App Dev Project Report

1. Student Details

Name: Sarthak Bhatia
Roll Number: 23F2004447
Email: 23f2004447@ds.study.iitm.ac.in
About Me: I am a student of BS degree at IIT Madras BS Degree program. I have a deep interest in web application development and data - driven technologies. I enjoy building meaningful applications that could have a realtime impact in people's life through combining learning, analytics and user experience.

---

2. Project Details

Project Title: Hospital Management System

Problem Statement:

Patients in modern healthcare settings often encounter difficulties in booking appointments, accessing their medical history and communicating with healthcare providers due to inefficient manual processes and lack of digital infrastructure. Simultaneously, doctors and administrators struggle with managing schedules, patient records, and departmental operations. This project addresses these challenges through an integrated, role-based hospital management system.

Approach:

This app was built using **Flask** for application back-end, **Jinja2** templating, **HTML**, **CSS** and **Bootstraps** for application front-end. **SQLite** is used for databases.

---

3. AI/LLM Declaration

I used **ChatGPT (GPT-5)** to assist in writing SQLAlchemy model definitions, creating API documentation samples, and improving variable naming consistency. The extent of

AI/LLM usage is around **10–15%**, limited to **code suggestions and documentation formatting**.

---

4. Technologies and Frameworks used

Technology/Library	Purpose
Flask	The main web framework used to build the application.
SQLAlchemy	The database toolkit and Object-Relational Mapper (ORM).
Jinja2	The templating engine used by Flask to render HTML pages.
Bootstrap 5	CSS framework for responsive design and UI components
SQLite	The default relational database engine
WTForms	Flexible forms validation and rendering library.
E-mail Validator	Used for validating email addresses in forms.
Blinker	Provides support for the Flask signaling system.

---

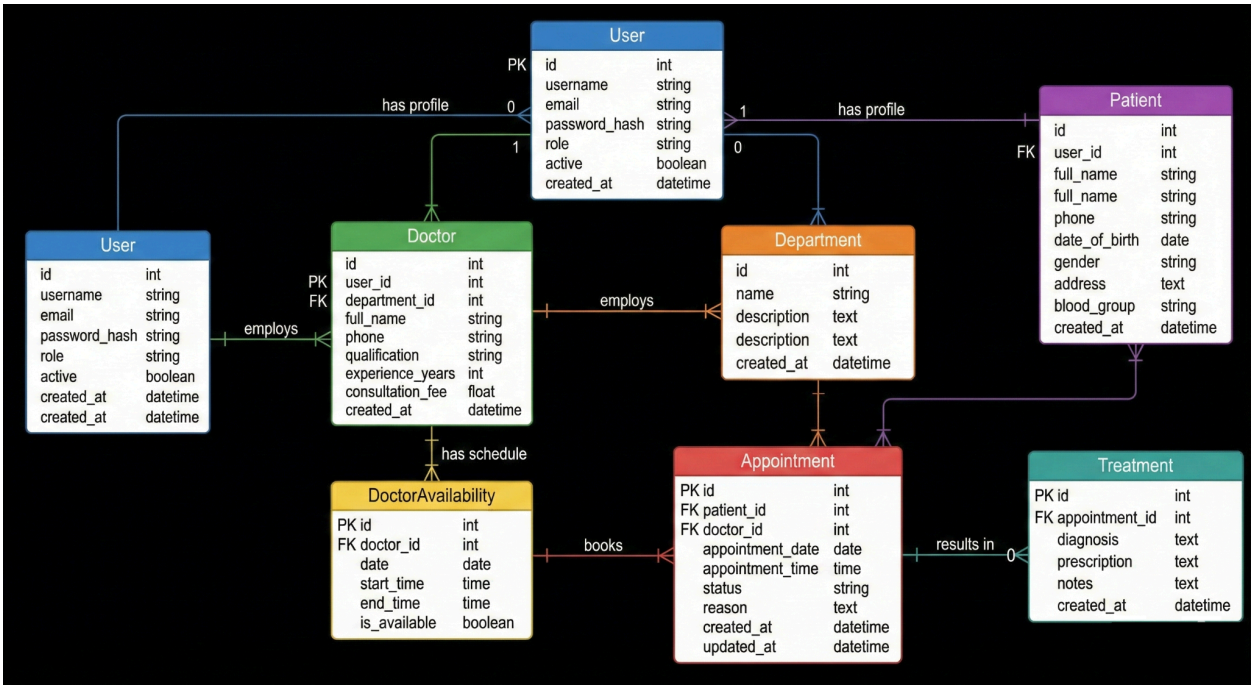
5. DB Schema Design

Tables:

1. **User**: Stores login credentials and role information for all system users (Admins, Doctors, Patients).
2. **Departments Table**: Stores the various medical departments (Cardiology, Neurology, etc.).
3. **Doctors Table**: Stores profile details for doctors, linked to the users table.
4. **Patients Table**: Stores profile details for patients, linked to the users table.
5. **Doctor Availability Table**: Stores the schedule/slots when a doctor is available.
6. **Appointments Table**: Stores the booking details linking a patient and a doctor.
7. **Treatments Table**: Stores medical records for a completed appointment.

Relationships:

One - to - One: User ↔ Doctor
One - to - One: User ↔ Patient
One - to - Many: Department ↔ Doctor
One - to - Many: Doctor ↔ Appointment
One - to - Many: Patient ↔ Appointment
One - to - One: Appointment ↔ Treatment



---

6. API Resource Endpoints

1. Authentication

/login	GET	Login page
/login	POST	Authenticate user
/register	GET	Registration page
/register	POST	Register new page
/logout	GET	End session

2. Admin Resources

/admin/dashboard	GET	Admin Overview
/admin/doctors	GET	List all doctors
/admin/doctor/add	POST	Create new doctor
/admin/patients	GET	List all patients
/admin/appointments	GET	System-wide appointment

3. Doctor Resources

/doctor/dashboard	GET	Doctor’s personal dashboard
/doctor/appointments	GET	List assigned assignments
/doctor/availability	POST	Set Schedule slots
/doctor/appointment/{id}/complete	POST	Submit diagnosis/treatment

4. Patient Resources

/patient/dashboard	GET	Patient's personal dashboard
/patient/doctors	GET	Search/Filter doctors
/patient/book-appointment/{id}	POST	Book a slot
/patient/history	GET	View medical history

---

7. Architecture and Features

Architecture

The project follows the Model-View-Controller (MVC) architectural pattern using the Flask framework. **Controllers** and routing logic are centralized in app.py, which handles HTTP requests and directs business flow. **Models** are defined in models.py using SQLAlchemy, representing the database structure for Users, Doctors, and Appointments. **Templates** (Views) are stored in the templates/ directory and are organized by user role (Admin, Doctor, Patient) to ensure clear separation of concerns. Input validation is managed separately in forms.py using Flask-WTF, while static assets like CSS and images are organized in the static/ folder. This modular structure keeps the codebase clean and maintainable.

Features

Default Features include a robust authentication system with role-based access control, ensuring Admins, Doctors, and Patients see only their specific dashboards. It has a secure authentication and an interactive UI with gradient aesthetics, micro interactions

and instant feedback alerts. The core functionality allows Admins to manage hospital records, Doctors to view their schedules, and Patients to book appointments.

I have implemented a dynamic search functionality allowing patients to filter doctors by name or department. A patient can select a lab diagnostic test which will add up in the cart. Then he/she can further confirm it by paying the total amount(future scope). Users can also book packages for body checkups for which the administration team will contact them later. All these are in the MyCart option. A doctor can see upcoming visits and change schedules as per availability. He/She can add the diagnosis and prescription for a patient and complete the appointment. Access to a patient's previous medical records is made available for informed decision-making.

These are implemented using Flask-Login for session management and SQLAlchemy for database operations. The application is fully responsive using Bootstrap 5, and includes a custom seeding script that populates the system with realistic test data for immediate use.

---

8. Video Presentation

Drive link :
https://drive.google.com/file/d/14JCO1Fs4jFkgD5Ey9-zBKli4y0nsp2Gi/view?usp=sharing