Software Requirements Specification

Hospital Management System

Prepared by: Md Mahsin Mia & Jaid Hassan

Date: May 18, 2025

Table of Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document provides a detailed description of the requirements for The Hospital Management System is a comprehensive web-based application designed to streamline and automate hospital operations. It facilitates efficient management of patients, doctors, appointments, and laboratory tests, enabling healthcare providers to deliver better patient care.

1.2 Project Scope

The Hospital Management System is a comprehensive web-based application designed to manage the day-to-day operations of a hospital. The system includes the following core components:

- User authentication and role-based access control
- Patient registration and management
- Doctor profile management
- Appointment scheduling
- Laboratory test and result management
- Reporting and dashboard statistics

2. Overall Description

2.1 Product Perspective

The Hospital Management System is a self-contained system designed to operate within a hospital's IT infrastructure. It replaces manual record-keeping processes and provides a centralized database for all hospital operations.

2.2 Product Functions

The Hospital Management System shall provide the following major functions:

1. **User Management**
    - Registration, authentication, and authorization of users
    - Role-based access control for different user types
    - Profile management and password recovery
2. **Patient Management**
    - Registration of new patients
    - Management of patient profiles and contact information
    - Viewing of appointments and laboratory tests
3. **Doctor Management**
    - Registration and profile management for doctors
    - Patient appointment viewing and management
4. **Appointment System**
    - Scheduling new appointments between patients and doctors
5. **Laboratory Management**
    - Ordering laboratory tests for patients
    - Tracking test status and results

    - Managing test reports and file attachments

3. External Interface Requirements

3.1 User Interface

- Web-based responsive UI for desktop and mobile (min. 1280x800 resolution).
- Consistent design across all modules.

3.2 Authentication Interface

- Secure login with email, password, and "Forgot Password" option.
- Displays messages for invalid login or maintenance.

3.3 Dashboard Interface

- Role-based dashboards:
    - **Admin**: Stats, user/config management
    - **Doctor**: Appointments, patients, lab results

- Includes navigation and quick actions.

4. Non-Functional Requirements

4.1 Performance

- Responses within 2 seconds under normal load.

4.3 Documentation

- **System Docs**: Architecture, DB schema, versioning
- **Dev Docs**: Code comments, build/deploy steps, test plans

5. Data Requirements

1. Data Entities

Based on the SRS document, the following core data entities are required for the Hospital Management System:

Primary Entities

1. **Patients**
   - Attributes: PatientID, name, age, phone number,password, registration date.

Patients	
PatientID	int
Name	varchar
Age	int
PhoneNumber	varchar
Password	varchar
RegistrationD...	timestamp

2. **Doctors**
   - Attributes: DoctorID, name, specialization, password, phoneNumber, registration date

Doctors	
DoctorID	int
Name	varchar
PhoneNumber	varchar
Specialization	varchar
Password	varchar
RegistrationD...	timestamp

3. **Appointments**
   - Attributes: AppointmentID, DoctorID, patientID, appointmentDate, appointmentTime, status.
   - Statuses: In-progress, Completed, Cancelled

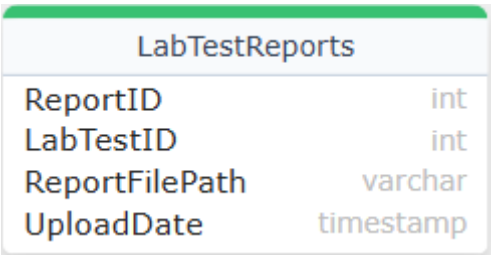
Appointments	
AppointmentID	int
DoctorID	int
PatientID	int
AppointmentDate	date
AppointmentTime	time
Status	enum

4. **Laboratory Test**
   - Attributes: LabTestID, patientID, test name, test date, results, status

LabTests	
LabTestID	int
PatientID	int
TestName	varchar
TestDate	date
Results	text
Status	enum

