

CIS 550 – DATABASE DESIGN

SPRING 2024

FINAL PROJECT

**HOSPITAL MANAGEMENT
SYSTEM**

GROUP- 32

Presentation Date- 20th April 2024

Professor:

Name: Dr. Ashok Patel

Office: Dion 302B

Email: apatel38@umassd.edu

Group Members:

R.No	Name	Student ID
71	Jai Saxena	02129249
61	Mayank Raj	02135041

PART-1

INTRODUCTION

The Hospital Management System (HMS) employs a relational schema to streamline operations across various entities such as Patients, Doctors, Appointments, Medical Records, Bills, Staff, and Inventory. Each entity is defined with unique identifiers and specific attributes: Patients are recorded with details such as ID, name, date of birth, address, phone number, and insurance information. Doctors are cataloged by ID, name, specialization, and schedule. The system enhances healthcare delivery by linking these entities through well-defined relationships. For instance, the one-to-many relationship between Patients and Appointments allows multiple appointments per patient, facilitating efficient scheduling. Similarly, Doctors and Appointments are linked to optimize time management and patient care continuity.

Further, each patient's multiple Medical Records ensure comprehensive tracking of health history, crucial for ongoing treatment strategies. Bills associated with each patient are managed to streamline financial processes, enhancing transaction efficiency. The relational dynamics extend to staff and doctors, where multiple staff members, like nurses and administrative workers, support various doctors, indicating a versatile and interconnected workforce. Additionally, the system manages Inventory by tracking items, quantities, and reorder levels, which is critical for the operational needs of medical facilities. This structured data organization not only supports day-to-day administrative and medical tasks but also contributes to long-term health management and operational planning in hospitals.

APPLICATION REQUIREMENTS/ USE- CASE

The planned Hospital Management System (HMS) utilizes a database-driven application to manage and automate key hospital operations efficiently and effectively. This system is designed to enhance operational efficiency and improve the user experience by integrating core functionalities into a cohesive platform.

- 1. Patient Management:** The system enables patient registration, updates to patient profiles, and the management of their medical histories. By using a relational database, the application can quickly retrieve and update patient information, ensuring that medical personnel have access to up-to-date data. This aids in delivering personalized patient care and maintaining accurate health records.
- 2. Appointment Scheduling:** This functionality supports seamless booking, rescheduling, and cancellation of appointments. The database stores appointment details, including patient and doctor IDs, which helps in managing and optimizing doctor schedules and reducing patient wait times. The system can also send automated reminders to both doctors and patients, improving the efficiency of the scheduling process.

- 3. Medical Records Management:** After each consultation, the system allows for the creation and secure access to detailed medical records. These records are stored in the database with restricted access based on user roles, ensuring confidentiality and compliance with healthcare regulations. Medical staff can access these records as needed, providing continuity in patient care, and facilitating better clinical decisions.
- 4. Billing and Insurance Processing:** The HMS automates the generation of bills for medical services and manages insurance claims processing. This module uses the database to track all billing transactions linked to patient and insurance records, streamlining financial operations and ensuring accurate billing and timely insurance processing.
- 5. Staff and Schedule Management:** This functionality manages detailed records of hospital staff, including their roles, departments, and schedules. The database system enables HR managers to allocate and manage resources efficiently, ensuring that there are adequate staff members available for different shifts and specialties.
- 6. Inventory Management:** The inventory management module tracks medical supplies and medications, monitoring stock levels, reorder points, and expiration dates. This helps in maintaining optimal stock levels, avoiding overstock and outages, and ensuring that the necessary medical supplies are always available for patient care.

By leveraging modern database technologies, the HMS ensures that all these functionalities are integrated into a single, user-friendly platform. This not only enhances the efficiency of hospital operations but also significantly improves the quality of patient care. The database-driven approach allows for scalability and flexibility, making it easier to update the system as healthcare practices evolve and new needs emerge.

RELATIONAL SCHEMA-

ENTITES USED IN THE HOTEL MANGEMENT SYSTEM DATABASE:

1. PATIENTS
2. DOCTORS
3. APPOINTMENTS
4. MEDICAL RECORDS
5. BILLS
6. STAFF
7. INVENTORY

Attributes used for Entities.

1. Patients

- PatientID (PK)
- Name
- DOB (Date of Birth)
- Address
- Phone
- InsuranceInfo

2. Doctors

- DoctorID (PK)
- Name
- Specialization
- Schedule

3. Appointments

- AppointmentID (PK)
- PatientID (FK to Patients)
- DoctorID (FK to Doctors)
- Date
- Time
- Purpose

4. Medical Records

- RecordID (PK)
- PatientID (FK to Patients)
- Visit Date
- Diagnosis
- Treatment

5. Bills

- BillID (PK)
- PatientID (FK to Patients)
- Date
- Amount
- Status (e.g., Paid, Unpaid, Pending Insurance)

6. Staff

- StaffID (PK)
- Name
- Role
- Department

7. Inventory

- temID (PK)
- Name
- Quantity
- ReorderLevel

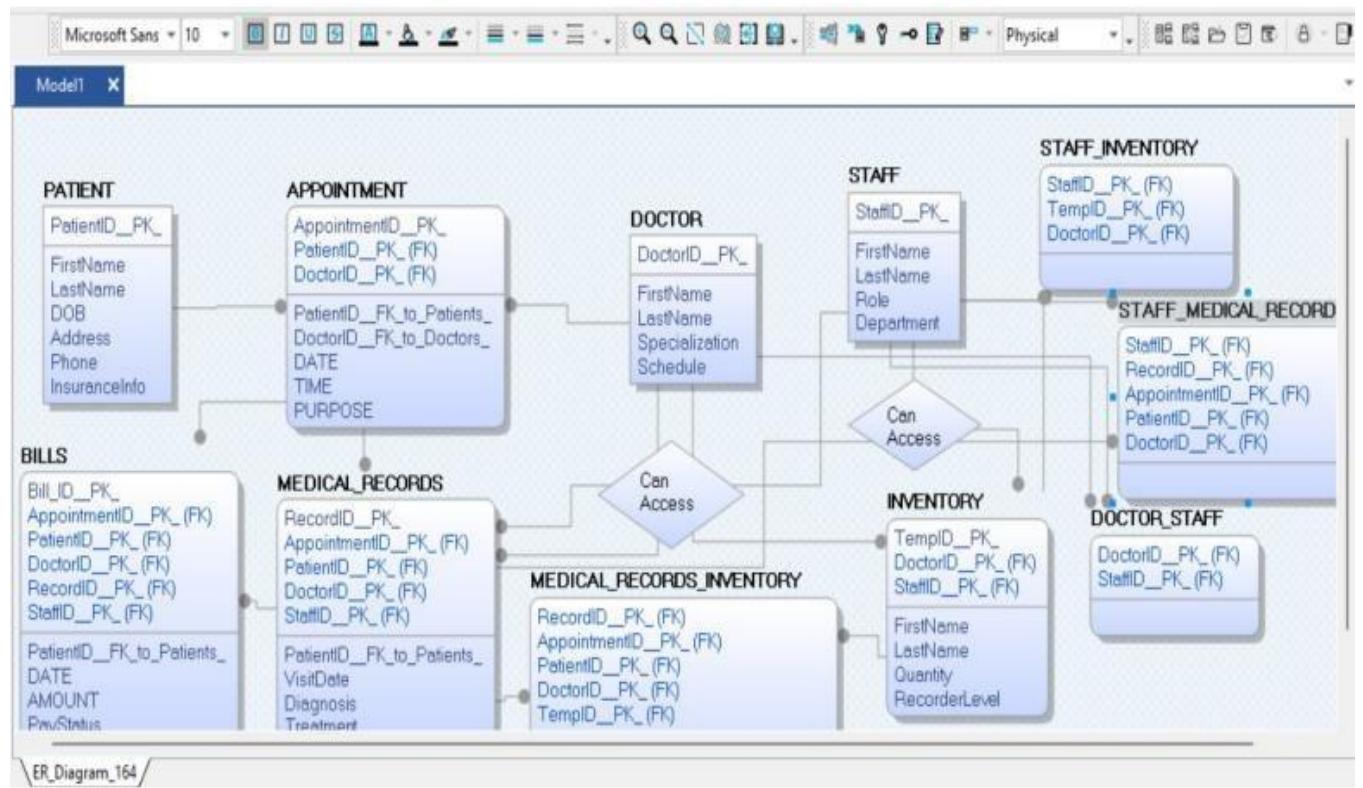
RELATIONSHIP BETWEEN ENTITIES IN THIS HOTEL MANAGEMENT SYSTEM DATABASE

In the Hospital Management System, the relationships between entities ensure efficient dataorganization and functionality:

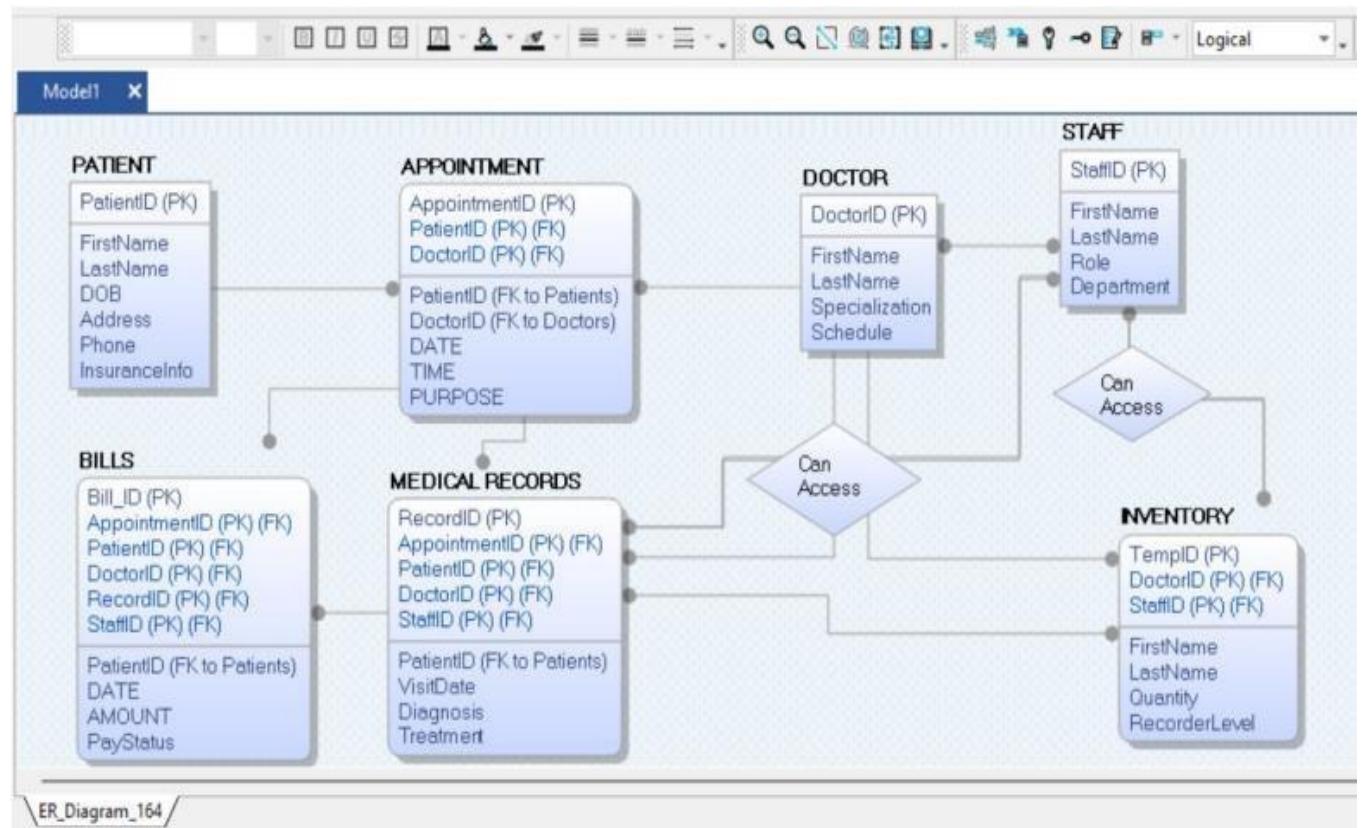
1. Patients and Appointments (One-to-Many): A patient can have multiple appointments,organize schedules, and facilitate healthcare management for individuals.
2. Doctors and Appointments (One-to-Many): Each doctor is linked to many appointments,streamlining doctor schedules and patient care.
3. Patients and Medical Records (One-to-Many): Multiple medical records are maintained foreach patient, centralizing health history for better care continuity.
4. Patients and Bills (One-to-Many): Patients generate multiple bills over time, simplifyingfinancial transactions and tracking.
5. 'M' Staff could consist of 'N' many Doctors.
6. 'M' Doctors could have 'N' many staff. (e.g. compounder, ward boy, etc.)
7. '1' Doctor (based on specialization) could access many 'N' Medical Records.
8. 'M' Staff could access many 'N' Medical Records.
9. '1" Medical Record could have many 'N' Bills.
10. 'M' Inventory could access many 'N' Medical Records.
11. 'M' Medical Records could access many 'N' Inventory.

These relationships enable the HMS to coordinate care, manage schedules, track health histories, and handle billing efficiently, ensuring comprehensive and integrated healthcare services. This schema covers the basic structure and relationships needed for a hospital management system, focusing on patient care, appointments, medical records, billing, staff management, andinventory tracking. It lays the groundwork for developing a database that supports the system'skey functionalities.

PHYSICAL MODEL



LOGICAL MODEL



PART-2: DATABASE CODES

Below is the database code we used to create all the entities and their structures in the database. I explicitly created the database and its entities to reduce the amount of code at the backend.

The screenshot shows the MySQL Workbench interface with two databases selected in the Navigator:

- hospitalmanagement**: Contains tables for appointments, bills, doctor_staff, doctors, inventory, medicalrecords, patients, and staff.
- sakila**: Contains tables for actors, address, category, country, customer, film, film_actor, film_category, inventory, language, payment, rental, and staff.

