Hospital Management System — Project Report

Student Details

Name: Abhishek Saini

Roll no: DS23F3001168

Email: 23f3001168@ds.study.iitm.ac.in

Course: Modern Application Development

Project Title: Hospital Management System

Project Details

Problem Statement

Build a full-stack Flask-based **Hospital Management System** that supports 3 different roles:

ADMIN

- Must be pre-created (no admin registration).
- Manage departments, doctors, and patients.
- View all appointments (past and upcoming).
- Search patients/doctors.
- Blacklist/remove users.
- Edit doctor/patient details.

DOCTOR

- View upcoming appointments (week view).
- View all assigned patients.
- Mark appointments as completed/cancelled.
- Add diagnosis, prescription, and notes for each completed appointment.
- Provide weekly availability.
- View full treatment history of each assigned patient.

PATIENT

- Register & login.
- View departments & doctor availability.
- Book/cancel appointments.
- Avoid duplicate booking for the same doctor/time.
- View upcoming appointments + status.
- View complete past history (diagnosis, prescriptions, notes).
- Edit their profile.

ADDITIONAL REQUIREMENTS

- Avoid double booking of the same time slot.
- Dynamic booking status (Booked → Completed → Cancelled).
- Treatment history is maintained permanently.
- Admin & patient must be able to search doctors by specialization/name.

- Admin must be able to search patients by name/ID/contact.
- Complete CRUD operations for all required entities.
- Clean UI using Bootstrap (no external frontend frameworks).
- NO base.html structure required.

Approach to the Problem

I divided the project into 4 major phases:

Phase 1 — Database & Models

I designed relational models for:

- User (for login & roles)
- Doctor
- Patient
- Department
- Appointment
- Treatment
- Availability

Every entity directly matched the required features, especially:

- **Treatment table** for patient history
- **Availability table** for doctor slots
- **Blacklist fields** for admin control

I created relationships such as:

- One department → many doctors
- One doctor → many appointments, availability slots
- One patient → many appointments
- One appointment → one treatment

Phase 2 — Authentication System

I implemented a login system for all roles:

- /login
- /login/admin
- /login/doctor

Admin was auto-created programmatically inside app_context().

Role-based access control was implemented using session["role"].

Phase 3 — Core Functionalities

Each role was developed separately:

- Admin CRUD pages for departments, doctors, patients.
- Appointment system with slot conflict prevention.
- Doctor dashboard showing availability, week schedule, assigned patients.
- Patient dashboard showing departments, availability, upcoming & past appointments.
- Treatment module for storing doctor's notes + diagnosis.

Each functionality was tested individually.

Phase 4 — Frontend & UX

Using pure Bootstrap:

- Clean dashboards
- Form-based CRUD
- Search bars built into list views
- Tables styled with action buttons
- Back buttons added everywhere to maintain navigation

AI / LLM Declaration

I have used **ChatGPT as an assistant for debugging, structuring the project, cleaning code, and ensuring core feature coverage.**

All final project logic, database schemas, Python code, HTML files, and implementation decisions have been written, structured, organized, and tested by me.

I understand the project thoroughly and can explain every line of code during evaluation.

Frameworks & Libraries Used

Backend

- **Python 3.9**
- **Flask**
- **Flask-SQLAlchemy**

- **SQLite3 (database)**

Frontend

- **HTML5**
- **CSS3**
- **Bootstrap 5.3**
- **Bootstrap Icons**

Other:

- **datetime module for date handling**
- **werkzeug (packaged via Flask)**

No external frontend frameworks or JavaScript libraries were used.

ER Diagram (Database)