# APP DEV PROJECT REPORT
# ON

## HOSPITAL MANAGEMENT SYSTEM


NAME: - ELIAS HUSSAIN

ROLL NO: - 22F3000155

EMAIL ID :- 22f3000155@ds.study.iitm.ac.in




**IITM Online BS Degree Program,**

**Indian Institute of Technology, Madras, Chennai**

**Tamil Nadu, India, 600036**

**About Me:** I am a BS student at IIT Madras (IITM), currently in the Foundation Level. I am pursuing Diploma in Data Science and a Diploma in Programming at the same time, which are helping me build strong skills in computational thinking, analytics, and software development.

## 1) About Project: Hospital Management System (Flask Framework)

### Introduction

The Hospital Management System (HMS) is a web-based application developed using the Flask framework in Python.
The objective of the project is to simplify hospital operations such as user registration, doctor availability, appointment booking, patient tracking, and treatment management.
The system uses a modular MVC-like structure with separate files for routes, models, templates, and static assets.

## 2) Technologies and Frameworks Used

| Technology | Purpose |
|---|---|
| **Python 3** | Main programming language |
| **Flask** | Web framework for backend development |
| **Flask SQLAlchemy** | ORM for database operations |
| **HTML, CSS** | Frontend user interface |
| **Bootstrap (optional)** | Styling and responsive UI |
| **SQLite / MySQL** | Database storage |
| **Jinja2** | Template rendering engine used by Flask |
| **Werkzeug** | Password hashing and authentication utilities |

## 3) System Architecture

The project follows a **Model–View–Controller (MVC)-like structure**:

- **Models** → Define database tables
- **Views/Templates** → HTML pages displayed to the user
- **Controllers** → Flask routes in app.py
- **Database** → Stores user, appointment, availability, treatment data

**A simple explanation of how the system works:**

- User interacts with web UI (HTML/CSS/Bootstrap)
- Request goes to Flask backend
- Flask processes the request
- SQLAlchemy communicates with database
- Response rendered using Jinja2 templates

**Relationships:**

1) One – Many
2) 2) Many - One

## 4) Database Models (models.py)

### User Table

**user:** user_id (PK), name, email, password, gender, phone, role (admin/doctor/patient), created_at

### Doctor Table

**doctor:** doctor_id (PK), user_id (FK), specialization, qualification, experience_years, availability_status

### Patient Table

**patient:** patient_id (PK), user_id (FK), age, blood_group, address, medical_history

### Appointment Table

**appointment:** appointment_id (PK), patient_id (FK), doctor_id (FK), appointment_date, appointment_time, status (pending/approved/completed/cancelled), notes

### Admin Table

**admin:** admin_id (PK), user_id (FK), access_level

### Prescription Table

**prescription:** prescription_id (PK), appointment_id (FK), doctor_id (FK), patient_id (FK), medicines, instructions, created_at

