

DocSpot USING MERN

Introduction:

The DocSpot App is a modern healthcare appointment system designed to bridge the gap between patients and healthcare providers through an intuitive digital platform. It enables users to seamlessly search, schedule, and manage medical consultations, making healthcare more accessible and organized.

Patients can explore doctors based on specialty, location, and real-time availability, ensuring personalized care. The platform supports appointment booking, schedule management, and secure document uploads. Doctors receive a dedicated dashboard to manage availability, view bookings, and maintain patient records, while administrators handle platform oversight, doctor verification, and issue resolution.

Technically, the app adopts a robust client-server architecture. The frontend, developed with React, Bootstrap, and Material UI, ensures a responsive and engaging user experience. The backend, built with Express.js and MongoDB, provides secure data handling and smooth API interactions. This architecture ensures the system is scalable, reliable, and tailored to meet the increasing demand for efficient healthcare services in a digital-first world.

KEY 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 or reject doctor profiles.

- Monitor platform activities and manage users.
- Enforce platform policies and resolve disputes.

ADMIN ANALYTICS MODULE

Admins can now access an **Analytics Dashboard** with real-time insights into platform performance:

- **Total Appointments Booked** (daily, weekly, monthly)
- **Active Doctors Count**
- **Registered Patients Count**
- **Peak Booking Hours**
- **Most Requested Specialties**
- **Appointment Completion Rate**
- **Cancellation & No-Show Trends**
- **Geographical Breakdown** of bookings (city-wise)

Built using MongoDB Aggregation Pipelines and displayed with charting libraries like **Chart.js** or **Recharts** in React for data visualization.

DESCRIPTION :

The **DocSpot** is a full-stack healthcare booking platform designed to simplify and modernize the appointment scheduling process between patients and medical professionals. This user-focused application enables patients to effortlessly register, create detailed profiles, and book appointments based on real-time availability, specialty, and location.

Patients can securely upload documents, receive automated reminders, and manage appointments through a clean and intuitive interface. For doctors, the platform offers a dedicated dashboard to control availability, confirm appointments, access patient information, and record post-visit notes.

Administrators play a key role in overseeing platform activity. They manage doctor approvals, monitor compliance, and ensure operational efficiency through system analytics and user oversight tools.

Technically, the app leverages React with Bootstrap and Material UI for a responsive and modern frontend, while Express.js and MongoDB handle backend logic and secure data storage. Communication between client and server is managed via Axios, and features like secure authentication are implemented using bcrypt. Moment.js is used to ensure accurate and flexible time-based scheduling.

The system is designed for scalability, security, and ease of use, making it a robust solution to meet the rising demand for digital healthcare accessibility and streamlined medical services.

SCENARIO-BASED CASE STUDY :

1. USER REGISTRATION :

Danush, a patient seeking a routine check-up, begins his journey by downloading and launching the DocSpot App. He proceeds to create a new account by entering his email address and setting a secure password. After successfully completing the registration process, Danush gains access to the app and is presented with the option to log in and begin exploring available doctors and services tailored to his healthcare needs.

2. BROWSING DOCTORS :

After logging in, Danush navigates to the doctor search section. He uses intuitive filters to find medical professionals based on specialization, location, and availability. The app displays a list of doctors along with their profiles, ratings, and consultation hours. Real-time availability ensures Danush can quickly identify open time slots and choose the right doctor that fits both his medical needs and schedule preferences.

3. BOOKING AN APPOINTMENT :

Danush selects Dr. Mahesh, a trusted family physician, and clicks the “Book Now” option. A user-friendly form appears, allowing him to choose his preferred date and time for the consultation. He also uploads relevant medical documents, such as health records and insurance information, to assist the doctor in pre-evaluation. After submitting the form, Danush receives an instant acknowledgment confirming that his appointment request has been received and is currently pending confirmation from the doctor.

4. APPOINTMENT CONFIRMATION :

After carefully reviewing Danush’s booking request and cross-checking his availability, Dr. Mahesh approves the appointment. The system automatically updates the appointment status to “Scheduled.” Danush is immediately notified with comprehensive details, including the confirmed date, time, and consultation location. These updates are delivered via both email and SMS to ensure prompt and effective communication.

5. APPOINTMENT MANAGEMENT :

As the appointment date nears, Danush can access and manage all his upcoming consultations through the app’s intuitive dashboard. He has full control to reschedule, cancel, or check the real-time status of each booking. The platform also offers direct communication channels, allowing Danush to message the doctor or reach out to support for any questions or assistance. This ensures a hassle-free and responsive experience throughout his healthcare journey.

