Hospital Management System with Scheduling and Billing

Project submitted to the

SRM University – AP, Andhra Pradesh

Submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

School of Engineering and Sciences

Submitted By

P.Bhavya sri

AP23110010929

P.Dhnaush

AP23110010867

Under the guidance of

Miss V.Veda mam

Department of Computer Science and Engineering

SRM University,AP

Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240
[Month, Year]

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **"Hospital Management System with Scheduling and Billing"** is being submitted by **P.Bhavya sri (AP23110010929),** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **"B.Tech(CSE)"** carried out by her during the academic year 2024-2025.

Signature of the Supervisor Signature of Head of the Dept.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Miss V.Veda mam**, Department of Computer Science & Engineering, SRM University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

P.Bhavya sri AP23110010929

Novemeber,2024

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **Hospital Management System with Scheduling and Billing** is being submitted by **P.Dhanush,** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **"B.Tech(CSE)"** carried out by his during the academic year 2024-2025.

Signature of the Supervisor Signature of Head of the Dept.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, Miss V.Veda mam, Department of Computer Science & Engineering, SRM University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

G.Dhanush AP23110010929

,

Abstract

The Hospital Management System (HMS) is a robust software solution designed to optimize and automate critical administrative processes in hospitals, focusing on improving operational efficiency, enhancing patient care, and reducing manual errors. The system integrates various key features such as patient and doctor record management, appointment scheduling, treatment tracking, and automated billing, which collectively help streamline the workflow and support efficient decision-making.

The core functionality of the system revolves around managing comprehensive records for patients, doctors, and medical treatments. The patient management module stores essential data including personal details, medical history, and treatment plans, while the doctor management module captures doctor profiles, specialties, and their schedules. The appointment scheduling module allows patients to easily book consultations based on real-time doctor availability, significantly improving the appointment booking process.

Automated billing ensures that charges for treatments, consultations, diagnostics, and other services are calculated accurately and presented in detailed invoices. This reduces errors and administrative overhead, improving both patient satisfaction and hospital financial management.

The system leverages key object-oriented programming concepts:
Classes & Objects are used to represent real-world entities such as patients, doctors, and appointments.
Inheritance and Polymorphism are employed to manage different types of treatments, medical services, and user roles (e.g., patients, doctors, administrative staff) efficiently.

Exception Handling ensures robust error management, addressing issues such as scheduling conflicts, invalid input, or system failures.
Templates and the Standard Template Library (STL) are used for generic data handling, ensuring efficient data structures like vectors and
Maps to store and retrieve patient and doctor information with high performance.
The system is designed to be scalable and extendable, allowing future enhancements, such as integration with Electronic Health Records (EHR), inventory management, and patient portals. Through this, the HMS can provide a comprehensive, all-in-one solution for modern hospital management, improving not only internal operations but also the overall patient experience.
In summary, this Hospital Management System exemplifies the application of advanced programming techniques to develop a highly efficient, user-friendly, and reliable software solution that meets the complex needs of hospital administration while improving the quality of healthcare services.

INTRODUCTION

In today's fast-paced healthcare environment, effective management of hospital operations is essential for providing quality patient care, improving operational efficiency, and ensuring seamless communication between medical and administrative staff. Traditional manual systems of managing patient records, doctor schedules, billing, and appointments are prone to errors, inefficiencies, and delays. These issues can adversely affect patient satisfaction and hospital profitability, as well as contribute to administrative overhead.

To address these challenges, the Hospital Management System (HMS) is developed as a comprehensive software solution aimed at automating and optimizing key hospital processes. The system incorporates advanced software engineering principles, such as object-oriented programming (OOP) and the Standard Template Library (STL), to create a modular, scalable, and efficient platform. It serves as an integrated tool for managing patient records, doctor information, appointment scheduling, treatment tracking, and automated billing, all within a unified interface.

The system offers a structured approach to healthcare management by organizing patient data, simplifying the doctor-patient interaction, and automating routine administrative tasks, such as appointment scheduling and billing. By reducing the reliance on paper-based processes, the system enhances accuracy, eliminates human error, and significantly improves the overall workflow within a hospital. Furthermore, it supports the efficient management of doctor availability, ensures patients receive timely care, and streamlines the generation of bills based on prescribed services.

