

**HOSPITAL MANAGEMENT SYSTEM**

**BY**

**USING SQL AND JAVA**

# A PROJECT REPORT

Submitted by

RAVINDERAN(231501133)

ROHITH S (231501136)

SAI SENTHIL (231501140)

# CS23332 - DATABASE MANAGEMENT SYSTEMS

Department of Artificial Intelligence and Machine Learning

Rajalakshmi Engineering College, Thandalam

BONAFIDE CERTIFICATE

This is to certify that the Mini project work titled “**HOSPITAL MANAGEMENT SYSTEM USING SQL AND JAVA**” done by , RAVINDERAN(231501133), ROHITH S (231501136), SAI SENTHIL(231501140) is a record of Bonafide work carried out by him/her under my supervision as a part of MINI PROJECT for the subject CS23332 - DATABASE MANAGEMENT SYSTEMS

by Department of Artificial Intelligence and Machine Learning.

SIGNATURE SIGNATURE

Dr.Sekar K M.E., Ph.D,

HEAD OF THE DEPARTMENT FACULTY IN CHARGE

Department of Artificial Intelligence Department of Artificial

Intelligence and Machine Learning, and Machine Learning,

Rajalakshmi Engineering College, Rajalakshmi EngineerinCollege,

Thandalam, Thandalam,

Chennai-602 105. Chennai- 602 105.

Submitted for the project viva-voce examination held on .

INTERNAL EXAMINER EXTERNAL **EXAMINER**

**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Chapter** | **Page Number** |
| 1. | ABSTRACT |  |
| 2. | INTRODUCTION |  |
| 3. | RELATED WORK |  |
| 4. | ER DIAGRAM |  |
| 5. | IMPLEMENTATION |  |
| 6. | PROGRAM CODE |  |
| 7. | CONCLUSION |  |
| 8. | RESULT AND DISCUSION |  |
| 9. | REFERENCES |  |

# CHAPTER 1

# ABSTRACT

# This project focuses on developing a comprehensive Hospital Management System (HMS) to streamline hospital operations by managing patient data, appointments, medical records, staff details, and billing processes. Utilizing Java for the frontend and SQL for the backend, the system ensures a robust, modular, and user-friendly interface while providing secure and efficient data handling. Key functionalities include real-time appointment scheduling, electronic medical record management, automated billing, and staff administration, all designed to enhance operational productivity and improve patient care. Role-based access control ensures data security and compliance with industry standards, while reporting and analytics support informed decision-making. By automating routine tasks and centralizing information, this HMS reduces administrative burden, minimizes errors, and facilitates faster patient registration, ultimately delivering a scalable solution that enhances overall hospital efficiency and patient satisfaction.

# 

# CHAPTER 2

# INTRODUCTION

Healthcare institutions handle vast amounts of data daily, encompassing critical information such as patient records, staff details, medical histories, appointments, and billing transactions. Managing this data manually often leads to inefficiencies, inaccuracies, and delays that can compromise patient care and disrupt hospital operations. Human errors in record-keeping, scheduling conflicts, and billing discrepancies are common challenges in manual systems, resulting in reduced productivity and potential financial losses.

The proposed Hospital Management System (HMS) addresses these challenges by automating core administrative and clinical processes, enhancing both accuracy and operational efficiency. The system is designed to centralize data management, facilitating seamless access and retrieval of information across departments. By employing Java for the frontend, the HMS delivers a user-friendly, interactive interface that accommodates various roles within the hospital, such as administrators, medical staff, and patients. This ensures intuitive navigation and functionality tailored to different user needs, enhancing overall usability.

On the backend, SQL databases are utilized to store, retrieve, and manage data securely and efficiently. The relational database structure supports complex queries and ensures data integrity, enabling real-time updates and accurate information sharing. Robust data security protocols protect sensitive information, ensuring compliance with healthcare data regulations and safeguarding patient confidentiality.

Key functionalities of the system include automated appointment scheduling, electronic medical record (EMR) management, and streamlined billing processes. These features not only reduce the administrative workload but also minimize errors associated with manual data entry. Real-time data access improves decision-making for medical staff, while automated billing reduces financial discrepancies and enhances transparency. Overall, the HMS provides a scalable and reliable solution that optimizes hospital workflows, improves resource allocation, and elevates patient care standards.

**Project Overview:**

The Hospital Management System (HMS) is designed to streamline and automate key hospital operations, including patient data management, appointment scheduling, medical records, staff details, and billing. Developed with Java for the frontend and SQL for the backend, the system provides a secure, user-friendly interface and robust data handling capabilities. By centralizing information and automating processes, the HMS reduces inefficiencies and errors associated with manual management. Key features include real-time access to patient records, automated appointment scheduling, and simplified billing processes. Role-based access control ensures data security and compliance with healthcare standards. Overall, this scalable solution enhances operational productivity, optimizes resource management, and improves patient care.

**Key Features and Benefits:**

1. Patient Data Management:  
   This module securely stores and organizes comprehensive patient information, including personal details, medical history, treatment plans, diagnostic reports, and prescriptions. Authorized personnel can quickly access or update records, ensuring that medical staff have accurate, real-time data to make informed decisions, ultimately improving patient care.
2. Appointment Scheduling:  
   The system facilitates real-time scheduling of patient appointments, allowing both patients and staff to book, reschedule, or cancel appointments seamlessly. It minimizes scheduling conflicts, reduces wait times, and sends automated reminders via email or SMS to both patients and healthcare providers, ensuring better time management.
3. Electronic Medical Records (EMR):  
   EMR functionality digitizes patient health records, making them easily accessible to authorized users. This feature includes detailed patient charts, lab results, and imaging reports. EMRs ensure data consistency and continuity across different departments, improving collaboration and patient safety.
4. Staff Management:  
   This module maintains a database of all hospital staff, including doctors, nurses, administrative personnel, and support staff. It handles staff scheduling, role assignments, leave management, and performance tracking, ensuring that human resources are effectively managed and allocated.
5. Billing and Financial Management:  
   The billing system automates financial processes, including the generation of invoices, tracking of payments, and management of insurance claims. It reduces errors associated with manual billing, provides transparency, and ensures that all financial transactions comply with accounting standards. Integration with payment gateways simplifies online payments.
6. Role-Based Access Control:  
   To protect sensitive data, the system implements role-based access control (RBAC), granting specific permissions based on user roles such as doctors, nurses, administrators, and patients. This ensures that only authorized users can access or modify certain information, enhancing data security and maintaining compliance with healthcare regulations.
7. Inventory Management:  
   This feature monitors hospital inventory, including medical supplies, medications, and equipment. It tracks stock levels in real-time, sends alerts for low inventory, and generates purchase orders when necessary. Proper inventory management prevents stockouts and ensures that critical supplies are always available.
8. Reporting and Analytics:  
   The system generates detailed reports on various aspects of hospital operations, such as patient admission rates, financial performance, staff productivity, and resource utilization. Advanced analytics provide insights into trends and patterns, aiding in strategic planning and data-driven decision-making.

