DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi 110042

DEPARTMENT OF SOFTWARE ENGINEERING

SE209: Software Engineering Methodologies
Lab file

Submitted To:
Mr. Ankur
Department of Software Engineering

Submitted By:
Yug 23/SE/181
Aditya 23/SE/182
Aditya 23/SE/183
Astha 23/SE/184

TABLE OF CONTENTS

EXPERIMENT 1

AIM: To write problem statement for Hospital Management System

THEORY:
In today's fast-paced world, waiting in long hospital queues can be incredibly frustrating and time-consuming. Typically, hospital queues are managed manually by administrative staff, requiring patients to make an appointment, wait for their turn, and inquire about doctor availability. What makes this even worse is travelling a long distance, only to find out that the doctor is unavailable or on leave, making the trip unnecessary.

Developing a Hospital Management System enables the administration to efficiently manage patient data, doctor appointments, and hospital services. The system provides functionalities for both hospital staff (administrators, doctors) and patients.A Hospital Management System (HMS) can address multiple issues by allowing patients to book appointments from home and check the availability of their preferred doctor in advance. Doctors, in turn, can confirm or decline appointments, helping both parties. All the information is stored centrally by the system in files for easy access and updation. Bills can also be generated by the system to ensure ease of payment.

The goal is to create a menu driven console application with appropriate classes and methods, implementing object oriented principles like inheritance, polymorphism, and file handling.

Objective
The primary objective of the Hospital Management System (HMS) is to streamline and automate various administrative and clinical operations within a healthcare facility. By digitising patient data, appointment scheduling, and billing processes, HMS aims to:

1. **Improve Patient Experience:** Allow patients to book appointments, check doctor availability, and make payments online, reducing the need for long waits and unnecessary hospital visits.
2. **Enhance Doctor and Staff Efficiency:** Enable doctors to manage their appointments, confirm or decline bookings, and access patient records digitally, improving workflow and reducing administrative burden.
3. **Centralize Data Management:** Ensure all patient records, doctor information, and hospital services are stored in a centralised file, allowing quick retrieval and updating of information.
4. **Increase Operational Efficiency**: Automate tasks such as patient registration, appointment scheduling, and billing to reduce manual errors and administrative delays, ensuring smoother hospital operations.
5. **Enhance Communication:** Foster better communication between patients and doctors by providing timely updates on appointment status and medical records.

By achieving these objectives, the HMS contributes to improved healthcare services, better patient outcomes, and efficient hospital management.

Functionality

For a Hospital Management System (HMS), the following basic functionalities are essential to achieve the outlined objectives:

1. Patient Registration and Management

- Register new patients and create unique patient IDs.
- Maintain patient profiles including personal details, medical history, and contact information.
- Allow patients to update their details securely.

2. Appointment Scheduling

- Patients can book, reschedule, or cancel appointments online.
- Provide real-time availability of doctors and appointment slots.

3. Doctor and Staff Management

- Maintain a database of doctors, including specialisations, work schedules, and availability.
- Track doctor and staff performance, attendance, and workload.
- Enable staff to assign duties, manage shifts, and track leave.

4. Security and Access Control

- Implement strong access control to ensure only authorized personnel can access sensitive information.
- Use encryption for data storage and transfer to protect patient privacy.

Technical Requirements

1. Development Environment

- Compiler: A C++ compiler like GCC (GNU Compiler Collection), MSVC (Microsoft Visual C++), or Clang.
- IDE/Text Editor: An Integrated Development Environment (IDE) like Visual Studio, Code::Blocks, CLion, or a text editor like VSCode or Sublime Text.

2. Basic Knowledge Requirements

- C++ Fundamentals: Understanding of C++ syntax, data types, operators, control structures, and functions.
- Object-Oriented Programming (OOP): Knowledge of classes, objects, inheritance, polymorphism, encapsulation, and abstraction.
- File Handling: Understanding of file input/output operations for data persistence.
- Standard Template Library (STL): Familiarity with containers (like vectors, lists, and maps) and algorithms.
- Pointers and Dynamic Memory Management: Knowledge of pointers, memory allocation, and deallocation.

3. System Design

- User Management: Implementing different user roles such as admin, doctors, nurses, and patients with appropriate access controls.
- Patient Management: Recording and retrieving patient information, medical history, treatment details, and billing.
- Appointment Scheduling: Managing appointments for patients with doctors.

4. Modules and Features

- User Authentication and Authorization:
    - Login system with different roles.
    - Password management.
- Patient Module:
    - Add, update, and delete patient records.
    - Search and display patient details.
- Doctor Module:
    - Add, update, and delete doctor records.
    - Schedule and manage appointments.
- Billing Module:
    - Generate and manage bills.
    - Record payments and print invoices.

5. Database Integration

- File-based Storage: For simple implementations, using text files or binary files to store data.

- Database Management System (DBMS): For more complex systems, using a DBMS like SQLite, MySQL, or PostgreSQL for data storage and retrieval.

6. Libraries and Frameworks

- Standard Libraries: Utilizing the C++ standard libraries for data structures, file handling, and other utilities.
- Third-Party Libraries: Optionally, using libraries like Boost for additional functionality.

7. Testing and Debugging

- Writing test cases to verify the functionality of different modules.
- Using debugging tools to identify and fix issues.