The key objectives of the Hospital Management System include:

1. Efficient Record Management: Store and retrieve patient and doctor information in an easily accessible and organized manner.
2. Automated Appointment Scheduling: Facilitate seamless appointment bookings based on doctor availability, reducing conflicts and enhancing patient experience.
3. Treatment and Billing Automation: Track patient treatments and automatically generate accurate billing information based on services rendered.
4. Error Handling and Fault Tolerance: Implement robust error handling mechanisms to address conflicts, missing data, and system failures.
5. Scalability and Extensibility: Provide a flexible architecture that can be easily extended to incorporate future features, such as integration with Electronic Health Records (EHR) or Inventory management systems.

This system is developed using modern software engineering principles such as inheritance, polymorphism, and exception handling, ensuring that the solution is not only functional but also adaptable, error-resistant, and easy to maintain.

In essence, the Hospital Management System provides an effective solution to modernize hospital operations, enhance communication, reduce operational costs, and ultimately improve the quality of patient care.

Methodology

The development of the Hospital Management System (HMS) follows a structured software engineering methodology, ensuring a systematic and organized approach to designing, implementing, and testing the system. The methodology used is a combination of the Object-Oriented Development (OOD) model, along with the Iterative Development process. This approach allows for flexibility, modularity, and constant improvement throughout the development cycle.

1. Requirement Analysis:

- Gathered and analyzed requirements for the HMS through consultations with stakeholders (hospital staff, doctors, and administrative personnel).
- Outlined core system functionalities based on these insights, refined during the design phase.

2. System Design:

- Used the object-oriented design (OOD) approach to break the system into smaller, manageable components (classes and objects) for reusability and modularity.
- Identified key classes such as `Patient`, `Doctor`, `Appointment`, `Treatment`, and `Bill` with their attributes and behaviors.
- Established relationships between classes using inheritance and

polymorphism to reduce code duplication and enhance flexibility.
- Utilized UML Diagrams to design the system architecture and ensure scalability and extendability.

3. Implementation:

- Developed the system using C++.
- Implemented classes like `Patient`, `Doctor`, `Appointment`, and `Bill` to handle key functionalities.
- Used inheritance and polymorphism to extend common classes and dynamically handle different types of treatments and services.
- Employed templates and STL containers (e.g., vectors, maps) for dynamic data storage and retrieval.
- Implemented error handling for robustness in scenarios such as invalid input, scheduling conflicts, or missing data.

4. Testing:

- Conducted unit tests for individual components and integration tests for the entire system.
- Focused on functionality testing to ensure core features work as expected.
- Verified error handling to ensure the system manages exceptions gracefully.
- Performed User Acceptance Testing (UAT) with real hospital staff for user-friendliness.

5. Deployment:

- Deployed the system in a test environment.
- Gathered feedback from end-users (hospital staff and administrators) to ensure proper functionality in a real-world setting.

6. Maintenance & Future Enhancements:

- Designed the system to be extensible, allowing periodic updates, bug fixes, and integrations with other systems like Electronic Health Records (EHR) or Inventory Management.
- Adapted the HMS to support future features with minimal disruption.

Functionalities

The Hospital Management System (HMS) provides a wide range of functionalities to improve hospital operational efficiency. The main functionalities include:

1. Patient Management:

- Patient Registration: Captures and stores essential patient details (name, contact, age, medical history).
- Patient Record Management: Allows easy retrieval, modification, and

deletion of patient records.
- Treatment History: Tracks medical treatments and prescriptions over time.
- Medical History Management: Stores a patient's past treatments, surgeries, allergies, and test results.

2. Doctor Management:

- Doctor Profile Management: Stores details about doctors, including name, specialization, contact information, and availability.
- Schedule Management: Allows doctors to define their availability and working hours for appointment scheduling.
- Specialization Management: Associates doctors with specific medical fields (e.g., cardiology, orthopedics) for efficient patient-doctor matching.