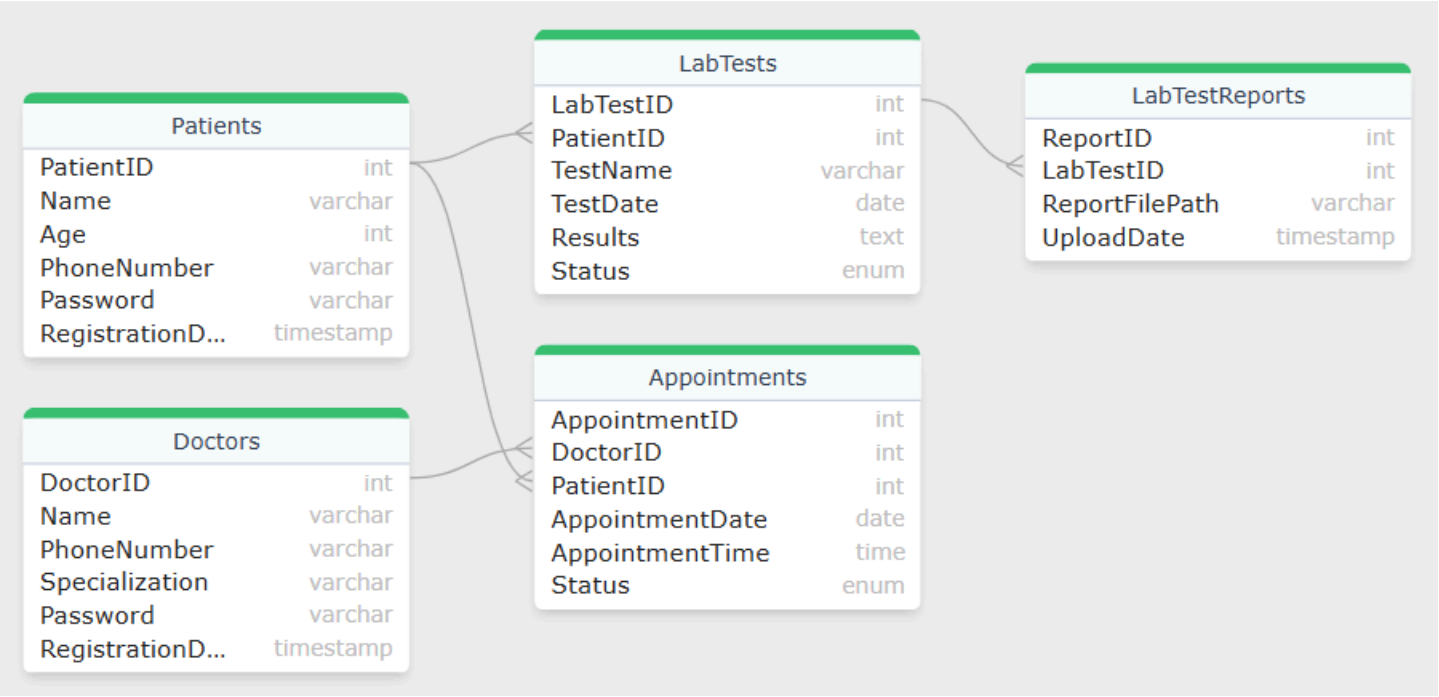
5. **Laboratory Report**

- Attributes: LabTestID, ReportID, reportFilePath, upload date
- Related data: Test reports, result files



2. Data Relationships

The following relationships exist between the core entities:



One-to-Many Relationships

1. **Patient to Other Entities**
   - One patient can have many appointments.
   - One patient can have many lab tests.
2. **Doctor to Other Entities**
   - One doctor can have many appointments.
   - One doctor can be assigned to many patients.
   - One doctor can request many lab tests.

Many-to-One Relationships

Appointments → Doctor

- Many appointments belong to one doctor.

Appointments → Patient

- Many appointments belong to one patient.

LabTests → Patient

- Many lab tests belong to one patient.

LabTests → Doctor

- Many lab tests are requested by one doctor.

One-to-One Relationships

User → Doctor (Professional Profile)

- One user account corresponds to one doctor.

3. Database Queries

Based on the functional requirements in the SRS, the system will need to support the following types of database queries:

✅ Authentication and User Management

```
-- ✅ Validate login

SELECT id, email FROM users WHERE email = ? AND password = ?;



-- ✅ Update last login (requires a `last_login` column in `users`)

UPDATE users SET last_login = CURRENT_TIMESTAMP WHERE id = ?;



-- ✅ Lock account after failed attempts (requires a `status` column in `users`)

UPDATE users SET status = 'Locked' WHERE email = ? AND status = 'Active';
```

👨‍⚕️ Patient Management

```
-- ⚠️ Register new patient (Fixed: `Email` and `EmergencyContactName` do not
exist in schema)

-- You need to update your Patients table or remove these fields.
```

```sql
-- Here's the version according to your schema:
INSERT INTO Patients (Name, Age, PhoneNumber, Password)
VALUES (?, ?, ?, ?);

-- ⚠️ Search patients (Fixed: removed `Email`, which isn't in the schema)
SELECT * FROM Patients
WHERE PatientID = ? OR Name LIKE ? OR PhoneNumber = ?;

-- ✅ Get patient profile
SELECT * FROM Patients WHERE PatientID = ?;

-- ✅ Get appointment history
SELECT * FROM Appointments WHERE PatientID = ? ORDER BY AppointmentDate DESC;

-- ✅ Get lab test history
SELECT * FROM LabTests WHERE PatientID = ? ORDER BY TestDate DESC;
```