6. ADMIN APPROVAL (BACKGROUND PROCESS) :

Behind the scenes, the app's admin actively reviews and verifies new doctor registrations. Dr. Mahesh, after submitting valid credentials and documentation, is successfully verified and approved. Only authenticated healthcare professionals are granted access to the platform, ensuring that patients connect with trusted providers. The admin plays a key role in maintaining regulatory compliance and upholding the platform’s standards for safety and credibility.

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 :

On the day of the appointment, Dr. Mahesh logs into his dashboard and reviews his scheduled appointments. He sees Danush’s appointment and confirms the time. Throughout the day, Dr. Mahesh manages other appointments, updates their statuses, and ensures patients are attended to efficiently.

9. APPOINTMENT CONSULTATION :

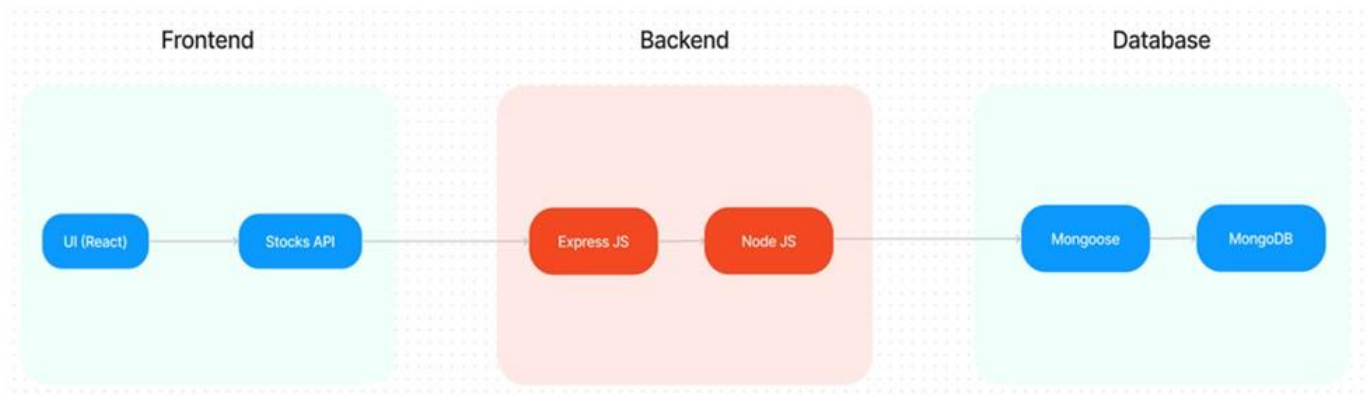
At the scheduled time, Danush visits Dr. Mahesh's office. During the consultation, Dr. Mahesh provides medical care, performs the check-up, and gives advice on maintaining good health. Danush's health concerns are addressed, and he feels assured that his routine check-up is complete.

10. POST-APPOINTMENT FOLLOW-UP :

After the consultation, Dr. Mahesh updates Danush's medical records within the app, noting any important observations, medications prescribed, or further treatments recommended. Danush receives a summary of his visit, including a prescription and any follow-up instructions via the app.

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 easy to maintain.

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 Dashboard:** Offers administrators the tools to manage doctor onboarding, configure platform settings, and monitor the system's overall performance and integrity.
- **Role-Based Access Control (RBAC):** Enforces secure access by assigning specific permissions based on user roles, ensuring that patients, doctors, and admins can only interact with the features relevant to their responsibilities. This helps maintain data confidentiality, system security, and regulatory compliance.

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.