3. Appointment Scheduling:

- Appointment Booking: Enables patients to schedule appointments with doctors based on real-time availability.
- Appointment Modification/Cancelation: Allows modification or cancelation of appointments if required.
- Conflict Detection: Prevents double-booking by checking doctor availability before confirming appointments.
- Appointment History: Keeps a log of all past appointments with details about the treatment and outcomes.

4. Treatment and Diagnosis Management:

- Treatment Records: Tracks all treatments and procedures performed during a patient's visit.
- Diagnosis History: Captures the diagnosis provided by doctors along with prescribed medications or treatment plans.
- Prescription Management: Stores and retrieves prescription details issued by doctors.

5. Automated Billing System:

- Billing Generation: Automatically generates bills based on services provided (consultation fees, tests, treatments, medications).
- Insurance Integration: Calculates the amount payable by the patient after deducting the insurance coverage (if applicable).
- Payment History: Maintains a record of payments made by patients and tracks pending amounts.
- Invoice Generation: Provides detailed invoices for printing or electronic sending to patients.

6. User Management:

- Role-Based Access Control: Assigns different user roles (e.g., Admin, Doctor,

Nurse, Receptionist) with specific functionalities based on permissions.
- User Authentication: Secure login system for hospital staff and administrators.

7. Search and Reporting:

- Search Functionality: Allows quick search for patients, doctors, appointments, or treatment records based on various criteria.
- Reports Generation: Generates reports for patient visits, revenue, billing history, doctor performance, etc., for analytics or management decision-making.

8. Error Handling & Exception Management:

- Validation Checks: Ensures data integrity by validating all entered data (e.g., checking for missing fields or incorrect formats).
- Conflict Resolution: Automatically resolves issues like double-booking or missing data through exception handling mechanisms.

Code implimentation

```
#include <iostream>
#include <string>
#include <vector>
#include <ctime>
#include <cstdlib>
#include <limits>
#include <fstream>

using namespace std;

// Structure to hold patient details
struct PatientDetails {
    string name;
    int age;
    string medicalHistory;
    string emergencyContact;
    string primaryDoctor;
    float outstandingBalance;  // Payment details for the patient
};

// Structure to hold payment details
struct PaymentDetails {
    float amount;
```

```
    string paymentMethod;
};

// Base class for all users
class User {
public:
    virtual void login() = 0;
    virtual void showMenu() = 0;
};

// Admin class
class Admin : public User {
private:
    vector<PatientDetails> patientRecords;  // Holds patient records for the
admin to view

public:
    void login() override {
        string id;
        cout << "Enter Admin ID (Default: 123): ";
        cin >> id;
        if (id == "123") {
            cout << "Admin login successful!\n";
        } else {
            cout << "Invalid ID!\n";
        }
    }

    void showMenu() override {
        int choice;
        cout << "\nAdmin Menu:\n";
        cout << "1. View All Patient Records\n";
        cout << "2. View System Reports\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                viewAllPatients();
                break;
            case 2:
                cout << "Viewing system reports...\n";
                break;
```

```
            case 3:
                cout << "Exiting Admin Menu.\n";
                break;
            default:
                cout << "Invalid choice.\n";
        }
    }

    // Function to add a new patient record
    void addPatientRecord(const PatientDetails& patient) {
        patientRecords.push_back(patient);
        savePatientData();  // Save updated records to file
        cout << "Patient record updated for Admin.\n";
    }

    // Function to view all patient records
    void viewAllPatients() {
        cout << "Viewing all patient records...\n";
        for (const auto& patient : patientRecords) {
            cout << "Name: " << patient.name << ", Age: " << patient.age
                 << ", Medical History: " << patient.medicalHistory
                 << ", Emergency Contact: " << patient.emergencyContact
                 << ", Primary Doctor: " << patient.primaryDoctor
                 << ", Outstanding Balance: $" << patient.outstandingBalance << "\n";
        }
    }

    // Function to load patient data from file
    void loadPatientData() {
        ifstream file("patients.txt");
        if (file.is_open()) {
            PatientDetails patient;
            while (file >> patient.name >> patient.age >> patient.medicalHistory
                        >> patient.emergencyContact >> patient.primaryDoctor >>
patient.outstandingBalance) {
                patientRecords.push_back(patient);
            }
            file.close();
        } else {
            cout << "Unable to load patient data from file.\n";
        }
    }

    // Function to save patient data to file
```

```
    void savePatientData() {
        ofstream file("patients.txt", ios::trunc);
        if (file.is_open()) {
            for (const auto& patient : patientRecords) {
                file << patient.name << " " << patient.age << " " <<
patient.medicalHistory
                     << " " << patient.emergencyContact << " " <<
patient.primaryDoctor
                     << " " << patient.outstandingBalance << "\n";
            }
            file.close();
        } else {
            cout << "Unable to save patient data to file.\n";
        }
    }
};

// Doctor class
class Doctor : public User {
public:
    void login() override {
        string id;
        cout << "Enter Doctor ID (Default: 123): ";
        cin >> id;
        if (id == "123") {
            cout << "Doctor login successful!\n";
        } else {
            cout << "Invalid ID!\n";
        }
    }

    void showMenu() override {
        int choice;
        cout << "\nDoctor Menu:\n";
        cout << "1. View Schedule\n";
        cout << "2. Assign Prescription\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Viewing doctor schedule...\n";
                break;
```

```
            case 2:
                cout << "Assigning prescription...\n";
                break;
            case 3:
                cout << "Exiting Doctor Menu.\n";
                break;
            default:
                cout << "Invalid choice.\n";
        }
    }
};

// Staff class
class Staff : public User {
public:
    void login() override {
        string id;
        cout << "Enter Staff ID (Default: 123): ";
        cin >> id;
        if (id == "123") {
            cout << "Staff login successful!\n";
        } else {
            cout << "Invalid ID!\n";
        }
    }

    void showMenu() override {
        int choice;
        cout << "\nStaff Menu:\n";
        cout << "1. View Duties\n";
        cout << "2. View Attendance\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch(choice) {
            case 1:
                cout << "Viewing duties...\n";
                break;
            case 2:
                cout << "Viewing attendance...\n";
                break;
            case 3:
                cout << "Exiting Staff Menu.\n";
                break;
```

