

Project Proposal

On

Project Title

Guided By:-

Trainer Name: Anuj Kumar

Created By:-

**Student Name: Anuj Kumar
AFid : AF04991734**

**Student Name: Himanshu
AFid : AF049717773**

Student Name: Rahul Prajapati

AFid :AF04991823

Batch Code: ANP-D2405

Course Code: ITPR

Table Of Contents

Index

- 1.** Title of the Project.....
- 2.** Introduction.....
- 3.** Objective
- 4.** Project Category
- 5.** Analysis
 - Modules and Description
 - Database Design
 - ER Diagram
 - Data Flow Diagram
- 6.** Complete Structure
 - Process Logical Diagram
- 7.** Platform Used
 - Hardware Requirement
 - Software Requirement
- 8.** Future Scope
- 9.** Bibliography

Title of the Project

Hospital Management System

INTRODUCTION

The Hospital Management System (HMS) is a software solution designed to digitalize and streamline the daily operations of a hospital. Instead of maintaining separate registers for patients, doctors, appointments, lab tests, and billing, the system stores all data in a centralized database.

The system mainly focuses on:

- Managing patient information and visit details.
- Managing doctors and departments with their specialization.
- Scheduling and tracking appointments between patients and doctors.
- Recording laboratory tests and results.
- Generating billing information for services availed by patients.

OBJECTIVE

- To automate basic operations of a hospital using a computer-based system.
- To maintain accurate and up-to-date records of patients, doctors, and departments.
- To provide a proper system for booking and managing appointments.
- To store and retrieve lab test details and results efficiently.
- To generate bills for patients based on appointments and lab tests.
- To reduce manual paperwork and chances of errors.
- To allow quick searching and reporting from stored data.

PROJECT CATEGORY

- Database Management/Application Development
- Frontend: Core Java (Console/Command Line)
- Backend: MySQL Database
- Connectivity: JDBC (Java Database Connectivity).

ANALYSIS

a) Modules and Description

1. Patient Management Module

- Add new patient records (name, age, gender, phone, address, etc.).
- Update patient details if any information changes.
- Search and view patient details by Patient ID, name, or phone number.
- Maintain history of visits and tests linked to the patient.

2. Department Management Module

- Maintain a master list of hospital departments (e.g., Cardiology, Orthopedic, Neurology).
- Add Department details such as name and Id
- View Departments
- Update Departments
- Delete Department

3. Doctor Management Module

- Add doctor details such as name, specialization, phone, and department.
- Link each doctor to a specific department.
- Update doctors.
- View the list of doctors under each department.

4. Appointment Management Module

- Book appointments between patient and doctor.
- Record appointment date, time, and purpose.

- Check doctor availability before booking.
- Update or cancel appointments when needed.
- View all appointments for a doctor or patient.

5. Test Management Module

- Register lab tests for patients (e.g., Blood test, X-ray, MRI).
- Store test name, type, date, and results.
- Link each lab test to the corresponding patient (and optionally to an appointment/doctor).
- Provide quick access to the patient's test history during future visits.

6. Billing Module

- Generate bills for patients based on consultation fees, lab test charges, and services.
- Store bill amount, date, and payment status (Paid/Unpaid).
- View bills patient-wise or date-wise.
- Help hospital administration maintain financial records.

7. Pharmacy Management Module

- Storing **medicine details** (name, type, price, expiry, stock).
- Tracking **stock in/out** of medicines.
- Issuing medicines to **patients** (IPD / OPD).
- Keeping record of **which patient bought which medicines** and on which date.

b) Database Design

1. DEPARTMENT Table Design

| Field Name | Datatype | Properties |
|-----------------|--------------|-----------------------------|
| department_id | int | primary key, auto increment |
| department_name | varchar(100) | not null |

2. PATIENT Table Design

| Field Name | Datatype | Properties |
|------------|--------------|-----------------------------|
| patient_id | int | primary key, auto increment |
| name | varchar(100) | not null |
| gender | varchar(10) | not null |
| age | int | not null |
| phone | varchar(15) | not null |
| address | varchar(200) | not null |

3. DOCTOR Table Design

| Field Name | Datatype | Properties |
|----------------|--------------|---|
| doctor_id | int | primary key, auto increment |
| name | varchar(100) | not null |
| specialization | varchar(50) | not null |
| phone | varchar(15) | not null |
| department_id | int | foreign key → department(department_id) |

Relation: One patient can have multiple appointment with multiple Doctors.

