# HOSPITAL MANAGEMENT SYSTEM

A PROJECT REPORT

*Submitted by*

DINESHWAR S 231001041

ABIJEETH J 231001004

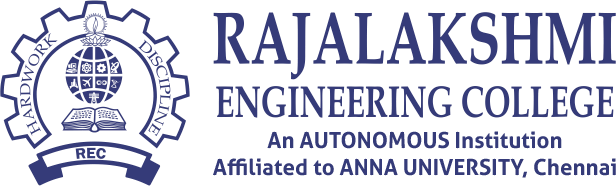
*In partial fulfilment for the award of the degree Of*

# BACHELOR OF TECHNOLOG



*IInn*

**INFORMATION TECHNOLOGY**



# DEPARTMENT OF INFORMATION TECHNOLOGY RAJALAKSHMI ENGINEERING COLLEGE

**November 2024**

**BONAFIDE CERTIFICATE**

Certified that this project titled “**Hospital Management System**” is the bonafide work of **Abijeeth .G(2310010004),Dineshwar.s(231001041)** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Dr. P. Valarmathie Mrs. Usha S**

**HEAD OF THE DEPARTMENT COURSE INCHARGE**

Department of Information Technology Department of Information Technology Rajalakshmi Engineering College, Rajalakshmi Engineering College

This project is submitted for CS23333 –Object oriented programming using Java held on

**INTERNAL EXAMINAR EXTERNAL EXAMINAR**

**TABLE CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| **1** | **QUIZ MANAGEMENT SYSTEM** | 4 |
|  | 1.1 ABSTRACT | 4 |
|  | 1.2 INTRODUCTION | 4 |
|  | 1.3 PURPOSE | 4 |
|  | 1.4 SCOPE OF PROJECT | 4 |
|  | 1.5 SOFTWARE REQUIREMENTS | 5 |
| **2** | **SYSTEM FLOW DIAGRAM** | 10 |
|  | 2.1 USE CASE DIAGRAM | 10 |
|  | 2.2 ENTITY DIAGRAM | 10 |
|  | 2.3 DATA FLOW DIAGRAM | 11 |
| **3** | **MODULE DESCRIPTION** | 11 |
| **4** | **IMPLEMENTATION** | 12 |
|  | 4.1 DESIGN | 12 |
|  | 4.2 DATA BASE DESIGN | 18 |
|  | 4.3 CODE | 19 |
| **5** | **CONCLUSION** | 47 |
| **6** | **REFERENCES** | 47 |

# Abstract

The primary objective of the JDBC-powered Hospital Management System in Java isto provide a reliable and efficient platform for patien to seamlessly browse, select seats, and reserve movie tickets, all while leveraging the power of Java's database connectivity capabilitiesand automate various administrative, financial, and clinical functions within a healthcare facility. It helps healthcare providers manage critical tasks such as patient registration, appointment scheduling, medical record keeping, billing, pharmacy management, and inventory control, all within a centralized system. By offering real-time data access and enhanced collaboration across departments, HMS improves operational efficiency, reduces human errors, and ensures better patient care. Additionally, it supports features such as secure data management, billing and insurance integration, and compliance with healthcare regulations (e.g., HIPAA), making it a vital tool for healthcare organizations to optimize their resources and enhance the quality of services provided to patients. hospital functions, including patient registration, appointment scheduling, medical record management, billing, pharmacy management, and inventory control, into a unified platform. The system improves operational efficiency, reduces manual errors, and enhances the quality of patient care by providing real-time access to accurate and updated information. By enabling secure data storage, role-based access control, and seamless communication between departments, an HMS ensures better coordination, improved decision-making, and enhanced patient safety. Additionally, it supports regulatory compliance, backup and recovery, and detailed reporting, making it an indispensable tool for modern healthcare institutions aiming to deliver exceptional medical services while optimizing resource utilization. It integrates various functions such as patient registration, appointment scheduling, billing, inventory

# Introduction

Introduce the concept of a hospital management system designed toautomate and

manage the day-to-day administrative and medical tasks in hospitals. Highlight the

management, pharmacy management, and medical record maintenance into a unified platform. The system ensures seamless communication between departments, facilitates real-time access to patient data, and improves overall workflow within the hospital. By automating administrative tasks, managing resources effectively, and ensuring compliance with healthcare regulations, the HMS reduces operational costs, minimizes human errors, and enhances the quality of care provided to patients. With robust security features and data integrity controls, the system ensures patient confidentiality and safeguards sensitive medical information, making it an indispensable tool for modern healthcare facilities

a comprehensive software solution designed to streamline and automate the administrative, financial, and clinical operations of a healthcare facility. The system integrates various modules, including patient registration, appointment scheduling, medical records management, billing, pharmacy, inventory control, and reporting, to provide seamless coordination across hospital departments. By centralizing critical data and workflows, HMS enhances the efficiency of hospital operations, improves patient care, and ensures data accuracy. The system enables healthcare providers to access real-time information, track patient histories, manage resources effectively, and ensure compliance with healthcare regulations. Ultimately, a well-implemented Hospital Management System leads to better management of hospital resources, optimized workflows, reduced operational costs, and enhanced patient satisfaction, contributing to improved healthcare delivery. offsite backups with granular recovery options and thorough disaster recovery planning, hospitals can safeguard patient data and maintain business continuity even in challenging situations.

purpose

A **Hospital Management System (HMS)** is a comprehensive software solution designed to manage and streamline various hospital operations. When developed using **Java**, the system is designed to offer flexibility, scalability, and robustness. Here are the primary purposes and features of a Hospital Management System using Java:

**Hospital Operations**

* Automate and integrate key functions like patient registration, appointment scheduling, billing, and inventory management to improve hospital efficiency and reduce manual work.

**Enhance Patient Care**

* Provide doctors, nurses, and healthcare professionals with real-time access to patient medical records, treatment plans, and test results, ensuring timely and accurate care.

**Improve Data Accuracy and Accessibility**

* Centralize medical records, billing information, and other critical data to reduce errors and provide easy access to information for staff, improving decision-making.

**Efficient Appointment and Resource Management**

* Manage patient appointments, hospital bed availability, and resource allocation (e.g., operating rooms, medical equipment) to optimize hospital operations and reduce wait times.

medical equipment) to optimize hospital operations and reduce wait times.

**Decision Making with Analytics and Reporting**

* Provide comprehensive reports and analytics on hospital performance, patient outcomes, resource utilization, and financial health to assist in strategic decision-making.

