DOC SPOT APPOINTMENT

DocSpot is a modern, intuitive platform developed to simplify how patients connect with medical professionals. By integrating intelligent features into a single system, DocSpot enhances the healthcare appointment experience for patients, doctors, and system administrators. This platform enables real-time doctor discovery, appointment booking accessible via an easy-to-navigate interface.

The system empowers patients to locate healthcare professionals by filtering through various criteria such as medical specialization, proximity, consultation availability, and user ratings. After identifying a suitable doctor, users can schedule visits, adjust bookings.

For doctors, DocSpot provides a personalized portal where they can manage patient bookings.

Admins play a crucial role in maintaining the integrity of the platform by moderating activity, resolving user issues, and ensuring policy compliance across the application.

Technologically, DocSpot is architected using a modular client-server design. The frontend leverages UI libraries such as React and Bootstrap, ensuring a dynamic and responsive user experience across all devices. On the backend, technologies like Node.js , while MongoDB ensures efficient and secure data storage. Features like encrypted authentication, role-based access control, and error handling contribute to system reliability and security.

Ultimately, DocSpot transforms the way people access healthcare by offering a digital-first solution that reduces wait times, eliminates scheduling complexities, and fosters improved communication between patients and healthcare providers.

KEY FEATURES

PATIENT REGISTRATION & PROFILE CREATION

- Secure Sign-UP using email and password authentication.
- Profile Creation of the user with therir Name

DOCTOR BROWSING & FILTERING

- Allow users to search doctors based on specialty
- Availability updates of doctor that can user/patient can book based on the timings

APPOINTMENT BOOKING & MANAGEMENT

- Booking interface where patients choose appointment dates, times and basic details of the person
- Notification to the user dashbord that the appointmnet is approved or not

DOCTOR'S DASHBOARD

- Doctors can manage timings and booking avilability and can approve or reject the perosn

ADMIN CONTROLS & APPROVAL

- Admin manage both the doctor and patient page

DESCRIPTION

The DOC is a user-friendly platform designed to make healthcare appointment booking easy and efficient. This connects patients and healthcare providers 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 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 modern frontend, the app also uses React.js 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 witht the platform supports the growing demand for accessible healthcare options, providing patients with convenient, reliable access to healthcare while helping providers manage their schedules effectively.

SCENARIO-BASED CASE STUDY

1. USER REGISTRATION

John, a patient in need of a routine check-up, opens the DOC platfrom . He starts by registering as a patient, providing his email address and creating a password to ensure a secure login. Once the registration process is complete, John is welcomed to the platform 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 appointments. The platform offers various filters for him to search for healthcare providers based on criteria such as specialty, location, and availability. John filters the list to find a family physician in his area, available for a routine check-up.

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. 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. John receives a notification that the doctor is approved the appointment

5. 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.

6. 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.

7. DOCTOR'S APPOINTMENT MANAGEMENT

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

TECHNICAL ARCHITECTURE

The DocSpot Appointment System is built on a scalable and secure client-server architecture designed to deliver a seamless experience for patients, doctors, and administrators. The frontend is developed using React, enhanced with styling frameworks such as UI and Bootstrap. Communication between the client and server is efficiently handled through Axios, ensuring smooth data exchange.

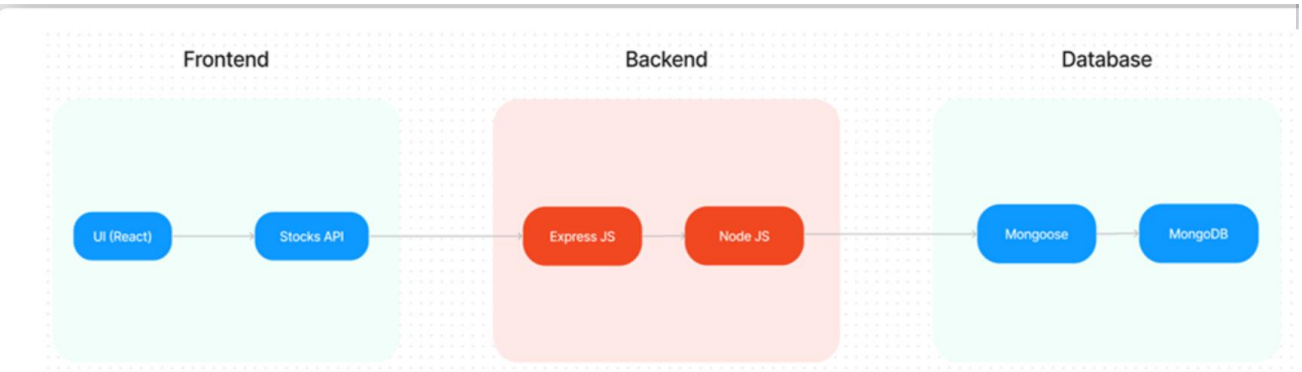
On the backend, Node.js combined with Express.js powers the server-side logic, offering fast and efficient handling of routes, middleware, and RESTful APIs. MongoDB, a NoSQL database, serves as the primary data store, managing structured collections for users, appointments, doctor profiles, and digital health records.

