Project Report
On
Healthcare Management System

Course Name: Devops Fundamentals

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1	Vinay Kuthey	EN22CS3011084
2	Viraj Agrawal	EN22CS3011089
3	Lakshita Singh	EN22CS3011135
4	Vishakha Shadija	EN22ME304114
5	Vishakha Hilsayan	EN22EL301065

Group Name: 09D10

Project Number: D0-22

Industry Mentor Name:

University Mentor Name: Prof. Avnesh Joshi

Academic Year: 2026

Table of Contents

1. Problem Statement & Objectives

1.1 Introduction

The Healthcare Management System (HMS) is a full-stack web application designed to digitize and streamline core healthcare operations. It provides healthcare professionals with a centralized platform to efficiently manage patient records, schedule appointments, and maintain medical histories — all through a modern, intuitive dashboard.

The system is built using Python Flask for the backend REST API, React.js for the frontend, and PostgreSQL as the database. The entire application is containerized using Docker and deployed live on AWS EC2, making it accessible from anywhere.

Key highlights include a premium dark-themed dashboard with user authentication (Login/Signup), full CRUD operations for patients, appointments, and medical records, real-time system health monitoring, and a published Docker image on Docker Hub — making it production-ready and easily deployable.

1.2 Problem Statement

Healthcare organizations often struggle with managing patient data, appointments, and medical records efficiently. Traditional paper-based or siloed systems lead to data inconsistencies, delayed access to critical information, and poor coordination between healthcare providers. There is a need for a centralized, digital Healthcare Management System that streamlines these operations.

1.3 Project Objectives

- Build a REST API backend to manage patients, appointments, and medical records
- Develop a modern, responsive frontend dashboard with authentication
- Containerize the application using Docker for consistent deployment
- Deploy the application on AWS EC2 cloud infrastructure
- Ensure data persistence using PostgreSQL database

1.4 Scope of the Project

- User authentication (Login & Signup)
- Patient registration and management (CRUD operations)
- Appointment scheduling and management (CRUD operations)
- Medical records management (CRUD operations)
- Medical records management (CRUD operations)
- System health monitoring via API health endpoint
- Cloud deployment on AWS EC2
- Docker containerization and Docker Hub image publishing

2. Proposed Solution

2.1 Key Features

- User Authentication — Login & Signup with session management
- Patient Management — Add, View, and Delete patient records
- Appointment Scheduling — Book and manage doctor appointments
- Medical Records — Create and view patient medical history
- Premium Dashboard — Overview stats, collapsible sidebar, dark theme
- Dockerized Deployment — Multi-container setup with Docker Compose
- AWS EC2 Cloud Deployment — Live and publicly accessible
- RESTful API — Clean endpoints with health check monitoring

