Project - High Level Design

on

Healthcare Planning Assistant Agent

Course Name: Agentic AI

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
01	MEET SOLANKI	EN22CS301595
02	MAYURI PATIDAR	EN22CS301593
03	MOKSHIKA YADAV	EN22CS301614
04	PALAK SHAHDADPURI	EN22CS301677
05	NIRDESH SHARMA	EN22CS301655
06	PANKAJ	EN22CS301679

Group Name: 11D6

Project Number: AAI-22

Industry Mentor Name:

University Mentor Name: Prof. Nishant Srivastava

Academic Year: 2025-26

MEDICAPS UNIVERSITY

Datagami
"Lead Digital Technology"

Table of Contents

1. Introduction

1.1 Scope of the Document

This document provides the High-Level Design (HLD) for the Healthcare Planning Assistant, an AI-powered healthcare advisory application.
The scope of this document includes:

- Overall system architecture and design approach
- Major modules and components of the system
- Workflow and data flow across system layers
- Integration with external AI services (Google Gemini API and Groq LLM API)

1.2 Intended Audience

This document is intended for:

- **Project evaluators and faculty members** reviewing the system design
- **Developers** who will implement, test, or enhance the application
- **System designers** responsible for architecture and modular design

1.3 System Overview

The Healthcare Planning Assistant is a web-based AI application designed to provide structured health guidance based on user-entered symptoms. The system uses Large Language Models (LLMs) through Google Gemini API and Groq LLM API to generate responses in a step-by-step manner.

The system provides:

- Possible health conditions (advisory, not diagnostic)
- Precautionary measures
- General medication guidance (non-prescriptive)
- Nutrition and diet recommendations

The application is developed using Python, Streamlit for the user interface, and LangChain for prompt orchestration and sequential chaining. The system is currently designed as a prototype for local execution and does not store user medical data permanently.

2. System Design

2.1 Application Design

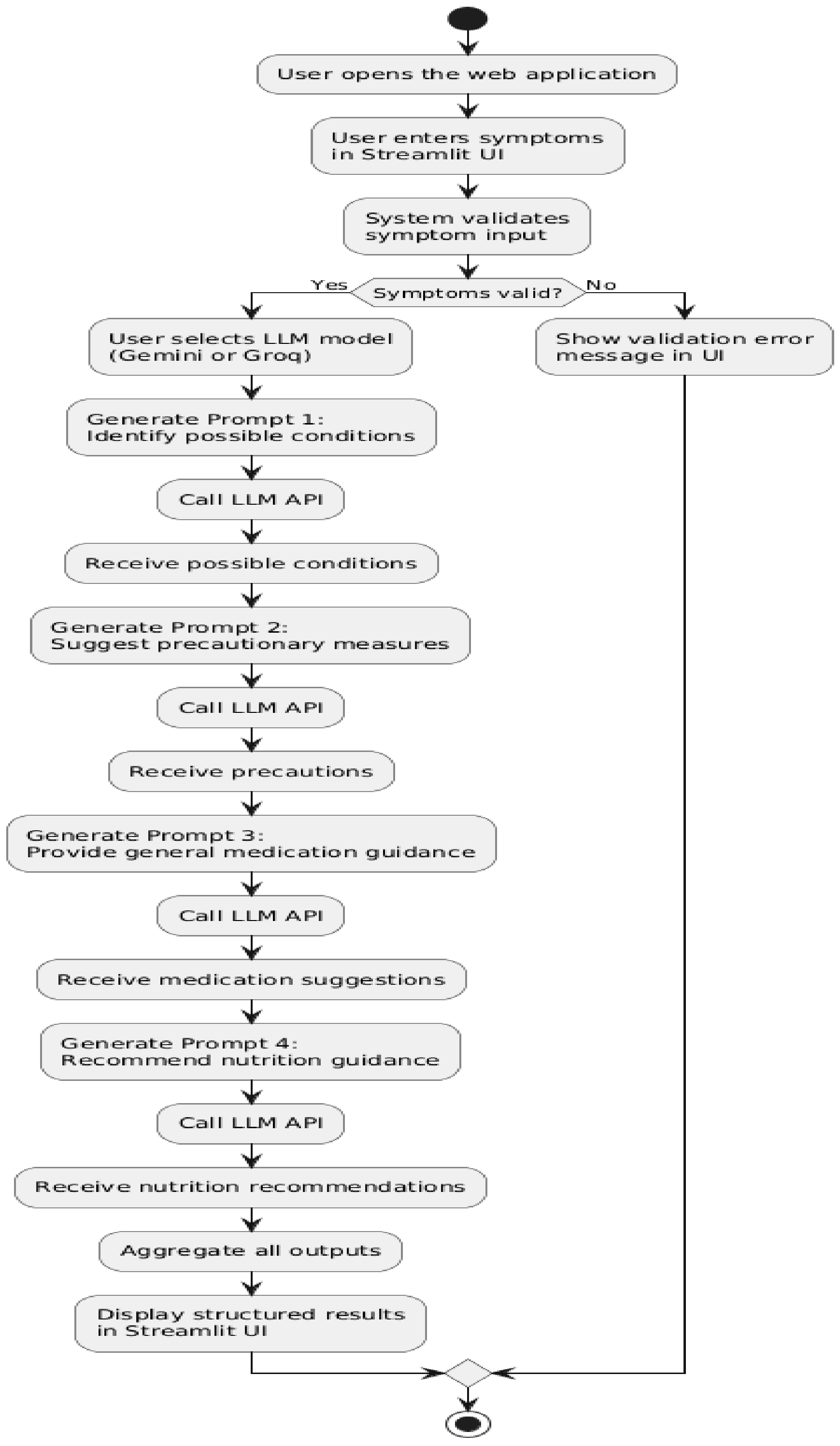
The Healthcare Planning Assistant is designed as a modular AI-driven healthcare advisory application. The application follows a layered design to separate user interaction, business logic, LLM orchestration, and external service integration.

