Project - High Level Design

On

Healthcare Content Generator

Course Name: Generative AI

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
1	Janhavi Gupta	EN22CS301454
2	Harshita Songara	EN22CS301411
3	Hariom Patidar	EN22CS301383
4	Hemant Mandloi	EN22CS301422
5	Uday Dubey	EN22CS301058

Group Name:

Project Number: D4 06

Industry Mentor Name:

University Mentor Name: Prof. Divya Kumawat

Academic Year:2025 - 2026

Table of Contents

1. Introduction

1.1 Scope of the Document

This High-Level Design (HLD) document provides a comprehensive architectural overview of the **Healthcare Content Generator** system. The document describes the system's architecture, components, modules, interfaces, and data design at a high level, enabling stakeholders to understand how the system will be structured and how its components will interact.

The document covers:

- System architecture and modules
- LLM integration (Gemini via LangChain)
- Data storage and history management
- Application workflows
- Security and performance considerations
- Clinical workflow alignment

This HLD focuses strictly on components relevant to medical report generation for healthcare professionals.

1.2 Intended Audience

This document is intended for:

- General Physicians – The application enables general physicians to quickly document, access, and manage patient records in a structured digital format.
- Specialists – The application supports specialists by organizing detailed clinical data and specialty-specific documentation for accurate diagnosis and treatment planning.
- Clinical Consultants – The application allows consultants to review comprehensive patient histories and provide informed recommendations efficiently.
- Healthcare IT Teams – The application helps IT teams manage system integration, data security, and maintenance of digital medical records infrastructure.

- Hospital Administrators – The application provides administrators with oversight of documentation workflows, compliance tracking, and operational efficiency metrics.
- Medical Documentation Professionals – The application streamlines clinical transcription, coding, and standardized medical record creation.
- Hospital Digital Transformation Teams – The application serves as a core platform for implementing and scaling digital documentation and workflow automation initiatives.

The application is specifically designed for doctors and specialists who require structured, standardized, and clinically formatted documentation.

1.3 System Overview

Healthcare Content Generator is an AI-powered web application designed to assist medical professionals in generating structured, professional medical reports efficiently. The system leverages Google's Gemini AI to create comprehensive reports based on patient information, reducing documentation time and ensuring consistency in medical reporting. **Healthcare Content Generator** is a web-based AI-powered medical documentation assistant built using:

- **Frontend/UI:** Streamlit
- **LLM Integration:** LangChain + Gemini (Google Generative AI)
- **Data Storage:** Local JSON-based history system
- **Programming Language:** Python
- May omit important sections
- Reduces patient interaction time

Primary Capabilities

- Generate structured clinical reports
- Maintain consistent medical documentation format
- Store historical reports
- Allow report download
- Provide quick retrieval of previous cases

Vision Statement:

"To revolutionize medical documentation by providing healthcare professionals with an intelligent, intuitive tool that generates accurate, structured medical reports in seconds, allowing them to focus more on patient care and less on paperwork."

1.4 Benefits to Healthcare Professionals

MediGen provides several advantages:

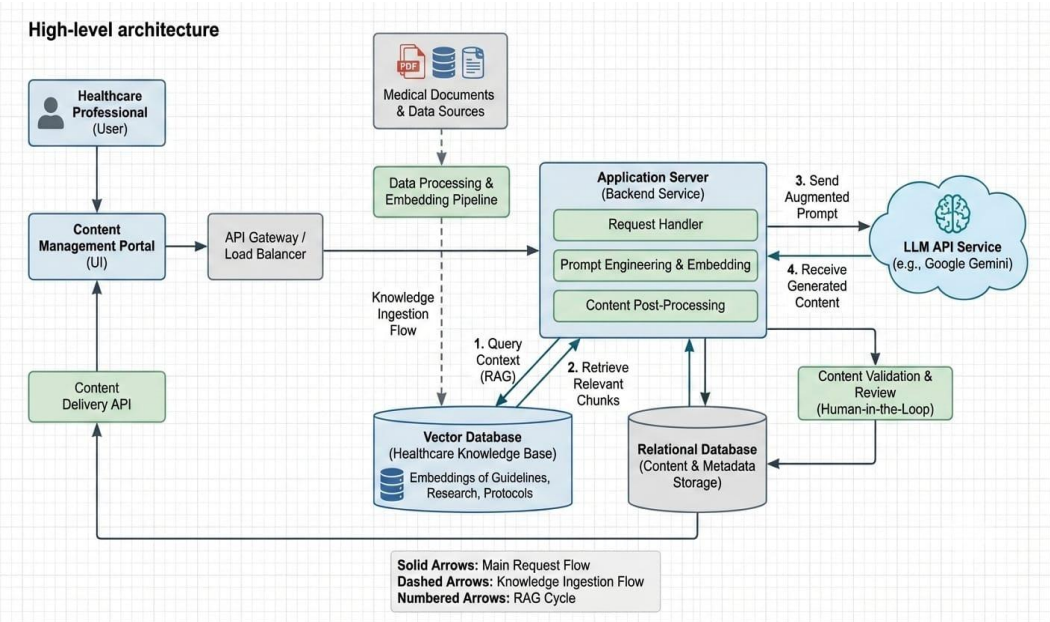
- Time Efficiency – Reduces documentation workload.
- Consistency – Ensures structured, standardized reports.
- Clinical Completeness – Covers essential medical sections.
- Accessibility – Provides quick access to previous reports.
- Professional Formatting – Maintains clinical tone and terminology.

By assisting with documentation, MediGen allows doctors to focus more on patient care rather than administrative tasks.

2. System Design

2.1 Application Design

The application is structured into distinct, loosely coupled modules that communicate through well-defined interfaces.



Logical Modules

1. UI Module (app.py)
2. Report Generation Engine (model.py)
3. History Management (history_manage.py)
4. Storage Layer (report_history.json)