The SQL Editor displays the creation scripts for these entities:

```
-- Creating the database
CREATE DATABASE IF NOT EXISTS HospitalManagement;
USE HospitalManagement;

-- Creating the table for Patients
CREATE TABLE Patients (
    PatientID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Specialization VARCHAR(100) NOT NULL,
    Schedule TEXT NOT NULL
);

-- Creating the table for Doctors
CREATE TABLE Doctors (
    DoctorID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Specialization VARCHAR(100) NOT NULL,
    Schedule TEXT NOT NULL
);

-- Creating the table for Appointments
CREATE TABLE Appointments (
    AppointmentID INT AUTO_INCREMENT PRIMARY KEY,
    PatientID INT NOT NULL,
    DoctorID INT NOT NULL,
    Date DATE NOT NULL,
    Time TIME NOT NULL,
    Purpose TEXT,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID) ON DELETE CASCADE,
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID) ON DELETE CASCADE
);

-- Creating the table for Medical Records
CREATE TABLE MedicalRecords (
    RecordID INT AUTO_INCREMENT PRIMARY KEY,
    PatientID INT NOT NULL,
    VisitDate DATE NOT NULL,
    Diagnosis TEXT NOT NULL,
    Treatment TEXT NOT NULL,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID) ON DELETE CASCADE
);

-- Creating the table for Bills
CREATE TABLE Bills (
    BillID INT AUTO_INCREMENT PRIMARY KEY,
    PatientID INT NOT NULL,
    Date DATE NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL,
    Status ENUM('Paid', 'Unpaid', 'Pending Insurance') DEFAULT 'Unpaid',
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID) ON DELETE CASCADE
);

-- Creating the table for Staff
CREATE TABLE Staff (
    StaffID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Role VARCHAR(100) NOT NULL,
    Department VARCHAR(100) NOT NULL
);
```

```

MySQL Workbench - Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator: HospitalManagement SQL File 3*
SCHEMAS
Filter objects
hospitalmanagement
Tables
appointments
bills
doctor_staff
doctors
inventory
medicalrecords
patients
staff
Views
Stored Procedures
Functions
sakila
sys
world
Administration Schemas
Information
Table: bills
Columns:
BillID int AI PK
PatientID int
Date date
Amount decimal(10,2)
Status varchar(50)
59     Role VARCHAR(100) NOT NULL,
60     Department VARCHAR(100) NOT NULL
61   );
62
63   -- Creating the table for Inventory
64 • CREATE TABLE Inventory (
65     ItemID INT AUTO_INCREMENT PRIMARY KEY,
66     Name VARCHAR(255) NOT NULL,
67     Quantity INT DEFAULT 0,
68     ReorderLevel INT NOT NULL
69   );
70
71   -- Creating additional tables to handle many-to-many relationships if necessary
72   -- For instance, a Doctor_Staff link table to associate doctors with their staff members if needed
73 • CREATE TABLE Doctor_Staff (
74     DoctorID INT NOT NULL,
75     StaffID INT NOT NULL,
76     PRIMARY KEY (DoctorID, StaffID),
77     FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID) ON DELETE CASCADE,
78     FOREIGN KEY (StaffID) REFERENCES Staff(StaffID) ON DELETE CASCADE
79   );
80
81   -- Indexes to improve search performance
82 • CREATE INDEX idx_patients_name ON Patients(Name);
83 • CREATE INDEX idx_doctors_specialization ON Doctors(Specialization);
84 • CREATE INDEX idx_appointments_date ON Appointments(Date);
85 • CREATE INDEX idx_bills_status ON Bills(Status);
86

```

Below are some of the **CRUD** operations we did during the presentation and its evidence that it got saved in the database as intended to work.

As in the presentation, I added **PatientID 9 as Roman** with all the mentioned detailed on the database as can be seen in the screenshot below. The **PatientID is unique**.

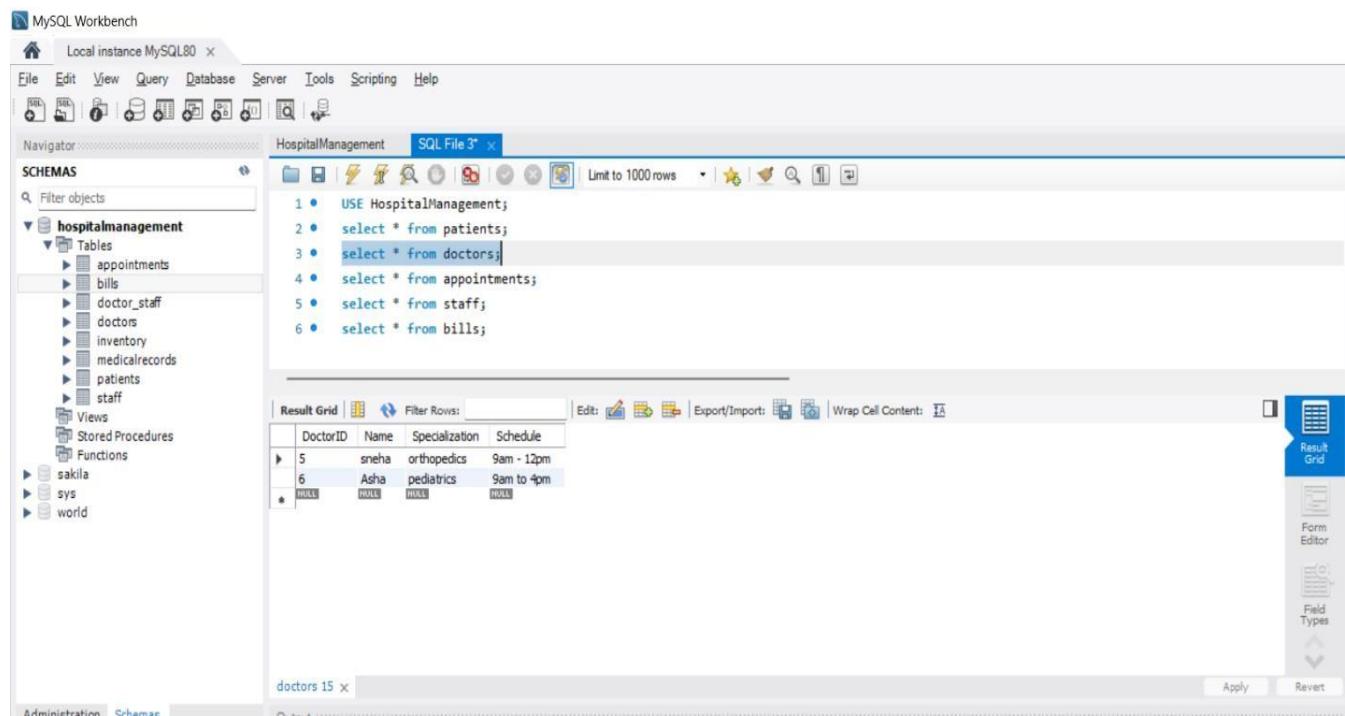
```

MySQL Workbench - Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator: HospitalManagement SQL File 3*
SCHEMAS
Filter objects
hospitalmanagement
Tables
appointments
bills
doctor_staff
doctors
inventory
medicalrecords
patients
staff
Views
Stored Procedures
Functions
sakila
sys
world
HospitalManagement SQL File 3* | Limit to 1000 rows
1 • USE HospitalManagement;
2 • select * from patients;
3 • select * from doctors;
4 • select * from appointments;
5 • select * from staff;
6 • select * from bills;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |
PatientID Name DOB Address Phone InsuranceInfo
7 mayank 2024-04-20 285 old westportroad 7742586586 123
8 raj 2024-04-19 285 old westportroad 7742586586 234
9 Roman 2024-04-10 285 old westportroad 7743657257 456
NULL NULL NULL NULL NULL NULL
patients 14 x Apply Revert

```

As seen in the screenshot, I added the **DoctorID** as **6** with name **Asha** during the live presentation in class. **DoctorID** is unique for every doctor.



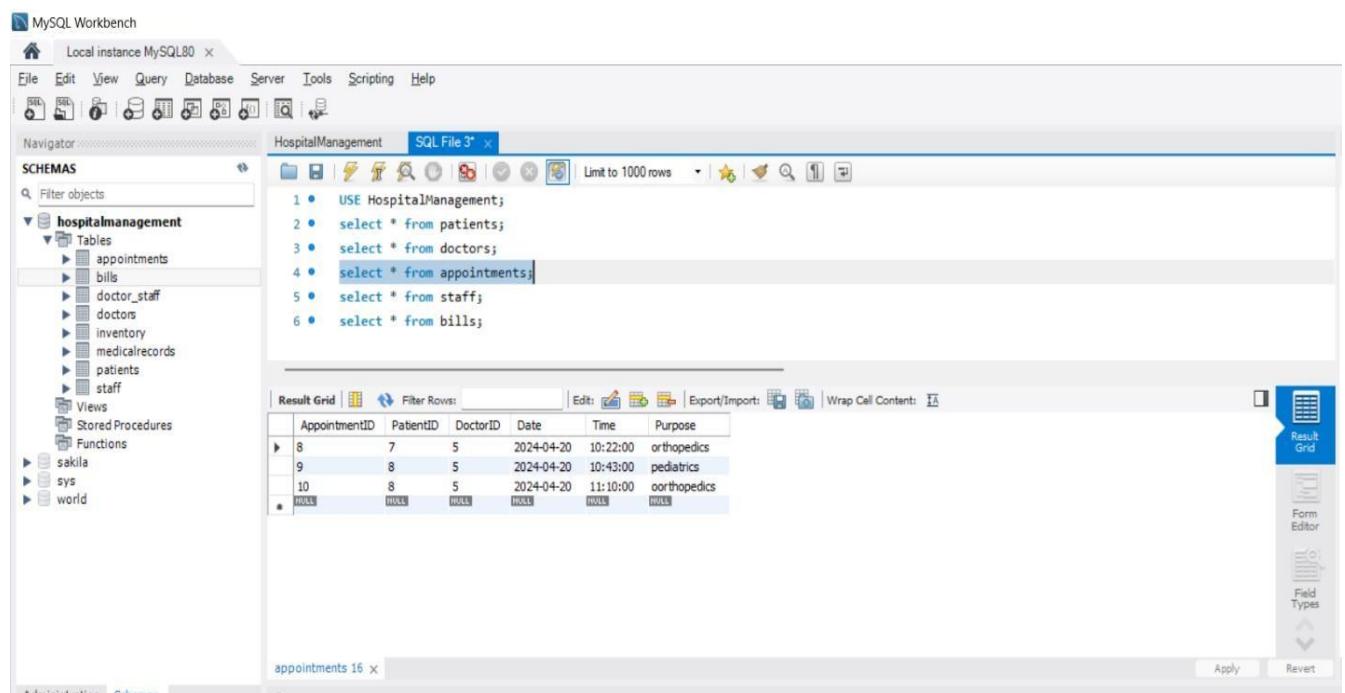
```

USE HospitalManagement;
select * from patients;
select * from doctors;
select * from appointments;
select * from staff;
select * from bills;

```

DoctorID	Name	Specialization	Schedule
5	sneha	orthopedics	9am - 12pm
6	Asha	pediatrics	9am to 4pm
*	HULL	HULL	HULL

The **PatientID** and **DoctorId** are unique as can be seen in the screenshots attached with the database code, Here as shown in the live demonstration in the class the DoctorID and **PatientID** needs to be inserted with other details needed to make an appointment. This can be seen below as well.



```

USE HospitalManagement;
select * from patients;
select * from doctors;
select * from appointments;
select * from staff;
select * from bills;

```

AppointmentID	PatientID	DoctorID	Date	Time	Purpose
8	7	5	2024-04-20	10:22:00	orthopedics
9	8	5	2024-04-20	10:43:00	pediatrics
10	8	5	2024-04-20	11:10:00	oorthopedics
*	HULL	HULL	HULL	HULL	HULL

PART-3 FRONTEND INTERFACE

As can be seen in the screenshot below, **patient 9 named Roman** is added or **CREATED** in the patients list with all the details as mentioned above in the database section.

The screenshot shows a web browser window with multiple tabs open. The active tab is titled 'Patients' and displays a table of patient information. The table has columns for ID, Name, DOB, Address, Phone, Insurance Info, and Actions. There are three rows of data: one for 'mayank' (ID 7), one for 'raj' (ID 8), and one for 'Roman' (ID 9). The 'Actions' column for each row contains 'Edit' and 'Delete' links. Below the table, there is a form titled 'Add New Patient' with fields for Name, DOB, Address, Phone, and Insurance Info, followed by a 'Submit' button.