Layered Structure

- **Presentation Layer (Streamlit UI):** Handles user interaction, symptom input, model selection, and output display.
- **Application Logic Layer (Python Core):** Controls workflow execution, manages session state, and structures responses.
- **LLM Orchestration Layer (LangChain):** Manages prompt templates, sequential chaining, and model abstraction.
- **External AI Services Layer:** Provides LLM reasoning through Gemini API and Groq LLM API.

2.2 Process Flow

The system follows a sequential workflow where each stage depends on the output of the previous stage.


User opens the web application
User enters symptoms in Streamlit UI
System validates symptom input
Symptoms valid?
Yes
No
User selects LLM model (Gemini or Groq)
Show validation error message in UI
Generate Prompt 1: Identify possible conditions
Call LLM API
Receive possible conditions
Generate Prompt 2: Suggest precautionary measures
Call LLM API
Receive precautions
Generate Prompt 3: Provide general medication guidance
Call LLM API
Receive medication suggestions
Generate Prompt 4: Recommend nutrition guidance
Call LLM API
Receive nutrition recommendations
Aggregate all outputs
Display structured results in Streamlit UI

2.3 Information Flow

Information flow describes how data moves between different layers and modules.

Input Information

- User-entered symptoms (text)
- Selected model (Gemini / Groq)
- API key (entered temporarily for session use)

Processing Information

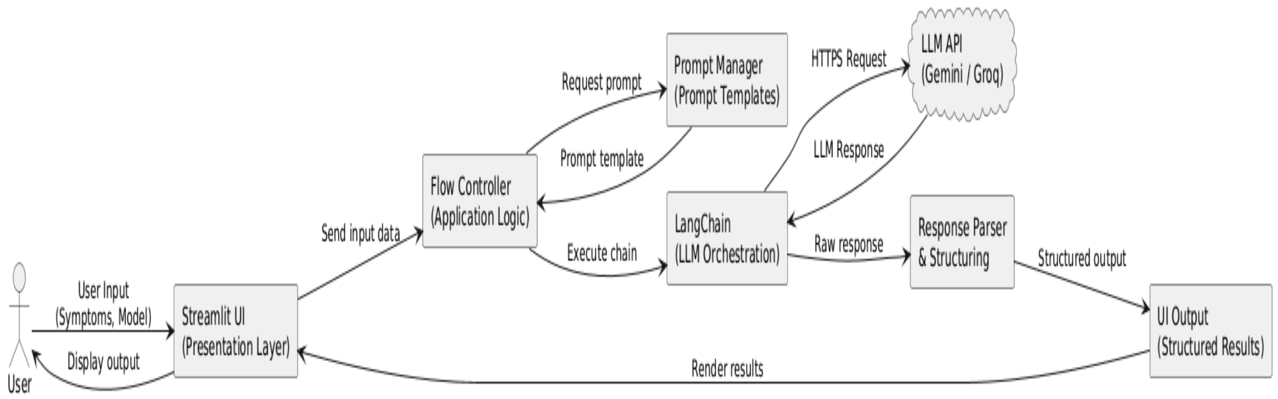
- Prompts generated from templates
- Context passed between sequential prompts
- Responses received from LLM APIs
- Output parsing and structuring

