Author

Name:Shashank Ashutosh Kulkarni
Roll No:23f3000699
Student email:23f3000699@ds.study.iitm.ac.in
About me:I am a student at IITM online Bsc degree I am currently completing both my diplomas.I am passionate about app development and machine learning

AI/LLM usage

I used Chatgpt(gpt 5) mainly to assist me in the beautification process and project structure. Overall 15-20%.

Description

Project Title:Hospital Management System
Problem Statement:We are tasked with creating a hospital management system which allows Admins, Doctors, and Patients to interact with the system based on their roles
Approach:I created various html pages and connected the backend using flask allowing various functionality for each stakeholder such as accessing the system,managing appointments,tracking medical records and performing role specific operations

Technologies used

Technology / Library	Purpose in the Project
Flask	Core backend framework used to build routes, authentication, dashboards, and all server logic.
Flask-Login	Handles user authentication (login, logout), session management, and role-based access (Admin/Doctor/Patient).
SQLAlchemy	ORM used to define models (User, Doctor, Patient, Appointment, Availability, Treatment) and interact with the SQLite database.

SQLite	Lightweight local database used to store all application data (users, appointments, availability, departments).
Jinja2	Template engine used for rendering dynamic HTML pages for dashboards, forms, lists, and details.
HTML / CSS	Used to build and style all frontend pages, layouts, tables, buttons, forms, and dashboards.
Python (Standard Library)	Used for utilities like datetime, validation, role checking, filtering, and general backend logic.

DB Schema Design

1. **User** — stores user details (id, name, email, passwordwithhash, role, active)
2. **Patient** — stores patient info (id, user_id, age, gender, phone, address)
3. **Doctor** — stores doctor info (id, user_id, specialization, department_id, bio)
4. **Department** — stores medical departments (id, name, description)
5. **Appointment** — stores appointment details (id, patient_id, doctor_id, date, time, status)
6. **Treatment** — stores treatment records (id, appointment_id, diagnosis, prescription, notes)
7. **DoctorAvail** — stores doctor availability slots (id, doctor_id, date, start_time, end_time)

Relationships:

1]One to One relationship between User and Doctor because one user can only be one doctor

2]One to One relationship between User and patient because one user can only be one patient

3]One to many relationship Department-> Doctor because One department can have many doctors but one doctor can only belong to one department

4]One to many relationship Doctor->Appointment because One doctor can have many appointments but one appointment can have only one doctor

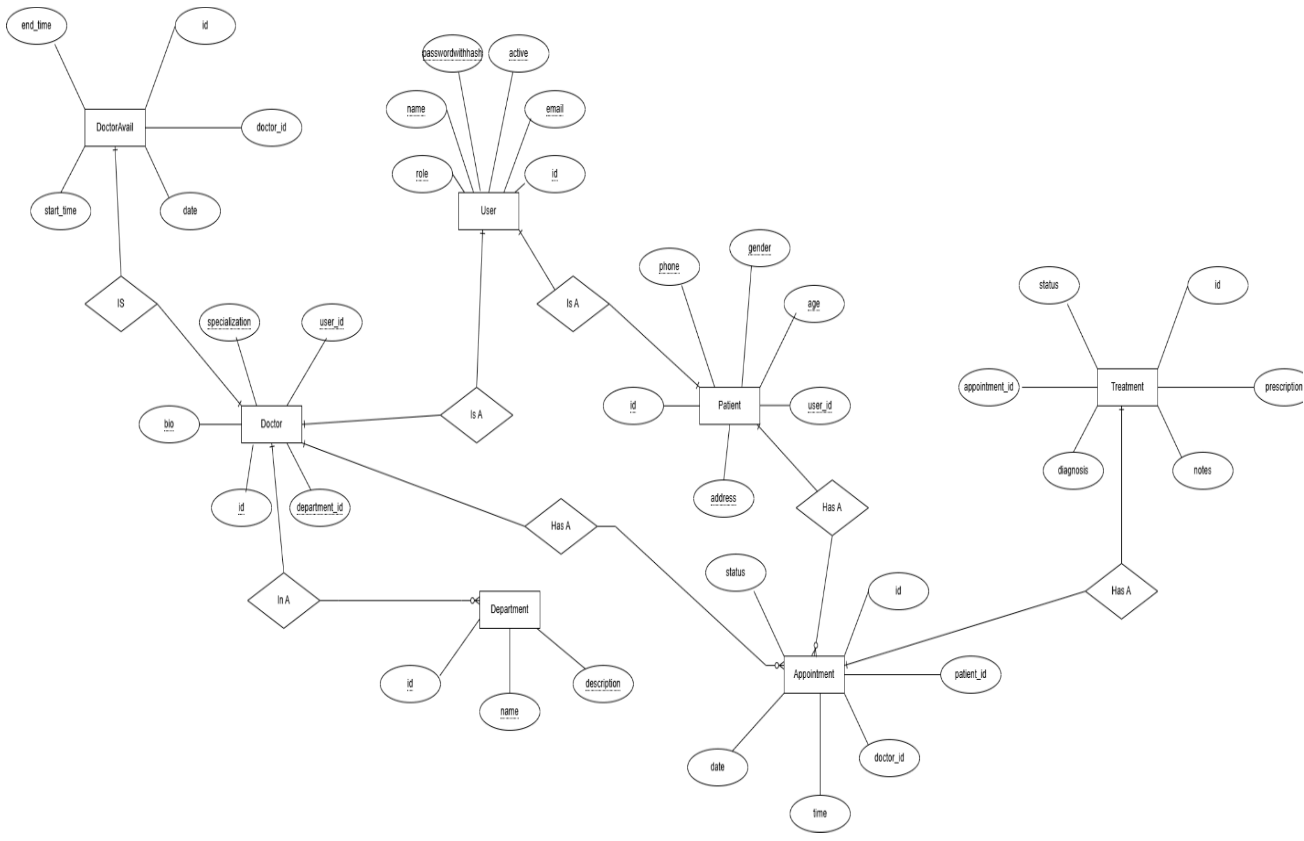
5]One to many Patient->Appointment because One patients can have many appointments but one appointment can have only one patient