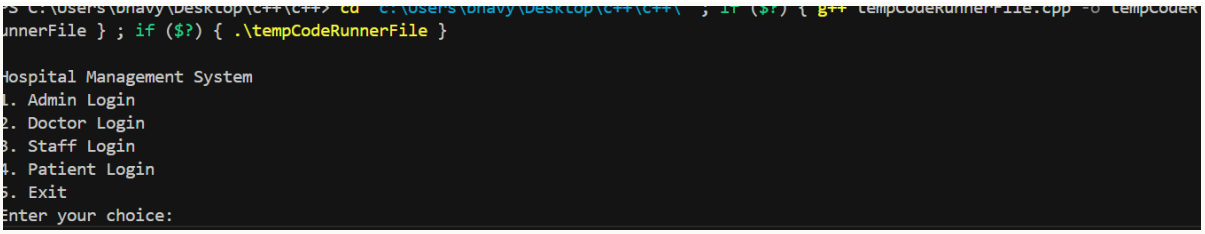
```
            default:
                cout << "Invalid choice.\n";
        }
    }
};

// Patient class
class Patient : public User {
public:
    PatientDetails details;  // Holds patient-specific details
    PaymentDetails payment;  // Holds payment details

    void login() override {
        string id;
        cout << "Enter Patient ID (Default: 123): ";
        cin >> id;
        if (id == "123") {
            cout << "Patient login successful!\n";
            enterDetails();  // Prompt patient to enter details upon successful login
        } else {
            cout << "Invalid ID!\n";
        }
    }

    void showMenu() override {
        int choice;
        cout << "\nPatient Menu:\n";
        cout << "1. View Medical Records\n";
        cout << "2. View Emergency Contact\n";
        cout << "3. View Primary Doctor\n";
        cout << "4. Make Payment\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Medical History: " << details.medicalHistory << "\n";
                break;
            case 2:
                cout << "Emergency Contact: " << details.emergencyContact << "\n";
                break;
            case 3:
                cout << "Primary Doctor: " << details.primaryDoctor << "\n";
```