# Scope of the Project

The **Hospital Management System (HMS)** is a comprehensive software application that assists healthcare providers in managing various functions of a hospital or healthcare facility. The system streamlines hospital operations, enhances patient care, and reduces administrative costs by automating essential tasks. Implementing this system using **Java** allows for scalability, security, and platform independence..

# Software Requirement Specification Introduction

The **Software Requirements Specification (SRS)** provides a detailed description of the system's functionality, design, and implementation details. For the **Hospital Management System (HMS)**, the SRS defines the requirements that must be met to ensure that the system functions as expected.

# Document Purpose

The purpose of this document is to specify the functional and non-functional requirements of the **Hospital Management System (HMS)**. This system will be designed to automate and manage various hospital operations, including patient management, staff management, billing, inventory, and more, to improve efficiency and reduce errors.

# Product Scope

The system will support the management of hospital operations, such as\

* Patient Registration and Appointment Scheduling
* Patient Medical History Management



* Staff Management (Doctors, Nurses, Administrative Staff)



* Billing and Payments



* Pharmacy and Inventory Management



* Laboratory Management



* Reporting and Analytics



* Role-based Access Control for various user types

REFERENCES & ACKNOWLEDGEMENT:

[**https://www.javatpoint.com/javaawt**](https://www.javatpoint.com/javaawt%20)

[**https://www.javatpoint.com/java**](https://www.javatpoint.com/java%20) [**swing**](https://www.javatpoint.com/java)

# Overall Description

The system will be implemented in **Java**, using a **relational database (e.g., MySQL orPostgreSQL)** for storage.

Definitions, Acronyms, and Abbreviations:

**HMS**: Hospital Management System

**UI**: User Interface

* **CRUD**: Create, Read, Update, Delete
* **DBMS**: Database Management System
* **API**: Application Programming Interface
* **User Roles**: Admin, Doctor, Nurse, Patient, Receptionist

# User Interface Components

:

* + Main Frame
  + Menu Bar
  + Login Screen
  + Patient Management Section
  + Appointment Scheduling Section
  + Patient Details
  + Logout and Exit Button
  + Validation and Action Listeners
  + Useful in reality

# Admin Interface Components

The **Admin Interface** in a **Hospital Management System** is crucial because itallows the administrative users (like hospital administrators) to manage key aspects of the system, including managingdoctors, patients, appointments, hospital departments, and more. Below is a comprehensive set of UI components that you can include for the **Admin Interface** using Java Swing:

* **Manage Doctors**: Add, update, delete, and view doctors.
* **Manage Patients**: Add, update, delete, and view patient details.
* **Manage Appointments**: Schedule, cancel, and view appointments.
  + **Generate Reports**: View reports for appointments, patient history, etc.
  + **Navigation**: A menu bar for easy navigation to different sections of the hospital system.
  + **Logout/Exit**: Allows the admin to log out or exit the system.

# Product Functionality

* Admin Login: Enables existing administrators to log in securely.
* User signup: Facilitates the addition of new users.
* user: Allows users to attempt the quiz in two difficulty levels
* Admin interface: displays the phone number, name and the difficulty level

# User and Characteristics

* **Admin**: Manages the overall system, including user roles, reports, and hospital operations.
* **Doctor**: Handles patient care, consultations, diagnoses, prescriptions, and appointments
* **Nurse**: Assists in patient care and administers medications as prescribed by doctors.
* **Patient**: Registers for appointments, views medical records, and makes payments

.

* **Receptionist**: Manages patient check-ins, schedules appointments, and handles billing.

# Operating Environment:

* **Backend**: Java (JDK 11 or higher)
* **Database**: MySQL or PostgreSQL
* **Frontend**: JavaFX or Swing (GUI) for desktop or web interface using Java Servlet/JSP
* **Server**: Tomcat or Jetty (for web deployment, if applicable

## Constraints

* The system should be accessible via the internet if deployed as a web application, or locally ifdeveloped as a desktop application.
* The database must be regularly backed up to prevent data loss.
* The system must be able to handle a high volume of users simultaneously without performancedegradation.
* Security requirements include encrypted storage for sensitive patient data and role-based accesscontrol.
* The database must be secured

**Data Integrity Constraints**

**Unique Patient ID**: Each patient must have a unique identifier.

* **Unique Doctor ID**: Each healthcare provider (doctor, nurse, etc.) must have a unique ID.
* **Valid Data Formats**: Fields such as phone numbers, email addresses, and dates of birth must conform to correct formats.
* **Required Fields**: Certain fields (e.g., patient name, date of birth, etc.) must always be filled.
* **Foreign Key Constraints**: For relational databases, foreign key relationships (e.g., between patients and appointments or doctors and departments) must be maintained to ensure consistency.

**Operational Constraints**

* **Shift Hours**: Staff (e.g., nurses, doctors) must adhere to shift schedules and cannot exceed maximum working hours as per labor laws.
* **Room Availability**: Rooms or hospital facilities (e.g., operation theaters, ICU) must not be overbooked.
* **Supply Availability**: Certain medical supplies (e.g., medications, surgical tools) must be in stock for specific procedures or treatments.

# User Interface

* + the **Look and Feel**: Use UIManager.setLookAndFeel(UIManager.getSys temLookAndFeelClassName()) to ensure the application has the native look and feel of the OS.
  + **Swing Layout Managers**: Use layout managers like BorderLayout, GridBagLayout, FlowLayout,and BoxLayout to manage the layout of components in your panels.
  + **Data Binding**: If you're connecting to a database (e.g., MySQL), you can use JDBC (Java DatabaseConnectivity) to display and manage the data.

These are some of the key UI components you can include in a Hospital Management System using JavaSwing. Depending on the functionality you need, you can add or modify the components.

# User and Characteristics

* **Admin**: Manages the overall system, including user roles, reports, and hospital operations.
* **Doctor**: Handles patient care, consultations, diagnoses, prescriptions, and appointments
* **Nurse**: Assists in patient care and administers medications as prescribed by doctors.
* **Patient**: Registers for appointments, views medical records, and makes payments

.

* **Receptionist**: Manages patient check-ins, schedules appointments, and handles billing.

.