For authentication and user session management, the system uses JSON Web Tokens (JWT), providing stateless and secure login mechanisms. Password security is reinforced with bcrypt, which hashes sensitive user credentials. Day.js (or equivalent modern alternatives to Moment.js) ensures precise time tracking and appointment slot management across different time zones.

The administrative module includes tools for managing user roles, verifying doctor credentials, monitoring booking trends, and resolving operational issues. Role-Based Access Control (RBAC) governs feature accessibility depending on whether the user is a patient, doctor, or admin, maintaining strict access boundaries.

To support performance and scalability, the architecture integrates caching strategies (such as in-memory caching) and is designed to accommodate load balancing for handling increasing user demands. Data integrity and uptime are maintained with proper API error handling, logging, and asynchronous operations.

DocSpot's architecture ensures that the platform remains highly available, secure, and flexible, adapting to the evolving demands of digital healthcare services.



FRONTEND TECHNOLOGIES

● Bootstrap and Material UI: Provide a responsive and modern UI that adapts to various devices, ensuring a user-friendly experience.

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 Interface: Provides functionality for platform admins to approve doctor registrations, manage platform settings, and oversee day-to-day operations.

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.

TIME MANAGEMENT AND SCHEDULING

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

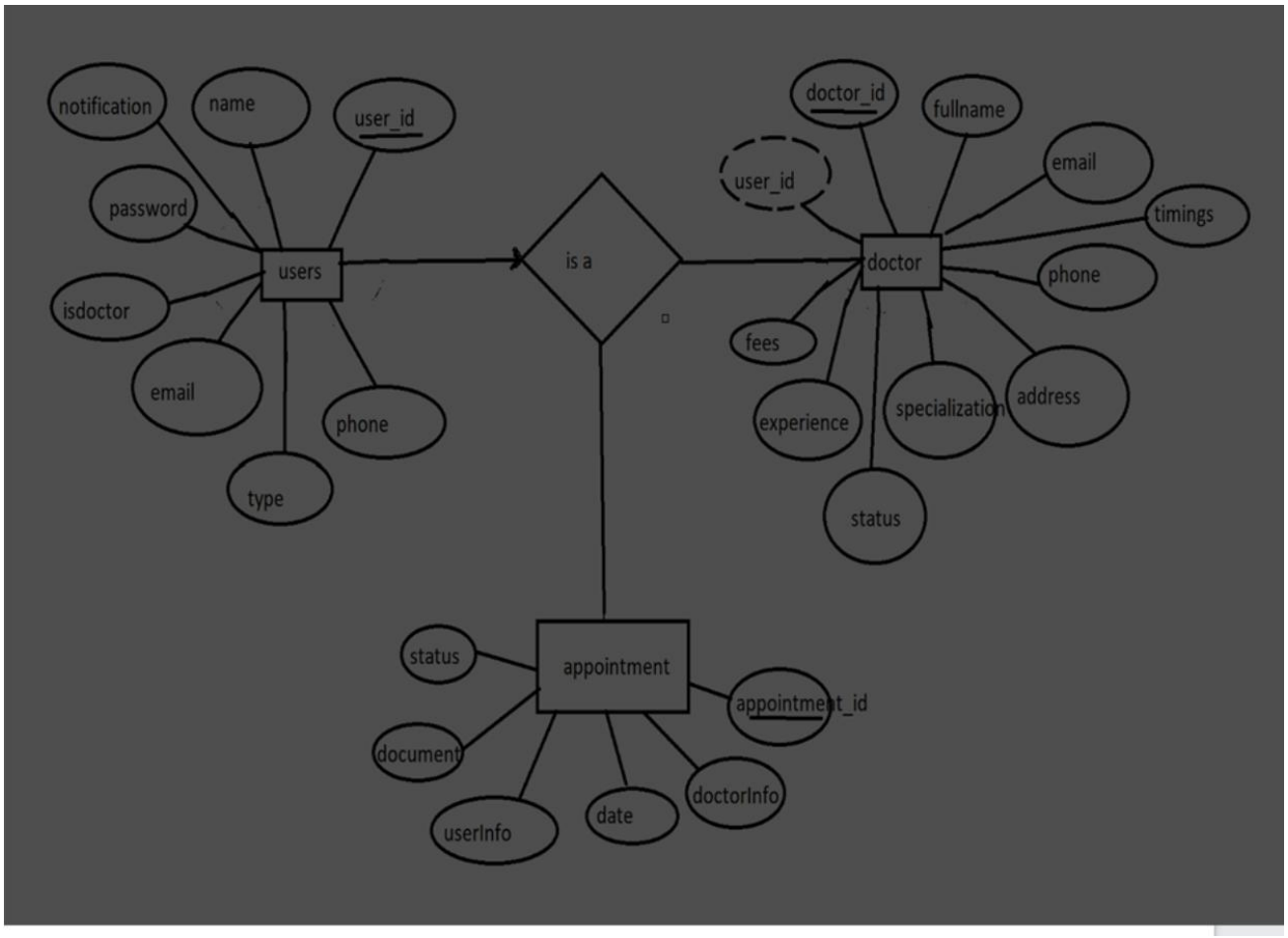
SECURITY FEATURES

● 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

- Get Notification via theo notification page when user login

ER Diagram



ER Diagram Overview

The Entity-Relationship (ER) model for the DOC application consists of three primary entities: Users,Doctors, and Appointments. Each entity contains distinct attributes and shares defined relationships to facilitate the application's core functionalities.

1**. Users**

This entity captures general user information and includes the following fields:

* `_id`: Unique identifier for the user

* `name`: Full name of the user

