DocSpot: Seamless Appointment Booking For Health

Institution Name : GATES INSTITUTE OF TECHNOLOGY

Team ID :LTVIP2025TMID49232

Team Members :

1.Team Leader : Y Mahesh Naidu

2.Team member : V Priyanka

3.Team member : Thota Harshavardhan Reddy

4.Team member : Venkatesh Chakali

INTRODUCTION

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.

Patients can search for doctors based on specialty, location, and availability, ensuring theyfind the right healthcare professional for their needs. Once a suitable doctor is selected, users can book appointments, manage their schedules, and receive notifications and reminders.

Doctors benefit from a dedicated interface to manage appointments, update patient records, and communicate effectively, while administrators oversee the app's smooth operation, ensuring compliance and resolving any disputes. Built with a robust technical architecture, the Book a Doctor App leverages a client-server model, using front-end frameworks like Bootstrap and Material UI for an engaging user experience, and a back end powered by Express.js and MongoDB to handle secure data transactions. This system offers a seamless, efficient, and secure healthcare booking

experience, meeting the growing demand for accessible and well-organised healthcare

services.

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,

Scenario based case-study

1. USER REGISTRATION :

John, a patient in need of a routine check-up, downloads and opens the Book a Doctor App. He starts by registering as a customer, providing his email address and creating a password to ensure a secure login. Once the registration process is complete, John is welcomed to the app with the option to log in.

2. BROWSING DOCTORS :

After logging in, John is directed to a dashboard showcasing a list of doctors available for

3. BOOKING AN APPOINTMENT :

John selects Dr. Smith, a family physician, and clicks the “Book Now” button. A booking form appears, prompting John to select his preferred appointment date and time. He is also asked to upload relevant documents, such as his medical records and insurance details. Once the form is completed, John submits the appointment request. He receives an immediate confirmation message indicating that his request is under review.

4. APPOINTMENT CONFIRMATION :

Dr. Smith, upon reviewing the request and his schedule, confirms the appointment. The status of John’s appointment changes to “Scheduled,” and John receives a notification with the appointment details—date, time, and location—via both email and SMS.

5. APPOINTMENT MANAGEMENT :

As the appointment date nears, John can access his booking history through the app’s dashboard. Here, he can manage upcoming appointments, cancel or reschedule them, and update their status. If needed, he can contact the doctor or the support team for assistance.

6. ADMIN APPROVAL (BACKGROUND PROCESS) :

In the background, the app’s admin is reviewing new doctor registrations. Dr. Smith, as a legitimate healthcare professional, is approved and added to the platform. The admin ensures that only verified doctors are listed, and the platform remains compliant with healthcare regulations and policies.

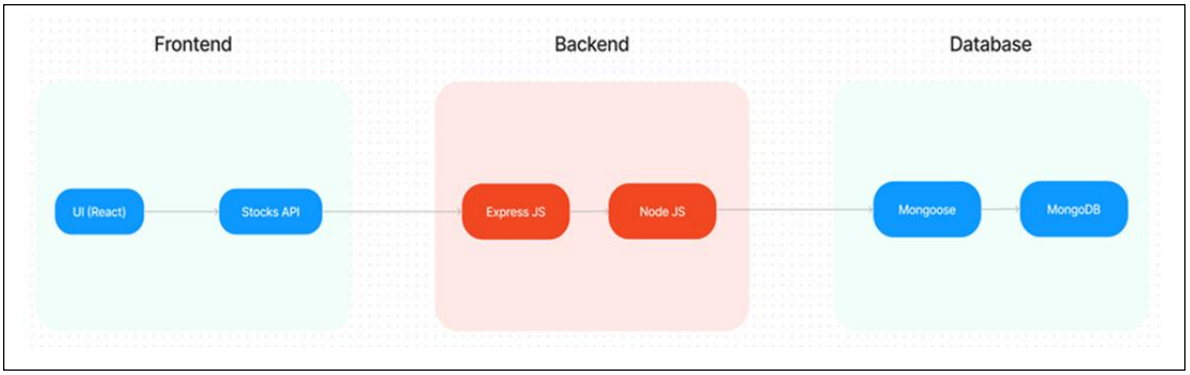
7. PLATFORM GOVERNANCE :

The admin monitors the platform’s overall operation, addressing any issues, disputes, or system improvements. Ensuring the app’s compliance with privacy regulations and the terms of service is also a key responsibility, ensuring a smooth and secure experience for all users.