# Operating Environment:

* **Backend**: Java (JDK 11 or higher)
* **Database**: MySQL or PostgreSQL
* **Frontend**: JavaFX or Swing (GUI) for desktop or web interface using Java Servlet/JSP
* **Server**: Tomcat or Jetty (for web deployment, if applicable

## Constraints

* The system should be accessible via the internet if deployed as a web application, or locally ifdeveloped as a desktop application.
* The database must be regularly backed up to prevent data loss.
* The system must be able to handle a high volume of users simultaneously without performancedegradation
* Security requirements include encrypted storage for sensitive patient data and role-based accesscontrol.

MySQL), you can use JDBC (Java DatabaseConnectivity) to display and manage the data.

These are some of the key UI components you can include in a Hospital Management System using JavaSwing. Depending on the functionality you need, you can add or modify the components.

# Functional Requirements

## Patient Management

**FR1.1**: The system must allow for the registration of new patients, including personal and medical information.

**FR1.2**: The system must allow patients to schedule, reschedule, and cancel appointments.

**FR1.3**: The system should store a patient's medical history, including diagnoses, treatment plans, prescriptions,and test results.

**FR1.4**: The system must allow patients to view their medical records and appointment history

**FR2.1**: The system must allow for the registration of doctors, nurses, and other hospital staff, with detailedprofile management.

* + **FR2.2**: The system must allow doctors to set their availability and manage their schedule.
  + **FR2.3**: The system should allow administrative users to generate reports about staff performance andattendance.
  + **FR2.4**: The system must allow salary and compensation management
  + **FR2.**5: Allow administrators to schedule shifts and appointments for doctors and other healthcare providers, ensuring there are no scheduling conflicts.
  + **FR2.6**:Manage and track staff roles, responsibilities, qualifications, and shift assignments for doctors, nurses, technicians, and administrative personnel.
  + **FR2.7**:Define user roles (admin, doctor, nurse, receptionist, etc.) and assign appropriate access rights based on their roles.

## Appointment Scheduling

* + **FR3.1**: The system should allow patients to book appointments with doctors, including time slots based on doctor availability.
  + **FR3.2**: The system should provide automatic reminders to both doctors and patients about upcoming appointments.

## Billing and Payment Management

* + **FR4.1**: The system must allow for the generation of patient invoices for consultations, tests, and treatments.
  + **FR4.2**: The system should allow patients to make payments and track payment history.
  + **FR4.3**: The system must support integration with insurance providers for claim management



* + **FR4.4**: The system must allow for generating financial reports for hospital administrators.

## Inventory and Pharmacy Management

* + **FR5.1**: The system must track medical supplies and medications available in the hospital.
  + **FR5.2**: The system should generate alerts when stock levels are low or when medications are nearing expiration.
  + **FR5.3**: The system should allow for the generation of prescriptions for patients, which can be fulfilled by the pharmacy

**FR5.3**: Monitor stock levels for medicines, medical supplies, and equipment, ensuring timely restocking and avoiding shortages.

**Integration**: Integrate the pharmacy module with the doctor’s prescription system so that medications prescribed are automatically available in the pharmacy for dispensing.

Notify administrators and pharmacy staff when stock levels of essential items are low or near expiry.

Enable doctors to generate and track prescriptions for patients.

**FR5.**4 Track Medical Supplies and Equipment

Maintain real-time tracking of medical supplies (e.g., gloves)

Ensure stock levels are sufficient to meet the hospital’s needs,

Stock Level Monitoring.

## Laboratory Management

* + **FR6.1**: The system must allow doctors to order lab tests for patients and record the results.



* + **FR6.2**: The system should allow laboratory staff to update test results and notify doctors and patients whenresults are available.

## Reporting and Analytics

* + **FR7.1**: The system must generate various reports for hospital management, including:



* Patient visit reports
* Billing and payment reports
* Financial and revenue reports
* Staff performance reports

o

* + **FR7.2**: The system must allow for exporting reports in various formats, such as PDF, Excel, and CSV.

## Security and Access Control

* + **FR8.1**: The system must support user authentication (login) and role-based access control.
  + **FR8.2**: Sensitive patient data must be encrypted, both during storage and transmission.
  + **FR8.3**: Only authorized users (admin, doctors) should be able to access or modify sensitive data.

* **FR8.4**: Define roles like **Doctor**, **Nurse**, **Pharmacist**, **Administrator**, **Receptionist**, **IT Staff**, etc.

**Permissions**: Set permissions based on roles

**Doctors** can access patient medical records, prescribe medications, and view test results.

**Nurses** may have access to patient care records but cannot prescribe medications.

**Administrators** may have unrestricted access to all parts of the system for management purposes.

**Receptionists** may access patient appointment data but not sensitive medical records.

**Granular Access**: Define specific permissions for actions like **viewing**, **editing**, **adding**, or **deleting** data.

**FR8.5**:hospitals can ensure that their data remains secure, compliant with regulations, and only accessible to authorized personnel. These measures help prevent unauthorized access, protect patient confidentiality, and reduce the risks associated with cyber threats and data breaches.

.

# Non-functional Requirements

## Performance

* + **NFR1**: The system should be able to handle a high number of concurrent users without performance degradation, ensuring smooth operations at peak times.

## Scalability

* + **NFR2**: The system must be scalable to accommodate the addition of new hospitals, departments, and users.

## Usability

* + **NFR3**: The system must provide a user-friendly interface, with an intuitive design that minimizes the learningcurve for hospital staff.

## Security

* + **NFR4**: The system must implement robust data security measures, such as encryption and secure login protocols, to safeguard sensitive patient data.

## Availability

* + **NFR5**: The system must ensure high availability, with minimal downtime for maintenance and updates.

## Backup and Recovery

* + **NFR6**: The system must implement regular backups and a disaster recovery plan to prevent data loss in case of asystem failure.
  + **NFR7**:Ensure that backups are performed on a regular, automated schedule (e.g., daily, weekly) to prevent data loss.

Automate the process to avoid human error, and ensure that critical data (patient records, financial data, inventory) is backed up consistently and timely.

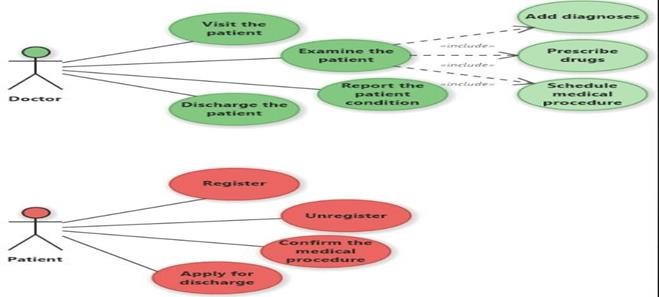
** Periodically verify the integrity and completeness of backup files to ensure that they can be successfully restored when needed.**

**NFR8**: ITshould be encrypted to protect sensitive hospital data (e.g., patient health records, medical histories) during storage and transfer.