**Technological Aspects:**

1. Frontend Development: Java  
   The frontend of the Hospital Management System is built using Java, ensuring a responsive, robust, and user-friendly interface. Java’s platform independence (via the Java Virtual Machine) allows the system to run across various operating systems. Its object-oriented design facilitates modular development, simplifying code maintenance and future upgrades. Java frameworks like JavaFX or Swing can be used to develop intuitive graphical user interfaces (GUIs), enhancing user interaction.
2. Backend Development: SQL Databases  
   The system employs SQL databases (e.g., MySQL, PostgreSQL) for backend data management. SQL ensures structured data storage, allowing complex queries, relational integrity, and efficient retrieval of large datasets. The use of SQL facilitates seamless data integration, ensuring consistency across different modules like patient records, billing, and inventory management. Additionally, stored procedures and triggers enhance data processing efficiency and security.
3. Middleware Integration  
   Middleware components, such as APIs (Application Programming Interfaces), facilitate communication between the frontend and backend. These APIs enable the system to interact with external services, such as laboratory systems, radiology departments, and payment gateways, ensuring smooth data exchange. RESTful APIs are commonly used to ensure scalability and platform-agnostic connectivity.
4. Security Technologies
   * Data Encryption: Sensitive information, such as patient records and financial data, is encrypted using industry-standard protocols like AES (Advanced Encryption Standard) and SSL/TLS.
   * Role-Based Access Control (RBAC): Ensures that users access only the data relevant to their roles, enhancing security and regulatory compliance.
   * Authentication Mechanisms: Multi-factor authentication (MFA) and password hashing (e.g., bcrypt) are implemented to strengthen user verification processes.
5. Cloud Computing Support  
   Cloud platforms like AWS, Azure, or Google Cloud can host the system, providing benefits such as data redundancy, remote access, and scalability. Cloud deployment reduces infrastructure costs and ensures high availability, while cloud-based data backup services prevent data loss.
6. Data Analytics and Reporting  
   The system incorporates data analytics tools to generate actionable insights. Business Intelligence (BI) tools, such as Tableau or Power BI, can be integrated for advanced reporting and visualization. This helps in monitoring hospital performance metrics, optimizing resource allocation, and improving decision-making.
7. User Interface Design  
   The system's user interface (UI) is designed with frameworks such as JavaFX or Spring Boot for web applications. Responsive design principles ensure compatibility with various devices, including desktops, tablets, and mobile phones. User experience (UX) considerations focus on intuitive navigation and accessibility.
8. Integration with IoT and Medical Devices  
   For advanced functionalities, the system can integrate with Internet of Things (IoT) devices used in patient monitoring, diagnostics, and inventory tracking. This enables real-time data collection from medical equipment, improving accuracy and patient outcomes.
9. Version Control and Deployment  
   Git is used for version control, facilitating collaborative development and efficient tracking of changes. Continuous Integration/Continuous Deployment (CI/CD) pipelines ensure smooth deployment processes, reducing downtime and enabling rapid updates.
10. Compliance and Standards  
    The system adheres to healthcare standards such as HIPAA (Health Insurance Portability and Accountability Act) for data security and HL7 (Health Level Seven) for medical data interoperability. Compliance with these standards ensures legal adherence and smooth data exchange with other healthcare systems.

**Purpose and Objectives:**

The primary purpose of the Hospital Management System (HMS) is to streamline and automate various administrative, clinical, and financial processes within healthcare institutions. By centralizing patient data, appointment scheduling, medical records, staff management, and billing operations, the system aims to enhance efficiency, accuracy, and patient care quality. It eliminates the complexities and errors associated with manual data handling, ensuring secure and seamless information flow across departments. Ultimately, the HMS supports healthcare providers in delivering better services, reducing operational costs, and improving overall hospital management.

1. **Streamline Data Management:**  
   Centralize and automate patient records, medical histories, appointment schedules, and billing details to ensure quick access and reduce manual errors.
2. **Improve Operational Efficiency:**  
   Automate administrative tasks such as patient registration, staff scheduling, and billing to reduce workloads and increase overall hospital productivity.
3. **Enhance Patient Care:**  
   Provide healthcare professionals with real-time access to accurate patient information, enabling better decision-making and more personalized treatment.
4. **Ensure Data Security and Compliance:**  
   Implement robust security features to protect sensitive patient data and comply with healthcare regulations, such as HIPAA.
5. **Optimize Resource Management:**  
   Efficiently manage hospital resources, including staff, medical supplies, and equipment, to prevent shortages, reduce waste, and ensure smooth hospital operations.

**Conclusion:**

The **Hospital Management System (HMS)** is a crucial tool for modern healthcare institutions, designed to enhance operational efficiency, improve patient care, and streamline administrative tasks. By centralizing and automating processes such as patient data management, appointment scheduling, medical records handling, billing, and resource allocation, the system reduces errors, minimizes delays, and optimizes hospital workflows. With robust security measures in place, the HMS ensures compliance with healthcare regulations, protecting sensitive patient information. Ultimately, the system leads to improved patient outcomes, cost savings, and greater overall efficiency, making it an indispensable solution for healthcare providers striving to deliver high-quality, patient-centered care.

**CHAPTER 3**

**RELATED WORK**

**Limitations of Hospital Management System**

1. High Initial Setup Costs:  
   The implementation of a Hospital Management System often involves a significant initial financial investment. This includes the cost of purchasing the software, upgrading hardware infrastructure, and setting up the necessary networking systems to support the HMS. In addition, hospitals may need to hire IT specialists or consultants to ensure proper installation and configuration. Furthermore, migrating data from legacy systems and integrating the new system with existing hospital processes can incur additional costs. For smaller healthcare institutions with limited budgets, these expenses can pose a major challenge.
2. Complexity in Implementation:  
   Integrating a new Hospital Management System into the daily operations of a healthcare facility can be a complex and disruptive process. The customization of the system to meet specific needs, such as adapting to the hospital’s workflows and patient care protocols, can take time. Data migration from older systems is also often a challenge, as data may be fragmented or stored in incompatible formats. During the initial stages of implementation, staff may experience temporary disruptions in their workflow, which could impact patient care and operational efficiency.
3. Dependency on Technology:  
   The Hospital Management System relies heavily on technology, meaning any issues with the system’s infrastructure—such as server crashes, network failures, or software glitches—can halt hospital operations. This could lead to delays in processing patient information, scheduling appointments, or even billing patients. In extreme cases, technical failures could result in a loss of critical data. Therefore, it is essential for hospitals to have robust IT support, backup systems, and disaster recovery plans in place to minimize the risk of downtime and maintain continuous operations.
4. User Resistance and Training Needs:  
   Transitioning from manual processes to a digital system often meets with resistance from hospital staff, especially those who have been accustomed to traditional methods. The complexity of the system may intimidate some users, and overcoming this resistance requires comprehensive training programs. Staff members must be trained not only on how to use the system but also on how to adapt to new workflows and responsibilities. The training process can be time-consuming and costly, requiring ongoing support to ensure that all users become proficient in the system. Any gaps in training can lead to inefficiencies or errors in using the system.
5. Security and Data Privacy Concerns:  
   Hospitals manage a vast amount of sensitive data, including patient medical records, personal information, and financial details. Despite implementing security measures such as encryption and firewalls, the risk of data breaches remains a concern. Cyberattacks targeting healthcare institutions are on the rise, and any lapse in security could result in unauthorized access to confidential patient information. Hospitals must continually update their security protocols, conduct regular audits, and ensure compliance with industry regulations like HIPAA (Health Insurance Portability and Accountability Act) to protect patient privacy and avoid legal consequences. Additionally, improper handling of data by hospital staff, such as unauthorized access or data entry errors, can also compromise patient confidentiality.

These limitations illustrate the challenges associated with adopting and maintaining a Hospital Management System. While the benefits of automating and streamlining hospital operations are significant, addressing these limitations through careful planning, investment in training, and strong security measures is crucial for the successful implementation and long-term success of the system.

**Importance of User-Centered Design:**

**1) Improved Usability and Efficiency:**  
User-Centered Design (UCD) places the needs, preferences, and limitations of end users at the forefront of system design. In a Hospital Management System (HMS), the primary users include doctors, nurses, administrative staff, and patients. By focusing on their requirements and ensuring the system is intuitive and easy to use, UCD helps streamline workflows, minimize the learning curve, and increase user efficiency. An easy-to-navigate system with clearly organized interfaces allows healthcare providers to quickly access and update patient information, manage appointments, and complete other tasks, which ultimately enhances the overall operational efficiency of the hospital.

**2) Enhanced User Adoption and Satisfaction:**  
Healthcare professionals are often resistant to adopting new technologies, particularly when systems are complex or poorly designed. A user-centered approach ensures that the system is designed with input from the actual users, resulting in a product that aligns with their daily tasks and workflows. This fosters higher user acceptance and satisfaction, as the system feels more relevant and easier to use. When users find the system intuitive and supportive of their needs, they are more likely to adopt it enthusiastically, leading to smoother integration into hospital operations.

**3) Reduced Errors and Increased Accuracy:**  
In the healthcare industry, errors in patient data management, medication orders, or appointment scheduling can have serious consequences. A well-designed HMS, guided by UCD principles, reduces the risk of such errors by providing clear, error-resistant interfaces. Features like auto-complete fields, error validation, and intuitive navigation help prevent mistakes that might occur when users are rushed or under stress. By simplifying interactions and making key information readily accessible, the system can enhance decision-making and minimize human error, which is critical in delivering high-quality patient care.