2.2 Process Flow

Medical Report Generation Workflow

Doctor Inputs Patient Data

↓

Streamlit UI Validates Fields

↓

generate_medical_report() Called

↓

LangChain Prompt Template Applied

↓

Gemini LLM Generates Structured Report

↓

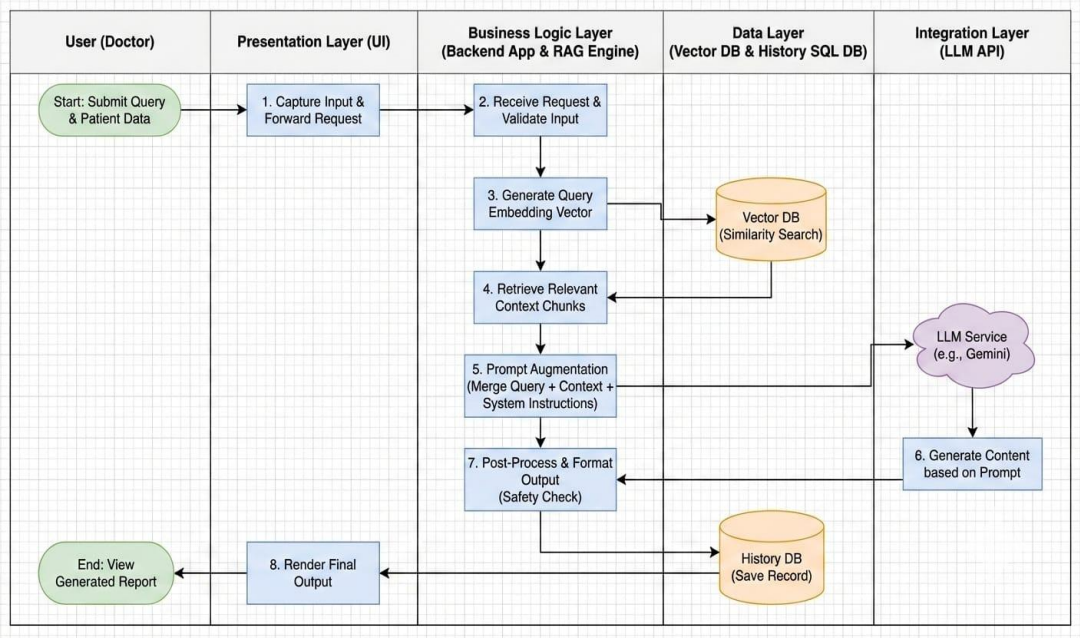
Report Returned to UI

↓

Report Saved in JSON History

↓

Report Displayed + Download Option



2.3 Information Flow

Input Information

- Age
- Gender
- Report Type
- Primary Symptoms

Processing

- Injected into structured medical prompt
- Sent to Gemini LLM
- Formatted using Markdown structure
- Parsed via StrOutputParser

Output Information

- Structured Medical Report

  Saved metadata:

  - Timestamp
  - Patient demographics

- Report type
- Symptoms

2.4 Component Design

2.4.1 UI Layer (app.py)

Responsibilities:

- Navigation (Home, Report Generation, History)
- Input capture
- Validation
- Display reports
- Download functionality

Key Functions:

2.4.1 UI Layer (app.py)

Responsibilities:

- Navigation (Home, Report Generation, History)
- Input capture
- Validation
- Display reports
- Download functionality
- Session state management

Key Functions:

- st.button()
- st.text_input()
- st.text_area()
- st.download_button()

2.4.2 Report Generation Engine (model.py)

Responsibilities:

- Configure Gemini API
- Define structured medical prompt
- Invoke LangChain chain
- Handle API errors

Components:

- ChatPromptTemplate
- ChatGoogleGenerativeAI
- StrOutputParser
- Chain pipeline:

➢ prompt | llm | output_parser

Design Principles:

- Strict report formatting
- No AI disclaimers
- Clinical tone
- Deterministic structured sections

2.4.3 History Management Module (history_manage.py)

Responsibilities:

- Save report
- Load report history
- Sort reports (latest first)
- Clear history

Key Functions:

- save_report()
- load_history()
- get_all_reports()
- clear_all_reports()

2.4.4 Storage Layer

File: report_history.json

Stores:

- Age
- Gender
- Symptoms
- Report Type
- Generated Report
- Timestamp

2.5 Key Design Considerations

1. Clinical Structure Enforcement
   Strict Markdown headings and medical sections.
2. Data Simplicity
   JSON storage chosen for lightweight deployment.
3. Modularity
   Separation of UI, Model, and Data logic.
4. Error Handling
   API error detection:
   - 429 (Rate limit)
   - 500/503 (Service unavailable)
   - Invalid API key
5. Usability for Doctors
   - Minimal input required
   - Clear sectioned output
   - Fast generation

2.6 API Catalogue

1. generate_medical_report()

Input:

- age (str)
- gender (str)
- primary_symptoms (str)
- report_type (str)

Output:

- Structured medical report (Markdown string)

2. Gemini LLM API

Model:

- gemini-2.5-flash

Configuration:

- Temperature: 0.7
- Retry: 2
- Timeout: None

3. Data Design

3.1 Data Model

Report Object Structure

{
"age": "45",
"gender": "Male",
"symptoms": "Chest pain for 3 days",
"type": "Initial Consultation",
"report": "Generated Markdown Report",
"timestamp": "2026-02-19T10:45:30"
}

3.2 Data Access Mechanism

- JSON file read/write
- ISO format timestamp conversion
- Sorting by timestamp (descending)
- In-memory loading via session state

3.3 Data Retention Policies

Current Implementation:

- Reports stored locally indefinitely
- Manual deletion available via "Clear All"

Recommended Future Enhancements:

- Configurable retention period
- Encrypted storage
- Backup mechanism

3.4 Data Migration

Current System:

- No database migration required (JSON-based)

Future Upgrade Path:

- Migrate to:
    - PostgreSQL
    - MongoDB
    - Hospital EMR integration

Migration Steps:

1. Convert JSON schema to relational table.
2. Add patient anonymization layer.
3. Implement secure database connectors.

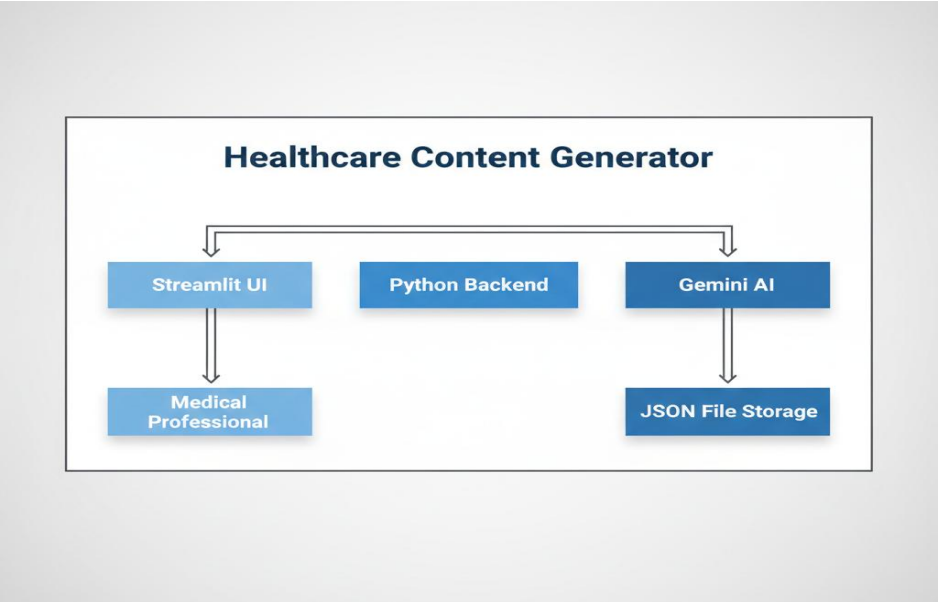
4. Interfaces

User Interface

- Web-based (Streamlit)
- Sidebar navigation
- Responsive layout

External Interface

- Gemini API (Google Generative AI)
- LangChain abstraction layer



5. State and Session Management

Managed using:

st.session_state

Stored States:

- Current page
- Selected report
- Loaded report history

Session is per user browser session.

6. Caching

Current Implementation:

- No explicit caching layer

Potential Improvements:

- Cache LLM responses for repeated identical inputs
- Cache history loading using:
  - @st.cache_data

7. Non-Functional Requirements

Non-Functional Requirements (NFRs) define how the system performs, rather than what it does.
For a medical documentation system used by doctors and specialists, NFRs are critical because they directly impact:

- Patient safety
- Clinical reliability
- Data privacy
- System usability
- Operational efficiency

In MediGen, NFRs ensure the application is not only functional but also secure, dependable, and suitable for a healthcare environment.

7.1 Security Aspects

Current Risks

- API key hardcoded in model.py ❗
- Local JSON storage not encrypted
- No authentication mechanism

Recommended Security Enhancements

1. Move API key to environment variables
2. Encrypt report_history.json

3. Add user authentication
4. Implement HTTPS deployment
5. Role-based access (Doctor/Admin)

7.2 Performance Aspects

Current Performance

- Lightweight local deployment
- Fast LLM (Flash model)
- Minimal storage overhead

Scalability Considerations

If deployed in hospital environment:

- Use asynchronous LLM calls
- Implement request queue
- Move to cloud database
- Load balancing for concurrent doctors

Expected Report Generation Time:

- 7-8 seconds (average)

8. References

- **Streamlit Documentation** - Official documentation for building interactive web applications with Python.
- **Google Gemini API Documentation** - Official guide for integrating Google's Gemini AI models.
- **LangChain Documentation** - Framework documentation for building LLM-powered applications.
- **Python Documentation** - Comprehensive reference for Python programming language version 3.9+.