Expected Outcomes

1. Improved Patient Care
a. Accurate Records: Centralized and accurate patient records accessible to authorized personnel.
b. Timely Treatment: Reduced wait times and timely treatment through efficient appointment scheduling and management.
c. Personalized Care: Enhanced ability to provide personalized care plans based on comprehensive patient data.
2. Enhanced Operational Efficiency
a. Streamlined Processes: Automation of administrative tasks such as patient registration, billing, and inventory management.
b. Resource Optimization: Better resource allocation and utilization, reducing waste and inefficiencies.
c. Quick Access: Immediate access to patient information and medical history, facilitating quicker decision-making.
3. Data Accuracy and Consistency
a. Error Reduction: Minimized human errors in data entry and record-keeping.
b. Consistent Data: Standardized data formats and procedures ensuring consistency across departments.
4. Improved Data Security and Compliance
a. Secure Data: Enhanced data security measures protecting sensitive patient information.
b. Regulatory Compliance: Compliance with healthcare regulations (e.g., HIPAA, GDPR) ensuring legal and ethical standards are met.
5. Enhanced User Experience
a. User-Friendly Interface: Intuitive and easy-to-navigate interface improving user satisfaction and productivity.
b. Training and Support: Effective training programs and ongoing support leading to proficient use of the system.
6. Better Resource Management

a. Inventory Control: Efficient management of medical supplies and inventory, reducing stock-outs and overstock situations.
b. Financial Management: Streamlined billing and payment processes improving financial management and reducing errors.

7. Comprehensive Reporting and Analytics

a. Insightful Reports: Generation of detailed reports and analytics for informed decision-making and strategic planning.
b. Performance Monitoring: Real-time monitoring and evaluation of hospital performance and patient outcomes.

8. Scalability and Future-Proofing

a. Scalable System: Ability to scale the system to accommodate growing data and user numbers.
b. Adaptability: Flexible architecture allowing for easy updates and integration with new technologies and systems.

9. Increased Patient Satisfaction

a. Patient Engagement: Improved patient engagement through better communication and access to their own health information.
b. Positive Experiences: Enhanced overall patient experience leading to higher satisfaction and trust in the hospital services.

Challenges

1. Requirement Analysis

- Comprehensive Understanding: Accurately gathering and understanding the diverse requirements from various stakeholders (doctors, nurses, administrative staff, patients) is crucial.
- Evolving Requirements: Healthcare regulations and standards can change, necessitating updates to the system.

2. Data Management

- Data Integration: Integrating data from various departments (e.g., labs, radiology, pharmacy) and ensuring seamless data flow can be complex.
- Data Volume: Managing large volumes of data, including patient records, medical histories, and treatment plans.
- Data Accuracy: Ensuring the accuracy and consistency of data entered into the system.

3. Security and Privacy

- Data Security: Protecting sensitive patient information from unauthorised access and breaches.
- Compliance: Ensuring the system complies with healthcare regulations

4. User Interface and Experience

- User-Friendly Interface: Designing an intuitive and easy-to-use interface for a diverse user base with varying levels of technical proficiency.
- Training: Providing adequate training to users to ensure they can effectively use the system.

5. System Integration

- Interoperability: Ensuring the HMS can communicate and exchange data with other systems (e.g., electronic health records (EHR) systems, lab information systems).

6. Scalability and Performance

- Scalability: Designing the system to handle increasing numbers of users and data as the hospital grows.
- Performance: Ensuring the system performs efficiently, with minimal latency, especially during peak usage times.

7. Testing and Validation

- Extensive Testing: Conducting thorough testing to identify and fix bugs, ensuring the system is reliable and functions as expected.
- Validation: Validating that the system meets all specified requirements and performs accurately under various scenarios.

8. Maintenance and Support

- Continuous Maintenance: Regularly updating the system to fix bugs, improve performance, and comply with new regulations.
- User Support: Providing ongoing support to users to resolve issues and ensure smooth operation.

9. Customizability and Flexibility

- Customization: Allowing the system to be customized to meet the specific needs of different hospitals or departments.
- Flexibility: Designing the system to be flexible enough to adapt to changing healthcare practices and technologies.

10. Backup and Disaster Recovery

- Data Backup: Implementing robust data backup solutions to prevent data loss.
- Disaster Recovery: Ensuring there are effective disaster recovery plans in place to quickly restore operations in case of a system failure.

EXPERIMENT 2

AIM: To write the use case description for Hospital Management System

THEORY:

1. Use Case: Patient Management

- Actor: Hospital Staff
- Description: The hospital staff manages patient records by adding new patients or viewing existing patient information.
- Preconditions: The user is authenticated and is on the main menu.
- Postconditions: New patient records are added, or existing patient information is viewed.

Workflow:

1. The staff selects "Patient Management" from the main menu.
2. The system invokes patientManagement().
3. The staff can choose to add a new patient or view patient records.
4. If adding a new patient:
    - The system prompts for the patient's details (name, age, disease).
    - A Patient object is created and saved to the file.
5. If viewing records:
    - The system loads patient data from the file and displays it.
6. The staff completes the task, and the system returns to the main menu.

2. Use Case: Doctor Management

- Actor: Hospital Staff
- Description: The hospital staff manages doctor records by adding new doctors or viewing existing doctor information.
- Preconditions: The user is authenticated and has navigated to the "Doctor Management" option.
- Postconditions: New doctor records are saved, or existing doctor information is displayed.

Workflow:

1. The staff selects "Doctor Management" from the main menu.
2. The system invokes doctorManagement().
3. The staff can choose to add a new doctor or view doctor records.
4. If adding a doctor:

    - The system prompts for the doctor’s details (name, specialty, etc.).
    - A new record is created and saved.
5. If viewing records:
    - The system retrieves and displays doctor data.
6. The staff completes the task, and the system returns to the main menu.

3. Use Case: Appointment Scheduling

- Actor: Hospital Staff
- Description: The staff schedules appointments or views existing appointments.
- Preconditions: The user is authenticated and navigates to "Appointment Scheduling."
- Postconditions: New appointments are scheduled, or existing appointment records are viewed.