Ensure that all backup data, whether stored on local servers or in the cloud, is encrypted using industry-standard encryption protocols (e.g., AES-256) to prevent unauthorized access.

**NFR9:** verify the integrity and completeness of backup files to ensure that they can be successfully restored when needed.

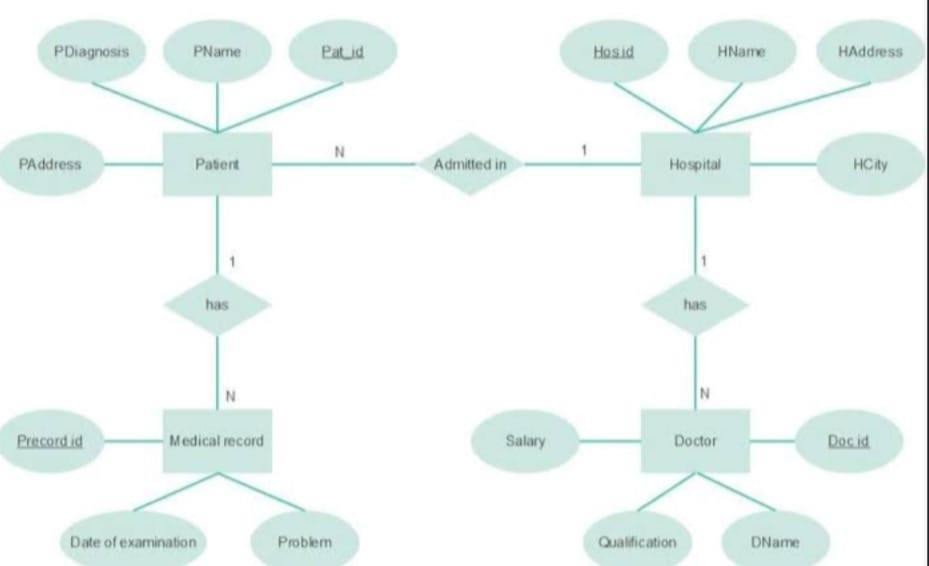
Implement automated integrity checks on backups to detect any corrupt or incomplete data, and conduct periodic restore drills to verify the process.



**Figure 2.1.1 Use Case Diagrams**

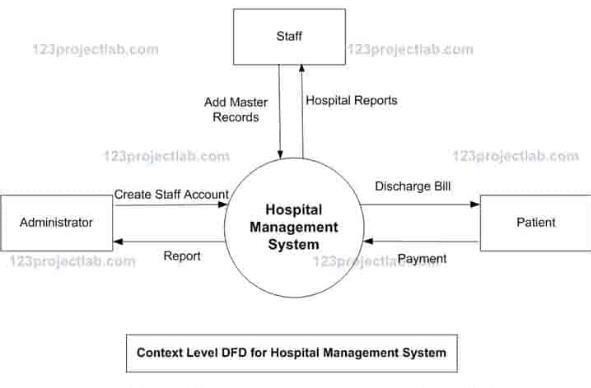
**Entity relationship diagram**

ER (Entity Relationship) Diagram is used to represents the relationship between entities in the table.



**Figure 2.2.1 Entity relationship diagram**

**Data flow diagram**



**Figure 2.3.1 Data flow diagram**

**System Architecture**

The system will follow a **3-tier architecture**:

1. **Presentation Layer** (UI) - Built using JavaFX or Swing.
2. **Business Logic Layer** - Implemented using Java classes to handle core hospital operations (appointments, billing, etc.).
3. **Data Access Layer** - Utilizing **JDBC** for database interactions and **Hibernate ORM** for object-relational mapping (if needed).

**Appendices**

* + **A.1**: List of supported browsers and operating systems (for web-based systems).
  + **A.2**: Sample database schema (tables for patients, doctors, appointments, etc.).
  + **A.3**: User interface wireframes or mockups.