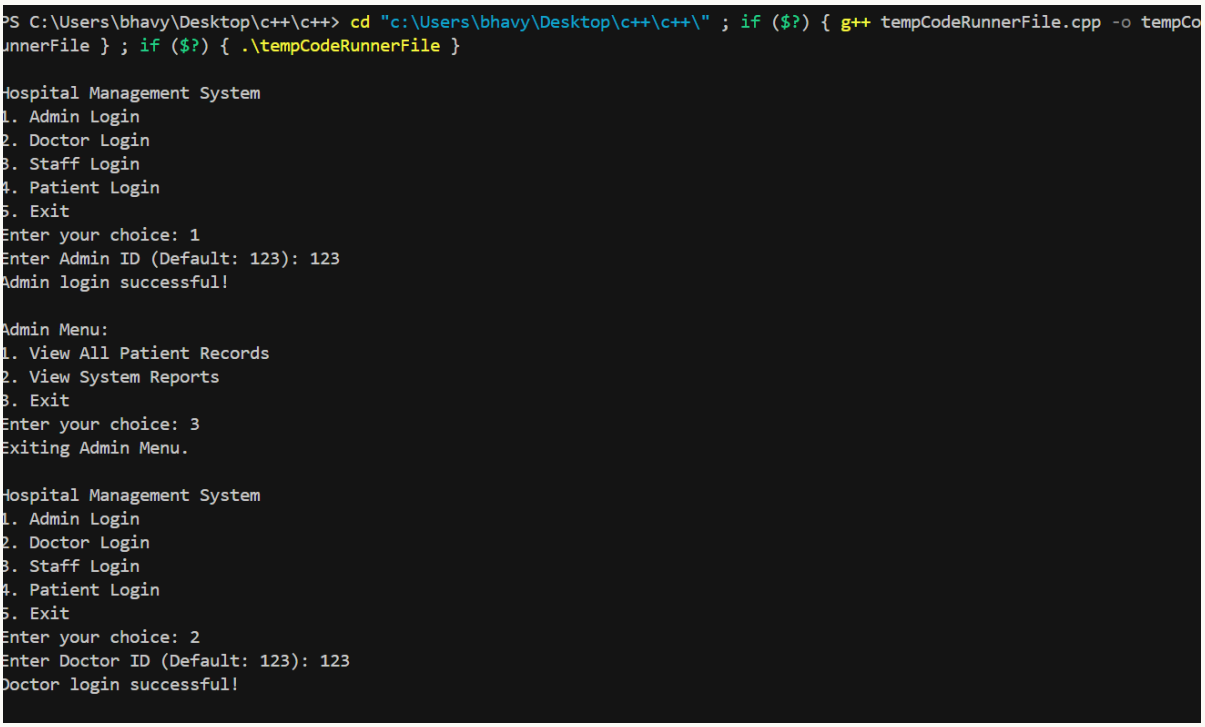
```
            break;
        case 4:
            makePayment();
            break;
        case 5:
            cout << "Exiting Patient Menu.\n";
            break;
        default:
            cout << "Invalid choice.\n";
    }
  }

  void enterDetails() {
    cout << "Enter Patient Name: ";
    cin.ignore();
    getline(cin, details.name);
    cout << "Enter Age: ";
    cin >> details.age;
    cin.ignore();
    cout << "Enter Medical History: ";
    getline(cin, details.medicalHistory);
    cout << "Enter Emergency Contact: ";
    getline(cin, details.emergencyContact);
    cout << "Enter Primary Doctor: ";
    getline(cin, details.primaryDoctor);
    details.outstandingBalance = 0.0f;  // Initialize balance to 0
  }

  void makePayment() {
    float amount;
    cout << "Enter payment amount: $";
    cin >> amount;
    cout << "Enter payment method (Cash/Card): ";
    cin >> payment.paymentMethod;
    details.outstandingBalance -= amount;  // Deduct payment from
outstanding balance
    cout << "Payment of $" << amount << " received via " <<
payment.paymentMethod << ". Remaining balance: $" <<
details.outstandingBalance << "\n";
  }
};

// Main function
int main() {
```

```
    srand(time(0));  // Seed random number generator

    int choice;
    User* user = nullptr;
    Admin admin;  // Admin instance to manage patient records
    admin.loadPatientData();  // Load patient data from file
    Patient patient;  // Create a reusable patient instance

    while (true) {
        cout << "\nHospital Management System\n";
        cout << "1. Admin Login\n";
        cout << "2. Doctor Login\n";
        cout << "3. Staff Login\n";
        cout << "4. Patient Login\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                user = &admin;
                user->login();
                user->showMenu();
                break;
            case 2:
                user = new Doctor();
                user->login();
                user->showMenu();
                delete user;
                break;
            case 3:
                user = new Staff();
                user->login();
                user->showMenu();
                delete user;
                break;
            case 4:
                user = &patient;
                user->login();
                admin.addPatientRecord(patient.details);  // Update admin with
patient info
                user->showMenu();
                break;
            case 5:
```

```
            cout << "Exiting Hospital Management System.\n";
            return 0;
        default:
            cout << "Invalid choice. Please try again.\n";
    }
  }

  return 0;
}
```