8. DOCTOR’S APPOINTMENT MANAGEMENT :

TECHNICAL 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. The admin interfaces overseas doctor registration, platform governance, and ensures compliance, with Role-based Access Control (RBAC) managing access levels. Scalability is supported by MongoDB, and performance optimization is achieved with load balancing and caching techniques.



FRONTEND TECHNOLOGIES :

- **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 FRAMEWORK :

- **Express.js:** A lightweight Node.js framework used to handle server-side logic, API routing, and HTTP request/response management, making the backend scalable and

DATABASE AND AUTHENTICATION :

- **MongoDB**: A NoSQL database used for flexible and scalable storage of user data, doctor profiles, and appointment records. It supports fast querying and large data volumes.
- **JWT (JSON Web Tokens):** Used for secure, stateless authentication, allowing users to remain logged in without requiring session storage on the server.
- **Bcrypt**: A library for hashing passwords, ensuring that sensitive data is securely stored in the database.

ADMIN PANEL & GOVERNANCE :

- **Admin Interface:** Provides functionality for platform admins to approve doctor registrations, manage platform settings, and oversee day-to-day operations.
- **Role-based Access Control (RBAC):** Ensures different users (patients, doctors, admins) have appropriate access levels to the system's features and data, maintaining privacy and security.

SCALABILITY AND PERFORMANCE :

- **MongoDB:** Scales horizontally, supporting increased data storage and high user traffic as the platform grows.

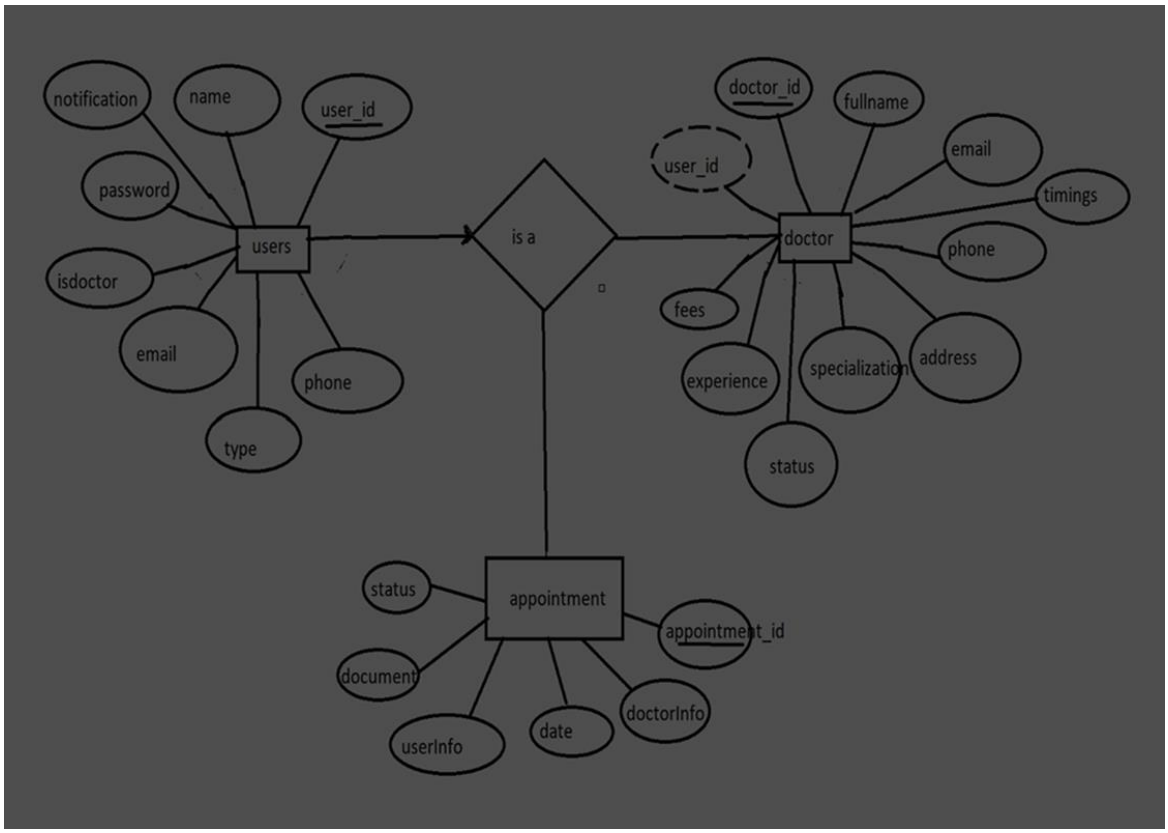
Load Balancing: Ensures traffic is evenly distributed across servers to optimise performance, especially during high traffic periods.