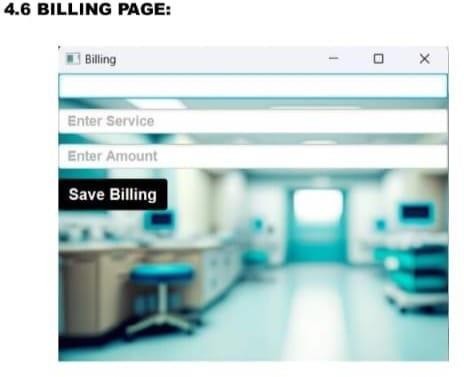


**User login**



**Figure 4.2 User login**

**Billing Page:**

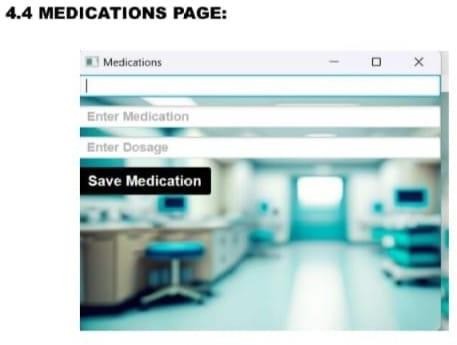
**Figure 4.3 Billing Page**

**Service Page:**

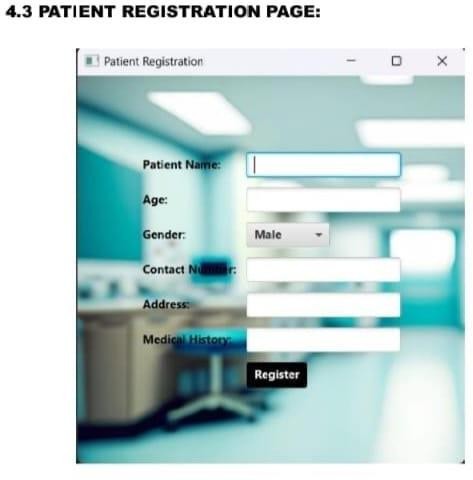


**Figure 4.4 Service Page**

**Medications Page:**

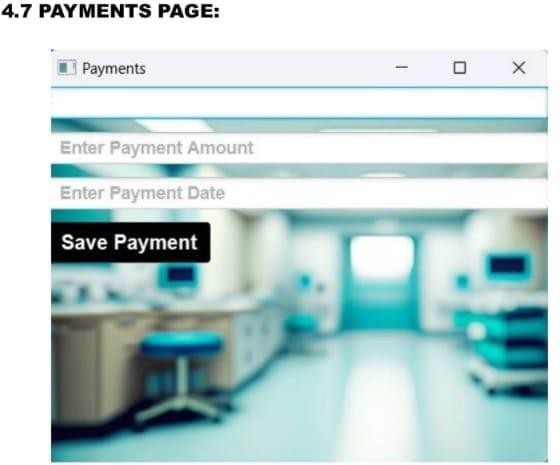


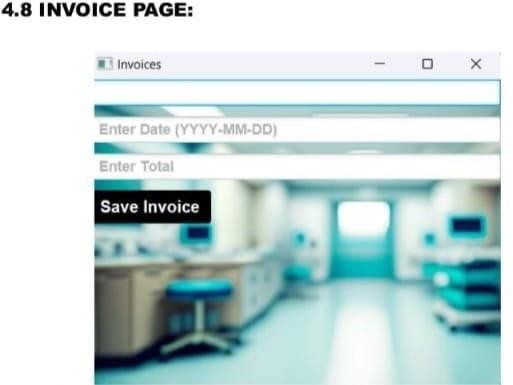
**Figure 4.5 Medication Page**

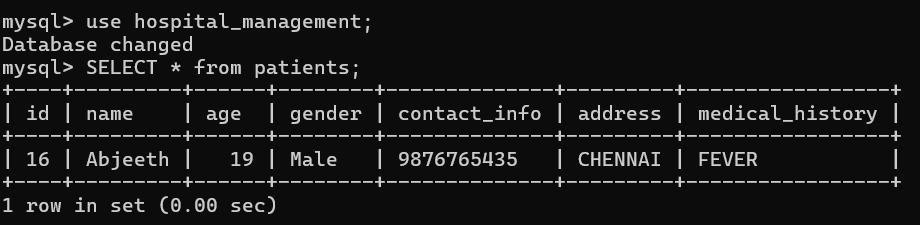


**Figure 4.6 Patients Registration Page**

**Payement Page:**





**Figure 4.8 Invoice Page**

**Figure 4.9 Output**

# Database Design:

Designing the database for a **Hospital Management System** (HMS) is a critical step in ensuring the efficient storage and retrieval of data related to patients, doctors, appointments, staff, and hospital services. Below is a conceptual **Database Design** for an HMS, including tables, their relationships, and the relevant fields

# Database Tables and Relationship:

The main entities in a Hospital Management System are typically **Doctors**, **Patients**, **Appointments**, **Staff**, **Departments**, and **Billing**. Here's a suggested database schema with key tables and relationships:

* Patients tables
* Doctor Table
* Department Table
* Appointment Table
* Medical Record Table
* Building Table
* Staff Table
* Hospital Service Table
* Prescription Table

# Relationships Between Tables:

* **Patients** can have many **Appointments**.
* Each **Appointment** is linked to a **Doctor** and a **Patient**.
* **Doctors** belong to a **Department** and can have multiple **Medical Records** linked tothem.
* **Patients** can have multiple **Medical Records**.
* **Staff** is assigned to a **Department** and has a role (e.g., nurse, admin).
* **Billing** is linked to **Patients** and stores the total amount and payment

**Medical Records** are tied to a **Patient** and **Doctor**, containing diagnosis, treatments, andprescriptions.

* + **Prescriptions** are tied to **Medical Records** and contain the prescribed medicines.
  +  **Many-to-One Relationship**  
    A **many-to-one** relationship is the inverse of a one-to-many relationship. For example, many **Appointments** can belong to one **Doctor**.
  + **Example:**
  + **Appointment Table** (AppointmentID - Primary Key, DoctorID - Foreign Key)
  + **Doctor Table** (DoctorID - Primary Key)
  + **Relationship**: Multiple appointments can be linked to a single doctor, but each appointment is associated with only one doctor.
  +  **One-to-One Relationship**  
    A **one-to-one** relationship occurs when a record in one table is linked to only one record in another table. For example, each **Patient** might have one **Medical ReRelationship**: Each patient has exactly one medical record, and each medical record is linked to only one patient.

# MAIN APPLICATION :

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.\*;

import javafx.scene.paint.Color;

import javafx.scene.paint.CycleMethod;

import javafx.scene.paint.LinearGradient;

import javafx.scene.paint.Stop;

import javafx.stage.Stage;

public class MainApp extends Application {

public Stage primaryStage;

@Override

public void start(Stage primaryStage) {

this.primaryStage = primaryStage;

showLoginPage(primaryStage);

}

private void showLoginPage(Stage primaryStage) {

primaryStage.setTitle("Login Page");

Label userLabel = new Label("Username:");

TextField userField = new TextField();

userField.setPromptText("Enter your username");

Label passLabel = new Label("Password:");

PasswordField passField = new PasswordField();

passField.setPromptText("Enter your password");

Button loginButton = new Button("Login");

Label messageLabel = new Label("");

userLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

passLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

[7:26 pm, 19/11/2024] abijeeth: loginButton.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

white; -fx-background-color: black;");

userField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

passField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

loginButton.setOnAction(e -> {

String username = userField.getText();

String password = passField.getText();

if (authenticateUser(username, password)) {

messageLabel.setText("Login Successful!");

messageLabel.setTextFill(Color.GREEN);

showMainWindow(primaryStage);

} else {

messageLabel.setText("Invalid username or password.");

messageLabel.setTextFill(Color.RED);

}

});

VBox vbox = new VBox(10, userLabel, userField, passLabel, passField, loginButton,

messageLabel);

vbox.setAlignment(Pos.CENTER);

vbox.setPadding(new Insets(20));

Stop[] stops = new Stop[] {

new Stop(0, Color.CORNFLOWERBLUE),

new Stop(1, Color.DARKSLATEBLUE)

};

LinearGradient gradient = new LinearGradient(0, 0, 1, 1, true,

CycleMethod.NO\_CYCLE, stops);

BorderPane root = new BorderPane();

root.setCenter(vbox);

root.setBackground(new javafx.scene.layout.Background(new

javafx.scene.layout.BackgroundFill(gradient, javafx.scene.layout.CornerRadii.EMPTY,

Insets.EMPTY)));

root.setStyle("-fx-background-image: url('file:/D:/image.jpg');"

+ "-fx-background-size: cover;");

