SOFTWARE DESIGN DESCRIPTION (SDD)

Project: Hospital Appointment Management System

Author: Breton Gaspard, Coste Thomas, Ceza Mathis, Cazac Florian

Course: Software Engineering

TABLE DES MATIERES

1. Introduction 2
2. System Scope 2
3. References and Definitions 2
4. Architectural Overview 3
5. Detailed Component Design 4
    5.1 Authentication Module 4
    5.2 Availability Management Module 4
    5.3 Appointment Management Module 4
    5.4 Notification Module 4
6. Data Model (Simplified Schema) 5
7. API and External Interfaces 6
8. Use Cases and Scenarios 6
9. Non-Functional Requirements 6
10. Security and Privacy 7
11. Testing and Validation 7
12. Deployment and Operations 7
13. Maintenance Plan 7

1. INTRODUCTION

Objective: This document provides a comprehensive design description for the hospital appointment management system, guiding development, testing, and maintenance activities.

Target Audience: Instructors, developers, testers, and project managers.

2. SYSTEM SCOPE

Purpose: To enable patients, practitioners, and hospital staff to book, modify, cancel, and track medical appointments efficiently.

Main Features:

- Authentication and role-based user management (patients, doctors, admin, receptionists)
- Search for available slots by specialty, doctor, date, or location
- Appointment booking (creation, confirmation via email/SMS)
- Modification and cancellation with configurable time limits
- Practitioner availability management (recurring schedules, exceptions, vacations)
- Automated notifications and reminders (24h/48h before)
- Administrative dashboard (manual booking, reports, user management)
- Basic medical record linkage (summary or attachments — optional, GDPR compliant)

Exclusions: Full medical record management, teleconsultation, and payment systems (may be added later).

3. REFERENCES AND DEFINITIONS

- **GDPR** — General Data Protection Regulation
- **WCAG** — Web Content Accessibility Guidelines

Abbreviations:

- API: Application Programming Interface
- DB: Database
- JWT: JSON Web Token

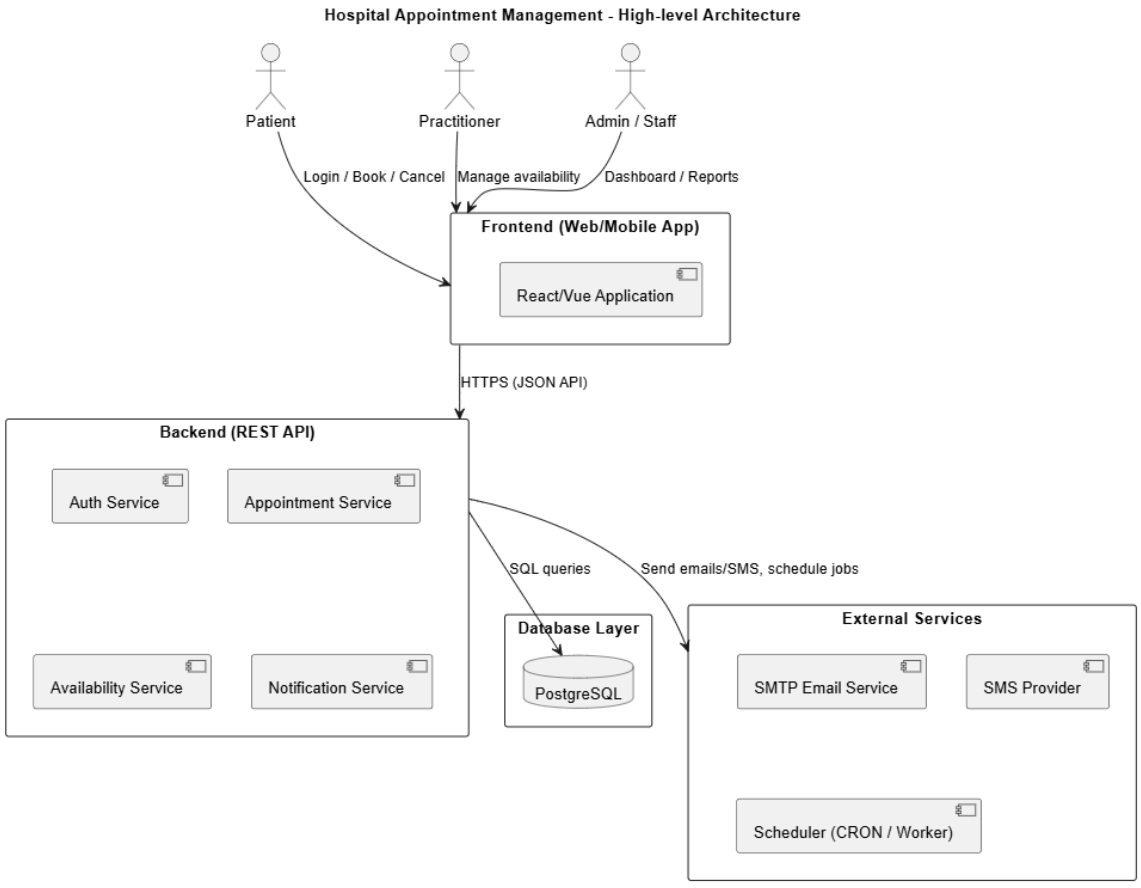
4. ARCHITECTURAL OVERVIEW

Architecture Style: 3-tier architecture (client — API — database). Future microservice evolution possible.

Main Components:

- **Frontend Web App** (React/Vue): Patient, doctor, and admin interfaces.
- **Backend API** (RESTful): Business logic, authentication, notifications.
- **Database** (PostgreSQL): User, appointment, and availability data.
- **Notification Service**: Email (SMTP) and SMS provider.
- **Scheduler**: Cron-based reminder and cleanup tasks.

High-level Architecture Diagram:



Recommended Technologies:

- Frontend: React or Vue.js
- Backend: Node.js (Express) or Python (Django/FastAPI)
- DB: PostgreSQL
- Auth: OAuth2 / JWT
- CI/CD: GitHub Actions / GitLab CI

5. DETAILED COMPONENT DESIGN

5.1 Authentication Module

Responsibilities: Registration, login, password reset, role management.
Flow:

- Registration → Email verification → Activation.
- Login → JWT generation (access + refresh tokens).

Constraints: Passwords hashed with bcrypt; lockout after multiple failed attempts.

5.2 Availability Management Module

Responsibilities: Manage time slots, recurring schedules, exceptions (vacations), default durations per specialty.

Business Rules:

- Default duration per specialty (e.g., Cardiology = 30 min)
- Recurring slots (e.g., Mondays 9am–12pm)
- Configurable buffer times between appointments.

5.3 Appointment Management Module

Responsibilities: Search, booking, modification, and cancellation.

Validations:

- Ensure slot availability upon booking (transactional check).
- Cancellation only allowed X hours before the appointment (e.g., 24h).

5.4 Notification Module

Responsibilities: Send confirmation, reminder, and cancellation messages via email/SMS.

Implementation: Message queue (RabbitMQ/Redis Streams) with background workers for sending.