4. APPOINTMENT Table Design

| Field Name | Datatype | Properties |
|------------------|--------------|-----------------------------------|
| appointment_id | int | primary key, auto increment |
| patient_id | int | foreign key → patient(patient_id) |
| doctor_id | int | foreign key → doctor(doctor_id) |
| appointment_date | date | not null |
| appointment_time | time | not null |
| remarks | varchar(200) | nullable |

5. LAB_TEST Table Design

| Field Name | Datatype | Properties |
|------------|--------------|-----------------------------------|
| test_id | int | primary key, auto increment |
| patient_id | int | foreign key → patient(patient_id) |
| test_name | varchar(100) | not null |
| test_date | date | not null |
| result | varchar(200) | nullable |

6. BILLING Table Design

| Field Name | Datatype | Properties |
|--------------|---------------|-----------------------------------|
| bill_id | int | primary key, auto increment |
| patient_id | int | foreign key → patient(patient_id) |
| bill_date | date | not null |
| total_amount | decimal(10,2) | not null |
| status | varchar(20) | default 'unpaid' |

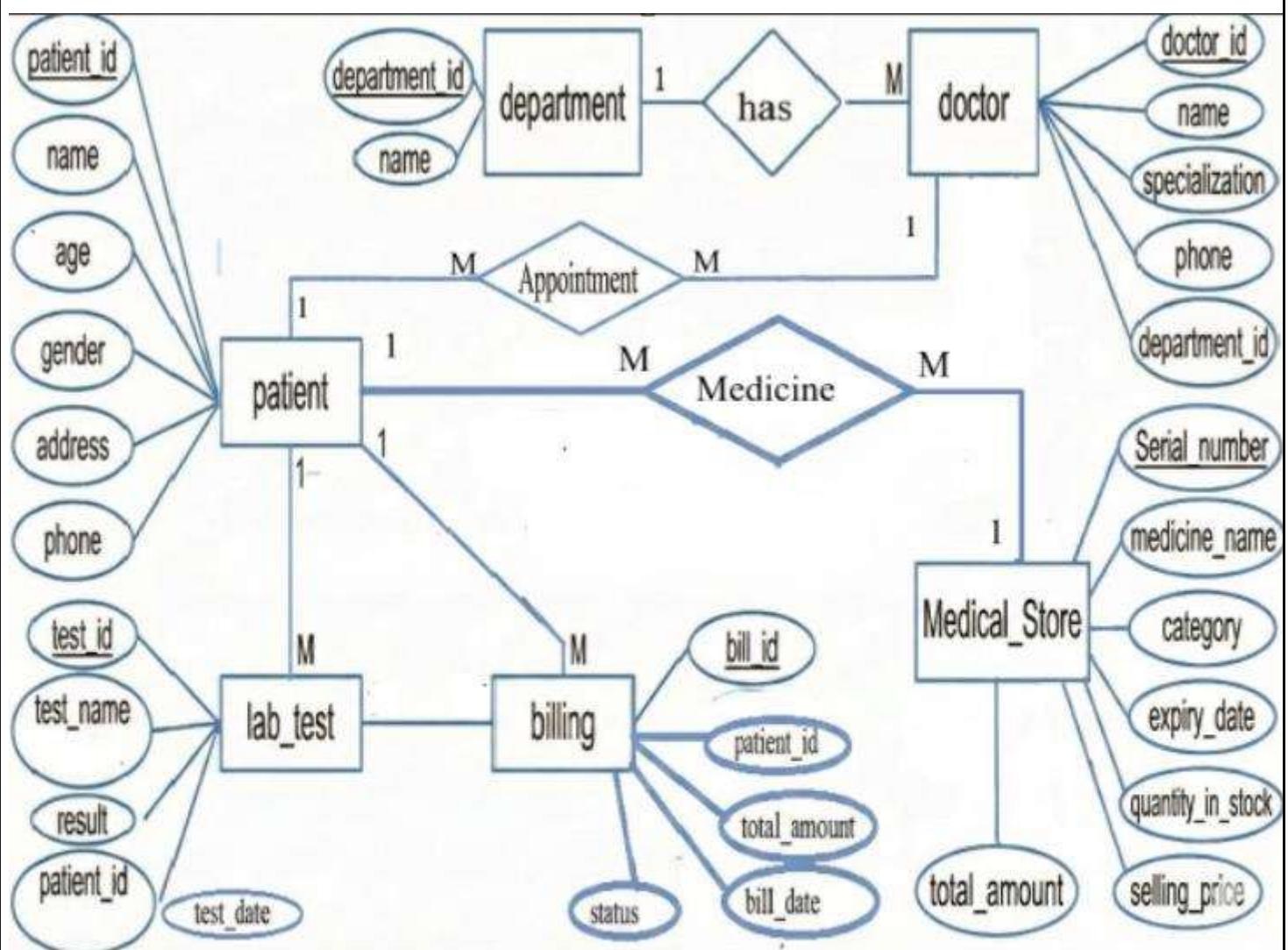
7. MEDICAL_STORE Table Design

| Field Name | Datatype | Properties |
|-------------------|---------------|--|
| Serial_number | int | Primary key |
| medicine_name | varchar(100) | not null |
| category | varchar(50) | not null (Tablet/Syrup/Injection etc.) |
| expiry_date | date | not null |
| quantity_in_stock | int | not null |
| selling_price | decimal(10,2) | not null |

8. MEDICINE Table Design

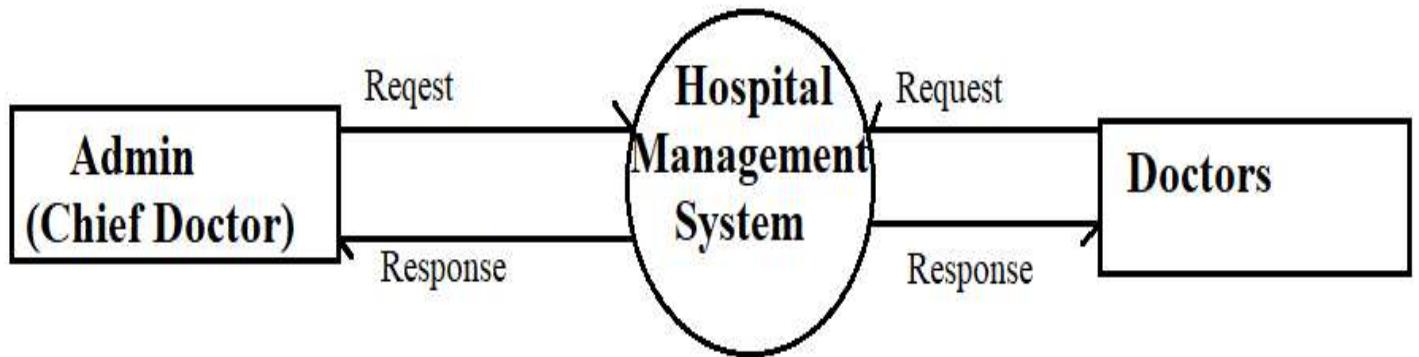
| Field Name | Datatype | Properties |
|---------------|---------------|---|
| patient_id | int | foreign key → patient(patient_id) |
| medicine_name | int | foreign key → medical_store(medicine_name) |
| doctor_id | int | foreign key → doctor(doctor_id), null allowed |
| issue_date | datetime | not null |
| quantity | int | not null |
| total_amount | decimal(10,2) | not null |