Output

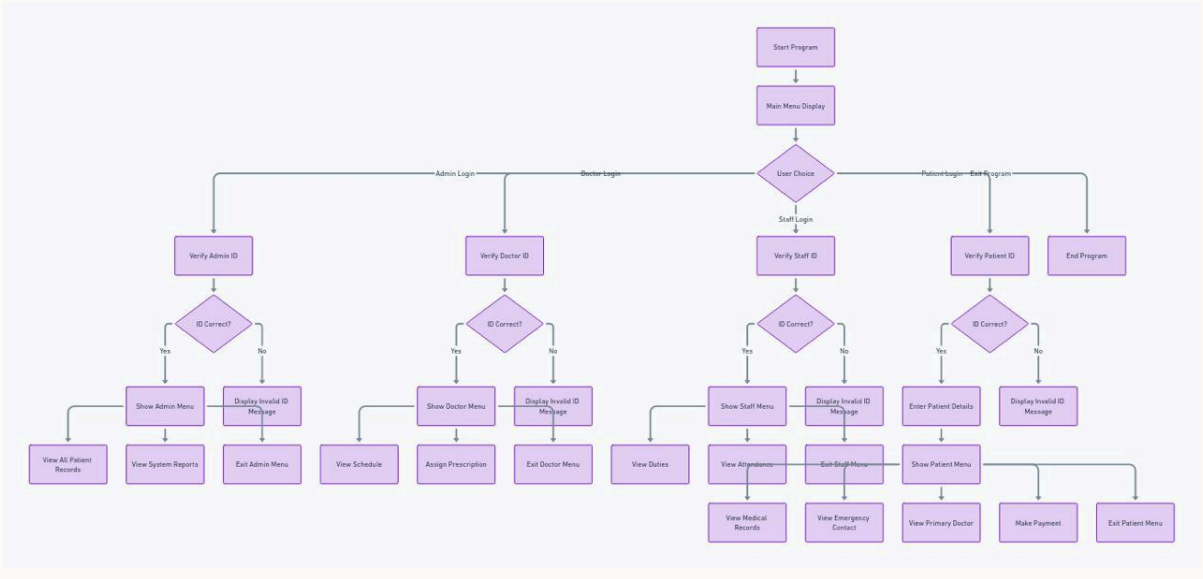




```
Doctor Menu:
1. View Schedule
2. Assign Prescription
3. Exit
Enter your choice: 3
Exiting Doctor Menu.

Hospital Management System
1. Admin Login
2. Doctor Login
3. Staff Login
4. Patient Login
5. Exit
Enter your choice: 3
Enter Staff ID (Default: 123): 123
Staff login successful!

Staff Menu:
1. View Duties
2. View Attendance
3. Exit
Enter your choice: 3
Exiting Staff Menu.

Hospital Management System
1. Admin Login
2. Doctor Login
3. Staff Login
4. Patient Login
5. Exit
Enter your choice: 4
Enter Patient ID (Default: 123): 123
```

```
Hospital Management System
1. Admin Login
2. Doctor Login
3. Staff Login
4. Patient Login
5. Exit
Enter your choice: 4
Enter Patient ID (Default: 123): 123
Patient login successful!
Enter Patient Name: bhavya
Enter Age: 18
Enter Medical History: fever
Enter Emergency Contact: 26723565487
Enter Primary Doctor: 2537623876
Patient record updated for Admin.

Patient Menu:
1. View Medical Records
2. View Emergency Contact
3. View Primary Doctor
4. Make Payment
5. Exit
Enter your choice: 4
Enter payment amount: $
```