**4) Better Patient Care and Experience:**  
A user-centered design directly impacts the quality of care provided to patients. Healthcare professionals can access patient records, schedules, and medical histories more quickly, ensuring they have the most up-to-date and complete information. This allows for faster, more accurate diagnoses and treatments. Additionally, a well-designed system can improve patient communication and engagement by providing them with easy access to appointment schedules, test results, and other relevant information. A seamless experience for patients and staff contributes to better healthcare outcomes and a more positive patient experience.

**5) Flexibility and Customization:**  
Hospitals have diverse needs, with different departments (e.g., emergency care, outpatient services, administration) requiring distinct functionalities within the HMS. A user-centered design allows for flexibility, where the system can be customized to meet specific department requirements or individual preferences. For instance, medical staff may need a more streamlined interface to access patient records, while administrative staff may need detailed reporting features. By designing the system with user flexibility in mind, the HMS can be tailored to suit a variety of needs, optimizing its effectiveness across various hospital functions.

In summary, applying User-Centered Design principles to a Hospital Management System ensures that the system is not only functional but also practical, user-friendly, and aligned with the needs of its users. This approach improves adoption, reduces errors, enhances patient care, and ensures that the system remains an efficient tool in the ever-evolving healthcare environment.

**Our Approach: Building on User-Centered Design Principles:**

1. **Requirement Analysis and Planning**

The first step in building a Hospital Management System (HMS) involves a detailed requirement analysis. This phase focuses on gathering insights from all relevant stakeholders, including doctors, nurses, administrative staff, IT teams, and even patients. Each user group has specific needs and challenges, and understanding these is crucial to developing a system that addresses those pain points. The primary goal is to define the core features of the system, such as patient data management, appointment scheduling, billing, medical records, and inventory management. This step also includes reviewing regulatory compliance requirements, such as HIPAA, to ensure the system meets all necessary legal and security standards. Effective requirement analysis helps in creating a comprehensive blueprint for the system, guiding the design and development phases.

1. **System Design and Architecture**

Once the requirements are gathered, the system design phase begins. The architecture of the Hospital Management System needs to be flexible, scalable, and capable of handling complex hospital workflows. One key approach is using **User-Centered Design (UCD)** principles, ensuring the system is intuitive and easy to navigate for all user groups. The design should be based on input from actual users to ensure that the system aligns with their daily tasks. At the same time, the database design is critical, as it must support large volumes of data, including patient information, medical histories, staff details, and billing records. A well-structured database, typically using relational databases like **MySQL** or **PostgreSQL**, ensures data integrity and supports complex queries. Additionally, a modular system architecture is ideal, allowing easy updates and the ability to add or modify features as hospital needs evolve.

1. **Technology Stack Selection**

Choosing the right technology stack is essential for the functionality, scalability, and security of the HMS. For the frontend, **Java** is often the preferred language due to its platform independence and robust security features. **JavaFX** or **Swing** can be used to develop rich graphical user interfaces (GUIs) that ensure a smooth and interactive user experience. On the backend, relational databases like **MySQL** or **PostgreSQL** provide structured and secure storage for hospital data, supporting complex queries and transactions. Security is a key consideration, so encryption standards like **AES** for data at rest and **SSL/TLS** for data in transit are implemented to protect sensitive patient information. Additionally, middleware technologies, such as **RESTful APIs**, ensure smooth integration between the frontend and backend, enabling data exchange with external systems like lab equipment or pharmacy software.

1. **Development and Implementation**

The actual development of the Hospital Management System follows the system design and architecture. An **Agile development methodology** is commonly adopted, enabling the team to develop the system in iterative cycles or sprints. This approach allows for continuous feedback from stakeholders, ensuring that the system evolves to meet user needs. The backend, including database integration, user authentication, and business logic, is implemented first, ensuring that all data transactions are handled securely and efficiently. Afterward, the frontend user interface is developed, focusing on ease of use and ensuring that all user groups can navigate the system without difficulties. The implementation of core features like patient data management, appointment booking, and billing is done with close attention to user requirements and operational efficiency.

1. **Testing and Quality Assurance**

Testing is a critical phase in the development process, ensuring that the system functions correctly, is secure, and meets user expectations. **Unit testing** is performed to verify that individual components work as expected, while **integration testing** checks the interaction between different modules. In addition, **User Acceptance Testing (UAT)** is conducted to allow real users to interact with the system, providing valuable feedback on usability and functionality. Any issues discovered during this phase are addressed before the system goes live. Security testing is particularly crucial in a Hospital Management System, as the system handles sensitive patient data. Penetration testing, encryption validation, and compliance checks with healthcare standards like **HIPAA** ensure that the system is secure against potential breaches and data leaks.

1. **Deployment and Training**

Once the system is thoroughly tested, it is ready for deployment. Depending on the hospital's infrastructure, the HMS can be deployed on local servers or cloud platforms such as **AWS** or **Azure**. Cloud deployment provides flexibility, scalability, and remote access for users, making it easier for healthcare professionals to access the system from various locations. After deployment, **staff training** becomes essential to ensure that all users, including doctors, nurses, and administrative personnel, can effectively use the system. Training sessions should cover everything from basic system navigation to more advanced features like report generation or medical record updates. Providing comprehensive training ensures smooth adoption of the system and maximizes its potential benefits.

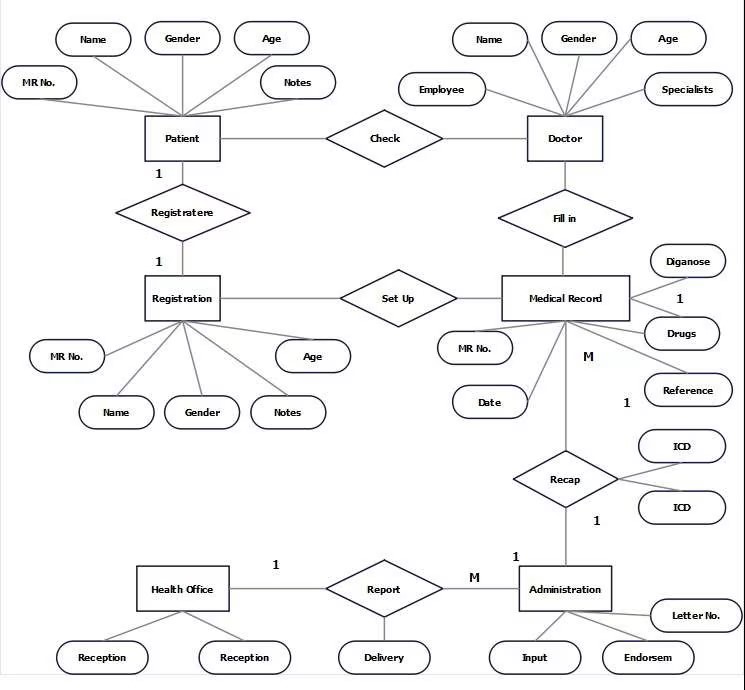
1. **Post-Deployment Support and Maintenance**

After the system is deployed and in use, ongoing **post-deployment support and maintenance** are essential to ensure its continued effectiveness. This phase involves monitoring the system's performance, addressing any technical issues, and applying regular updates to improve functionality or patch security vulnerabilities. Hospitals need a reliable support team to resolve any issues promptly and minimize downtime. Gathering **user feedback** after deployment is also crucial to identify any areas for improvement. The system should be adaptable, allowing for modifications based on changing hospital needs or advancements in technology. Over time, the system may require upgrades or new features, so scalability and flexibility in the design are key to ensuring the HMS remains relevant and effective.

Building a Hospital Management System requires a methodical approach that incorporates careful planning, appropriate technology selection, and a focus on user needs. Each phase, from requirement analysis and system design to development, testing, and deployment, is essential in creating a system that enhances hospital operations, improves patient care, and meets regulatory standards. Post-deployment support ensures that the system evolves with the hospital’s needs, providing long-term value. By following this approach, healthcare institutions can implement a robust HMS that improves operational efficiency, reduces errors, and supports high-quality care delivery.

