Hospital OPD Management System — Total Project Report

Date: 04 Jan 2026
Includes: Workflow explanation + key code excerpts
Stack: React + Spring Boot + MongoDB

1. Project Objective

This project implements a basic OPD (Out-Patient Department) management system for: **Patient records**, **Doctor records**, and **Appointment booking**.

It provides a React UI and a Spring Boot REST API backed by MongoDB.

2. High-Level Architecture

Frontend (React) → Axios HTTP calls → **Backend (Spring Boot REST)** → Spring Data Repositories → **MongoDB**

Frontend runs on `http://localhost:3000`. Backend runs on `http://localhost:8081`.

3. Main User Workflow (End-to-End)

3.1 Login / Register

- User registers using the Register page → calls `POST /api/auth/register`.
- User logs in using Login page → calls `POST /api/auth/login` → backend returns a token string.
- Frontend stores token in `localStorage`. `ProtectedRoute` checks token before allowing access to app routes.

3.2 Patient Management

- Patients page loads by calling `GET /api/patients`.
- Add Patient form validates client-side (including 10-digit mobile) and calls `POST /api/patients`.
- Delete patient calls `DELETE /api/patients/{id}` and updates UI state.

3.3 Doctor Management

- Add Doctor validates phone number as exactly 10 digits, then calls `POST /api/doctors`.
- Doctors list calls `GET /api/doctors` and shows status badges and filtering.
- Status can be set manually from UI to `inactive` or back to `active` using `PUT /api/doctors/{id}`.

3.4 Appointment Booking

- Booking screen loads patients and doctors.
- User selects patient, doctor, and datetime.
- Frontend prevents selecting past dates; backend also rejects past times.
- Booking calls `POST /api/appointments` and sets appointment status to `booked`.

4. Backend Workflow (Spring Boot)

4.1 Application Entry

The application starts via Spring Boot auto-configuration:

```
package com.hospital.opd;

@SpringBootApplication
public class HospitalOpdApplication {
  public static void main(String[] args) {
    SpringApplication.run(HospitalOpdApplication.class, args);
  }
}
```

4.2 Controllers → Services → Repositories

Controllers expose REST endpoints. Controllers call Services. Services call Spring Data Repositories. Mongo repositories persist and query data.

4.3 Authentication Flow (Register + Login)

AuthController receives requests and calls **AuthService**. Passwords are hashed using BCrypt. Login returns a JWT token.

```
// AuthServiceImpl (key logic)
@Override
public User register(User user) {
  user.setPassword(passwordEncoder.encode(user.getPassword()));
  return userRepository.save(user);
}

@Override
public String login(String username, String password) {
  User user = userRepository.findByUsername(username)
    .orElseThrow(() -> new RuntimeException("User not found"));

  if (!passwordEncoder.matches(password, user.getPassword())) {
    throw new RuntimeException("Invalid password");
  }

  return jwtUtil.generate(username, user.getRole());
}
```

Note: the backend currently permits most requests; the token is used mainly on the frontend route gate.

4.4 Patient CRUD Flow

Patient endpoints call `PatientServiceImpl` which delegates to `PatientRepository`.

```
// PatientServiceImpl
@Override
public List<Patient> getAllPatients() {
  return patientRepository.findAll();
}
```

```
@Override
public Patient savePatient(Patient patient) {
  return patientRepository.save(patient);
}

@Override
public void deletePatient(String id) {
  patientRepository.deleteById(id);
}
```

4.5 Appointment Booking Validation

Booking prevents invalid appointments server-side (time required and cannot be in the past).

```
// AppointmentServiceImpl.bookAppointment
if (appointment.getAppointmentTime() == null) {
  throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Appointment time is required");
}
if (appointment.getAppointmentTime().isBefore(LocalDateTime.now())) {
  throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Appointment time cannot be in the pas
}
appointment.setStatus("booked");
return appointmentRepository.save(appointment);
```

4.6 Doctor Phone Validation (Backend)

Doctor phone is enforced as exactly 10 digits using Bean Validation. When controller uses `@Valid`, invalid requests return 400.

```
// Doctor model (excerpt)
@NotBlank(message = "Phone number is required")
@Pattern(regexp = "^\\d{10}$", message = "Phone number must be exactly 10 digits")
private String phone;
```

4.7 Doctor Status Workflow (Active / Busy / Inactive)

Status is computed when returning doctors:

- If doctor is manually set to `inactive`, it stays `inactive`.
- Else if current time is outside `availableTime` → `inactive`.
- Else if any upcoming appointment with status `booked` exists → `busy`.
- Else → `active`.

```
// DoctorServiceImpl.applyComputedStatus (simplified excerpt)
if ("inactive".equalsIgnoreCase(doctor.getStatus())) {
  doctor.setStatus("inactive");
  return;
}

if (!isCurrentlyWithinAvailableTime(doctor.getAvailableTime())) {
  doctor.setStatus("inactive");
  return;
}

boolean hasBookedAppointment = appointmentRepository
  .findByDoctorId(doctor.getId())
  .stream()
  .anyMatch(a -> "booked".equalsIgnoreCase(a.getStatus())
    && a.getAppointmentTime() != null
    && !a.getAppointmentTime().isBefore(LocalDateTime.now()));

doctor.setStatus(hasBookedAppointment ? "busy" : "active");
```

5. Frontend Workflow (React)

5.1 Routing + Route Protection

React Router defines public auth routes and protected application routes. `ProtectedRoute` checks token in localStorage.

```
// ProtectedRoute
const token = localStorage.getItem("token");
if (!token) {
  return <Navigate to="/login" replace />;
}
return children;
```

5.2 API Services (Axios)

The frontend uses service modules wrapping Axios calls.

```
// doctorService base URL
const BASE_URL = 'http://localhost:8081/api/doctors';

// addDoctor
const response = await axios.post(BASE_URL, doctor);
return response.data;
```

5.3 Add Doctor — Phone Validation (UI)

The Add Doctor form filters to digits only and limits to 10 digits. Submit validation enforces exactly 10 digits.

```
// AddDoctor.handleChange (phone)
if (name === 'phone') {
  const digitsOnly = value.replace(/\D/g, '').slice(0, 10);
  setFormData(prev => ({ ...prev, [name]: digitsOnly }));
  return;
}

// validateForm
if (!/^\d{10}$/.test(formData.phone)) {
  setError('Phone number must be exactly 10 digits');
  return false;
}
```

5.4 Doctor List — Manual Inactive / Active Toggle

The Doctors list includes buttons to set a doctor inactive or active (writes status back via API).

```
// DoctorList.handleStatusChange
const updated = await doctorService.updateDoctor(id, { status });
setDoctors(prev => prev.map(d => (d.id === id ? updated : d)));
```

5.5 Appointment Booking Screen

Booking screen loads doctors and patients, and shows doctor status in the dropdown.

```
// Doctor option label
{doctor.name} - {doctor.specialization} ({doctor.status})
```

6. How to Run (Local)

- MongoDB: `mongodb://localhost:27017` (DB: `hospital_db`)
- Backend: run from `backend/` using `./mvnw.cmd spring-boot:run`
- Frontend: run from `opd-frontend/` using `npm start`

7. Notes About “Full Code in PDF”

A full repository code dump would create a very large PDF. This report includes the most important workflow code excerpts (controllers/services/pages) and the rest of the complete source code is already available in the project folder.

End of total project report.