BOOK A DOCTOR USING MERN

A PROJECT REPORT

Submitted by

RAJ KUMAR N - 310821104076
RAJARAJESHWARAN N - 310821104075
VISHWANATH R R - 310821104110
VIJAYAKUMAR S - 310821104116

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

JEPPIAAR ENGINEERING COLLEGE, CHENNAI

ANNA UNIVERSITY:: CHENNAI 600 025

NOVEMBER 2024

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **" BOOK A DOCTOR USING MERN "** is the bonafide work of " **RAJ KUMAR N (310821104076), RAJARAJESHWARAN N (310821104075), VIJAYA KUMAR S (310821104110), VISHWANATH R R (310821104116)** " who carried out the project work under my supervision.

DR.J.ANITHA GNANASELVI
HEAD OF THE DEPARTMENT

JEEVITHA D
SUPERVISOR

TABLE OF CONTENTS

1. Introduction:

Project-Title: “Book a Doctor using MERN”

The "Book a Doctor Using MERN" project addresses the growing need for accessible and efficient healthcare services. In recent years, online booking systems have become essential for managing patient appointments, reducing wait times, and optimizing healthcare provider schedules. This project leverages the MERN stack—MongoDB, Express, React, and Node.js—to create a responsive, user-friendly web application that allows users to book appointments with doctors conveniently.

The application provides real-time appointment availability, enabling users to select time slots that suit their schedules, whether they prefer early mornings, evenings, or weekends. Through scenario-based interactions, users can register, browse healthcare providers, and manage their appointments, receiving notifications and confirmations seamlessly. This ensures that both patients and doctors benefit from enhanced accessibility and communication.

Team Members:

1. Raj Kumar N - Worked on both the frontend and backend to ensure seamless integration and overall functionality of the application.

2. Rajarajeshwaran N - Focused on designing the platform’s interface and creating intuitive and user-friendly experiences.

3. Vijaya Kumar S - Handled the backend API development with Node.js, Express.js, and MongoDB.

4. Vishwanath R R - Worked on both the frontend and backend to ensure seamless integration and overall functionality of the application.

2. Project Overview:

Purpose

The purpose of the "Book a Doctor Using MERN" web application is to simplify the process of booking medical appointments by providing a seamless, user-friendly platform for patients, doctors, and administrators. This system aims to eliminate the inefficiencies of traditional appointment booking methods such as phone calls, long waiting times, and manual scheduling. The platform allows users to easily find doctors, schedule appointments in real-time, and manage their healthcare needs online. The goal is to enhance accessibility, reduce administrative burdens, and streamline the appointment management process for all parties involved, thereby improving the overall healthcare experience.

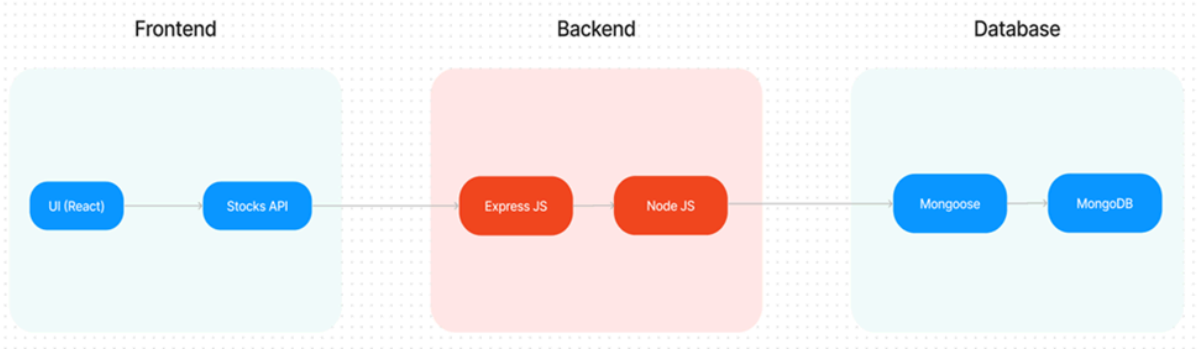
Features

- **User Registration & Login**:
    - **Patient Registration**: Patients can sign up by providing basic details like name, email, password, and contact information.
    - **Doctor Registration**: After a patient signs up, they can apply to register as a doctor. The doctor's registration will be reviewed and approved by the admin.
    - **Role-Based Login**: Users log in using their email and password. Based on their role (patient, doctor, or admin), they are redirected to the appropriate dashboard.
- **Doctor Search & Booking**:
    - **Search Functionality**: Patients can search for doctors by specialty, location, and availability. Detailed doctor profiles, including experience and services offered, are available for review.
    - **Real-Time Booking**: Patients can book appointments directly from the doctor's profile, selecting a time that works for them based on the doctor's available slots.
- **Appointment Management**:
    - **Pending Appointment Requests**: Once a patient books an appointment, the doctor reviews the request and confirms or rejects it. The patient is notified of the approval or rejection.

    - **Appointment History**: Both doctors and patients can view a history of past and upcoming appointments. Patients can reschedule or cancel appointments if needed.
- **Doctor Profile Management**:
    - **Doctor Dashboard**: Doctors can manage their availability, set their schedules, update personal information, and review patient appointments.
    - **Patient Records**: Doctors can maintain notes and records for each patient, ensuring they have the necessary information for consultations.
- **Admin Dashboard**:
    - **Doctor Verification**: The admin manages doctor registrations, approving or rejecting them based on the provided credentials.
    - **User Management**: Admins can monitor and manage user activity, including patients and doctors. They can suspend or delete users if necessary.
    - **Platform Analytics**: Admins can access data on platform usage, including the number of bookings, active users, and system performance metrics.
- **Notifications**:
    - Both doctors and patients receive email or in-app notifications for booking confirmations, appointment reminders, cancellations, and updates.
    - Admins are notified of any new user registrations or pending doctor verification requests.
- **Responsive & Intuitive UI**:
    - The platform is designed to be fully responsive, ensuring accessibility across a wide range of devices such as desktops, tablets, and smartphones.
    - An intuitive UI powered by React.js and material UI components provides a smooth, engaging experience for all users.
- **Secure Data Handling**:
    - User data is stored securely in MongoDB, with strong encryption for sensitive information.
    - Authentication and role-based access are handled using JWT tokens to ensure secure login and prevent unauthorized access.

These features collectively make the **Book a Doctor Using MERN** platform an efficient, secure, and user-friendly solution for modern healthcare appointment management. By streamlining the process, the system saves time for both patients and doctors while improving the overall healthcare experience.

3. Architecture:



Frontend Architecture: React.js

The frontend is designed using **React.js**, a popular JavaScript library for building user interfaces. It provides a responsive, dynamic, and interactive experience for all users, including patients, doctors, and admins.

- **Component-Based Structure**: The application is divided into reusable components like forms, dashboards, appointment cards, and search filters. Each component handles a specific function, ensuring modularity and ease of maintenance.
- **Routing**: **React Router** is used to enable seamless navigation between pages like the Landing Page, Login, Registration, Dashboards, and Doctor Profiles.
- **State Management**: State is managed using **Context API** for global state sharing and **useState/useEffect hooks** for local state handling. Real-time updates are fetched and displayed, such as appointment statuses and doctor availability.
- **Styling**: The user interface is designed using **Ant Design** and **Material-UI** for responsive and aesthetic design elements. **Bootstrap**

ensures compatibility across devices, including desktops, tablets, and smartphones.

- **API Integration**: The frontend communicates with the backend using **Axios** for RESTful API calls, ensuring secure and efficient data transfer.

Backend Architecture: Node.js and Express.js

The backend, built using **Node.js** and **Express.js**, serves as the server-side logic handler. It manages authentication, user roles, data processing, and API endpoints.

- **Routing & Middleware**:
  - **Express.js** handles routing for various endpoints, such as user registration, login, appointment booking, and admin verification.
  - Middleware like body-parser and cors ensures smooth data handling and secure cross-origin communication.

- **Authentication**:
  - **JWT (JSON Web Tokens)** is used for secure user authentication and role-based access control (admin, doctor, or patient).
  - Passwords are hashed using **bcrypt** for enhanced security.

- **API Development**: RESTful APIs are designed to handle CRUD operations for users, doctors, and appointments. APIs are optimized for scalability and efficiency.

- **Error Handling**: Robust error-handling mechanisms ensure the backend gracefully handles invalid requests, authentication failures, or database errors.

Database Architecture: MongoDB

The application uses **MongoDB**, a NoSQL database, for efficient and scalable data storage. It stores data in a JSON-like format, allowing flexibility in handling structured and semi-structured data.

