OK, JSON list of reviews for a doctor

Admin

GET /api/admin/users
Response: 200 OK, JSON list of all users

GET /api/admin/doctors
Response: 200 OK, JSON list of all doctors

GET /api/admin/appointments
Response: 200 OK, JSON list of all appointments

8. Authentication:

• Users and doctors register with name, email, and password.

• Secure login using email and password.

• JWT token issued on login for protected routes.

• Role-based access control (user, doctor, admin).

• Passwords hashed securely with bcrypt.

• Token expiry for session security.

• Logout clears token from client.

9. User Interface:

• Clean and intuitive design for patients, doctors, and admins.

• Easy registration and secure login screens.

• Patients can book appointments, view schedules, make payments, and give feedback.

• Doctors can manage profiles, appointments, and availability slots.

10. Testing:

• Tested all core functionalities including registration, login, appointment booking, and payment workflows.

• Verified role-based access and data validation to ensure security and smooth user experience.

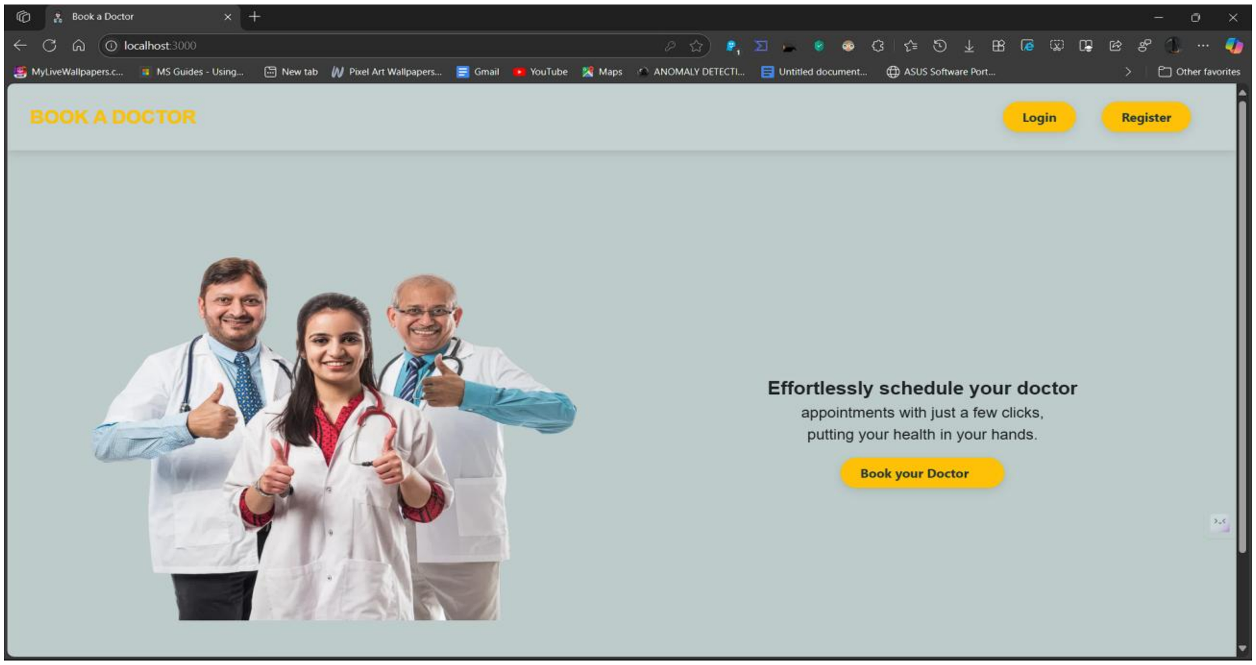
11. Screenshots or Demo:

• Tested all core functionalities including registration, login, appointment booking, and payment workflows.

• Verified role-based access and data validation to ensure security and smooth user experience.