Scene scene = new Scene(root, 400, 300);

primaryStage.setScene(scene);

primaryStage.show();

}

private boolean authenticateUser(String username, String password) {

String query

"SELECT 3.2 LOGIN WINDOW DESIGN :

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.scene.layout.BorderPane;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import javafx.scene.paint.LinearGradient;

import javafx.scene.paint.CycleMethod;

import javafx.scene.paint.Stop;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class LoginWindow extends Application {

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Login Page");

// Create login form components

Label userLabel = new Label("Username:");

TextField userField = new TextField();

userField.setPromptText("Enter your username");

Label passLabel = new Label("Password:");

PasswordField passField = new PasswordField();

passField.setPromptText("Enter your password");

Button loginButton = new Button("Login");

Label messageLabel = new Label(""); // To display login status

// Styling components

userLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

passLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

loginButton.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

white; -fx-background-color: black;");

userField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

passField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

// Action for login button

loginButton.setOnAction(e -> {

String username = userField.getText();

String password = passField.getText();

if (authenticateUser(username, password)) {

messageLabel.setText("Login Successful!");

messageLabel.setTextFill(Color.GREEN);

openMainWindow(primaryStage); // Open the main window after successful login

} else {

messageLabel.setText("Invalid username or password.");

messageLabel.setTextFill(Color.RED);

}

});

// Arrange nodes in a VBox layout

VBox vbox = new VBox(10, userLabel, userField, passLabel, passField, loginButton,

messageLabel);

vbox.setAlignment(Pos.CENTER);

vbox.setPadding(new Insets(20));

// Create a gradient background

Stop[] stops = new Stop[]{

new Stop(0, Color.CORNFLOWERBLUE),

new Stop(1, Color.DARKSLATEBLUE)

};

LinearGradient gradient = new LinearGradient(0, 0, 1, 1, true,

CycleMethod.NO\_CYCLE, stops);

// Root pane setup

BorderPane root = new BorderPane();

root.setCenter(vbox);

root.setBackground(new javafx.scene.layout.Background(new

javafx.scene.layout.BackgroundFill(gradient, javafx.scene.layout.CornerRadii.EMPTY,

Insets.EMPTY)));

// Add background image using CSS style in root node

root.setStyle("-fx-background-image: url('file:/D:/image.jpg');"

+ "-fx-background-size: cover;");

Scene scene = new Scene(root, 400, 300);

primaryStage.setScene(scene);

primaryStage.show();

}

// Authenticate user using the database

private boolean authenticateUser(String username, String password) {

String query = "SELECT \* FROM users WHERE username = ? AND password = ?";

try (Connection connection = DatabaseConnection.connect();

PreparedStatement stmt = connection.prepareStatement(query)) {

stmt.setString(1, username);

stmt.setString(2, password);

ResultSet rs = stmt.executeQuery();

return rs.next();

} catch (SQLException e) {

System.out.println("Error authenticating user: " + e.getMessage());

return false;

}

}

// Open the main window after successful login

private void openMainWindow(Stage primaryStage)

3.3 HOME PAGE DESIGN :

import javafx.application.Application;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.image.Image;

import javafx.scene.layout.Background;

import javafx.scene.layout.BackgroundImage;

import javafx.scene.layout.BackgroundPosition;

import javafx.scene.layout.BackgroundRepeat;

import javafx.scene.layout.BackgroundSize;

import javafx.scene.layout.StackPane;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.stage.Stage;

public class Home extends Application {

@Override

public void start(Stage primaryStage) {

// Call method to open the main window after successful login

openMainWindow(primaryStage);

}

// Method to open the main window with buttons for different sections

private void openMainWindow(Stage primaryStage) {

// Create buttons for different sections (Patient Registration, Billing, etc.)

Button patientRegistrationButton = new Button("Patient Registration");

Button billingButton = new Button("Billing");

Button invoiceButton = new Button("Invoices");

Button medicationButton = new Button("Medications");

Button servicesButton = new Button("Services");

Button paymentsButton = new Button("Payments");

// Set button styles (fonts, size, etc.)

patientRegistrationButton.setFont(new Font("Arial", 14));

billingButton.setFont(new Font("Arial", 14));

invoiceButton.setFont(new Font("Arial", 14));

medicationButton.setFont(new Font("Arial", 14));

servicesButton.setFont(new Font("Arial", 14));

paymentsButton.setFont(new Font("Arial", 14));

// Event handlers for buttons to open the corresponding sections

patientRegistrationButton.setOnAction(e -> openPatientRegistration());

billingButton.setOnAction(e -> openBilling());

invoiceButton.setOnAction(e -> openInvoices());

medicationButton.setOnAction(e -> openMedications());

servicesButton.setOnAction(e -> openServices());

paymentsButton.setOnAction(e -> openPayments());

// Layout for the main window with vertical arrangement of buttons

VBox layout = new VBox(20); // Adjust spacing between buttons

layout.getChildren().addAll(

patientRegistrationButton,

billingButton,

invoiceButton,

medicationButton,

servicesButton,

paymentsButton

);

layout.setAlignment(Pos.CENTER); // Center-align buttons

// Set background image

StackPane root = new StackPane();

BackgroundImage backgroundImage = new BackgroundImage(

new Image("file:/D:/image.jpg"), // Replace with your image path

BackgroundRepeat.NO\_REPEAT,

BackgroundRepeat.NO\_REPEAT,

BackgroundPosition.CENTER,

BackgroundSize.DEFAULT

);

root.setBackground(new Background(backgroundImage));

// Add layout to the root pane

root.getChildren().add(layout);

// Set the scene for the main window

Scene scene = new Scene(root, 600, 400); // Increase window size for better visibility

primaryStage.setTitle("Hospital Management System");

primaryStage.setScene(scene);

primaryStage.show();

}

// Methods to handle button clicks for each section

private void openPatientRegistration() {

// Logic to open the Patient Registration page

System.out.println("Patient Registration Page Opened");

}

private void openBilling() {

// Logic to open the Billing page

System.out.println("Billing Page Opened");

}

private void openInvoices() {

// Logic to open the Invoices page

System.out.println("Invoices Page Opened");

}

private void openMedications() {

// Logic to open the Medications page

System.out.println("Medications Page Opened");

}

private void openServices() {

// Logic to open the Services page

System.out.println("Services Page Opened");

}

private void openPayments() {

// Logic to open the Payments page

System.out.println("Payments Page Opened");