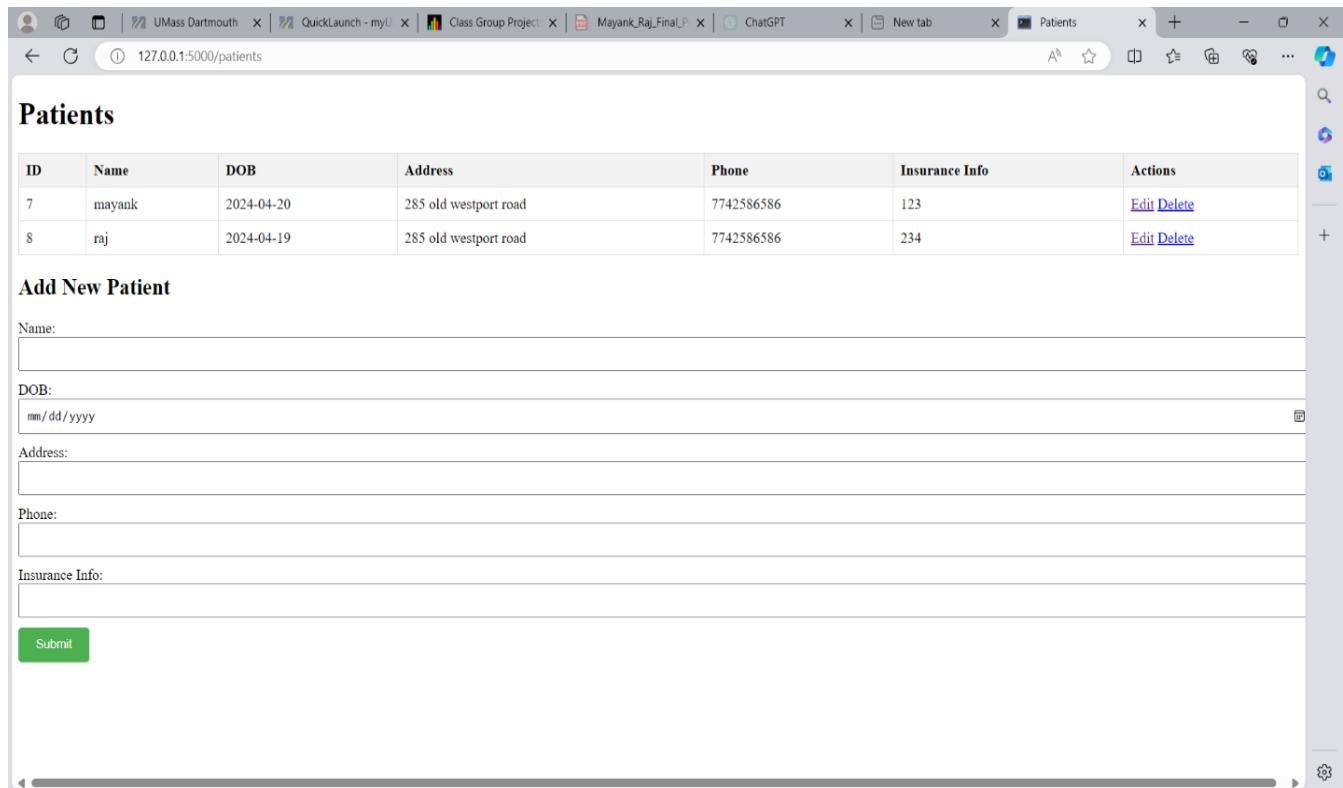
ID	Name	DOB	Address	Phone	Insurance Info	Actions
7	mayank	2024-04-20	285 old westport road	7742586586	123	Edit Delete
8	raj	2024-04-19	285 old westport road	7742586586	234	Edit Delete
9	Roman	2024-04-10	285 old westport road	7743657257	456	Edit Delete

As seen in the screenshot below, I **UPDATED** patient **9** by adding the last name to his **firstname** seen in the above screenshot.

The screenshot shows a web browser window with multiple tabs open. The active tab is titled 'Patients' and displays a table of patient information. The table has columns for ID, Name, DOB, Address, Phone, Insurance Info, and Actions. There are three rows of data: one for 'mayank' (ID 7), one for 'raj' (ID 8), and one for 'Roman Louis' (ID 9). The 'Actions' column for each row contains 'Edit' and 'Delete' links. Below the table, there is a form titled 'Add New Patient' with fields for Name, DOB, Address, Phone, and Insurance Info, followed by a 'Submit' button.

ID	Name	DOB	Address	Phone	Insurance Info	Actions
7	mayank	2024-04-20	285 old westport road	7742586586	123	Edit Delete
8	raj	2024-04-19	285 old westport road	7742586586	234	Edit Delete
9	Roman Louis	2024-04-10	285 old westport road	7743657257	456	Edit Delete

As seen in the below screenshot, I **DELETED Patient 9** from the patient list we created during the live demonstration in the class presentations.



ID	Name	DOB	Address	Phone	Insurance Info	Actions
7	mayank	2024-04-20	285 old westport road	7742586586	123	Edit Delete
8	raj	2024-04-19	285 old westport road	7742586586	234	Edit Delete

Add New Patient

Name:

DOB: mm/dd/yyyy

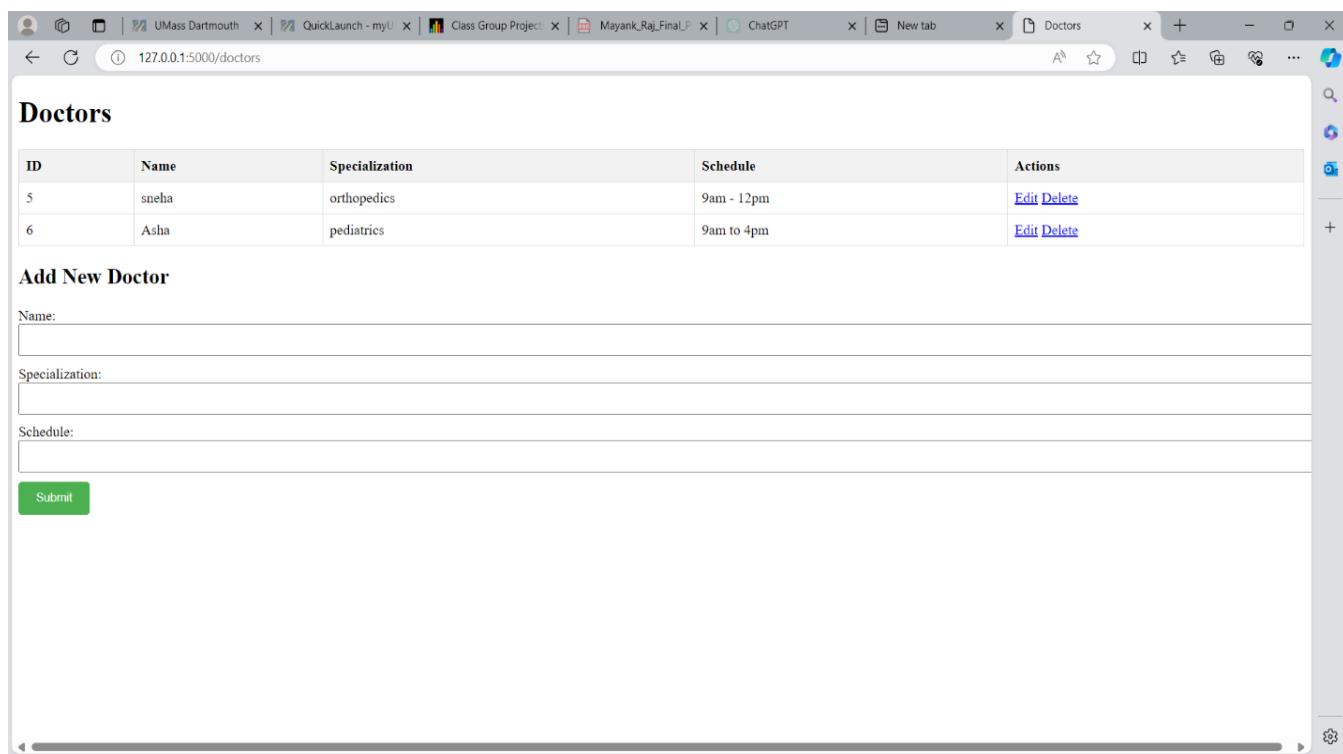
Address:

Phone:

Insurance Info:

Submit

Similarly, the screenshot below shows the Doctors List as per the database and the names we created in the live demonstration.



ID	Name	Specialization	Schedule	Actions
5	sneha	orthopedics	9am - 12pm	Edit Delete
6	Asha	pediatrics	9am to 4pm	Edit Delete

Add New Doctor

Name:

Specialization:

Schedule:

Submit

Below is the **Appointments** page with the mentioned **PatientID** and **DoctorID** which are unique and their scheduled appointments.

The screenshot shows a web browser window with the URL `127.0.0.1:5000/appointments`. The page title is "Appointments". It displays a table of scheduled appointments:

Patient ID	Doctor ID	Date	Time	Purpose	Action
7	5	2024-04-20	10:22:00	orthopedics	Edit Delete
8	5	2024-04-20	10:43:00	pediatrics	Edit Delete
8	5	2024-04-20	11:10:00	oorthopedics	Edit Delete

Below the table is a form titled "Add New Appointment" with fields for Patient ID, Doctor ID, Date, Time, and Purpose, along with a "Submit" button.

Below is the **Bills** Page as can be seen in the screenshot below. It also shows the bills currently in the database as created during the live demonstration in the class.

The screenshot shows a web browser window with the URL `127.0.0.1:5000/bills`. The page title is "Bills". It displays a table of bills:

Patient ID	Date	Amount	Status	Action
8	2024-04-20	300.00	unpaid	Edit Delete

Below the table is a form titled "Add New Bill" with fields for Patient ID, Date, Amount, and Status, along with a "Submit" button.

Below screenshot shows the **Staff Management** page and can see a staff added to the list that shows the exact copy of the data stored in the database.

The screenshot shows a web browser window with the title "Staff Management". A success message at the top states "• Staff member added successfully". Below it, a section titled "Add New Staff Member" contains input fields for Name, Role, and Department, along with a "Add Staff Member" button. A table titled "Current Staff" displays one row of data:

ID	Name	Role	Department	Actions
3	mayank raj	nurse	orthopedics	Edit Delete

Below screenshot shows the **Manage Inventory** page and can see an item added to the list that shows the exact copy of the data stored in the database.

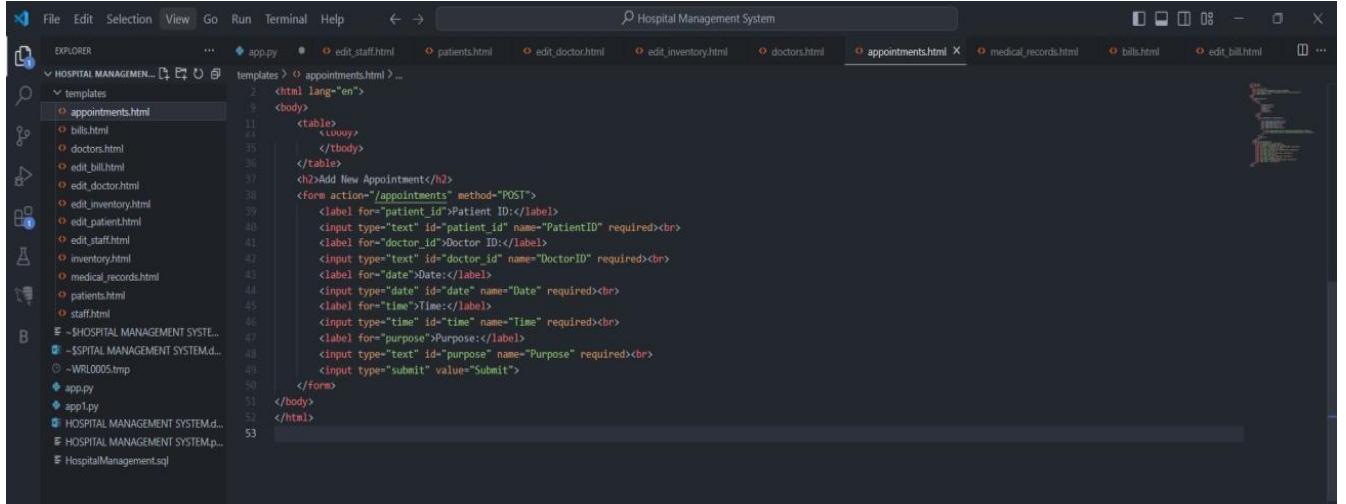
The screenshot shows a web browser window with the title "Manage Inventory". A success message at the top states "• Item added successfully". Below it, a section titled "Add New Inventory Item" contains input fields for Name, Quantity, and Reorder Level, along with an "Add" button. A table titled "Manage Inventory" displays one row of data:

ID	Name	Quantity	Reorder Level	Actions
2	mayank	1	12	Edit Delete

PART- 4: SOURCE CODES BACKEND AND FRONTEND CODES

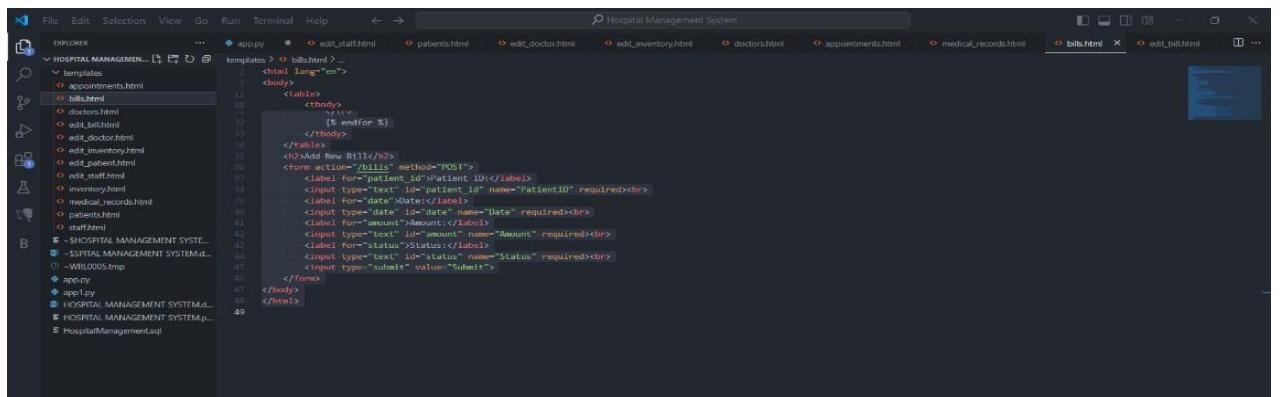
FRONTEND CODES

Below is the screenshot of the **appointment page** HTML codes that makes CRUD operations on the managing appointment side of the HMS.