Schema Design:

1.Users Collection:

_id:ObjectId('67381a065b93f50d933a4ed8')
fullName:"Admin"
email:"admin@gmail.com"
password:"$2a$10$dfdrR.iHk1hMU.i40y3O1ea7GeOZm0jDgeEpZCwS4YWk5kXCLV03u"
phone:"4598763210"
type:"admin"
notification:Array (1)
seennotification:Array (empty)
Isdoctor:false

2.Doctors Collection:

_id:ObjectId('67385b5e2762c39ce54f7a1e')
userId:ObjectId('67384be35b93f50d933a4f7f')
fullName:"Doctor"
email:"doctor@gmail.com"
phone:"7894561230"
address:"Chennai"
specialization:"Neurologist"
experience:"2"
fees:2000
status:"approved"
timings:"17:00,22:00"
image:"docimage_1731746654589.jpeg"
createdAt:2024-11-16T08:44:14.642+00:00
updatedAt:2024-11-16T08:46:01.222+00:00

3.Appointments Collection:

_id:ObjectId('67385c262762c39ce54f7a89')
userId:ObjectId('67384bfa5b93f50d933a4f82')
doctorId:ObjectId('67385b5e2762c39ce54f7a1e')
userInfo:Object
doctorInfo:Object
date:"2024-11-18 18:00"

document:Object
status:"approved"
createdAt:2024-11-16T08:47:34.916+00:00
updatedAt:2024-11-16T08:48:34.993+00:00

- **Interactions**:
  - **User Registration**: Stores new user credentials and role (patient or doctor).
  - **Doctor Registration**: Adds doctor details to the "Doctors" collection, flagged as "pending" for admin review.
  - **Appointment Management**: Manages bookings by linking patients and doctors using their respective IDs.
- **Indexes**: MongoDB indexes are used on fields like email, doctorId, and patientId to optimize search and retrieval operations.

4. Setup Instructions:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, and React.js:

- ☐ Node.js and npm
- ☐ Express.js
- ☐ MongoDB
- ☐ Moment.js
- ☐ React.js
- ☐ Antd
- ☐ HTML, CSS, and JavaScript
- ☐ Database Connectivity(Mangoose)

- **Node.js and npm:**

  Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server side. It provides a scalable and efficient platform for building network applications.

  Install Node.js and npm on your development machine, as they are required to run JavaScript on the server side.

  Download: https://nodejs.org/en/download/

Installation instructions: https://nodejs.org/en/download/package-manager/

npm init

- **Express.js:**

  Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

  Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

  Installation: Open your command prompt or terminal and run the following command:

  npm install express

- **MongoDB:**

  MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

  Set up a MongoDB database to store your application's data.

  Download: https://www.mongodb.com/try/download/community

  Installation instructions: https://docs.mongodb.com/manual/installation/

- **Moment.js:**

  Momentjs is a JavaScript package that makes it simple to parse, validate, manipulate, and display date/time in JavaScript. Moment. js allows you to display dates in a human-readable format based on your location. Install React.js, a JavaScript library for building user interfaces.

  Follow the installation guide: https://momentjs.com/

- **React.js:**

  React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

  Install React.js, a JavaScript library for building user interfaces.

  Follow the installation guide: https://reactjs.org/docs/create-a-new-react-app.html

- **Antd:**

  Ant Design is a React. js UI library that contains easy-to-use components that are useful for building interactive user interfaces. It is very easy to use as well as integrate. It is one of the smart options to design web applications using react.

  Follow the installation guide: https://ant.design/docs/react/introduce

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

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

https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/

- **Front-end Framework**: Utilize Reactjs to build the user-facing part of the application, including entering booking room, status of the booking, and user interfaces for the admin dashboard.

For making better UI we have also used some libraries like material UI and boostrap.

Install Dependencies:

• Navigate into the cloned repository directory:

cd book-a-doctor

• Install the required dependencies by running the following commands:

Create a split in the terminal to handle both backend and frontend initialization.

To get all the necessary frontend packages,

>>cd frontend

>>npm install

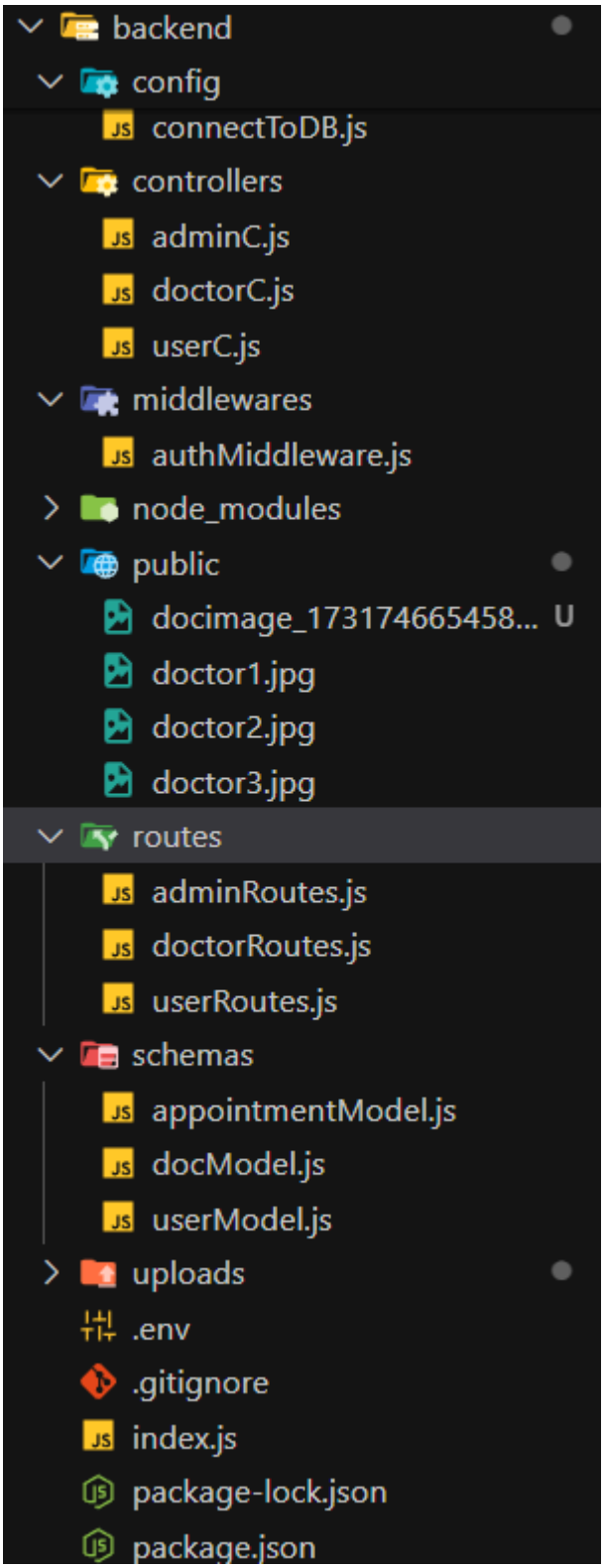
To get all the necessary backend packages,

>>cd backend

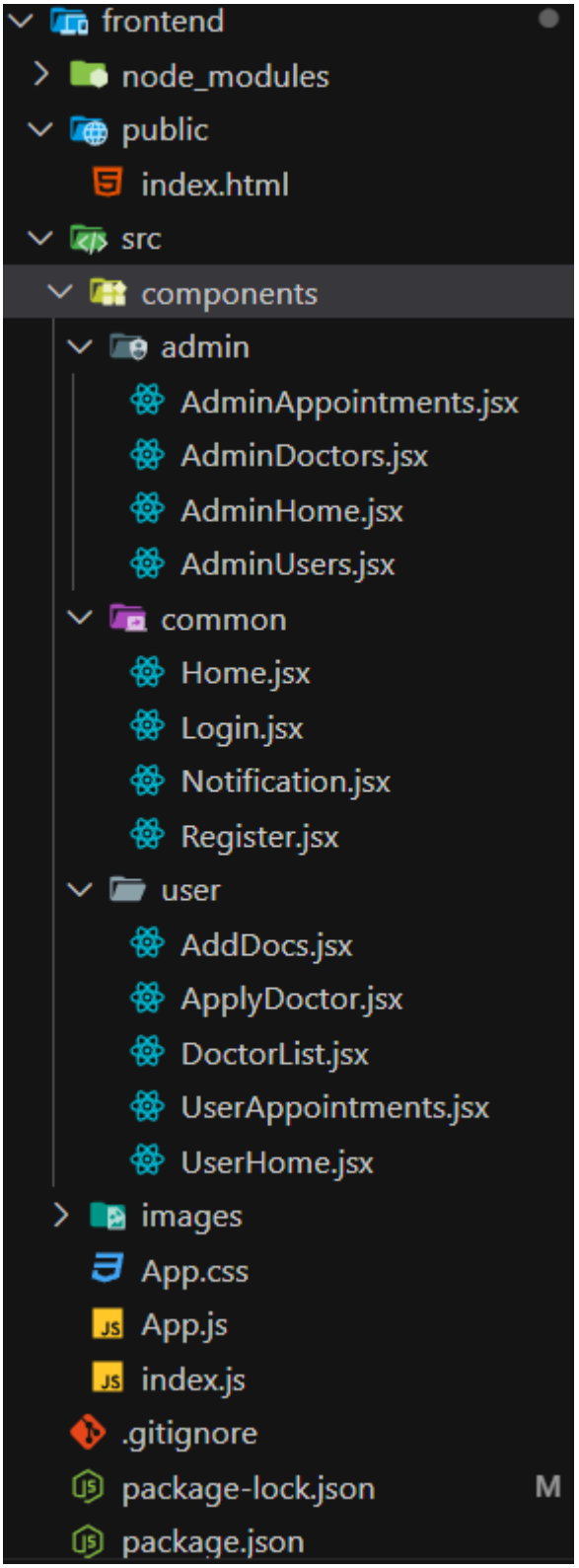
>>npm install

You have successfully installed and set up the online complaint registration and management app on your local machine. You can now proceed with further customization, development, and testing.