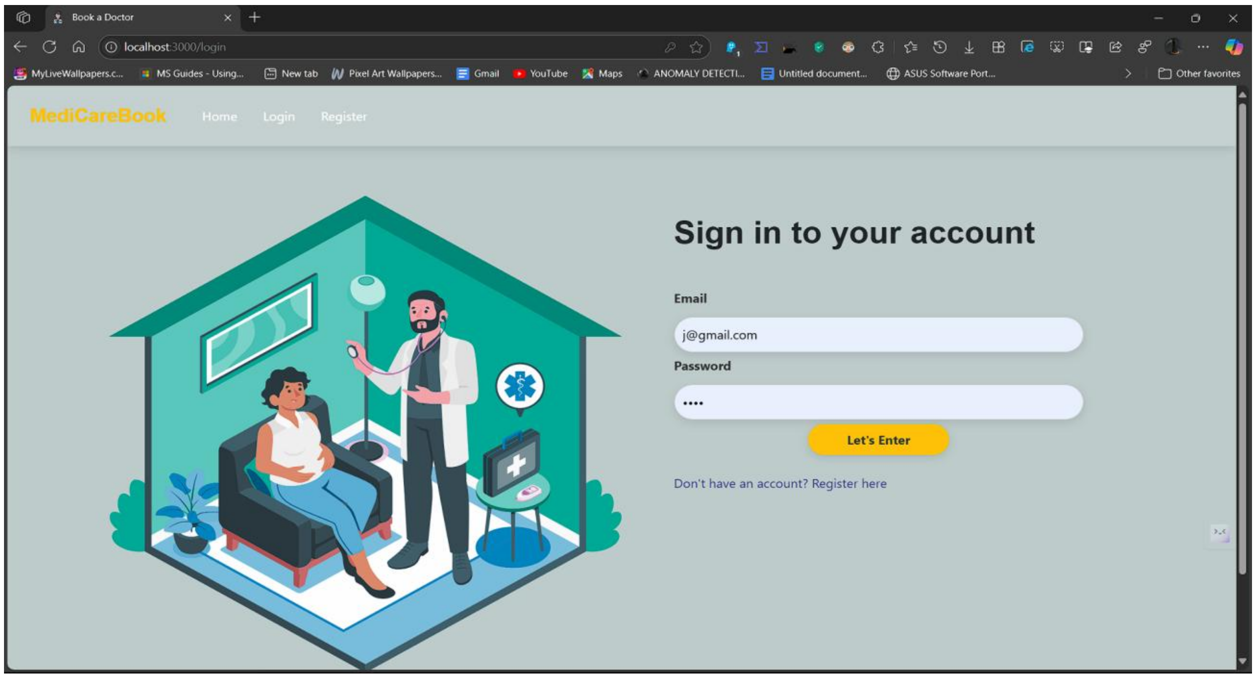
LANDING PAGE :

Team ID: LTVIP2025TMID46216



Effortlessly schedule your doctor appointments with just a few clicks, putting your health in your hands.

LOGIN PAGE :



Sign in to your account

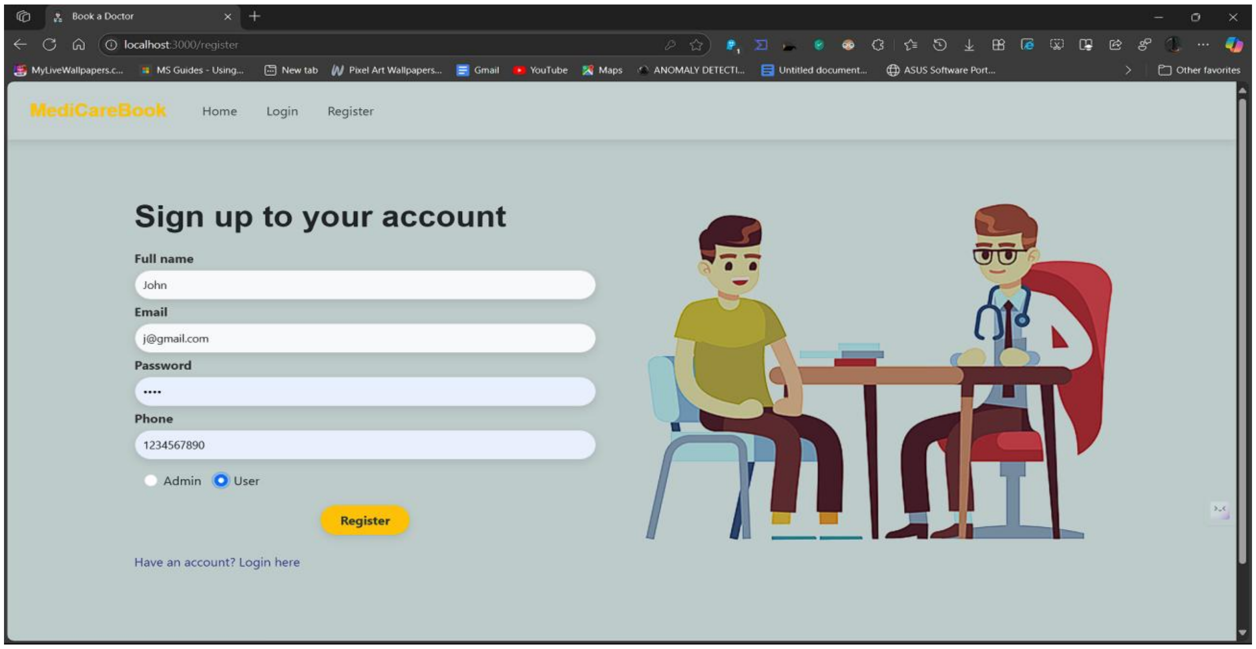
Email: j@gmail.com

Password: ••••

Don't have an account? Register here

REGISTRATION PAGE :

