Author

Kaushik Harsha
kaushikharsha2020@gmqil.com
I am a Data Science Enthusiast, currently pursuing B.Tech in Computer Science & Engineering. I also enjoy full stack web development.

AI/LLMs Usage

AI/LLM tools were used ~20% to assist with styling, backend jobs, caching, etc.

Description

HealthLink, the Hospital Management System is a Flask-based full-stack application designed to streamline hospital workflows such as patient registration, appointment handling, doctor availability, and treatment record management. The system supports three user roles: Admin, Doctor, and Patient, each with distinct permissions and dashboards. The backend is powered by Flask, SQLite (via SQLAlchemy ORM), Redis caching, and Celery-based asynchronous task processing for reminders, monthly reports, and CSV exports. The frontend is developed using Vue.js and Bootstrap to ensure a responsive and interactive user interface.

Technologies Used

- Python – Main Backend handler using flask and other backend jobs
- Flask – Backend Server
- Flask-SQLAlchemy – Backend Database Management
- Flask-Security – Authentication and RBAC
- SQLite3 - Database
- Redis – For Caching and running celery tasks
- Celery – Running asynchrounous tasks using beats and workers on redis server
- Vue.js – Main Frontend Framework
- Bootstrap – For styling and responsive design(PWA)
- HTML – For templating monthly PDF reports for doctors
- JavaScript – For client side services
- WSL2 – For running the Redis server and Celery workers(Backend)

DB Schema Design

The database schema consists of the following tables:

• users (id, name, email, password, fs_uniquifier, active)
• roles (id, name, description)
• user_roles (user_id, role_id)
• specializations (id, name, description)
• doctors (id, u_id, specialization_id, availability, contact_number)
• patients (id, u_id, age, gender, contact_number, address, emergency_contact)
• appointments (id, doctor_id, patient_id, appointment_date, appointment_time, status)
• treatments (appointment_id, diagnosis, tests_done, prescription, notes)

Design Rationale:

The schema follows a normalized relational model ensuring clear relationships between users, doctors, patients, appointments, and treatments. Cascade rules are used for maintaining integrity across dependent entities.

API Design:

• GET - /users
• GET - /users/<id>
• GET - /doctors
• GET - /doctor?id=<id>
• GET - /doctor?uid=<uid>
• GET - /patients
• GET - /patient?id=<id>
• GET - /patient?uid=<uid>
• GET - /appointments
• GET - /specialization?id=<id>
• GET - /specialization?name=<name>

• POST - /auth/login
• POST - /auth/register
• POST - /doctors
• POST - /appointments
• POST - /treatments
• POST - /specializations

• PUT - /users/<id>

• PATCH /users/<id>
• PATCH /doctors/<id>
• PATCH /patients/<id>
• PATCH /appointments/<id>

• DELETE /users/<id>
• DELETE /patients/<id>
• DELETE /appointments/<id>

Features Implemented

• Role-based login for Admin, Doctor, and Patient.
• Admin dashboard displaying total doctors, patients, and appointments.
• Doctor dashboard showing upcoming appointments and patient lists.
• Patient dashboard showing available doctors, booking interface, and history.
• Doctor availability management for 7 days.
• Appointment booking, cancellation, and rescheduling logic.
• Treatment recording with diagnosis, notes, and prescriptions.
• Redis caching of common queries for speed optimization.
• Celery-based scheduled reminders and monthly reports.
• CSV export background job for patient treatment history.

Video

https://drive.google.com/file/d/1Ri6pbtvTVafEx6gPJZcFDz353iEkWJca/view?usp=drive_link