Workflow:

1. The staff selects "Appointment Scheduling" from the main menu.
2. The system invokes appointmentScheduling().
3. The staff can choose to schedule a new appointment or view existing appointments.
4. If scheduling:
    - The system prompts for the patient, doctor, date, and time.
    - A new appointment record is created.
5. If viewing:
    - The system retrieves and displays appointment details.
6. The system returns to the main menu.

4. Use Case: Exit System

- Actor: Hospital Staff
- Description: The staff exits the HMS interface.
- Preconditions: The user is authenticated and chooses to exit from the main menu.
- Postconditions: The system displays an exit message and terminates.

Workflow:

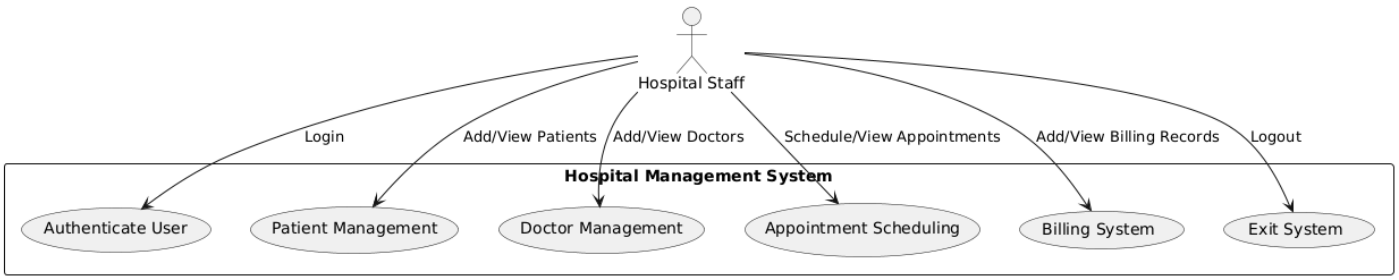
1. The staff selects "Exit" from the main menu.
2. The system displays a message confirming the exit.
3. The program terminates.

EXPERIMENT 3

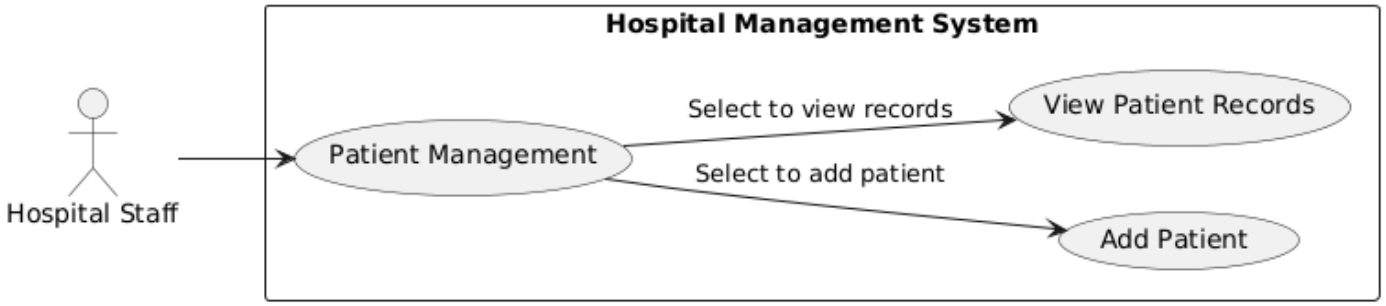
AIM: To make the use case diagrams for Hospital Management System

THEORY:

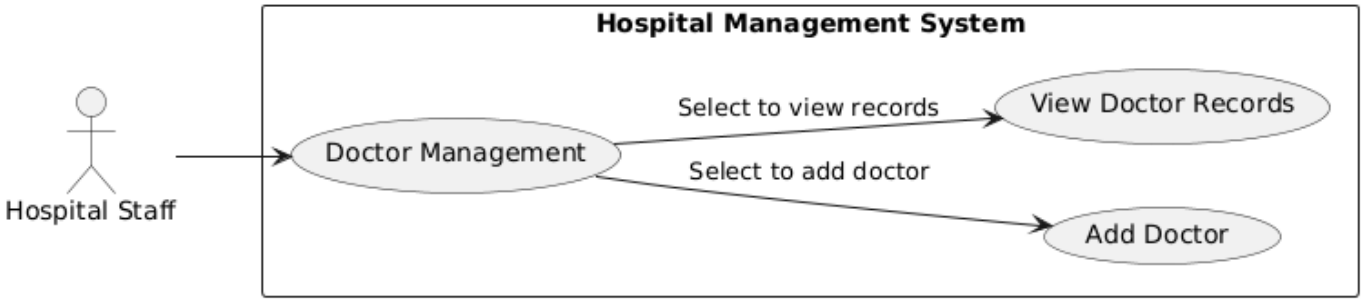
1. **System use case diagram**



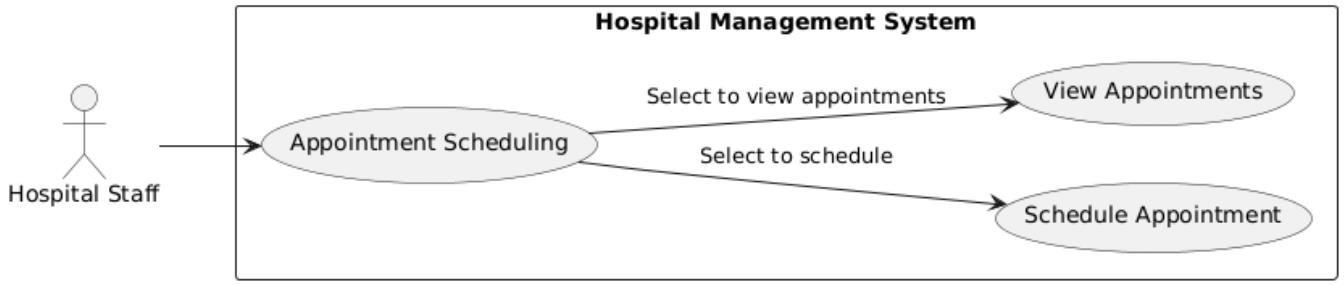
2. **Patient management use case diagram**



