HOSPITAL MANAGEMENT SYSTEM

Project submitted to the
SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering
School of Engineering and Sciences

Submitted by
Nandan (AP23110010815)
SivaRam (AP23110010809)
Manvith(AP23110010818)
Bharath(AP23110010859)

Under the Guidance of
Dr. Aurobindo Behera
SRM University–AP

Neerukonda, Mangalagiri, Guntur Andhra Pradesh – 522 240,

[November, 2024]

Certificate

Date: November-2024

This is to certify that the work present in this Project entitled “**Hospital Management System**” has been carried out by group-4 under my/our supervision. The work is genuine, original,and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in School of Engineering and Science.

Supervisor

Prof. / Dr. Aurobindo Behera.
Assistant Professor.
Department of Computer Science.

Co-supervisor

Prof. / Dr. Aurobindo Behra.
Assistant Professor.
Department of Computer Science.

Acknowledgements

We would like to express our gratitude Dr. Aurobindo Behera for assisting us with the research at every step. Without the advice and assistance of our adviser, this study effort would not be feasible. The suggestions, feedbacks, and encouragement from our advisors were extremely helpful in assisting us in finishing this research.

<u>ACCURACY CHECKERS:</u>

Manvith Kumar
Jhansi Bharath

<u>REVIEWERS:</u>

Siva Ram
Nandan Chowdary

<u>DATA FROM:</u>

Whole team…

Table of contents **Page No:**

Abstract

The "Hospital Management System" is a C++ project designed to manage the patient operations of a hospital service called "E-hospitality".The system provides a user-friendly interface through the console for customers to interact with various functionalities such as displaying available medicines, hospitalizing a patient, and returning a patient information.

The system utilizes object-oriented programming principles with two main classes: "Patient" and "Hospital" The "Patient" class represents individual Patients with attributes such as the Patient name, medicine brand, type of medicine model, and availability status. The "Hospital" class serves as the core of the application, featuring methods to add patients, display about their health status, to take medicine, and return a day status. The process involves collecting customer information, calculating medical fares based on disease type and duration, and supporting both cash and online payment methods.

1. Introduction

The project "Hospital Management System" is a comprehensive solution designed to automate the diverse operations of a Hospital service. The software is developed using C++,demonstrating the effective use of object-oriented principles, data structures, and algorithmic design.

1.1 Core Classes:

The software is built around several classes and functions that collectively form the Hospital Management System. The Patient class represents a bike with attributes such as Patient name, medicine model, and availability status. The Hospital class encapsulates the operations of the Patient service, including adding Patients,medicines to the system, displaying available medicines, renting a room in a hostel, and accepting medicines.

1.2 Key Features:

Key features of the system include the ability to treat a patient on an hourly basis or for an entire day, calculate the total fare based on the duration and type of disease, and accept payments through different methods. The system also generates unique token numbers and QR codes at a time.

1.2.1 Display of Registered Patients:

The system allows users to view a list of registered Patients available, transparency and community engagement.

1.3 Security Measures:

This section emphasizes the security measures implemented to safeguard the integrity of the Hospital system. Specific operations, such as adding custom options are protected by passwords to ensure that only authorized individuals,typically system administrators, can perform sensitive operations. The introduction briefly summarizes the overall contributions of the Online Quiz System project. It highlights the application of fundamental C++ concepts, the modular and structured codebase, and the practical implementation of features aimed at creating a functional and user-friendly online quiz application. The introduction concludes by positioning the project as a practical example of applying C++ principles in an educational context.

2. Methodology

The methodology used in this Hospital system employs a modular approach, offering various functionalities through a menu-driven interface. Here's an overview:

2.1. Modular Design:

Main Function:

Displays a menu of options (Display Available Medicines, Return a bill, Exit). Uses a switch statement to execute the chosen action.

1. Patient Class:

- The `Patient` class represents a Disease with attributes such as `Patient Name`, `Disease`,`medicine`, and `available`.
- The constructor initializes the Patient's attributes, with `available` set to `true` by default.

2. Medicine Class:

- The `Medicine` class shows the hospital mangement system.
- It contains a vector `medicines` to store instances of the `Medicine` class.
- The constructor initializes the random seed for generating random numbers.

3. Member Functions of `HospitalManagementSystem':

- `addPatient': Adds a new Patient to the system with the specified details.
- `displayAvailableMedicines': Displays the available Medicines in the Hospital.
- `Payment': Pays a bill based on the user's input for Disease index, duration, and payment method. It calculates the fare and generates payment details.
- `returnBill': Handles the return of a final bill, updating its availability and generating a token number for payment.