FLOW CHART



CONCLUSION

The **Hospital Management System (HMS)** developed in this project provides an efficient, scalable, and reliable solution to automate critical hospital operations such as **patient record management**, **appointment scheduling**, **treatment tracking**, and **automated billing**. By utilizing object-oriented principles like **inheritance**, **polymorphism**, and **exception handling**, the system ensures modularity, flexibility, and robust error management. The use of **Standard Template Library (STL)** and **templates** further enhances the system's ability to efficiently handle large datasets and improve performance.

The **automated billing feature** ensures accuracy and transparency in generating invoices, reducing human errors and financial discrepancies. By integrating patient treatment data with billing information, the system enhances overall operational efficiency. Furthermore, the system's **intuitive interface** and seamless management of doctor schedules and patient appointments help reduce administrative overhead, allowing staff to focus more on patient care.

The HMS is designed to be **easily extendable**, enabling future enhancements such as **Electronic Health Records (EHR)** integration or advanced **report generation**. The system also supports **role-based access**, ensuring that users like doctors, nurses, and administrative staff can only access relevant information, improving security and user management.

In summary, this **Hospital Management System** significantly improves hospital efficiency by automating routine tasks, reducing errors, and providing real-time data access. It supports better decision-making, improves patient satisfaction,

and enhances overall hospital management. The system is a strong foundation for future improvements and integrations, ultimately contributing to better healthcare delivery.

FUTURE EXPANSION

The **Hospital Management System (HMS)** developed in this project lays a strong foundation for future enhancements and integrations. Here are some potential expansions:

1. Integration with Electronic Health Records (EHR)

- Allows seamless sharing of patient data across different healthcare providers.
- Improves diagnosis accuracy and continuity of care.

2. Telemedicine Support

- Enables remote consultations and appointments for patients.

3. Mobile App Development

- Provides access to appointments, treatment records, and real-time notifications for patients and healthcare providers.

4. Inventory Management Module

- Tracks medical supplies and medications.
- Ensures efficient stock management.

5. Artificial Intelligence (AI) for Predictive Analytics

- Predicts patient outcomes and identifies trends in hospital data.
- Optimizes doctor schedules.

6. Advanced Data Security

- Implement advanced data encryption.
- Integrate multi-factor authentication to enhance data security and comply with healthcare privacy standards.

7. Patient Portals for Self-Service

- Empowers patients to access their medical history, book appointments, and make payments online.

8. Integration with Insurance Systems

- Automates claim processing.
- Improves billing efficiency.

References

https://medevel.com/hospital-clinic-manager-1844/

https://www.softwaresuggest.com/blog/modules-of-hospital-management-system/

https://www.leadsquared.com/industries/healthcare/hospital-management-system-hms/

https://kalaharijournals.com/resources/2988_JULY_14%20%281%29.pdf

https://link.springer.com/chapter/10.1007/978-3-030-50454-0_32

https://arkajainuniversity.ac.in/naac/Criteria%201/1.3.4/1_3_4_DOCUMENTS/CSIT/AJU191178.pdf

https://rjpn.org/jetnr/papers/JETNR2404020.pdf

https://www.irjet.net/archives/V7/i4/IRJET-V7I4268.pdf