Hospital Management System

Author
Name : Ritesh Kumar Sah
Roll No : 24f1000739
Email : 24f1000739@ds.study.iitm.ac.in

Introduction : I'm Ritesh Kumar Sah, Diploma level Student, Pursuing this degree as standalone.

1. Project Description :
The Hospital Management System is a web application that streamlines hospital operations by managing patients, doctors, appointments, and treatments. It enables three user roles (Admin, Doctor, Patient) with role-specific functionalities for efficient healthcare management.

2. Technology Stack :
- Backend: Python Flask + Flask-SQLAlchemy ORM
- **Frontend:** HTML5, CSS3, Bootstrap 5 + Jinja2 templating
- **Database:** SQLite (auto-created using SQLAlchemy models)
- **Authentication:** Secure password hashing via Werkzeug
- **Session Handling:** Flask sessions with role-based route protection

3. System Features:
- Centralized dashboard for fast hospital operations
- Structured role-based access: Admin, Doctor & Patient
- Medical history storage with diagnostics and prescriptions
- Doctor availability system for appointment planning
- Appointment lifecycle management (Book → Complete / Cancel)
- Unique constraints to block double-booking of time slots
- Clean and responsive UI for better user experience

3.1 Admin Features:
- Add / update / delete doctor profiles
- Search doctors & patients by name, contact, or specialization
- Monitor upcoming & historical appointments
- Ability to blacklist users (block system access)
- Pre-created administrator user for system bootstrapping

3.2 Doctor Features:
- View upcoming assigned appointments & patients list
- Set availability for next 7 days (Morning/Evening slots)
- Update treatment records per appointment:
- Visit type, tests, diagnosis, prescription & notes
- Mark appointments as Completed or Cancelled

3.3 Patient Features:
- Self-registration & secure login management
- Browse departments & doctors before booking

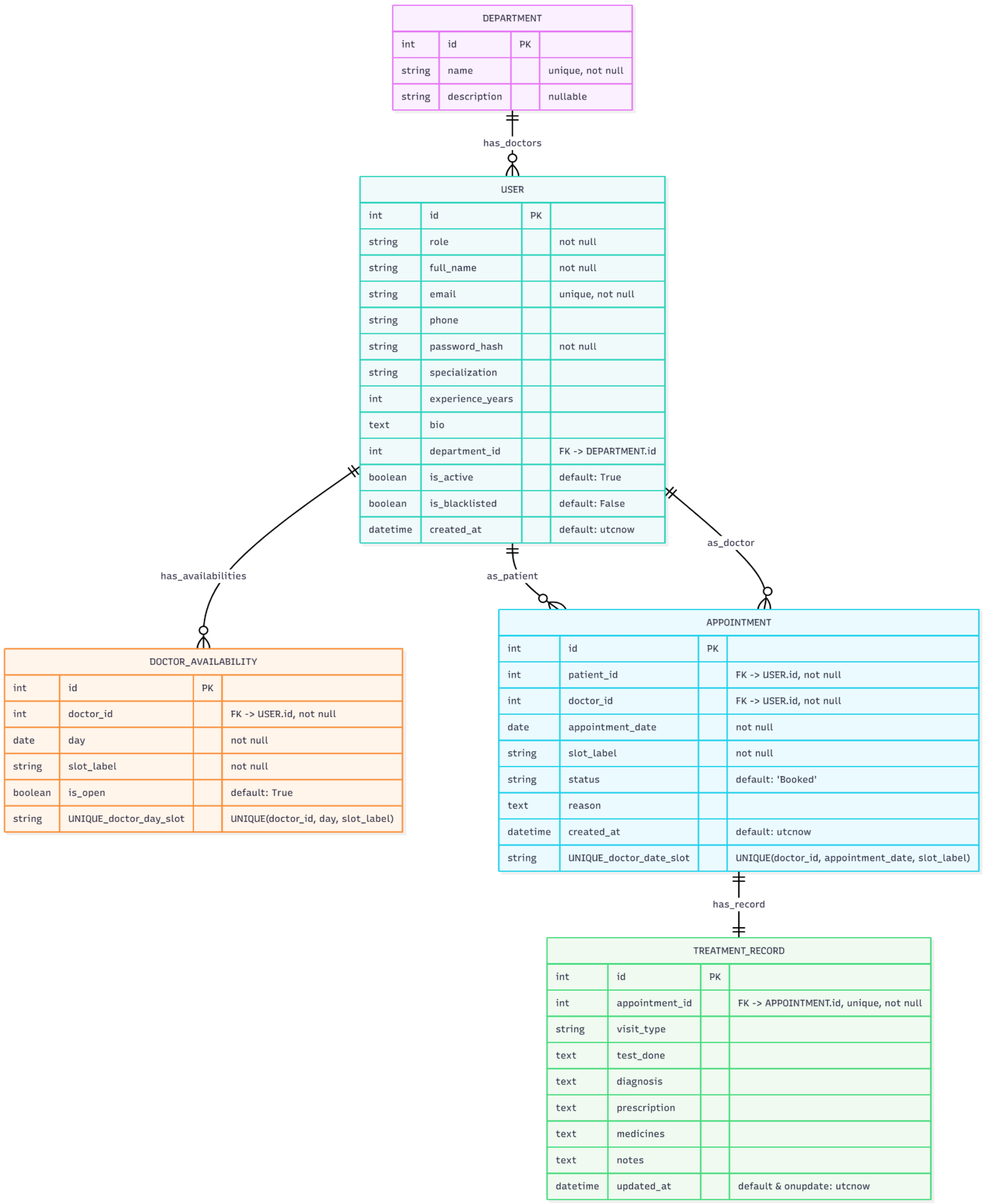
- Book / cancel appointments with live availability data
- View full medical visit history with prescriptions
- Manage own profile details