}

public static void main(String[] args) {

launch(args);

}

}

PATIENT REGISTRATION PAGE DESIGN

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class PatientRegistration extends Application {

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Patient Registration");

// Create the root layout (GridPane)

GridPane grid = new GridPane();

grid.setAlignment(Pos.CENTER);

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(25, 25, 25, 25));

// Set the background image

grid.setStyle("-fx-background-image: url(file:/D:/image.jpg);"

+ "-fx-background-size: cover; -fx-background-position: center;");

// Create and add form fields for patient registration

Label nameLabel = new Label("Patient Name:");

nameLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(nameLabel, 0, 0);

TextField nameField = new TextField();

nameField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(nameField, 1, 0);

Label ageLabel = new Label("Age:");

ageLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(ageLabel, 0, 1);

TextField ageField = new TextField();

ageField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill: black;");

grid.add(ageField, 1, 1);

Label genderLabel = new Label("Gender:");

genderLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(genderLabel, 0, 2);

ComboBox<String> genderComboBox = new ComboBox<>();

genderComboBox.getItems().addAll("Male", "Female", "Other");

genderComboBox.setValue("Male"); // Default value

genderComboBox.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-textfill: black;");

grid.add(genderComboBox, 1, 2);

Label contactLabel = new Label("Contact Number:");

contactLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(contactLabel, 0, 3);

TextField contactField = new TextField();

contactField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(contactField, 1, 3);

// Create an address field

Label addressLabel = new Label("Address:");

addressLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(addressLabel, 0, 4);

TextField addressField = new TextField();

addressField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

black;");

grid.add(addressField, 1, 4);

// Create a medical history field

Label medicalHistoryLabel = new Label("Medical History:");

medicalHistoryLabel.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-textfill: black;");

grid.add(medicalHistoryLabel, 0, 5);

TextField medicalHistoryField = new TextField();

medicalHistoryField.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-textfill: black;");

grid.add(medicalHistoryField, 1, 5);

// Register button setup

Button registerButton = new Button("Register");

registerButton.setStyle("-fx-font-family: 'Roboto'; -fx-font-weight: bold; -fx-text-fill:

white; -fx-background-color: black;");

grid.add(registerButton, 1, 6);

// Set action for Register button

registerButton.setOnAction(e -> {

String name = nameField.getText();

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

String gender = genderComboBox.getValue();

String contactInfo = contactField.getText();

String address = addressField.getText();

String medicalHistory = medicalHistoryField.getText();

// Insert the patient into the database

MEDICATIONS PAGE DESIGN

import javafx.scene.Scene;

import javafx.scene.control.TextField;

import javafx.scene.control.Button;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

public class Medications extends Stage {

public Medications() {

setTitle("Medications");

// Create TextFields with CSS styles

TextField patientIDField = new TextField();

patientIDField.setPromptText("Enter Patient ID");

patientIDField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight:

bold; -fx-text-fill: black;");

TextField medicationField = new TextField();

medicationField.setPromptText("Enter Medication");

medicationField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight:

bold; -fx-text-fill: black;");

TextField dosageField = new TextField();

dosageField.setPromptText("Enter Dosage");

dosageField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight: bold;

-fx-text-fill: black;");

// Create Save Button with style

Button saveButton = new Button("Save Medication");

saveButton.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 16px; -fx-font-weight: bold; -

fx-text-fill: white; -fx-background-color: black;");

// Set the save action

saveButton.setOnAction(e -> {

String patientID = patientIDField.getText();

String medication = medicationField.getText();

String dosage = dosageField.getText();

// Input validation

if (patientID.isEmpty() || medication.isEmpty() || dosage.isEmpty()) {

showAlert(AlertType.ERROR, "Validation Error", "Please fill out all fields.");

return;

}

// Save medication to the database

saveMedicationToDatabase(patientID, medication, dosage);

showAlert(AlertType.INFORMATION, "Success", "Medication saved successfully.");

// Clear fields after saving

patientIDField.clear();

medicationField.clear();

dosageField.clear();

});

// Layout for the medications form

VBox layout = new VBox(10);

layout.getChildren().addAll(patientIDField, medicationField, dosageField, saveButton);

// Set background image with CSS style

layout.setStyle("-fx-background-image: url('file:/D:/image.jpg');"

+ "-fx-background-size: cover;");

// Set the scene and window properties

Scene scene = new Scene(layout, 400, 300);

setScene(scene);

}

// Database connection details

private Connection connectToDatabase() throws SQLException {

String url = "jdbc:mysql://localhost:3306/hospital\_management";

String user = "root"; // replace with your MySQL username

String password = "kumaran"; // replace with your MySQL password

return DriverManager.getConnection(url, user, password);

}

// Save medication information to the database

private void saveMedicationToDatabase(String patientID, String medication, String

dosage) {

String sql = "INSERT INTO medications (patient\_id, medication, dosage) VALUES (?,

?, ?)";

try (Connection conn = connectToDatabase();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, patientID);

pstmt.setString(2, medication);

pstmt.setString(3, dosage);

pstmt.executeUpdate();

} catch (SQLException e) {

showAlert(AlertType.ERROR, "Database Error", "Failed to save medication: " +

e.getMessage());

}

}

// Show alert helper method

private void showAlert(AlertType alertType, String title, String message) {

Alert alert = new Alert(alertType);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

SERVICES PAGE DESIGN

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.TextField;

import javafx.scene.control.Button;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javafx.scene.control.Alert;

public class Services extends Application {

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Services");

// Create TextFields with styling

TextField serviceNameField = new TextField();

serviceNameField.setPromptText("Enter Service Name");

serviceNameField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight:

bold; -fx-text-fill: black;");

BILLING PAGE DESIGN

import javafx.scene.Scene;

import javafx.scene.control.TextField;

import javafx.scene.control.Button;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

public class Billing {

public Billing() {

Stage billingStage = new Stage(); // Create a new Stage instance for the Billing window

billingStage.setTitle("Billing");

// Create TextFields

TextField patientIDField = new TextField();

patientIDField.setPromptText("Enter Patient ID");

patientIDField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 1

PAYMENTS PAGE DESIGN

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class Payments extends Stage {

public Payments() {

setTitle("Payments");

// Create TextFields with styling

TextField billingIDField = new TextField();

billingIDField.setPromptText("Enter Billing ID");

billingIDField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight:

bold; -fx-text-fill: black;");

TextField amountField = new TextField();

amountField.setPromptText("Enter Payment Amount");

amountField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight: bold;