Team ID: LTVIP2025TMID46216



Sign up to your account

Full name: John

Email: j@gmail.com

Password: ••••

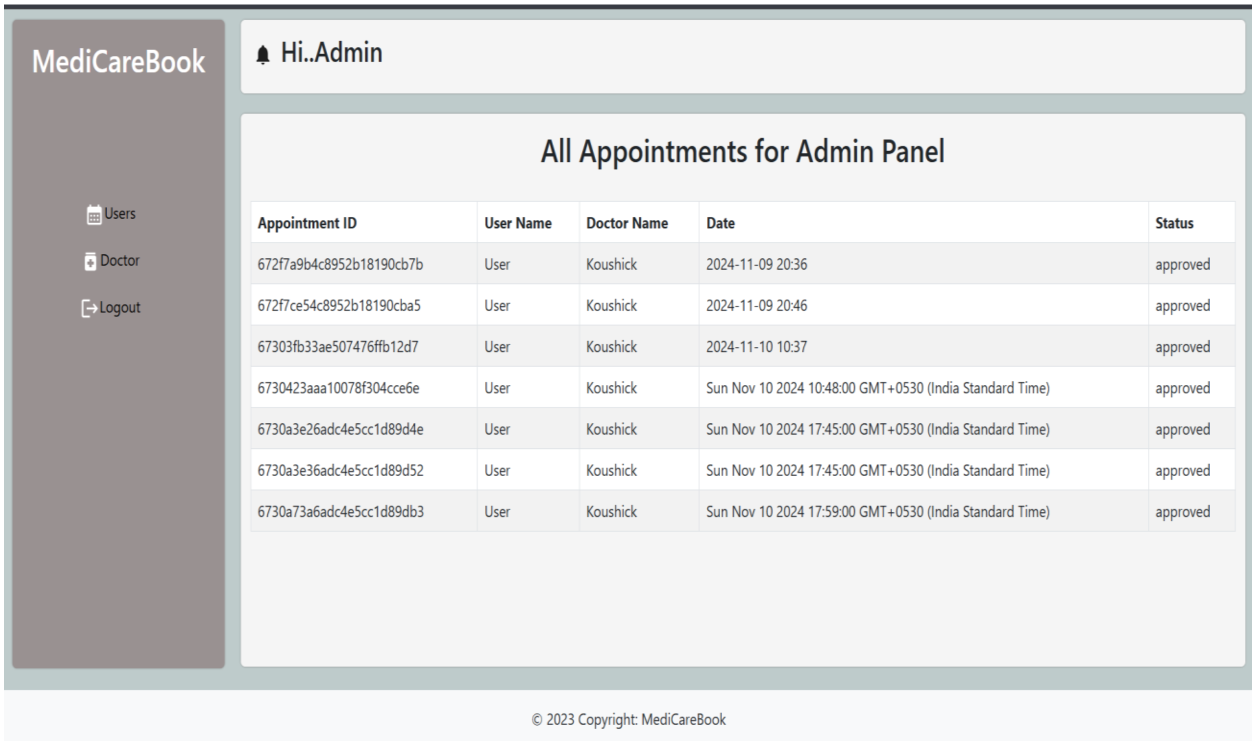
Phone: 1234567890

Admin [✓ User]

Register

Have an account? Login here

USER PAGE :