Entity Relationship Diagram

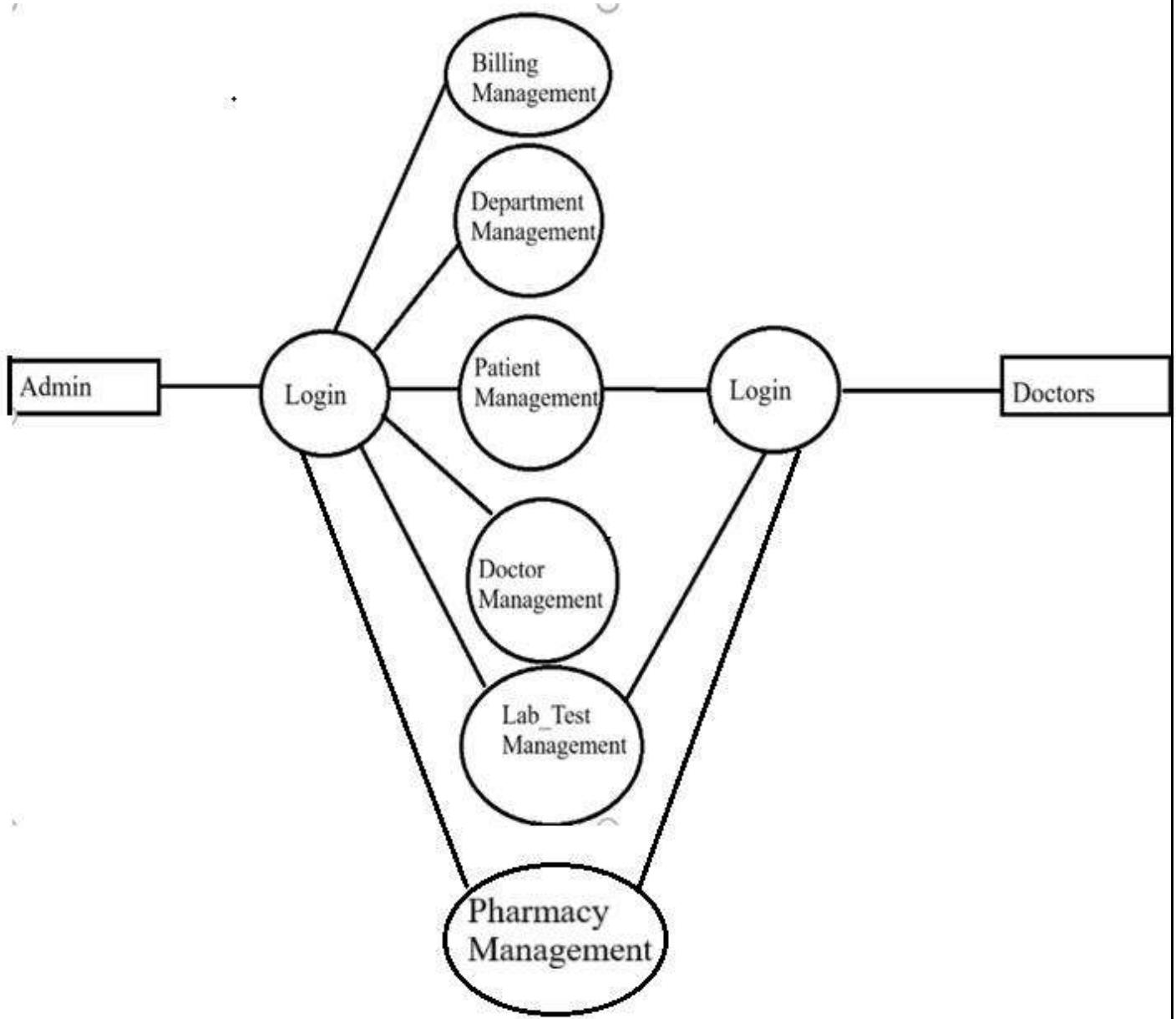


Data Flow Diagram

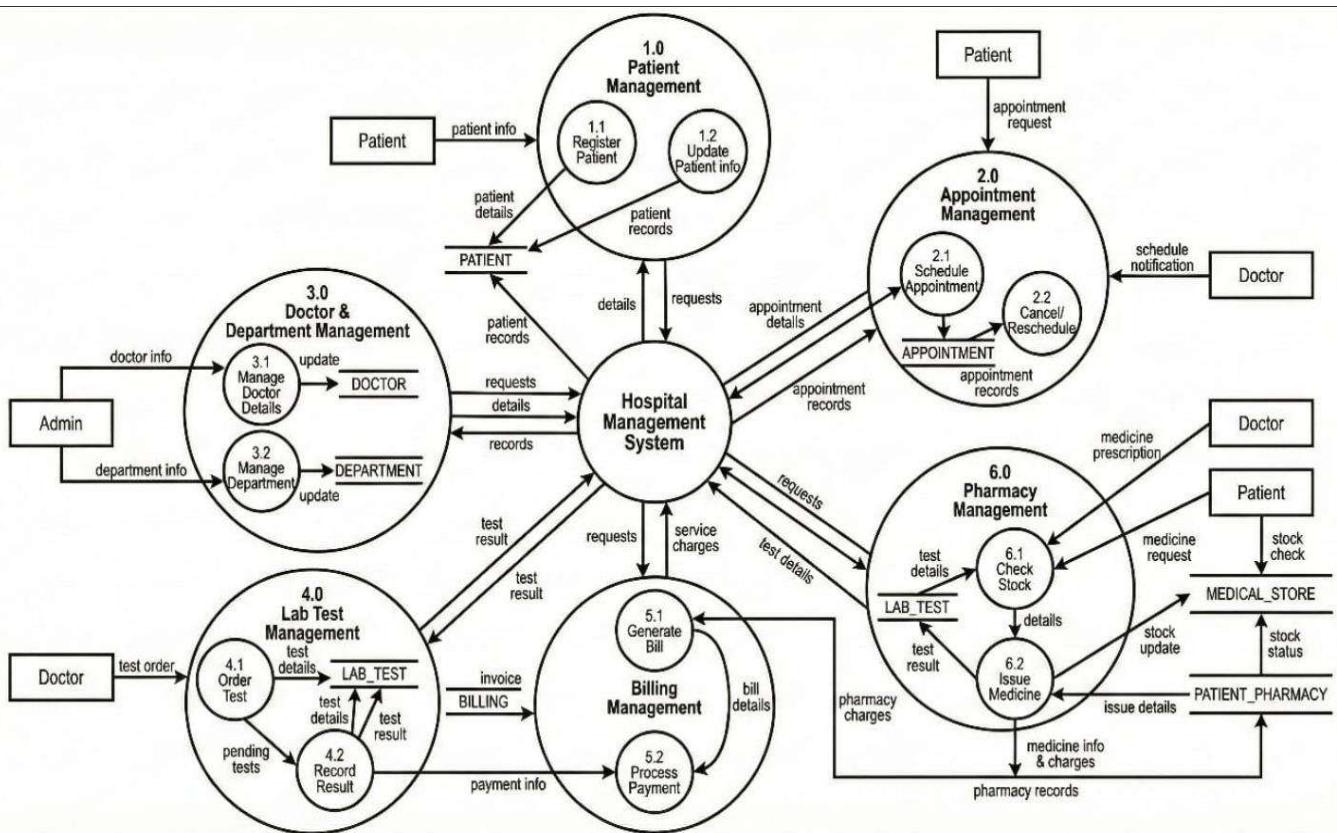
a) 0 Level DFD



b) 1 level DFD

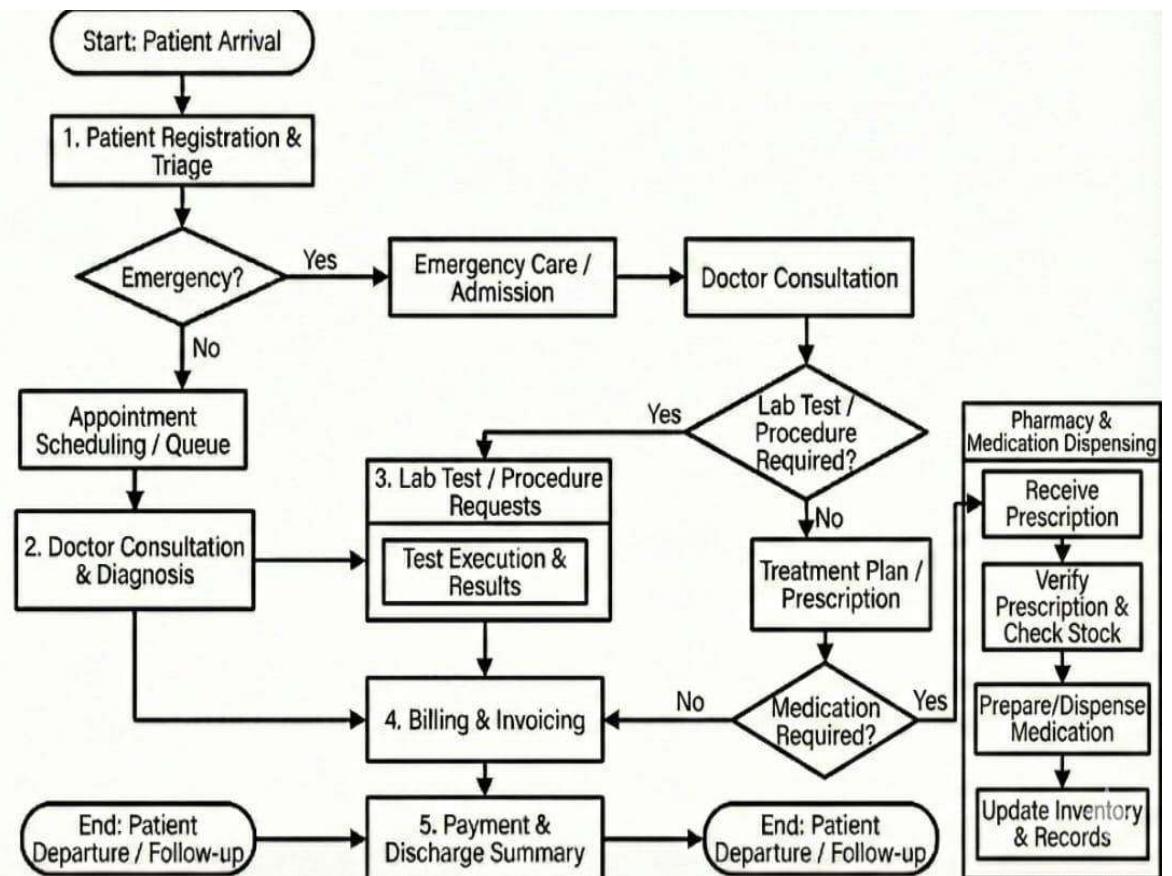


c) 2 level DFD



Complete Structure

Process Logic Diagram



Platform Used

- **Hardware Requirements:** Any standard PC or laptop is sufficient. Since this is a simple command-line program, minimal specs (e.g. 1–2 GHz CPU, ~1–2 GB RAM) are adequate. It does not require specialized hardware; even a low-end machine can run the Java CLI app and MySQL server for demo purposes.
- **Software Requirements:**
 - **Operating System:** Windows, Linux, or macOS (any OS that can run Java and MySQL).
 - **Java Development Kit (JDK):** Java SE (version 8 or later recommended). The application is written in Java, so a JDK must be installed and configured in the system PATHgithub.com.
 - **MySQL Server:** MySQL Community Server (or any MySQL-compatible database) to host the hospital database. Ensure MySQL is running and accessiblegithub.com.
 - **JDBC Connector:** MySQL Connector/J library (JAR) added to the project so Java can connect to MySQLgithub.com.