Output Information

The final output is shown in the following structured sections:

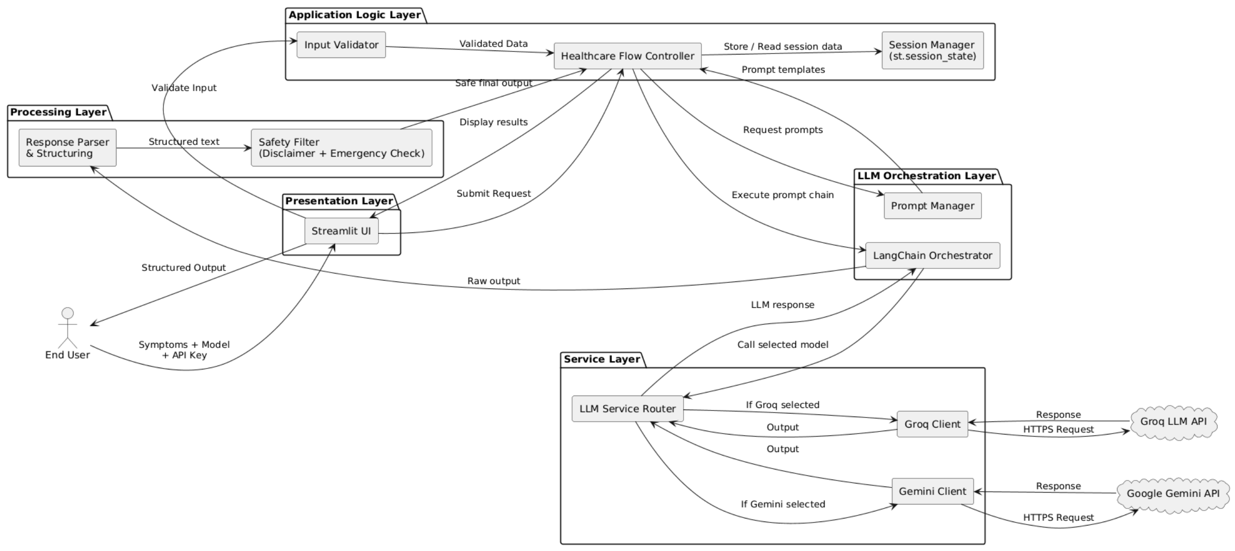
- Possible Conditions
- Precautions
- General Medication Guidance
- Nutrition Recommendations

Flow Representation


User
User Input (Symptoms, Model)
Display output
Streamlit UI (Presentation Layer)
Send input data
Flow Controller (Application Logic)
Request prompt
Prompt template
Prompt Manager (Prompt Templates)
Execute chain
LangChain (LLM Orchestration)
HTTPS Request
LLM Response
LLM API (Gemini / Groq)
Raw response
Response Parser & Structuring
Structured output
UI Output (Structured Results)
Render results

2.4 Components Design

The system consists of multiple components grouped into layers.



2.5 Key Design Considerations

The following design considerations were applied during system development:

1. Medical Safety

- The system is advisory only.
- The system avoids diagnosis claims.

2. Modular Architecture

- Separation into layers improves maintainability.
- Each module can be modified without affecting others.

3. Multi-Model Support

- Supports Gemini and Groq models.
- Allows better reliability and response comparison.

4. Privacy

- No user medical data is stored permanently.
- API keys are not saved.

5. Performance

- System is optimized for single-user prototype usage.
- Response time depends mainly on LLM API latency.

2.6 API Catalogue

This section lists the APIs used in the Healthcare Planning Assistant.