## CHAPTER 4

## ER DIAGRAM



**MODEL ARCHITECTURE:**

**START**

**Client Layer | | (Frontend - HTML, CSS, | | JavaScript)**

**Application Layer | | (Backend - PHP)**

**Database Layer | | (Database - MySQL)**

**Security Layer**

**STOP**

**Integeration Layer**

**(API and Middleware)**

## Flowchart Description:

## 1. Client Layer (Frontend)

## The client layer represents the user interface (UI) of the Hospital Management System, where users interact with the system. This layer is typically built using technologies like JavaFX or Swing for desktop applications or React or Angular for web-based applications. This layer communicates with the backend via APIs and presents data in a user-friendly manner. Different types of users interact with the system, including:

## Doctors and Healthcare Providers: Access patient records, medical histories, diagnostic information, prescriptions, and treatment plans.

## Nurses: View patient records, update medical charts, and manage patient care.

## Administrative Staff: Manage patient registrations, appointment scheduling, and billing.

## Patients: View appointments, access medical results, and update personal information.

## The client layer also provides real-time updates to users, such as alerts, notifications, and system status.

## 2. Application Layer (Backend)

## The application layer handles all business logic, processes user requests, and enforces the core functionalities of the HMS. This layer is typically developed using Java (with frameworks like Spring Boot or Java EE) or other backend technologies like Node.js or Python. The application layer includes the following key components:

## Authentication and Authorization: Manages user logins, access control, and permissions. Ensures that only authorized personnel can access specific information (e.g., patient data, medical records).

## Patient Management: Handles patient registrations, updates, and retrieval of personal and medical information.

## Appointment Management: Manages the scheduling, rescheduling, and cancellation of appointments.

## Billing and Payment: Tracks services rendered, calculates costs, and generates invoices for patients.

## Inventory Management: Monitors medical supplies, equipment usage, and reordering of stock.

## Reports and Analytics: Generates reports related to hospital performance, financial data, patient demographics, and treatment outcomes.

## 3. Database Layer (Data Storage)

## The database layer is where all data, including patient information, medical histories, appointments, billing details, and inventory data, is stored. This layer is responsible for ensuring data integrity, security, and efficient retrieval. The database is typically a relational database management system (RDBMS), such as MySQL, PostgreSQL, or Oracle Database. The database layer is divided into several components:

## Patient Database: Stores patient personal information, medical history, diagnoses, treatments, and test results.

## Staff Database: Contains details of hospital staff, including medical professionals, nurses, and administrative personnel.

## Appointment Database: Manages data related to patient appointments, scheduling, cancellations, and consultations.

## Billing Database: Stores information related to patient billing, including service charges, payment status, and transaction history.

## Inventory Database: Tracks hospital supplies, equipment, and medication stock levels.

## A well-organized and normalized database ensures that data is not duplicated, reducing redundancy and improving query performance.

## 4. Integration Layer (API and Middleware)

## The integration layer acts as a bridge between the client (frontend) and the backend systems. It handles the communication between the different layers of the system and external services. The integration layer typically uses RESTful APIs or GraphQL for client-server communication. Key components of the integration layer include:

## API Gateway: An entry point for client requests that routes them to the appropriate service or microservice. It handles request load balancing, API versioning, and security.

## Middleware Services: Middleware services are responsible for orchestrating complex tasks, like checking for user authorization, logging, error handling, and ensuring communication between various subsystems.

## External System Integration: The HMS may integrate with external systems like laboratory systems, pharmacy systems, or electronic health records (EHR) through RESTful APIs or other integration protocols (HL7, FHIR) to exchange data seamlessly.

## 5. Security Layer

## The security layer is critical for safeguarding sensitive patient data and maintaining compliance with regulations like HIPAA. This layer implements various security protocols and features to ensure that only authorized users can access and manipulate data. Key components of the security layer include:

## Encryption: Ensures that data at rest (in the database) and in transit (between client and server) is encrypted using robust encryption algorithms like AES for storage and SSL/TLS for communication.

## Access Control: Role-based access control (RBAC) or attribute-based access control (ABAC) ensures that different types of users have different levels of access to system features and data.

## Authentication: Multi-factor authentication (MFA) can be implemented to further secure user logins.

## Audit Logs: Maintain logs of user activities for compliance and security monitoring.

## 6. Reporting and Analytics Layer

## The reporting and analytics layer is designed to provide actionable insights based on hospital data. It aggregates data from various sources, such as patient records, billing data, and inventory management, to produce reports and visualizations for hospital management. This layer uses business intelligence (BI) tools like Power BI, Tableau, or JasperReports. Key components include:

## Financial Reports: Generate revenue reports, expense breakdowns, and billing summaries.

## Operational Reports: Analyze hospital efficiency, patient flow, and resource utilization.

## Clinical Reports: Provide insights into patient outcomes, treatment effectiveness, and health trends.

## These reports help hospital administrators and healthcare professionals make informed decisions.

## CHAPTER 5

## IMPLEMENTATION

**1. Database Design**

The database is the core for storing hospital data like patient records, appointments, and billing.

* **Core Tables:**
  + **Patients**: Stores personal and medical info (e.g., patient\_id, name, address).
  + **Staff**: Stores staff details (e.g., staff\_id, role, specialization).
  + **Appointments**: Tracks patient appointments (e.g., appointment\_id, patient\_id, date).
  + **Billing**: Records payment details (e.g., bill\_id, amount, payment\_status).
  + **Inventory**: Manages medical supplies (e.g., item\_id, name, stock).
* **Relationships**:
  + One-to-many (e.g., one patient can have multiple appointments).
  + Many-to-many (e.g., doctors treating multiple patients).

**2. Frontend Development**

The frontend is the user interface for hospital staff and patients to interact with the system.

* **Technologies**:
  + **Web**: HTML, CSS, JavaScript (React or Angular).
  + **Desktop**: JavaFX or Swing.
* **Key Features**:
  + **Patient Dashboard**: Patients can view appointments and medical records.
  + **Admin Panel**: Admins manage patients, appointments, and staff.
  + **Doctor’s Interface**: Doctors view patient history and schedule appointments.
  + **Forms**: Patient registration, appointment booking, billing.
* **UI/UX Design**:
  + Easy navigation, responsive design, and error handling.
  + Data validation for accurate input (e.g., email, phone number).

**3. Backend Development**

The backend handles business logic, data management, and system integration.

* **Technologies**:
  + Java (Spring Boot), Node.js (Express), Python (Django).
  + Database: MySQL, PostgreSQL.
* **Key Features**:
  + **Authentication**: Role-based access (admin, doctor, patient) using JWT.
  + **Patient Management**: APIs to add, update, and view patient details.
  + **Appointment Scheduling**: Manage appointments and check doctor availability.
  + **Billing**: Generate invoices and track payments.
  + **Inventory Management**: Track medical supplies and reorder when needed.
* **Security**:
  + SSL/TLS encryption for secure data transfer.
  + RBAC for restricting access to sensitive data.
* **API Endpoints**:
  + POST /api/login for user login.
  + GET /api/appointments for fetching appointments.
  + POST /api/billing to generate bills.