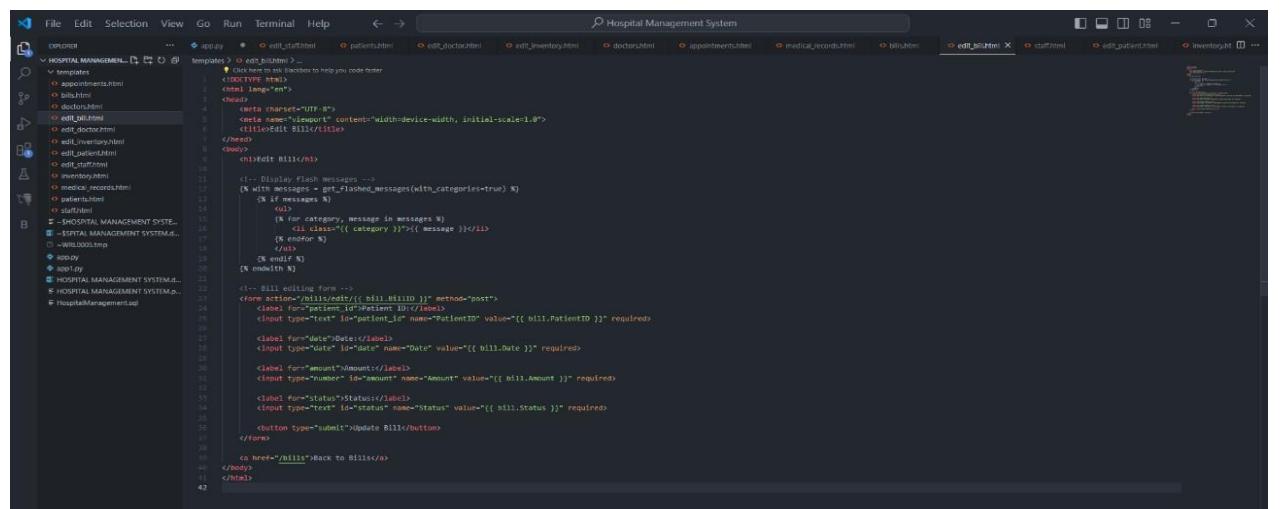


```
<html lang="en">
  <body>
    <table>
      <tr><td>Add New Appointment</td></tr>
    </table>
    <form action="/appointments" method="POST">
      <label for="patient_id">Patient ID:</label>
      <input type="text" id="patient_id" name="PatientID" required><br>
      <label for="doctor_id">Doctor ID:</label>
      <input type="text" id="doctor_id" name="DoctorID" required><br>
      <label for="date">Date:</label>
      <input type="date" id="date" name="Date" required><br>
      <label for="time">Time:</label>
      <input type="time" id="time" name="Time" required><br>
      <label for="purpose">Purpose:</label>
      <input type="text" id="purpose" name="Purpose" required><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Below is the screenshot of the **Bills page** and **Edit Bill** HTML codes that makes CRUD operations on the managing bills side of the HMS.



```
<table>
  <thead>
    <tr><th>Bill ID</th><th>Patient ID</th><th>Date</th><th>Amount</th><th>Status</th><th>Actions</th></tr>
  </thead>
  <tbody>
    <tr><td>1</td><td>P001</td><td>2023-10-01</td><td>1000</td><td>Paid</td><td><a href="#">Edit</a> <a href="#">Delete</a></td></tr>
  </tbody>
</table>
<h2>Add New Bill</h2>
<form action="/bills" method="POST">
  <label for="patient_id">Patient ID:</label>
  <input type="text" id="patient_id" name="PatientID" required><br>
  <label for="date">Date:</label>
  <input type="date" id="date" name="Date" required><br>
  <label for="amount">Amount:</label>
  <input type="text" id="amount" name="Amount" required><br>
  <label for="status">Status:</label>
  <input type="text" id="status" name="Status" required><br>
  <input type="submit" value="Submit">
</form>
```



```
<!-- Flash message -->
<% with messages = get_flashed_messages(with_categories=True) %>
<% if messages %>
  <ul>
    <% for category, message in messages %>
      <li class="category-{{ category }}">{{ message }}</li>
    <% endfor %>
  <% endif %>
<% endif %>

<!-- Bill editing form -->
<form action="/bills/edit/{{ bill.BillID }}" method="post">
  <label for="patient_id">Patient ID:</label>
  <input type="text" id="patient_id" name="PatientID" value="{{ bill.PatientID }}" required><br>

  <label for="date">Date:</label>
  <input type="date" id="date" name="Date" value="{{ bill.Date }}" required><br>

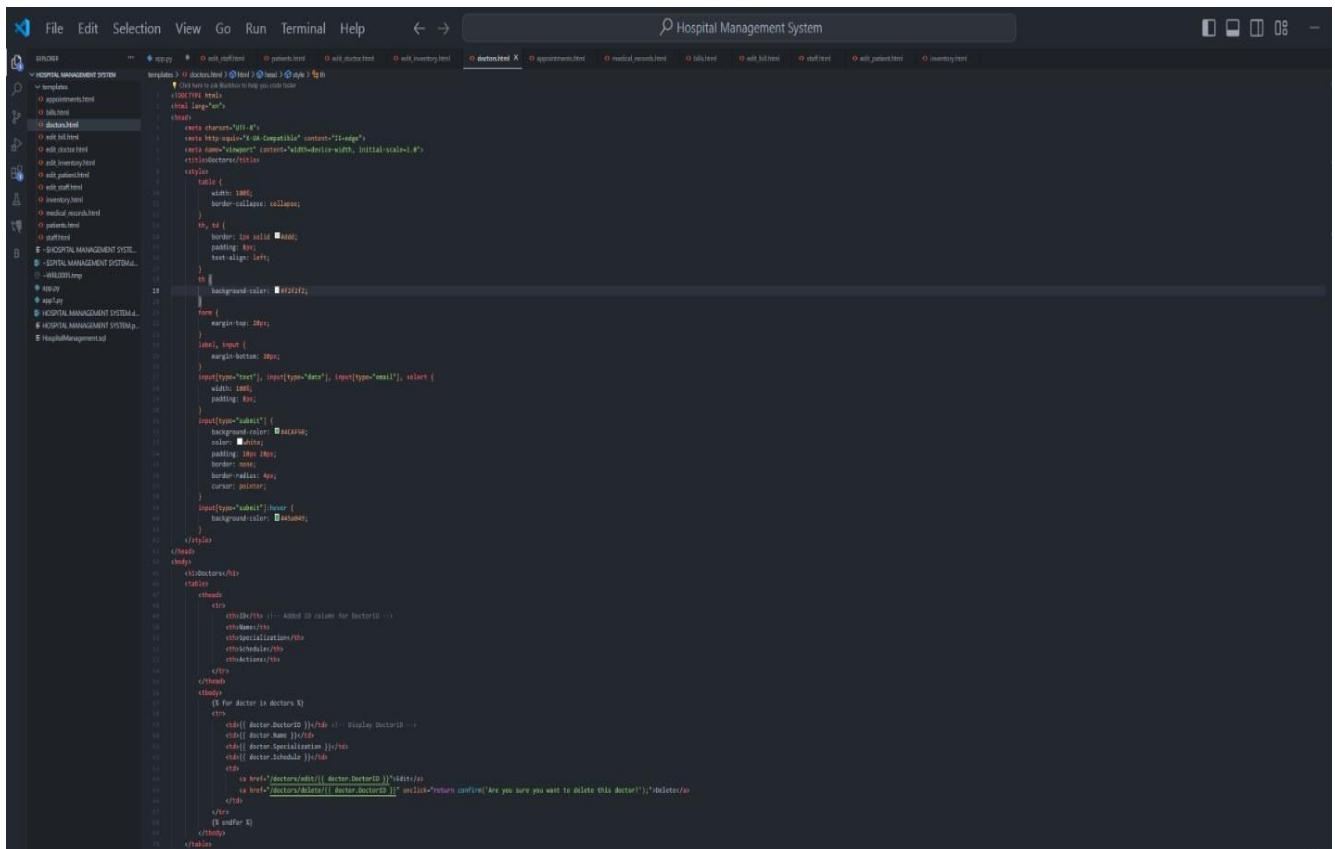
  <label for="amount">Amount:</label>
  <input type="number" id="amount" name="Amount" value="{{ bill.Amount }}" required><br>

  <label for="status">Status:</label>
  <input type="text" id="status" name="Status" value="{{ bill.Status }}" required><br>

  <button type="submit">Update Bill</button>
</form>

<a href="/bills">Back to Bills</a>
</body>
</html>
```

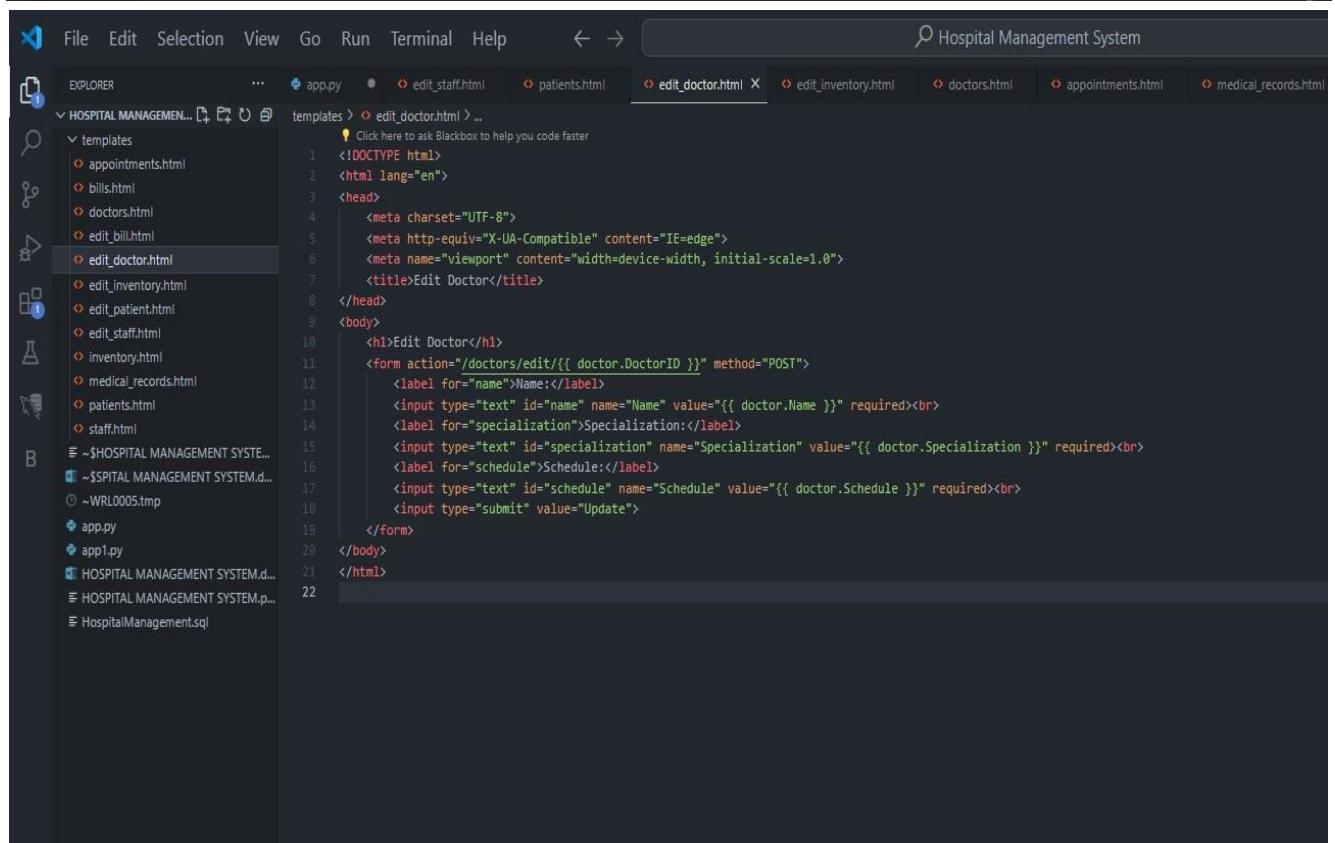
Below is the screenshot of the **Doctors page** and **Edit Doctors** HTML codes that makes CRUDoperations on the managing doctors side of the HMS.



```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hospital Management System</title>
</head>
<body>
    <table border="1">
        <thead>
            <tr>
                <th>Doctor ID</th>
                <th>Name</th>
                <th>Specialization</th>
                <th>Schedule</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>101</td>
                <td>Dr. John Doe</td>
                <td>Orthopedist</td>
                <td>Monday, Wednesday, Friday</td>
            </tr>
            <tr>
                <td>102</td>
                <td>Dr. Jane Smith</td>
                <td>Neurologist</td>
                <td>Tuesday, Thursday, Saturday</td>
            </tr>
            <tr>
                <td>103</td>
                <td>Dr. Michael Green</td>
                <td>Cardiologist</td>
                <td>Wednesday, Friday, Sunday</td>
            </tr>
        </tbody>
    </table>
    <form action="/doctors/edit/{{ doctor.DoctorID }}" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="Name" value="{{ doctor.Name }}" required><br>
        <label for="specialization">Specialization:</label>
        <input type="text" id="specialization" name="Specialization" value="{{ doctor.Specialization }}" required><br>
        <label for="schedule">Schedule:</label>
        <input type="text" id="schedule" name="Schedule" value="{{ doctor.Schedule }}" required><br>
        <input type="submit" value="Update">
    </form>
    <div style="text-align: right; margin-top: 10px;">
        <a href="/doctors/{{ doctor.DoctorID }}?delete=1" style="color: red; text-decoration: none; font-weight: bold;">Delete</a>
    </div>
</body>
</html>

```

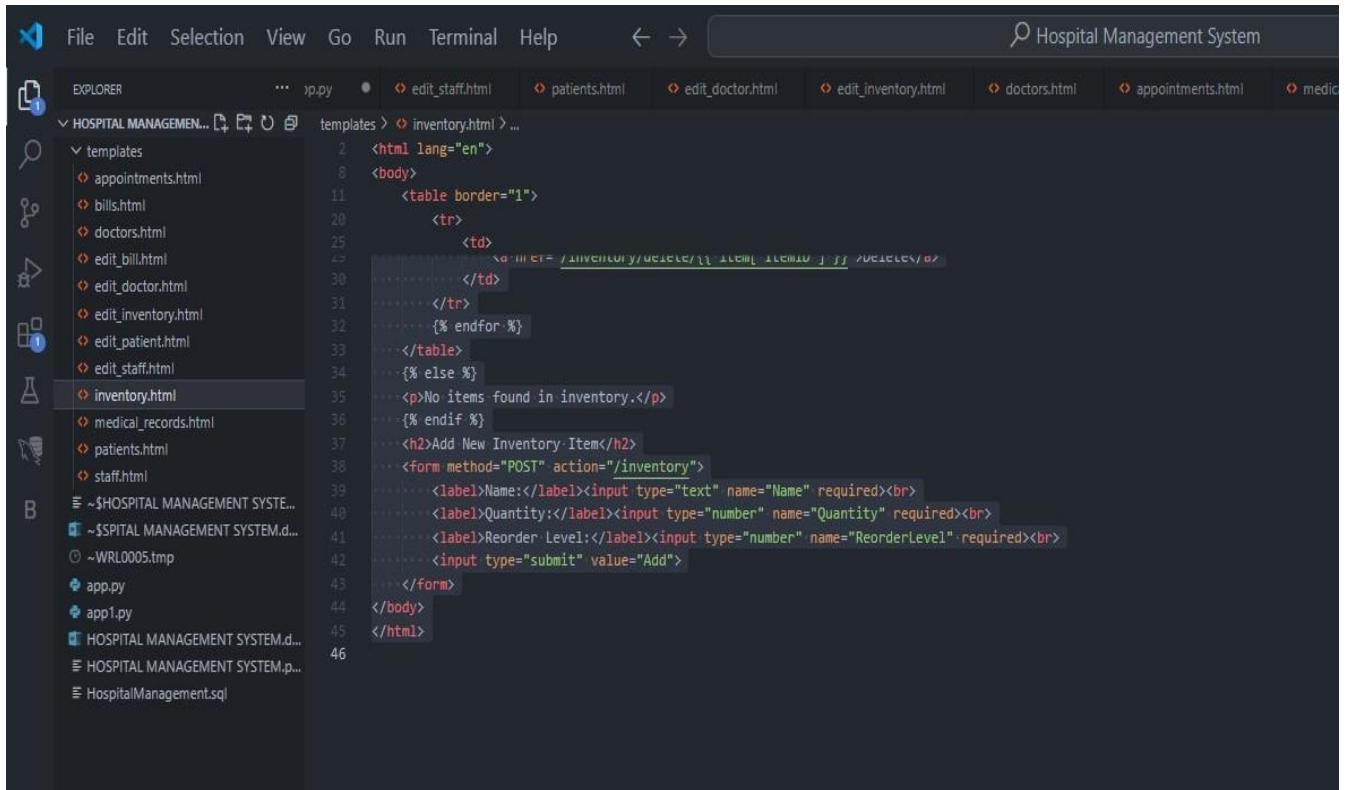