5. Folder Structure:



Backend



Frontend

Key Components:

a. Config

Manages the connection to the **MongoDB** database.

b. Models

The **models** directory defines the structure of data stored in MongoDB using **Mongoose** schemas.

c. Controllers

The **controllers** handle business logic and are responsible for processing incoming requests and interacting with the database.

d. Middleware

Middleware ensures the application is secure and handles errors effectively.

e. Routes

The **routes** directory defines API endpoints and maps them to specific controllers.

f. Utils

Utility functions are reusable helpers used across the backend.

Authentication & Authorization:

- Authentication is handled using **JWT (JSON Web Tokens)**, which ensures that only authorized users can access secure routes.

Error Handling:

- Centralized error handling using **errorMiddleware.js** ensures consistent API responses in case of invalid requests or server issues.

Server Initialization:

The **server.js** file initializes the backend:

- Loads environment variables using dotenv.
- Sets up database connection by importing db.js.

6. Running the Application:

Start the Development Server:

• To start the development server, execute the following command in each terminal i.e: the both frontend and backend:

npm start

• The book a doctor app will be accessible at http://localhost:3000

7. API Documentation:

The backend provides a comprehensive set of RESTful APIs for managing user, doctor, admin, and appointment operations. Below is the detailed documentation of the endpoints, including request methods, parameters, and example responses.

1. User APIs

1.1 Register User

- **Endpoint**: /api/users/register
- **Method**: POST
- **Description**: Registers a new user (patient or doctor).

Request Body:

```
{
  "username": "John Doe",
  "email": "john@example.com",
  "password": "password123",
  "role": "patient"
}
```

Response:


```
{
  "message": "User registered successfully",
  "user": {
    "id": "64fa2be2ed123abc456def",
    "username": "John Doe",
    "email": "john@example.com",
    "role": "patient"
  }
}
```


1.2 Login User

- **Endpoint**: /api/users/login
- **Method**: POST
- **Description**: Authenticates a user and returns a token.

Request Body:


```
{
  "email": "john@example.com",
  "password": "password123"
}
```


Response:


```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

1.3 Get User Profile

- **Endpoint**: /api/users/profile
- **Method**: GET
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>
- **Description**: Retrieves the logged-in user's profile.

Response:

```
{
  "id": "64fa2be2ed123abc456def",
  "username": "John Doe",
  "email": "john@example.com",
  "role": "patient"
}
```

2. Doctor APIs

2.1 Register as Doctor

- **Endpoint**: /api/doctors/register
- **Method**: POST
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>
- **Description**: Registers a user as a doctor (requires admin approval).

Request Body:

```
{
  "name": "Dr. Smith",
  "specialization": "Cardiology",
  "experience": 10,
  "availability": ["2024-11-18T10:00:00Z", "2024-11-18T14:00:00Z"]
}
```

Response:


```
{
  "message": "Doctor registration submitted. Awaiting admin approval."
}
```


2.2 Get All Approved Doctors

- **Endpoint**: /api/doctors
- **Method**: GET
- **Description**: Retrieves all approved doctors.

Response:


```
[
  {
    "id": "64fa3de9ed123abc456efg",
    "name": "Dr. Smith",
    "specialization": "Cardiology",
    "experience": 10,
    "status": "approved"
  },
  {
    "id": "64fa4ae2ed123abc456hij",
    "name": "Dr. Alice",
    "specialization": "Dermatology",
    "experience": 7,
    "status": "approved"
  }
]
```

3. Admin APIs

3.1 Approve Doctor Registration

- **Endpoint**: /api/admin/doctors/:id/approve
- **Method**: PATCH
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>
- **Description**: Approves a doctor registration.
- **Path Parameters**:
  - id: Doctor ID.

Response:

```
{

  "message": "Doctor approved successfully."

}
```

3.2 Get Pending Doctor Approvals

- **Endpoint**: /api/admin/doctors/pending
- **Method**: GET
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>
- **Description**: Retrieves all pending doctor registration requests.

Response:

```
[

  {

    "id": "64fa5be2ed123abc456klm",

    "name": "Dr. Robert",

    "specialization": "Neurology",

    "experience": 5,

    "status": "pending"

  }]
```

4. Appointment APIs

4.1 Book an Appointment

- **Endpoint**: /api/appointments/book
- **Method**: POST
- **Headers**:
    - Authorization: Bearer <JWT_TOKEN>
- **Description**: Books an appointment with a doctor.

Request Body:

```
{
  "doctorId": "64fa3de9ed123abc456efg",
  "date": "2024-11-18T10:00:00Z"
}
```

Response:

```
{
  "message": "Appointment booked successfully.",
  "appointment": {
    "id": "64fa6cd2ed123abc456nop",
    "doctorId": "64fa3de9ed123abc456efg",
    "date": "2024-11-18T10:00:00Z",
    "status": "pending"
  }
}
```

4.2 Get User Appointments

- **Endpoint**: /api/appointments
- **Method**: GET
- **Headers**:
    - Authorization: Bearer <JWT_TOKEN>

- **Description**: Retrieves appointments for the logged-in user.

Response:

```
[
  {
    "id": "64fa6cd2ed123abc456nop",
    "doctor": "Dr. Smith",
    "date": "2024-11-18T10:00:00Z",
    "status": "confirmed"
  }
]
```

Authentication

- Every endpoint, except /api/users/register and /api/users/login, requires a **JWT Token** for authorization.
- Tokens are passed in the Authorization header as Bearer <TOKEN>.

Error Responses

Invalid Token:

```
{
  "error": "Unauthorized"
}
```

Validation Error:

```
{
  "error": "Invalid input data"
}
```

8. Authentication:

User Authentication

The project employs **JWT (JSON Web Tokens)** for user authentication. JWT is a stateless, scalable, and secure method to authenticate users and manage their sessions.

- **Process**:
  - **User Login**:
    - Users provide their email and password to log in.
    - The backend verifies the credentials against the database.
    - If valid, the server generates a JWT.
  - **Token Structure**:
    - The token consists of three parts:
      - Header (Algorithm and Token Type)
      - Payload (User data like id, role, etc.)
      - Signature (to verify authenticity)
  - **Token Issuance**:
    - Tokens are issued using a secret key stored in an environment variable for security.
    - Tokens have an expiration time (e.g., 24 hours) to enhance security.
- **Storage**:
  - The JWT is returned to the client and typically stored in the browser's **localStorage** or **HTTP-only cookies** for enhanced security.

Registration and Password Handling

- **Registration**:
  - Users register by providing a unique email and password, which are stored securely in the MongoDB database.
- **Password Security**:
  - Passwords are hashed using **bcrypt** before storage.
  - During login, the entered password is hashed and compared with the stored hash.

9. Authorization:

Authorization ensures users can access only the features and data they are permitted to. The project employs role-based access control (RBAC) to manage permissions.

Role-Based Access Control

- **Roles:**
  - **Admin**: Can manage users, approve doctors, and oversee all operations.
  - **Doctor**: Can view and manage their appointments.
  - **User (Patient)**: Can browse doctors, book appointments, and manage bookings.
- **Implementation**:
  - The user role is included in the JWT payload.
  - Middleware checks the user role and grants access based on permissions.

Sessions

While JWT is stateless, optional session management is implemented using **HTTP-only cookies** to enhance security in the following cases:

- For storing refresh tokens (if implemented).
- To prevent CSRF attacks.

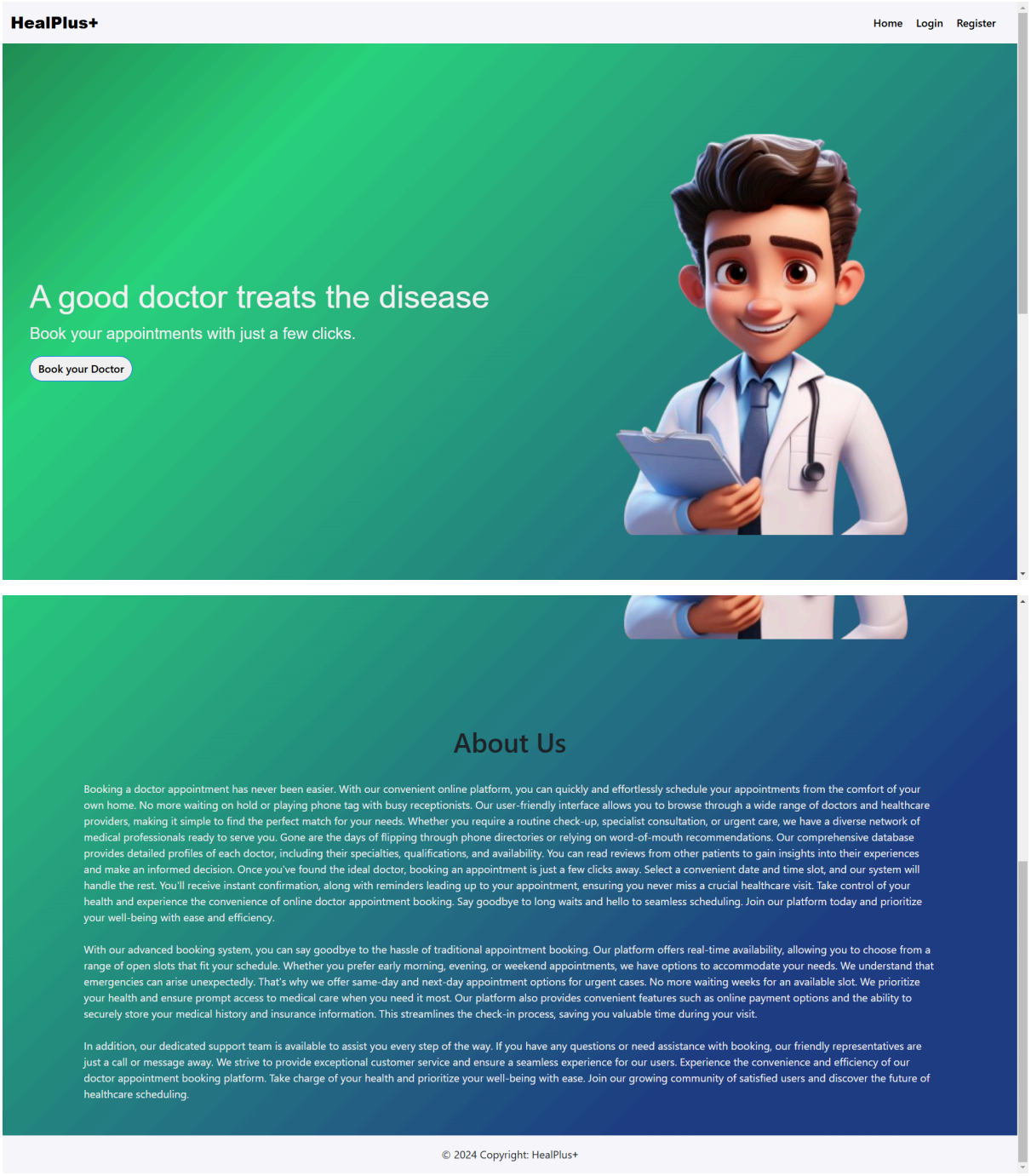
Security Measures

1. **Secure Storage**:
   - JWTs are stored in **HTTP-only cookies** to protect against XSS attacks.
2. **Environment Variables**:
   - Secret keys and database credentials are stored in .env files to prevent exposure.
3. **Rate Limiting**:
