Hospital Management System (HMS) - Business Requirements Document

Executive Summary

A multi-tenant, cloud-enabled Hospital Management System enabling hospitals to self-register, manage operations, and maintain complete patient lifecycle management with role-based access control.

1. Product Overview

1.1 Vision

Build a comprehensive, scalable, multi-tenant SaaS platform that enables hospitals of any size to digitize their operations with zero upfront infrastructure investment.

1.2 Objectives

- Enable hospitals to self-onboard and become operational within 24 hours.
- Enable a multi-tenant solution so that data of each hospital (tenant) is private to them without need of separate infrastructure.

1.3 Target Users

- **Hospital Administrators:** Onboarding, configuration, user management
- **Doctors:** Patient management, prescriptions, diagnostics
- **Nurses:** Patient care, vital monitoring
- **Pharmacists:** Prescription fulfillment
- **Receptionists:** Appointment and patient registration handling

2. Functional Requirements

2.1 Multi-Tenancy & Hospital Onboarding

FR-1: Hospital Self-Registration

Description: Hospitals can register themselves without manual intervention.

Acceptance Criteria:

- Hospital provides: Name, Address, Contact Details, Admin Email, Phone, License Number
- System validates license number uniqueness
- Auto-generates tenant ID (UUID-based)
- Creates isolated database schema per tenant
- Sends verification email with activation link
- Admin credentials created automatically (username: admin@{hospital-domain})

Business Rules:

- License number must be unique across platform
- Email verification mandatory before activation
- Hospital status flow: PENDING → VERIFIED → ACTIVE → SUSPENDED → INACTIVE

FR-2: Tenant Isolation

Acceptance Criteria:

- Each tenant has dedicated database schema (schema-per-tenant approach)
- Redis namespacing: tenant:{tenantId}:*
- Shared infrastructure, isolated data
- Cross-tenant data access prevention via middleware (multi-tenancy context adapter)
- Tenant context extracted from JWT token

2.2 Authentication & Authorization

FR-3: OAuth2 Authentication

Description: Support OAuth2 with JWT tokens for user authentication.

Acceptance Criteria:

- Grant Types Supported: Authorization Code, Password, Refresh Token
- Token Structure: Access Token (1 hour expiry), Refresh Token (7 days expiry)
- Token contains: userId, tenantId, roles, permissions

FR-4: Role-Based Access Control (RBAC)

Description: Hierarchical role-based permission system.

Pre-defined Roles:

Role	Description	Default Permissions
SUPER_ADMIN	Platform administrator	All system operations
HOSPITAL_ADMIN	Hospital administrator	Tenant configuration, user management
DOCTOR	Medical practitioner	Patient management, prescriptions
NURSE	Nursing staff	Patient care, vitals
PHARMACIST	Pharmacy staff	Prescription view, dispensing
RECEPTIONIST	Front desk	Patient registration, appointments

Acceptance Criteria:

- Admin can create custom roles
- Roles have many-to-many relationship with permissions
- Users can have multiple roles
- Permission format: RESOURCE:ACTION (e.g., PATIENT:CREATE, PRESCRIPTION:READ)
- Hierarchical inheritance: HOSPITAL_ADMIN inherits all lower-level permissions

---

FR-5: Attribute-Based Access Control (ABAC)

Description: Fine-grained access control based on attributes.

Attributes:

- User Attributes: department, specialization, shift
- Resource Attributes: patient_department, confidentiality_level

Use Case Example:

- A doctor can only view patients assigned to their department.

2.3 User Management

FR-6: User Registration & Management

Acceptance Criteria:

- Hospital admin can create users
- Required fields: firstName, lastName, email, phone, department, roles
- Auto-generate username: {firstName}.{lastName}@{hospitalDomain}
- Password policy: Min 8 chars, 1 uppercase, 1 lowercase, 1 number, 1 special char
- Send welcome email with temporary password
- User status: ACTIVE, INACTIVE, LOCKED, PASSWORD_EXPIRED

FR-7: Password Management

Forgot Password:

- User requests reset via email
- System generates reset token (valid for 1 hour)
- Email sent with reset link
- User sets new password

- All active sessions invalidated

Reset Password:

- User must be authenticated
- Must provide old password
- New password cannot be same as last 3 passwords
- Password history maintained

Force Password Change:

- Admin can force password change
- User redirected to reset screen on next login

2.5 Patient Management

FR-8: Patient Registration

Acceptance Criteria:

- Register patient with: Name, DOB, Gender, Blood Group, Contact, Address, Emergency Contact
- Generate unique Patient ID: {tenantId}-P-{sequential}
- Support photo upload (max 5MB, JPG/PNG)
- Mark as OPD or IPD

Patient Types:

- OPD (Outpatient): Consultation, diagnosis, prescription
- IPD (Inpatient): Admission, treatment, discharge summary

FR-9: Patient Search & Listing

Acceptance Criteria:

- Search by: Patient ID, Name, Phone, Email
- Filters: Patient Type, Department, Doctor, Date Range
- Pagination: 20 records per page
- Export to CSV/PDF

2.6 Prescription Management

FR-10: Prescription Creation

Acceptance Criteria:

- Doctor can create prescription for patient
- Medicine details: Name, Dosage, Frequency, Duration, Instructions
- Support for multiple medicines in single prescription
- Template support for common prescriptions
- Prescription ID: {tenantId}-RX-{sequential}

2.7 Dynamic Menu & UI

FR-11: Role-Based Menu Rendering

Acceptance Criteria:

- Menu structure based on user's roles and permissions
- Hierarchical menu: Module → Feature → Action
- Hide/disable menu items if user lacks permission
- Frontend permission validation
- Backend API-level permission validation

Sample Menu Structure:

```
{
  "Dashboard": {
    "permission": "DASHBOARD:VIEW",
    "icon": "dashboard"
  },
  "Patients": {
    "permission": "PATIENT:READ",
    "icon": "people",
    "children": {
      "Register Patient": {"permission": "PATIENT:CREATE"},
      "OPD Patients": {"permission": "PATIENT:READ"},
      "IPD Patients": {"permission": "PATIENT:READ"}
    }
  }
}
```

---

3. Technical Architecture

3.1 Architecture Overview

The system follows a **modular monolithic architecture**, designed for flexibility and scalability. Each functional area — Authentication, User Management, Patient Management, and Prescription Management — operates as a distinct module within a unified application context.

This architecture ensures:

- Better maintainability and performance
- Easier deployment compared to distributed microservices
- Tenant-based data isolation through middleware
- Centralized authentication and role-based authorization
- Use of Redis for caching and session optimization

3.2 Technical Stack

Backend Options:

Option 1: Java-based Stack

- Java 17, Spring Boot 3.2.x
- Spring Cloud (Gateway, Config, Eureka, Sleuth)
- Spring Security + OAuth2 Resource Server
- MySQL or PostgreSQL
- Redis 7.x (Caching, Session Management)

Option 2: Node.js-based Stack

- Node.js (v20+)
- Express.js (RESTful API framework)
- MongoDB (Atlas) with Mongoose ORM (schema-per-tenant)
- Redis (v7.x) for caching and session management
- JWT + Passport.js for authentication and authorization
- Nodemailer for email notifications
- Winston / Morgan for logging
- Docker for containerization and deployment

Frontend Options:

Option 1: Angular-based Stack

- Angular 17+
- TypeScript
- Angular Material / PrimeNG

Option 2: React-based Stack

- React (v18+)
- Vite or Create React App
- React Router for navigation
- Redux Toolkit / Context API for state management
- Material UI (MUI) for UI components
- Axios for API communication

Other Components:

- Nginx as reverse proxy and static asset server
- GitHub Actions / Jenkins for CI-CD pipelines
- Docker Compose or Kubernetes for container orchestration

Excellent — that's a strong and clear instruction Here's a refined and professional version of your statement that you can include in your **team or project guidelines**, **assignment briefs**, or **evaluation instructions**:

Instruction to Participants / Developers

> You are allowed to use **AI tools** (such as ChatGPT, GitHub Copilot, or other code assistants) to support your development process.
>
> However, you must **not simply copy-paste** AI-generated code or content.
>
> You are expected to:
>
> - **Understand every line** of the code or content you include in your project.
> - Be able to **explain the logic, flow, and purpose** of all functionalities implemented.
> - **Debug, extend, or refactor** your work independently if needed.
> - Use AI **responsibly** — as a tool for learning, ideation, and assistance, not as a shortcut to submission.
> - Ensure that the final implementation reflects your own understanding and reasoning.
>
> ⚠ **Note:** During evaluation or review, you may be asked to **demonstrate and explain** specific parts of your work to confirm ownership and comprehension. Failure to do so will lead to disqualification

---