4. Private Helper Functions of `HospitalManagementSystem':

- `calculateBill': Calculates the Final bill based on the type of medicine, duration, and hours.

- `generateTokenr’: Generates a random token number for payment.
- `generateQRCode’: Generates a placeholder QR code for online payment.

5. handleMainMenu Function:
- Manages the main menu and user interactions.
- Allows the user to display available medicines, return a bill, or exit the system.
- Keep track of selected options to prevent redundant actions.

6. Main Function:
- Welcomes the user to the Hospital Management system and waits for a key press to continue.
- Creates an instance of `HospitalManagementSystem’.
- Adds sample Medicines to the system.
- Enters a loop to handle the main menu until the user chooses to exit.

7. Infinite Loop:
- The program is structured to restart from the beginning, creating a new instance of the `HospitalManagementSystem’ and allowing the user to interact with the system continuously.

8. Miscellaneous:
- The program uses the `conio.h’ library for `getch()’ to wait for a key press before proceeding.

Overall, the code provides a simple and interactive Hospital system with options to display available Medicines, return a bill, and exit the system. The system incorporates basic user input validation and generates payment details based on the chosen payment method.

3. Discussion

3.1 INPUT OF THE CODE

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

// Patient class to store patient information
class Patient {
public:
    int id;
    std::string name;
    std::string disease;
    std::string medicine;
    bool admitted;

    Patient(int pid, std::string pname, std::string pdisease, std::string pmedicine,
bool padmitted = false)
        : id(pid), name(pname), disease(pdisease), medicine(pmedicine),
admitted(padmitted) {}

    void display() const {
        std::cout << "ID: " << id << ", Name: " << name << ", Disease: " << disease
                  << ", Medicine: " << medicine << ", Admitted: " << (admitted ? "Yes" :
"No") << "\n";
    }
};
```

```
// Doctor class to represent a doctor with specialization
class Doctor {
public:
    int doctorId;
    std::string name;
    std::string specialization;

    Doctor(int did, std::string dname, std::string dspecialization)
        : doctorId(did), name(dname), specialization(dspecialization) {}

    void display() const {
        std::cout << "Doctor ID: " << doctorId << ", Name: " << name << ",
Specialization: " << specialization << "\n";
    }
};

// Medicine class to manage medicine details
class Medicine {
public:
    std::string name;
    std::string brand;
    int quantity;

    Medicine(std::string mname, std::string mbrand, int mquantity)
        : name(mname), brand(mbrand), quantity(mquantity) {}

    void display() const {
        std::cout << "Medicine Name: " << name << ", Brand: " << brand << ",
Quantity: " << quantity << "\n";
    }
};

// Appointment class to schedule and manage appointments
class Appointment {
public:
    int appointmentId;
    int patientId;
    int doctorId;
    std::string date;
    std::string time;

    Appointment(int aid, int pid, int did, std::string adate, std::string atime)
        : appointmentId(aid), patientId(pid), doctorId(did), date(adate), time(atime) {}

    void display() const {
        std::cout << "Appointment ID: " << appointmentId << ", Patient ID: " <<
patientId
                  << ", Doctor ID: " << doctorId << ", Date: " << date << ", Time: " <<
time << "\n";
    }
};

// Billing class to manage patient billing
class Billing {
public:
    int patientId;
    int totalAmount;
    bool paid;

    Billing(int pid, int amount)
        : patientId(pid), totalAmount(amount), paid(false) {}

    void makePayment() {
        paid = true;
        std::cout << "Payment of $" << totalAmount << " made successfully.\n";
    }

    void display() const {
                        std::cout << "Patient ID: " << patientId << ", Total Amount: $"
                                                                      <<totalAmount
                  << ", Paid: " << (paid ? "Yes" : "No") << "\n";
    }
};

// Hospital Management System class
class HospitalManagementSystem {
private:
    std::vector<Patient> patients;
    std::vector<Doctor> doctors;
    std::vector<Medicine> medicines;
    std::vector<Appointment> appointments;
    std::vector<Billing> bills;

    // Generates a unique ID based on vector size
    int generateUniqueId() {
        return rand() % 10000 + 1000;
    }

public:
    // Constructor to seed the random generator
    HospitalManagementSystem() {
        srand(static_cast<unsigned int>(time(0)));
    }

    // Add a new patient to the system
    void addPatient(const std::string& name, const std::string& disease, const
std::string& medicine) {
        int id = generateUniqueId();
        patients.push_back(Patient(id, name, disease, medicine));
        std::cout << "Patient " << name << " added with ID: " << id << "\n";
    }