   - APIs have rate limits to mitigate brute-force attacks.
4. **CORS**:
   - CORS (Cross-Origin Resource Sharing) is configured to allow requests only from trusted domains.

10. User Interface:

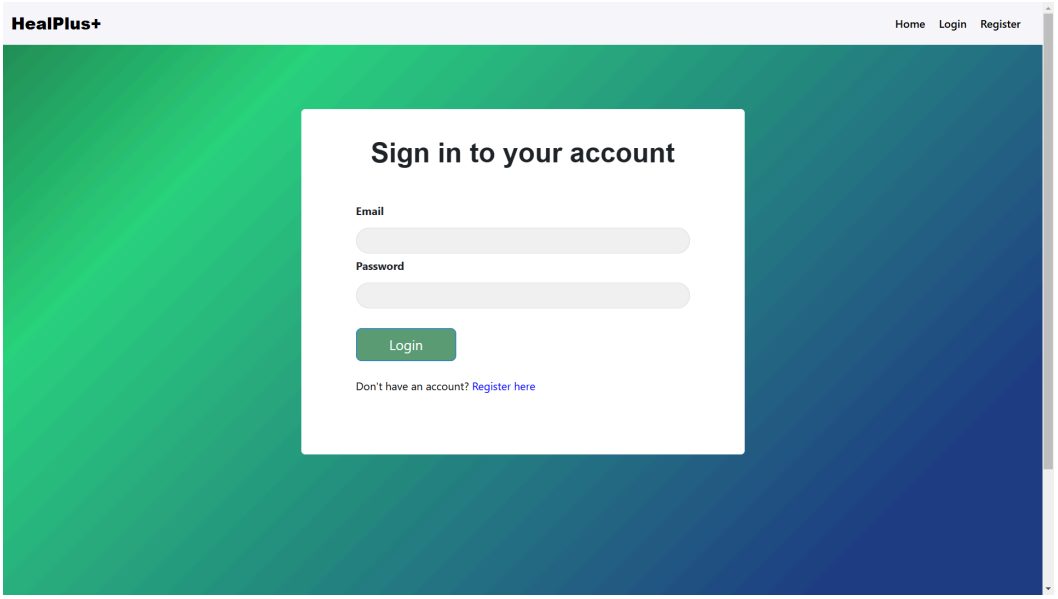
Landing Page:

This landing page contains the navbar which links to the home, login and register pages. There is a book a doctor btn which redirects the user to the login page. Below there is a About us section , which gives the complete details of this web application. At the end there is a Footer.



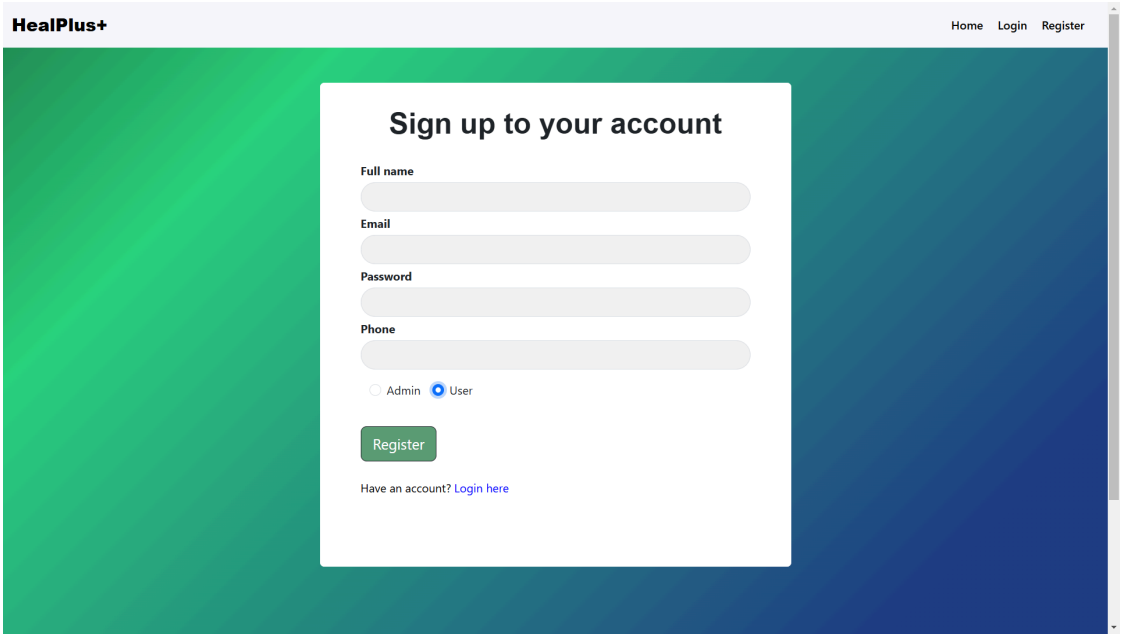
Login Page:

Features: Secure login with email and password, role-based redirects, and error messages for invalid credentials. Provides authenticated access to dashboards for patients, doctors, and admins.



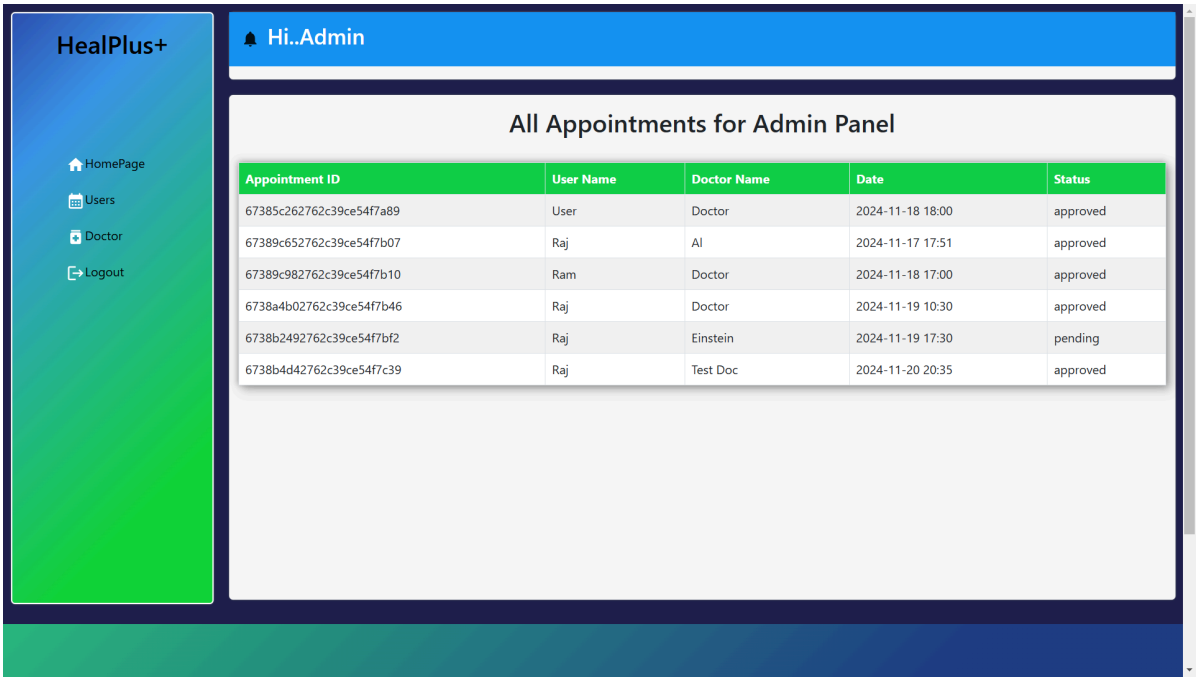
Register Page

Features: Form fields for user details (name, email, password, and role), real-time validation, and error handling. Enables new users (patients or doctors) to sign up securely.

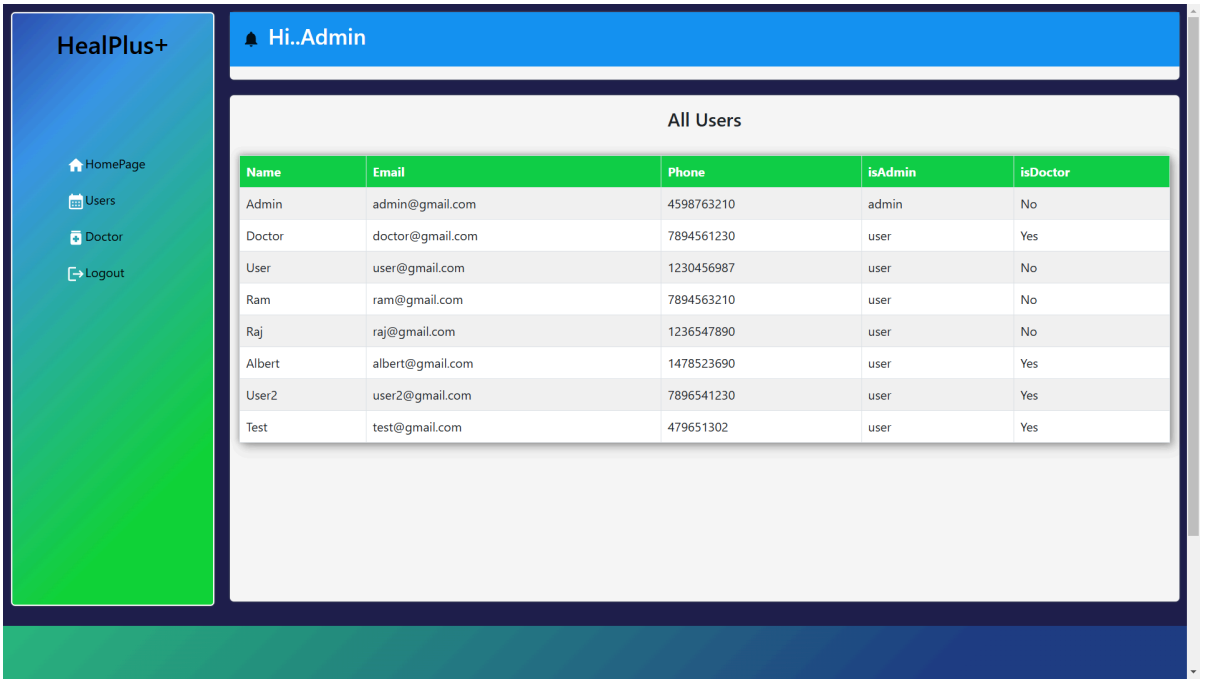


Admin Dashboard

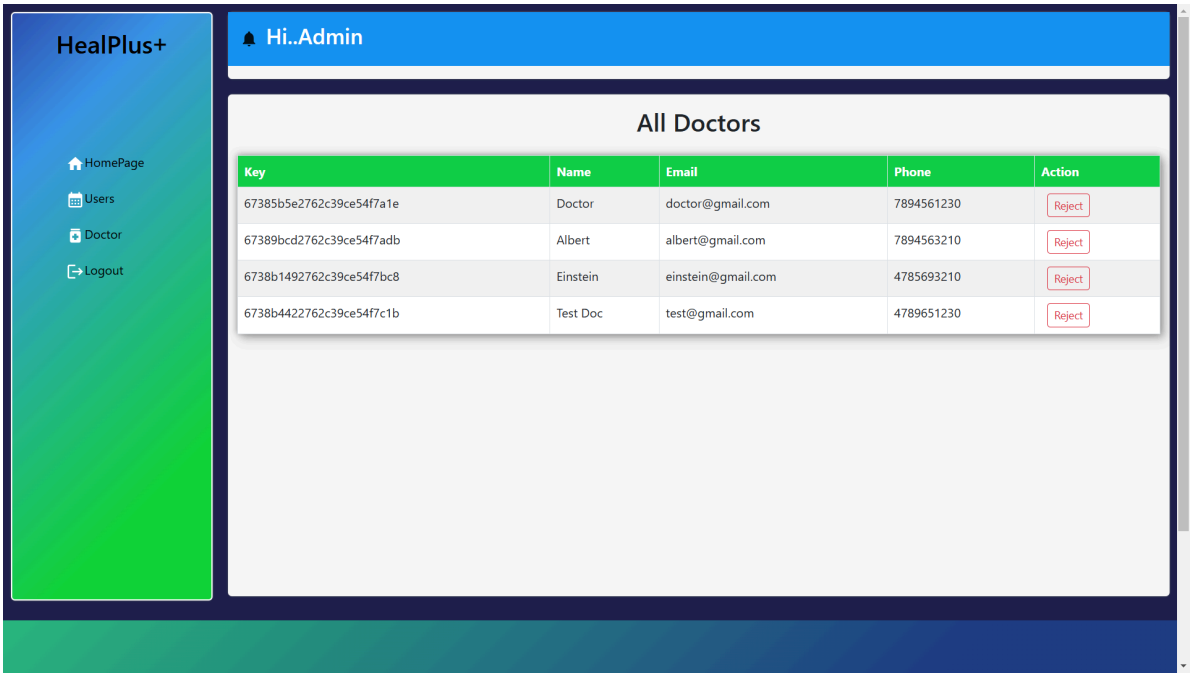
Features: Manage users (approve/reject doctors, view patients), analytics for platform usage, and content management. Allows admins to monitor and manage the platform's operations effectively.



Panel which displays all the users of the application.

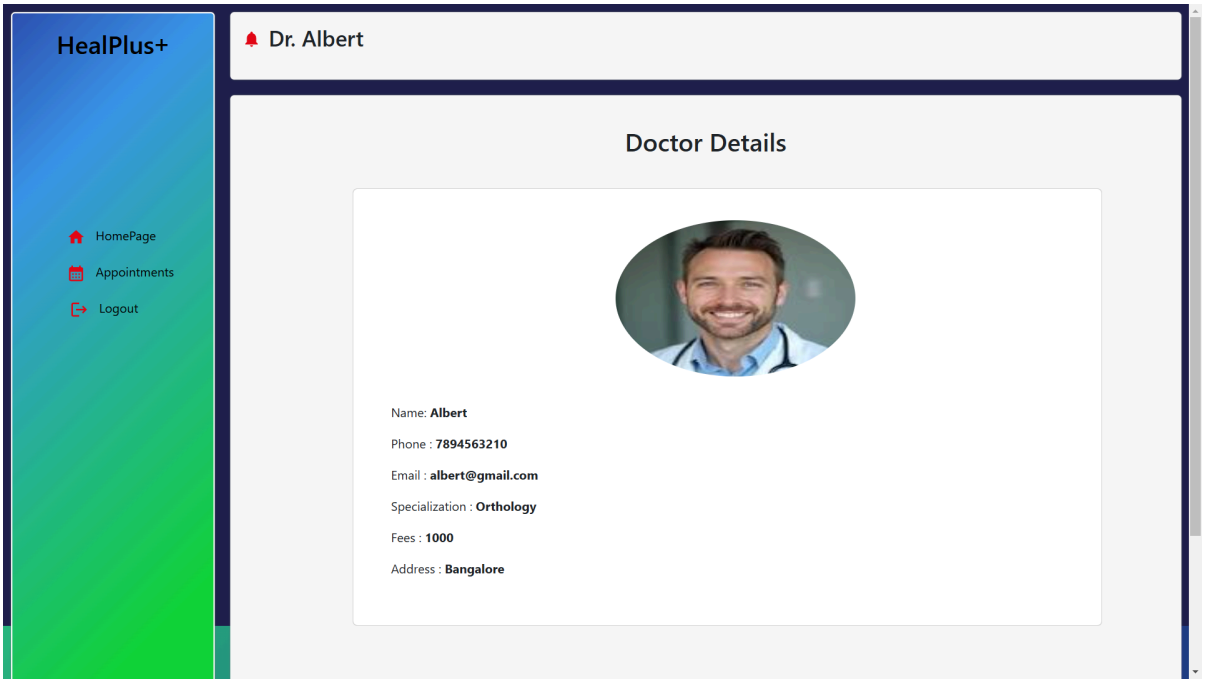


Panel displays the list of doctors registered for this application.

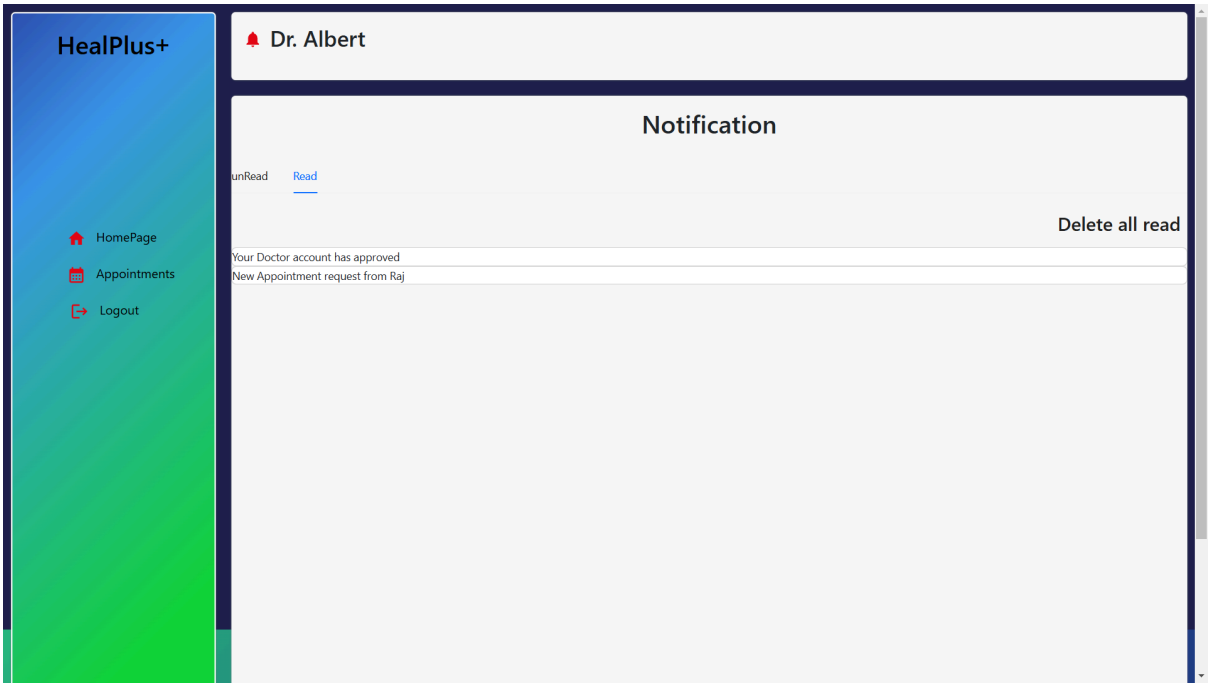


Doctor Dashboard:

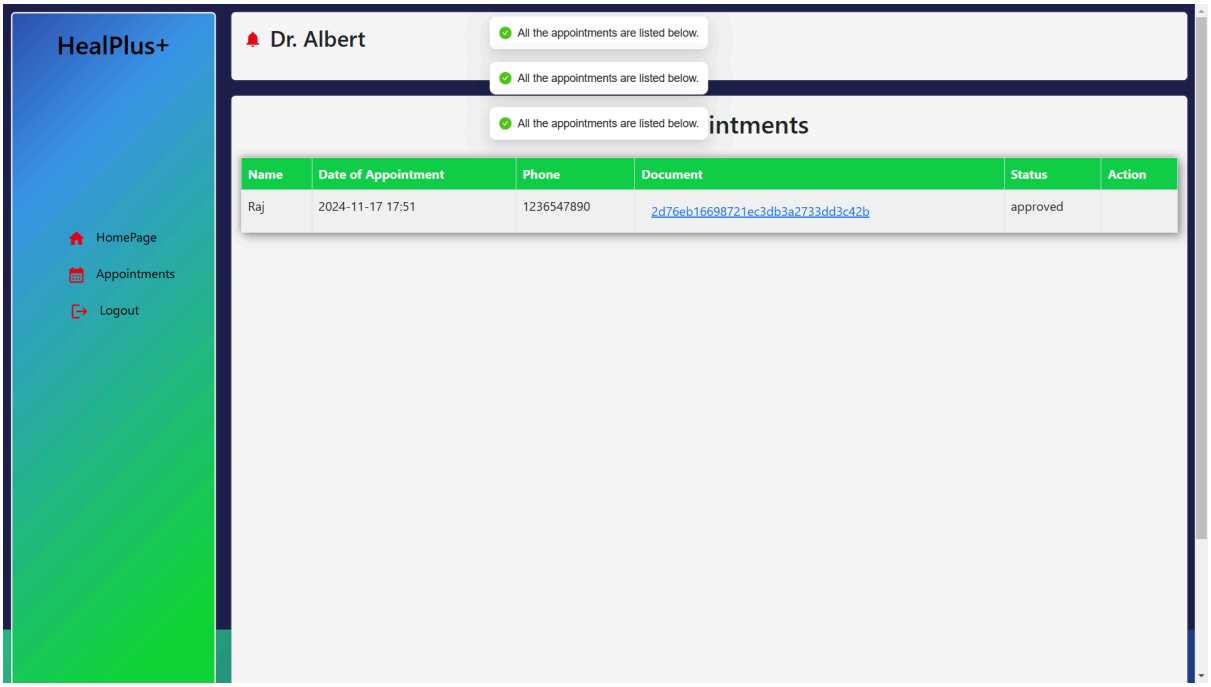
Features: Display profile, manage schedules, access patient records, and confirm or decline appointments. Empowers doctors to efficiently manage their availability and patient interactions. Allow to manage notifications from patients.



Doctors notification window,



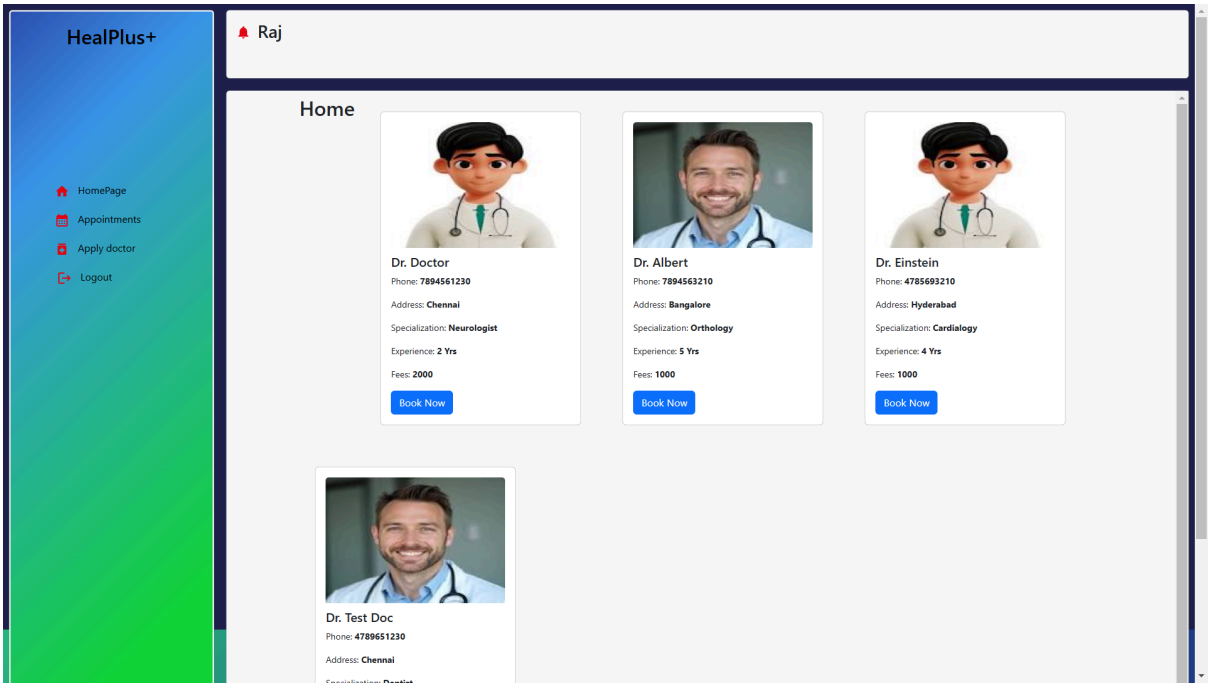
Doctor approving the appointment from patient,



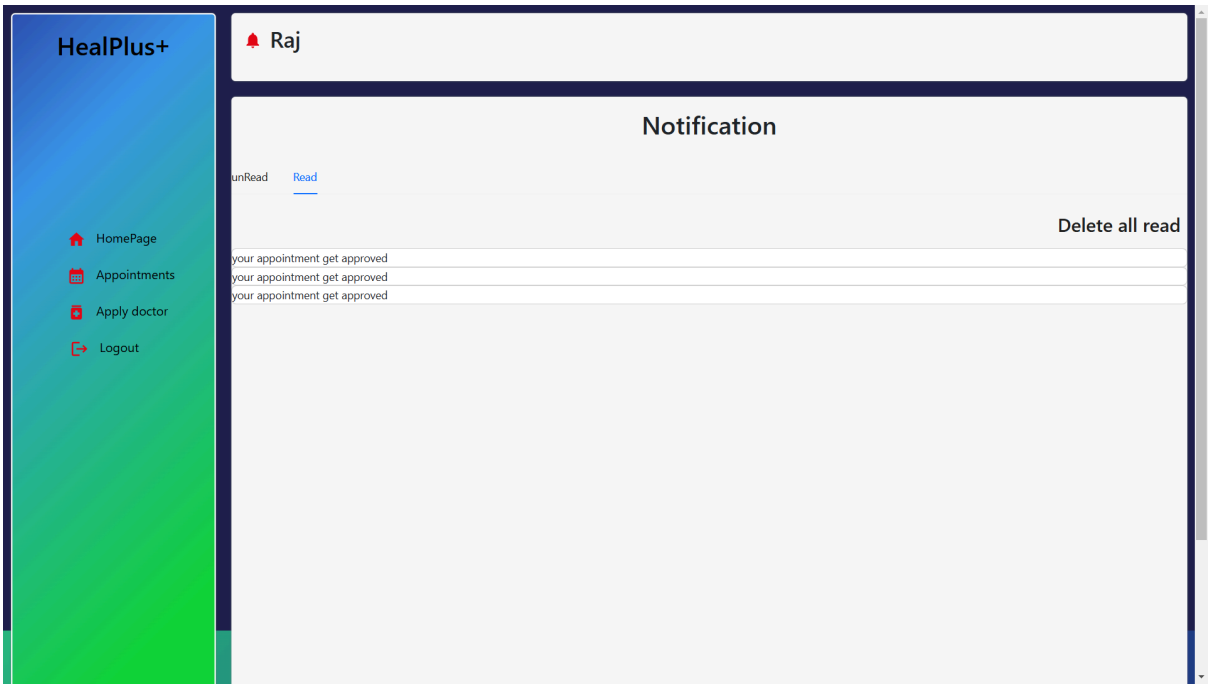
User Dashboard:

Features: Manage profile, book appointments, view medical records, and reschedule bookings. Provides patients with tools to manage appointments and personal data.

It displays the doctors available for treatment in the user interface. On clicking Book now the user can book an appointment for the doctor.

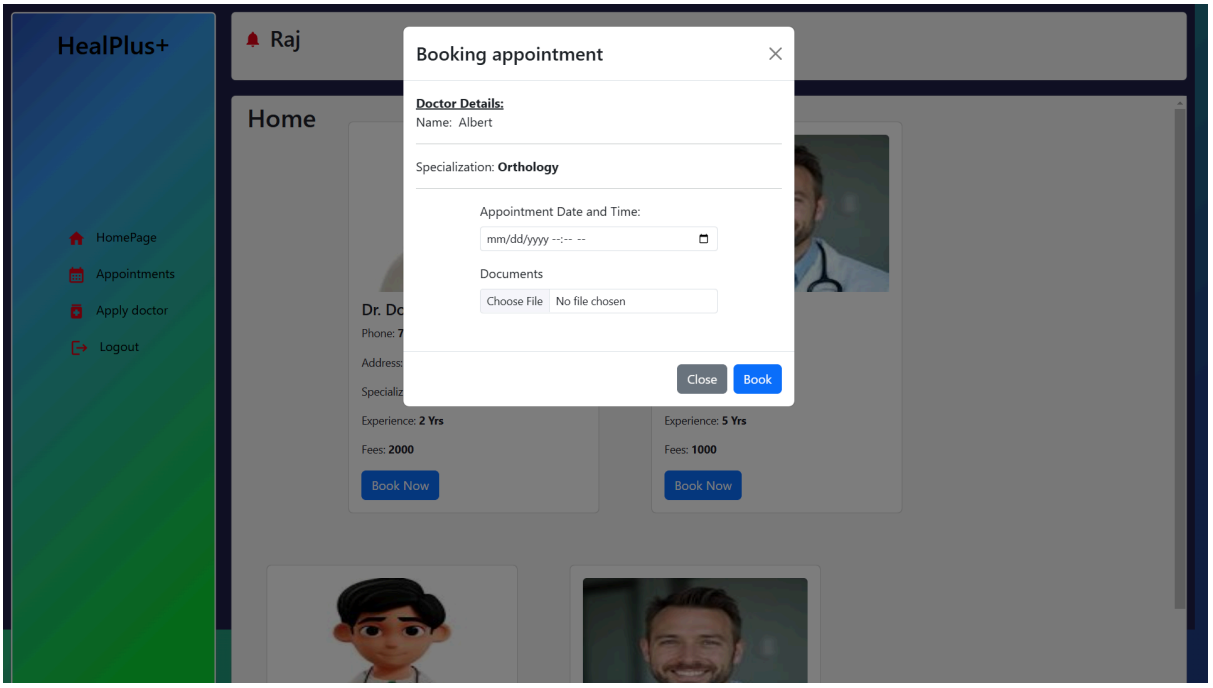


Notifications for the user,

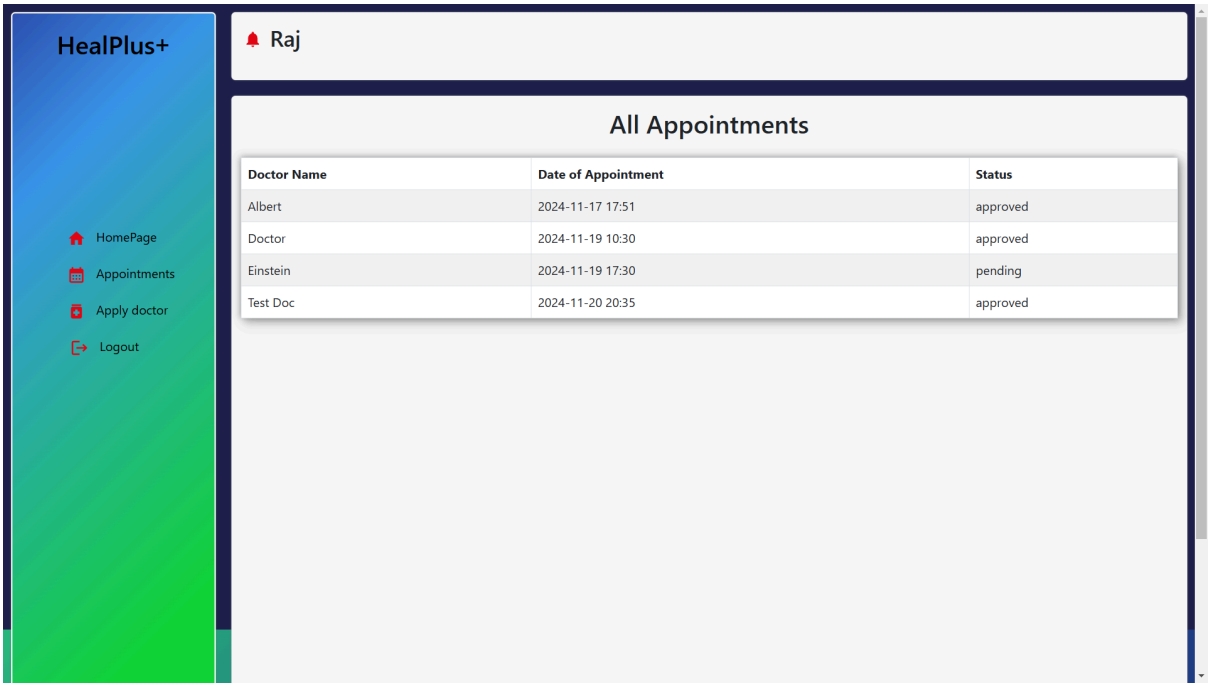


Appointments made by the user, mentioning the dae and time for scheduling the appointment.

