

# Hospital OPD Management System — Project Report

**Date** 04 Jan 2026

**Scope** React frontend + Spring Boot backend + MongoDB

---

## 1. Overview

The **Hospital OPD Management System** is a full-stack web application built to support outpatient department (OPD) workflows: patient registration, doctor management, and appointment booking.

The system is split into two deployable parts:

- **Backend:** Spring Boot REST API with MongoDB persistence.
- **Frontend:** React single-page application (SPA) using React Router.

Primary goals:

- Maintain a clean record of **Patients**, **Doctors**, and **Appointments**.
- Provide a simple UI for staff to add/list/delete records and book appointments.
- Track doctor availability via a three-state status model: **active**, **busy**, **inactive**.

## 2. Technology Stack

### Backend

- Java 21
- Spring Boot (Web, Data MongoDB, Security)
- Bean Validation (Jakarta Validation)
- MongoDB

### Frontend

- React + React Router
- Axios
- Tailwind (configured) + app CSS

## 3. Repository Structure (High Level)

- backend/: Spring Boot project
  - src/main/java/com/hospital/opd/: controllers, services, repositories, models
  - src/main/resources/application.properties: MongoDB + server port
- opd-frontend/: React application
  - src/pages/: Patients, Doctors, Appointments, Auth
  - src/services/: Axios service modules

## 4. Functional Modules

### 4.1 Authentication

The UI provides Login and Register pages. Route protection is handled by a token gate in the client (a token stored in `localStorage`).

Endpoint	Description
POST /api/auth/register	Registers a new user
POST /api/auth/login	Returns a token string

### 4.2 Patient Management

Endpoint	Description
GET /api/patients	Lists patients
POST /api/patients	Creates a patient
DELETE /api/patients/{id}	Deletes a patient

### 4.3 Doctor Management

Doctors can be created, listed, filtered by status, deleted, and manually set inactive/active.

Endpoint	Description
GET /api/doctors	Lists all doctors (with computed status)
GET /api/doctors/{id}	Gets doctor by id
GET /api/doctors/status/{status}	Filters doctors by stored status value
GET /api/doctors/specialization/{specialization}	Filters doctors by specialization
POST /api/doctors	Creates doctor (validates phone)
PUT /api/doctors/{id}	Updates doctor (supports manual status)
DELETE /api/doctors/{id}	Deletes doctor

### 4.4 Appointment Management

Appointments are booked with a selected patient, selected doctor, and date/time. The backend prevents booking an appointment in the past.

Endpoint	Description
POST /api/appointments	Books appointment (sets status=booked)
GET /api/appointments	Lists all appointments

Endpoint	Description
GET /api/appointments/patient/{patientId}	Appointments for a patient
GET /api/appointments/doctor/{doctorId}	Appointments for a doctor
PUT /api/appointments/{id}/complete	Marks completed
PUT /api/appointments/{id}/cancel	Marks cancelled

## 5. Data Storage (MongoDB)

Database name: hospital\_db

Collections:

- patients
- doctors
- appointments
- users

Spring Data Mongo repositories (MongoRepository) provide CRUD access.

## 6. Doctor Status Rules (Active / Busy / Inactive)

The system supports three statuses:

- **inactive**: doctor is not available right now
- **busy**: doctor has an upcoming booked appointment
- **active**: doctor is available and not booked

Rules used by the backend when returning doctors:

1. If the doctor is manually set to *inactive*, the API keeps it *inactive*.
2. Otherwise, if current time is outside *availableTime*, the API returns *inactive*.
3. Otherwise, if any upcoming appointment with status *booked* exists, the API returns *busy*.
4. Otherwise, the API returns *active*.

Available time parsing supports examples like 9:00 AM - 5:00 PM and 09:00 - 17:00, including overnight ranges.

## 7. Input Validation (Doctor Phone)

Doctor phone numbers are enforced as **exactly 10 digits**.

- Frontend: digits-only input, max length 10, regex validation.
- Backend: Bean Validation ensures requests cannot store invalid phone numbers.

## 8. Configuration & Runtime

- Backend port: 8081
- MongoDB: mongodb://localhost:27017/hospital\_db
- Frontend: http://localhost:3000

## 9. Security Notes

The backend includes Spring Security but currently permits most requests. The frontend applies a token gate, but server-side JWT authorization is not enforced.

Recommendation: add a JWT authentication filter and protect API routes server-side.

## 10. Known Limitations & Suggested Enhancements

- “Busy” is currently computed as existence of upcoming booked appointments, not time-overlap per slot.

- For production use, implement server-side authorization and token verification.
- Consider adding edit/update pages for patients and doctors.

## 11. How to Run

### Prerequisites

- MongoDB running locally
- Node.js + npm
- Java 21

### Backend

From backend/:

```
./mvnw.cmd spring-boot:run
```

### Frontend

From opd-frontend/:

```
npm install then npm start
```

---

End of report.