Full Stack Development with MERN

Book a Doctor

Book Smarter. Heal Faster

1. Introduction

Project Title: Book a Doctor — Online Healthcare Appointment Booking System

Team Members:

Chandamala Chandrika — Project Coordinator & Backend Developer
B Ganga — Frontend Developer
Gudur Meghana — UI/UX Designer
K Mounika — Full Stack Developer & Database Specialist

2. Project Overview

- **Purpose:**
  The purpose of the Book a Doctor project is to develop an intuitive and efficient online platform that connects patients with healthcare providers seamlessly. The system aims to simplify the appointment booking process by allowing users to search for doctors by specialty, location, and availability, book appointments, manage schedules, and receive timely notifications. This project addresses the challenges of traditional appointment systems by providing a user-friendly, secure, and accessible healthcare booking solution for patients, doctors, and administrators.

- **Features:**

  - **Patient Registration & Profile Management:** Secure sign-up and profile creation with personal and medical details.

  - **Doctor Search & Filtering:** Browse and filter doctors by specialty, location, and real-time availability.

  - **Appointment Booking & Management:** Easy scheduling of appointments with document uploads and automated notifications.

  - **Doctor Dashboard:** Doctors can manage their availability, appointments, and patient records securely.

  - **Admin Controls:** Admin panel for approving doctor registrations, managing users, and platform governance.

  - **Notifications:** Automated email and SMS reminders for appointments to reduce no-shows.

  - **Secure Authentication & Data Protection:** Utilizes JWT and bcrypt for secure login and password management, with encrypted data handling.

3. Architecture

- **Frontend:**
  The frontend is built using **React.js**, a component-based JavaScript library that facilitates building dynamic and responsive user interfaces. The architecture follows a modular design where reusable components handle different functionalities like doctor browsing, appointment booking, and user profile management. Routing is managed using React Router to navigate between pages seamlessly. Styling is done with Bootstrap, Material UI, and Ant Design to ensure a consistent and modern look across devices. Axios handles API requests, enabling smooth communication with the backend.

- **Backend:**
  The backend uses **Node.js** with the **Express.js** framework to build a RESTful API. Express handles HTTP request routing, middleware processing, and business logic execution. The server manages user authentication, appointment scheduling, doctor and patient data management, and notification services. Security is enforced using JWT for token-based authentication and bcrypt for password hashing. The backend exposes various endpoints to support frontend operations such as user login, appointment creation, and admin controls.

- **Database:**
  The database is implemented with **MongoDB**, a NoSQL document-oriented database that stores data in flexible JSON-like documents. The schema consists of three main collections:

    - **Users:** Stores basic user details including login credentials, role flags (doctor/patient/admin), and contact information.

    - **Doctors:** Contains doctor-specific information linked to user accounts via userID, including specialization, availability timings, experience, and fees.

    - **Appointments:** Records appointment details linking users and doctors, including appointment date, status, and uploaded documents.
      Relationships are managed through references (foreign keys) between collections, enabling efficient queries and maintaining data integrity.

4. Setup Instructions

Prerequisites

- Node.js & npm: JavaScript runtime and package manager for running the backend and frontend.
  Download: https://nodejs.org/

- MongoDB: NoSQL database to store application data. Can be installed locally or use cloud services like MongoDB Atlas.
  Download: https://www.mongodb.com/try/download/community

- Git: Version control tool to clone the project repository.
  Download: https://git-scm.com/downloads

Installation

1. Clone the Repository
   Open your terminal and run the following commands one by one:
   git clone
   cd

2. Setup Backend
   Navigate to the backend directory and install dependencies:
   cd backend
   npm install

   Create a .env file inside the backend folder with the following environment variables:
   PORT=5000
   MONGO_URI=
   JWT_SECRET=

   Start the backend server by running:
   npm start

   The backend API will be available at http://localhost:5000.

3. Setup Frontend
   Open a new terminal window, navigate to the frontend directory, and install dependencies:
   cd frontend
   npm install

   Start the frontend server by running:
   npm start

   The frontend will be accessible at http://localhost:3000.