* `email`: Contact email

* `notification`: Notification preferences/settings

* `password`: Encrypted password

* `isdoctor`: Boolean flag indicating if the user is a doctor

* `type`: Role of the user (e.g., patient, admin)

* `phone`: Contact number

2. Doctors

Each record in this entity represents a doctor and contains:

* `_id`: Unique identifier

* `userID`: References the user in the Users entity (foreign key)

* `fullname`, `email`, `phone`, `address`: Contact and identity details

* `specialisation`: Medical field of expertise

* `timings`: Available consultation hours

* `experience`: Years of practice

* `fees`: Consultation charges

* `status`: Approval or verification status

3. Appointments

This entity manages the interactions between users and doctors by storing:

* `_id`: Appointment ID

* `doctorInfo`: References the doctor (foreign key to Doctors)

* `userInfo`: References the user who booked the appointment

* `date`: Appointment date and time

* `document`: Attached medical files or notes

* `status`: Current state (e.g., pending, confirmed, cancelled)

Entity Relationships

User–Doctor: A one-to-one mapping where a doctor has a corresponding user account.

User–Appointments: A one-to-many relationship, as users (patients) can book several appointments.

Doctor–Appointments: A one-to-many relationship since each doctor can attend multiple appointments.

The system uses foreign keys such as `userID`, `doctorInfo`, and `userInfo` to maintain integrity and establish these associations.

PRE-REQUISITES

To get started with the Doctor Appointment Booking application, several tools and technologies need to be in place. These include server-side, client-side, and database dependencies that are essential for both development and execution.

NODE.JS AND NPM

Node.js serves as the runtime environment for executing JavaScript on the server side, making it ideal for building scalable backend services. It comes bundled with npm (Node Package Manager), which is used to manage dependencies and install required packages. You can download Node.js from its official website and follow the installation guide specific to your operating system. Once installed, initialize your project with npm init, which generates a package.json file to keep track of project configurations and dependencies.