Hi..Admin

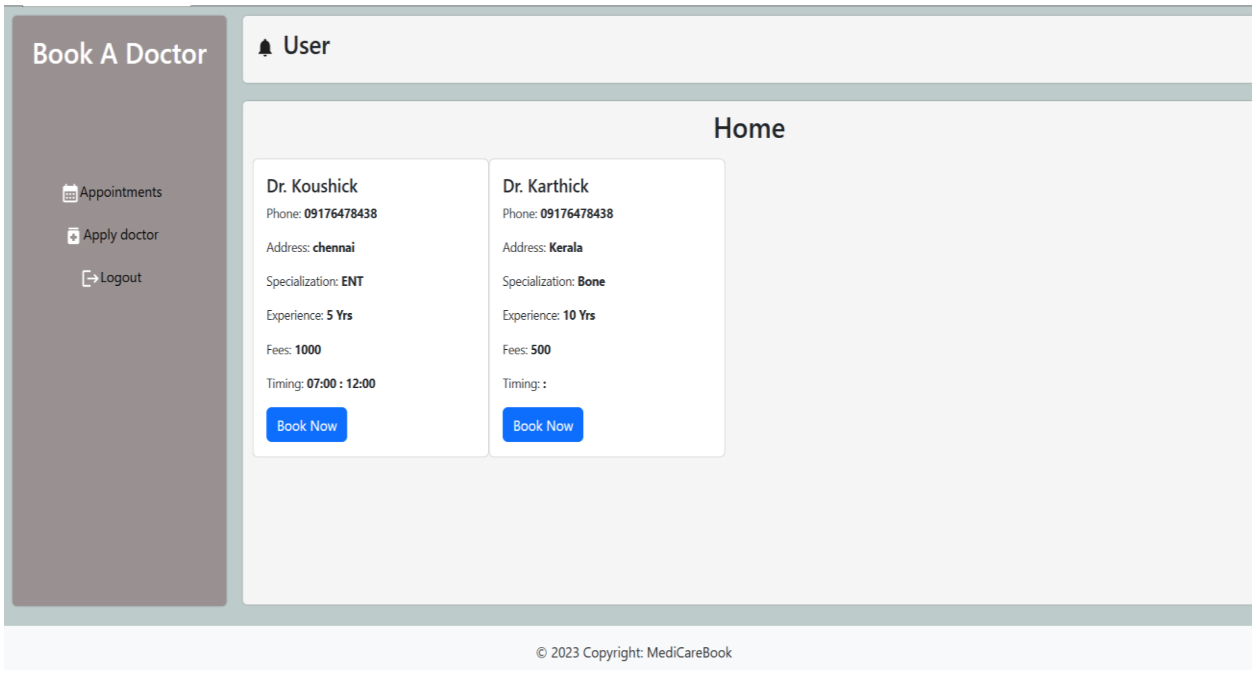
All Appointments for Admin Panel

Appointment ID	User Name	Doctor Name	Date	Status
672f7a9b4c8952b18190cb7b	User	Koushick	2024-11-09 20:36	approved
672f7ce54c8952b18190cba5	User	Koushick	2024-11-09 20:46	approved
67303fb33ae507476ffb12d7	User	Koushick	2024-11-10 10:37	approved
6730423aaa10078f304cce6e	User	Koushick	Sun Nov 10 2024 10:48:00 GMT+0530 (India Standard Time)	approved
6730a3e26adc4e5cc1d89d4e	User	Koushick	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	approved
6730a3e36adc4e5cc1d89d52	User	Koushick	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	approved
6730a73a6adc4e5cc1d89db3	User	Koushick	Sun Nov 10 2024 17:59:00 GMT+0530 (India Standard Time)	approved

© 2023 Copyright: MediCareBook

ADMIN PAGE :

Team ID: LTVIP2025TMID46216



Book A Doctor

User

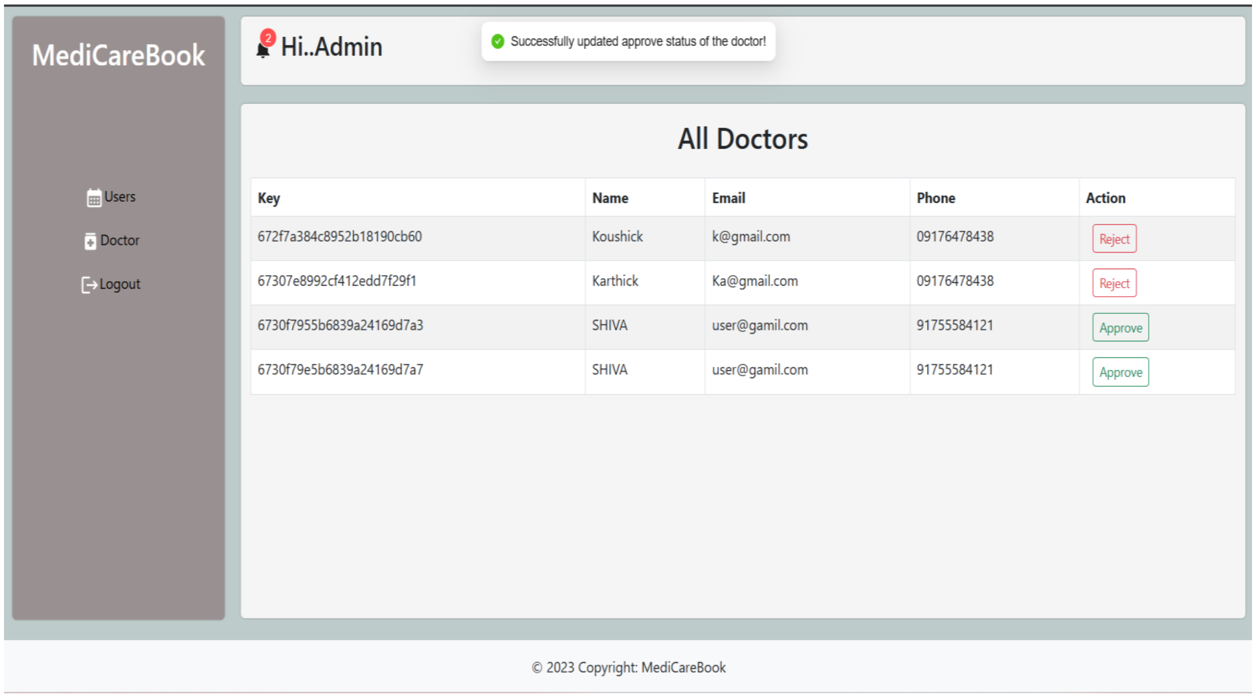
Appointments
Apply doctor
Logout

Home

Dr. Koushick	Dr. Karthick
Phone: **09176478438**	Phone: **09176478438**
Address: **chennai**	Address: **Kerala**
Specialization: **ENT**	Specialization: **Bone**
Experience: **5 Yrs**	Experience: **10 Yrs**
Fees: **1000**	Fees: **500**
Timing: **07:00 : 12:00**	Timing: **:**
Book Now	Book Now

© 2023 Copyright: MediCareBook

APPLY AS DOCTOR :



MediCareBook

Hi..Admin

Successfully updated approve status of the doctor!