Provide the past records of the patient or any prescriptions of the doctor.

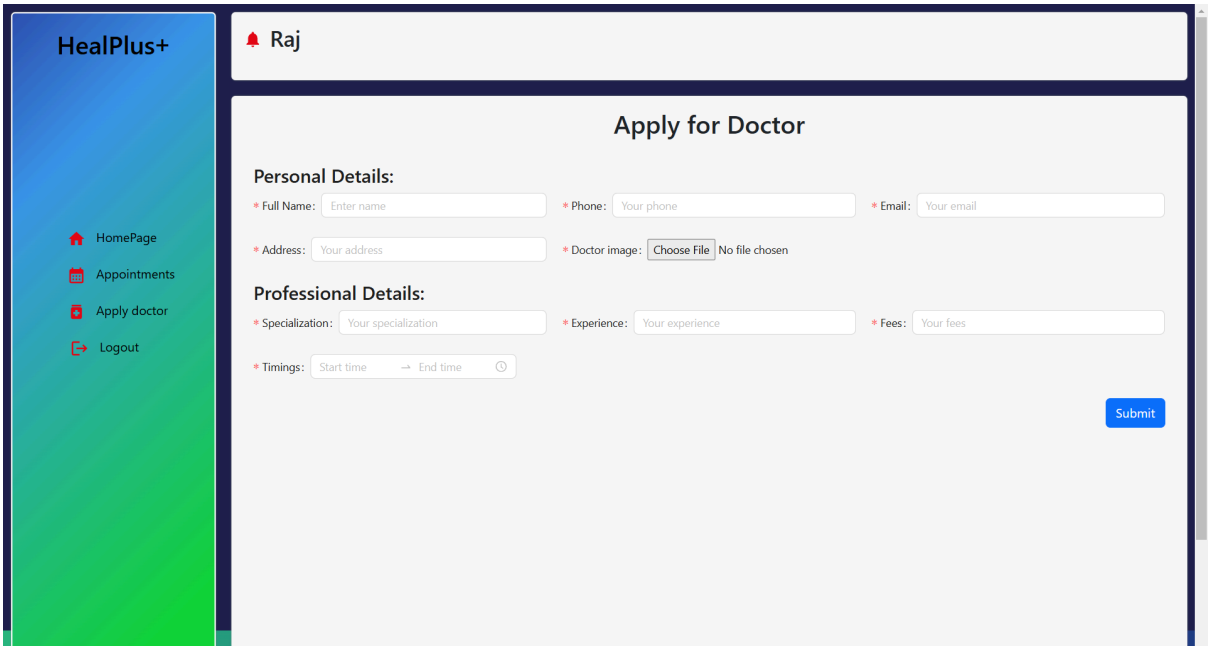


All the appointments made by the user are listed here,

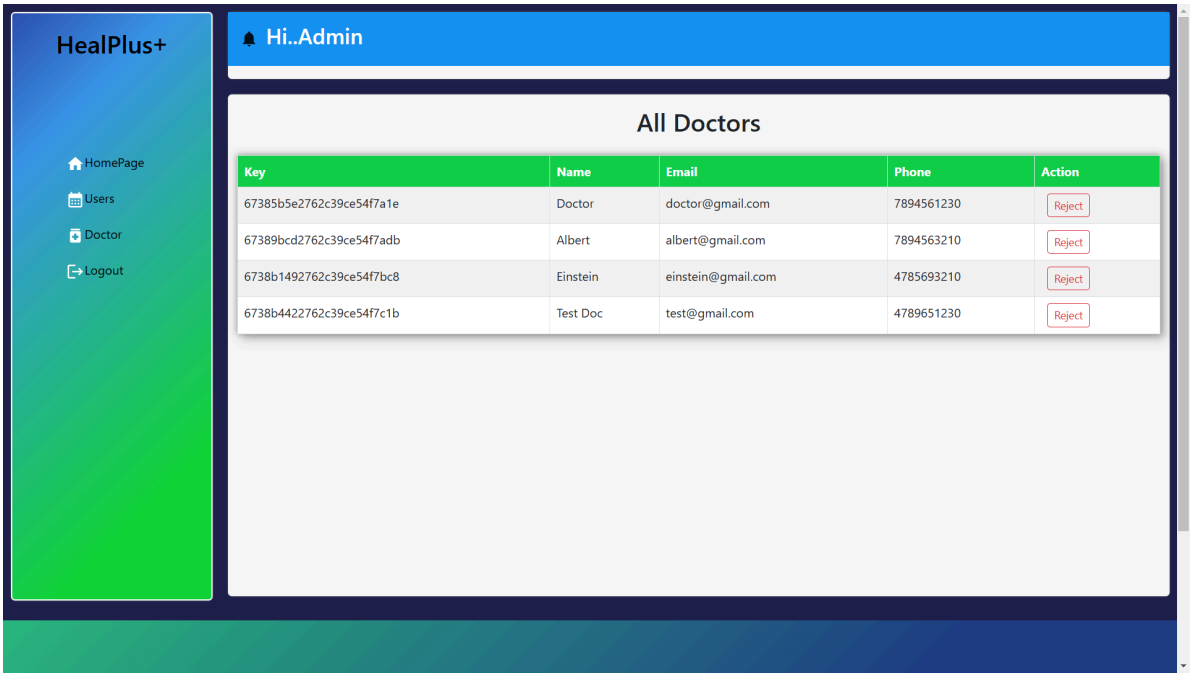


Doctor Registration:

The **Doctor Registration** feature enables users to register as doctors on the platform. The registration process ensures proper validation and admin approval for authenticity and security. This process ensures only verified doctors are registered on the platform, maintaining trust for users seeking medical services.



After the user apply as a doctor, the admin can approve or reject the doctor application, by verifying the history records of the doctor.



11. Testing:

Manual Testing Process:
To ensure the application functions as expected by simulating real-world user interactions and validating database operations.

Testing Steps

- **Landing Page**:
    - Verify that the page loads correctly across devices.
    - Check that navigation links work (e.g., Home, About Us, Login, Register).
    - Test the search functionality for doctors using different criteria (e.g., specialty, location).
    - Confirm that clicking "Register" or "Book an Appointment" redirects to the correct pages.
- **Registration Page**:
    - Attempt to register with valid details for both patients and doctors.
    - Test error messages for invalid inputs (e.g., empty fields, invalid email format, weak password).
    - Check if the data is saved correctly in the database (MongoDB).
    - Verify that a duplicate email entry is rejected.
- **Login Page**:
    - Test login with valid credentials for all roles (Admin, Doctor, User).
    - Check error messages for invalid credentials or missing input fields.
    - Confirm redirection to the appropriate dashboard (Admin/User/Doctor) upon successful login.
- **Admin Dashboard**:
    - Log in as an admin and approve/reject doctor registrations.
    - Verify that approved doctors are added to the database and visible on the user dashboard.
    - Test content management features, like updating platform FAQs or policies.
- **User Dashboard**:
    - Log in as a user and attempt to book an appointment with a doctor.
    - Check if appointment data is stored correctly in the database.

    - Test rescheduling or canceling appointments.
    - Verify the display of past appointments and medical records (if available).
- **Doctor Dashboard**:
    - Log in as a doctor and manage the schedule (update availability, confirm/cancel appointments).
    - Validate the doctor's access to patient appointment requests and records.
    - Check if status changes reflect correctly in the database and on the user dashboard.

Database Verification

- Use MongoDB queries to check data integrity after each user interaction.
- Examples:
    - After registration: Confirm the user record exists in the `users` collection.
    - After booking: Ensure appointment data is stored in the `appointments` collection.
    - On status updates: Verify that appointment status changes (e.g., "pending" to "confirmed") are correctly updated.

Observations During Testing

- All navigation and redirection links are working as intended.
- Data validation at the front-end and back-end prevents invalid or malicious input.
- Database records are updated accurately for all operations (registration, bookings, and updates).
- Role-based access control ensures that users can only access relevant features.

Challenges Encountered

- Minor UI inconsistencies on mobile devices(reducing screen size).
- Delay in real-time updates when multiple users interact simultaneously.

12. Known Issues:

- **Doctor Registration Delay**: Doctors need manual approval from the admin, causing delays in availability. Solution: Implement automatic approval or third-party verification.
- **Inconsistent Date/Time Formatting**: Date and time handling issues due to time zone discrepancies. Solution: Use libraries like Moment.js for standardization.
- **Lack of Real-Time Availability Updates**: No live updates of doctor availability. Solution: Implement real-time communication with **Socket.IO** or polling.
- **Limited User Profile Management**: Users cannot update personal details or change passwords. Solution: Add profile management features.
- **No Appointment Rescheduling/Cancellation**: Once booked, appointments cannot be modified. Solution: Add functionality for rescheduling and cancellations.
- **Basic Admin Dashboard**: Admin can only approve doctors, lacking deeper management features. Add features like viewing appointments and generating reports.
- **Mobile Responsiveness Issues**: Some pages, especially dashboards, do not render well on mobile.
- **No Email/SMS Notifications**: No notifications for appointment updates. Integrate email/SMS notifications using services like **SendGrid** or **Twilio**.