```

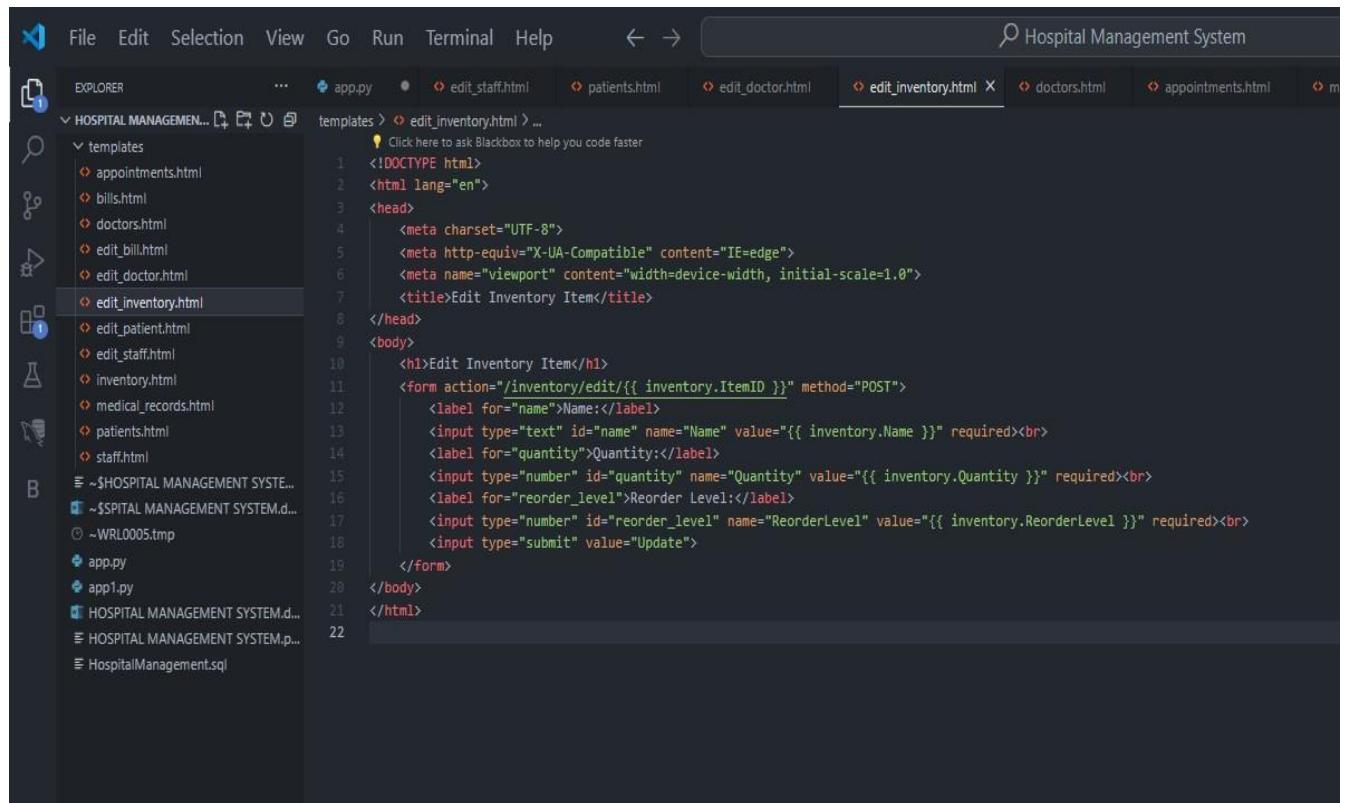
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit Doctor</title>
</head>
<body>
    <h1>Edit Doctor</h1>
    <form action="/doctors/edit/{{ doctor.DoctorID }}" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="Name" value="{{ doctor.Name }}" required><br>
        <label for="specialization">Specialization:</label>
        <input type="text" id="specialization" name="Specialization" value="{{ doctor.Specialization }}" required><br>
        <label for="schedule">Schedule:</label>
        <input type="text" id="schedule" name="Schedule" value="{{ doctor.Schedule }}" required><br>
        <input type="submit" value="Update">
    </form>
    <div style="text-align: center; margin-top: 10px;">
        <small>Are you sure you want to delete this doctor?</small>
        <a href="/doctors/{{ doctor.DoctorID }}?delete=1" style="color: red; text-decoration: none; font-weight: bold;">Delete</a>
    </div>
</body>
</html>

```

Below is the screenshot of the **Inventory and Edit Inventory** page HTML codes that makes CRUD operations on the managing inventory side of the HMS.

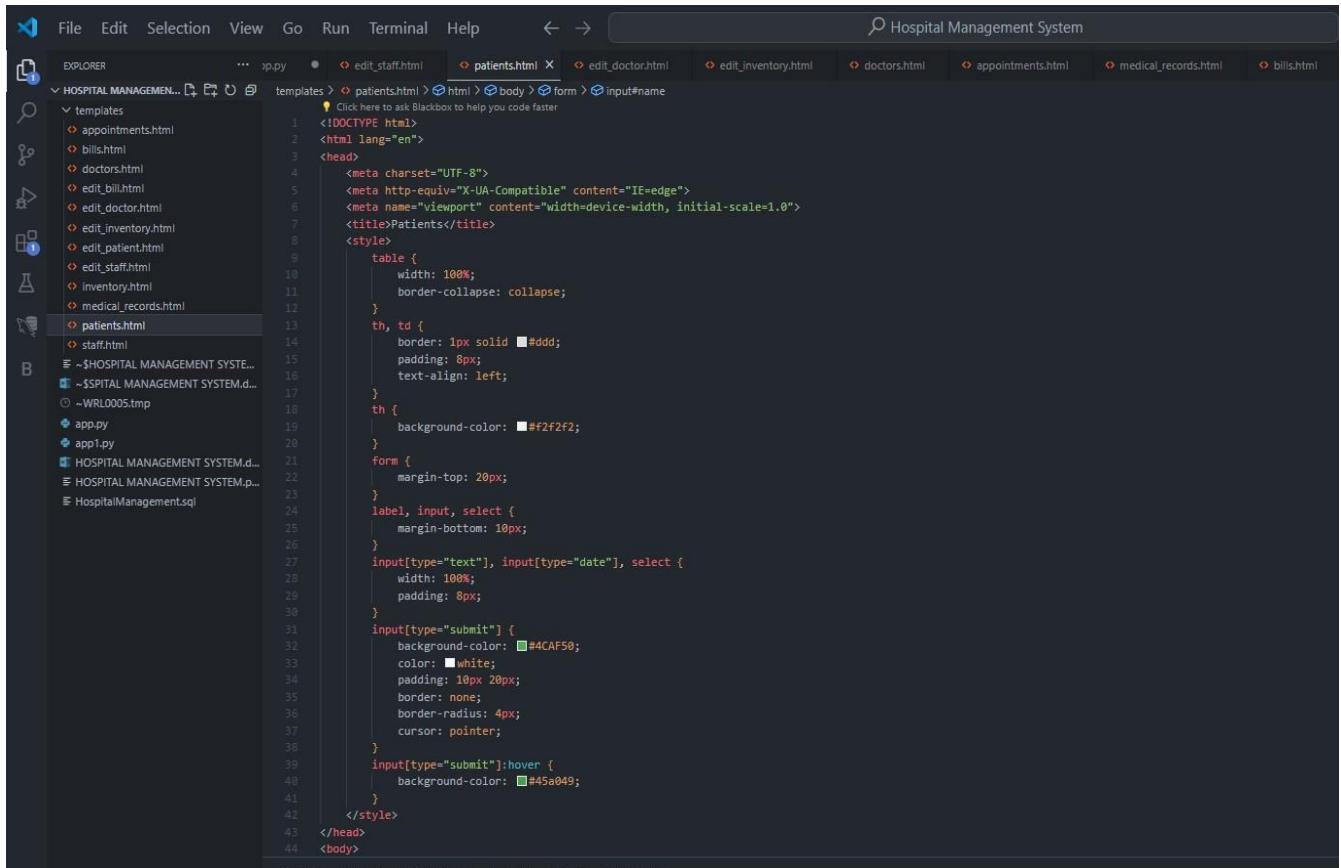


```
<html lang="en">
<body>
    <table border="1">
        <tr>
            <td>
                <a href="/Inventory/delete/{{ inventory.ItemID }}>Delete</a>
            </td>
        </tr>
        <% endfor %>
    </table>
    <% else %>
        <p>No items found in inventory.</p>
    <% endif %>
    <h2>Add New Inventory Item</h2>
    <form method="POST" action="/inventory">
        <label>Name:</label><input type="text" name="Name" required><br>
        <label>Quantity:</label><input type="number" name="Quantity" required><br>
        <label>Reorder Level:</label><input type="number" name="ReorderLevel" required><br>
        <input type="submit" value="Add">
    </form>
</body>
</html>
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit Inventory Item</title>
</head>
<body>
    <h1>Edit Inventory Item</h1>
    <form action="/Inventory/edit/{{ inventory.ItemID }}" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="Name" value="{{ inventory.Name }}" required><br>
        <label for="quantity">Quantity:</label>
        <input type="number" id="quantity" name="Quantity" value="{{ inventory.Quantity }}" required><br>
        <label for="reorder_level">Reorder Level:</label>
        <input type="number" id="reorder_level" name="ReorderLevel" value="{{ inventory.ReorderLevel }}" required><br>
        <input type="submit" value="Update">
    </form>
</body>
</html>
```

Below is the screenshot of the **Patient and edit Patients page** HTML codes that makes CRUD operations on the managing patients' side of the HMS.

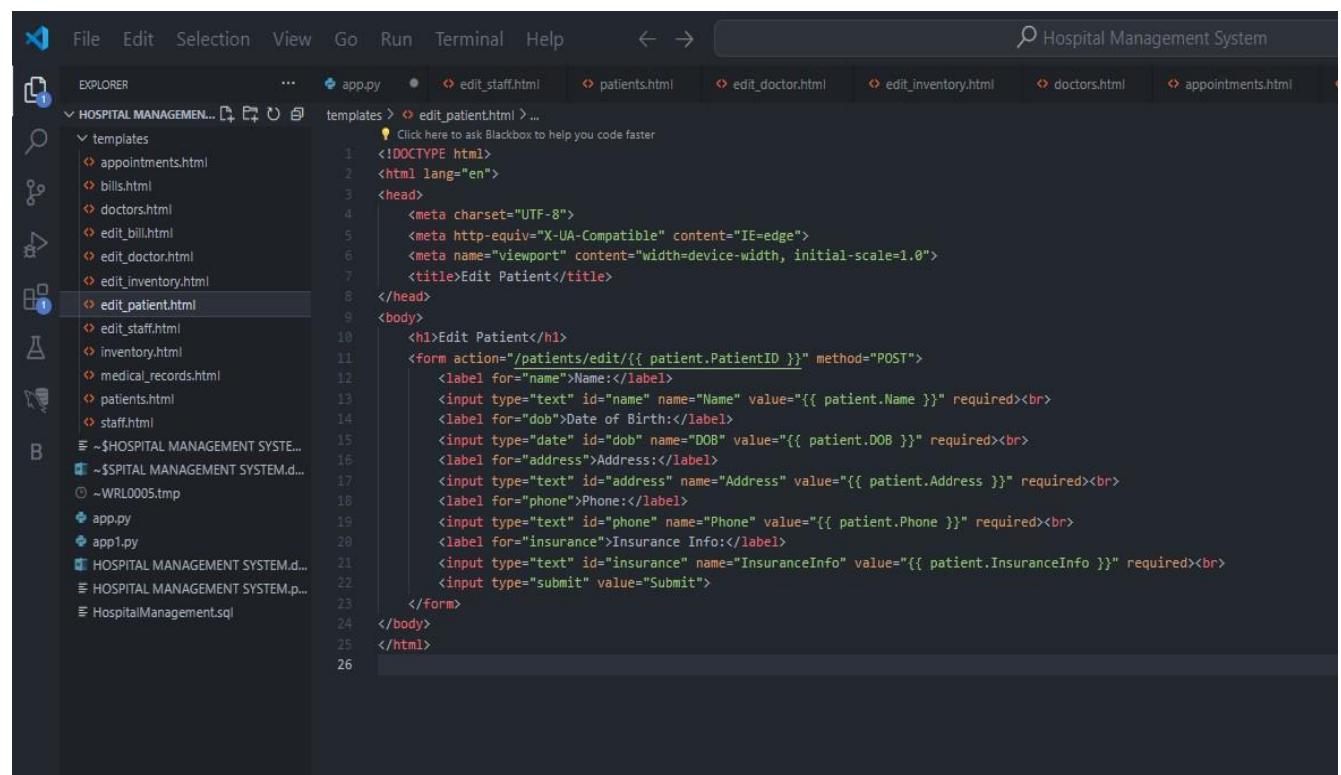


