Project Title:

Hospital Management System (HMS)

BY: SRIHARIHARAN
DATE 21/10/2024

AIM OF THE PROJECT

Project Description:

The Hospital Management System (HMS) is a comprehensive, integrated software solution that manages all aspects of a hospital's operations, from patient registration and billing to medical records and inventory management. The goal of this system is to provide a more efficient and error-free approach to hospital administration by automating various processes that are traditionally managed manually.

Objectives:

To provide seamless patient management and improve the patient experience.

To automate the hospital's administrative tasks to reduce manual effort.

To maintain accurate records of patients, treatments, and hospital resources.

To provide real-time access to medical records, reducing wait times for doctors and patients.

To generate reports for better financial and operational decision-making.

Scope:

This HMS project can be used by small clinics, medium-sized hospitals, and large healthcare facilities. It can also be scaled to accommodate additional features such as telemedicine, online consultation, and mobile app integration.

1. Class Definitions:

The code defines four main classes: `Patient`, `Doctor`, `Appointment`, and `Hospital`. Each class represents an entity in the hospital system, and they are interconnected.

Patient Class:

- **Attributes (Variables):**
  - patient_id: A unique identifier for the patient.
  - name: The patient's name.
  - age: The patient's age.
  - gender: The patient's gender.
  - medical_history: The patient's medical history.
- **Methods:**
  - __init__: This is the constructor, called when a Patient object is created. It initializes the patient's details.
  - get_details: This method returns a formatted string with the patient's details

```
class Patient:
    def __init__(self, patient_id, name, age, gender, medical_history):
        self.patient_id = patient_id
        self.name = name
        self.age = age
        self.gender = gender
        self.medical_history = medical_history

    def get_details(self):
        return f"Patient ID: {self.patient_id}, Name: {self.name}, Age: {self.age}, Gender: 
{self.gender}, Medical History: {self.medical_history}"
```

Doctor Class:

Attributes (Variables):

doctor_id: A unique identifier for the doctor.

name: The doctor's name.

specialty: The doctor's area of specialization (e.g., Cardiologist).

availability: The doctor's working hours or availability.

Methods:

__init__: Constructor for initializing the doctor's details.

get_details: Returns a formatted string with the doctor's details.

```
class Doctor:
    def __init__(self, doctor_id, name, specialty, availability):
        self.doctor_id = doctor_id
        self.name = name
        self.specialty = specialty
        self.availability = availability

    def get_details(self):
        return f"Doctor ID: {self.doctor_id}, Name: {self.name}, Specialty: 
{self.specialty}, Availability: {self.availability}"
```

Appointment Class:

- **Attributes (Variables):**
  - appointment_id: A unique identifier for the appointment.
  - patient: This links to a Patient object.
  - doctor: This links to a Doctor object.
  - appointment_time: The time of the appointment.
- **Methods:**
  - __init__: Constructor for initializing appointment details.
  - get_details: Returns a formatted string with the appointment's details, including the patient’s name, doctor's name, and the time.

```
class Appointment:
    def __init__(self, appointment_id, patient, doctor, appointment_time):
        self.appointment_id = appointment_id
        self.patient = patient
        self.doctor = doctor
        self.appointment_time = appointment_time

    def get_details(self):
        return f"Appointment ID: {self.appointment_id}, Patient: {self.patient.name}, 
Doctor: {self.doctor.name}, Time: {self.appointment_time}"
```

Hospital Class:

- **Attributes (Variables):**
    - name: The name of the hospital.
    - patients: A list to store the Patient objects.
    - doctors: A list to store the Doctor objects.
    - appointments: A list to store the Appointment objects.
- **Methods:**
    - __init__: Constructor that initializes the hospital's name and creates empty lists for patients, doctors, and appointments.
    - add_patient: Adds a Patient object to the list of patients.
    - add_doctor: Adds a Doctor object to the list of doctors.
    - schedule_appointment: Adds an Appointment object to the list of appointments.
    - show_patients: Loops through the list of patients and prints their details using the get_details method from the Patient class.
    - show_doctors: Loops through the list of doctors and prints their details using the get_details method from the Doctor class.
    - show_appointments: Loops through the list of appointments and prints their details using the get_details method from the Appointment class.

```
class Hospital:
    def __init__(self, name):
        self.name = name
        self.patients = []
        self.doctors = []
        self.appointments = []

    def add_patient(self, patient):
        self.patients.append(patient)

    def add_doctor(self, doctor):
        self.doctors.append(doctor)

    def schedule_appointment(self, appointment):
        self.appointments.append(appointment)

    def show_patients(self):
        for patient in self.patients:
            print(patient.get_details())
    def show_doctors(self):
        for doctor in self.doctors:
            print(doctor.get_details())


    def show_appointments(self):
        for appointment in self.appointments:
            print(appointment.get_details())
```

Example Usage:

First, a Hospital object called hospital is created.

Then, two Patient objects (patient1 and patient2) are created and added to the hospital using the add_patient method.

Two Doctor objects (doctor1 and doctor2) are created and added to the hospital using the add_doctor method.

Two Appointment objects are created, associating a patient with a doctor, and scheduled in the hospital using the schedule_appointment method.

Finally, the details of all patients, doctors, and appointments are displayed using the show_patients, show_doctors, and show_appointments methods.

```
if __name__ == "__main__":
    hospital = Hospital("City Hospital")


    # Adding patients
    patient1 = Patient(1, "John Doe", 30, "Male", "No prior conditions")
    patient2 = Patient(2, "Jane Smith", 25, "Female", "Asthma")
    hospital.add_patient(patient1)
    hospital.add_patient(patient2)


    # Adding doctors
    doctor1 = Doctor(101, "Dr. Emily Brown", "Cardiologist", "9am - 5pm")
    doctor2 = Doctor(102, "Dr. James Wilson", "Dermatologist", "10am - 6pm")
    hospital.add_doctor(doctor1)
    hospital.add_doctor(doctor2)

    # Scheduling appointments
    appointment1 = Appointment(1, patient1, doctor1, "10:30 AM")
    appointment2 = Appointment(2, patient2, doctor2, "11:00 AM")
    hospital.schedule_appointment(appointment1)
    hospital.schedule_appointment(appointment2)

    # Display data
    print("\nPatients in the Hospital:")
    hospital.show_patients()

    print("\nDoctors in the Hospital:")
    hospital.show_doctors()

    print("\nScheduled Appointments:")
    hospital.show_appointments()
```

Expected Outcome:

A fully functional, user-friendly Hospital Management System that enhances hospital efficiency, improves patient care, and ensures secure data handling**.**

Summary of Concepts:

1) Classes and Objects: This code makes use of object-oriented programming (OOP). Each entity (Patient, Doctor, Appointment, Hospital) is represented by a class, and specific instances are created (objects).
2) Encapsulation: Information about patients, doctors, and appointments is encapsulated inside their respective classes, accessed through methods.
3) Lists: The hospital maintains lists of patients, doctors, and appointments, which can grow dynamically.
4) Composition: The Appointment class uses objects of the Patient and Doctor classes, showing a relationship between different objects.