App Dev Project Report

1. Student Details

Name: Sadhana kumari

Roll Number: 23f3002768

Email: 23f3002768@ds.study.iitm.ac.in

I am a college student. I enjoy learning about programming and data science. This project was a lot too fun to make. There were points where I stuck but felt great joy after I tacked those challenges.

---

2. Project Details

Project Title: Hospital Management System (HMS)

Problem Statement:
To design and build a comprehensive web-based hospital management platform that enables seamless appointment booking, doctor availability management, medical history tracking, and role-based access for patients, doctors, and administrators. The system addresses key healthcare management challenges such as appointment scheduling conflicts, doctor availability coordination, patient record tracking, and secure user authentication.

Approach:
The HMS was built using Flask as the backend framework with SQLAlchemy ORM for database management. The system implements a modular architecture with role-based access control (RBAC) supporting three user roles: Admin, Doctor, and Patient. Key features include doctor availability slots (1-hour intervals), appointment booking with conflict resolution, treatment history tracking, and appointment status management (Scheduled, Completed, Cancelled). The design ensures data consistency, prevents double-booking, and maintains comprehensive patient visit records.

---

3. AI/LLM Declaration

I used ChatGPT/Claude to assist with:

Debugging Flask routing issues and SQLAlchemy query optimization

Template refinement and HTML/Jinja2 suggestions

Documentation and error handling patterns

The extent of AI/LLM usage is approximately 20%, limited to debugging, code suggestions, and documentation support. All core logic implementation, database design, feature development, and final integration were completed manually.

---

4. Technologies and Frameworks Used

Technology / Library	Purpose
Flask	Core backend web framework for routing and request handling
SQLAlchemy (Flask-SQLAlchemy)	ORM for database modeling and relationships
Jinja2	Template engine for rendering dynamic HTML
SQLite	Lightweight relational database for data persistence
Bootstrap / HTML / CSS	Frontend styling and responsive UI design
Flask-Login / Session Management	User authentication, role-based access control, and session handling
Python (datetime, time, date)	Date/time utilities for appointment and availability scheduling
Werkzeug	Password hashing for secure authentication

---

5. Database Schema / ER Diagram

Tables:

- **User** — Stores user profile details (id, name, email, password, role, contact, gender, specialization, experience, department_id, **blacklisted**)
- **Department** — Hospital departments (id, name, description)
- **Availability** — Doctor availability slots (id, doctor_id, date, start_time, end_time)
- **Appointment** — Appointment records (id, patient_id, doctor_id, date, time, status, notes)
- **Treatment** — Medical treatment details (id, appointment_id, diagnosis, prescription, notes)

Key Relationships:

- One-to-Many: Department ↔ User (doctors belong to departments)
- One-to-Many: User (Doctor) → Availability (each doctor has multiple availability slots)
- One-to-Many: User (Patient) → Appointment (patient side)
- One-to-Many: User (Doctor) → Appointment (doctor side)
- One-to-Many: Appointment → Treatment (each appointment may have treatment details)

ER Diagram: https://drive.google.com/file/d/1L-B929VNjfk344jLj6T_sZ8pvufbVAUc/view?usp=drive_link

---

6. API Resource Endpoints

Endpoint	Method	Description
`/register`	POST	Register new user (patient, doctor, or admin)
`/login`	POST	Authenticate user and establish session
`/admin_dashboard`	GET	Admin dashboard with appointments and user management
`/doctor_dashboard`	GET	Doctor dashboard showing appointments and patient list
`/patient_dashboard`	GET	Patient dashboard for searching and booking appointments
`/check_availability/<int:doc_id>`	GET	View available doctors and slots for booking
`/book_appointment/<int:slot_id>`	POST	Book an appointment for a doctor's available slot
`/provide_availability`	GET/POST	Doctor provides weekly availability (morning/evening slots)
`/reschedule_appointment/<int:appt_id>`	GET/POST	Patient reschedules existing appointment
`/mark_completed/<int:appt_id>`	POST	Doctor marks appointment as completed
`/cancel_appointment/<int:appt_id>`	POST	Cancel an appointment
`/edit_history/<int:appointment_id>`	GET/POST	Doctor updates treatment details (diagnosis, prescription)
`/patient_history/<int:patient_id>`	GET	View treatment history with doctor and department details
`/admin_patient_history/<int:patient_id>`	GET	Admin views full patient visit history
`/blacklist_doctor/<int:doc_id>`	POST	Admin blacklists a doctor
`/blacklist_patient/<int:patient_id>`	POST	Admin blacklists a patient

