Project - High Level Design

On

Dockerized Healthcare Python Flask Service

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: : 09

Industry Mentor Name:

University Mentor Name: Prof. Avnesh Joshi

Academic Year: 2026

Table of Contents

1. Introduction.

This document provides a comprehensive overview of the architectural design of the Healthcare Management System (HMS) — a full-stack web application designed to digitize and streamline core healthcare operations.

The system ensures consistent, repeatable, and scalable environment setup for healthcare organizations by eliminating manual configuration errors. It covers the automated provisioning of local Docker environments.

The document outlines the project objectives, scope, tools used, architecture, workflow, and implementation strategy.

1.1 Scope of the document.

This document covers:

- The overall project objectives and problem statement
- RESTful API backend (Flask)
- Automated deployment using Docker or cloud platforms
- Configuration management and dependency installation
- Workflow and architecture explanation
- Tools and technologies used
- Key features and expected outcomes
- Security and scalability considerations

1.2 Intended Audience

- Software Architects Review system design decisions.
- Frontend Developers - Understand UI components and API contracts
- Backend Developers - Understand API and database design
- DevOps Engineers – For infrastructure automation and deployment practices
- QA Engineers – Understand system behavior for test planning
- Project Managers – To track project scope, deliverables, and objectives
- Stakeholders – Review overall system capabilities

1.3 System overview

- The Healthcare Management System is a 3-tier web application that enables healthcare providers to manage patients, appointments, and medical records through a modern, responsive web interface.

  Core Capabilities:

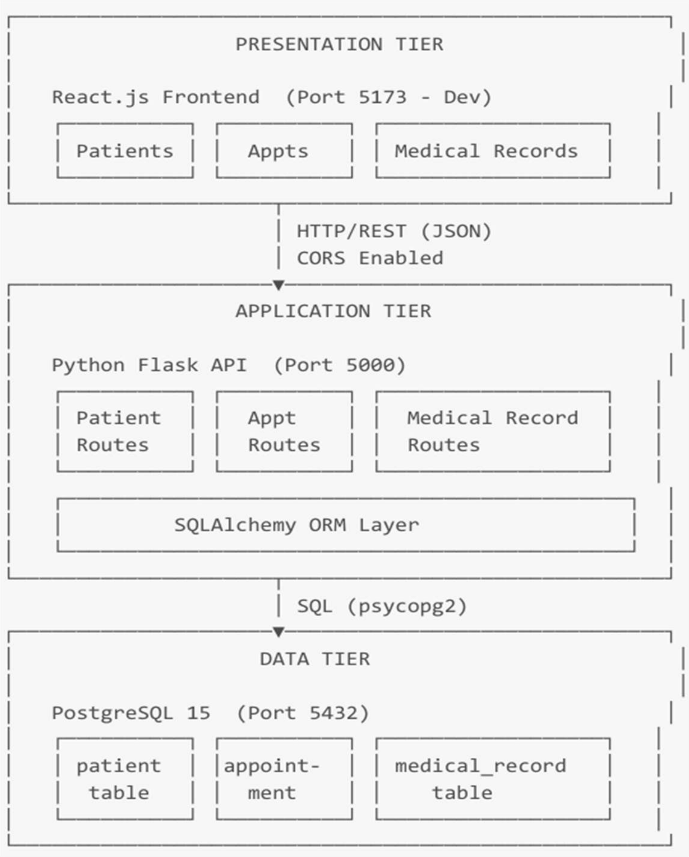
  - Patient lifecycle management (Create, Read, Update, Delete)
  - Appointment scheduling with doctor assignment
  - Medical record creation and retrieval
  - Real-time system health monitoring
  - Secure containerized deployment
  - Automated CI/CD pipeline

Technology Summary:

Layer	Technology
Frontend	React + Tailwind CSS
Backend	Python Flask
Database	PostgreSQL
Container	Docker + Docker Compose
CI/CD	Github Actions

2. System Design

2.1 Application Design



```
                PRESENTATION TIER

   React.js Frontend  (Port 5173 - Dev)

   Patients      Appts       Medical Records

                      HTTP/REST (JSON)
                      CORS Enabled

                APPLICATION TIER

   Python Flask API  (Port 5000)

   Patient       Appt        Medical Record
   Routes        Routes      Routes

           SQLAlchemy ORM Layer

                      SQL (psycopg2)

                  DATA TIER

   PostgreSQL 15  (Port 5432)

   patient       appoint-    medical_record
    table        ment           table
```

