# Hospital Management System (MERN)

## 1. Objective

Build a **Full Stack Hospital Management System** using the **MERN Stack** (MongoDB, Express, React/Next.js, Node.js). The system should handle the end-to-end flow of patient registration, appointment booking, doctor checkups, and lab report management.

## 2. Tech Stack Requirements

Based on your profile, the following stack is mandatory:

### Frontend

- **Framework:** Next.js (App Router preferred) or React.js (Vite)
- **Styling:** Tailwind CSS or Material UI (MUI)
- **State Management:** Redux Toolkit or Context API
- **Data Fetching:** Axios or TanStack Query (React Query)

### Backend

- **Runtime:** Node.js
- **Framework:** Express.js
- **Language:** TypeScript
- **Database:** MongoDB
- **ODM: Mongoose** (Strictly no Prisma)
- **Architecture:** Microservices (preferred) or Modular Monolith

## 3. Architecture Overview

The system should consist of a backend API layer and a frontend client.

- **Backend:** Divided into independent services (or modular routes) handling specific domains (Auth, Patient, Doctor, Appointment).
- **Frontend:** A responsive web application with role-based access (Patient, Doctor, Lab Staff).
- **Communication:** Frontend consumes Backend REST APIs.

## 4. Frontend Modules (New)

### A. Public / Patient Portal

1. **Landing Page:** Basic information about the hospital.

2. **Authentication:** Login/Signup for Patients.
3. **Patient Dashboard:**
    - View profile & medical history.
    - **Book Appointment:** Select Doctor -> Select Date/Slot -> Confirm.
    - **My Appointments:** View upcoming and past appointments.
    - **Reports:** Download/View lab reports uploaded by the lab team.

### B. Doctor Dashboard

1. **Login:** Secure login for doctors.
2. **OPD Queue Management:**
    - View list of patients assigned for the current day.
    - "Call Next Patient" feature (Update status to 'In Progress').
3. **Checkup Interface:**
    - View selected patient's details.
    - Input **Diagnosis** and **Doctor Notes**.
    - **Prescribe Medicine:** Add medicines (Name, Dosage, Duration) dynamically.
    - **Recommend Tests:** Select required lab tests (e.g., Blood Test, X-Ray).
    - "Complete Checkup" button.

### C. Lab & Admin Panel

1. **Pending Tests:** View list of patients requiring lab tests.
2. **Upload Report:** Form to upload test results/PDFs for a specific patient/appointment.

## 5. Backend Services & Business Logic

**Note:** Use **Mongoose Schemas** for data modeling.

### Service 1: Auth & User Service

- **Responsibilities:** Handle Registration and Login (JWT) for Patients, Doctors, and Staff.
- **Mongoose Models:** `User`, `Role`.

### Service 2: Patient Service

- **Responsibilities:** CRUD operations for patient profiles.
- **Fields:** Name, Age, Gender, Contact, Medical History.

### Service 3: Doctor Service

- **Responsibilities:** Manage doctor profiles and availability.
- **Fields:** Name, Specialization, OPD Timings, Status (Active/Inactive).

### Service 4: Appointment Service

- **Responsibilities:**
  - Book appointments.
  - Generate **Token Numbers**.
  - Manage status (Pending, Completed, Cancelled).
- **Logic:** Ensure no double booking for the same slot.

### Service 5: Medical Record Service (Checkup & Lab)

- **Responsibilities:**
  - Store checkup details (Symptoms, Diagnosis).
  - Store Lab Test requests and Results.
  - Store Prescriptions.
- **Relations:** Must link `PatientId`, `DoctorId`, and `AppointmentId`.

## 6. Core User Flow (End-to-End)

1. **Registration:** Patient creates an account on the frontend.
2. **Booking:** Patient books an appointment; System generates a Token (e.g., Token #12).
3. **OPD:** Doctor logs in, sees Token #12 in the queue, and starts the checkup.
4. **Prescription:** Doctor adds "Paracetamol" and requests a "Blood Test".
5. **Lab:** Lab staff sees the request, performs the test, and uploads the result.
6. **Review:** Patient logs in and views the Prescription and Lab Report.

## 7. Code Quality & Deliverables

### Requirements

1. **Type-Safety:** Use TypeScript Interfaces for all Mongoose models and API responses.
2. **Component Design:** Reusable frontend components (e.g., Buttons, Form Inputs, Tables).
3. **Validation:**
   - **Backend:** Joi or Zod (for API inputs).
   - **Frontend:** React Hook Form or Formik.
4. **Error Handling:** Proper error messages on frontend (e.g., "Slot already booked") and backend (try-catch).

### Deliverables

1. **GitHub Repository:** Monorepo (client/server folders) or separate repos.
2. **README.md:**
   - Setup instructions (`npm install`, `.env` setup).
   - Screenshots of the application.
3. **Environment Setup:** Provide a `.env.example` file.

## Bonus Points (Optional)

- **Docker:** Dockerfile for Backend and Frontend.
- **Real-time Updates:** Use Socket.io to update the Doctor's queue instantly when a patient books.

---

## Evaluation Criteria

- **Full Stack Integration:** How well the Frontend talks to the Backend.
- **UI/UX:** Cleanliness and responsiveness of the dashboard (using Tailwind/MUI).
- **Code Structure:** Folder structure, separation of concerns (Controllers vs Services vs UI Components).
- **Mongoose Usage:** Proper schema design and relationship handling (populate, references).