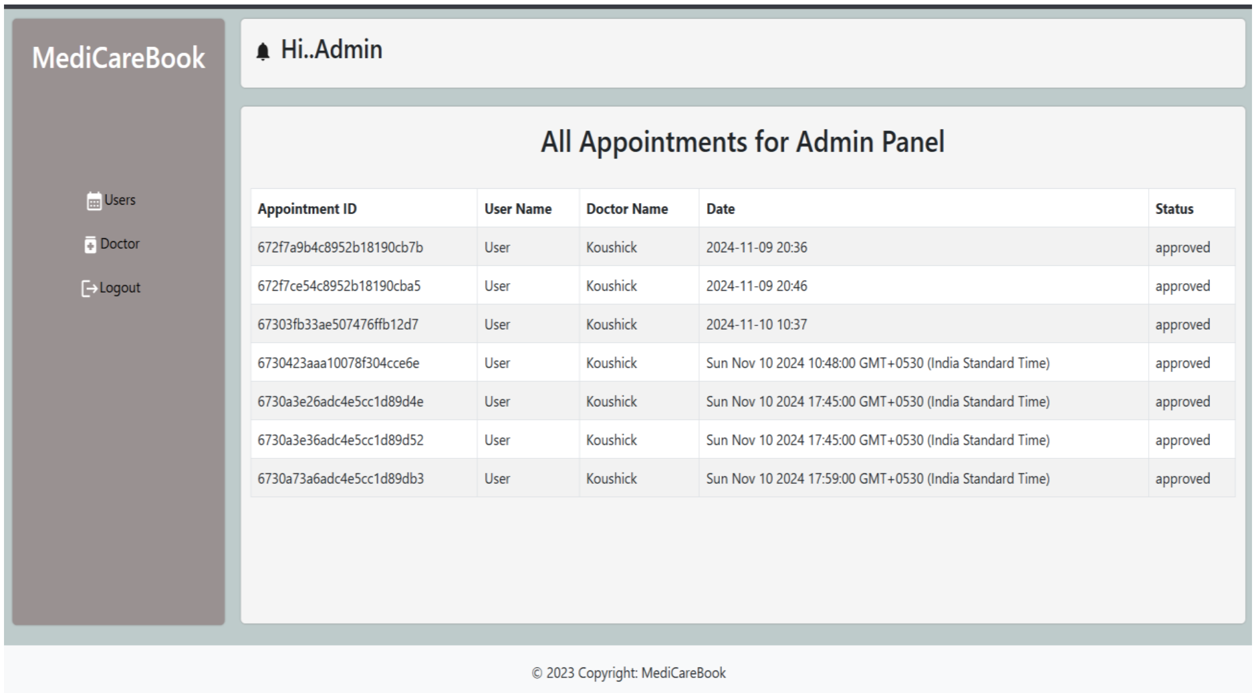
Users
Doctor
Logout

All Doctors

Key	Name	Email	Phone	Action
672f7a384c8952b18190cb60	Koushick	k@gmail.com	09176478438	Reject
67307e8992cf412edd7f29f1	Karthick	Ka@gmail.com	09176478438	Reject
6730f7955b6839a24169d7a3	SHIVA	user@gamil.com	91755584121	Approve
6730f79e5b6839a24169d7a7	SHIVA	user@gamil.com	91755584121	Approve