- **Admin Authentication Weakness**: Admins use the same authentication system as users. Solution: Implement stricter admin authentication (e.g., MFA).
- **Scalability Issues**: MongoDB may struggle with increased traffic and data.
- **No Payment Integration**: No way to handle payments for appointments. Solution: Integrate payment systems like **Stripe** or **PayPal**.

13.Future Enhancements:

- **Automated Doctor Approval**: Implement automatic verification for doctor registrations to reduce delays.
- **Real-Time Availability Updates**: Integrate **Socket.IO** or polling to show live doctor availability.
- **Appointment Management**: Add features for rescheduling and canceling appointments.
- **Advanced Search Filters**: Implement filters for location, specialization, rating, and availability to improve doctor search.
- **Enhanced Admin Dashboard**: Add features to view all appointments, track users, and generate reports.
- **Mobile Optimization**: Improve mobile responsiveness, especially for dashboards and user booking flows.
- **Email/SMS Notifications**: Integrate notification systems for appointment confirmations, cancellations, and reminders.
- **Stronger Admin Authentication**: Add multi-factor authentication (MFA) for admin accounts to enhance security.
- **Scalability Improvements**: Use database sharding or caching strategies to handle higher traffic and data loads.
- **Payment Integration**: Implement a payment gateway like **Stripe** or **PayPal** for handling appointment payments.

14. Conclusion:

The "Book a Doctor" platform offers a user-friendly solution for scheduling doctor appointments online, bridging the gap between patients and healthcare providers. Through a seamless front-end interface built with React and a robust back-end powered by Node.js, Express.js, and MongoDB, the platform enables users to book appointments, while doctors and admins can manage their schedules and registrations efficiently.

Despite its functionality, the project has room for improvement, particularly in areas like real-time availability updates, appointment management, and scalability. Future enhancements such as automated doctor approval, mobile optimization, and payment integration will significantly enhance the user experience and operational efficiency.

15. Result:

The "Book a Doctor" platform was successfully developed and deployed, meeting the core objectives of allowing users to book doctor appointments, doctors to manage their schedules, and admins to oversee the platform's operations.