Software Requirements Specification (SRS) Document

For Hospital Management System

1. Introduction

1.1 Purpose

This document outlines the functional and non-functional requirements for the **Hospital Management System (HMS)**. It serves as a guide for developers, testers, and stakeholders to ensure the system meets all necessary specifications.

1.2 Scope

The HMS will:

- Manage patient registrations, admissions, and discharges.
- Track medical history and test requests.
- Schedule doctor appointments.
- Handle emergency cases.
- Maintain doctor and patient records.

1.3 Definitions

- **Patient**: A person receiving medical care.
- **Doctor**: A medical professional providing treatment.
- **Admission**: Process of assigning a patient to a room.
- **Discharge**: Process of releasing a patient.
- **Emergency**: A high-priority medical case.

---

2. Overall Description

2.1 System Functions

- **Patient Management**: Register, admit, discharge, and track medical history.
- **Doctor Management**: Assign doctors, schedule appointments.
- **Emergency Handling**: Prioritize urgent cases.
- **Test Management**: Request and perform medical tests.

2.2 User Classes

1. **Administrators**: Manage hospital operations.
2. **Doctors**: View appointments and patient records.
3. **Patients**: Access their medical history.

2.3 Operating Environment

- **Platform**: C++ (Console-based)
- **Dependencies**: Standard C++ libraries (<vector>, <queue>, <stack>, etc.)

---

3. Detailed Requirements

3.1 Functional Requirements

3.1.1 Patient Class

Function	**Description**	**Developer Notes**
Patient(int pid, string n, int a, string c)	Constructor initializes patient details.	Ensure all fields are set (id, name, age, contact).
void admitPatient(RoomType type)	Admits patient to a specified room type.	Update isAdmitted and roomType. Log in medicalHistory.
void dischargePatient()	Discharges patient.	Set isAdmitted = false. Log in medicalHistory.
void addMedicalRecord(string record)	Adds a medical note.	Push record into medicalHistory stack.

Function	Description	Developer Notes
void requestTest(string testName)	Requests a medical test.	Enqueue test in testQueue. Log in medicalHistory.
string performTest()	Performs the next test in queue.	Dequeue test and log it. Return test name or "No tests pending".
void displayHistory()	Prints medical history.	Use a temporary stack to reverse order (LIFO).
int getId()	Returns patient ID.	Simple getter.
string getName()	Returns patient name.	Simple getter.
bool getAdmissionStatus()	Returns admission status.	Simple getter.

3.1.2 Doctor Class

Function	Description	Developer Notes
Doctor(int did, string n, Department d)	Constructor initializes doctor details.	Set id, name, and department.
void addAppointment(int patientId)	Adds a patient to the appointment queue.	Enqueue patientId.
int seePatient()	Removes next patient from queue.	Return patientId or -1 if empty.

Function	Description	Developer Notes
int getId()	Returns doctor ID.	Simple getter.
string getName()	Returns doctor name.	Simple getter.
string getDepartment()	Returns department name.	Convert enum to string (e.g., CARDIOLOGY → "Cardiology").

3.1.3 Hospital Class

Function	Description	Developer Notes
Hospital()	Initializes counters.	Set patientCounter = 1, doctorCounter = 1.
int registerPatient(string name, int age, string contact)	Registers a new patient.	Create Patient object, add to patients vector, return ID.
int addDoctor(string name, Department dept)	Adds a new doctor.	Create Doctor object, add to doctors vector, return ID.
void admitPatient(int patientId, RoomType type)	Admits a patient.	Find patient by ID, call admitPatient().
void addEmergency(int patientId)	Adds patient to emergency queue.	Enqueue patientId.
int handleEmergency()	Processes next emergency case.	Dequeue and return patientId or -1 if empty.

Function	Description	Developer Notes
void bookAppointment(int doctorId, int patientId)	Books an appointment.	Find doctor, call addAppointment(). Log in patient’s history.
void displayPatientInfo(int patientId)	Shows patient details.	Find patient, display ID, name, admission status, and history.
void displayDoctorInfo(int doctorId)	Shows doctor details.	Find doctor, display ID, name, and department.

3.2 Non-Functional Requirements

Requirement	Description
Performance	Should handle at least 1000 patients/doctors efficiently.
Security	No sensitive data (simplified for this project).
Usability	Console-based, clear menu navigation.
Maintainability	Well-structured code with comments.

4. System Features

4.1 Patient Management Workflow

1. **Registration** → registerPatient()
2. **Admission** → admitPatient()
3. **Test Requests** → requestTest() → performTest()
4. **Discharge** → dischargePatient()

4.2 Doctor Management Workflow

1. **Add Doctor** → addDoctor()
2. **Book Appointment** → bookAppointment()
3. **See Next Patient** → seePatient()

4.3 Emergency Handling Workflow

1. **Add Emergency Case** → addEmergency()
2. **Process Emergency** → handleEmergency()