-fx-text-fill: black;");

TextField dateField = new TextField();

dateField.setPromptText("Enter Payment Date");

dateField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight: bold; -

fx-text-fill: black;");

// Create Save Button with styling

Button saveButton = new Button("Save Payment");

saveButton.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 16px; -fx-font-weight: bold; -

fx-text-fill: white; -fx-background-color: black;");

// Action for Save Button

saveButton.setOnAction(e -> savePayment(billingIDField.getText(),

amountField.getText(), dateField.getText()));

// Layout for the payment form

VBox layout = new VBox(10);

layout.getChildren().addAll(billingIDField, amountField, dateField, saveButton);

// Set background image and styling

layout.setStyle("-fx-background-image: url('file:/D:/image.jpg');"

+ "-fx-background-size: cover;");

// Set the scene and window properties

Scene scene = new Scene(layout, 400, 300);

setScene(scene);

}

// Method to save payment information to the database

private void savePayment(String billingID, String amount, String date) {

// SQL query to insert payment details

String query = "INSERT INTO payments (billing\_id, amount, payment\_date) VALUES

(?, ?, ?)";

// Connect to the database and execute the query

try (Connection connection = DatabaseConnection.connect();

PreparedStatement statement = connection.prepareStatement(query)) {

statement.setString(1, billingID);

statement.setString(2, amount);

statement.setString(3, date);

int rowsAffected = statement.executeUpdate();

if (rowsAffected > 0) {

showAlert(Alert.AlertType.INFORMATION, "Success", "Payment saved

successfully.");

}

} catch (SQLException e) {

showAlert(Alert.AlertType.ERROR, "Error", "Failed to save payment: " +

e.getMessage());

}

}

// Helper method to display alerts

private void showAlert(Alert.AlertType alertType, String title, String message) {

Alert alert = new Alert(alertType);

alert.setTitle(title);

alert.setContentText(message);

alert.showAndWait();

}

INVOICE PAGE DESIGN

import javafx.scene.Scene;

import javafx.scene.control.TextField;

import javafx.scene.control.Button;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

public class Invoices extends Stage {

public Invoices() {

setTitle("Invoices");

// Create TextFields with CSS styles

TextField billingIDField = new TextField();

billingIDField.setPromptText("Enter Billing ID");

billingIDField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight:

bold; -fx-text-fill: black;");

TextField dateField = new TextField();

dateField.setPromptText("Enter Date (YYYY-MM-DD)");

dateField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight: bold; -

fx-text-fill: black;");

TextField totalField = new TextField();

totalField.setPromptText("Enter Total");

totalField.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 14px; -fx-font-weight: bold; -

fx-text-fill: black;");

// Create Save Button with style

Button saveButton = new Button("Save Invoice");

saveButton.setStyle("-fx-font-family: 'Arial'; -fx-font-size: 16px; -fx-font-weight: bold; -

fx-text-fill: white; -fx-background-color: black;");

// Set the save action

saveButton.setOnAction(e -> {

String billingID = billingIDField.getText();

String date = dateField.getText();

String totalText = totalField.getText();

// Input validation

if (billingID.isEmpty() || date.isEmpty() || totalText.isEmpty()) {

showAlert(AlertType.ERROR, "Validation Error", "Please fill out all fields.");

return;

}

try {

double total = Double.parseDouble(totalText);

saveInvoiceToDatabase(billingID, date, total);

showAlert(AlertType.INFORMATION, "Success", "Invoice saved successfully.");

// Clear fields after saving

billingIDField.clear();

dateField.clear();

totalField.clear();

} catch (NumberFormatException ex) {

showAlert(AlertType.ERROR, "Invalid Input", "Total must be a valid number.");

}

});

// Layout for the invoice form

VBox layout = new VBox(10);

layout.getChildren().addAll(billingIDField, dateField, totalField, saveButton);

// Set background image with CSS style

layout.setStyle("-fx-background-image: url('file:/D:/image.jpg');"

+ "-fx-background-size: cover;");

// Set the scene and window properties

Scene scene = new Scene(layout, 400, 300);

setScene(scene);

}

// Database connection details

private Connection connectToDatabase() throws SQLException {

String url = "jdbc:mysql://localhost:3306/hospital\_management";

String user = "root"; // replace with your MySQL username

String password = "kumaran"; // replace with your MySQL password

return DriverManager.getConnection(url, user, password);

}

// Save invoice information to the database

private void saveInvoiceToDatabase(String billingID, String date, double total) {

String sql = "INSERT INTO invoices (billing\_id, date, total) VALUES (?, ?, ?)";

try (Connection conn = connectToDatabase();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, billingID);

pstmt.setString(2, date);

pstmt.setDouble(3, total);

pstmt.executeUpdate();

} catch (SQLException e) {

showAlert(AlertType.ERROR, "Database Error", "Failed to save invoice: " +

e.getMessage());

}

}

// Show alert helper method

private void showAlert(AlertType alertType, String title, String message) {

Alert alert = new Alert(alertType);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

}

3.10 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\_management";

private static final String USER = "root";

private static final String PASSWORD = "kumaran";

public static Connection connect() {

try {

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

} catch (SQLException e) {

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

return null;

}

}

# Conclusion

**The Hospital Management System project**, developed under experienced guidance, demonstrates a meticulous approach to both design and implementation. With user friendly functionalities such as adding/viewing quiz questions, managingdifficulty levels, and tracking user performance, the system provides an engaging and efficient quiz taking experience.

Robust security measures, particularly in the administrator loginand question management modules, underscore a commitment todata integrity and system protection. This project serves as a comprehensive solution that meets current educational and training needs while providing a flexible foundation for future enhancements scalability.

# References

* + 1. https://[www.javatpoint.com/java](http://www.javatpoint.com/java) awt
    2. https://[www.javatpoint.com/java](http://www.javatpoint.com/java) swing
    3. [http://www.codeproject.com](http://www.codeproject.com/)
    4. [http://www.udemy.com](http://www.udemy.com/)
    5. [http://www.support.microsoft.com](http://www.support.microsoft.com/)
    6. [http://www.codeacademy.com](http://www.codeacademy.com/)

# Github LINK:

1. [Hospital Management System Project on GitHub](https://github.com/Harish2511/Hospital-Management-System)
2. [GitHub - Hospital Management System in Java](https://github.com/search?q=hospital%2Bmanagement%2Bsystem%2Bjava)