App Dev Project Report

1. Student Details

- **Name:** Kunj Kariya
- **Roll Number:** 23f3002760
- **Email:** 23f3002760@ds.study.iitm.ac.in
- **About Me:** A student of the IIT Madras BS Degree program enthusiastic about full-stack web development. I enjoyed building this HMS project to solve real-world scheduling problems using Python and Flask.

---

2. Project Details

- **Project Title:** Hospital Management System (HMS)
- Problem Statement:
  To design and build a web-based application that digitizes hospital operations. Traditional manual booking leads to double-booking conflicts, lack of patient history records, and inefficient doctor scheduling.
- Approach:
  The app was built using Flask as the backend framework. It utilizes a Role-Based Access Control (RBAC) system to provide distinct dashboards for Admins, Doctors, and Patients. It includes robust validation to prevent scheduling conflicts and uses RESTful APIs for data accessibility.

---

3. AI/LLM Declaration

- I utilized AI tools (Gemini/ChatGPT) primarily as a development assistant to troubleshoot complex errors, particularly regarding database integrity issues.
- They were also helpful in generating the initial CSS/HTML structure for the dashboards, allowing me to focus on the backend logic.
- I estimate the AI usage to be around 15–20%, mostly used for syntax verification and styling suggestions. The core application logic, database schema design, and final testing were all executed manually.

---

4. Technologies and Frameworks Used

Technology / Library	Purpose
Flask	Core backend web framework
SQLAlchemy	Object Relational Mapper (ORM) for SQLite database
Jinja2	Template engine for rendering dynamic HTML pages
Bootstrap 5	Frontend styling and responsive grid layout
Flask-Login	User authentication, session management, and role protection
Flask-RESTful	Building and managing REST API endpoints
SQLite	Lightweight local database for storing hospital data

---

5. Database Schema / ER Diagram

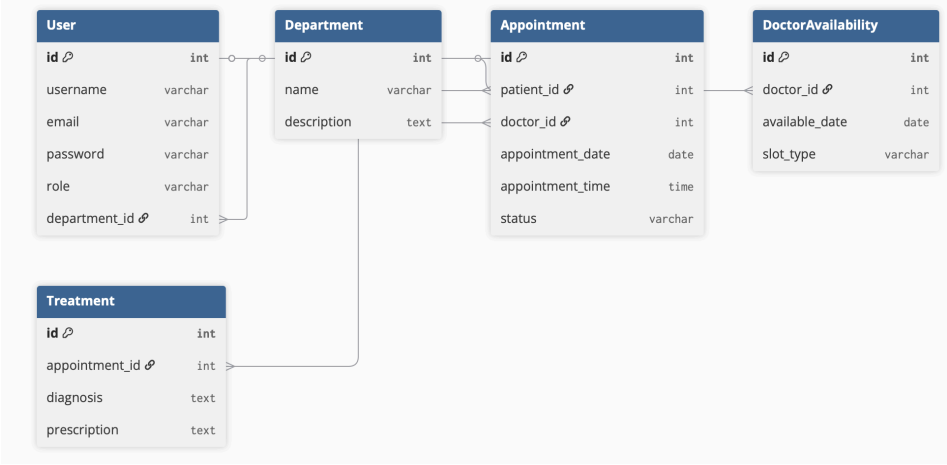
Tables:

- **User:** Stores profile details (id, username, email, password, role, department_id).
- **Department:** Stores medical specialties (id, name, description).
- **Appointment:** Links patients and doctors (id, patient_id, doctor_id, date, time, status).
- **DoctorAvailability:** Logs specific slots doctors are free (id, doctor_id, date, slot_type).
- **Treatment:** Stores medical history (id, appointment_id, diagnosis, prescription).

Relationships:

- **One-to-Many:** Department -> User (Doctors)
- **One-to-Many:** User (Doctor/Patient) -> Appointment
- **One-to-One:** Appointment -> Treatment

ER Diagram:



6. API Resource Endpoints

Endpoint	Method	Description
/api/appointments	**GET**	Fetch all appointments (JSON format)
/api/appointments	**POST**	Create a new appointment with validation
/api/appointments/<id>	**PUT**	Update appointment status or reschedule
/api/appointments/<id>	**DELETE**	Delete/Cancel an appointment record

---

7. Architecture and Features

Architecture Overview:

- app.py: Main application file containing routes, models, and API logic.
- /templates: Contains Jinja2 HTML files (dashboards, forms, base layout).
- /static: Contains CSS files and images.
- site.db: SQLite database file.

Implemented Features:

- **Role-Based Access:** Secure login for Admin, Doctor, and Patient.
- **Conflict Prevention:** Backend logic prevents double-booking of the same doctor slot.
- **Sticky Forms:** Forms retain user input if validation errors occur (e.g., duplicate email).
- **Medical History:** Patients can view past diagnoses and prescriptions.
- **API Integration:** Full CRUD capabilities via Flask-RESTful.

---

8. Video Presentation

- Drive Link: HMS-Presentation.mov