    // Add a new doctor to the system
    void addDoctor(const std::string& name, const std::string& specialization) {
int id = generateUniqueId();
```

```
        doctors.push_back(Doctor(id, name, specialization));
        std::cout << "Doctor " << name << " added with ID: " << id << "\n";
    }

    // Add a new medicine to the system
    void addMedicine(const std::string& name, const std::string& brand, int
quantity) {
        medicines.push_back(Medicine(name, brand, quantity));
        std::cout << "Medicine " << name << " added successfully.\n";
    }

    // Schedule an appointment
    void scheduleAppointment(int patientId, int doctorId, const std::string& date,
const std::string& time) {
        int appointmentId = generateUniqueId();
        appointments.push_back(Appointment(appointmentId, patientId, doctorId,
date, time));
        std::cout << "Appointment scheduled successfully with ID: " <<
appointmentId << "\n";
    }

    // Display all patients
    void displayPatients() const {
        std::cout << "\n--- Patient List ---\n";
        for (const auto& patient : patients) {
            patient.display();
        }
    }

    // Display all doctors
    void displayDoctors() const {
        std::cout << "\n--- Doctor List ---\n";
        for (const auto& doctor : doctors) {
            doctor.display();
        }
    }

    // Display all medicines
    void displayMedicines() const {
        std::cout << "\n--- Medicine List ---\n";
        for (const auto& medicine : medicines) {
            medicine.display();
        }
    }

    // Generate bill for a patient based on treatment cost
    void generateBill(int patientId, int amount) {
        bills.push_back(Billing(patientId, amount));
        std::cout << "Bill generated for Patient ID " << patientId << " with amount:
$" << amount << "\n";
    }

    // Display all appointments
    void displayAppointments() const {
        std::cout << "\n--- Appointment List ---\n";
        for (const auto& appointment : appointments) {
            appointment.display();
        }
    }

    // Main Menu for the system
    void mainMenu() {
        int choice, id, amount;
        std::string name, disease, medicine, specialization, date, time;

        do {
            std::cout << "\n--- Hospital Management System Menu ---\n";
            std::cout << "1. Add Patient\n";
            std::cout << "2. Add Doctor\n";
            std::cout << "3. Add Medicine\n";
```

```
std::cout << "4. Schedule Appointment\n";
std::cout << "5. Generate Bill\n";
std::cout << "6. Display Patients\n";
std::cout << "7. Display Doctors\n";
std::cout << "8. Display Medicines\n";
std::cout << "9. Display Appointments\n";
std::cout << "10. Exit\n";
std::cout << "Enter choice: ";
std::cin >> choice;

switch (choice) {
    case 1:
        std::cout << "Enter Patient Name: ";
        std::cin >> name;
        std::cout << "Enter Disease: ";
        std::cin >> disease;
        std::cout << "Enter Medicine: ";
        std::cin >> medicine;
        addPatient(name, disease, medicine);
        break;
    case 2:
        std::cout << "Enter Doctor Name: ";
        std::cin >> name;
        std::cout << "Enter Specialization: ";
        std::cin >> specialization;
        addDoctor(name, specialization);
        break;
    case 3:
        std::cout << "Enter Medicine Name: ";
        std::cin >> name;
        std::cout << "Enter Brand: ";
        std::cin >> medicine;
        std::cout << "Enter Quantity: ";
        std::cin >> amount;
        addMedicine(name, medicine, amount);
        break;
    case 4:
        std::cout << "Enter Patient ID: ";
        std::cin >> id;
        std::cout << "Enter Doctor ID: ";
        std::cin >> amount; // Temporary variable for doctorId
        std::cout << "Enter Date (DD-MM-YYYY): ";
        std::cin >> date;
        std::cout << "Enter Time (HH:MM): ";
        std::cin >> time;
        scheduleAppointment(id, amount, date, time);
        break;
    case 5:
        std::cout << "Enter Patient ID: ";
        std::cin >> id;
        std::cout << "Enter Treatment Cost: ";
        std::cin >> amount;
        generateBill(id, amount);
        break;
    case 6:
        displayPatients();
        break;
    case 7:
        displayDoctors();
        break;
    case 8:
        displayMedicines();
        break;
    case 9:
        displayAppointments();
        break;
    case 10:
        std::cout << "Exiting...\n";
        break;
    default:
```

```
                std::cout << "Invalid choice. Try again.\n";
            }
        } while (choice != 10);
    }
};

int main() {
    HospitalManagementSystem hms;
    hms.mainMenu();
    return 0;
}
```

3.2. Output of code:

```
WELCOME TO HOSPITAL SERCICES
Press Any key to continue…

Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:
```

```
1. Add Patient
Enter Patient Name: JohnDoe
Enter Disease: Flu
Enter Medicine: Paracetamol

Patient JohnDoe added with ID: 1578

Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:


2. Add Doctor
Enter Doctor Name: DrSmith
Enter Specialization: Pediatrics

Doctor DrSmith added with ID: 3421
```

```
Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:


3. Add Medicine
Enter Medicine Name: Ibuprofen
Enter Brand: MedCo
Enter Quantity: 50

Medicine Ibuprofen added successfully.

Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
```

```
10. Exit

Select the action:


4. Schedule Appointment
Enter Patient ID: 1578
Enter Doctor ID: 3421
Enter Date (DD-MM-YYYY): 12-11-2024
Enter Time (HH:MM): 10:30

Appointment scheduled successfully with ID: 4567

Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:


5. Generate Bill
Enter Patient ID: 1578
Enter Treatment Cost: 250

Bill generated for Patient ID 1578 with amount: $250
```

```
Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:


6. Display Patients
--- Patient List ---
ID: 1578, Name: JohnDoe, Disease: Flu, Medicine: Paracetamol, Admitted: No
ID: 2205, Name: JaneSmith, Disease: Cold, Medicine: VitaminC, Admitted: Yes

Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit
```

```
Select the action:


7. Display Doctors

--- Doctor List ---
Doctor ID: 3421, Name: DrSmith, Specialization: Pediatrics
Doctor ID: 4896, Name: DrBrown, Specialization: Cardiology

Hospital System MENU:

1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:


8. Display Medicines

--- Medicine List ---
Medicine Name: Ibuprofen, Brand: MedCo, Quantity: 50
Medicine Name: Paracetamol, Brand: HealthPlus, Quantity: 100

Hospital System MENU:

1. Add Patient
2. Add Doctor
```

```
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:


9. Display Appointments

--- Appointment List ---
Appointment ID: 4567, Patient ID: 1578, Doctor ID: 3421, Date: 12-11-2024,
Time: 10:30
Appointment ID: 7890, Patient ID: 2205, Doctor ID: 4896, Date: 13-11-2024,
Time: 14:00

Hospital System MENU:
1. Add Patient
2. Add Doctor
3. Add Medicine
4. Schedule Appointment
5. Generate Bill
6. Display Patients
7. Display Doctors
8. Display Medicines
9. Display Appointments
10. Exit

Select the action:
10. Exit
Exiting…
```

4. Concluding Remarks

In summary, the C++ code presents a well-structured Hospital Management system that incorporates essential features such as user registration, and customization functionalities. The use of classes enhances code organization, making it modular and easy to understand. The inclusion of password-protected operations ensures a level of security, allowing certain functions to be accessible only to authorized users.

The code effectively utilizes standard input/output mechanisms for user interactions and implements a loop structure that facilitates repeated use until the user opts to exit.

The provided menu-driven interface simplifies user navigation and enhances the overall user experience. Additionally, the code accounts for potential input issues, incorporating measures like cin. ignore () to handle newline characters.

While the code offers a solid foundation for an interactive Hospital management application.

Overall, this C++ implementation demonstrates a commendable balance between functionality, user-friendliness, and security in the context of a Hospital Management system.

5. Future Work

5.1. Enhanced User Authentication:

Login System:
Implement a robust login system with usernames and passwords for users to access the application or website securely.

5.2. Expanded Functionality:

Online Platform Integration:
Extend the rental system to an online platform allowing remote access and participation.
Integrate social logins or authentication methods for wider accessibility.

5.3. Security Measures:

Data Encryption:
Incorporate encryption algorithms to safeguard sensitive data like user details and profile information, ensuring privacy and security.

5.4. User Experience Enhancements:

Graphical User Interface (GUI):
Develop a user-friendly GUI to replace or complement the text-based interface for improved interaction and visual appeal.

5.5. Additional Features for Users:

History and Statistics:
Provide users with a feature to view their past bike bookings, including the amount of time and fare statistics.
These potential future enhancements aim to elevate the by focusing on security, user
experience, expanded functionality, and administrative capabilities. By implementing
these improvements, the program can become more user-friendly, secure, and versatile, catering to a wider audience and providing the best experience.

References

1. Thinking in C++, Bruce, Eckel, Pearson, Second edition, Volume 1, 2002.
2. Object-oriented programming in C++, Robert Lafore, Course Sams Publishing, Fourth edition,2001.
3. Lischner, Ray. STL Pocket Reference: Containers, Iterators, and Algorithms. " O'Reilly Media,Inc.", 2003.