© 2023 Copyright: MediCareBook

Team ID: LTVIP2025TMID46216



MediCareBook

Users
Doctor
Logout

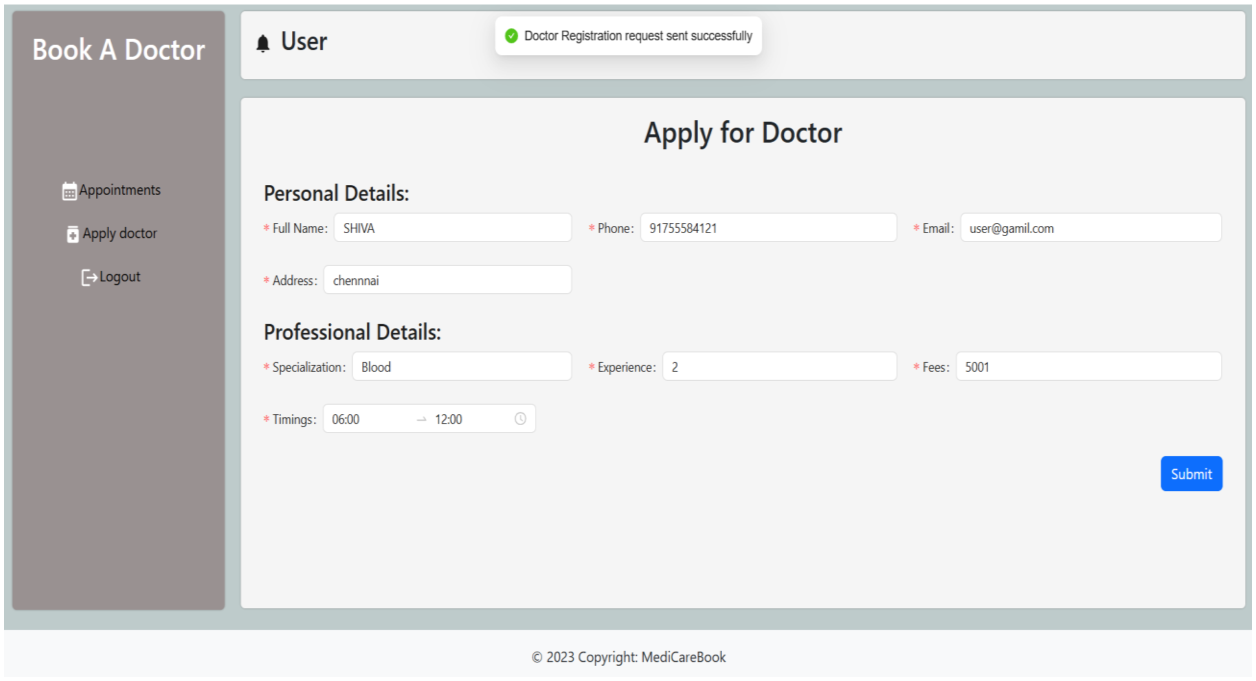
Hi..Admin

All Appointments for Admin Panel

Appointment ID	User Name	Doctor Name	Date	Status
672f7a9b4c8952b18190cb7b	User	Koushick	2024-11-09 20:36	approved
672f7ce54c8952b18190cba5	User	Koushick	2024-11-09 20:46	approved
67303fb33ae507476ffb12d7	User	Koushick	2024-11-10 10:37	approved
6730423aaa10078f304cce6e	User	Koushick	Sun Nov 10 2024 10:48:00 GMT+0530 (India Standard Time)	approved
6730a3e26adc4e5cc1d89d4e	User	Koushick	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	approved
6730a3e36adc4e5cc1d89d52	User	Koushick	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	approved
6730a73a6adc4e5cc1d89db3	User	Koushick	Sun Nov 10 2024 17:59:00 GMT+0530 (India Standard Time)	approved

© 2023 Copyright: MediCareBook

ADMIN APPROVE DOCTOR :



Book A Doctor

Appointments
Apply doctor
Logout

User

Doctor Registration request sent successfully

Apply for Doctor

Personal Details:

* Full Name: SHIVA	* Phone: 91755584121	* Email: user@gamil.com
* Address: chennnai		

Professional Details:

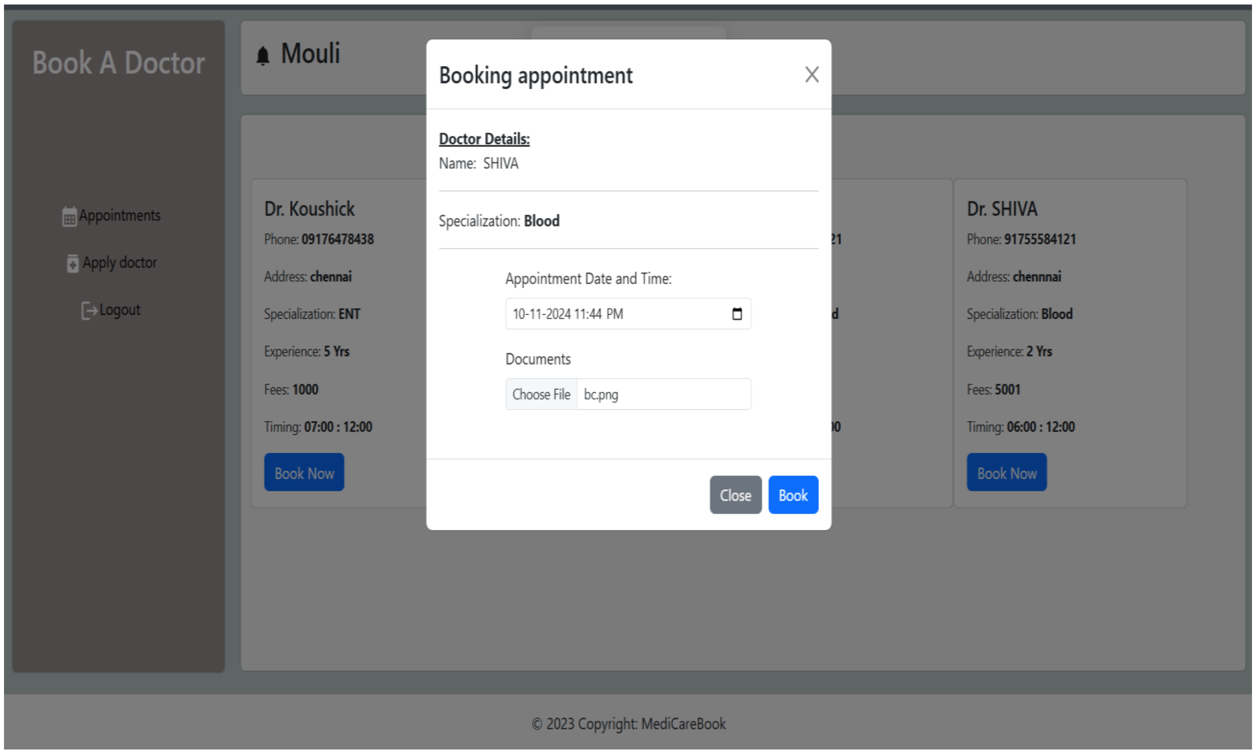
* Specialization: Blood	* Experience: 2	* Fees: 5001
* Timings: 06:00 → 12:00		