- **Caching:** Reduces database load by storing frequently requested data temporarily, speeding up response times and improving user experience.

TIME MANAGEMENT AND SCHEDULING

- **Moment.js:** Utilised for handling date and time operations, ensuring precise appointment scheduling, time zone handling, and formatting.

ER DIAGRAM :



- The Entity-Relationship (ER) diagram for the Book a Doctor app represents three key entities: Users, Doctors, and Appointments, with their respective attributes and relationships.

- The Users collection holds basic user information, including _id, name, email, notification, password, isdoctor (to differentiate between patients and doctors), type, and phone. The isdoctor field identifies users who are doctors, while others are treated as patients or admins.

- The Doctors collection stores information specific to doctors, such as their _id, userID (acting as a foreign key referencing the Users collection), fullname, email, timings, phone, address, specialisation, status, experience, and fees. The userID` links each doctor to their corresponding user account.

- The Appointments collection stores details about appointments, including the _id, doctorInfo (foreign key referencing the Doctors collection), date, userInfo (foreign

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

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 :

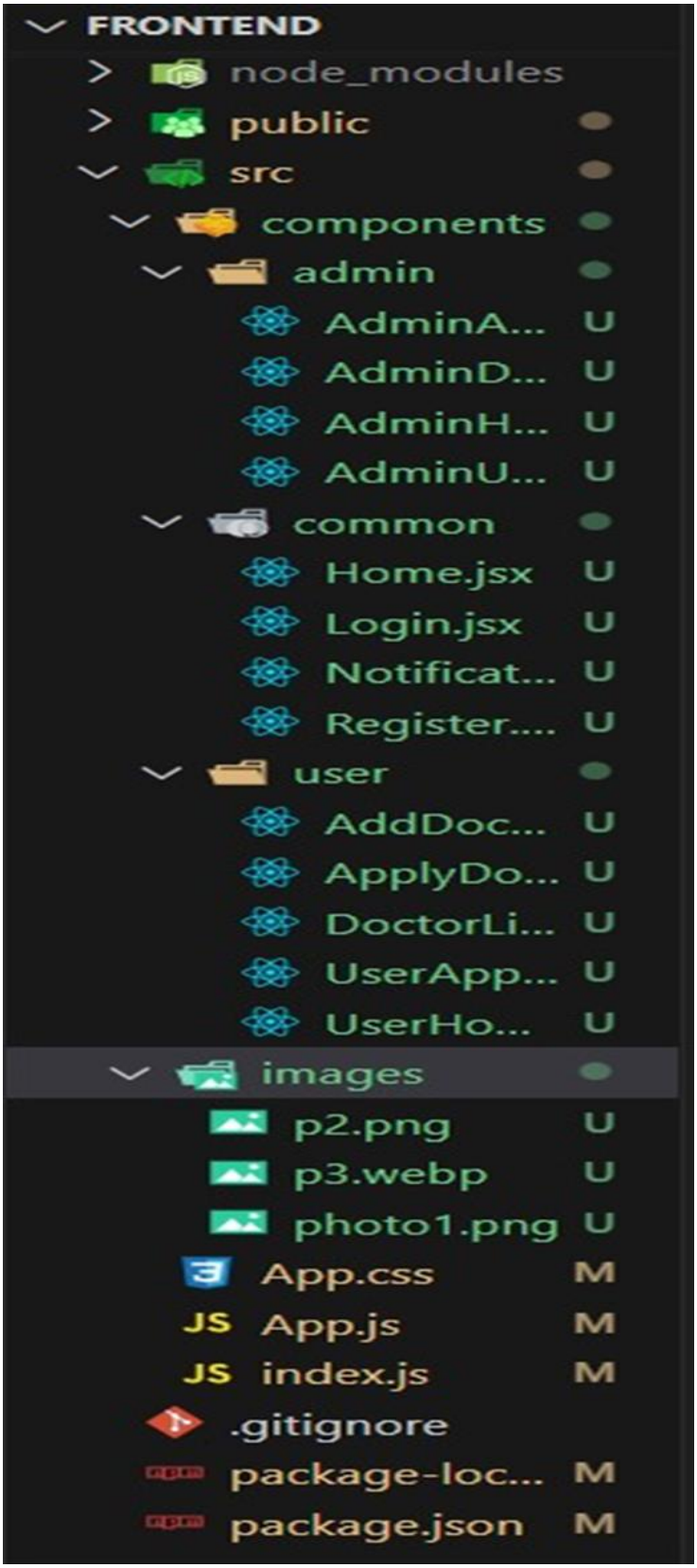
CLONE THE PROJECT REPOSITORY:

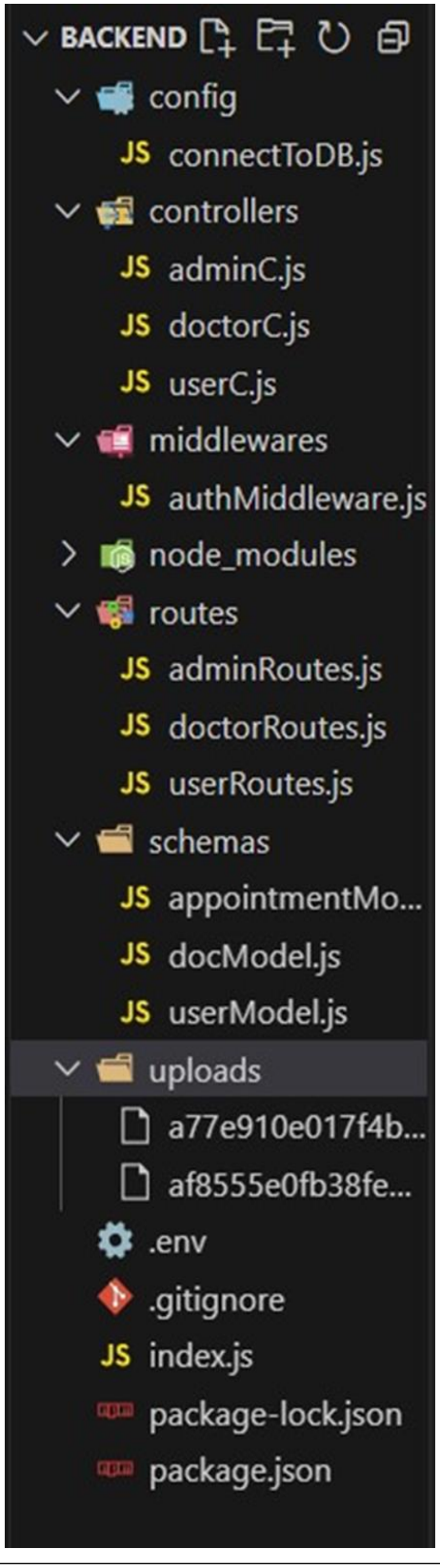
- Download the project files from GitHub or clone the repository using Git.