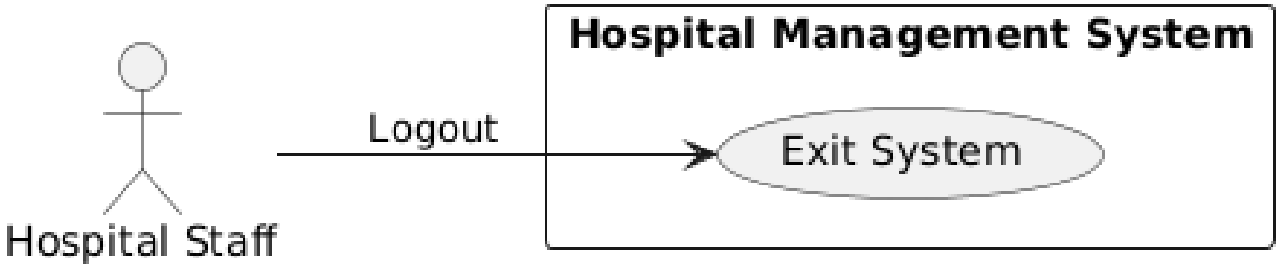
3. **Doctor management use case diagram**



4. **Appointment scheduling use case diagram**



5. **Exit use case diagram**



EXPERIMENT 4

AIM: To make the data flow diagrams for Hospital Management System

THEORY:
Data Flow Diagrams (DFDs) are graphical representations used to visualize the flow of data within a system. They illustrate how data moves from input to process to output and highlight the interactions between different components of a system. Here are some key points about DFDs:

Purpose:
- To provide a clear picture of how data flows through a system.
- To identify data sources, data destinations, data storage points, and data transformations.

Components:
- External Entities: Represented by squares or rectangles, these are outside sources or destinations of data, such as users, external systems, or organizations.
- Processes: Represented by circles or rounded rectangles, these show the transformation of data. Each process has a unique identifier and a descriptive name.
- Data Stores: Represented by open-ended rectangles or parallel lines, these indicate places where data is stored within the system, such as databases or files.
- Data Flows: Represented by arrows, these show the direction of data movement between processes, data stores, and external entities.

PROCEDURE

Levels of DFDs:
- Context Diagram (Level 0 DFD): Provides a high-level overview of the system, showing the system as a single process with its relationships to external entities.
- Level 1 DFD: Breaks down the main process of the context diagram into sub-processes, showing more detail about data flow within the system.
- Level 2 (and beyond) DFDs: Further decompose the processes from the Level 1 DFD into more detailed sub-processes.

Benefits:
- Simplifies understanding of complex systems by breaking them down into manageable components.
- Helps in identifying inefficiencies or redundancies in the data flow.
- Aids in communication among stakeholders by providing a common visual language.

- Useful for system analysis and design, helping developers and analysts ensure all data paths are accounted for.

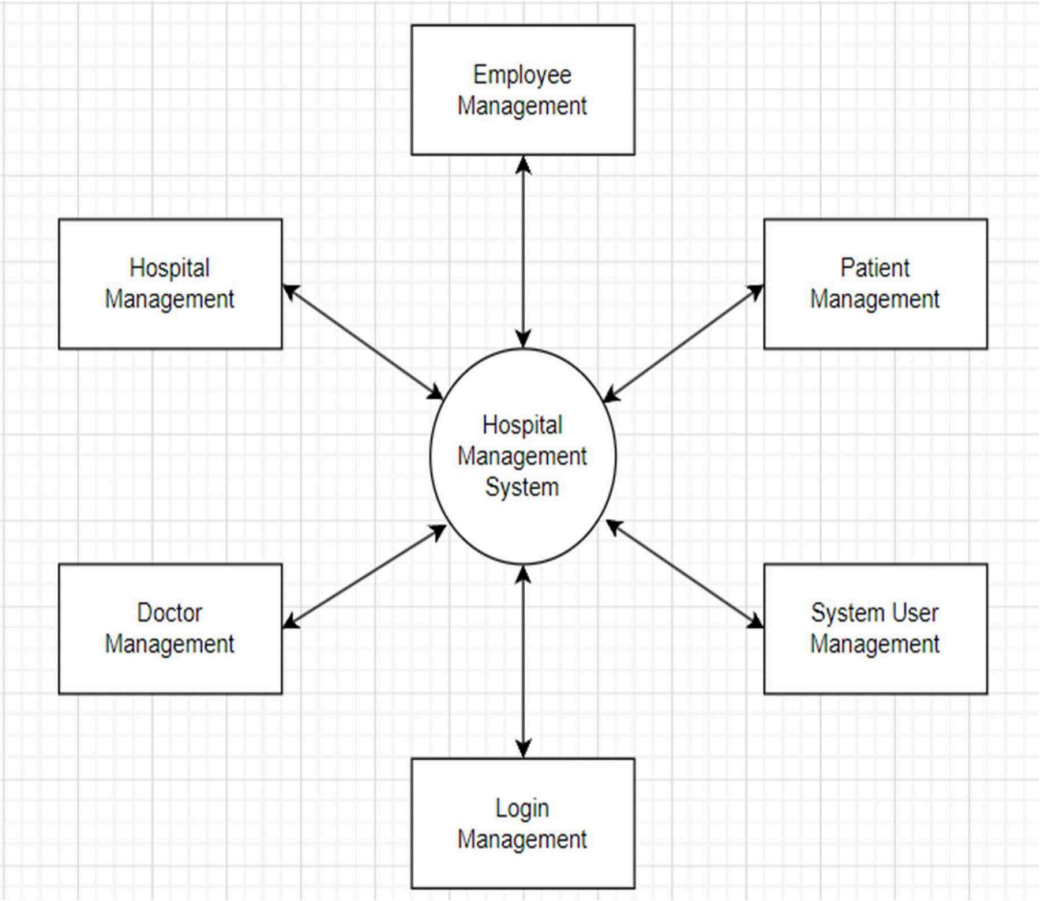
Usage:

- System development and analysis to understand existing systems or design new systems.
- Documentation for system requirements and specifications.
- Identifying and solving problems in data flow within business processes.

Design Level 0 DFD (Context Diagram):

o The Level 0 DFD, also known as the Context Diagram, represents the entire hospital management system as a single process and shows its interactions with external entities (Patient, Doctor, Admin).

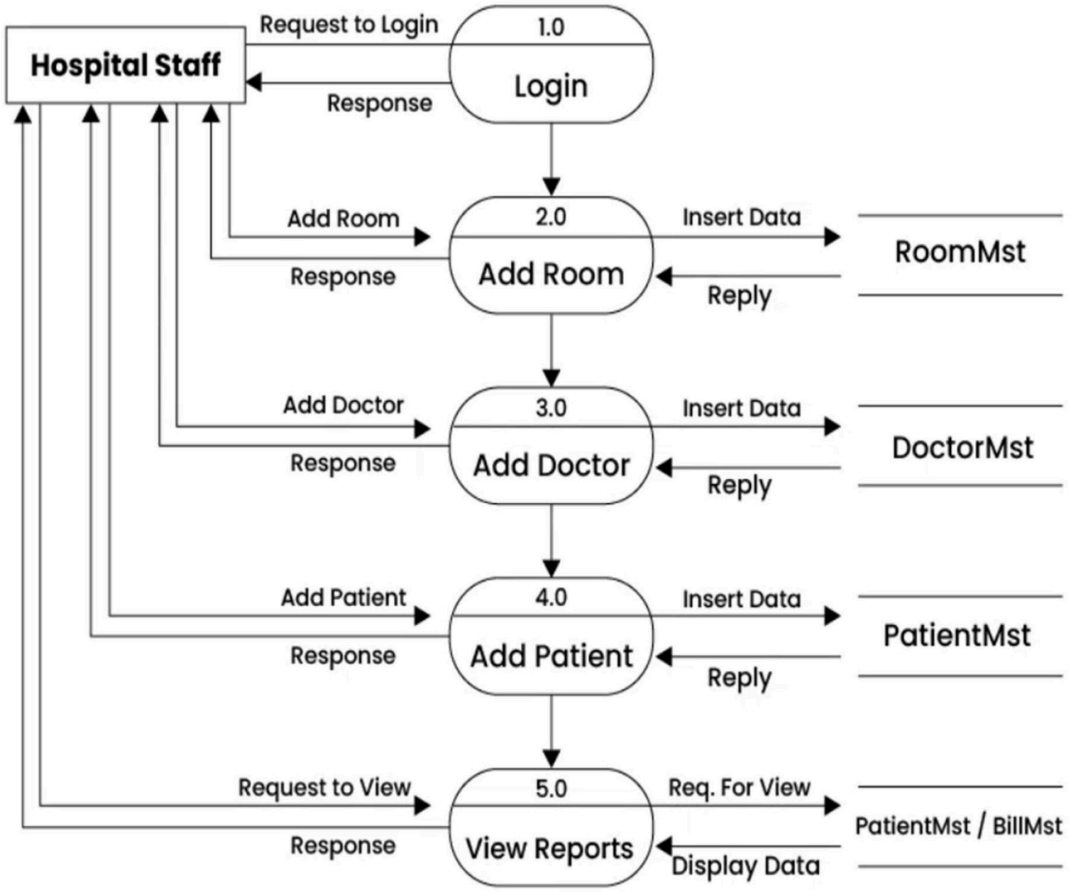
o External entities such as Patient, Doctor, and Admin interact with the system by providing and receiving data. For example, the Patient requests appointments and receives confirmation, while the Doctor accessed patient records.



