**1. Wireframe Document**

The wireframe provides a **visual representation of the frontend components** and how they interact with the backend and database. This helps align the development team with the user experience (UX) and design elements.

**Wireframe Designs Overview**

1️. **Landing Page**

* Header: Logo, Login/Signup buttons.
* Main Section: Search doctors by specialty/location.
* Call-to-action: "Book Appointment Now" button.
* Footer: Contact, FAQs, Terms & Conditions.

2️. **Login/Signup Page**

* Input fields: Username, password, email.
* Login and Signup toggles.
* CTA: Login/Register buttons with OAuth (Google, Facebook).

3️. **User Dashboard (Patient/Doctor)**

* User Details section (Name, Role).
* List of appointments (with status).
* Button: "Book Appointment" (Patients) / "View Schedule" (Doctors).

4️. **Booking Page**

* Dropdown: Select Doctor.
* Calendar view: Choose available time slots.
* CTA: "Confirm Appointment" button.

5️. **Admin Panel Wireframe**

* Sidebar: Manage users, view appointments.
* Main Panel: List of all bookings with CRUD options.

**2. Architecture Design Document**

The architecture document gives a **comprehensive view of the technical structure**, outlining how the components (frontend, backend, database) work together to fulfill system requirements

**System Architecture Overview**

This system follows the **MERN stack architecture**:

1. **Frontend**:
   * Built using React.js.
   * Users interact with a responsive UI.
   * Uses Axios to send/receive data from the backend.
2. **Backend (API Layer)**:
   * Built with Node.js and Express.js.
   * RESTful API handles CRUD operations (appointments, users, schedules).
   * JWT Authentication ensures secure login.
3. **Database Layer**:
   * MongoDB for storing user, appointment, and doctor data.
   * Collections: Users, Doctors, Appointments, Admins.
4. **Security Layer**:
   * JSON Web Tokens (JWT) used for authentication.
   * Role-based access ensures proper permissions.

**System Architecture Diagram**

+----------------------+

|Frontend (React) |

|----------------------|

| User Dashboard |

| Booking Page|

| Admin Panel |

+----------------------+

|

| (HTTPS Requests)

V

+------------------------------------------------+

| Backend API Layer (Node + Express) |

|------------------------------------------------|

| Authentication: JWT |

| RESTful APIs: User, Appointment, Doctor CRUD |

|------------------------------------------------|

| Business Logic |

+------------------------------------------------+

|

| (Database Queries)

V

+------------------+

| MongoDB Atlas |

|------------------|

| Collections: |

| - Users |

| - Doctors |

| - Appointments |

+------------------+

**Component Responsibilities**

1. **Frontend Components**:
   * Handle UI and user interactions.
   * Send API requests to fetch/post data.
   * Manage session using JWT stored in local storage.
2. **Backend Components**:
   * Validate incoming requests (authentication).
   * Manage business logic (appointment creation, updates).
   * Handle role-based access for users, doctors, and admins.
3. **Database Components**:
   * Store structured data in collections.
   * Maintain relationships between appointments, users, and doctors.

**Data Flow in the System**

1. **Login Flow**:
   * User submits credentials → Backend validates → JWT generated and stored in local storage.
2. **Appointment Booking Flow**:
   * User selects a doctor → Backend checks availability → Appointment confirmed and stored in MongoDB → Notification sent.
3. **Admin Panel Operations**:
   * Admin retrieves user and appointment data → Makes necessary updates → Changes reflected instantly in the frontend.

**Conclusion**

This architecture provides a **scalable, secure, and modular** design for a doctor appointment booking system. It ensures smooth interaction between the frontend, backend, and database, maintaining data integrity and system performance.

This document serves as a blueprint for developers, ensuring **smooth coordination** throughout the project lifecycle.