1. Student Details

Name: Roushan
Roll Number: 24f2003408
Email: 24f2003408@ds.study.iitm.ac.in
About Me:
I am a student in the IIT Madras BS in Data Science program with a strong interest in full-stack application development. I enjoy building real-world applications that combine backend engineering, frontend design, and scalable architectures. This Hospital Management System represents an end-to-end full-stack project demonstrating authentication, dashboards, scheduling systems, background jobs, and caching.

---

2. Project Details

Project Title: *Hospital Management System V2*

Problem Statement:
To build a comprehensive web-based Hospital Management System that enables secure and efficient management of doctors, patients, appointments, treatment histories, and background tasks such as reminders and reporting.

Approach:
The system is built using **Flask** for the backend and **Vue.js** for the frontend. The backend follows a modular architecture with structured Blueprints for routes, SQLAlchemy ORM models, Celery for background jobs, and Redis for caching. The frontend is organized using Vue Router, reusable components, and Axios-based API integration. The application supports role-based access control for Admin, Doctor, and Patient.

---

3. AI/LLM Declaration

AI/LLM Declaration
I used ChatGPT (GPT-5) **only for documentation support and conceptual clarification**, not for writing core application logic.
All backend routes, models, API integrations, and frontend components were implemented manually by me.

I am still in the process of learning and understanding the underlying concepts behind **Celery and Redis**.
To complete the milestone related to background jobs and caching, I used AI specifically for:

- Understanding how Celery workers and brokers work

- Clarifying how Redis is used as both a cache and task queue
- Getting guidance on configuring Celery, Redis, and periodic tasks

However, **all final code—including Celery task definitions, Redis caching logic, background job configuration, and integration with Flask—was written, structured, and tested manually** by me.

The use of AI was strictly limited to:

- Conceptual explanation
- Syntax clarification
- Documentation refinement

No direct code copying from AI was used for the final implementation.

---

4. Technologies and Frameworks Used

Technology / Library	Purpose
Flask	Backend REST API framework
SQLAlchemy	ORM for database models and queries
SQLite	Lightweight relational database
Vue.js 3	Frontend application framework
Axios	API communication
Bootstrap 5	UI design and layout
Redis	Caching + Celery message broker
Celery	Background task processing (async jobs)
Vite	Frontend build tool
Flask-Login	Authentication and session management

Werkzeug	Password hashing

---

5. Database Schema / ER Diagram

Main Tables

- **User** – authentication + shared fields
- **Doctor** – specialization, department, experience
- **Patient** – personal and medical details
- **Department** – hospital departments
- **DoctorAvailability** – available time slots
- **Appointment** – booking and tracking
- **Treatment** – diagnosis & prescription records

Relationships

- One-to-Many → Department → Doctor
- One-to-Many → Doctor → Appointment
- One-to-Many → Patient → Appointment
- One-to-One → Appointment → Treatment
- One-to-Many → Doctor → Availability

.

---

6. API Resource Endpoints

Authentication

Endpoint	Method	Description

`/api/auth/login`	POST	Login for all roles
`/api/auth/register`	POST	Register patient
`/api/auth/logout`	POST	Logout
`/api/auth/me`	GET	Current user

Admin APIs

- Manage doctors
- Manage patients
- Manage appointments
- View department list
- Dashboard statistics

Doctor APIs

- View appointments
- Complete appointment
- Update treatment record
- Manage availability

Patient APIs

- Search doctors
- Book appointment
- Cancel/reschedule
- View treatment history
- Export history (CSV — async job)

Appointment APIs

- Book appointment
- Check availability
- View/Update/Cancel appointment

---

7. Architecture and Features

Architecture Overview

- `start_backend.py` – Flask entry point
- `/app/routes` – separate route modules for admin, doctor, patient, appointments
- `/app/models.py` – database models
- `/app/tasks.py` – Celery jobs
- `/frontend/src/views/` – Vue pages
- `/frontend/src/components/` – shared components like Navbar
- Redis used for caching and background job queue

Key Features Implemented

- Role-based access (Admin, Doctor, Patient)
- Appointment scheduling with conflict prevention
- Dashboard summaries
- Treatment record management
- Search and filtering
- Redis caching for performance
- Celery-based jobs:

    - Daily reminders
    - Monthly doctor activity report
    - CSV export for medical history

---

8. Video Presentation

Drive link to project walkthrough and demo:
https://drive.google.com/file/d/10aT4EXqFlHMB5DRset13KsMCuFMNGY7Y/view?usp=drive_link

---