**CHAPTER 6**

**PROGRAM CODE**

1. **Java code**

import java.sql.\*;

import java.util.Scanner;

public class HospitalManagementSystem {

private Connection connection;

private Scanner scanner;

public HospitalManagementSystem() {

scanner = new Scanner(System.in);

try {

connection = DatabaseConnection.getConnection();

System.out.println("Database connected successfully!");

} catch (Exception e) {

System.out.println("Database connection error: " + e.getMessage());

}

}

public void addPatient() {

System.out.print("Enter patient name: ");

String name = scanner.nextLine();

System.out.print("Enter age: ");

int age = scanner.nextInt();

scanner.nextLine(); // Clear buffer

System.out.print("Enter gender: ");

String gender = scanner.nextLine();

System.out.print("Enter contact number: ");

String contact = scanner.nextLine();

System.out.print("Enter address: ");

String address = scanner.nextLine();

System.out.print("Enter symptoms: ");

String symptoms = scanner.nextLine();

System.out.print("Enter admission fee: ");

double admissionFee = scanner.nextDouble();

scanner.nextLine(); // Clear buffer

System.out.print("Enter admission date (yyyy-MM-dd): ");

String admissionDate = scanner.nextLine();

String sql = "INSERT INTO patients (name, age, gender, contact\_number, address, symptoms, admission\_fee, admission\_date) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, name);

stmt.setInt(2, age);

stmt.setString(3, gender);

stmt.setString(4, contact);

stmt.setString(5, address);

stmt.setString(6, symptoms);

stmt.setDouble(7, admissionFee);

stmt.setString(8, admissionDate);

stmt.executeUpdate();

System.out.println("Patient added successfully!");

} catch (SQLException e) {

System.out.println("Error adding patient: " + e.getMessage());

}

}

public void showAllPatients() {

String sql = "SELECT \* FROM patients";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

System.out.println("Patient Information:");

while (rs.next()) {

System.out.println("ID: " + rs.getInt("patient\_id"));

System.out.println("Name: " + rs.getString("name"));

System.out.println("Age: " + rs.getInt("age"));

System.out.println("Gender: " + rs.getString("gender"));

System.out.println("Contact: " + rs.getString("contact\_number"));

System.out.println("Address: " + rs.getString("address"));

System.out.println("Symptoms: " + rs.getString("symptoms"));

System.out.println("Admission Fee: " + rs.getDouble("admission\_fee"));

System.out.println("Admission Date: " + rs.getDate("admission\_date"));

System.out.println("Discharge Date: " + rs.getDate("discharge\_date"));

System.out.println();

}

} catch (SQLException e) {

System.out.println("Error retrieving patients: " + e.getMessage());

}

}

public void showAllDoctors() {

String sql = "SELECT \* FROM doctors";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

System.out.println("Doctor Information:");

while (rs.next()) {

System.out.println("ID: " + rs.getInt("doctor\_id"));

System.out.println("Name: " + rs.getString("name"));

System.out.println("Specialization: " + rs.getString("specialization"));

System.out.println("Contact: " + rs.getString("contact\_number"));

System.out.println("Availability: " + rs.getString("availability"));

System.out.println();

}

} catch (SQLException e) {

System.out.println("Error retrieving doctors: " + e.getMessage());

}

}

public void bookAppointment() {

try {

showAllDoctors();

System.out.print("\nEnter patient name: ");

String patientName = scanner.nextLine();

System.out.print("Enter doctor ID: ");

int doctorId = scanner.nextInt();

scanner.nextLine(); // Clear buffer

System.out.print("Enter date (yyyy-MM-dd): ");

String date = scanner.nextLine();

System.out.print("Enter time (HH:mm): ");

String time = scanner.nextLine();

int patientId = getPatientIdByName(patientName);

String sql = "INSERT INTO appointments (patient\_id, doctor\_id, appointment\_date, appointment\_time, status) VALUES (?, ?, ?, ?, 'Scheduled')";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

pstmt.setInt(2, doctorId);

pstmt.setString(3, date);

pstmt.setString(4, time);

pstmt.executeUpdate();

System.out.println("Appointment booked successfully!");

}

} catch (SQLException e) {

System.out.println("Error booking appointment: " + e.getMessage());

}

}

private int getPatientIdByName(String name) throws SQLException {

String sql = "SELECT patient\_id FROM patients WHERE name = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, name);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return rs.getInt("patient\_id");

} else {

throw new SQLException("Patient not found: " + name);

}

}

}

public static void main(String[] args) {

HospitalManagementSystem hms = new HospitalManagementSystem();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nHospital Management System");

System.out.println("1. Add Patient");

System.out.println("2. View All Patients");

System.out.println("3. View All Doctors");

System.out.println("4. Book Appointment");

System.out.println("5. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Clear buffer

switch (choice) {

case 1:

hms.addPatient();

break;

case 2:

hms.showAllPatients();

break;

case 3:

hms.showAllDoctors();

break;

case 4:

hms.bookAppointment();

break;

case 5:

System.out.println("Thank you for using the system!");

return;

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

}

1. **GUI Interface**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.\*;

public class HospitalManagementSystemGUI extends JFrame {

private Connection connection;

private JTextArea textArea;

private JTextField nameField, ageField, genderField, contactField, addressField, symptomsField, admissionFeeField, admissionDateField, dischargeField;

private JButton addPatientButton, updatePatientButton, showAllPatientsButton, showAllDoctorsButton, bookAppointmentButton, showAllAppointmentsButton, assignRoomButton, showAllRoomsButton;

private JButton generateBillButton, viewBillHistoryButton, makePaymentButton;

public HospitalManagementSystemGUI() {

try {

connection = DatabaseConnection.getConnection();

System.out.println("Database connected successfully!");

} catch (Exception e) {

System.out.println("Database connection error: " + e.getMessage());

}

setTitle("Hospital Management System");

setSize(900, 850);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setLayout(new BorderLayout());

setBackground(new Color(245, 245, 245));

JTabbedPane tabbedPane = new JTabbedPane();

tabbedPane.setFont(new Font("SansSerif", Font.BOLD, 15));

tabbedPane.setBackground(new Color(60, 100, 150));

tabbedPane.setForeground(Color.WHITE);

tabbedPane.addTab("Patients", createPatientPanel());

tabbedPane.addTab("Hospital Services", createServicesPanel());

tabbedPane.addTab("Billing", createBillingPanel());

textArea = new JTextArea();

textArea.setEditable(false);

textArea.setFont(new Font("SansSerif", Font.PLAIN, 14));

JScrollPane scrollPane = new JScrollPane(textArea);

scrollPane.setPreferredSize(new Dimension(850, 300));

scrollPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

add(tabbedPane, BorderLayout.NORTH);

add(scrollPane, BorderLayout.CENTER);

UIManager.put("TabbedPane.selected", new Color(72, 120, 192));

}

private JPanel createPatientPanel() {

JPanel panel = new GradientPanel(new Color(230, 240, 255), new Color(190, 210, 240));

JPanel patientInfoPanel = new JPanel(new GridLayout(9, 2, 8, 8));

patientInfoPanel.setOpaque(false);

patientInfoPanel.setBorder(BorderFactory.createTitledBorder(

BorderFactory.createEmptyBorder(), "Patient Information",

0, 0, new Font("SansSerif", Font.BOLD, 16), new Color(80, 120, 180)));

nameField = new JTextField(); ageField = new JTextField(); genderField = new JTextField(); contactField = new JTextField();

addressField = new JTextField(); symptomsField = new JTextField(); admissionFeeField = new JTextField();

admissionDateField = new JTextField(); dischargeField = new JTextField();

addLabelAndField(patientInfoPanel, "Name:", nameField);

addLabelAndField(patientInfoPanel, "Age:", ageField);

addLabelAndField(patientInfoPanel, "Gender:", genderField);

addLabelAndField(patientInfoPanel, "Contact Number:", contactField);

addLabelAndField(patientInfoPanel, "Address:", addressField);

addLabelAndField(patientInfoPanel, "Symptoms:", symptomsField);

addLabelAndField(patientInfoPanel, "Admission Fee:", admissionFeeField);

addLabelAndField(patientInfoPanel, "Admission Date:", admissionDateField);

addLabelAndField(patientInfoPanel, "Discharge Date:", dischargeField);

JPanel patientActionPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 20));

patientActionPanel.setOpaque(false);

addPatientButton = createStyledButton("Add Patient");

updatePatientButton = createStyledButton("Update Patient");

showAllPatientsButton = createStyledButton("View All Patients");

addPatientButton.addActionListener(e -> addPatient());

updatePatientButton.addActionListener(e -> updatePatient());

showAllPatientsButton.addActionListener(e -> showAllPatients());

patientActionPanel.add(addPatientButton);

patientActionPanel.add(updatePatientButton);

patientActionPanel.add(showAllPatientsButton);

panel.setLayout(new BorderLayout(10, 10));

panel.add(patientInfoPanel, BorderLayout.NORTH);

panel.add(patientActionPanel, BorderLayout.SOUTH);

return panel;