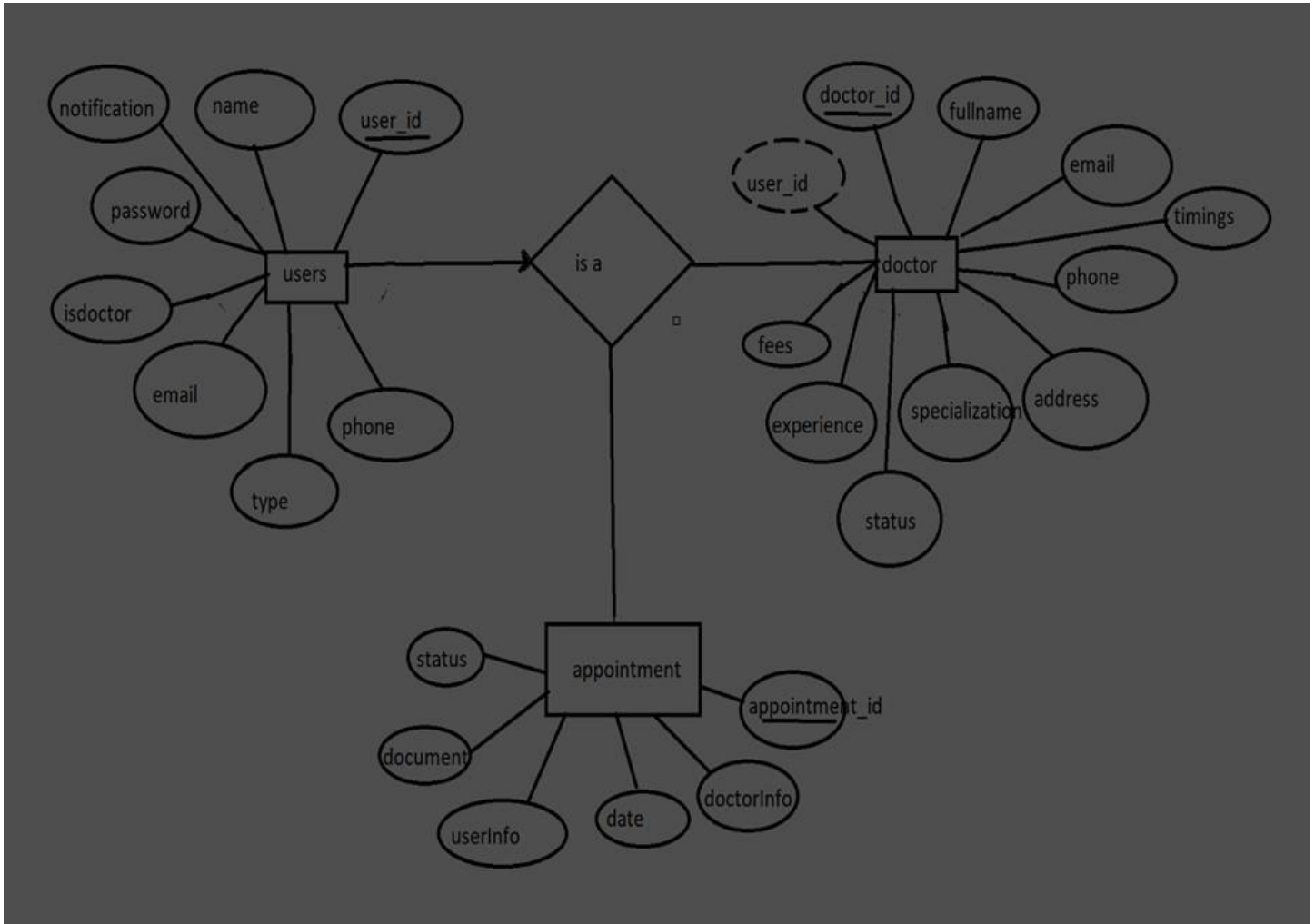
SECURITY FEATURES :

- **HTTPS:** The platform uses SSL/TLS encryption to secure data transmission between the client and server.
- **Data Encryption:** Sensitive user information, such as medical records, is encrypted both in transit and at rest, ensuring privacy and compliance with data protection regulations.
-

NOTIFICATIONS AND REMINDERS :

- **Email/SMS Integration:** Notifications for appointment confirmations, reminders, cancellations, and updates are sent to users via email or SMS, ensuring timely communication.

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 key referencing the Users collection),

document (medical records or other files), and status (e.g., pending, confirmed). This collection maintains the relationship between users and doctors for each appointment.

- The relationships are as follows: one User can be linked to one Doctor (one-to-one), a User can have multiple Appointments (one-to-many), and a Doctor can handle multiple Appointments (one-to-many). The foreign keys userID in the Doctors collection and doctorInfo and userInfo in the Appointments collection establish these connections, enabling the app to manage the interactions between patients and doctors effectively.

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.

