HOSPITAL MANAGEMENT SYSTEM V2

1. Student Details

Name: Sripathi Dhruv Reddy
Roll Number: 23f3001736
Email: 23f3001736@ds.study.iitm.ac.in
About Me: I am a passionate developer with strong interest in full-stack engineering, system design, and real-world problem solving. I enjoy building production-ready applications that combine clean architecture, performance, and great user experience. I am particularly fascinated by backend engineering, distributed systems, automations, and scalable web applications.

2. Project Details

Project Title: Hospital Management System - V2

Problem Statement:Hospitals often rely on manual records or fragmented software systems, resulting in scheduling conflicts, inaccessible patient history, and inefficiencies in doctor-patient interactions. This project aims to develop a robust web application that streamlines hospital operations through role-based access for Admins, Doctors, and Patients.

Approach:The system was built using Flask for backend APIs and VueJS for the frontend, ensuring a modular and responsive architecture. Redis and Celery were used for task queuing and background jobs. The database schema was implemented using SQLAlchemy ORM with programmatic initialization.

3. AI / LLM Usage Declaration

I used ChatGPT (GPT-5) during this project primarily as a support tool to clarify concepts, validate design approaches, and enhance the overall quality of the implementation. AI assistance was used occasionally for refining documentation, generating small boilerplate patterns, improving UI structure and styling, and resolving minor script-level issues. I also referred to AI for guidance while setting up Redis caching, Celery background tasks, and a few additional optional functionalities.
The overall AI involvement is estimated at **25–35%**, limited to suggestions, explanations, and incremental improvements.
All core development — including backend API logic, database schema design, appointment flow, Vue.js frontend pages, authentication mechanisms, Celery job execution, Redis caching logic, and complete debugging was fully designed, implemented, and executed by me.

4. Technologies and Frameworks Used

Technology / Library	Purpose
Flask	Core backend framework used to build REST APIs, handle routing, authentication, and business logic
SQLite	Lightweight local database for storing users, doctors, patients, appointments, and treatments
SQLAlchemy	ORM used to define models, relationships, and interact with the SQLite database programmatically
Jinja2	Used only for generating email templates and monthly activity report HTML inside Celery tasks
Bootstrap 5	Provides responsive design, layout structure, and styling for frontend pages

Technology / Library	Purpose
Vue.js (Vue 3)	Frontend JavaScript framework for building reactive Admin, Doctor, and Patient dashboards
Vue Router	Handles client-side navigation and routing between role-based screens
Axios	Used for asynchronous communication between the Vue frontend and Flask backend APIs
Flask-JWT-Extended	Provides secure authentication using JWT tokens for role-based access control
Redis	Used for caching frequently accessed data (doctor availability, search results) and improving API performance
Celery	Background job scheduler for daily reminders, monthly reports, and CSV export tasks
Flask-Mail	Sends automated emails such as appointment reminders and monthly doctor reports
JavaScript (within Vue components)	Manages interactive UI logic, form handling, and dynamic rendering
HTML / CSS	Structural markup and custom styling within Vue single-file components
Python (Core)	Primary programming language for backend APIs, background tasks, and application logic

5. DB Schema Design

Tables:

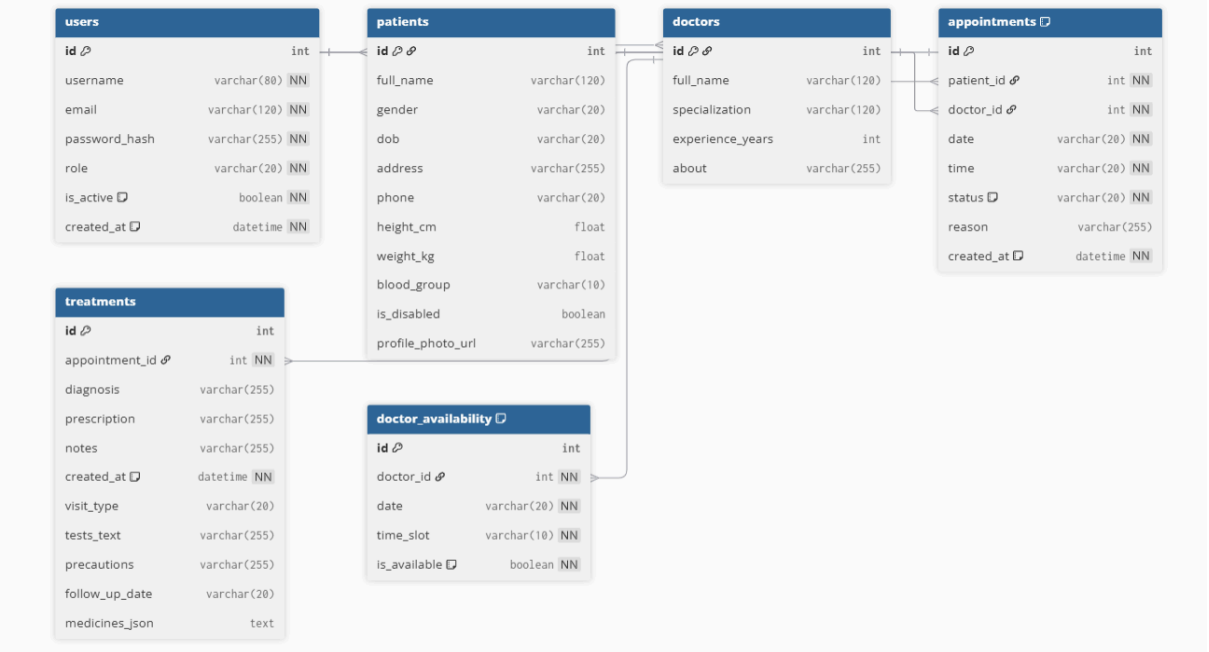
- **User** — stores all system user accounts (id, username, email, password_hash, role, is_active, created_at)
- **Patient** — stores patient profile information (id, full_name, gender, dob, address, phone, height_cm, weight_kg, blood_group, is_disabled, profile_photo_url)
- **Doctor** — stores doctor profile and specialization details (id, full_name, specialization, experience_years, about)
- **Appointment** — stores booking and visit details (id, patient_id, doctor_id, date, time, status, reason, created_at)
- **Treatment** — stores medical diagnosis and treatment records (id, appointment_id, diagnosis, prescription, notes, visit_type, tests_text, precautions, follow_up_date, medicines_json, created_at)
- **DoctorAvailability** — stores per-day, per-slot availability of doctors
  (id, doctor_id, date, time_slot, is_available)

Relationships:

- **One-to-One → User → Patient**
  (Each user with role *patient* has exactly one patient profile)
- **One-to-One → User → Doctor**
  (Each user with role *doctor* has exactly one doctor profile)
- **One-to-Many → Patient → Appointment**
  (A patient can have many appointments)
- **One-to-Many → Doctor → Appointment**
  (A doctor can have many appointments)
- **One-to-Many → Appointment → Treatment**
  (Each appointment can have multiple treatment records)

- **One-to-Many → Doctor → DoctorAvailability**
  (A doctor has many availability slots per date)

ER Diagram: **HMS_V2_ER_Diagram.png**



6. API Resource Endpoints

Method	Endpoint	Description
POST	`/api/auth/register`	Register user
POST	`/api/auth/login`	Login & get JWT
GET	`/api/auth/me`	Current user details
GET	`/api/admin/stats`	Dashboard stats
GET	`/api/admin/reports-analytics`	Analytics
GET	`/api/admin/monthly-report`	Monthly report
POST	`/api/admin/doctors`	Create doctor
GET	`/api/admin/doctors`	List doctors
GET	`/api/admin/doctors/<id>`	Get doctor details
PUT	`/api/admin/doctors/<id>`	Update doctor
DELETE	`/api/admin/doctors/<id>`	Delete doctor

