Medaleon - Hospital Management System (HMS-V2)
Modern Application Development - 2

Author: Dharsan Arunkumar V P
Roll No: 23F3000446
Email: 23F3000446@ds.study.iitm.ac.in

About Me

I am a Dual Degree student pursuing BE CSE while completing this degree programme. I am deeply passionate about technology, design, and creative problem-solving. I enjoy building clean and functional user experiences, combining engineering with creativity. My interests include UI/UX design, full-stack development, AIML, AR/VR, and creative arts such as music and storygraphing.

My long-term goal is to develop & create impactful digital products at the intersection of technology, creativity and design.

Introduction

Hospitals manage a large number of patients, doctors, and appointments every day. Without a structured digital system, issues such as double booking, missing treatment history, inefficient coordination, and poor visibility for stakeholders quickly arise.

This project implements **Hospital Management System V2 (HMS-V2)** as part of **Modern Application Development II** (MAD-2). It is an upgraded and redesigned version of my MAD-1 project. The system now uses a **Flask REST API backend**, **VueJS frontend**, and a properly structured SQLite database.

Three distinct roles are supported:

- **Admin** – manages doctors, views appointments, monitors activity
- **Doctor** – manages assigned appointments and records treatments
- **Patient** – registers, books/cancels appointments, views medical history

The system is designed to be clean, efficient, and reflective of real-world hospital workflows. This system extends all **MAD-1 milestones** by adding background jobs, Redis caching, and a fully separated SPA architecture

AI / LLM Usage Disclosure

I used LLMs to understand specific concepts in Flask, SQLAlchemy, Vue Router, frontend-backend integration, and to improve documentation structure based on the requirements and how I can accomplish it in simpler techniques.

All final code, integration, UI implementation, debugging, and testing were performed manually by me. (Around 20-25% of Assistance)

Technologies Used

Backend (Flask API)	Frontend (Vue 3 + Vite)
● Python 3 ● Flask + SQLAlchemy (ORM, SQLite DB) ● bcrypt (password hashing) ● Flask-CORS (frontend API access) ● PWA basics (manifest + service worker) ● Fake background simulation	● Vue 3 (Composition API) ● Vue Router (role-based route guards) ● Axios (HTTP requests) ● Bootstrap 5 (UI layout and components) ● Custom theme styling and loading screen

Progressive Web App (PWA) Characteristics

The frontend includes basic Progressive Web App features through the existing `manifest.json`, application icons, and the Vite service worker integration. This allows the system to be installable on supported devices and provides an app-like experience through standalone display mode and controlled asset caching. While advanced offline caching is not implemented, the essential PWA scaffolding is present.

Database Design
The database uses SQLite with SQLAlchemy ORM.

User Table
- Stores username, password hash, name, email, and role
- Single table for all roles: admin, doctor, patient

Doctor Table
- One-to-one link to User
- Stores specialization

Patient Table = One-to-one link to User

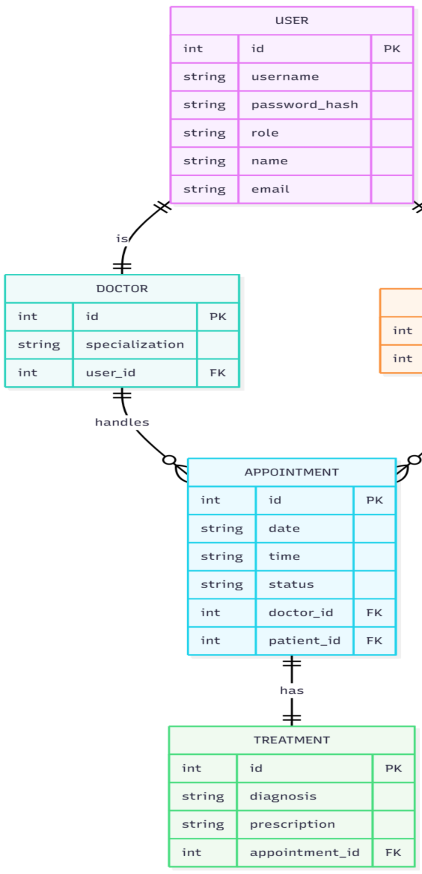
Appointment Table

- Links doctor and patient
- Stores date, time, and status
- Status values: Booked, Completed, Cancelled
- Validation prevents double booking and invalid dates