---

7. Architecture and Features

Architecture Overview

Implemented Features

Authentication & Authorization:

- Secure user registration and login with password
- Role-based access control (Admin, Doctor, Patient)
- Session-based authentication

- Blacklist feature to restrict user access

Doctor Availability Management:

- Doctors set weekly availability with 1-hour slots (Morning: 8am–12pm, Evening: 4pm–8pm)
- Fixed slot structure prevents scheduling conflicts
- Admin can view and manage all doctor schedules

Appointment Booking System:

- Patients search doctors by department or specialty
- Real-time availability display
- Book appointments for available slots
- Prevent double-booking with database constraints
- Dynamic appointment status management (Scheduled → Completed/Cancelled)

Medical History Tracking:

- Doctors record diagnosis, prescription, and treatment notes
- View appointment history with doctor and department information
- Admin can access full patient history across all doctors
- Treatment details persist after appointment completion

Appointment Management:

- Reschedule existing appointments
- Mark appointments as Completed or Cancelled
- Admin views past appointments (status-based filtering, not date-based)
- Booked slots are locked and cannot be modified

User Management (Admin):

- Add/edit/delete doctors and patients
- View all appointments (upcoming and past)
- Blacklist doctors/patients for policy violations

---

8. Video Presentation

Drive Link:
https://drive.google.com/file/d/16Y_CuVro0Wfo2C_mOrSLRtWy95-UvxQT/view?usp=drive_link

Video Duration: 9 minutes demonstrating video:

- User registration and login workflow
- Admin dashboard features (user management, appointment history)
- Doctor availability setup and appointment management
- Patient appointment booking and history viewing
- Treatment record management

---

9. Key Implementation Highlights

Conflict Resolution:

- Database constraints prevent booking overlapping slots
- Booked appointments are immutable (cannot edit availability for booked times)
- Status-based filtering ensures accurate past/upcoming categorization

Data Integrity:

- Foreign key constraints maintain referential integrity
- Cascading deletes ensure orphaned records are cleaned up
- Transactional commits prevent partial updates

User Experience:

- Intuitive navigation between dashboards
- Clear button labels and status indicators
- Role-specific views prevent unauthorized data access

Scalability Considerations:

- Modular route structure allows easy addition of new features
- ORM-based queries facilitate future indexing and optimization
- SQLite suitable for current scale; upgradeable to PostgreSQL for production

---

10. Testing & Validation

Tested Scenarios:

- User registration with duplicate email prevention

- Login with invalid credentials
- Appointment booking preventing double-booking
- Doctor availability slot management and locking
- Status transitions (Scheduled → Completed → History)
- Role-based access control
- Blacklist functionality

Known Limitations:

- Currently SQLite (suitable for local development; production would use PostgreSQL)
- No real-time notifications (can be added with WebSockets/Celery)
- Email notifications not yet implemented

---

11. Submission Contents

ZIP File includes:

1. **Project_Report.pdf** — This report in PDF format
2. **Code Folder** — Complete project structure
    - `app.py` — Main Flask application
    - `models.py` — Database models
    - `templates/` — All HTML templates
    - `static/` — CSS and assets
    - `requirements.txt` — Python dependencies
    - `README.md` — Instructions to run the project
3. **ER Diagram** — Database schema visualization

---

12. How to Run the Project

Requirements:

- Python 3.8+
- Flask 2.0+
- SQLAlchemy