**High-Level Design (HLD) Document**

**Purpose**: This document gives an **architectural overview** of the solution, including technologies, data flow, and core modules.

**1. System Architecture Overview**

* **Architecture Type**: MERN Stack (MongoDB, Express, React, Node.js)
* **Deployment Model**: Cloud-based using services like **Heroku** or **AWS**.
* **Components**:
  + **Frontend**: React.js (for user interface)
  + **Backend**: Node.js & Express.js (for APIs)
  + **Database**: MongoDB Atlas (for data storage)
  + **Authentication**: JSON Web Tokens (JWT)
  + **Admin Panel**: Role-based access using React.js components

**2. Key Modules and Functionalities**

1️. **User Management Module**

* Registration, Login (with JWT Authentication)
* Roles: Patient, Doctor, Admin

2️. **Appointment Management Module**

* View doctor schedules
* Book, Reschedule, or Cancel Appointments

3️. **Admin Module**

* Manage Users and Appointments
* Monitor overall system status

4️. **Notification Module**

* Email and SMS notifications (for booking confirmation)

**3. Data Flow Diagram (DFD)**

**Level 1 DFD** (Abstract Overview):

* **Users** (Patients/Doctors/Admins) interact with the **Frontend**, which sends requests to the **Backend API**.
* **Backend** communicates with **MongoDB** to fetch or store data, and the **Admin Panel** offers advanced control.

**4. System Security**

* **Authentication**: JWT-based login for secure sessions
* **Data Encryption**: Use HTTPS for communication
* **Role-Based Access Control**: Ensure only authorized users access critical modules

**Low-Level Design (LLD) Document**

**Purpose**: This document offers **detailed descriptions** of each component, class, and data flow.

**1. Database Design (ER Diagram)**

**Entities:**

* **Users** (UserID, Name, Email, Role, Password)
* **Doctors** (DoctorID, Specialty, AvailableSlots)
* **Appointments** (AppointmentID, PatientID, DoctorID, TimeSlot, Status)

**Relationships:**

* Users and Appointments: One-to-Many (A user can have multiple appointments)
* Doctors and Appointments: One-to-Many

**2. Backend API Design**

| **API Endpoint** | **Method** | **Description** | **Authentication** |
| --- | --- | --- | --- |
| /api/login | POST | User login | No |
| /api/register | POST | User registration | No |
| /api/appointments | GET | Fetch all appointments | Yes (Admin) |
| /api/appointments/ | GET | Fetch specific appointment details | Yes |
| /api/doctors/schedule | GET | Fetch doctor schedules | Yes |

**3. Class Diagram**

* **User Class**:

class User:

def \_\_init\_\_(self, user\_id, name, email, password, role):

self.user\_id = user\_id

self.name = name

self.email = email

self.password = password

self.role = role

* **Appointment Class**:

class Appointment:

def \_\_init\_\_(self, appointment\_id, patient\_id, doctor\_id, time\_slot, status):

self.appointment\_id = appointment\_id

self.patient\_id = patient\_id

self.doctor\_id = doctor\_id

self.time\_slot = time\_slot

self.status = status

**4. Frontend Design (Component Hierarchy)**

1️. **Landing Page Component**: Introduction and login/signup options  
2️. **Dashboard Component**: User-specific dashboard (doctor or patient view)  
3️. **Booking Component**: Form for selecting doctor, time slot, and confirming the appointment  
4️. **Admin Panel Component**: Display for managing appointments and users

**5. Workflow Diagram (Sequence)**

**User Flow**:

1. A patient logs in and views available doctors and their schedules.
2. The patient selects a doctor and books a time slot.
3. The backend validates the request, stores the appointment in the database, and triggers a confirmation email.
4. Doctors can view their schedules through their dashboard.
5. Admins have access to manage users and appointments via the admin panel.

**6. Authentication Workflow**

1. **Login API** generates a **JWT token** upon successful user validation.
2. Token is stored in **local storage** on the frontend.
3. Each subsequent request from the frontend attaches the JWT for authorization.
4. If the token expires or is invalid, the system redirects the user to the login page.

**7. Error Handling**

* **Form Validations**: Ensure all required fields are filled and properly formatted.
* **API Errors**: Use appropriate status codes (e.g., 401 for unauthorized access, 404 for not found).
* **Graceful Failures**: Inform users with friendly messages if the system encounters any issues.

**8. Deployment Strategy**

* **Frontend**: Deployed using **Netlify** or **Vercel**.
* **Backend**: Hosted on **Heroku** or **AWS Lambda**.
* **Database**: Managed via **MongoDB Atlas**.

**9. Performance Optimization**

* **Caching**: Use **Redis** to cache frequently accessed data.
* **Pagination**: Implement pagination for large datasets (e.g., appointment history).
* **Database Indexing**: Add indexes to improve query performance.

**10. Testing Strategy**

* **Unit Testing**: Test individual components and APIs using **Jest** or **Mocha**.
* **Integration Testing**: Test end-to-end functionality across components.
* **User Acceptance Testing (UAT)**: Ensure the system meets user requirements.

This HLD and LLD combination offers a **comprehensive solution** for building and deploying the **Doctor Appointment Booking System**. These documents provide clarity on the project’s overall architecture, data flow, backend logic, frontend integration, and deployment strategy, helping ensure smooth development and collaboration.