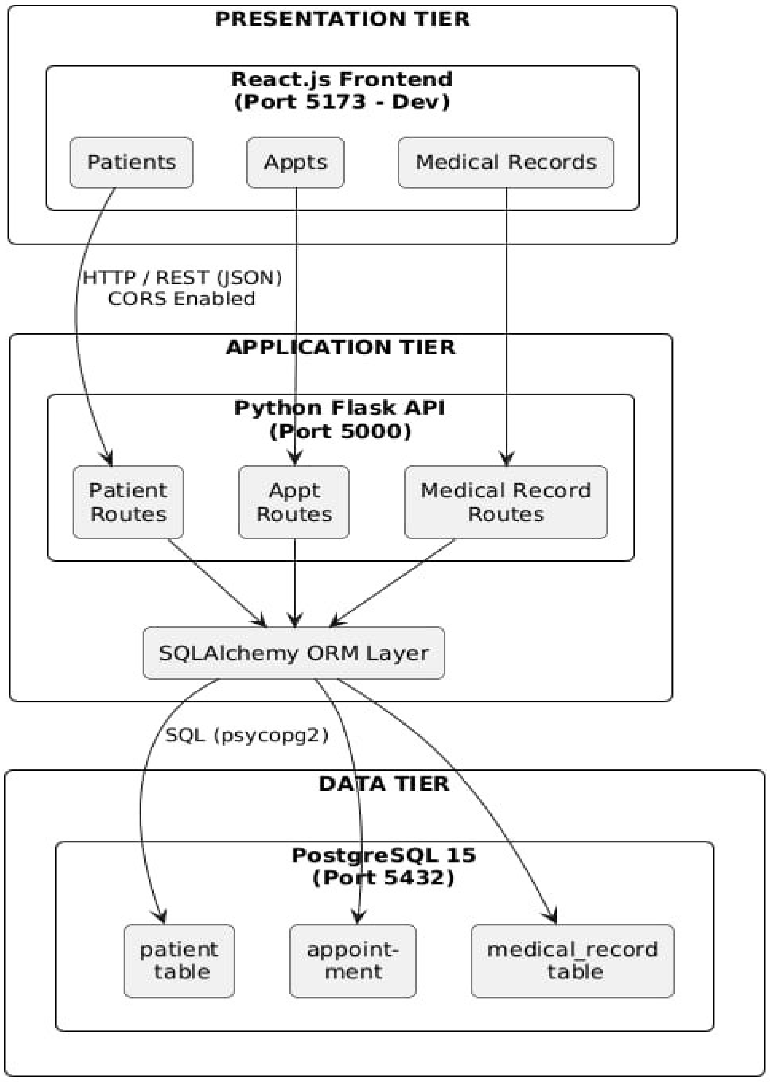
2.2 Overall Architecture / Workflow

Workflow:

- User interacts with React.js Frontend (Patients / Appointments / Medical Records).
- Frontend sends HTTP REST API request (JSON) to backend.
- CORS allows communication between React (Port 5173) and Flask (Port 5000).
- Request reaches Flask API Application Tier.
- Flask routes handle requests:
  - Patient Routes
  - Appointment Routes
  - Medical Record Routes
- Backend performs validation & business logic.
- Flask calls SQLAlchemy ORM layer.
- ORM converts Python code → SQL queries.
- Uses psycopg2 driver to connect to the database.
- Query executed in PostgreSQL 15 (Port 5432).

- Data stored/ retrieved from tables:
    - patient
    - appointment
    - medical_record
- Database sends response → ORM → Flask.
- Data stored/ retrieved from tables:
- Flask returns a JSON response to React.
- React updates the UI and displays results to users.

Architecture:

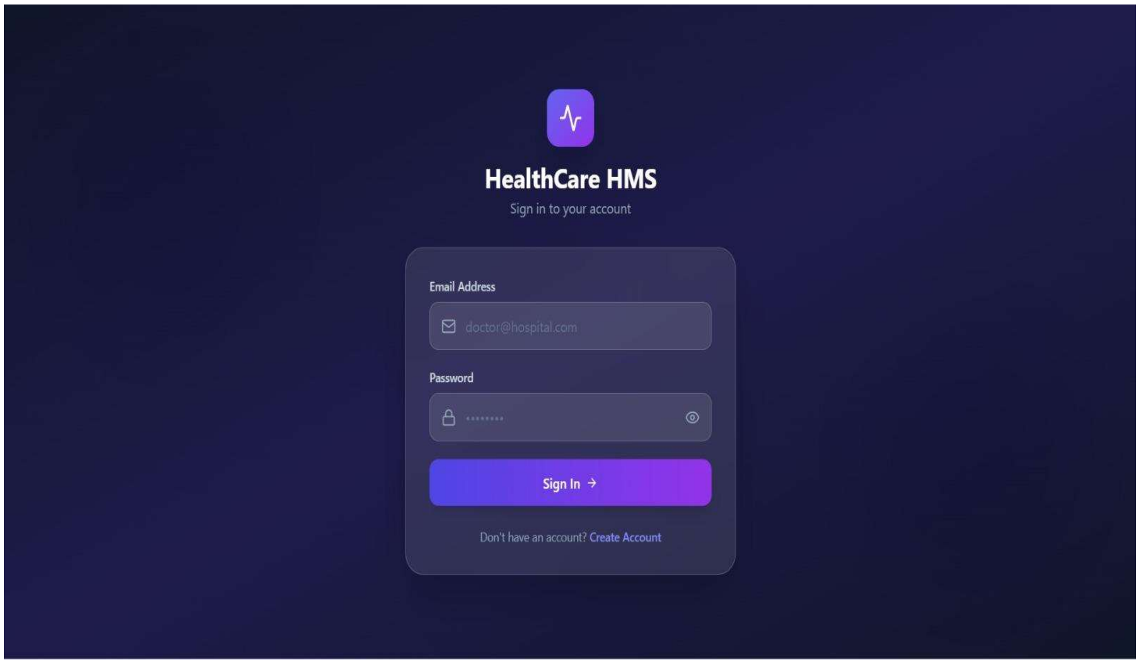


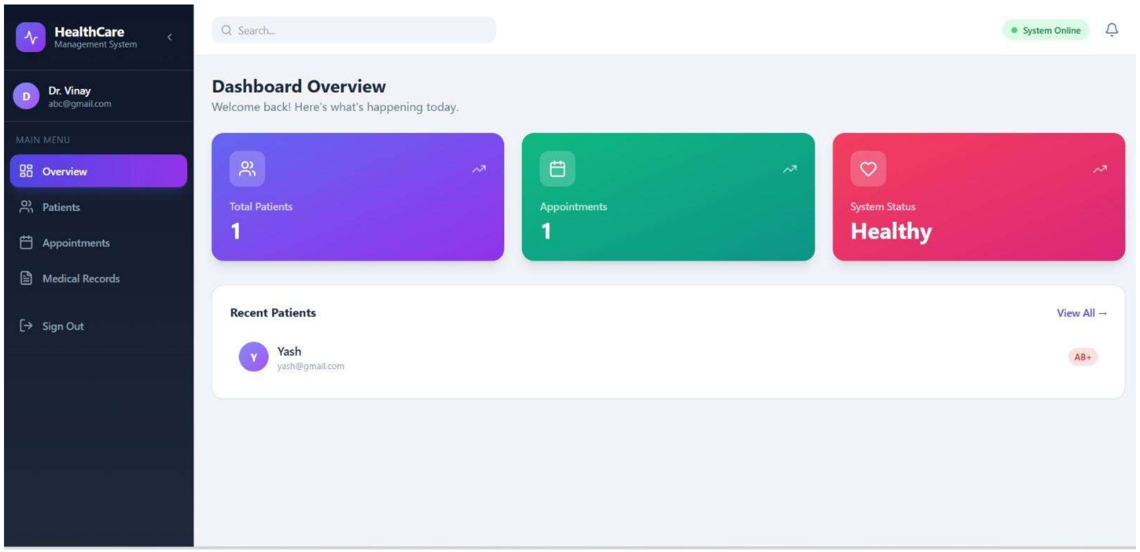
2.3 Tools & Technologies

- **Frontend:** React.js, Vite, TailwindCSS, Lucide Icons, React Router DOM
- **Backend:** Python 3, Flask, SQLAlchemy, Gunicorn
- **Database:** PostgreSQL 15
- **Containerization:** Docker, Docker Compose
- **Container Registry:** Docker Hub (2200314/healthcare-app)
- **Cloud Platform:** AWS EC2 (Ubuntu 22.04, t2.micro — Free Tier)
- **Version Control:** Git, GitHub
- **CI/CD:** GitHub Actions

3. Results & Output

3.1 Screenshots / Outputs





HealthCare
Management System

Dr. Vinay
abc@gmail.com

MAIN MENU

- Overview
- Patients
- Appointments
- Medical Records
- Sign Out

Search...

