Author

NAGAVENGADESHWARAN S
24f1000802
24f1000802@ds.study.iitm.ac.in

I am a student pursuing the IIT Madras BS Degree program, specializing in application development. This project demonstrates my understanding of full-stack web development using Flask framework.

AI/LLM usage

I implemented most important segments of the project independently, including the core business logic for appointment management, user authentication flow, database model relationships, role-based access control, and validation rules. The AI was primarily used for syntax reference, Bootstrap styling recommendations, and template structure suggestions. All critical features such as preventing duplicate appointments, managing appointment statuses, handling medical records, and implementing the three-tier user system were designed and coded by me.

Overall, the LLM helped me approximately 20% in this project, mainly in accelerating development through code suggestions and debugging assistance, while I was responsible for the architecture, feature implementation, and testing.

Description

This project is a Hospital Management System web application that enables three types of users (Admin, Doctor, and Patient) to manage hospital operations including doctor profiles, patient registrations, appointment scheduling, and medical record management. The system provides role-based access control where admins manage doctors and patients, doctors handle appointments and treatments, and patients can book appointments and view their medical history.

Technologies used

Backend:

- **Flask 3.0.0** - Web framework for routing and backend logic
- **Flask-SQLAlchemy 3.1.1** - ORM for database operations
- **Flask-Login 0.6.3** - User authentication and session management
- **Werkzeug 3.0.1** - Password hashing and security
- **SQLite** - Relational database for data storage

Frontend:

- **Jinja2 3.1.2** - Template engine for dynamic HTML
- **Bootstrap 5.3.0** - Responsive UI framework
- **Font Awesome 6.4.0** - Icon library
- **HTML5/CSS3** - Markup and styling

Purpose:

Flask provides a lightweight framework for rapid web development. SQLAlchemy simplifies database operations through ORM. Flask-Login handles authentication seamlessly. Bootstrap ensures responsive design across devices. SQLite offers portability without requiring separate database server setup, making the application easy to deploy and test locally.

DB Schema Design

1. **Admin** - id, username, password_hash, email, created_at
2. **Department** - id, name, description (One-to-Many with Doctor)
3. **Doctor** - id, username, password_hash, name, email, phone, department_id, experience_years, qualification, is_active, created_at
4. **Patient** - id, username, password_hash, name, email, phone, date_of_birth, gender, address, blood_group, is_active, created_at
5. **DoctorAvailability** - id, doctor_id, date, start_time, end_time, is_available
6. **Appointment** - id, patient_id, doctor_id, appointment_date, appointment_time, status, reason, created_at
7. **Treatment** - id, appointment_id, diagnosis, prescription, notes, treatment_date (One-to-One with Appointment)

Relationships: Department → Doctors (1:N), Doctor → Appointments (1:N), Patient → Appointments (1:N), Appointment → Treatment (1:1)

Design Rationale: Schema follows 3NF for data normalization. Separate user tables maintain role-specific attributes. Foreign keys ensure referential integrity. Soft deletion via `is_active` flag preserves historical data. One-to-one Appointment-Treatment relationship ensures unique medical records per visit.

API Design

API Implementation: No REST APIs were implemented in this project as they were listed as optional features in the requirements. The application uses traditional server-side rendering with Flask routes and Jinja2 templates.

Request Handling: All user interactions are handled through HTML forms with POST/GET requests directly to Flask route controllers. Data operations are performed server-side using SQLAlchemy ORM, and responses are rendered as HTML pages.

Future API Scope: If implemented, RESTful endpoints would include user authentication, doctor/patient/appointment CRUD operations, and treatment record management, returning JSON responses for potential mobile app or SPA integration.

Architecture and Features

Project Architecture:

MVC pattern implementation:

- **Models** (`models.py`) - SQLAlchemy database models with password hashing
- **Controllers** (`app.py`) - Flask routes organized by role with authentication decorators
- **Views** (`templates/`) - Jinja2 templates in role-based folders (admin/, doctor/, patient/)
- **Static** (`static/css/`) - Custom styling
- **Config** (`config.py`) - Database and app settings

Core Features:

Admin: Dashboard with statistics, manage doctors (add/edit/delete), manage patients, view all appointments, search functionality

Doctor: Dashboard with 7-day appointments, complete appointments with diagnosis/prescription/notes, view patient history, manage availability schedule

Patient: Self-registration, browse departments/doctors, book/cancel appointments (7-day window), view medical history and treatment records, edit profile

Additional: Duplicate appointment prevention, role-based access control, secure authentication, status management (Booked/Completed/Cancelled), responsive UI, form validation

Video

https://drive.google.com/file/d/10KqLyqqGnuEraXIoWvV8WKFFvPuYRfrq/view?usp=sharing