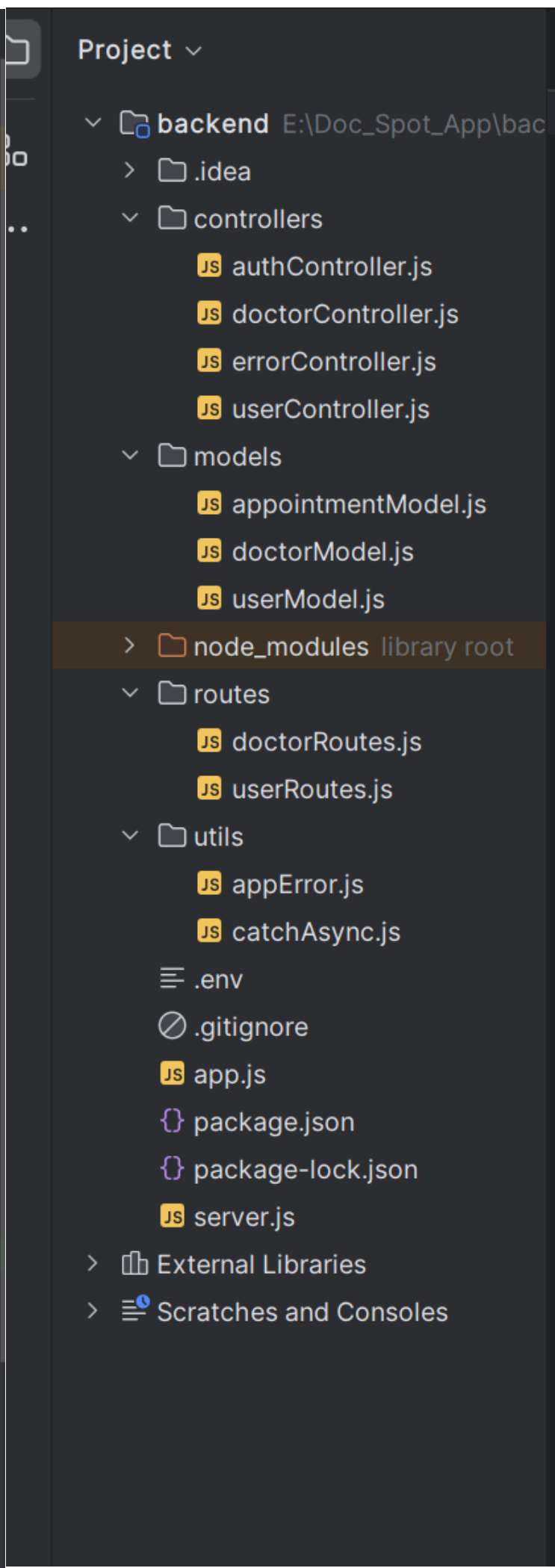
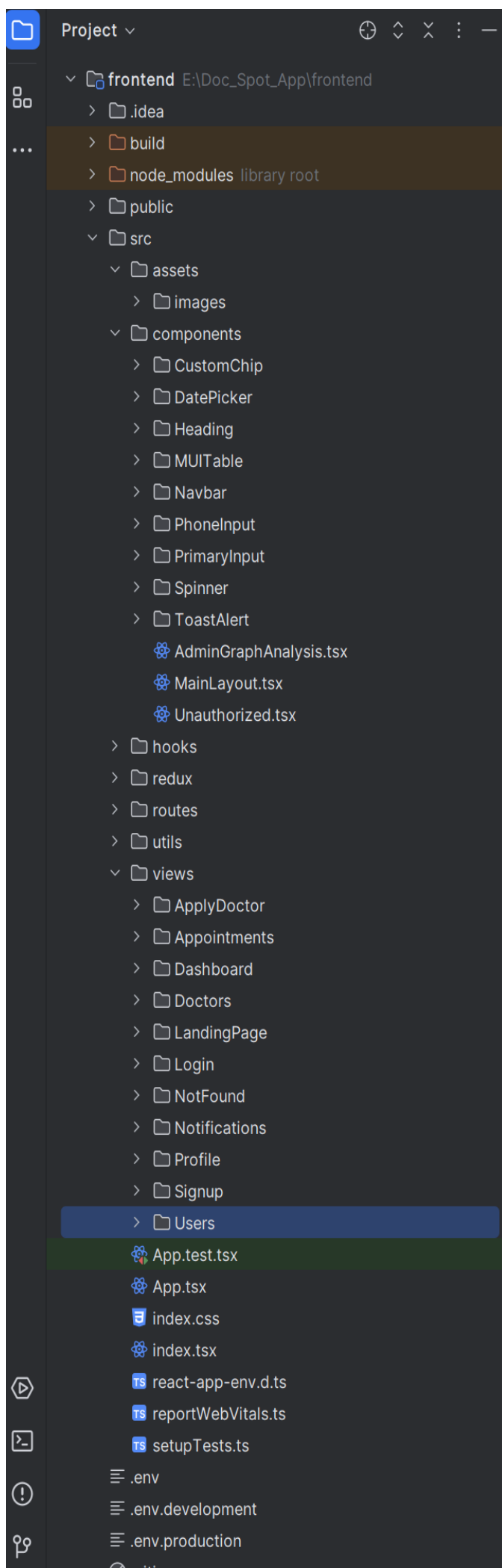
START THE DEVELOPMENT SERVER:

- 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: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.

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.

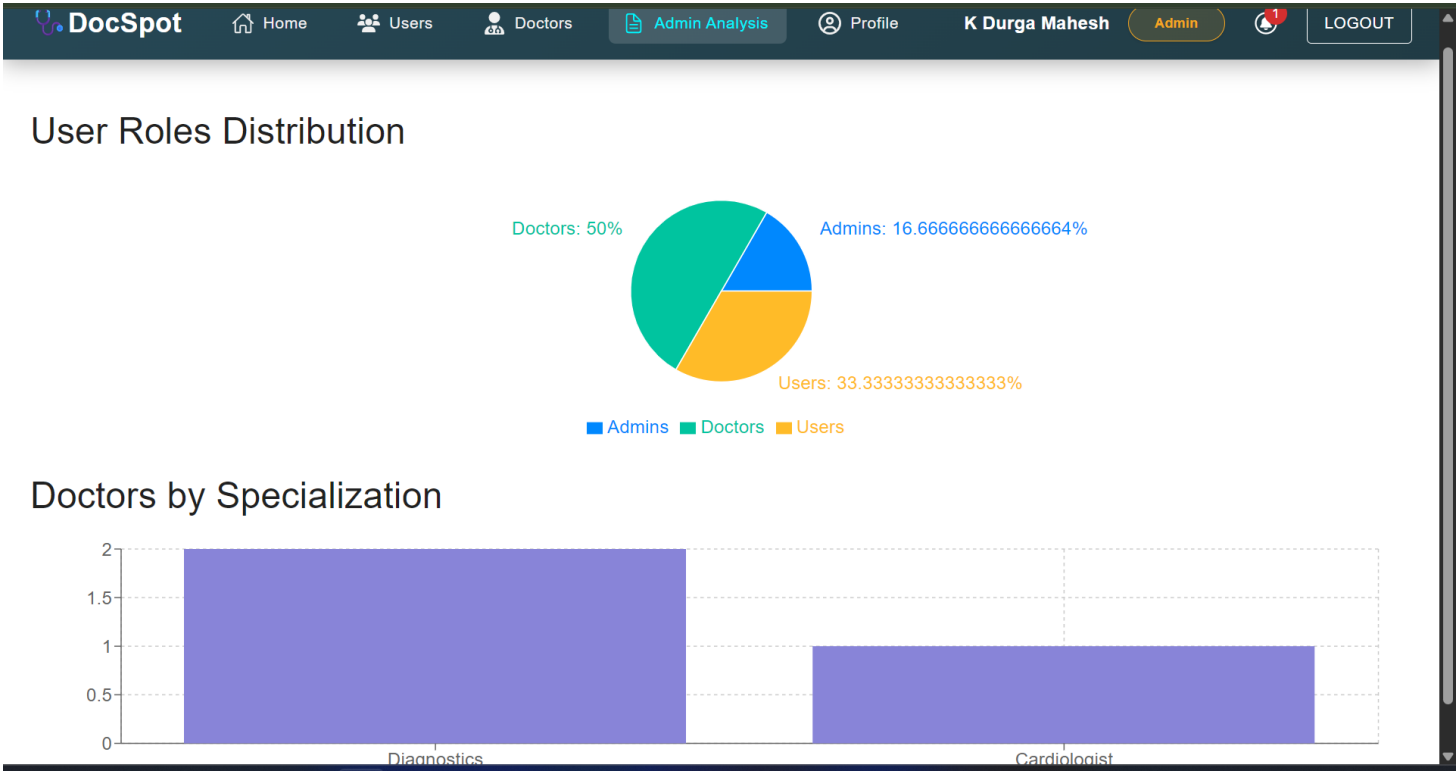
CUSTOMER/ORDINARY USER:

- 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.