```

File Edit Selection View Go Run Terminal Help ← → Hospital Management System

EXPLORER File Edit Selection View Go Run Terminal Help Hospital Management System
patients.html
templates > patients.html > body > form > input#name
Click here to ask Blackbox to help you code faster
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Patients</title>
    <style>
        table {
            width: 100%;
            border-collapse: collapse;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
        form {
            margin-top: 20px;
        }
        label, input, select {
            margin-bottom: 10px;
        }
        input[type="text"], input[type="date"], select {
            width: 100%;
            padding: 8px;
        }
        input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
        input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>

```

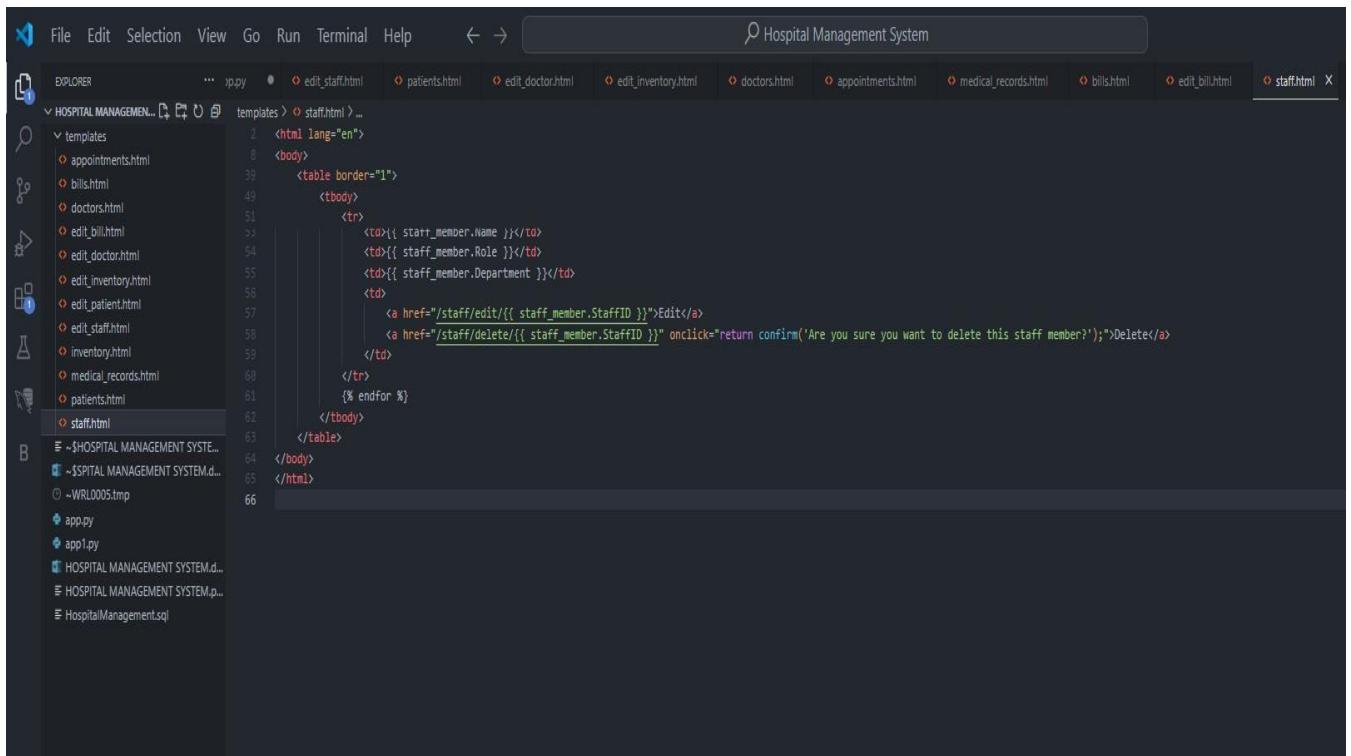


```

File Edit Selection View Go Run Terminal Help ← → Hospital Management System
edit_patient.html
templates > edit_patient.html > ...
Click here to ask Blackbox to help you code faster
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit Patient</title>
</head>
<body>
    <h1>Edit Patient</h1>
    <form action="/patients/edit/{{ patient.PatientID }}" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="Name" value="{{ patient.Name }}" required><br>
        <label for="dob">Date of Birth:</label>
        <input type="date" id="dob" name="DOB" value="{{ patient.DOB }}" required><br>
        <label for="address">Address:</label>
        <input type="text" id="address" name="Address" value="{{ patient.Address }}" required><br>
        <label for="phone">Phone:</label>
        <input type="text" id="phone" name="Phone" value="{{ patient.Phone }}" required><br>
        <label for="insurance">Insurance Info:</label>
        <input type="text" id="insurance" name="InsuranceInfo" value="{{ patient.InsuranceInfo }}" required><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>

```

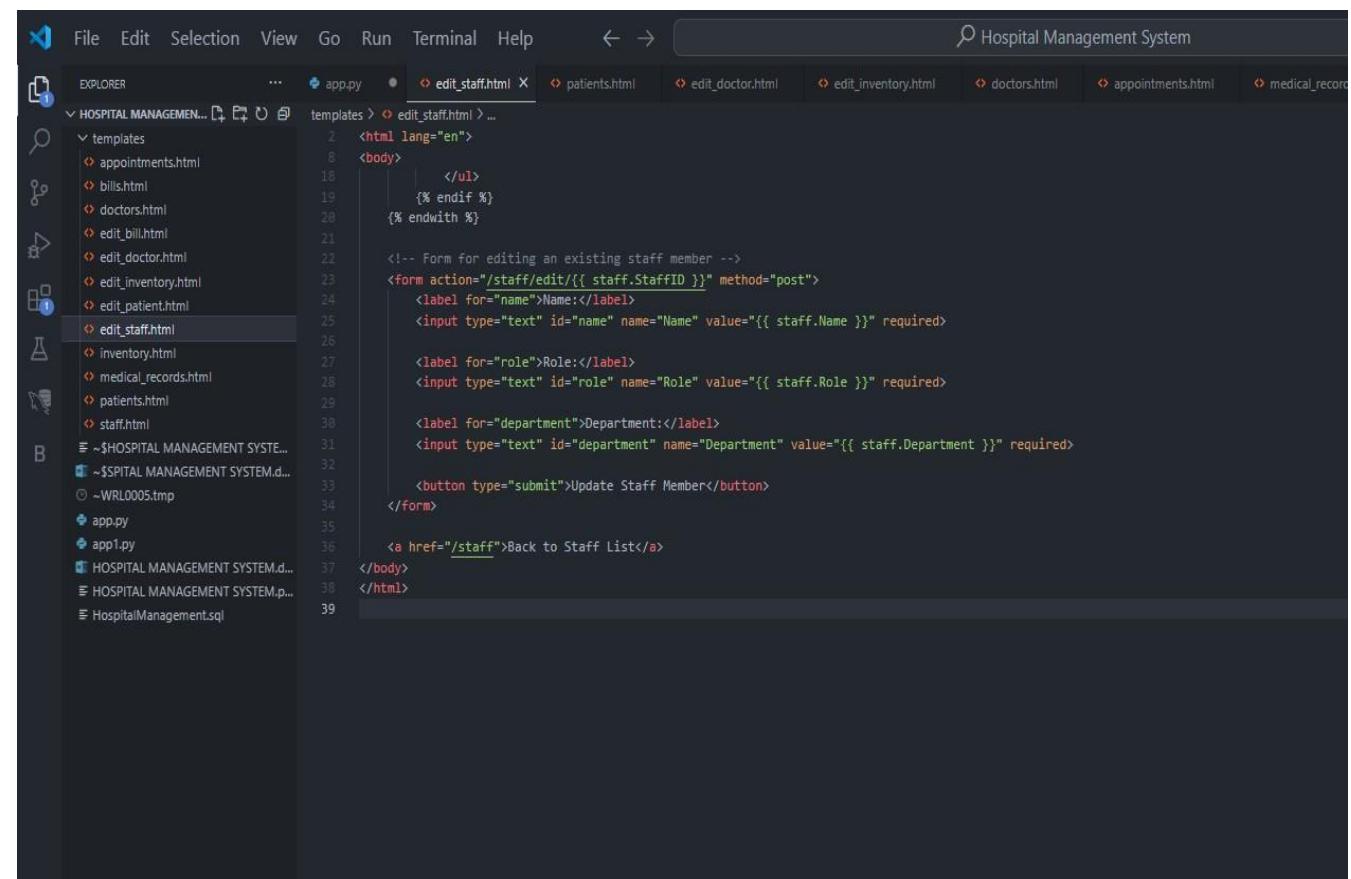
Below is the screenshot of the **Staff page** and **edit Staff** HTML codes that makes CRUD operations on the managing Staff side of the HMS.



```

<html lang="en">
  <body>
    <table border="1">
      <thead>
        <tr>
          <td>{{ staff_member.Name }}</td>
          <td>{{ staff_member.Role }}</td>
          <td>{{ staff_member.Department }}</td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <a href="/staff/edit/{{ staff_member.StaffID }}">>Edit</a>
          <a href="/staff/delete/{{ staff_member.StaffID }}" onclick="return confirm('Are you sure you want to delete this staff member?');">>Delete</a>
        </tr>
      <% endfor %>
    </tbody>
  </table>
</body>
</html>

```



```

<ul style="list-style-type: none; padding-left: 0;">
  <% endif %>
  <% endwith %>

```

```

<!-- Form for editing an existing staff member -->
<form action="/staff/edit/{{ staff.StaffID }}" method="post">
  <label for="name">Name:</label>
  <input type="text" id="name" name="Name" value="{{ staff.Name }}" required>

  <label for="role">Role:</label>
  <input type="text" id="role" name="Role" value="{{ staff.Role }}" required>

  <label for="department">Department:</label>
  <input type="text" id="department" name="Department" value="{{ staff.Department }}" required>

  <button type="submit">Update Staff Member</button>
</form>

<a href="/staff">Back to Staff List</a>
</body>
</html>

```

Below is the screenshot of the **medical records** page HTML codes that makes CRUD operations on the managing medical records side of the HMS.

```

<html lang="en">
  <body>
    <table>
    </table>
    <h2>Add New Medical Record</h2>
    <form action="/medicalrecords" method="POST">
      <label for="patient_id">Patient ID:</label>
      <input type="text" id="patient_id" name="PatientID" required><br>
      <label for="visit_date">Visit Date:</label>
      <input type="date" id="visit_date" name="VisitDate" required><br>
      <label for="diagnosis">Diagnosis:</label>
      <input type="text" id="diagnosis" name="Diagnosis" required><br>
      <label for="treatment">Treatment:</label>
      <input type="text" id="treatment" name="Treatment" required><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>

```

BACKEND CODES

As seen in the screenshot below, the database connection can be seen as connected to the **local host with the user name as “user” and password as “Root@root” also database connected to is “Hospital Management” with the port id as “3306”**. As the database is on the same computer as the backend codes, the connection host is mentioned as **“Localhost”**.

```

# Click here to ask Blackline to help you code faster
from flask import Flask, request, render_template, redirect, flash, session
import mysql.connector
from mysql.connector import Error

app = Flask(__name__)
app.secret_key = '123' # Needed to use sessions and flash messages

# Database Connection
def create_connection():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='Root@root',
            database='HospitalManagement',
            port=3306
        )
        return connection
    except Error as e:
        print(f"The error '{e}' occurred")
        return None

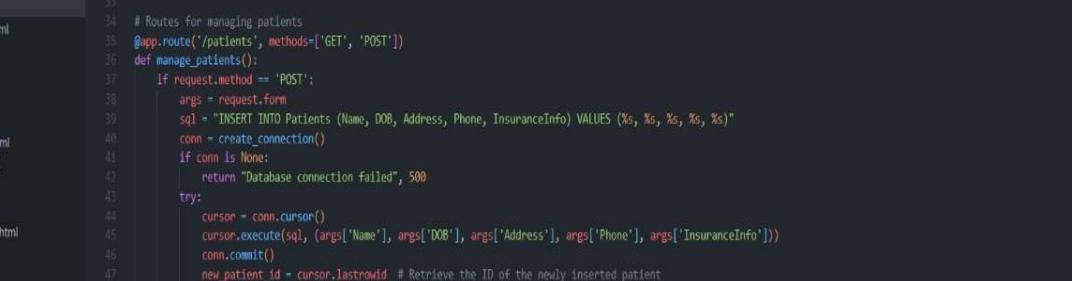
def close_connection(connection):
    if connection:
        connection.close()

# Utility for simplifying update operations
def construct_update_sql(table, args, primary_key, id_value):
    columns = ["%s = %s" for k, v in args.items() if v is not None]
    sql = f"UPDATE {table} SET {', '.join(columns)} WHERE {primary_key} = %s"
    values = tuple(v for v in args.values() if v is not None) + (id_value,)
    return sql, values

```

Also, can be seen in the screenshot above, only the update codes is given in the backend so as to give the update commands from the backend as the database was created explicitly as mentioned above in the database codes above.

Below are the codes for managing the patients and the Rest codes for GET and POST requests, also includes error handling codes and validation of the details entered in the frontend for the patients. The codes also include the **EDIT codes** to perform the CRUD operations.



The screenshot shows a browser window titled "Hospital Management System". The address bar contains the URL "http://127.0.0.1:5005/patients". The left sidebar of the browser displays a file tree with several files under "HOSPITAL MANAGEMENT SYSTEM" and "templates". The "edit_patient.html" file is currently selected. The main content area shows the Python code for the "patients.html" template:

```
# Routes for managing patients
@app.route('/patients', methods=['GET', 'POST'])
def manage_patients():
    if request.method == 'POST':
        args = request.form
        sql = "INSERT INTO Patients (Name, DOB, Address, Phone, InsuranceInfo) VALUES (%s, %s, %s, %s, %s)"
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        try:
            cursor = conn.cursor()
            cursor.execute(sql, (args['Name'], args['DOB'], args['Address'], args['Phone'], args['InsuranceInfo']))
            conn.commit()
            new_patient_id = cursor.lastrowid # Retrieve the ID of the newly inserted patient
            return redirect(f'/patients/edit/{new_patient_id}') # Redirect to the edit page of the new patient
        finally:
            close_connection(conn)
    else:
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        try:
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT * FROM Patients")
            rows = cursor.fetchall()
            return render_template('patients.html', patients=rows)
        finally:
            close_connection(conn)
```

The screenshot shows a web browser window titled "Hospital Management System". The left sidebar of the browser displays a file tree for a Python application named "HOSPITAL MANAGEMENT SYSTEM". The "app.py" file is open, showing code for managing patients. The code includes routes for editing patients, deleting patients, and rendering templates like "edit_patient.html". The browser's status bar indicates the URL is "http://127.0.0.1:5000/patients/edit/1". The main content area of the browser shows a table with patient data, with the first row selected.