}

private JPanel createServicesPanel() {

JPanel panel = new GradientPanel(new Color(230, 240, 255), new Color(190, 210, 240));

panel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));

showAllDoctorsButton = createStyledButton("View All Doctors");

bookAppointmentButton = createStyledButton("Book Appointment");

showAllAppointmentsButton = createStyledButton("View All Appointments");

assignRoomButton = createStyledButton("Assign Room");

showAllRoomsButton = createStyledButton("View All Rooms");

showAllDoctorsButton.addActionListener(e -> showAllDoctors());

bookAppointmentButton.addActionListener(e -> bookAppointment());

showAllAppointmentsButton.addActionListener(e -> showAllAppointments());

assignRoomButton.addActionListener(e -> assignRoom());

showAllRoomsButton.addActionListener(e -> showAllRooms());

panel.add(showAllDoctorsButton);

panel.add(bookAppointmentButton);

panel.add(showAllAppointmentsButton);

panel.add(assignRoomButton);

panel.add(showAllRoomsButton);

return panel;

}

private JPanel createBillingPanel() {

JPanel panel = new GradientPanel(new Color(230, 240, 255), new Color(190, 210, 240));

panel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));

generateBillButton = createStyledButton("Generate Bill");

viewBillHistoryButton = createStyledButton("View Bill History");

makePaymentButton = createStyledButton("Make Payment");

generateBillButton.addActionListener(e -> generateBill());

viewBillHistoryButton.addActionListener(e -> viewBillHistory());

makePaymentButton.addActionListener(e -> makePayment());

panel.add(generateBillButton);

panel.add(viewBillHistoryButton);

panel.add(makePaymentButton);

return panel;

}

private JButton createStyledButton(String text) {

JButton button = new JButton(text);

button.setFont(new Font("SansSerif", Font.PLAIN, 15));

button.setBackground(new Color(72, 120, 192));

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createEmptyBorder(10, 15, 10, 15));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR));

button.setPreferredSize(new Dimension(160, 50));

return button;

}

private void addLabelAndField(JPanel panel, String labelText, JTextField textField) {

JLabel label = new JLabel(labelText);

label.setFont(new Font("SansSerif", Font.PLAIN, 15));

label.setForeground(new Color(60, 60, 60));

panel.add(label);

panel.add(textField);

textField.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(190, 210, 240)),

BorderFactory.createEmptyBorder(5, 10, 5, 10)));

}

private class GradientPanel extends JPanel {

private Color color1, color2;

public GradientPanel(Color color1, Color color2) {

this.color1 = color1;

this.color2 = color2;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setPaint(new GradientPaint(0, 0, color1, 0, getHeight(), color2));

g2d.fillRect(0, 0, getWidth(), getHeight());

}

}

private void addPatient() {

String name = nameField.getText();

int age;

try {

age = Integer.parseInt(ageField.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Invalid age value.");

return;

}

String gender = genderField.getText();

String contact = contactField.getText();

String address = addressField.getText();

String symptoms = symptomsField.getText();

double admissionFee;

try {

admissionFee = Double.parseDouble(admissionFeeField.getText());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Invalid admission fee value.");

return;

}

String admissionDate = admissionDateField.getText();

String dischargeDate = dischargeField.getText().isEmpty() ? null : dischargeField.getText();

String sql = "INSERT INTO patients (name, age, gender, contact\_number, address, symptoms, admission\_fee, admission\_date, discharge\_date) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, name);

stmt.setInt(2, age);

stmt.setString(3, gender);

stmt.setString(4, contact);

stmt.setString(5, address);

stmt.setString(6, symptoms);

stmt.setDouble(7, admissionFee);

stmt.setString(8, admissionDate);

stmt.setString(9, dischargeDate);

stmt.executeUpdate();

JOptionPane.showMessageDialog(this, "Patient added successfully!");

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error adding patient: " + e.getMessage());

}

}

private void updatePatient() {

// Ask for the patient name

String patientName = JOptionPane.showInputDialog(this, "Enter patient name to update:");

if (patientName == null || patientName.isEmpty()) {

JOptionPane.showMessageDialog(this, "Patient name is required.");

return;

}

try {

// Check if patient exists in the database

int patientId = getPatientIdByName(patientName);

if (patientId == -1) {

JOptionPane.showMessageDialog(this, "Patient not found: " + patientName);

return;

}

// Ask the user for the field to update

String[] fields = {"Name", "Age", "Gender", "Contact Number", "Address", "Symptoms", "Admission Fee", "Admission Date", "Discharge Date"};

String fieldToUpdate = (String) JOptionPane.showInputDialog(

this,

"Select the field to update:",

"Update Patient",

JOptionPane.QUESTION\_MESSAGE,

null,

fields,

fields[0]

);

if (fieldToUpdate == null || fieldToUpdate.isEmpty()) {

JOptionPane.showMessageDialog(this, "Field selection is required.");

return;

}

// Ask for the new value for the selected field

String newValue = JOptionPane.showInputDialog(this, "Enter new value for " + fieldToUpdate + ":");

if (newValue == null || newValue.isEmpty()) {

JOptionPane.showMessageDialog(this, "New value is required.");

return;

}

// SQL query to update the specific field

String sql = "UPDATE patients SET " + getDatabaseFieldName(fieldToUpdate) + " = ? WHERE patient\_id = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

// Convert and set the appropriate data type for the field

switch (fieldToUpdate) {

case "Age":

stmt.setInt(1, Integer.parseInt(newValue));

break;

case "Admission Fee":

stmt.setDouble(1, Double.parseDouble(newValue));

break;

case "Admission Date":

case "Discharge Date":

stmt.setDate(1, java.sql.Date.valueOf(newValue));

break;

default:

stmt.setString(1, newValue);

}

stmt.setInt(2, patientId);

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

JOptionPane.showMessageDialog(this, "Patient " + fieldToUpdate + " updated successfully!");

} else {

JOptionPane.showMessageDialog(this, "Failed to update patient " + fieldToUpdate + ".");

}

}

} catch (SQLException | NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Error updating patient: " + e.getMessage());

}

}

// Helper method to map GUI fields to database column names

private String getDatabaseFieldName(String fieldName) {

switch (fieldName) {

case "Name": return "name";

case "Age": return "age";

case "Gender": return "gender";

case "Contact Number": return "contact\_number";

case "Address": return "address";

case "Symptoms": return "symptoms";

case "Admission Fee": return "admission\_fee";

case "Admission Date": return "admission\_date";

case "Discharge Date": return "discharge\_date";

default: throw new IllegalArgumentException("Unknown field: " + fieldName);

}

}

private int getPatientIdByName(String name) throws SQLException {

String sql = "SELECT patient\_id FROM patients WHERE name = ?";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {

stmt.setString(1, name);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return rs.getInt("patient\_id");

} else {

return -1;

}

}

}

private void showAllPatients() {

String sql = "SELECT \* FROM patients";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

StringBuilder patientsList = new StringBuilder();

while (rs.next()) {

patientsList.append("ID: ").append(rs.getInt("patient\_id")).append("\n")

.append("Name: ").append(rs.getString("name")).append("\n")

.append("Age: ").append(rs.getInt("age")).append("\n")

.append("Gender: ").append(rs.getString("gender")).append("\n")

.append("Contact: ").append(rs.getString("contact\_number")).append("\n")

.append("Address: ").append(rs.getString("address")).append("\n")

.append("Symptoms: ").append(rs.getString("symptoms")).append("\n")

.append("Admission Fee: ").append(rs.getDouble("admission\_fee")).append("\n")

.append("Admission Date: ").append(rs.getDate("admission\_date")).append("\n")

.append("Discharge Date: ").append(rs.getDate("discharge\_date")).append("\n\n");

}

textArea.setText(patientsList.toString());

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error retrieving patients: " + e.getMessage());

}

}

private void showAllDoctors() {

String sql = "SELECT \* FROM doctors";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

StringBuilder doctorsList = new StringBuilder();

while (rs.next()) {

doctorsList.append("ID: ").append(rs.getInt("doctor\_id")).append("\n")

.append("Name: ").append(rs.getString("name")).append("\n")

.append("Specialization: ").append(rs.getString("specialization")).append("\n")

.append("Contact: ").append(rs.getString("contact\_number")).append("\n")