Submit

© 2023 Copyright: MediCareBook

BOOK DOCTOR :

Team ID: LTVIP2025TMID46216



Book A Doctor

Appointments
Apply doctor
Logout

Mouli

Dr. Koushick
Phone: **09176478438**
Address: **chennai**
Specialization: **ENT**
Experience: **5 Yrs**
Fees: **1000**
Timing: **07:00 : 12:00**
Book Now

Dr. SHIVA
Phone: **91755584121**
Address: **chennnai**
Specialization: **Blood**
Experience: **2 Yrs**
Fees: **5001**
Timing: **06:00 : 12:00**
Book Now

Booking appointment

Doctor Details:
Name: SHIVA

Specialization: **Blood**

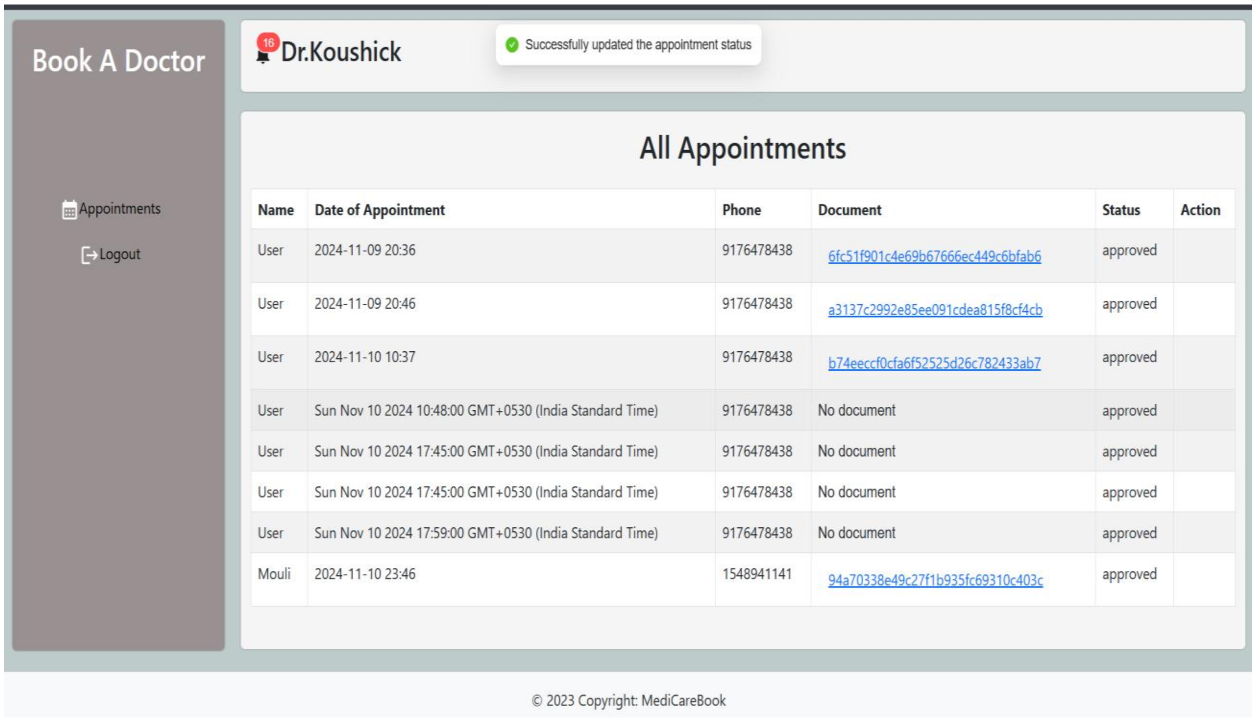
Appointment Date and Time: 10-11-2024 11:44 PM

Documents: Choose File bc.png

Close Book

© 2023 Copyright: MediCareBook

DOCTOR APPROVE USER APPOINTMENT :