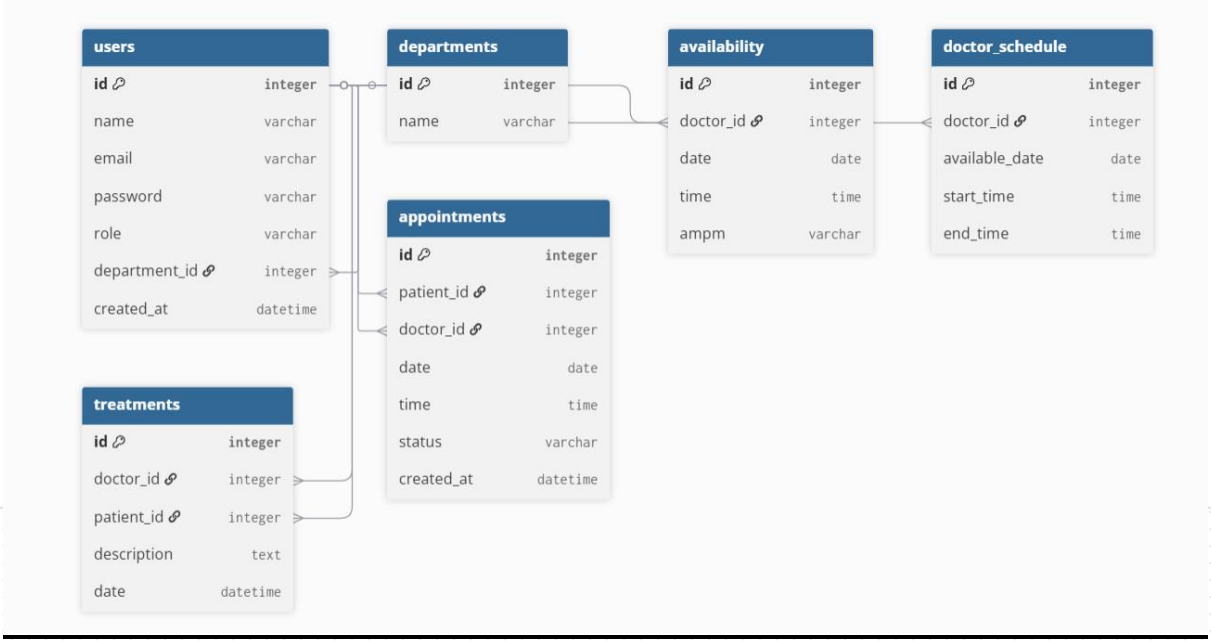
### Medical Report Table

**report**: report_id (PK), patient_id (FK), doctor_id (FK), diagnosis, test_results, report_date

### Describe each module shortly:

User Module: Handles registration, login, sessions, and user role management.
Admin Module: Manages doctors, patients, blacklist status, and all appointments.
Doctor Module: Manages schedules, availability, view appointments, complete/cancel visits.
Patient Module: Allows appointment booking, viewing status, and personal dashboard.
Appointment Module: Handles booking logic, slot assignment, and appointment status updates.
History Module: Stores and updates patient medical records like tests, diagnosis, and prescriptions.

## ER Diagram:




---

## ROUTES

### Main Routes

- / → Loads homepage (index.html)
- /register/patient (GET, POST) → Patient registration: store name, email, password, gender, address
- /login (GET, POST) → User login (admin/doctor/patient redirect based on role)
- /logout → Clears session and logs out

### Admin Routes

- /admin → Admin dashboard showing all doctors, patients, and appointments
- /search?q=… → Admin search for doctors & patients by name/email
- /admin/add_doctor (GET, POST) → Add doctor with name, email, password, specialization, department
- /admin/edit_doctor/<id> → Edit doctor details
- /admin/delete_doctor/<id> → Delete a doctor
- /admin/toggle_blacklist/<doctor_id> → Blacklist/unblock a doctor
- /admin/edit_patient/<id> → Edit patient details
- /admin/delete_patient/<id> → Delete patient
- /admin/toggle_blacklist_patient/<id> → Blacklist/unblock a patient

- /admin/appointment/complete/<appointment_id> → Mark appointment as completed
- /admin/patient_history/<patient_id> → View full appointment history of patient

**Doctor Routes**

- /doctor → Doctor dashboard with booked appointments + available slots
- /doctor/add_availability (GET, POST) → Add availability slot (old Availability model)
- /doctor/complete/<appointment_id> → Mark appointment completed
- /doctor/cancel/<appointment_id> → Cancel appointment
- /doctor/add_schedule (GET, POST) → Add schedule (date + time) into DoctorSchedule

**Patient Routes**

- /patient → Patient dashboard showing all booked appointments
- /patient/book (GET, POST) → Book a slot from DoctorSchedule for next 7 days
- /patient/appointment/cancel/<appointment_id> (POST) → Patient cancels appointment

**AI / LLM Usage Declaration**

- I used ChatGPT(GPT-5) to assist ijn writing SQLAlchemy model definitions. In accordance with institutional guidelines on the responsible use of Artificial Intelligence (AI) and Large Language Models (LLMs), I confirm that AI assistance was utilized only to the extent of **approximately 10–15%** limited to code suggestions and documentation formatting.
- All final implementation logic, debugging and integration were done manually.

Videolink:
https://drive.google.com/file/d/1CfaQqBY0T3ZfKYi5m6u0h2Afj55iYhs2/view?usp=drive_link