The application follows a **3-tier Client Server Architecture**:

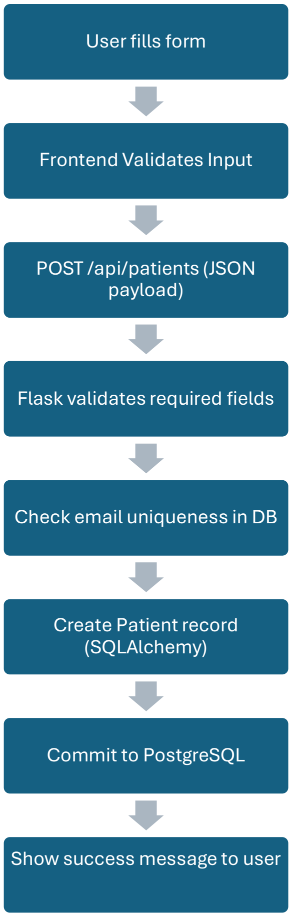
- **Presentation Tier-** Frontend (Browser) that sends user requests and displays responses.
- **Application Tier-** Flask backend that processes logic and handles API requests.
- **Data tier-** PostgreSQL database that stores and manages application data.

Design Principles:

- **Separation of Concerns** — Each tier has a distinct responsibility
- **Loose Coupling** — Frontend and backend communicate only via REST API
- **Single Responsibility** — Each module handles one domain
- **DRY (Don't Repeat Yourself)** — Shared utilities and base classes

2.2 Process Flow

Step-by-step Process Flow:

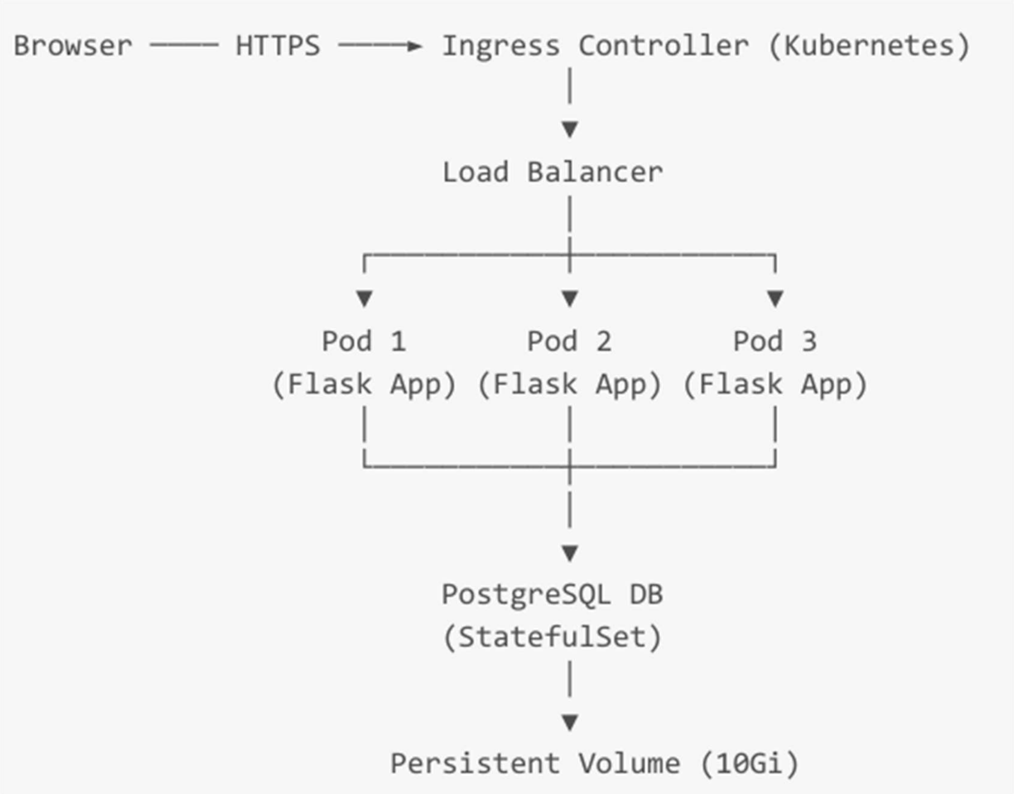

User fills form
Frontend Validates Input
POST /api/patients (JSON payload)
Flask validates required fields
Check email uniqueness in DB
Create Patient record (SQLAlchemy)
Commit to PostgreSQL
Show success message to user