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
Copy code
"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
- Copy code
- git init

Add your project files and commit them:

- bash • Copy code • 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
- 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.
- 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.

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

PROJECT FLOW :

Project Demo :

- Before diving into development, you can view a demo of the project to understand its functionality and user interactions.
- 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, follow the video tutorials below:
- Video Guide 1: Setting Up the Backend
- Video Guide 2: Configuring Frontend and API Endpoints
- Video Guide 3: Testing and Deployment
- These resources should give you a solid foundation and clear understanding of the project structure and workflow before development begins.

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": "server",
  "version": "1.0.0",
  "description": "DocSpot App",
  "main": "server.js",
  "engines": {
    "node": "18.14.2",
    "npm": "9.5.0"
  },
  "scripts": {
    "start": "node server.js",
    "server": "nodemon server.js",
    "build": "npm install --legacy-peer-deps && cd ../client && npm install --legacy-p",
  },
  "author": "K Durga Mahesh",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "colors": "^1.4.0",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "moment": "^2.29.4",
    "mongoose": "^8.0.0",
    "morgan": "^1.10.0",
    "recharts": "^3.1.0",
    "validator": "^13.11.0"
  }
},

```

```

},
  "scripts": {
    "start": "react-scripts --openssl-legacy-provider start --env=development",
    "build": "react-scripts --openssl-legacy-provider build --env=production",
    "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"
    ]
  }
}

```

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:

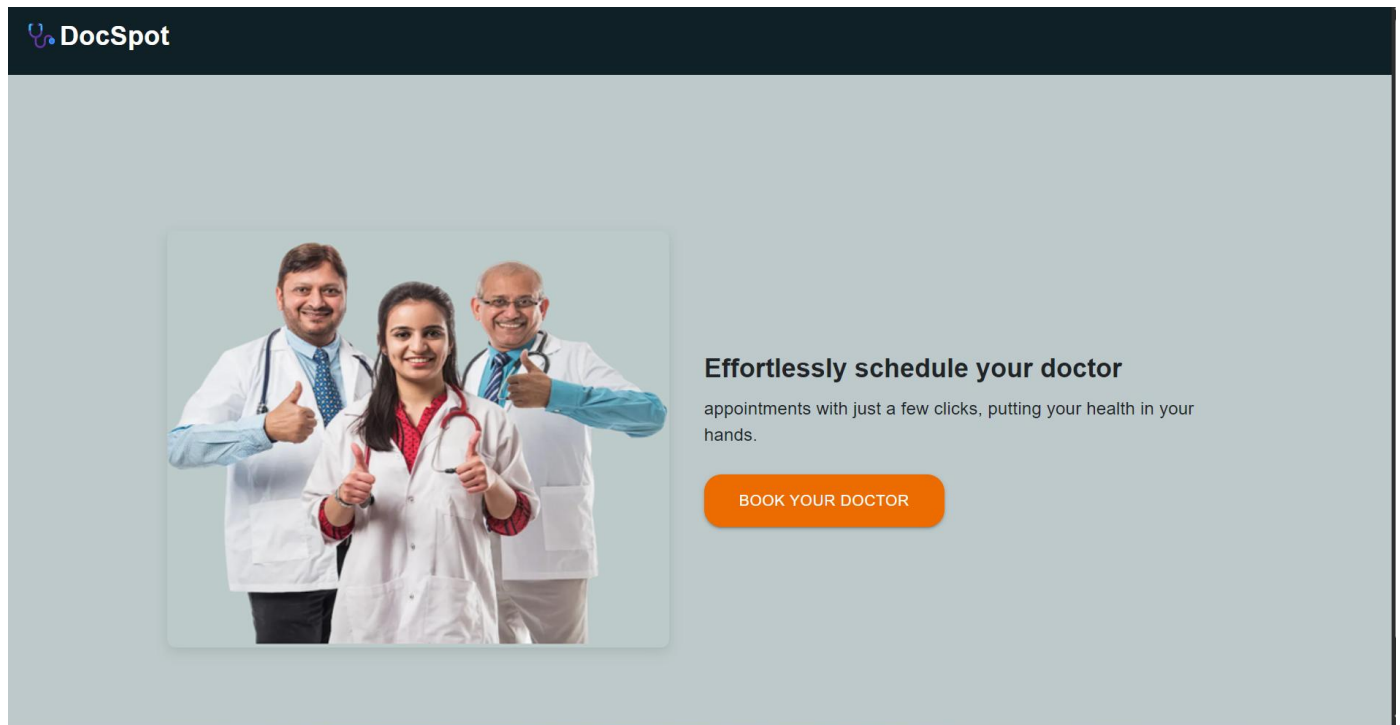