```
File Edit Selection View Go Run Terminal Help ← → Hospital Management System EXPLORER ... app.py edit_staff.html patients.html edit_doctor.html edit_inventory.html doctors.html appointments.html medical_records.html bills.html edit_bill.html HOSPITAL MANAGEMENT SYSTEM templates appointments.html bills.html doctors.html edit_bill.html edit_doctor.html edit_inventory.html edit_patient.html edit_staff.html inventory.html medical_records.html patients.html staff.html -HOSPITAL MANAGEMENT SYSTEM... -$PITAL MANAGEMENT SYSTEM.d... ~WRL0005.mp app.py app1.py HOSPITAL MANAGEMENT SYSTEM.d... HOSPITAL MANAGEMENT SYSTEM.p... HospitalManagement.sql
```

```
app.py > manage_doctors
  # Routes for editing patients
  @app.route('/patients/edit<int:id>', methods=['GET', 'POST'])
  def edit_patient(id):
    if request.method == 'POST':
      args = request.form
      sql = "UPDATE Patients SET Name = %s, DOB = %s, Address = %s, Phone = %s, InsuranceInfo = %s WHERE PatientID = %s"
      conn = create_connection()
      cursor = conn.cursor()
      cursor.execute(sql, (args['Name'], args['DOB'], args['Address'], args['Phone'], args['InsuranceInfo'], id))
      conn.commit()
      close_connection(conn)
      return redirect('/patients')
    else:
      conn = create_connection()
      cursor = conn.cursor(dictionary=True)
      cursor.execute("SELECT * FROM Patients WHERE PatientID = %s", (id,))
      patient = cursor.fetchone()
      close_connection(conn)
      return render_template('edit_patient.html', patient=patient)

  # Routes for deleting patients
  @app.route('/patients/delete<int:id>', methods=['GET'])
  def delete_patient(id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute('DELETE FROM Patients WHERE PatientID = %s', (id,))
    conn.commit()
    close_connection(conn)
    return redirect('/patients')
```

Below are the codes for managing the **doctors** and the Rest codes for GET and POST requests, also includes error handling codes and validation of the details entered in the frontend for the doctors. The codes also include the **EDIT codes** to perform the CRUD operations.

The image shows three vertically stacked code editors, each displaying a portion of a Python Flask application for a Hospital Management System. The code is written in Python and uses Jinja2 templating for HTML pages. It connects to a MySQL database named 'HospitalManagement'.

- Top Editor:** Handles routes for managing doctors. It includes functions for creating a new doctor (POST), updating a doctor's information (PUT), and listing all doctors (GET). It uses SQLAlchemy to interact with the 'Doctors' table.
- Middle Editor:** Handles the edit functionality for doctors. It shows how to update a specific doctor's record by ID (PUT) and delete a doctor by ID (DELETE).
- Bottom Editor:** Handles appointments management. It includes a route for deleting an appointment by ID (DELETE) and a route for managing appointments (POST). It interacts with the 'Appointments' table.

```

app.route('/doctors', methods=['GET', 'POST'])
def manage_doctors():
    if request.method == 'POST':
        args = request.form
        sql = "INSERT INTO Doctors (Name, Specialization, Schedule) VALUES (%s, %s, %s)"
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        try:
            cursor = conn.cursor()
            cursor.execute(sql, (args['Name'], args['Specialization'], args['Schedule']))
            conn.commit()
            return redirect('/doctors')
        except Error as e:
            print(f"SQL Error: {e}")
            return "Error in database operation", 500
    finally:
        close_connection(conn)

@app.route('/doctors/edit/', methods=['GET', 'POST'])
def edit_doctor(id):
    if request.method == 'POST':
        args = request.form
        sql = "UPDATE Doctors SET Name = %s, Specialization = %s, Schedule = %s WHERE DoctorID = %s"
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        try:
            cursor = conn.cursor()
            cursor.execute(sql, (args['Name'], args['Specialization'], args['Schedule'], id))
            conn.commit()
            return redirect('/doctors')
        except Error as e:
            print(f"SQL Error: {e}")
            return "Error in database operation", 500
    finally:
        close_connection(conn)

@app.route('/doctors/delete/', methods=['GET'])
def delete_doctor(id):
    conn = create_connection()

# Routes for managing appointments
@app.route('/doctors/delete/<int:id>', methods=['GET'])
def delete_doctor_id():
    conn = create_connection()
    if conn is None:
        return "Database connection failed", 500
    try:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM Doctors WHERE DoctorID = %s", (id,))
        conn.commit()
        return redirect('/doctors')
    except Error as e:
        print(f"SQL Error: {e}")
        return "Error in database operation", 500
    finally:
        close_connection(conn)

@app.route('/appointments', methods=['GET', 'POST'])
def manage_appointments():
    if request.method == 'POST':
        args = request.form
        patient_id = args['PatientID']
        doctor_id = args['DoctorID']
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        cursor = conn.cursor()
        # Validate PatientID and DoctorID
        cursor.execute("SELECT 1 FROM Patients WHERE PatientID = %s", (patient_id,))
        if cursor.fetchone() is None:
            return "Invalid Patient ID", 400
        cursor.execute("SELECT 1 FROM Doctors WHERE DoctorID = %s", (doctor_id,))
        if cursor.fetchone() is None:
            return "Invalid Doctor ID", 400
        sql = "INSERT INTO Appointments (PatientID, DoctorID, Date, Time, Purpose) VALUES (%s, %s, %s, %s, %s)"
        cursor.execute(sql, (patient_id, doctor_id, args['Date'], args['Time'], args['Purpose']))
        conn.commit()
        return "Appointment created successfully", 201
    else:
        return render_template('appointments.html')

```

Below are the codes for managing the **Appointments** and the Rest codes for GET and POST requests, also includes error handling codes and validation of the details entered in the frontend for the appointments. The codes also include the **EDIT codes** to perform the CRUDoperations.

The screenshot displays two windows of a Python application titled "Hospital Management System" running on a Windows operating system. Both windows show the same file structure and code content, indicating a single application with multiple open tabs.

Left Window (appointments.html):

```
File Edit Selection View Go Run Terminal Help < > Hospital Management System
```

EXPLORER

HOSPITAL MANAGEMENT SYSTEM

- templates
 - appointments.html
 - bills.html
 - doctors.html
 - edit_bill.html
 - edit_doctor.html
 - edit_inventory.html
 - edit_patient.html
 - edit_staff.html
 - inventory.html
 - medical_records.html
 - patients.html
 - staff.html

```
app.py > edit_appointment
def manage_appointments():
    cursor.execute(sql, (patient_id, doctor_id, args['Date'], args['Time'], args['Purpose']))
    conn.commit()
    close_connection(conn)
    return redirect('/appointments')

else:
    conn = create_connection()
    if conn is None:
        return "Database connection failed", 500
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Appointments")
    rows = cursor.fetchall()
    close_connection(conn)
    return render_template('appointments.html', appointments=rows)

# Routes for editing appointments
@app.route('/appointments/edit<int:id>', methods=['GET', 'POST'])
def edit_appointment(id):
    if request.method == 'POST':
        args = request.form
        sql = "UPDATE Appointments SET PatientID = %s, DoctorID = %s, Date = %s, Time = %s, Purpose = %s WHERE AppointmentID = %s"
        conn = create_connection()
        cursor = conn.cursor()
        cursor.execute(sql, (args['PatientID'], args['DoctorID'], args['Date'], args['Time'], args['Purpose'], id))
        conn.commit()
        close_connection(conn)
        return redirect('/appointments')

    else:
        conn = create_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM Appointments WHERE AppointmentID = %s", (id,))
        appointment = cursor.fetchone()
        close_connection(conn)
        return render_template('edit_appointment.html', appointment=appointment)
```

Right Window (medical_records.html):

```
File Edit Selection View Go Run Terminal Help < > Hospital Management System
```

EXPLORER

HOSPITAL MANAGEMENT SYSTEM

- templates
 - appointments.html
 - bills.html
 - doctors.html
 - edit_bill.html
 - edit_doctor.html
 - edit_inventory.html
 - edit_patient.html
 - edit_staff.html
 - inventory.html
 - medical_records.html
 - patients.html
 - staff.html

```
app.py > manage_medical_records
# Routes for deleting appointments
@app.route('/appointments/delete<int:id>', methods=['GET'])
def delete_appointment(id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Appointments WHERE AppointmentID = %s", (id,))
    conn.commit()
    close_connection(conn)
    return redirect('/appointments')

# Routes for managing medical records
@app.route('/medicalrecords', methods=['GET', 'POST'])
def manage_medical_records():
    if request.method == 'POST':
        args = request.form
        patient_id = args['PatientID']
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        cursor = conn.cursor()
        # Validate PatientID
        cursor.execute("SELECT 1 FROM Patients WHERE PatientID = %s", (patient_id,))
        if cursor.fetchone() is None:
            close_connection(conn)
            return "Invalid Patient ID", 400
        sql = "INSERT INTO MedicalRecords (PatientID, VisitDate, Diagnosis, Treatment) VALUES (%s, %s, %s, %s)"
        cursor.execute(sql, (patient_id, args['VisitDate'], args['Diagnosis'], args['Treatment']))
        conn.commit()
        close_connection(conn)
        return redirect('/medicalrecords')

    else:
        conn = create_connection()
        if conn is None:
            return "Database connection failed", 500
        cursor = conn.cursor(dictionary=True)
```

Above are the codes for managing the **medical records** and the Rest codes for GET and POSTrequests, also includes error handling codes and validation of the details entered in the frontend for the medical records.

The codes below also include the **EDIT** codes to perform the CRUD operations.



```
File Edit Selection View Go Run Terminal Help < > Hospital Management System Explorer ... app.py edit_staff.html patients.html edit_doctor.html edit_inventory.html doctors.html appointments.html medical_records.html bills.html edit_bill.html HOSPITAL MANAGEMENT SYSTEM ... templates appointments.html bills.html doctors.html edit_bill.html edit_doctor.html edit_inventory.html edit_patient.html edit_staff.html inventory.html medical_records.html patients.html staff.html - HOSPITAL MANAGEMENT SYSTEM... - SPITAL MANAGEMENT SYSTEM... - WRL0005.tmp app.py app1.py HOSPITAL MANAGEMENT SYSTEM... HOSPITAL MANAGEMENT SYSTEM... HospitalManagement.sql app.py > delete_medical_record 238 def manage_medical_records(): 239     """ 240         return "Database connection failed", 500 241     cursor = conn.cursor(dictionary=True) 242     cursor.execute("SELECT * FROM MedicalRecords") 243     rows = cursor.fetchall() 244     close_connection(conn) 245     return render_template('medical_records.html', medicalrecords=rows) 246 247 # Routes for editing medical records 248 @app.route('/medicalrecords/edit/', methods=['GET', 'POST']) 249 def edit_medical_record(id): 250     if request.method == 'POST': 251         args = request.form 252         sql = "UPDATE MedicalRecords SET PatientID = %s, VisitDate = %s, Diagnosis = %s, Treatment = %s WHERE RecordID = %s" 253         conn = create_connection() 254         cursor = conn.cursor() 255         cursor.execute(sql, (args['PatientID'], args['VisitDate'], args['Diagnosis'], args['Treatment'], id)) 256         conn.commit() 257         close_connection(conn) 258         return redirect('/medicalrecords') 259     else: 260         conn = create_connection() 261         cursor = conn.cursor(dictionary=True) 262         cursor.execute("SELECT * FROM MedicalRecords WHERE RecordID = %s", (id,)) 263         record = cursor.fetchone() 264         close_connection(conn) 265         return render_template('edit_medical_record.html', record=record) 266 267 # Routes for deleting medical records 268 @app.route('/medicalrecords/delete/', methods=['GET']) 269 def delete_medical_record(id): 270     conn = create_connection() 271     cursor = conn.cursor() 272     cursor.execute("DELETE FROM MedicalRecords WHERE RecordID = %s", (id,))
```

Below are the codes for managing the **bills** and the Rest codes for GET and POST requests, also includes error handling codes and validation of the details entered in the frontend for the bills. The codes also include the **EDIT codes** to perform the CRUD operations.



The screenshot shows a web browser window titled "Hospital Management System". The URL bar displays "Hospital Management System". The page content is a Python application with code for managing bills. The code includes imports for Flask, SQLAlchemy, and various database models. It defines a route for editing bills, which handles GET and POST requests. For a POST request, it constructs an SQL UPDATE statement to update the Bills table based on the provided PatientID, Date, Amount, and Status. It then commits the transaction and flashes a success message. If an error occurs during the update, it flashes an error message. Finally, it closes the connection and redirects to the bills page. The browser's sidebar shows the file structure, including files like app.py, edit_bill.html, and various templates for managing patients, doctors, inventory, appointments, and medical records.