 - **IDE/Text Editor (optional):** Any Java editor (e.g. Eclipse, IntelliJ, VS Code) can be used for coding; but since the app is CLI, an IDE is not strictly necessary. The GitHub example used VS Code

Future Scope

1. Addition of Advanced Modules:

The system can be expanded to include Billing, Pharmacy Management, Laboratory Reports, Ward/Bed Allocation, and Staff Payroll to transform it into a complete hospital information management solution.

2. Upgrade to GUI or Web Application:

The current command-line system can be redesigned with JavaFX/Swing for a graphical interface or converted into a web-based solution using Spring Boot, improving accessibility and usability.

3. Integration of Online Portals:

Future enhancements may include patient and doctor portals for booking appointments, checking reports, updating medical records, and managing schedules remotely.

4. Automated Notifications:

The system can be integrated with SMS/email gateways to send reminders for appointments, medicine refills, and follow-up visits, reducing patient no-shows and improving hospital workflow.

5. Cloud-Based Deployment:

Deploying the system on cloud platforms can enable real-time access, automatic backups, multi-branch connectivity, and high scalability for larger healthcare networks.

6. Enhanced Security:

Implementation of encryption, role-based access control, secure logins, and audit trails can strengthen data confidentiality and meet healthcare compliance standards.

7. Mobile App Development:

A mobile application can be developed for doctors, staff, and patients to access the system from anywhere, improving responsiveness and convenience.

Bibliography

- Java Documentation, Oracle: Official reference for Java programming concepts, object-oriented structure, and JDBC connectivity.
- MySQL Documentation: Comprehensive guide for SQL queries, relational database design, and MySQL server

configuration.

- GeeksforGeeks: Conceptual references used for understanding Hospital Management System modules, database design, ER diagrams, and Java examples.
- Tutorials Point: Learning resource for Java fundamentals, JDBC operations, and structured programming techniques.
- GitHub Community Projects. Open-source Java–MySQL command-line project examples used for understanding modular structure.