Treatment Table

- One-to-one link to Appointment
- Records diagnosis and prescription

ER Diagram (Generated using Mermaid Code)



API Design

All backend endpoints are under “/api/”.

Authentication

- POST /api/register – patient registration
- POST /api/login – login with JWT

- GET /api/me – get current user details

Admin Endpoints

- GET /api/admin/summary
- GET /api/admin/doctors
- POST /api/admin/doctors
- PUT /api/admin/doctors/<id>
- DELETE /api/admin/doctors/<id>
- GET /api/admin/appointments
- GET /api/admin/run-simulation-task
- GET /api/admin/simulation-task-status

Doctor Endpoints

- GET /api/doctor/appointments
- PUT /api/doctor/appointments/<id>

Patient Endpoints

- POST /api/patient/appointments
- GET /api/patient/appointments
- PUT /api/patient/appointments/<id>

All protected endpoints require valid JWT and role checks.

Application Architecture

The system follows a two-tier architecture:

Client (Vue 3 SPA) - > Flask REST API -> SQLite Database

Backend Layers	**Frontend Layers**
● Routing and controllers	● Router-based navigation
● Business logic (validation, role checks, booking rules)	● State persistence via localStorage
● Database models and queries	● Axios-based API calls
	● Component-based dashboards

Features Implemented

Admin

- Pre-created admin user on first run
- View summary metrics
- Manage doctors (add, edit, remove)

- View all appointments
- View list of unique patients
- Run background simulation
- View simulation status with timestamp
- Status-based filtering

Doctor

- View all assigned appointments
- Update status of appointments
- Record diagnosis and prescription
- View treatment history of patients

Patient

- Register and login
- Book appointments
- Prevent double-booking
- Cancel or reschedule appointments
- View appointment and treatment history

To align with the MAD-2 requirement for asynchronous behavior, the project includes a **simulated background task system.**
The "Simulation Runner" in the Admin Dashboard mimics the execution of backend jobs such as auto-cancellations, reminders, or activity summaries.
While Redis/Celery are not used in this implementation, the simulation accurately demonstrates the workflow, status response, timestamp updates, and backend interaction expected from async processing in real systems.

Testing and Validation

Testing was carried out across all major workflows:

- JWT authentication validation
- Unauthorized access correctly returning 401/403
- Role-based navigation guards
- Double-booking prevention logic
- Appointment status transitions
- Doctor edit/remove functionality
- Admin summary data validation
- Simulation run and status update
- Persistence across refresh via localStorage

All the primary functional requirements of the system have been validated.

How to Run the Project

Backend (Flask)	Frontend (Vue 3)
cd backend python -m venv venv venv/Scripts/activate pip install -r requirements.txt python app.py Runs at: http://localhost:5000	cd frontend/hmsvue npm install npm run dev Runs at: http://localhost:5173

Screenshots

(Added in the Reports Folder)

Conclusion

The "Medaleon" HMS-V2 app implements a complete workflow for hospital scheduling and treatment records with separate dashboards for admins, doctors, and patients. The system provides secure authentication, validated booking operations, treatment tracking, role isolation, and a modern Vue-based interface.

Planned future improvements include:

- Email/SMS appointment reminders
- Pagination and advanced filtering
- Dark mode interface
- Multi-hospital/branch support
- Analytical dashboards and charts

This project demonstrates full-stack skills across backend design, frontend UI development, routing, state management, and database integration.