.append("Availability: ").append(rs.getString("availability")).append("\n\n");

}

textArea.setText(doctorsList.toString());

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error retrieving doctors: " + e.getMessage());

}

}

private boolean isDoctorAvailable(int doctorId, String appointmentDate) throws SQLException {

String sql = "SELECT availability FROM doctors WHERE doctor\_id = ?";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, doctorId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

// Check if doctor is available

String availability = rs.getString("availability");

if (!"Available".equals(availability)) {

return false;

}

// Check if doctor already has appointment at that time

sql = "SELECT COUNT(\*) FROM appointments WHERE doctor\_id = ? AND appointment\_date = ?";

try (PreparedStatement apptStmt = connection.prepareStatement(sql)) {

apptStmt.setInt(1, doctorId);

apptStmt.setString(2, appointmentDate);

ResultSet apptRs = apptStmt.executeQuery();

if (apptRs.next()) {

return apptRs.getInt(1) == 0;

}

}

}

}

return false;

}

// Add this method to check if patient already has a room

private boolean isPatientAssignedRoom(int patientId) throws SQLException {

String sql = "SELECT COUNT(\*) FROM rooms WHERE patient\_id = ?";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return rs.getInt(1) > 0;

}

}

return false;

}

private void bookAppointment() {

String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");

if (patientName == null || patientName.isEmpty()) {

JOptionPane.showMessageDialog(this, "Patient name is required.");

return;

}

try {

int patientId = getPatientIdByName(patientName);

if (patientId == -1) {

JOptionPane.showMessageDialog(this, "Patient not found.");

return;

}

String doctorIdStr = JOptionPane.showInputDialog(this, "Enter doctor ID for appointment:");

if (doctorIdStr == null || doctorIdStr.isEmpty()) {

return;

}

int doctorId = Integer.parseInt(doctorIdStr);

String date = JOptionPane.showInputDialog(this, "Enter appointment date (yyyy-MM-dd):");

if (date == null || date.isEmpty()) {

return;

}

// Check doctor availability

if (!isDoctorAvailable(doctorId, date)) {

JOptionPane.showMessageDialog(this, "Doctor is not available for the selected date.");

return;

}

String time = JOptionPane.showInputDialog(this, "Enter appointment time (HH:mm):");

if (time == null || time.isEmpty()) {

return;

}

String sql = "INSERT INTO appointments (patient\_id, doctor\_id, appointment\_date, appointment\_time, status) VALUES (?, ?, ?, ?, 'Scheduled')";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

pstmt.setInt(2, doctorId);

pstmt.setString(3, date);

pstmt.setString(4, time);

pstmt.executeUpdate();

JOptionPane.showMessageDialog(this, "Appointment booked successfully!");

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error booking appointment: " + e.getMessage());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Invalid doctor ID format.");

}

}

private void showAllAppointments() {

String sql = "SELECT \* FROM appointments";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

StringBuilder appointmentsList = new StringBuilder();

while (rs.next()) {

appointmentsList.append("Appointment ID: ").append(rs.getInt("appointment\_id")).append("\n")

.append("Patient ID: ").append(rs.getInt("patient\_id")).append("\n")

.append("Doctor ID: ").append(rs.getInt("doctor\_id")).append("\n")

.append("Date: ").append(rs.getDate("appointment\_date")).append("\n")

.append("Time: ").append(rs.getTime("appointment\_time")).append("\n")

.append("Status: ").append(rs.getString("status")).append("\n\n");

}

textArea.setText(appointmentsList.toString());

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error retrieving appointments: " + e.getMessage());

}

}

private void assignRoom() {

String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");

if (patientName == null || patientName.isEmpty()) {

JOptionPane.showMessageDialog(this, "Patient name is required.");

return;

}

try {

int patientId = getPatientIdByName(patientName);

if (patientId == -1) {

JOptionPane.showMessageDialog(this, "Patient not found.");

return;

}

// Check if patient already has a room

if (isPatientAssignedRoom(patientId)) {

JOptionPane.showMessageDialog(this, "Patient is already assigned to a room.");

return;

}

String roomNumber = JOptionPane.showInputDialog(this, "Enter room number:");

if (roomNumber == null || roomNumber.isEmpty()) {

JOptionPane.showMessageDialog(this, "Room number is required.");

return;

}

if (isRoomAvailable(roomNumber)) {

String sql = "INSERT INTO rooms (patient\_id, room\_number) VALUES (?, ?)";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

pstmt.setString(2, roomNumber);

pstmt.executeUpdate();

JOptionPane.showMessageDialog(this, "Room assigned successfully!");

}

} else {

JOptionPane.showMessageDialog(this, "Room is already occupied.");

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error assigning room: " + e.getMessage());

}

}

private void showAllRooms() {

String sql = "SELECT p.name, r.room\_number FROM patients p JOIN rooms r ON p.patient\_id = r.patient\_id";

try (Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

StringBuilder roomsList = new StringBuilder();

while (rs.next()) {

roomsList.append("Patient Name: ").append(rs.getString("name")).append("\n")

.append("Room Number: ").append(rs.getString("room\_number")).append("\n\n");

}

textArea.setText(roomsList.toString());

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error retrieving rooms: " + e.getMessage());

}

}

private boolean isRoomAvailable(String roomNumber) throws SQLException {

String sql = "SELECT COUNT(\*) FROM rooms WHERE room\_number = ?";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setString(1, roomNumber);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return rs.getInt(1) == 0;

}

}

return false;

}

private void generateBill() {

String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");

if (patientName == null || patientName.isEmpty()) {

return;

}

try {

int patientId = getPatientIdByName(patientName);

if (patientId == -1) {

JOptionPane.showMessageDialog(this, "Patient not found.");

return;

}

// Get room charges and doctor fees for reference

double roomCharges = calculateRoomCharges(patientId);

double doctorFees = calculateDoctorFees(patientId);

// Show the calculated charges as reference

JOptionPane.showMessageDialog(this,

"Calculated Charges (For Reference):\n" +

"Room Charges: $" + roomCharges + "\n" +

"Doctor Fees: $" + doctorFees + "\n" +

"Total Calculated: $" + (roomCharges + doctorFees));

// Ask for manual entry of charges

String roomChargesStr = JOptionPane.showInputDialog(this, "Enter room charges amount:");

String doctorFeesStr = JOptionPane.showInputDialog(this, "Enter doctor fees amount:");

if (roomChargesStr == null || doctorFeesStr == null) {

return;

}

double manualRoomCharges = Double.parseDouble(roomChargesStr);

double manualDoctorFees = Double.parseDouble(doctorFeesStr);

double totalAmount = manualRoomCharges + manualDoctorFees;

// Create bill in database

String sql = "INSERT INTO billing (patient\_id, total\_amount, paid\_amount, balance\_amount, bill\_date, payment\_status) VALUES (?, ?, 0, ?, CURDATE(), 'PENDING')";

try (PreparedStatement pstmt = connection.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS)) {

pstmt.setInt(1, patientId);

pstmt.setDouble(2, totalAmount);

pstmt.setDouble(3, totalAmount);

pstmt.executeUpdate();

// Get the generated bill\_id

ResultSet rs = pstmt.getGeneratedKeys();

if (rs.next()) {

int billId = rs.getInt(1);

// Add billing items with manual amounts

addBillingItem(billId, "Room Charges", manualRoomCharges);

addBillingItem(billId, "Doctor Fees", manualDoctorFees);

}

}

JOptionPane.showMessageDialog(this, "Bill generated successfully!\nTotal Amount: $" + totalAmount);

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error generating bill: " + e.getMessage());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Invalid amount format. Please enter valid numbers.");

}

}

private void addBillingItem(int billId, String description, double amount) throws SQLException {

String sql = "INSERT INTO billing\_items (bill\_id, item\_type, description, quantity, unit\_price, total\_price, date\_added) VALUES (?, ?, ?, 1, ?, ?, CURDATE())";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, billId);

pstmt.setString(2, description);

pstmt.setString(3, description);

pstmt.setDouble(4, amount);

pstmt.setDouble(5, amount);

pstmt.executeUpdate();

