MediCare Hospital Management System - Project Report

1. Student Details

Name: Tanikella Lakshmi Narayana
Roll Number: 24f1000542
Email: 24f1000542
About Me:I am currently pursuing b.tech in mechanical engineering from nit durgapur (3 rd year) along with bsc data science from iit madras

2. Project Details

Project Title: MediCare Hospital Management System

Problem Statement:

Modern healthcare facilities face significant challenges in managing appointments, patient records, doctor availability, and billing operations across multiple roles. The need for a centralized, efficient system that streamlines operations for administrators, doctors, and patients while maintaining data integrity and providing real-time analytics is critical.

Approach:

I developed a comprehensive hospital management system using Flask as the backend framework. The application implements role-based access control (Admin, Doctor, Patient), features a system in which
s, maintains a availability management system for doctors and suggestions . The system
automatically generates a unique medical ID for each patient, tracks appointment of them.

3. AI/LLM Declaration

I spent time with ChatGPT learning about SQLlite patterns and got some ideas on building the appointment scheduling system. I also used it to debug a few Flask errors . For UI stuff, it helped me .structure the Bootstrap templates. Overall, I used AI for 20- 25% of the project—mostly suggestions and learning. But all the actual implementation, testing, and problem-solving was done by me.

4. Technologies and Frameworks Used

Technology / Library	Purpose
Flask	Core backend web framework
Flask-SQLAlchemy	Object-relational mapper for SQLite database
SQLite	Lightweight persistent database
Jinja2	Template engine for rendering dynamic HTML
Bootstrap 5	Frontend styling and responsive design

5. Database Schema / ER Diagram

Core Tables:

1. **User** — Central user profile table (id, username, email, password_hash, role, full_name, DOB, gender, address),
2. **Doctor** — Doctor-specific information (user_id FK, specialization, rating, license_number, qualifications, experience_years, consultation_fee, department, room_number)
3. **Patient** — Patient-specific information (user_id FK, medical_id, blood_group, emergency_contact, insurance_provider, allergies, chronic_conditions)
4. **Availability** — Doctor time slot availability (doctor_id FK, date, start_time, end_time, is_available)
5. **Appointment** — Appointment records (doctor_id FK, patient_id FK, appointment_date, appointment_time, status)
6. **Treatment** — Treatment details per appointment (appointment_id FK, diagnosis, prescription, notes)
7. **Rating** — Post-appointment feedback (appointment_id FK, doctor_id FK, patient_id FK, rating 1-5, feedback)

Key Relationships:

•One-to-Many: User → Doctor (one user has one doctor profile) •One-to-Many: User → Patient (one user has one patient profile) •One-to-Many: Doctor → Availability (one doctor has multiple availability slots) •One-to-Many: Doctor → Appointment (one doctor has multiple appointments) •One-to-Many: Patient → Appointment (one patient has multiple appointments) •One-to-One: Appointment → Treatment (one appointment has one treatment record) •One-to-One: Appointment → Rating (one appointment has one rating) •One-to-Many: Appointment → Bill (one appointment generates one bill)

ER Diagram:

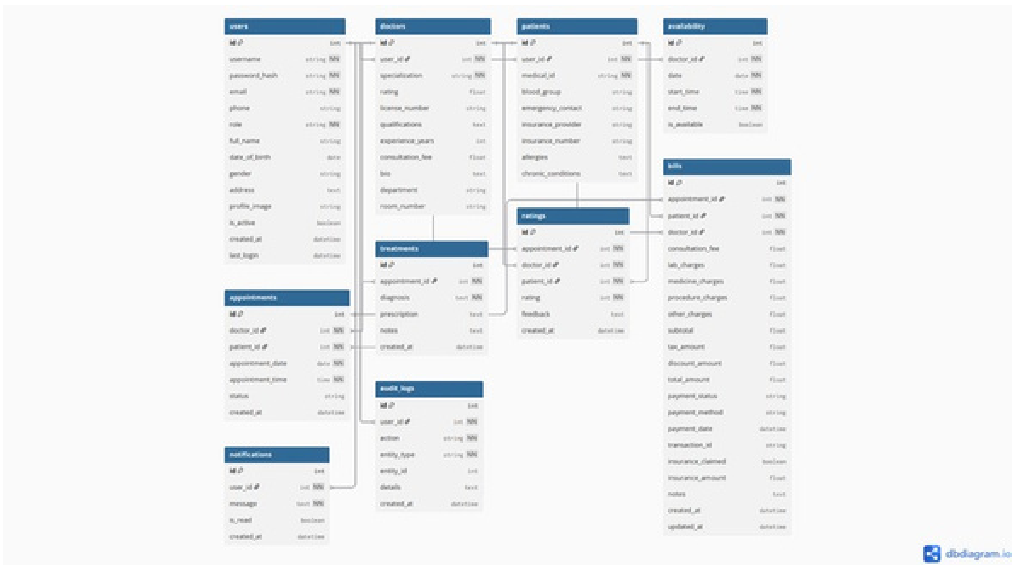

Figure 1: MediCare Hospital Management System - Entity Relationship Diagram showing all tables and relationships. The diagram illustrates the hierarchical structure where User is the central entity with relationships to Doctor and Patient, which then connect to Appointment-related tables (Treatment, Rating, Bill) and Doctor availability management.

6. API Resource Endpoints

Endpoint	Method	Description
/api/doctors	GET	Retrieve list of all doctors with specialization
/api/patients	GET	Retrieve list of all patients
/api/appointments	GET	Retrieve all appointments (admin access)
/api/specializations	GET	Get specializations with doctor counts
/user/register	POST	User registration for patients
/user/login	POST	Authenticate user and create session
/appointments/book	POST	Create new appointment with conflict detection
/admin/dashboard	GET	Access admin analytics and reports
/doctor/availability	GET/POST	Manage doctor availability

7. Architecture and Features

Architecture Overview:

•**app.py** — Main Flask application entry point with configuration and database initialization

•**/models.py** — SQLAlchemy ORM models defining all entities and relationships

•**/routes** — Flask blueprints organized by role (admin.py, doctor.py, patient.py, api.py, shared.py)

•**/templates** — Jinja2 templates organized by role with reusable components

•**/static** — CSS, JavaScript, and client-side logic

Core Features Implemented:

Multi-role authentication (Admin, Doctor, Patient) with session management
Patient self-registration with automatic medical ID generation

Doctor availability management with intelligent time slot system

Smart appointment booking with double-booking prevention and 1-hour slot system

Automatic appointment suggestions prioritizing next 3 days and morning slots

Complete billing system with itemized charges (consultation, lab, medicine, procedures)

Appointment lifecycle management (Booked → Completed → Cancelled)

Post-appointment rating and feedback system (1-5 stars)

Admin analytics dashboard with revenue tracking and performance metrics

Audit logging for compliance and tracking all CRUD operations

Real-time notifications for users

Responsive Bootstrap 5 UI with mobile optimization

8. Video Presentation

https://drive.google.com/file/d/1n_tX89dAS432KhDiY3QKAXG-jysdN6Km/view?usp=sharing