App Dev Project Report

1. Student Details

Name: Kirti Gupta
Roll Number: 21f1000474
Email: 21f1000474@ds.study.iitm.ac.in
About Me: I am a Data Science student at IIT Madras with a strong interest in AI / ML, and Data Analytics. I enjoy working on real-world projects that turn data into meaningful insights and intelligent solutions.

---

2. Project Details

Project Title: Hospital Management System

Problem Statement:

Hospitals need efficient systems to manage patients, doctors, hospital appointments, and treatments. Currently, many hospitals use manual registers or disconnected software, which makes it difficult to manage records, avoid scheduling conflicts, and track patient history.

Approach:

The app is built using Flask with a modular structure for Admin, Doctor, and Patient roles. It uses SQLAlchemy for models, creates the database programmatically, and leverages Jinja2 + Bootstrap for a responsive UI. Chart.js powers the dashboards, while the system supports availability-based appointment booking, role-specific views, CRUD operations, and treatment updates.

---

3. AI/LLM Declaration

I used **ChatGPT (GPT-5)** to assist with specific parts of the development process, mainly for improving clarity, refining patterns, and resolving template-level issues. The areas where AI assistance was used include:

- **SQLite + SQLAlchemy (≈15%)**
  Help in understanding relationship structuring, improving model readability, and resolving migration-related errors.
- **Bootstrap / CSS (≈5%)**
  Styling suggestions, layout clean-ups, and improving responsiveness of tables and cards.
- **Charts / Data Visualization (≈3%)**
  Chart.js configuration patterns, fixing JSON serialization issues in Jinja templates.

- **Testing / Debugging (≈5%)**
  Reading traceback errors and receiving guidance on where potential logical or template mismatches existed.

- **APIs / External Integration (≈2–3%)**
  Limited assistance in structuring optional JSON endpoints.

Overall, the **extent of AI/LLM usage is approximately 30–40%**, largely restricted to **code suggestions, template refactoring, debugging guidance, and documentation formatting**.

> **All core implementation logic, database design, appointment workflows, booking/rescheduling logic, role-based routing, and end-to-end integration were implemented manually.**
> AI was *not* used for automatic code generation of business logic.

---

4. Technologies and Frameworks Used

Technology / Library	Purpose
Flask	Core backend web framework
SQLAlchemy	Object Relational Mapper for SQLite database
Jinja2	Template engine for rendering dynamic HTML pages
Bootstrap 5	Frontend styling and responsive design
Chart.js	Dashboards and status analytics
Flask-Login	User authentication and session management
WTForms	Frontend form validation
SQLite	Lightweight local database for storing user data

---

5. Database Schema / ER Diagram

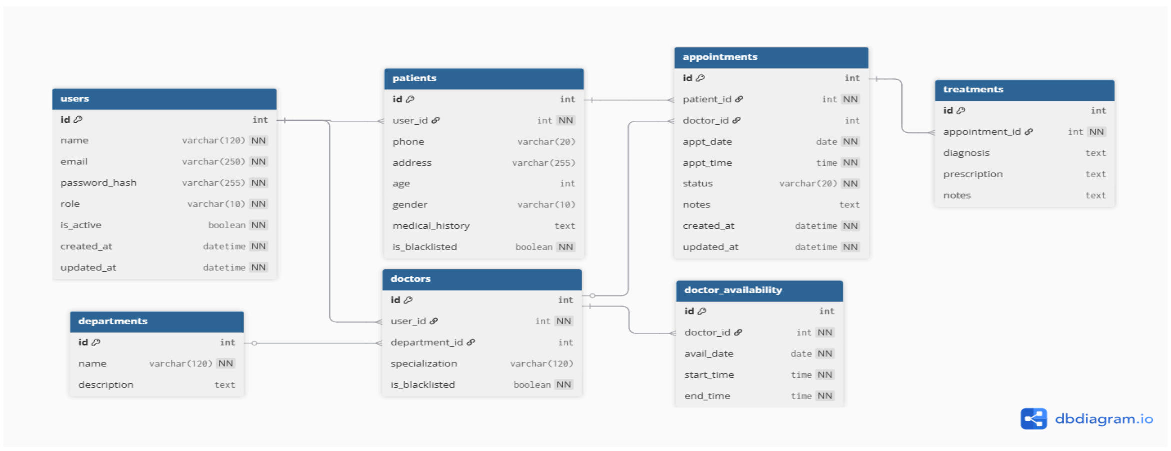
Tables:

1. **User** — stores login and authentication details (id, name, email, password, role, is_blacklisted)
2. **Doctor** — stores doctor-specific data (id, user_id, specialization, department_id, is_blacklisted)
3. **Patient** — stores patient-specific data (id, user_id, phone, address, age, gender, medical_history, is_blacklisted)
4. **Department** — stores hospital departments (id, name, description)
5. **Availability** — stores doctor availability slots (id, doctor_id, avail_date, start_time, end_time)

6. **Appointment** — stores appointment records (id, patient_id, doctor_id, appt_date, appt_time, status)
7. **Treatment** — stores diagnosis and prescription details (id, appointment_id, diagnosis, prescription, notes)

Relationships:

- One-to-One → `User → Doctor`
- One-to-One → `User → Patient`
- One-to-Many → `Doctor → Availability`
- One-to-Many → `Doctor → Appointment`
- One-to-Many → `Patient → Appointment`
- One-to-One → `Appointment → Treatment`
- One-to-Many → `Doctor → Department`



(ER diagram from dbdiagram.io)

---

6. API Resource Endpoints

Endpoint	Method	Role(s) Allowed	Description
`/api/doctors`	GET	admin, doctor, patient	List doctors, optional search by `q`
`/api/doctors/{doctor_id}`	GET	admin, doctor, patient	Get a single doctor by ID
`/api/patients`	GET	patient	Get current logged-in patient profile
`/api/patients/{patient_id}`	GET	admin, patient	Get patient by ID (with role-based checks)
`/api/appointments`	GET	admin, doctor, patient	List appointments for current user, optional `status` filter

`/api/appointments`	POST	patient	Create a new appointment for the current patient
`/api/appointments/{appt_id}`	GET	admin, doctor, patient	Get details of a specific appointment
`/api/appointments/{appt_id}`	PATCH	admin, doctor, patient	Update appointment status (with rules per role)
`/api/appointments/{appt_id}`	DELETE	admin, doctor, patient	Delete/cancel an appointment (with rules per role)

---

7. Architecture and Features (optional)

Architecture Overview:

- **app.py** – main Flask application entry point
- **models** – database models using SQLAlchemy
- **routes** – Flask Blueprints for patient, doctor and Admin
- **/templates** – Jinja2 HTML templates
- **/static** – CSS, JS, and chart visualization files

Implemented Features:

- User registration and login
- Appointment Management System
- Visualization dashboard using Chart.js
- Summary analytics with date filtering
- Responsive UI built with Bootstrap
- Secure password hashing and user session handling
- Blacklisting (doctors and patients) by admin

Additional Features:

- Modal-based delete and blacklist confirmation
- Doctor-wise appointment analytics
- Full patient history for both patient and doctor view
- Departments management by Admin
- Template enhancements like “no data” fallbacks

---

8. Video Presentation

Loom Link:
https://www.loom.com/share/4a4c7298a9d940b3a6090fec265c2d90
(Accessible to all with “View” permission.)