Author

Rayyan Mirza
24F2005955
24F2005955@ds.study.iitm.ac.in
I work at a start-up as a software developer, we mainly work on java. This application has been a new challenge for me as I had to learn a new framework ‘Flask’

Project Statement : Create a Hospital Management System using python+flask

Description

Firstly I created a rough er-model based on the information given in App Dev I project Doc, From then on I proceeded to create sqlalchemy models and the rest of the routes and template From there on.

Component	AI-Usage %
Flask (App + API),SQLite + SQLAlchemy , HTML + Jinja2,Authentication (Login/Auth),Admin CRUD Logic	10% (defining models), 30% (creating templates)
Bootstrap+css	5%
Appointment Booking logic	5%

Technologies used

Python, Flask, SqlAlchemy, HTML, Javascript, Css, Sqlite3

- Python : Used as a main language for the project
- Flask : A python based web framework, it provides methods for processing requests, rendering templates.
- SqlAlchemy : A python ORM that we are using to query, insert, update, delete data from our sqlite3.
- HTML : HTML is primarily used to create templates
- Javascript : primarily used for enhancing templates
- Css : primarily used for enhancing templates
- Sqlite3 : provides database for our application

Database Schema Design

1. Department

Column	Type	Notes
id	Integer	PK
name	String(120)	Unique, required

description	Text	Optional
doctors_count	Integer	Optional
doctors	Relationship	Doctor (1-to-many)

2. Admin

Column	Type	Notes
id	Integer	PK
username	String(80)	Unique, required
email	String(120)	Unique, required
password_hash	String(255)	Required
created_at	DateTime	Default: now
status	Enum(StatusEnum)	Default: active
role	Enum(UserRole)	Default: admin

3. Doctor

Column	Type	Notes
id	Integer	PK
name	String(120)	Required
email	String(120)	Unique, required
password_hash	String(255)	Required
phone	String(20)	Optional
department_id	Integer	FK → department.id
bio	Text	Optional
years_of_experience	Integer	Optional
status	Enum(StatusEnum)	Default: active

role	Enum(UserRole)	Default: doctor
created_at	DateTime	Default: now
schedules	Relationship	DoctorSchedule (1-to-many)
time_offs	Relationship	DoctorTimeOff (1-to-many)
appointments	Relationship	Appointment (1-to-many)

4. Patient

Column	Type	Notes
id	Integer	PK
name	String(120)	Required
email	String(120)	Unique, required
password_hash	String(255)	Required
age	Integer	Optional
gender	String(10)	Optional
phone	String(20)	Optional
address	Text	Optional
status	Enum(StatusEnum)	Default: active
role	Enum(UserRole)	Default: patient
created_at	DateTime	Default: now
notes	Text	Optional
appointments	Relationship	Appointment (1-to-many)

5. DoctorSchedule

Column	Type	Notes
id	Integer	PK

doctor_id	Integer	FK → doctor.id
weekday	Integer	0–6 (Mon–Sun)
start_time	Time	Required
end_time	Time	Required
max_patients	Integer	Optional
doctor	Relationship	Doctor
UniqueConstraint	doctor_id, weekday, start_time, end_time	Avoid duplicate schedule blocks

6. DoctorTimeOff

Column	Type	Notes
id	Integer	PK
doctor_id	Integer	FK → doctor.id
date	Date	Required
start_time	Time	Optional
end_time	Time	Optional
reason	String(255)	Optional
doctor	Relationship	Doctor
UniqueConstraint	doctor_id, date, start_time, end_time	

7. Appointment

Column	Type	Notes
id	Integer	PK
patient_id	Integer	FK → patient.id
doctor_id	Integer	FK → doctor.id
appointment_start	DateTime	Required

appointment_end	DateTime	Required
status	Enum(StatusEnum)	Default: booked
reason	Text	Optional
created_at	DateTime	Default: now
last_updated_at	DateTime	Auto-update
patient	Relationship	Patient
doctor	Relationship	Doctor
treatment	Relationship	Treatment (1-to-1)
UniqueConstraint	doctor_id, appointment_start	

8. TimeSlot

Column	Type	Notes
id	Integer	PK
doctor_id	Integer	FK → doctor.id
start	DateTime	Required
end	DateTime	Required
appointment_id	Integer	FK → appointment.id, nullable
doctor	Relationship	Doctor
appointment	Relationship	Appointment
UniqueConstraint	doctor_id, start	

9. Treatment

Column	Type	Notes
id	Integer	PK
appointment_id	Integer	FK → appointment.id

diagnosis	Text	Optional
prescription	Text	Optional
notes	Text	Optional
treatment_date	DateTime	Default: now
appointment	Relationship	Appointment

To keep the system modular and easy to update, we have ensured only high level entities with relations.

API Design Not done completely

Architecture and Features

The Hospital Management System is organized into a clean and modular structure.
The application entry point resides in **app.py**, located at the root of the project. The core components of the system are separated into dedicated directories to ensure clarity, scalability, and maintainability.

The **models.py** file contains all the SQLAlchemy ORM models that define the database schema, including users, doctors, patients, appointments, schedules, departments, and treatments.
The **routes** directory contains the controller blueprints that implement CRUD operations and business logic for admin management, doctor and patient management, scheduling, appointment booking, and treatment history.
The **templates** directory provides all HTML templates required for UI pages such as dashboards, forms, appointment calendars, doctor listings, and patient details.

Currently, the project includes a fully functional hospital workflow:
Patients can register and manage their profile, doctors can maintain availability schedules and time-off, and administrators can manage all core entities including departments, doctors, patients, schedules, and appointments. The system also supports appointment booking, automatic conflict prevention, treatment recording, and optional time-slot generation for upcoming days.

The project can be further enhanced by integrating real-time availability views, advanced analytics on the admin dashboard, appointment reminders, a calendar-based booking UI, and features such as prescription download, doctor performance metrics, or integration with telemedicine modules.