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.

PROJECT STRUCTURE :





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:

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

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:

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:

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:

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
- Copy code
- git init

Add your project files and commit them:

- bash

PROJECT ROOT STRUCTURE :

Create the Main Folders:

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

FRONTEND SETUP :

- React Project Initialization:
-
- Navigate to the frontend folder and initialize a new React application:
- plaintext
- Copy code
- frontend/

  ├── public/

  ├── src/

  │ ├── components/

  │ ├── pages/

  │ ├── services/

PROJECT FLOW :

- Use this code:
- File1:https://drive.google.com/drive/folders/1WeYIuwEcAXq0xdz_qaVqfpnbBF5MwuR8?usp=drive_link
- File2:https://drive.google.com/drive/folders/1iJz6F2oNdoylTkO9f-m6YcTXTB6c2Ve9?usp=drive_link
- Project FlowDemo Video : https://drive.google.com/file/d/1F6S9gYQX36KfqMZz-eqshRqB99zTDv7O/view?usp=sharing
- Project Code Repository :

  Doctor Appointment Booking Using MERN Source Code
- The source code for this project can be accessed and cloned from GitHub, providing a base structure and example code for further customization and understanding.
- GitHub Repository: Source Code

Video Tutorials :

- For a step by step guide on setting up and working with this project

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",
  "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"
  },
  "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": {
```