5. Folder Structure

Client (Frontend)

The React frontend is organized to support modular development and maintainability with the following key directories and files:

- **public/**: Contains static assets like the HTML template and images.
- **src/**: Main source folder containing all React components and related files.
  - **components/**: Reusable UI components such as doctor listings, appointment forms, and navigation bars.
  - **pages/**: Components representing full pages or views like Dashboard, Login, and Booking History.
  - **services/**: Contains API service functions to interact with the backend using Axios.
  - **App.js**: Root component handling routing and global state management.

    - **index.js**: Entry point for rendering the React application.

- **.env**: Environment variables specific to frontend configuration.
- **package.json**: Lists frontend dependencies and scripts.

Server (Backend)

The Node.js backend is structured to separate concerns, enhance scalability, and ensure clear API flow:

- **config/**: Configuration files such as database connection setup and environment variables.
- **controllers/**: Functions that handle the business logic for different API endpoints (users, doctors, appointments).
- **models/**: Mongoose schemas and models representing database collections like Users, Doctors, and Appointments.
- **routes/**: Defines Express route handlers mapping endpoints to controllers.
- **middleware/**: Custom middleware for tasks like authentication, error handling, and logging.
- **uploads/**: Folder to store uploaded files such as medical documents securely.
- **server.js**: Entry point that initializes the Express server, connects to MongoDB, and starts listening on the configured port.
- **.env**: Backend environment variables including database URI and JWT secrets.
- **package.json**: Backend dependencies and scripts.

6. Running the Application

To run the Doctor Booking application locally, follow these steps:

- **Frontend:**
  Open a terminal, navigate to the frontend directory (often named `frontend` or `client`), and run:
  ```
  npm start
  ```
  This will start the React development server, accessible at http://localhost:3000.

- **Backend:**
  Open another terminal, navigate to the backend directory (often named `backend` or `server`), and run:
  ```
  npm start
  ```
  This will start the Express server, typically running on http://localhost:5000.

Ensure both servers are running simultaneously for full functionality.

7. API Documentation

The backend exposes several RESTful API endpoints to support frontend operations. Below are the key endpoints, including methods, parameters, and example responses:

User Registration

POST `/api/users/register`

Request Body:

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

Response:

```
{
 "success": true,
 "message": "User registered successfully."
}
```

User Login

POST `/api/users/login`

Request Body:

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

Response:

```
{
 "token": "<jwt_token>",
 "user": {
  "_id": "user_id",
  "name": "John Doe",
  "email": "john@example.com",
  "type": "patient"
 }
}
```

Fetch Doctors List

GET `/api/doctors`

Query Parameters: Optional filters such as `specialization`, `location`

Response:

```
[
 {
  "_id": "doctor_id",
  "fullname": "Dr. Smith",
  "specialisation": "Cardiology",
  "experience": "10 years",
  "fees": 500
 },
 ...
]
```

Book Appointment

POST `/api/appointments`

Request Body:

```
{
 "doctorId": "doctor_id",
 "userId": "user_id",
 "date": "2025-07-01T10:00:00Z",
 "documents": ["document_url_1", "document_url_2"],
 "status": "pending"
}
```

Response:

```
{
 "success": true,
 "message": "Appointment booked successfully."
}
```

Other Endpoints:
Include routes for appointment status updates, doctor registration approval, user profile management, etc.

8. Authentication

The project uses **JWT (JSON Web Tokens)** for secure authentication and authorization, following this flow:

- **User Login:**
  Upon successful login, the backend generates a JWT signed with a secret key. This token contains user identification and role information.

- **Token Storage:**
  The token is sent to the frontend and stored securely (typically in memory or local storage). It is included in the Authorization header of subsequent API requests.

- **Authorization:**
  Middleware on the backend verifies the JWT on protected routes to ensure the user is authenticated and authorized to perform the requested action.

- **Password Security:**
  User passwords are hashed using **bcrypt** before storing in the database, preventing plaintext password leaks.

- **Session Management:**
  The system uses stateless token authentication, eliminating the need for server-side sessions, thus improving scalability.

9. User Interface

The Book a Doctor app features a clean, responsive, and intuitive interface designed with user experience in mind. Key UI features include:

- **Patient Dashboard:** Displays a searchable list of doctors with filters for specialty and location.

- **Appointment Booking Form:** Allows users to select available dates, upload medical documents, and confirm bookings easily.

- **Doctor Dashboard:** Enables doctors to view upcoming appointments, update statuses, and manage patient records.

- **Admin Panel:** Provides controls for verifying doctor registrations, managing users, and overseeing platform operations.