6]One to One between Appointment and Treatment as one appointment can give only one treatment(assumed) and one treatment can belong to only one appointment(assumed)

7]One to Many Doctor->DoctorAvail because One Doctor can have multiple availability slots but one slot belongs to only doctor

Reason was for simplicity and following the instructions provided

ER diagram



API Design

Endpoint	Method	Description
/api/login	POST	Authenticate user (Admin / Doctor / Patient)
/api/register	POST	Register a new Patient or Doctor
/api/departments	GET	Fetch list of all departments
/api/doctors	GET	Fetch all doctors (filter by department/specialization optional)
/api/doctor//availability	GET	Fetch availability slots for a specific doctor
/api/appointments	GET	Fetch appointments for logged-in user (Doctor/Patient)
/api/appointments/book	POST	Book appointment (Patient only)
/api/appointments//update	POST	Update appointment status & treatment (Doctor)

/api/appointments//cancel	POST	Cancel appointment (Patient or Doctor)
/api/patient//history	GET	Get complete appointment history for a patient
/api/admin/doctors	GET	Admin fetches all doctors
/api/admin/patients	GET	Admin fetches all patients
/api/admin/doctor/add	POST	Admin adds a doctor (with associated user)
/api/admin/doctor//blacklist	POST	Blacklist (deactivate) a doctor
/api/admin/patient//blacklist	POST	Blacklist (deactivate) a patient
/api/admin/dashboard	GET	Admin overview: doctor count, patient count, appointments, upcoming

Architecture and Features

Architecture Overview:

- **app.py** – main Flask application entry point
- **/models** – are in app.py
- **/routes-** are all in app.py
- **/templates** – Jinja2 HTML templates
- **/static** – CSS, JS, and chart visualization files

Features:

1]User Registration & Login{Secured using hashing,role based,also enables blacklist system}

2]Doctor and patient profile management, including specialization, department selection, and personal details.

3]Appointment booking and scheduling with real-time slot validation to avoid double bookings

4]Doctor availability management, allowing doctors to provide morning/evening slots for the upcoming week

5]Treatment recording, where doctors can update diagnosis, prescriptions, and notes for each patient visit

6]Admin dashboard with analytics, showing doctor, patient, and appointment counts plus upcoming appointments

7]Patient dashboard, displaying upcoming visits, past visit history, and department-wise doctor listings

Video

https://drive.google.com/file/d/1J52JqAI_-4U_BznSMKBC6zbhoDLiMPkJ/view?usp=sharing