```
{
  "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"
  }
}
```

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

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:

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.

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:

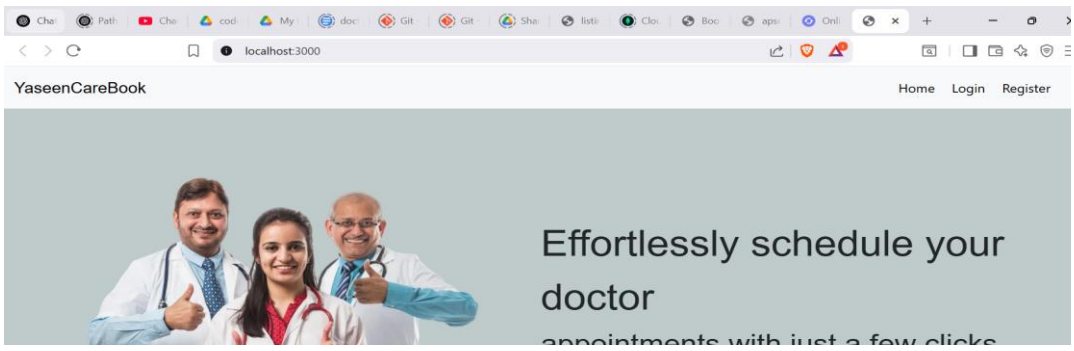
Testing the UI includes verifying the look and feel of each page—landing, login, registration, and dashboards for different user types.

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