👩‍⚕️ Doctor Management

```sql
-- ⚠️ Add new doctor (Fixed: `Email` not present in your Doctors schema)
INSERT INTO Doctors (Name, PhoneNumber, Specialization, Password)
VALUES (?, ?, ?, ?);

-- ⚠️ Set doctor schedule (Missing table `DoctorSchedule` — add it if needed)
-- Suggested table creation:
CREATE TABLE DoctorSchedule (
    ScheduleID INT AUTO_INCREMENT PRIMARY KEY,
```

```
    DoctorID INT NOT NULL,
    ScheduleDate DATE NOT NULL,
    StartTime TIME NOT NULL,
    EndTime TIME NOT NULL,
    Status ENUM('Available', 'Booked', 'Unavailable') DEFAULT 'Available',
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

-- Add schedule
INSERT INTO DoctorSchedule (DoctorID, ScheduleDate, StartTime, EndTime, Status)
VALUES (?, ?, ?, ?, 'Available');
```

📅 Appointment Management

```
-- ✅ Schedule new appointment (⚠️ Removed `CreatedBy`, not in schema)
INSERT INTO Appointments (DoctorID, PatientID, AppointmentDate, AppointmentTime)
VALUES (?, ?, ?, ?);

-- ✅ Check doctor availability
SELECT COUNT(*) AS booked FROM Appointments
WHERE DoctorID = ? AND AppointmentDate = ?
AND ? BETWEEN AppointmentTime AND ADDTIME(AppointmentTime, '00:30:00')
AND Status NOT IN ('Cancelled', 'Completed');

-- ✅ Update appointment status
UPDATE Appointments
SET Status = ?
```

```
WHERE AppointmentID = ?;


-- ✅ Search appointments within date range
SELECT a.*, p.Name AS PatientName, d.Name AS DoctorName
FROM Appointments a
JOIN Patients p ON a.PatientID = p.PatientID
JOIN Doctors d ON a.DoctorID = d.DoctorID
WHERE a.AppointmentDate BETWEEN ? AND ?;
```

🧪 Lab Test Management

```
-- ✅ Add lab test
INSERT INTO LabTests (PatientID, TestName, TestDate)
VALUES (?, ?, ?);


-- ✅ Update lab test result
UPDATE LabTests
SET Results = ?, Status = 'Completed'
WHERE LabTestID = ?;


-- ✅ Upload lab test report
INSERT INTO LabTestReports (LabTestID, ReportFilePath)
VALUES (?, ?);


-- ✅ Fetch patient reports
SELECT r.*, t.TestName, t.TestDate
FROM LabTestReports r
```

```
JOIN LabTests t ON r.LabTestID = t.LabTestID

WHERE t.PatientID = ?;
```

6. Appendix

6.1 Glossary

Term	Definition
Admin	System administrator with full access privileges to manage the HMS.
CRUD	Create, Read, Update, Delete; the four basic operations of persistent storage.
SRS	Software Requirements Specification; this document.
UI	User Interface; the space where interactions between humans and machines occur.
UX	User Experience; a person's emotions and attitudes about using a particular product, system, or service.

6.2 Analysis Models

6.2.1 System Context

The **System Context Diagram** shows the interactions between the system and external actors.

Actors:

- Admin
- Doctor
- Patient
- Lab Technician

6.2.2 Use Case

Actors: Admin, Patient, Doctor, Lab Technician

Use Cases:

- **Patient**: Register, Login, Book Appointment, View Test Result

- **Doctor**: Login, View Appointments, Update Appointment Status
- **Lab Technician**: Upload Test Results, Update Test Status
- **Admin**: Manage Users, View Reports

6.3 Processes

6.3.1 Issue Resolution Process

Steps:

1. **Issue Reporting**: User (Doctor/Patient/Lab) reports an issue via a support interface.
2. **Ticket Generation**: System generates a ticket ID and logs the issue.
3. **Admin Assignment**: Admin reviews and assigns the issue to a relevant developer or technician.
4. **Resolution**: Developer resolves the issue and marks the ticket as "Resolved".
5. **Feedback**: User is notified and can give feedback on resolution.
6. **Closure**: Admin closes the ticket after confirmation.

6.3.2 Change Request Process

Steps:

1. **Request Initiation**: Stakeholder submits a change request (feature update, schema change, etc.).
2. **Impact Analysis**: Admin/development team evaluates the scope and potential impact.
3. **Approval**: Admin or system owner approves/rejects the request.
4. **Implementation**: Approved changes are implemented in the development environment.
5. **Testing & Validation**: QA tests the new changes.
6. **Deployment**: Changes are deployed to production with version control.
7. **Documentation Update**: All affected documents and schema diagrams are updated.