Book A Doctor

Appointments
Logout

16 Dr.Koushick

Successfully updated the appointment status

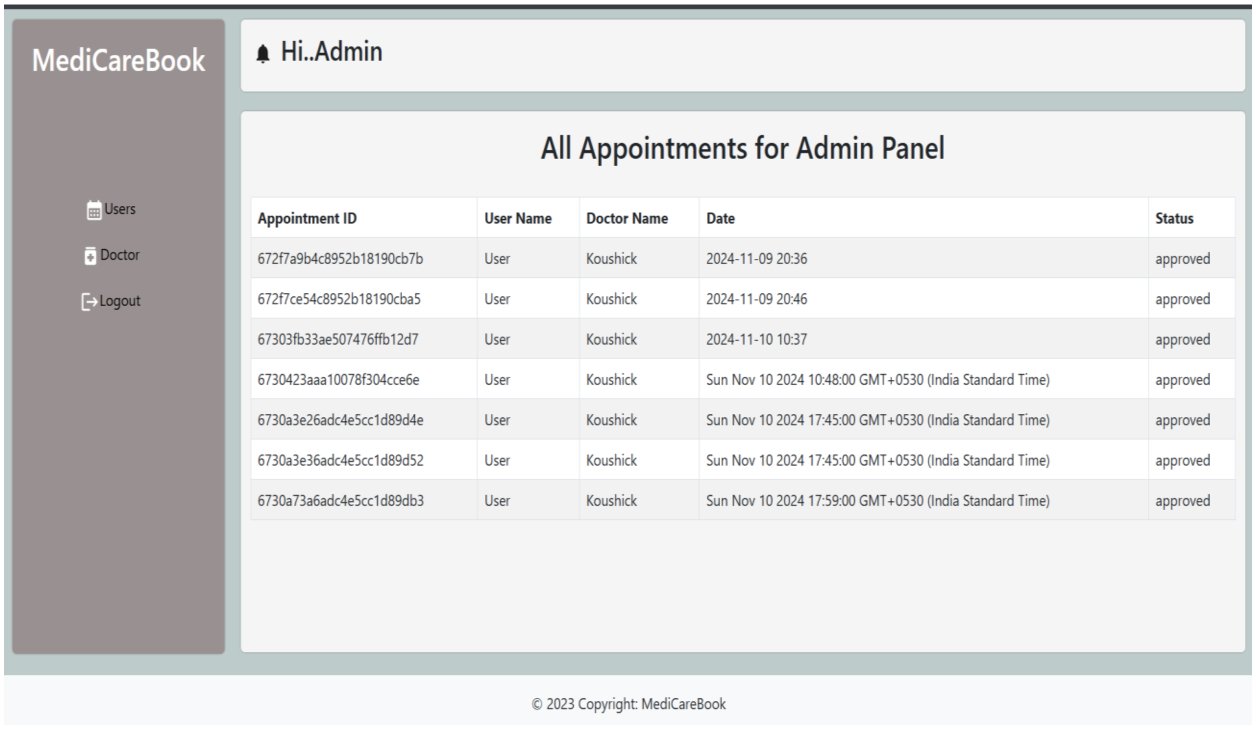
All Appointments

Name	Date of Appointment	Phone	Document	Status	Action
User	2024-11-09 20:36	9176478438	6fc51f901c4e69b67666ec449c6bfab6	approved	
User	2024-11-09 20:46	9176478438	a3137c2992e85ee091cdea815f8cf4cb	approved	
User	2024-11-10 10:37	9176478438	b74eeccf0cfa6f52525d26c782433ab7	approved	
User	Sun Nov 10 2024 10:48:00 GMT+0530 (India Standard Time)	9176478438	No document	approved	
User	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	9176478438	No document	approved	
User	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	9176478438	No document	approved	
User	Sun Nov 10 2024 17:59:00 GMT+0530 (India Standard Time)	9176478438	No document	approved	
Mouli	2024-11-10 23:46	1548941141	94a70338e49c27f1b935fc69310c403c	approved	

© 2023 Copyright: MediCareBook

Team ID: LTVIP2025TMID46216

ALL HISTORY :



All Appointments for Admin Panel

Appointment ID	User Name	Doctor Name	Date	Status
672f7a9b4c8952b18190cb7b	User	Koushick	2024-11-09 20:36	approved
672f7ce54c8952b18190cba5	User	Koushick	2024-11-09 20:46	approved
67303fb33ae507476ffb12d7	User	Koushick	2024-11-10 10:37	approved
6730423aaa10078f304cce6e	User	Koushick	Sun Nov 10 2024 10:48:00 GMT+0530 (India Standard Time)	approved
6730a3e26adc4e5cc1d89d4e	User	Koushick	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	approved
6730a3e36adc4e5cc1d89d52	User	Koushick	Sun Nov 10 2024 17:45:00 GMT+0530 (India Standard Time)	approved
6730a73a6adc4e5cc1d89db3	User	Koushick	Sun Nov 10 2024 17:59:00 GMT+0530 (India Standard Time)	approved

© 2023 Copyright: MediCareBook

App demo link: Demo

12. Known Issues

• Limited error handling for payment failures under unstable network conditions.

• No automated email/SMS reminders for upcoming appointments currently implemented.

13. Future Enhancements

• Add video consultation feature for remote appointments.

• Implement advanced search and filter options for doctors.

• Enable wallet system for easier payments and refunds.

• Add analytics dashboard for doctors and admins.