4. Security Features :

- Role-Based Access Control (RBAC) on every route
- Password hashing with PBKDF2 (no plaintext stored)
- Session validation for each restricted module
- Frontend + backend form input validation
- Blacklisting ensures banned users cannot log in

5. Database Schema – Key Entities:

- Department – Stores medical specialization data
- User – Unified table for Admin/Doctor/Patient accounts
- DoctorAvailability – Time slots published by each doctor
- Appointment – Patient-doctor bookings with status tracking
- TreatmentRecord – One-time medical report per visit



DEPARTMENT

int	id	PK	
string	name		unique, not null
string	description		nullable

USER

int	id	PK	
string	role		not null
string	full_name		not null
string	email		unique, not null
string	phone		
string	password_hash		not null
string	specialization		
int	experience_years		
text	bio		
int	department_id		FK -> DEPARTMENT.id
boolean	is_active		default: True
boolean	is_blacklisted		default: False
datetime	created_at		default: utcnow

DOCTOR_AVAILABILITY

int	id	PK	
int	doctor_id		FK -> USER.id, not null
date	day		not null
string	slot_label		not null
boolean	is_open		default: True
string	UNIQUE_doctor_day_slot		UNIQUE(doctor_id, day, slot_label)

APPOINTMENT

int	id	PK	
int	patient_id		FK -> USER.id, not null
int	doctor_id		FK -> USER.id, not null
date	appointment_date		not null
string	slot_label		not null
string	status		default: 'Booked'
text	reason		
datetime	created_at		default: utcnow
string	UNIQUE_doctor_date_slot		UNIQUE(doctor_id, appointment_date, slot_label)

TREATMENT_RECORD

int	id	PK	
int	appointment_id		FK -> APPOINTMENT.id, unique, not null
string	visit_type		
text	test_done		
text	diagnosis		
text	prescription		
text	medicines		
text	notes		
datetime	updated_at		default & onupdate: utcnow

Key Relationship:

- Department has a one-to-many relationship with User
  → A department can have multiple doctors, but a doctor belongs to only one department

- User (Doctor) has a one-to-many relationship with DoctorAvailability
  → A doctor can set multiple availability slots
- User (Doctor) has a one-to-many relationship with Appointment
  → A doctor can attend many appointments
- User (Patient) has a one-to-many relationship with Appointment
  → A patient can book multiple appointments
- Appointment has a one-to-one relationship with TreatmentRecord
  → Each visit/appointment has exactly one treatment record stored
- Unique Slot Constraint ensures:
  A doctor cannot publish duplicate availability slots for same day & label
  `(doctor_id, day, slot_label)`
- A doctor cannot have two appointments at the same date & time
  `(doctor_id, appointment_date, slot_label)`

Video Link:
https://drive.google.com/file/d/1x8okDZmdaFe6RH1CnpS8HaDhasBa8Oky/view?usp=drive_link

Declaration : I have taken help from AI in styling and layout of my web app. Used 10% AI in CSS/Bootstrap and for db setup.