}

}

private double calculateRoomCharges(int patientId) throws SQLException {

// Get room details and calculate charges

String sql = "SELECT DATEDIFF(COALESCE(discharge\_date, CURDATE()), admission\_date) as days FROM patients WHERE patient\_id = ?";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

int days = rs.getInt("days");

// Assuming room rate is $100 per day

return days \* 100.0;

}

}

return 0.0;

}

private double calculateDoctorFees(int patientId) throws SQLException {

// Calculate total doctor fees from appointments

String sql = "SELECT COUNT(\*) as visits FROM appointments WHERE patient\_id = ?";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

int visits = rs.getInt("visits");

// Assuming doctor fee is $50 per visit

return visits \* 50.0;

}

}

return 0.0;

}

private void viewBillHistory() {

String patientName = JOptionPane.showInputDialog(this, "Enter patient name:");

if (patientName == null || patientName.isEmpty()) {

return;

}

try {

int patientId = getPatientIdByName(patientName);

if (patientId == -1) {

JOptionPane.showMessageDialog(this, "Patient not found.");

return;

}

String sql = "SELECT b.\*, p.name FROM billing b JOIN patients p ON b.patient\_id = p.patient\_id WHERE b.patient\_id = ?";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, patientId);

ResultSet rs = pstmt.executeQuery();

StringBuilder billHistory = new StringBuilder();

billHistory.append("Bill History for Patient: ").append(patientName).append("\n\n");

while (rs.next()) {

billHistory.append("Bill ID: ").append(rs.getInt("bill\_id")).append("\n")

.append("Date: ").append(rs.getDate("bill\_date")).append("\n")

.append("Total Amount: $").append(rs.getDouble("total\_amount")).append("\n")

.append("Paid Amount: $").append(rs.getDouble("paid\_amount")).append("\n")

.append("Balance: $").append(rs.getDouble("balance\_amount")).append("\n")

.append("Status: ").append(rs.getString("payment\_status")).append("\n\n");

}

textArea.setText(billHistory.toString());

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error retrieving bill history: " + e.getMessage());

}

}

private void makePayment() {

String billIdStr = JOptionPane.showInputDialog(this, "Enter Bill ID:");

if (billIdStr == null || billIdStr.isEmpty()) {

return;

}

try {

int billId = Integer.parseInt(billIdStr);

// First get the current bill details

String checkSql = "SELECT total\_amount, paid\_amount, balance\_amount FROM billing WHERE bill\_id = ?";

double totalAmount = 0;

double paidAmount = 0;

double balanceAmount = 0;

try (PreparedStatement checkStmt = connection.prepareStatement(checkSql)) {

checkStmt.setInt(1, billId);

ResultSet rs = checkStmt.executeQuery();

if (rs.next()) {

totalAmount = rs.getDouble("total\_amount");

paidAmount = rs.getDouble("paid\_amount");

balanceAmount = rs.getDouble("balance\_amount");

} else {

JOptionPane.showMessageDialog(this, "Bill not found.");

return;

}

}

String amountStr = JOptionPane.showInputDialog(this, "Enter payment amount:");

if (amountStr == null || amountStr.isEmpty()) {

return;

}

double amount = Double.parseDouble(amountStr);

String[] paymentMethods = {"Cash", "Credit Card", "Debit Card", "Insurance"};

String paymentMethod = (String) JOptionPane.showInputDialog(

this,

"Select payment method:",

"Payment Method",

JOptionPane.QUESTION\_MESSAGE,

null,

paymentMethods,

paymentMethods[0]

);

if (paymentMethod != null) {

// Update the paid amount and balance amount

double newPaidAmount = paidAmount + amount;

double newBalanceAmount = totalAmount - newPaidAmount;

// Update billing table

String sql = "UPDATE billing SET paid\_amount = ?, balance\_amount = ?, payment\_status = CASE WHEN (paid\_amount + ?) >= total\_amount THEN 'PAID' ELSE 'PARTIAL' END WHERE bill\_id = ?";

try (PreparedStatement updateStmt = connection.prepareStatement(sql)) {

updateStmt.setDouble(1, newPaidAmount);

updateStmt.setDouble(2, newBalanceAmount);

updateStmt.setDouble(3, amount);

updateStmt.setInt(4, billId);

updateStmt.executeUpdate();

}

// Record payment

sql = "INSERT INTO payments (bill\_id, amount\_paid, payment\_date, payment\_method, payment\_status) VALUES (?, ?, CURDATE(), ?, 'COMPLETED')";

try (PreparedStatement pstmt = connection.prepareStatement(sql)) {

pstmt.setInt(1, billId);

pstmt.setDouble(2, amount);

pstmt.setString(3, paymentMethod);

pstmt.executeUpdate();

}

JOptionPane.showMessageDialog(this, "Payment processed successfully!");

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(this, "Error processing payment: " + e.getMessage());

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Invalid number format.");

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

new HospitalManagementSystemGUI().setVisible(true);

});

}

}

1. **Database Connectivity**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

private static final String URL = "jdbc:mysql://localhost:3306/hospital\_db"; // Correct DB URL

private static final String USERNAME = "root"; // Correct DB username

private static final String PASSWORD = "GobLiNOvO#03"; // Ensure the password is correct

public static Connection getConnection() throws SQLException {

try {

// Load and register the MySQL JDBC driver

Class.forName("com.mysql.cj.jdbc.Driver");

// Return the database connection

return DriverManager.getConnection(URL, USERNAME, PASSWORD);

} catch (ClassNotFoundException e) {

// Handle error where the JDBC driver is not found

throw new SQLException("MySQL JDBC Driver not found. Include the JDBC library in your project.");

} catch (SQLException e) {

// Handle general SQL exceptions

throw new SQLException("Error connecting to the database: " + e.getMessage());

}

}

}

## CHAPTER 7

## CONCLUSION AND FUTURE WORKS

**Conclusion:**

The Hospital Management System (HMS) offers a comprehensive solution to streamline hospital operations, enhance patient care, and improve administrative efficiency. By integrating essential features such as patient management, appointment scheduling, billing, and inventory tracking, the system optimizes hospital workflows and minimizes errors, leading to faster and more reliable service. Built with modern technologies, the HMS ensures scalability, security, and user-friendliness, making it an ideal tool for both hospital staff and patients. The centralized database and modular design facilitate better data management and compliance with healthcare regulations.

**Future Uses:**

Looking ahead, several enhancements can further elevate the HMS. Integrating telemedicine features will enable remote consultations and monitoring, expanding access to healthcare. Developing a mobile app will improve patient and doctor engagement, offering easy access to appointments and medical records. The incorporation of AI for predictive analytics can optimize hospital resource management and health outcomes. Additionally, future versions could integrate with Electronic Health Records (EHR) systems for seamless data exchange and improve continuity of care. Strengthening security features, such as biometric authentication, will ensure the privacy and safety of sensitive patient data.

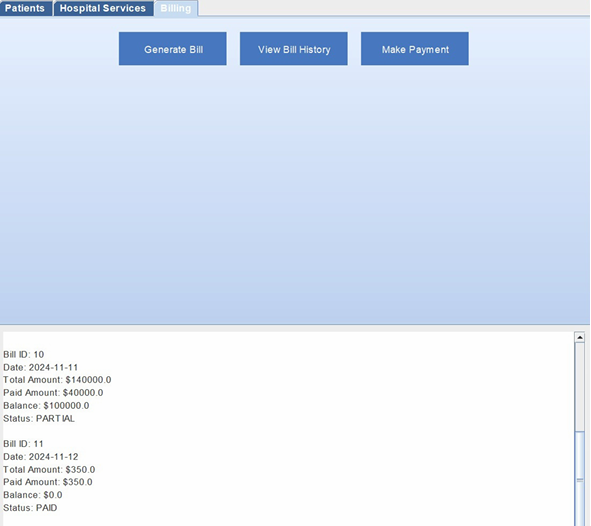
## CHAPTER 8

## RESULT AND DISCUSSION

## 

## 





**CHAPTER 9**

**REFRENCES:**

1. https://docs.oracle.com/javase/tutorial/jdbc/
2. <https://dev.mysql.com/doc/connector-j/en/>
3. https://leadsquared.com/industries/healthcare/hospital-management system-hms/