Validation Fail->Show Error

Database Updated

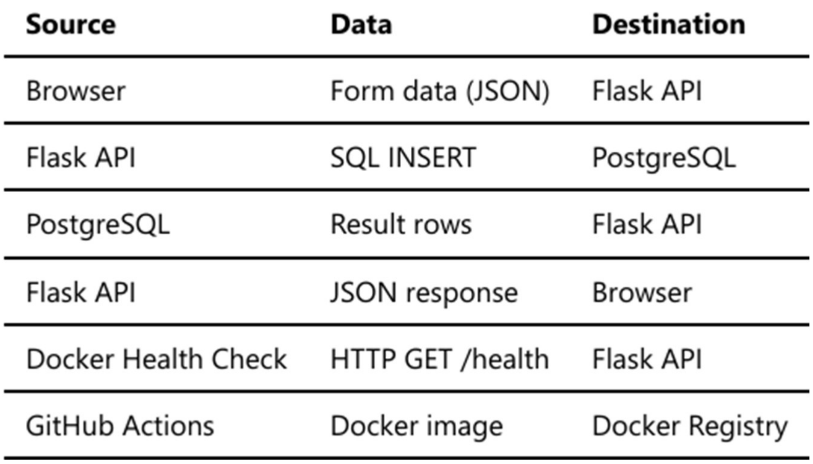
Fig. Process Flow Diagram

2.3 Information Flow



```
Browser ──── HTTPS ────► Ingress Controller (Kubernetes)
                                   │
                                   ▼
                             Load Balancer
                                   │
                        ┌──────────┼──────────┐
                        ▼          ▼          ▼
                      Pod 1      Pod 2      Pod 3
                  (Flask App) (Flask App) (Flask App)
                        │          │          │
                        └──────────┼──────────┘
                                   │
                                   ▼
                             PostgreSQL DB
                             (StatefulSet)
                                   │
                                   ▼
                        Persistent Volume (10Gi)
```

Fig. Data Flow Diagram



Source	Data	Destination
Browser	Form data (JSON)	Flask API
Flask API	SQL INSERT	PostgreSQL
PostgreSQL	Result rows	Flask API
Flask API	JSON response	Browser
Docker Health Check	HTTP GET /health	Flask API
GitHub Actions	Docker image	Docker Registry

2.4 Components Design

Core Components:

1. Version Control System (Git)
2. App Factory
3. Configuration
4. App Root (State Management, API calls)
5. Database (Postgre)
6. Docker Engine
7. Monitoring & Logging Tools

Each component operates independently but integrates through automated scripts.