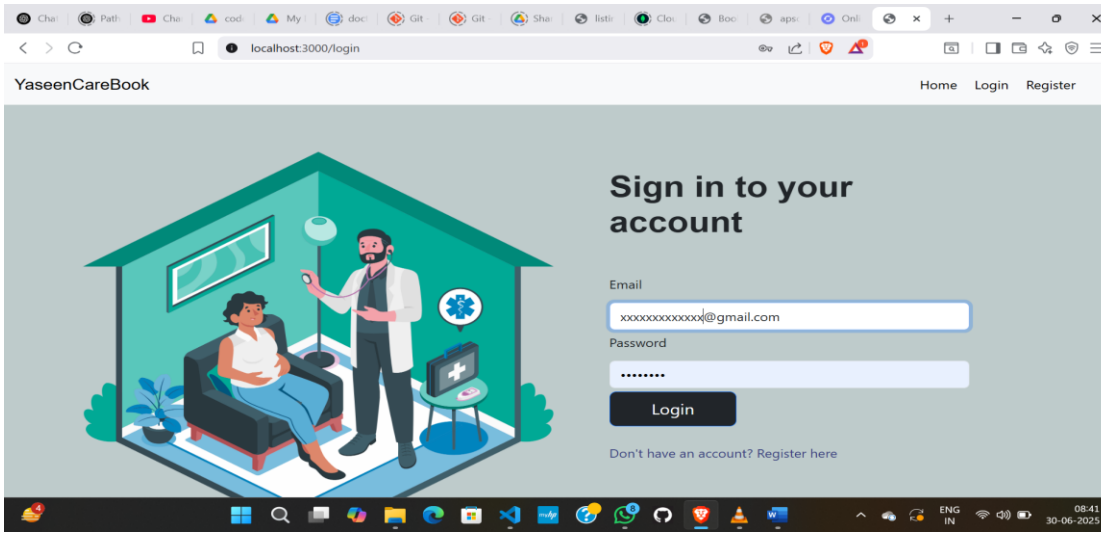
FINAL DEPLOYMENT:

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

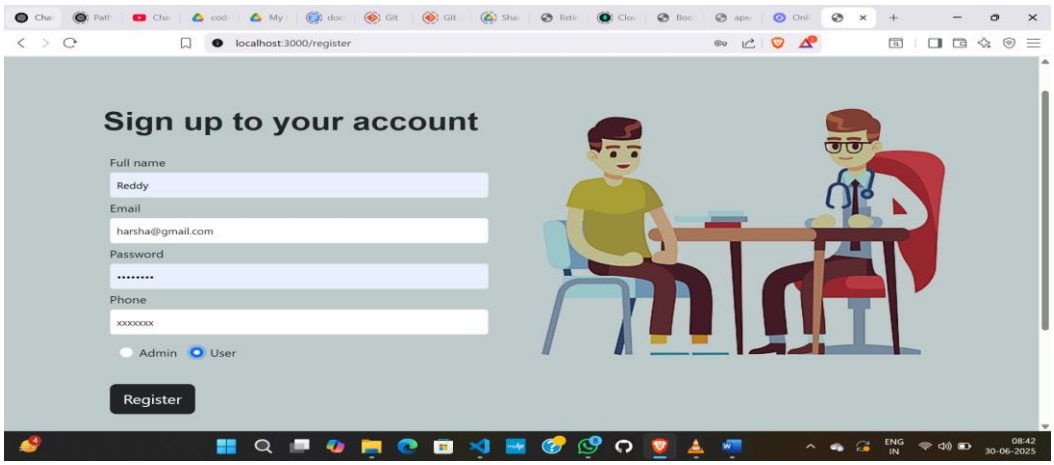
LANDING PAGE :



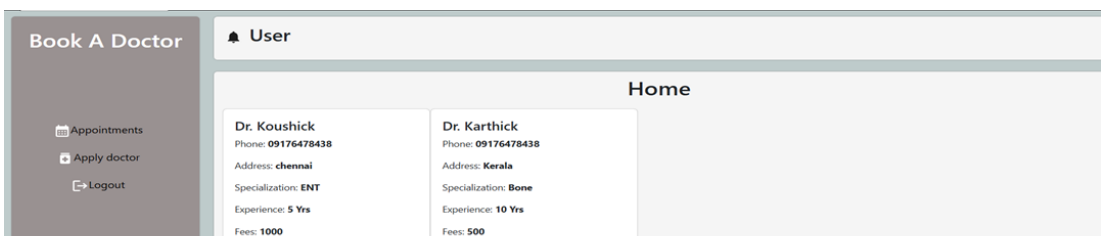
LOGIN PAGE :



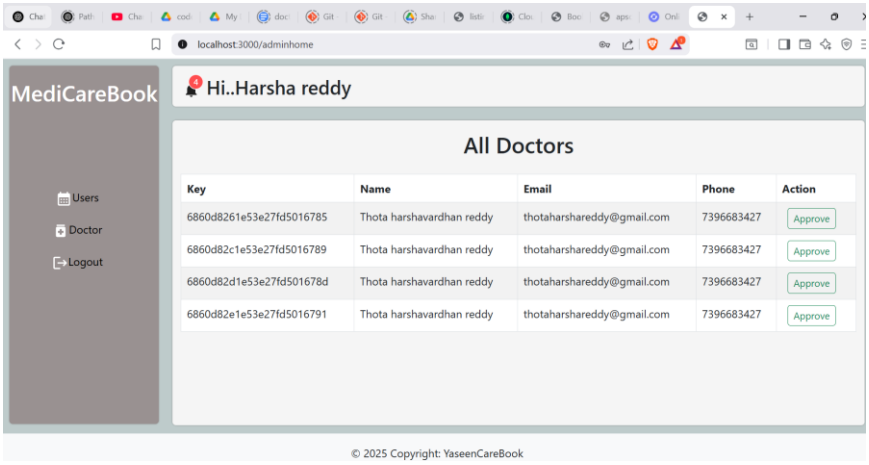
REGISTRATION PAGE :



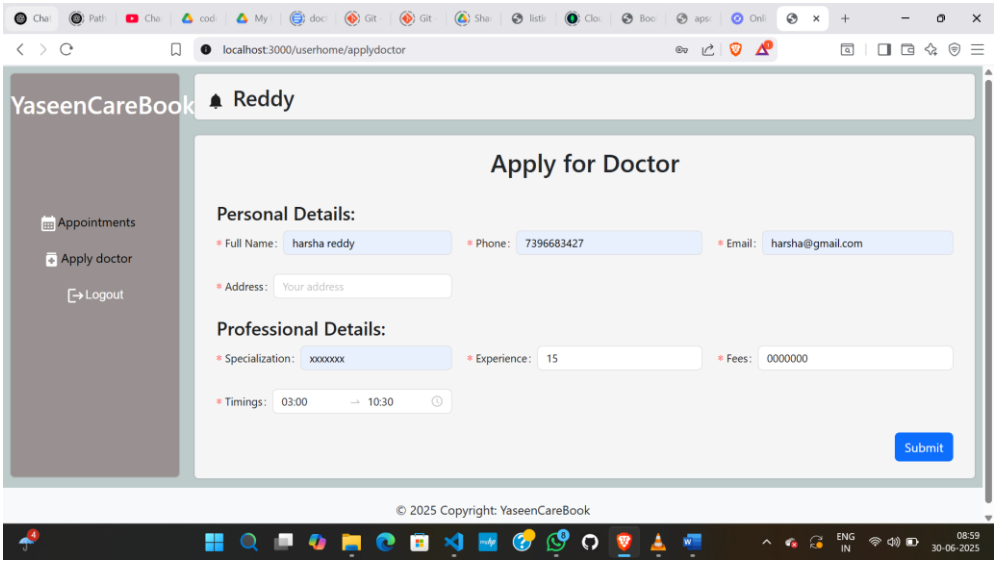
USER PAGE :



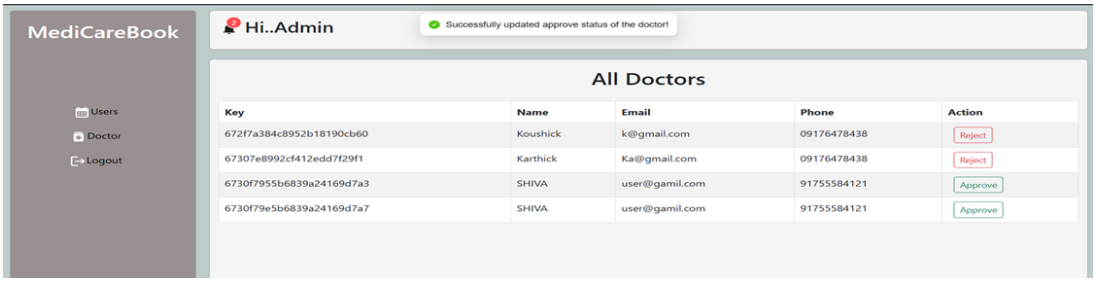
ADMIN PAGE :



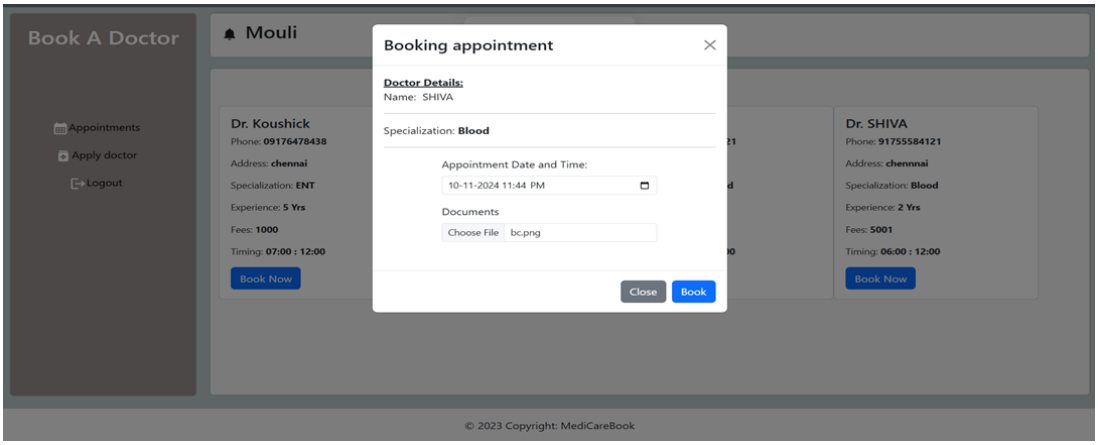
APPLY AS DOCTOR :



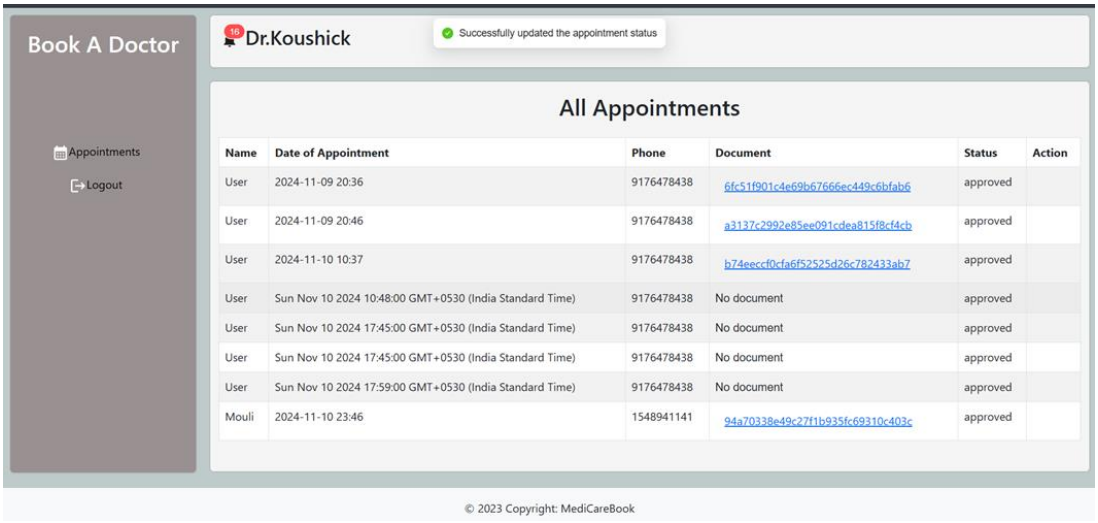
ADMIN APPROVE DOCTOR :



BOOK DOCTOR :



DOCTOR APPROVE USER APPOINTMENT :



ALL HISTORY :

CONCLUSION

This project outlines the development of a comprehensive appointment booking system with user roles for customers, doctors, and admin. Each milestone—setup, backend, database, frontend, and final implementation—forms a structured foundation for a scalable application. The integrated functionalities allow for seamless appointment management, user authentication, and role-specific operations.