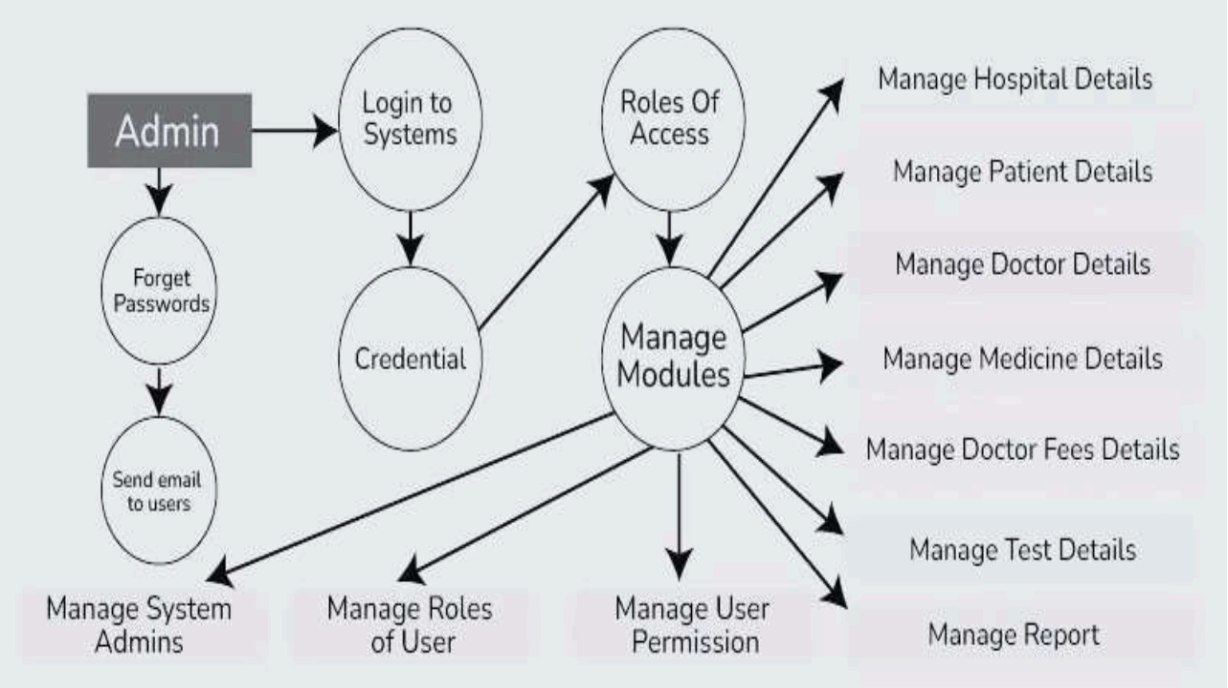
Design Level 1 DFD:

o In Level 1 DFD, the single process in the Context Diagram is decomposed into major sub- processes that correspond to core functionalities:

Appointment Management: Handles appointment scheduling, updates, and confirmation.
Billing Management: Processes bills and payments.
Record Management: Allows doctors to access and update patient medical records.
Staff Management: Administers staff data, including schedules and assignments.
o This level shows how data flows between these sub-processes and the external entities (Patient, Doctor, Admin) and data stores (Appointment Data, Billing Data, Medical Records, Staff Data).



Design Level 2 DFD:
o In Level 2 DFD, a specific process from Level 1 DFD (e.g., Appointment Management) is broken down further into more detailed sub-processes:
Schedule Appointment: Allows patients to request an appointment, checks available time slots, and schedules appointments.
Update Appointment: Doctors can update or modify existing appointments. Appointment Data Store: Stores the appointment details and updates records after changes.



OBSERVATIONS:

- In Level 0 DFD, the entire Hospital Management System is depicted as a single interaction between external entities (Patients, Doctors, Admins) and the system itself.
- Level 1 DFD breaks down the system into specific sub-processes: Appointment Management, Billing Management, Record Management, and Staff Management. Each sub-process receives inputs and generates outputs involving the external entities and relevant data stores.
- Level 2 DFD provides even more detailed insights into a single sub-process (Appointment Management in this case), showing the steps for scheduling, updating, and storing appointment data.

CONCLUSIONS:

The Data Flow Diagrams (DFDs) for the Hospital Management System illustrate the system's structure and data flow, from a high-level overview (Level 0) to more detailed sub-processes (Level 1 and Level 2). By modelling these levels of the DFD, we can better understand the roles of external entities, how data flows between processes, and the interaction of data stores. This process is crucial for designing an efficient, user- friendly system that ensures seamless operations within the hospital.

EXPERIMENT 5

AIM: To make the Entity Relationship Diagram for Hospital Management System

THEORY:
Entity-Relationship Diagrams (ERDs) are graphical representations used to model the data structures of a database in terms of entities, their attributes, and the relationships between these entities. They are a key tool in database design, helping to visualize and plan the structure of a database.

Entities:
- Represent major objects or concepts within the system.
- Depicted as rectangles.
- Examples include Customer, Order, Product, etc.

Attributes:
- Properties or characteristics of entities.
- Depicted as ovals connected to their entity with a line.
- Examples include CustomerID, Name, OrderDate, etc.

Relationships:
- Depict how entities interact with each other.
- Represented by diamonds (in some notations) or simply labelled lines connecting entities.
- Relationships can have attributes too (e.g., a Date attribute on an Order relationship).

Cardinality and Participation:
- Cardinality indicates the number of instances of one entity that can be associated with an instance of another entity.
- Common cardinalities are one-to-one (1:1), one-to-many (1), and many-to-many (M).
- Participation constraints specify whether all or only some instances participate in the relationship (total or partial participation).

PROCEDURE
1. Identify Entities
List out the main entities involved in the hospital system, focusing on the people, processes, and resources.
Common entities include:
o Patient
o Doctor
o Appointment
o Department
o Medical Record

2. Define Attributes for Each Entity
Determine the specific details or characteristics that need to be stored for each entity.
Examples:

- Patient: Patient_ID (Primary Key), Name, Age, Gender, Address, Contact Info, Medical History
- Doctor: Doctor_ID (Primary Key), Name, Specialty, Contact Info, Availability
- Appointment: Appointment_ID (Primary Key), Date, Time, Status, Patient_ID (Foreign Key), Doctor_ID (Foreign Key)
- Medical Record: Record_ID (Primary Key), Diagnosis, Treatment Details, Medication, Patient_ID (Foreign Key)

3. Establish Relationships Between Entities
Identify how the entities interact with each other.
Examples of relationships:
o Patient–Appointment: A patient can book multiple appointments (1-to-many).
o Doctor–Appointment: A doctor can attend multiple appointments (1-to-many).
o Patient–Medical Record: A patient has one medical record (1-to-1).
o Doctor–Department: A doctor can work in one or more departments (many-to-many).