EXPRESS.JS

Express.js is a lightweight and powerful framework built on Node.js, designed for building web applications and APIs with ease. It simplifies the development process with features like routing and middleware support. To include Express in your backend, run npm install express, which will allow you to define endpoints and manage HTTP requests efficiently.

MONGODB

MongoDB is a document-oriented NoSQL database, particularly suitable for applications dealing with flexible data structures such as user accounts, doctor profiles, and scheduled appointments. To use MongoDB, download and install it on your system or use a cloud-based instance. Refer to the official installation instructions for setup guidance.

MOMENT.JS

Moment.js is a utility library used to handle date and time operations in JavaScript. It offers various functions to parse, validate, manipulate, and display dates, making it extremely useful for features like appointment scheduling. You can learn more about its usage from the Moment.js documentation and install it in your project via npm.

REACT.JS

React.js is a front-end JavaScript library widely used for building user interfaces, especially for single-page applications. It helps in creating dynamic and interactive components. To begin with React, refer to its official documentation and create your app using npx create-react-app or another preferred method. React will form the core of the application's frontend.

ANTD (ANT DESIGN)

Ant Design is a comprehensive UI library tailored for React applications. It includes a wide range of customizable components like tables, modals, forms, and buttons, which help in creating a polished and responsive interface. For better UI consistency and user experience, install Ant Design and explore its React-based documentation.

HTML, CSS, AND JAVASCRIPT

A working knowledge of HTML, CSS, and JavaScript is essential for styling, layout design, and adding interactivity to the front-end of the application. These core technologies provide the foundation upon which React and other libraries operate.

DATABASE CONNECTIVITY (MONGOOSE)

Mongoose is an Object Document Mapper (ODM) for Node.js that simplifies communication with MongoDB. It offers schema-based solutions to model your application data and supports seamless CRUD operations. Use Mongoose to link your backend logic to MongoDB, enabling smooth database interactions.

FRONT-END FRAMEWORKS AND LIBRARIES

The frontend of the application relies heavily on React.js for rendering UI components and managing user interactions. In addition to Ant Design, you may incorporate frameworks like Material UI or Bootstrap to enhance design flexibility and responsiveness. These libraries help build professional-looking interfaces without starting from scratch.