Testing the UI includes verifying the look and feel of each page—landing, login, registration, and dashboards for different user types.

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

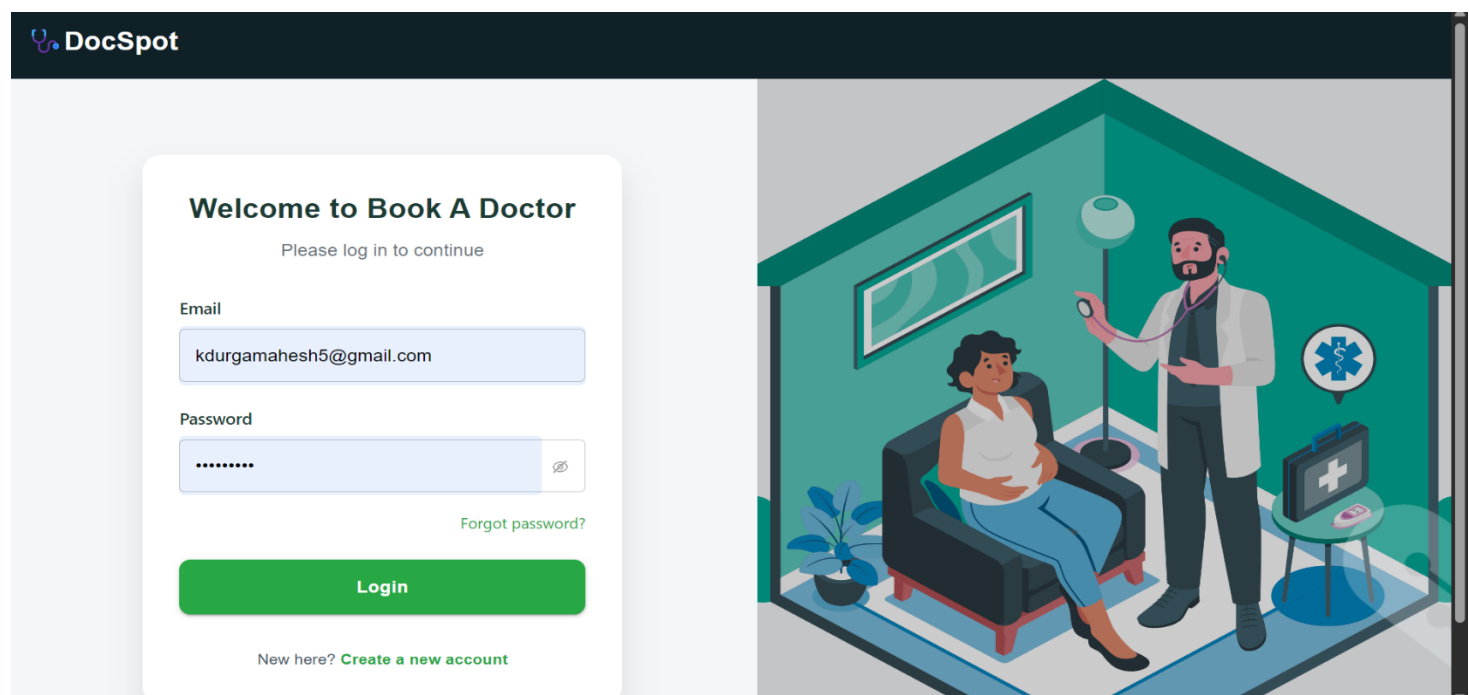
FINAL DEPLOYMENT:

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

LANDING PAGE :



LOGIN PAGE :



REGISTRATION PAGE :



DocSpot

[Home](#)[Appointments](#)[Profile](#)

Kammampalli Durga Mahesh

Doctor

LOGOUT

Available Doctors

Select Doctor to add Appointments

Dr. mahesh (Diagnostics)

Phone Number

092467 89087

Address

Hsno:-13-164 balaji nagar,veldurthi,kurnool,
veldurthi(mandal),kurnool(dist),ap

Fee Per Visit

20,000

Timings

9:00 AM to 6:00 PM

Dr. Kammampalli Durga Mahesh (Diagnostics)

Phone Number

091005 68715

Address

kurnool

Fee Per Visit

10,000

Timings

10:00 AM to 5:00 PM

Dr. hemanth (Cardiologist)

Phone Number

081990 19200

Address

kurnool

Fee Per Visit

500

Timings

10:00 AM to 4:00 PM


Select Doctor to add Appointments


Phone Number	092467 89087
Address	13-164-1 Balaji Nagar,veldurthi,kurnool, Hsno:-13-164 balaji nagar,veldurthi, veldurthi(mandal),kurnool(dist),ap
Fee Per Visit	20,000
Timings	9:00 AM to 6:00 PM


Phone Number	091005 68715
Address	kurnool
Fee Per Visit	10,000
Timings	10:00 AM to 5:00 PM


Phone Number	081990 19200
Address	kurnool
Fee Per Visit	500
Timings	10:00 AM to 4:00 PM


ADMIN PAGE :


 **DocSpot**

 Home


 Users


 Doctors

 Admin Analysis

 Profile

K Durga Mahesh


 Admin


 1

LOGOUT

Profile Details

Owner





K Durga Mahesh

091005 68714

Created At:

07/17/2025, 10:43 PM

ADMIN

K Durga Mahesh

091005 68714

Created At:

07/17/2025, 10:43 PM

APPLY AS DOCTOR :

DocSpot

Home

Appointments

Profile

Kammampalli Durga Mahesh

Doctor

1

LOGOUT

Profile

Basic Information

Dr.

Kammampalli Durga Mahesh

+91 91005-68715

www.dr.durga.com

kurnool

Professional Information

Diagnostics

3

10000

Start Time

End Time

10:00 AM

05:00 PM

UPDATE

ADMIN APPROVE DOCTOR :

DocSpot

Home