GET	`/api/admin/patients`	List patients
POST	`/api/admin/patients`	Create patient
GET	`/api/admin/patients/<id>`	Get patient details
PUT	`/api/admin/patients/<id>`	Update patient
DELETE	`/api/admin/patients/<id>`	Delete patient
GET	`/api/admin/appointments`	List all appointments
PUT	`/api/admin/appointments/<id>/status`	Update appointment status
GET	`/api/doctor/dashboard-summary`	Doctor dashboard summary
GET	`/api/doctor/profile`	Get doctor profile
PUT	`/api/doctor/profile`	Update doctor profile
GET	`/api/doctor/stats`	Doctor statistics
GET	`/api/doctor/monthly-report`	Doctor monthly report
GET	`/api/doctor/appointments`	Doctor appointments
POST	`/api/doctor/appointments/<id>/status`	Update appointment status
POST	`/api/doctor/appointments/<id>/treatment`	Add treatment details
GET	`/api/doctor/appointments/<id>/treatment`	Get treatment details
GET	`/api/doctor/availability`	Doctor availability for week
POST	`/api/doctor/availability/toggle`	Toggle slot availability
POST	`/api/doctor/availability/bulk`	Bulk availability update

GET	`/api/doctor/my-patients`	Patients treated by doctor
GET	`/api/doctor/patient-history`	History of a particular patient
GET	`/api/patient/profile`	Get patient profile
PUT	`/api/patient/profile`	Update patient profile
GET	`/api/patient/doctors`	List active doctors
GET	`/api/patient/available-slots`	Get available slots
GET	`/api/patient/appointments`	Get patient appointments
POST	`/api/patient/appointments`	Book appointment
GET	`/api/patient/appointments/<id>`	Appointment details
POST	`/api/patient/appointments/<id>/cancel`	Cancel appointment
GET	`/api/patient/history`	Patient medical history
GET	`/api/patient/history/export-csv`	Export history CSV directly
POST	`/api/patient/export-history`	Trigger Celery CSV export
GET	`/api/patient/export-history/status/<task_id>`	Check CSV export status
GET	`/api/patient/export-history/download/<task_id>`	Download exported CSV

7. Architecture and Features

/backend

/exports – stores generated CSV files from patient history exports.
/routes – holds all Flask API route controllers for admin, doctor, patient, and auth flows.
/tasks – Celery background jobs for reminders, monthly reports, and async exports.

/templates – Jinja2 templates for emails and auto-generated reports.
app.py – main Flask application entry point and service initialization.
cache_utils.py – helper utilities for Redis-based caching.
celery_worker.py – Celery worker + beat scheduler setup.
config.py – central configuration for Flask, DB, JWT, Redis, and mail.
hms.db – SQLite database storing all HMS data.
mail_config.py – mail server configuration for sending email notifications.
models.py – SQLAlchemy models defining all database tables and relationships.
redis_client.py – Redis connection client for caching and task queueing.

/frontend

/src/api (axios.js) – Axios instance for authenticated API requests.
/src/router (index.js) – Vue Router setup for route navigation by user role.
/src/store (authStore.js) – state management for JWT, roles, and authentication.
/src/views – all Vue pages grouped by dashboards:
- **admin** – admin dashboard, doctor mgmt, stats
- **doctor** – doctor dashboard, appointments, treatments
- **patient** – bookings, history, profile
- **pages** – common screens like Login, Register, Landing

/src/App.vue – root Vue component wrapping all UI.
/src/main.js – Vue application bootstrap and global config.
/src/index.html – base HTML template used by Vite/Vue build.

Features Implemented

- JWT-based authentication for Admin, Doctor, and Patient.
- Admin dashboard with doctor management, user search, and appointment overview.
- Doctor dashboard to view appointments, update status, and add treatment details.
- Patient dashboard for booking, **rescheduling**, canceling appointments, and viewing full history.
- Slot-based doctor availability with conflict-free appointment booking.
- Complete treatment history with diagnosis, prescriptions, tests, and follow-ups
- **Celery background jobs** for daily reminders, monthly reports, and CSV/PDF generation.
- **Redis caching** for faster doctor lists, availability, and dashboard statistics.

Additional / Advanced Features

- **Charts and analytics** for Admin and Doctor dashboards.
- **Downloadable history PDF** for Patients.
- **Downloadable monthly/weekly reports** for Doctors.
- Extended treatment fields and enhanced UI/UX improvements.

8. Video

Video Demonstration Link : HOSPITAL MANAGEMENT SYSTEM V2.mp4