Hospital Management System

Introduction

This document defines the design and features of a **Hospital Management System (HMS)** developed using **Node.js**, **Express.js**, and **MongoDB**.
The system is a **backend-only application** implementing **role-based authentication** and **management of hospital workflows**, including doctors, staff, appointments, and patients.

User Roles & Features

Admin

- Create, update, and delete **departments**.
- Create, update, and delete **staff**, **doctor**, and **sub-admin accounts**.
- Assign multiple doctors to departments with available days and time slots.
- View **hospital statistics** such as appointments by department, missed vs attended, etc.

Staff

- Login using credentials provided by Admin.
- Book appointments for patients with specific doctors.
- If a slot is full, create an **appointment request** in the next available slot.
- Update appointment status (**Attended**, **Missed**).
- View **today's schedules** for all doctors.

Doctor

- Login with credentials created by Admin.
- View **today's appointments** with patient details and timings.

- Access **monthly statistics** of appointments.
- View **historical data** of patients treated.

User (Patient)

- Register and login through the system.
- Book appointments with specific doctors.
- View **appointment history** (past and upcoming).

Aggregation Pipeline Use Cases

- **Doctors Monthly Appointment Report:** Count appointments grouped by month.
- **Admin Dashboard:** Appointments grouped by department.
- **Staff Daily Schedule:** Joined data of doctors and patients for today.
- **Missed vs Attended Appointment Report:** For hospital statistics.
- **User Appointment History:** With doctor and department details.

Database Design (Collections)

Users

Stores patient details and credentials.
Fields: name, email, password, age, gender, contact, address.

Doctors

Stores doctor details.
Fields: name, specialization, departmentId, availableDays, timeSlots, status.

Departments

Stores hospital departments.
Fields: name, description, createdBy.

Staff

Stores staff login credentials and details.
Fields: name, email, password, departmentId, role.

Appointments

Stores booking and status data.
Fields: doctorId, userId, date, time, status (Booked, Attended, Missed), createdBy.

Workflow Summary

1. **Admin** manages departments, doctors, and staff accounts.
2. **Users** register and book appointments.
3. **Staff** manages appointments, handles schedules, and updates statuses.
4. **Doctors** view daily schedules and appointment history.
5. **Admin** and **Staff** use aggregation pipelines for reports and analytics.

Cron Job Implementation

Purpose:
Cron Jobs are automated background tasks that execute on a scheduled basis to ensure data accuracy and automate hospital processes.

Example Use Cases:

1. **Appointment Status Update:**
   Automatically mark appointments as “Missed” if not attended by the end of the day.
2. **Daily Statistics Generation:**
   Generate and store daily summaries (number of attended/missed appointments).
3. **Notification Scheduler:**
   Send reminder notifications to patients before their appointment time.
4. **Data Cleanup Tasks:**
   Remove expired appointment requests or inactive user sessions weekly.

Execution Flow:

- Cron jobs are scheduled to run at fixed intervals (e.g., every midnight, every hour).
- Each job checks the database for records meeting specific conditions (like today's unmarked appointments).
- The system automatically updates or logs new entries for analytics.

Example Schedule (Concept):

- Appointment cleanup → Every night at 12:00 AM
- Daily report generation → Every morning at 6:00 AM
- Reminder notifications → Every hour before appointments

These Cron Jobs ensure automation, consistency, and smooth daily hospital operations.

Mongoose Index Usage

Purpose:
Mongoose Indexes are used to **speed up query performance** and **optimize data lookups**, especially in collections that handle large datasets such as appointments and users.

Where to Use Indexes:

1. **Users Collection**
   - **Fields:** `email` (unique index)
   - **Reason:** To ensure each user has a unique email and for fast login lookups.
2. **Doctors Collection**
   - **Fields:** `departmentId`, `specialization`
   - **Reason:** To quickly fetch doctors by department or specialization for appointments.
3. **Appointments Collection**
   - **Fields:** `doctorId`, `userId`, `date`, `status`

    - **Reason:**
        - Optimize searches for today's appointments or a doctor's schedule.
        - Speed up aggregation reports and analytics queries.

4. **Staff Collection**
    - **Fields:** `email`
    - **Reason:** For quick authentication and unique account validation.
5. **Departments Collection**
    - **Fields:** `name`
    - **Reason:** Prevent duplicate department names and allow fast lookups.

Types of Indexes Suggested:

- **Unique Index:** For `email` and `department name`.
- **Compound Index:** On `{ doctorId, date }` in appointments for faster schedule lookups.
- **Text Index:** On `doctor name` or `specialization` for search functionalities.

Benefits:

- Faster query response time.
- Improved performance for aggregation pipelines.
- Efficient filtering and reporting.