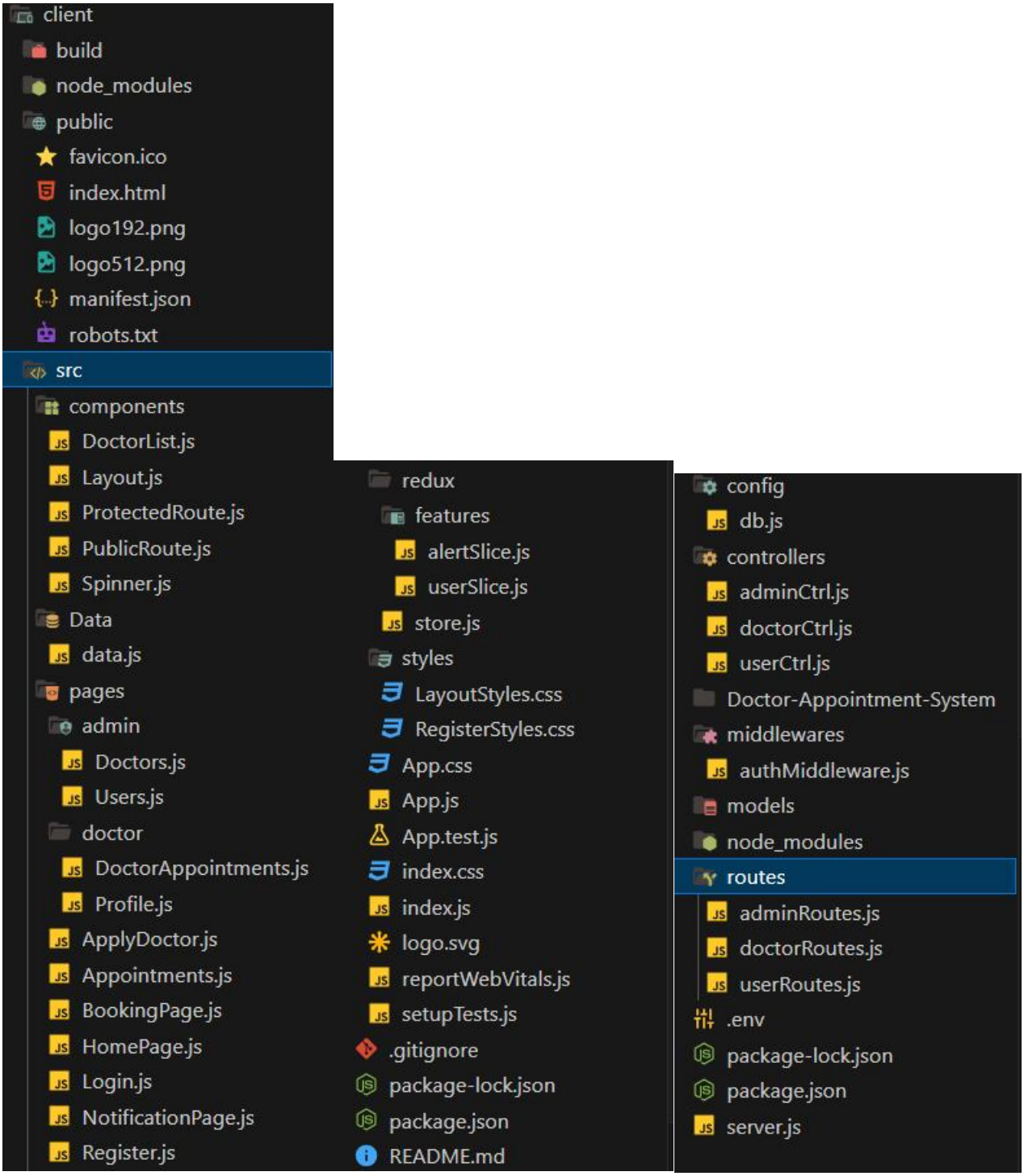
SETUP AND INSTALLATION INSTRUCTIONS

To run the project locally, start by cloning the repository from GitHub or downloading the source code. Navigate into the frontend and backend directories separately and execute npm install in each to install all necessary dependencies. After installation, you can start the development servers. Typically, the frontend will launch on http://localhost:3000, while the backend will be accessible on http://localhost:8001, unless a different port is configured.

ACCESS THE APPLICATION

Once both servers are up and running, you can interact with the application by visiting http://localhost:3000 in your browser for the frontend interface. Backend APIs will be available at http://localhost:8001, serving all the essential data operations like user registration, doctor listings, and appointment scheduling.

PROJECT STRUCTURE



FRONTEND PART

The frontend of the Doctor Appointment Booking application is built using React.js and is responsible for the visual interface and user interactions. It includes several organized components designed for specific actions such as displaying available doctors, booking appointments, managing notifications, and navigating user dashboards. The routing system allows smooth navigation between different pages like the booking form, appointment history, and user dashboards. State management is implemented to maintain user sessions, doctor profiles, and real-time appointment statuses. For visual consistency and responsiveness, the frontend uses a combination of custom CSS and UI libraries like Ant Design to style its components effectively.

BACKEND PART

The backend is developed using Node.js and Express.js and is responsible for all server-side logic and data processing. It manages API endpoints that handle actions such as user registration, appointment scheduling, and administrative controls. The backend also includes database models for Users, Doctors, and Appointments, defined using Mongoose to interact with MongoDB. Authentication and authorization mechanisms ensure that only verified users can access certain features, and user roles (admin, doctor, customer) are enforced. Additionally, the backend supports a notification system that alerts users of changes in appointment status, and it includes admin-specific routes for monitoring platform activities and managing application-level policies.

APPLICATION FLOW

The application supports three distinct user roles: Customer, Doctor, and Admin, each with specific capabilities managed through API routes and interface components. Customers begin by creating an account and logging in using their credentials. Once logged in, they can browse available doctors, book appointments by selecting a preferred date and uploading necessary documents, and view the status of their bookings. They can also cancel existing appointments directly from their booking history page. Admins oversee the entire platform, managing both doctors and customers. They review and approve doctor registrations, enforce policies, and ensure a secure and efficient system. Doctors, on the other hand, must receive approval from an admin before gaining access to the system. Once verified, they can manage incoming appointments, confirm or reject bookings, and respond to changes with real-time notifications.