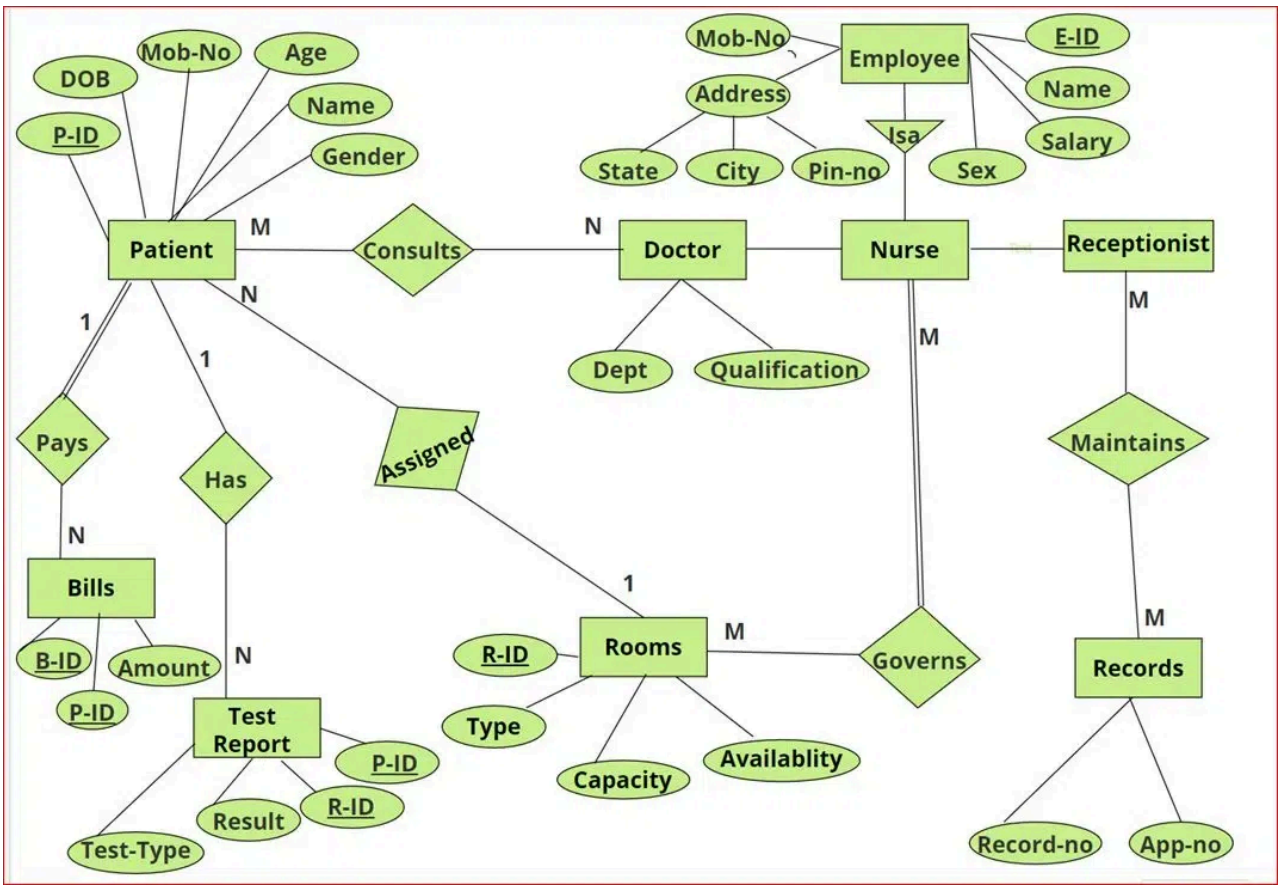
4. Determine Cardinality
Cardinality defines the nature of relationships (one-to-one, one-to-many, many-to-many).
For each relationship, define the cardinality:

- A Patient can have many Appointments, but an Appointment is associated with only one Patient (1-to- many).
- A Doctor can be assigned to multiple Appointments, and an Appointment will always have one Doctor (1-to-many).

5. Assign Primary and Foreign Keys

- Assign primary keys to uniquely identify each entity.
- Assign foreign keys to create relationships between entities by referencing the primary key of another entity.
- Example: In the Appointment entity, Patient_ID and Doctor_ID act as foreign keys to link Patient and Doctor entities.

6. Review and Finalise the Diagram
Review the entire ER diagram to ensure all required data and relationships are accurately represented.
Make sure the structure is efficient and reflects the hospital management workflow.



OBSERVATIONS:

1. Entities:

The diagram identifies the main entities such as Patient, Doctor, Appointment, Medical Record, Billing, and Department. These entities represent core components of the hospital system.

2. Attributes:

Each entity is associated with specific attributes that store relevant data.

o Patient has attributes like Patient_ID (primary key), Name, Age, Gender, and Address.

o Doctor includes Doctor_ID (primary key), Name, Specialty, and Contact Info.

o Appointment includes Appointment_ID (primary key), Date, Time, Status, along with foreign keys linking to the Patient and Doctor entities.

3. Relationships:

The diagram shows relationships between entities, such as:

o Patient has many Appointments (1-to-many).

o Doctor attends many Appointments (1-to-many).

o Patient is linked to a single Medical Record (1-to-1).

o Doctor works in one or more Departments (many-to-many).

CONCLUSION:

The Entity-Relationship (ER) diagram for the Hospital Management System serves as a comprehensive and structured representation of the hospital's data model. It effectively outlines the relationships between core entities such as Patients, Doctors, Appointments, Medical Records, Billing, and Departments, providing a clear understanding of how these components interact with one another.