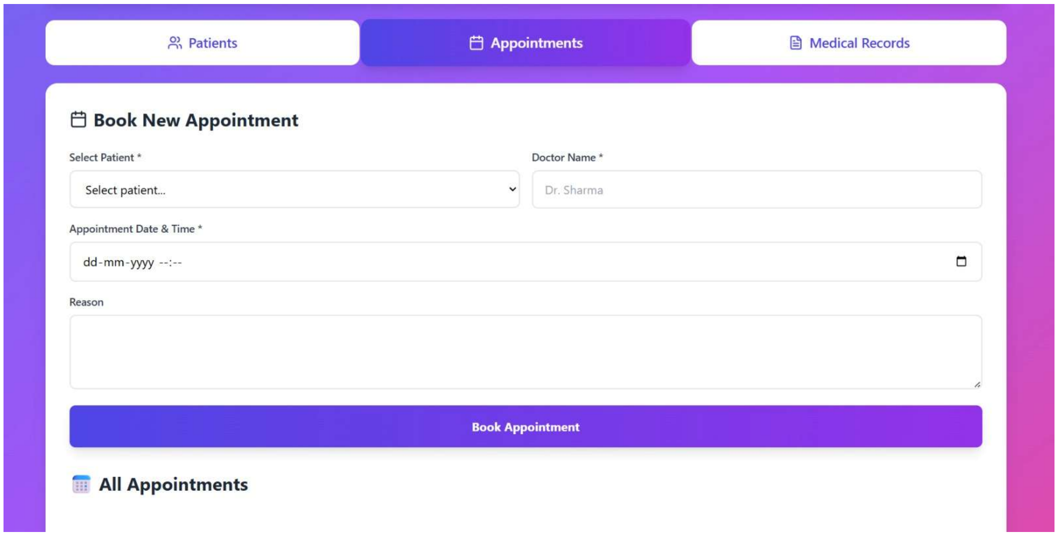
2.5 Key Design Considerations

1. Scalability
2. Security (IAM roles, SSH key management)
3. High Availability
4. Infrastructure Modularity
5. Environment Isolation
6. Cost Optimization
7. Disaster Recovery Readiness

2.6 API Catalogue

S No.	API Name	Purpose	Method	Description
1	AWS EC2 API	Instance Creation	POST	Creates virtual machines
2	AWS S3 API	Storage Management	PUT/GET	Stores files & backups
3	Docker API	Container Control	REST	Manages containers

3. Interfaces



Patients | Appointments | Medical Records

Book New Appointment

Select Patient *
Select patient...

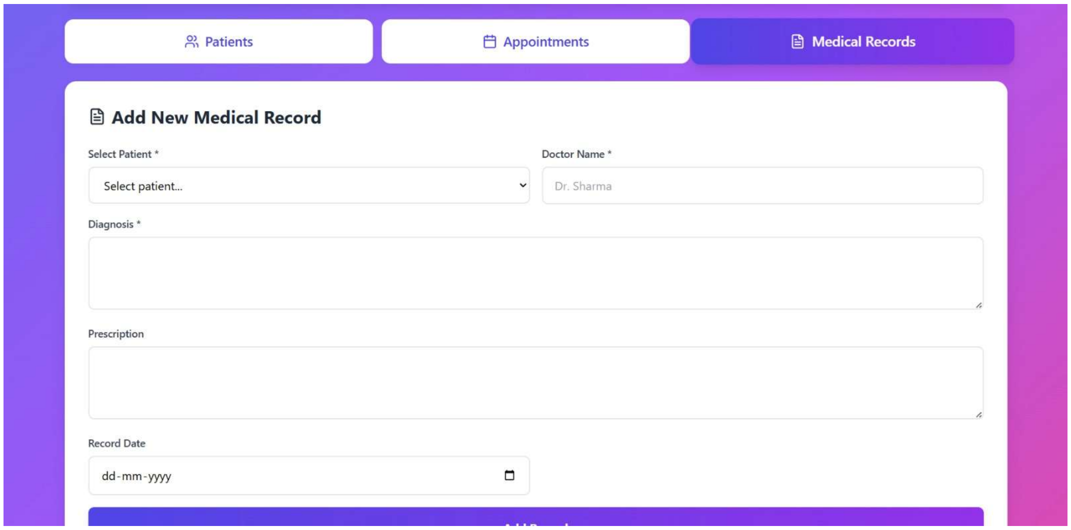
Doctor Name *
Dr. Sharma

Appointment Date & Time *
dd-mm-yyyy --:--

Reason

Book Appointment

All Appointments



Patients | Appointments | Medical Records

Add New Medical Record

Select Patient *
Select patient...

Doctor Name *
Dr. Sharma

Diagnosis *

Prescription

Record Date
dd-mm-yyyy

4. Non-Functional Requirements

4.1 Security Aspects

1. IAM Role-Based Access Control
2. Encrypted Storage (S3 Encryption)
3. HTTPS Communication
4. SSH Key Authentication
5. Secrets Management
6. Firewall & Security Groups

4.2 Performance Aspects

1. Auto Scaling Groups
2. Load Balancers
3. Optimized Docker images
4. Efficient Terraform Modules
5. Parallel resource provisioning
6. Monitoring with alerts

5. References

1. Docker Documentation
2. AWS Documentation
3. DevOps Best Practices
4. Infrastructure as Code Principles