SETUP & CONFIGURATION

To get the Doctor Appointment Webpage up and running, both the frontend and backend need to be configured properly. Begin by cloning or downloading the project repository. Then, navigate to the frontend and backend directories individually to install the required dependencies using npm install. Once the dependencies are in place, start the development servers: the frontend typically runs on http://localhost:3000 while the backend operates on http://localhost:8001, unless configured otherwise. After both servers are active, users can access the full application through their browser, enabling a seamless experience for patients, doctors, and admins alike.

FRONTEND CONFIGURATION

INSTALLATION

Install Dependencies

● Use npm (Node Package Manager) to install the necessary dependencies:

● 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

● 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 mondodb url

● Install Dependencies:

● Install the necessary backend dependencies using npm

● 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

MongoDB

● bash

● Copy code

● MONGO_URI=mongodb://localhost:27017/doctorapp

● 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=3000

● 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"

● 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.

● Installation Commands

Run these commands in the backend folder

● bash

● Copy code

● npm init -y

● npm install cors bcryptjs express dotenv mongoose multer jsonwebtoken

● npm install --save-dev nodemon

FRONTEND SETUP

● React Project Initialization

● Navigate to the frontend folder and initialise a new React application:

● plaintext

● Copy code

● frontend/

├── public/

├── src/

│

├── components/

│ ├── pages/

│ ├── services/

│ └── App.js

├── .env

└── package.json

● Setting Up the Frontend Project

● In the frontend folder, run the following commands to set up and install any initial dependencies:

● bash

MILESTONE 1: PROJECT SETUP AND CONFIGURATION

```
{
  "name": "doc-appointment-system",
  "version": "1.0.0",
  "main": "server.js",
  ▷Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "server": "nodemon server.js",
    "client": "npm start --prefix ./client ",
    "dev": "concurrently \"npm start\" \"npm run client\""
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "antd": "^5.26.1",
    "bcryptjs": "^3.0.2",
    "colors": "^1.4.0",
    "concurrently": "^9.1.2",
    "dotenv": "^16.5.0",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "moment": "^2.30.1",
    "mongoose": "^8.16.0",
    "morgan": "^1.10.0",
    "nodemon": "^3.1.10"
  }
}
```

PROJECT FOLDERS

The project is organized into two main folders to separate concerns and maintain a modular structure. The Frontend folder contains all the code related to the user interface, developed using JavaScript along with libraries and frameworks such as React, Material UI, and Bootstrap. This separation ensures that all UI components and visual logic are encapsulated in one place, making the interface easier to develop and maintain. On the other hand, the Backend folder is dedicated to handling server-side operations, including API route definitions, user authentication, and database interactions. Built using Node.js and Express.js, this folder manages all backend logic independently from the frontend, allowing updates or fixes in one part of the application without disrupting the other.

Library and Tool Installation

To begin the project, essential backend and frontend libraries were carefully selected to support robust functionality and seamless user experiences. On the backend, Node.js was used as the JavaScript runtime environment to execute code outside the browser, offering fast and scalable server-side execution. For data storage, MongoDB was chosen due to its flexibility as a NoSQL database, allowing for schema-less and dynamic data handling, which is especially useful in applications that require frequent updates and handle diverse data formats. To ensure user authentication and data security, Bcrypt was integrated to hash and encrypt passwords before storage, making them highly resistant to breaches. Additionally, body-parser middleware was included to handle various request body formats like JSON, enabling smooth parsing and manipulation of incoming data.

On the frontend, React.js was adopted to build a component-based interface, allowing for the development of reusable, modular UI elements that streamline design consistency and efficiency. To enhance visual design and responsiveness, Material UI and Bootstrap were employed, offering pre-built UI components and grid systems for cohesive styling across different devices. For HTTP communications, Axios served as the bridge between the client and server, making it easy to send and receive asynchronous API requests.

Milestone 2: Backend Development

Backend development formed the core of the application's logic and infrastructure, focusing on security, scalability, and maintainability. The foundation was laid using Express.js, which functioned as the central framework for handling server requests and responses. Express streamlined the routing mechanism, efficiently managing client interactions with various endpoints. Middleware tools like body-parser were used to handle incoming data formats, while cors was configured to allow cross-origin access, enabling the frontend and backend to operate in tandem across different domains.