System Online

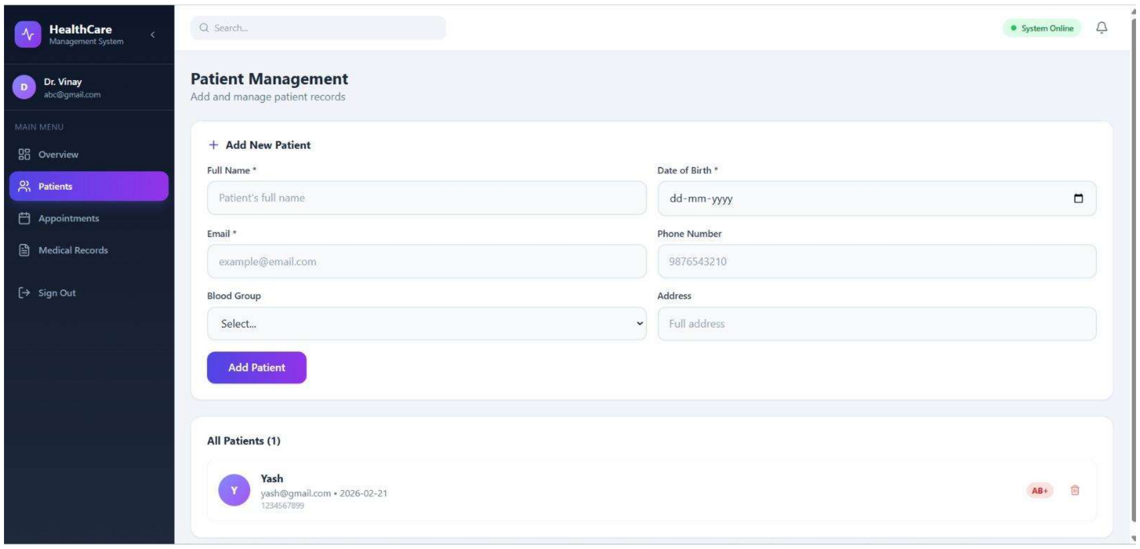
Dashboard Overview

Welcome back! Here's what's happening today.

Total Patients	Appointments	System Status
1	1	Healthy

Recent Patients View All →

Yash
yash@gmail.com
AB+



HealthCare
Management System

Dr. Vinay
abc@gmail.com

MAIN MENU

- Overview
- Patients
- Appointments
- Medical Records
- Sign Out

Search...

System Online

Patient Management

Add and manage patient records

+ Add New Patient

Full Name *	Date of Birth *
Patient's full name	dd-mm-yyyy
Email *	Phone Number
example@email.com	9876543210
Blood Group	Address
Select...	Full address

Add Patient

All Patients (1)

Yash
yash@gmail.com • 2026-02-21
1234567899
AB+

3.2 Reports / DashBoards / Models

Dashboard Overview Screen

- Total Patients count
- Total Appointments count
- System Status (Online/Offline)
- Recent Patients list

Database Models / Schema:

- Patient: id, name, email, phone, date_of_birth, blood_group, address
- Appointment: id, patient_id, doctor_name, datetime, reason, status
- Medical Record: id, patient_id, doctor_name, diagnosis, prescription, date

REST API Endpoints:

- GET /health → System health check
- GET /api/patients → All patients
- POST /api/patients → Add patient
- DELETE /api/patients/:id → Delete patient
- GET /api/appointments → All appointments
- POST /api/appointments → Book appointment
- GET /api/records → Medical records
- POST /api/records → Add record

3.3 Key Outcomes

- Fully functional Healthcare Management REST API deployed on AWS EC2
- PostgreSQL database running in Docker with persistent data volumes
- Modern React frontend with Login, Signup, and Premium Dashboard
- Docker image published to Docker Hub — accessible worldwide
- Complete source code on GitHub with version control

4. Conclusion

This project successfully delivers a full-stack Healthcare Management System that addresses the need for centralized patient, appointment, and medical records management. Built using modern industry-standard technologies — Flask, React, PostgreSQL, and Docker — and deployed on AWS EC2, the system demonstrates a complete software development lifecycle from design to cloud deployment.

Key Learnings:

- Building and consuming RESTful APIs with Flask and Axios
- Implementing authentication with React Context API
- Containerizing multi-service applications with Docker Compose
- Publishing Docker images to Docker Hub
- Deploying containerized applications on AWS EC2
- Configuring Security Groups, SSH, and cloud networking on AWS

5. Future Scope & Enhancements

- JWT Authentication: Replace localStorage with secure JWT token-based auth
- Mobile App: React Native version for mobile devices
- Email Notifications: Appointment reminders via email/SMS
- AI Integration: Symptom checker and diagnosis suggestion using ML
- Analytics Dashboard: Patient trends and appointment statistics with charts