EXPERIMENT 6

AIM: Software Requirement Specification for Hospital Management System

---

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for the Hospital Management System (HMS), which provides a digital platform for managing various administrative and clinical functions in a hospital setting. It streamlines processes like patient registration, appointment scheduling, billing, and record-keeping.

1.2 Scope

The HMS will enable:

- **Patients** to book appointments, access their medical records, and make payments.
- **Doctors** to view and manage patient records and appointments.
- **Administrative staff** to handle patient registration, billing, and staff scheduling.
- **Management** to oversee hospital operations, monitor patient and staff activities, and generate reports.

1.3 Definitions, Acronyms, and Abbreviations

- **HMS**: Hospital Management System.
- **User**: Anyone who interacts with the system (Patient, Doctor, Staff, Admin).
- **Admin**: User responsible for system-wide settings and access control.
- **GDPR**: General Data Protection Regulation.
- **HIPAA**: Health Insurance Portability and Accountability Act.

1.4 References

- IEEE Standard 830-1998 for Software Requirements Specifications.
- Relevant healthcare standards (GDPR, HIPAA).

1.5 Overview

This document provides a comprehensive description of the system architecture, interfaces, functionality, and constraints of the HMS.

---

2. Overall Description

2.1 Product Perspective

The HMS is a centralized, web-based system aimed at automating hospital operations and improving data accessibility and security.

2.1.1 System Interfaces

- **Web Server**: Manages user interactions and request handling.
- **Database Server**: Stores patient, appointment, billing, and staff data.

2.1.2 User Interfaces

- **Patient UI**: Allows appointment booking, record access, and payment viewing.
- **Doctor UI**: Enables viewing of patient records and managing appointments.
- **Admin UI**: For managing user access, hospital records, and system configuration.

2.1.3 Hardware Interfaces

Standard web server and database server, with no specific hardware requirements for end users.

2.1.4 Software Interfaces

- **Database**: MySQL/PostgreSQL for managing patient, billing, and scheduling data.
- **Payment Gateway API**: To facilitate online payment of bills.
- **Email API**: For sending appointment confirmations and updates.

2.1.5 Communication Interfaces

- **HTTPS**: For secure communication.
- **RESTful APIs**: For third-party integrations like payment gateways.

2.1.6 Memory Constraints

- Minimum 8GB RAM on the server to handle large datasets and user queries.

2.1.7 Operations

- Appointment booking, patient registration, billing, and secure data access.

2.1.8 Site Adaptation Requirements

The HMS should be adaptable to various hospital sizes and support future module additions.

2.2 Product Functions

- **Patient Registration and Management**: Register new patients and maintain profiles.
- **Appointment Scheduling**: Enable patients to book, cancel, or reschedule appointments.
- **Billing and Payments**: Generate and track bills, enable online payments.
- **Staff and Doctor Management**: Manage work schedules and availability.

2.3 User Characteristics

- **Patients**: Non-technical users needing easy access to records and appointment scheduling.
- **Doctors**: Technically capable of accessing patient records and schedules.
- **Admins**: Technically adept, with a background in data management.

2.4 Constraints

- Compliance with GDPR and HIPAA for data security and privacy.
- System scalability to handle growing patient and hospital data.

2.5 Assumptions and Dependencies

- Stable internet access for cloud-based storage.
- Reliable third-party services (e.g., payment gateways).

2.6 Apportioning of Requirements

- **High Priority**: Patient registration, appointment scheduling, billing.
- **Medium Priority**: Reporting and analytics.
- **Low Priority**: Integration with advanced health monitoring tools.

---

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- **Patient Portal**: Appointment booking, payment viewing, and record access.
- **Doctor Portal**: Patient data access, appointment management.
- **Admin Dashboard**: Manage hospital data, system settings, and reports.

3.1.2 Hardware Interfaces

Standard web server setups with database storage.

3.1.3 Software Interfaces

- **Database**: MySQL/PostgreSQL for all patient, billing, and appointment data.
- **Payment Gateway API**: To enable secure online payments.
- **Email API**: For communication with patients.

3.1.4 Communication Interfaces

- **HTTPS**: Secure user interactions.
- **API Integration**: For payments and notifications.

3.2 Functional Requirements

- **User Registration and Authentication**: Secure login for all users.
- **Patient Management**: Registration, profile updates, and secure data storage.
- **Appointment Scheduling**: Allow patients to book and manage appointments.
- **Billing System**: Automatic billing based on services provided, with payment history.
- **Notifications**: Appointment reminders and updates.
- **System Logging**: Record all user activities for audit purposes.

3.3 Performance Requirements

- Must support up to 2,000 concurrent users during peak hours.
- Response time under 2 seconds for standard requests.

3.4 Design Constraints

- Must encrypt sensitive data and ensure secure login.
- Adherence to healthcare privacy standards like HIPAA.

3.5 Software System Attributes

- **Reliability**: 99% uptime.
- **Security**: Secure user authentication, data encryption.
- **Scalability**: Support growing hospital data and user numbers.
- **Maintainability**: Modular design for easy updates.

3.6 Logical Database Requirements

- **Patients Table**: Stores patient information.
- **Appointments Table**: Manages appointment data.
- **Billing Table**: Tracks service fees and payments.
- **Staff Table**: Stores doctor and staff details.

3.7 Other Requirements

- Compliance with GDPR for data handling.
- Integration with a payment gateway for online bill processing.

Conclusion

This SRS outlines the essential requirements for a Hospital Management System that can streamline hospital operations, improve patient care, and ensure data security and compliance.