API endpoints were organized according to their purpose, such as authentication, appointment management, and complaint handling. Each functional domain was grouped into its own route file (e.g., auth.js), promoting modularity and ease of maintenance. Route handlers controlled the flow of data, whether for creating, fetching, updating, or deleting resources, ensuring a logical and consistent interaction model between frontend inputs and backend responses.

The backend leveraged Mongoose to define data models and schemas for structured interaction with MongoDB. The User schema encompassed user credentials, roles, and personal details, facilitating role-based access management. Similarly, models like Appointment and Complaint were structured to represent relational data and track interactions between users and the system. These models enabled clean CRUD operations, allowing data to be manipulated and queried in a predictable manner.

Authentication was secured using JWT (JSON Web Tokens), which allowed users to log in and access protected resources without the need to store session data on the server. Tokens were embedded in request headers, validating each transaction securely. Administrative capabilities were built into the system, granting elevated privileges to admins who managed user registrations, doctor approvals, and complaints. The transactional system allowed real-time booking, cancellation, and historical tracking of appointments, creating a dynamic and responsive backend environment. To enhance stability, dedicated error-handling middleware was added to catch runtime issues and return appropriate HTTP status codes and descriptive messages for debugging and user feedback.

Milestone 3: Database Development

The database was designed using MongoDB, a non-relational database well-suited for the evolving structure of the application's data. Its document-oriented architecture allowed developers to store and retrieve data in a flexible manner, avoiding the rigid constraints of SQL schemas. Multiple schemas were created using Mongoose to represent real-world entities and their relationships.

The User schema included fields such as name, email, password, and user role, ensuring clear differentiation between administrators, doctors, and regular users. Similarly, the Complaint schema was developed to handle customer complaints, tracking complaint status, assigned agents, and progress updates. A parallel schema, AssignedComplaint, was used to represent complaints allocated to specific agents for resolution. Additionally, a ChatWindow schema was introduced to manage real-time communications between users and support staff, with messages stored and retrieved based on complaint IDs, facilitating streamlined and relevant conversation threads.

Each schema was tied to its own MongoDB collection, such as users, complaints, and messages. This structure allowed the database to efficiently store and retrieve large volumes of data, with the flexibility to scale as user activity increased.

Milestone 4: Frontend Development

The frontend of the application was developed using React.js, chosen for its efficiency in building dynamic, component-driven user interfaces. A clear folder structure was established, dividing the codebase into components, services, pages, and utilities to keep the project organized and scalable. Essential libraries such as Material UI and Bootstrap were incorporated to maintain a visually consistent and responsive design across different screens and devices.

Reusable components played a crucial role in development, ensuring that common interface elements like input forms, buttons, dashboards, and modals were consistently implemented throughout the application. This modular approach reduced development effort and simplified debugging and updates. Custom styling was applied alongside Bootstrap classes to create a modern, intuitive look that enhanced the overall user experience.

To facilitate data communication with the backend, Axios was used to send HTTP requests, enabling the frontend to fetch or update data without reloading the page. React’s state management system bound data to components, automatically reflecting updates in real time as users interacted with the interface.

Milestone 5: Project Implementation and Testing

After completing the development phases, the application underwent a comprehensive testing phase to verify that all modules worked as expected. Functional testing was conducted on key user flows such as login, registration, appointment booking, complaint submission, and cancellation. The goal was to ensure smooth transitions and accurate data flow between the frontend, backend, and database.

The user interface was carefully reviewed to verify its responsiveness and consistency. Each screen—whether it was the landing page, login portal, or dashboard—was tested for visual clarity and interaction logic. Cross-

device and cross-browser testing ensured compatibility and performance across platforms, enhancing usability for a broader audience.

Following successful testing, the project proceeded to deployment on a production server, making the system accessible to end-users. Final configurations ensured that security, performance, and stability were prioritized, marking the transition from development to live operation.

Conclusion

This project showcases the development of a full-stack appointment booking system with distinct roles for customers, doctors, and administrators. From backend setup and secure user authentication to frontend design and responsive UI, each phase contributed to a scalable and functional application.

The system enables seamless appointment management, complaint handling, and real-time user interactions. Built with modern technologies like Node.js, MongoDB, React.js, and JWT, it ensures security, usability, and future scalability. Overall, the project highlights efficient integration of frontend and backend systems to deliver a user-friendly and maintainable solution.