```
File Edit Selection View Go Run Terminal Help < > Hospital Management System
```

```
EXPLORER ... app.py edit.staff.html patients.html edit.doctor.html edit.inventory.html doctors.html appointments.html medical_records.html bills.html edit_bill.html
```

```
HOSPITAL MANAGEMENT SYSTEM ... templates
```

```
appointmentshtml bills.html doctors.html edit_bill.html edit_doctor.html edit_inventory.html edit_patient.html edit_staff.html inventory.html medical_records.html patients.html staff.html
```

```
-$HOSPITAL MANAGEMENT SYSTEM... -$SPITAL MANAGEMENT SYSTEM... -WRL005.tmp
```

```
app.py app1.py
```

```
HOSPITAL MANAGEMENT SYSTEM.d... HOSPITAL MANAGEMENT SYSTEM.p... HospitalManagement.sql
```

```
app.py
```

```
def manage_bills():
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Bills")
    rows = cursor.fetchall()
    return render_template('bills.html', bills=rows)

finally:
    close_connection(conn)

# Routes for editing bills
@app.route('/bills/edit/', methods=['GET', 'POST'])
def edit_bill(id):
    if request.method == 'POST':
        args = request.form
        sql = "UPDATE Bills SET PatientID = %s, Date = %s, Amount = %s, Status = %s WHERE BillID = %s"
        conn = create_connection()
        if conn is None:
            flash("Database connection failed", "error")
            return redirect('/bills')

        try:
            cursor = conn.cursor()
            cursor.execute(sql, (args['PatientID'], args['Date'], args['Amount'], args['Status'], id))
            conn.commit()
            flash("Bill updated successfully", "success")
        except Error as e:
            flash(f"An error occurred: {e}", "error")
        finally:
            close_connection(conn)
        return redirect('/bills')

    else:
        conn = create_connection()
        if conn is None:
            flash("Database connection failed", "error")
            return redirect('/bills')
```

```

File Edit Selection View Go Run Terminal Help < > Hospital Management System

EXPLORER HOSPITAL MANAGEMENT SYSTEM... app.py ... templates
  appointments.html
  bills.html
  doctors.html
  edit_bill.html
  edit_doctor.html
  edit_inventory.html
  edit_patient.html
  edit_staff.html
  inventory.html
  medical_records.html
  patients.html
  staff.html

B -HOSPITAL MANAGEMENT SYSTEM...
-SPITAL MANAGEMENT SYSTEM...
-WRLD005.tmp
app.py
app1.py
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HospitalManagement.sql

343     def edit_bill(id):
344         try:
345             cursor = conn.cursor(dictionary=True)
346             cursor.execute("SELECT * FROM Bills WHERE BillID = %s", (id,))
347             bill = cursor.fetchone()
348             if bill is None:
349                 flash("No bill found with that ID", "error")
350                 return redirect('/bills')
351             return render_template('edit_bill.html', bill=bill)
352         finally:
353             close_connection(conn)
354
355     # Routes for deleting bills
356     @app.route('/bills/delete/<int:id>', methods=['GET'])
357     def delete_bill(id):
358         conn = create_connection()
359         if conn is None:
360             flash("Database connection failed", "error")
361             return redirect('/bills')
362
363         try:
364             cursor = conn.cursor()
365             cursor.execute("DELETE FROM Bills WHERE BillID = %s", (id,))
366             conn.commit()
367             flash("Bill deleted successfully", "success")
368         except Error as e:
369             flash(f"An error occurred: {e}", "error")
370         finally:
371             close_connection(conn)
372         return redirect('/bills')
373
374

```

Below are the codes for managing the **Staffs** and the Rest codes for GET and POST requests, also includes error handling codes and validation of the details entered in the frontend for the Staffs. The codes also include the **EDIT codes** to perform the CRUD operations.

```

File Edit Selection View Go Run Terminal Help < > Hospital Management System

EXPLORER HOSPITAL MANAGEMENT SYSTEM... app.py ... templates
  appointments.html
  bills.html
  doctors.html
  edit_bill.html
  edit_doctor.html
  edit_inventory.html
  edit_patient.html
  edit_staff.html
  inventory.html
  medical_records.html
  patients.html
  staff.html

B -HOSPITAL MANAGEMENT SYSTEM...
-SPITAL MANAGEMENT SYSTEM...
-WRLD005.tmp
app.py
app1.py
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HospitalManagement.sql

359     # Routes for managing staff
360     @app.route('/staff', methods=['GET', 'POST'])
361     def manage_staff():
362         if request.method == 'POST':
363             args = request.form
364             sql = "INSERT INTO Staff (Name, Role, Department) VALUES (%s, %s, %s)"
365             conn = create_connection()
366             if not conn:
367                 flash("Database connection failed", "error")
368                 return redirect('/staff') # Ensure a redirect if connection fails
369
370             try:
371                 cursor = conn.cursor()
372                 cursor.execute(sql, (args['Name'], args['Role'], args['Department']))
373                 conn.commit()
374                 flash("New member added successfully", "success")
375             except Error as e:
376                 flash(f"An error occurred: {e}", "error")
377             finally:
378                 close_connection(conn)
379             return redirect('/staff')
380         else:
381             conn = create_connection()
382             if not conn:
383                 flash("Database connection failed", "error")
384                 return redirect('/') # Redirect or show an error page
385
386             try:
387                 cursor = conn.cursor(dictionary=True)
388                 cursor.execute("SELECT * FROM Staff")
389                 staff_members = cursor.fetchall()
390                 return render_template('staff.html', staff=staff_members)
391             except Error as e:
392                 flash(f"An error occurred while retrieving data: {e}", "error")
393             finally:
394                 close_connection(conn)
395             return redirect('/') # Redirect or show an error page
396
397

```

```

File Edit Selection View Go Run Terminal Help < > Hospital Management System

EXPLORER HOSPITAL MANAGEMENT SYSTEM... app.py ... templates
  appointments.html
  bills.html
  doctors.html
  edit_bill.html
  edit_doctor.html
  edit_inventory.html
  edit_patient.html
  edit_staff.html
  inventory.html
  medical_records.html
  patients.html
  staff.html

B -HOSPITAL MANAGEMENT SYSTEM...
-SPITAL MANAGEMENT SYSTEM...
-WRLD005.tmp
app.py
app1.py
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HOSPITAL MANAGEMENT SYSTEM...
HospitalManagement.sql

401     def manage_staff():
402         finally:
403             close_connection(conn)
404
405     # Routes for editing staff
406     @app.route('/staff/edit/<int:id>', methods=['GET', 'POST'])
407     def edit_staff(id):
408         if request.method == 'POST':
409             args = request.form
410             sql = "UPDATE Staff SET Name = %s, Role = %s, Department = %s WHERE StaffID = %s"
411             conn = create_connection()
412             if not conn:
413                 flash("Database connection failed", "error")
414                 return redirect('/staff') # Redirect to staff list if connection fails
415
416             try:
417                 cursor = conn.cursor()
418                 cursor.execute(sql, (args['Name'], args['Role'], args['Department'], id))
419                 conn.commit()
420                 flash("Staff updated successfully", "success")
421             except Error as e:
422                 flash(f"An error occurred: {e}", "error")
423             finally:
424                 close_connection(conn)
425             return redirect('/staff') # Always redirect after POST
426
427         else:
428             conn = create_connection()
429             if not conn:
430                 flash("Database connection failed", "error")
431                 return redirect('/staff') # Redirect if connection fails
432
433             try:
434                 cursor = conn.cursor(dictionary=True)
435                 cursor.execute("SELECT * FROM Staff WHERE StaffID = %s", (id,))
436
437

```

```

File Edit Selection View Go Run Terminal Help ← → ⌘ Hospital Management System

EXPLORER HOSPITAL MANAGEMENT SYSTEM
templates
  appointments.html
  bills.html
  doctors.html
  edit_bill.html
  edit_doctor.html
  edit_inventory.html
  edit_patient.html
  edit_staff.html
  inventory.html
  medical_records.html
  patients.html
  staff.html
B ~$HOSPITAL MANAGEMENT SYSTEM...
  ~$SPITAL MANAGEMENT SYSTEM.d...
    ~WRL0005.tmp
    app.py
    app1.py
  HOSPITAL MANAGEMENT SYSTEM.d...
  HOSPITAL MANAGEMENT SYSTEM.p...
  HospitalManagement.sql

app.py
  def edit_staff(id):
    staff = cursor.fetchone()
    if not staff:
      flash("No staff found with that ID", "error")
      return redirect('/staff') # Redirect if no staff is found
    return render_template('edit_staff.html', staff=staff) # Render template if staff is found
  except Error as e:
    flash(f"An error occurred while retrieving staff data: {e}", "error")
    return redirect('/staff') # Redirect on error
  finally:
    close_connection(conn)

# Routes for deleting staff
@app.route('/staff/delete/<int:id>', methods=['GET'])
def delete_staff(id):
  conn = create_connection()
  cursor = conn.cursor()
  cursor.execute("DELETE FROM Staff WHERE StaffID = %s", (id,))
  conn.commit()
  close_connection(conn)
  return redirect('/staff')

# Routes for managing inventory
@app.route('/inventory', methods=['GET', 'POST'])
def manage_inventory():
  if request.method == 'POST':
    # Handle form submission for creating or updating inventory items
    pass
  else:
    conn = create_connection()
    if not conn:
      flash("Database connection failed", "error")

```

Below are the codes for managing the **Inventory** and the Rest codes for GET and POST requests, also includes error handling codes and validation of the details entered in the frontend for the Inventory. The codes also include **EDIT codes** to perform CRUD operations.

```

File Edit Selection View Go Run Terminal Help ← → ⌘ Hospital Management System

EXPLORER HOSPITAL MANAGEMENT SYSTEM
templates
  appointments.html
  bills.html
  doctors.html
  edit_bill.html
  edit_doctor.html
  edit_inventory.html
  edit_patient.html
  edit_staff.html
  inventory.html
  medical_records.html
  patients.html
  staff.html
B ~$HOSPITAL MANAGEMENT SYSTEM...
  ~$SPITAL MANAGEMENT SYSTEM.d...
    ~WRL0005.tmp
    app.py
    app1.py
  HOSPITAL MANAGEMENT SYSTEM.d...
  HOSPITAL MANAGEMENT SYSTEM.p...
  HospitalManagement.sql

app.py
  def manage_inventory():
    return redirect('/') # Redirect or show an error page if connection fails

  try:
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM Inventory")
    inventory_items = cursor.fetchall()
    return render_template('inventory.html', inventory=inventory_items)
  except Error as e:
    flash(f"An error occurred while retrieving inventory data: {e}", "error")
    return redirect('/') # Redirect or show an error page on error
  finally:
    close_connection(conn)

# Routes for editing inventory
@app.route('/inventory/edit/<int:id>', methods=['GET', 'POST'])
def edit_inventory_item(id):
  conn = create_connection()
  cursor = conn.cursor(dictionary=True)
  cursor.execute("SELECT * FROM Inventory WHERE ItemID = %s", (id,))
  item = cursor.fetchone()
  close_connection(conn)

  if not item:
    flash("Inventory item not found", "error")
    return redirect('/inventory')

  if request.method == 'POST':
    args = request.form
    sql = "UPDATE Inventory SET Name = %s, Quantity = %s, ReorderLevel = %s WHERE ItemID = %s"
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute(sql, (args['Name'], args['Quantity'], args['ReorderLevel'], id))
    conn.commit()

  if request.method == 'GET':
    args = request.form
    sql = "UPDATE Inventory SET Name = %s, Quantity = %s, ReorderLevel = %s WHERE ItemID = %s"
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute(sql, (args['Name'], args['Quantity'], args['ReorderLevel'], id))
    conn.commit()
    close_connection(conn)
    return redirect('/inventory')

  else:
    return render_template('edit_inventory.html', item=item)

# Ro (function) def delete_inventory(id: Any) -> Response
@app
def delete_inventory(id):
  conn = create_connection()
  cursor = conn.cursor()
  cursor.execute("DELETE FROM Inventory WHERE ItemID = %s", (id,))
  conn.commit()
  close_connection(conn)
  return redirect('/inventory')

if __name__ == '__main__':
  app.run(debug=True, port=5000)

```

CONCLUSION SUMMARY

In our journey to develop a Hospital Management System (HMS), we've created a comprehensive database-driven application that revolutionizes how healthcare facilities manage their operations. Our HMS encompasses a relational database schema that efficiently organizes and interconnects various entities critical for hospital operations. These entities include Patients, Doctors, Appointments, Medical Records, Bills, Staff, and Inventory. Each entity is defined with specific attributes, ensuring a detailed and structured approach to data management.

The application provides essential functionalities:

1. Patient Management: Registering patients, updating profiles, and managing medical histories.
2. Appointment Scheduling: Facilitating seamless booking, rescheduling, and cancellation of appointments.
3. Medical Records Management: Creating and accessing detailed medical records securely.
4. Billing and Insurance Processing: Automating bill generation and managing insurance claims.
5. Staff and Schedule Management: Maintaining records of staff roles, departments, and schedules.
6. Inventory Management: Tracking medical supplies and medications.

Our database structure ensures efficient relationships between entities, optimizing hospital workflows and ensuring comprehensive patient care. We gained hands-on experience on below:

1. Database Design: We learned the importance of a well-structured relational database schema in managing complex data relationships. Designing entities, attributes, and relationships helped us understand how data flows within an application.
2. Application Development: Building an application from scratch taught us valuable lessons in coding, including creating frontend interfaces and backend functionalities. We gained hands-on experience in CRUD (Create, Read, Update, Delete) operations.
3. User Experience (UX): We emphasized user-friendly interfaces to ensure easy navigation for hospital staff. Integrating functionalities into a cohesive platform enhanced user experience and operational efficiency.
4. Data Security and Privacy: Managing patient data required strict adherence to security and privacy standards. We implemented role-based access control and ensured data encryption to maintain confidentiality.
5. Error Handling and Validation: Developing robust error handling mechanisms and data validation processes was crucial for maintaining data integrity and application reliability.
6. Collaboration: Working in a group taught us the importance of effective communication and collaboration. Dividing tasks, sharing responsibilities, and coordinating efforts were key to the project's success.

7. Adaptability: As we encountered challenges and requirements evolved, we learned to adapt our designs and codebase. This flexibility is essential in real-world application development.

In conclusion, our HMS project represents a culmination of database design principles, application development, and teamwork. We've not only created a functional system but also gained valuable insights and skills that will serve us well in our future endeavors in the field of technology and healthcare.

REFERENCES:

1. <https://code.visualstudio.com/> - VS STUDIO – For Python Coding
2. <https://www.mysql.com/products/workbench/> - My SQL WorkBench – for SQL queries
3. <https://palletsprojects.com/p/jinja/> - Jinja2 – For Templating
4. <https://flask.palletsprojects.com/en/3.0.x/> - Flask Framework
5. <https://www.ibm.com/topics/rest-apis> - REST API