Users

Doctors

Admin Analysis

Profile

K Durga Mahesh

Admin

1

LOGOUT

Doctors

NAME	SPECIALTY	EMAIL	PHONE NUMBER	DATE	STATUS	ACTIONS
Dr. mahesh	Diagnostics	23at5a0502@gpcet.ac.in	092467 89087	07/17/2025, 10:51 PM	Approved	Block
Dr. Kammampalli Durga Mahesh	Diagnostics	kdurgamahesh5@gmail.com	091005 68715	07/19/2025, 11:53 PM	Approved	Block
Dr. hemanth	Cardiologist	hemanth@gmail.com	081990 19200	07/20/2025, 2:55 PM	Approved	Block

BOOK DOCTOR :

DocSpot

Home

Appointments

Profile

Kammampalli Durga Mahesh

Doctor

1

LOGOUT

Book Appointments

Timings

🕒 9:00 AM to 6:00 PM

Select Appointment Date

DD/MM/YYYY

Select Appointment Time

hh:mm (a)p.m

CHECK AVAILABILITY

Dr. mahesh (Diagnostics)

🕒 Consultation Time

🏢 Department

📅 Experience

💰 Fee Per Visit

📍 Location

30 Minutes

Diagnostics

3 Years

20,000

13-164-1 Balaji Nagar,veldurthi,kurnool, Hsno:-13-164 balaji nagar,veldurthi, veldurthi(mandal),kurnool(dist),ap

Booked Appointments

07/20/2025

Invalid Date - Invalid Date

DOCTOR APPROVE USER APPOINTMENT :

DocSpot

Home

Appointments

Profile

maresh

Doctor

4

LOGOUT

Appointments

ID	PATIENT	PHONE	DATE	STATUS	ACTIONS
687bd50b1341c4d394a32394	K Durga Mahesh	091005 68714	07/20/2025 Invalid Date	Approved	✓
687be23e1341c4d394a323f6	K Durga Mahesh	091005 68714	07/24/2025 Invalid Date	Pending	Approve Reject
687cb538d1950a7425b80524	suresh	099119 92299	07/22/2025 Invalid Date	Approved	✓

CONCLUSION

This project presents the structured development of a full-featured doctor appointment booking system with clearly defined user roles for patients, doctors, and administrators. From initial setup to backend integration, database design, frontend development, and final deployment, each phase contributes to a scalable and maintainable application. The system ensures secure user authentication, efficient appointment scheduling, and robust role-based access. Overall, it delivers a streamlined healthcare experience tailored to the unique needs of each user group.