- **Notifications:** Timely alerts for appointment confirmations, cancellations, and reminders via email and SMS.

10. Testing

A comprehensive testing strategy was adopted to ensure the reliability and performance of the Book a Doctor app:

- **Unit Testing:** Individual components and functions, both frontend (React components) and backend (API controllers), were tested to verify correctness. Tools like **Jest** and **React Testing Library** were used for frontend unit tests.

- **Integration Testing:** API endpoints and database interactions were tested using tools such as **Postman** and automated test scripts with **Mocha** and **Chai** on the backend.

- **User Acceptance Testing (UAT):** End users and stakeholders tested the complete system to validate the user experience and functional requirements.

- **Performance Testing:** Load testing was conducted to ensure the system can handle concurrent users and maintain responsiveness.

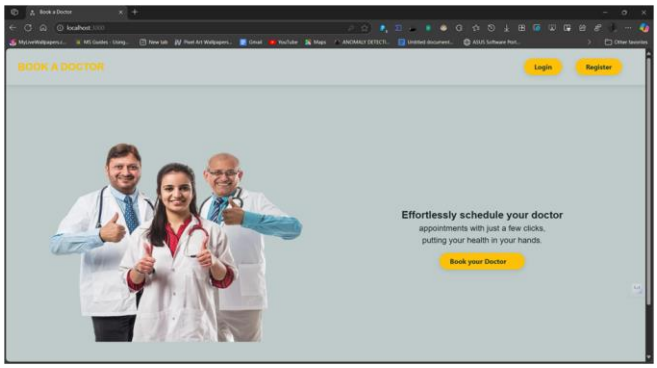
- **Security Testing:** Password encryption, authentication flows, and data transmission were tested for vulnerabilities using security audit tools.
  Here's a structured draft for sections **11**, **12**, and **13** to add to your document:

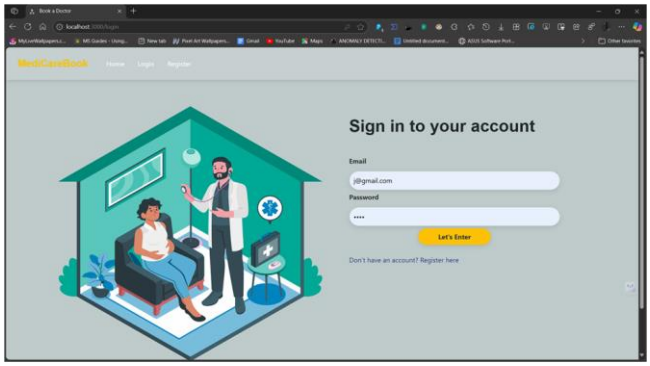
---

11. Screenshots or Demo

To provide a clear understanding of the Book a Doctor application, below are screenshots showcasing the key features of the platform:

- **Login and Registration Pages**
- **Patient Dashboard with Doctor Search and Filters**
- **Appointment Booking Form with Document Upload**
- **Doctor Dashboard showing Appointments and Status Management**





12. Known Issues

- **Appointment Rescheduling:** Currently, there is no direct option for patients to reschedule an existing appointment; they must cancel and rebook.
- **Notification Delays:** Occasional delays in email or SMS notifications have been observed under high server load.
- **File Upload Limits:** Uploaded medical documents have size restrictions that sometimes prevent larger files from being uploaded without warning.
- **Mobile Responsiveness:** Some minor UI elements require further optimization on smaller screen sizes.
- **Role Permissions:** Edge cases in role-based access control can occasionally allow restricted features to be visible but not accessible.

13. Future Enhancements

- **Appointment Rescheduling Feature:** Allow patients to directly reschedule appointments through the interface.
- **Real-Time Chat:** Integrate a secure messaging system for patients and doctors to communicate.
- **Multi-Language Support:** Add localization options to support multiple languages.

- **Advanced Analytics:** Provide doctors and admins with insights and reports on appointment trends and patient statistics.
- **Payment Integration:** Enable online payments and billing for consultation fees.
- **Mobile App:** Develop native Android and iOS applications to expand accessibility.
- **AI-based Doctor Recommendations:** Use AI to suggest doctors based on patient history and preferences.

Let me know if you want me to help create any visuals or elaborate on these sections!

-