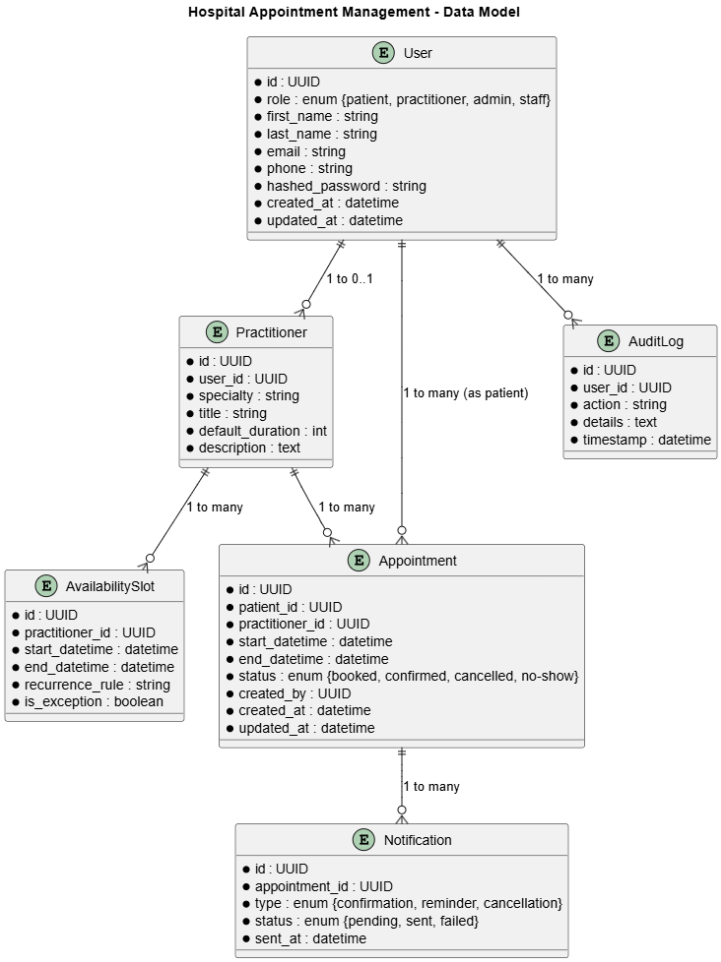
5.5 Administration Module

Responsibilities: Dashboard, CSV exports, manual appointment creation, user management.

6. DATA MODEL (SIMPLIFIED SCHEMA)

Main Tables

- **users** (id, role [patient/doctor/admin/staff], first_name, last_name, email, phone, hashed_password, created_at, updated_at)
- **practitioners** (id, user_id, specialty, title, default_duration, description)
- **availability_slots** (id, practitioner_id, start_datetime, end_datetime, recurrence_rule, is_exception)
- **appointments** (id, patient_id, practitioner_id, start_datetime, end_datetime, status [booked/confirmed/cancelled/no-show], created_by, created_at, updated_at)
- **notifications** (id, appointment_id, type, status, sent_at)
- **audit_logs** (id, user_id, action, details, timestamp)



7. API AND EXTERNAL INTERFACES

Main REST Endpoints (examples)

- POST /api/auth/register — Register patient
- POST /api/auth/login — Get JWT token
- GET /api/practitioners — List practitioners
- GET /api/practitioners/{id}/availabilities?from=...&to=... — Available time slots
- POST /api/appointments — Book appointment
- PUT /api/appointments/{id} — Modify appointment
- DELETE /api/appointments/{id} — Cancel appointment
- GET /api/dashboard/appointments — Admin/Staff dashboard

Data Format: JSON with standard error handling {code, message, details}.

Security: JWT middleware with role-based access control.

8. USE CASES AND SCENARIOS

Scenario: Book an Appointment (Patient)

1. Patient searches by specialty or doctor.
2. Frontend calls GET /practitioners/{id}/availabilities.
3. Available slots displayed.
4. Patient selects slot and confirms via POST /appointments.
5. Backend validates slot, stores record, and triggers confirmation notification.
6. Scheduler sets reminder notifications.

Scenario: Cancel Appointment

1. Patient or staff calls DELETE /appointments/{id}.
2. Cancellation rules applied.
3. Notification sent to confirm cancellation.

9. NON-FUNCTIONAL REQUIREMENTS

- **Performance:** Average response time < 300 ms under normal load.
- **Scalability:** Supports horizontal scaling with distributed notification service.
- **Availability:** Target SLA 99.5%.
- **Compatibility:** Works on modern browsers, mobile responsive.
- **Localization:** Multilingual support (EN/FR).

10. SECURITY AND PRIVACY

- **Data Minimization:** Only essential patient data stored.
- **Transport:** HTTPS enforced.
- **Storage:** Encryption at rest for sensitive fields.
- **Authentication:** JWT + refresh tokens.
- **Logging:** No sensitive data stored in logs.
- **Auditing:** Actions recorded in audit_logs.

11. TESTING AND VALIDATION

- **Unit Tests:** >70% coverage for core logic (booking, cancellation rules).
- **Integration Tests:** API-level testing (auth, booking, notifications).
- **End-to-End Tests:** User scenarios via Cypress/Playwright.
- **Load Tests:** JMeter/k6 for performance.
- **Acceptance Tests:** Example — booked appointment generates confirmation notification.

12. DEPLOYMENT AND OPERATIONS

- **Environments:** dev / staging / production.
- **CI/CD:** Automated build, test, and deploy pipelines.
- **Monitoring:** Prometheus + Grafana for metrics; Sentry for errors.
- **Backups:** Daily DB backups with restoration testing.

13. MAINTENANCE PLAN

- **Updates:** Critical dependency updates every 6 months.
- **Support:** Bug tickets handled within 48h for production blockers.
- **Logs Retention:** Access logs (90 days), audit events (1 year, configurable).