2.6.1 Google Gemini API

Purpose: Generates AI-based healthcare advisory responses.
Provider: Google
Usage in System: Used as one of the selectable LLM models.

Inputs:

- Prompt text
- API key

Outputs:

- Generated advisory response in natural language

2.6.2 Groq LLM API (Llama Models)

Purpose: Generates AI-based healthcare advisory responses using Groq-hosted Llama.
Type: External REST API (HTTPS)

Provider: Groq
Usage in System: Used as an alternative selectable LLM model.

Inputs:

- Prompt text
- API key

Outputs:

- Generated advisory response in natural language

3. Data Design

3.1 Data Model

The Healthcare Planning Assistant is currently designed as a prototype system and does not use a persistent database. The system mainly works with temporary runtime data collected from the user and generated by the LLM during execution.

Key Data Entities

1. User Input Data

- Symptoms (string / text)
- Selected Model (Gemini / Groq)
- API Key (temporary, session-based)

2. Generated Advisory Data

- Possible Conditions (list of strings)
- Precautions (list of strings)
- Medication Guidance (list of strings)
- Nutrition Recommendations (list of strings)

3. Session Data

- Current request state
- Intermediate LLM responses
- Final structured output

3.2 Data Access Mechanism

Since the system does not use a database, all data access is handled through in-memory mechanisms.

Data Access Methods

- **User Input Access:** Streamlit form inputs
- **Session Storage:** st.session_state for storing intermediate outputs
- **LLM Data Access:** API calls via LangChain integration

- **Output Access:** Structured results displayed through Streamlit components (st.markdown, st.write, etc.)

External Data Access

The system communicates with external AI services through HTTPS-based APIs:

- Google Gemini API
- Groq LLM API

3.3 Data Retention Policies

The Healthcare Planning Assistant is designed with privacy in mind. Since healthcare-related data is sensitive, the system avoids storing any personal medical data permanently.

Retention Policy

- User symptom data is stored only during the active session.
- API keys are not stored permanently.
- All generated responses are kept temporarily for display purposes.
- Once the user refreshes or closes the application, session data is cleared automatically.

Privacy Compliance Consideration

Since no persistent storage is used, the system reduces risks related to:

- patient data leakage
- long-term storage misuse
- compliance requirements

3.4 Data Migration

Data migration is not applicable in the current prototype because:

- No database is implemented
- No persistent storage is used
- No historical patient records are stored

4. Interfaces

4.1 User Interface (UI)

The Healthcare Planning Assistant provides a simple web-based user interface developed using Streamlit. The UI is designed to be minimal, interactive, and easy to use for non-technical users.

UI Features

- Symptom input textbox (multi-line)
- Model selection dropdown (Gemini / Groq)
- API key input field (secure input)
- Submit button to start processing
- Output displayed in structured sections:
    - Possible Conditions
    - Precautions
    - General Medication Guidance
    - Nutrition Recommendations

- Error messages for invalid input or missing API keys
- Medical disclaimer displayed in results

UI Design Considerations

- Clear section headings for readability
- Minimal user steps (input → submit → output)
- Responsive layout supported by Streamlit

4.2 External Interface (LLM APIs)

The system interacts with external AI services using HTTPS-based API calls. These services provide natural language reasoning and advisory responses.

External APIs Used

- Google Gemini API
- Groq LLM API (Llama Models)

4.3 Hardware Interface

No special hardware interface is required. The system runs on a standard machine capable of executing Python applications.

Minimum Requirements

- Laptop/PC
- Stable internet connection (for API calls)
- Web browser (Chrome, Edge, Firefox)

4.4 Software Interface

The software interfaces used by the system include:

Frontend

- Streamlit

Backend / Logic

- Python

LLM Orchestration

- LangChain

External Services

- Google Gemini API
- Groq LLM API

5. State and Session Management

The Healthcare Planning Assistant is a Streamlit-based web application. Since Streamlit applications rerun the script on each user interaction, the system requires a structured session management approach to preserve user inputs, intermediate LLM outputs, and final results during the active session.

Session State Elements

The following data is maintained temporarily in the session:

1. User Inputs

- symptoms: User-entered symptom description
- selected_model: Gemini / Groq
- api_key: User API key (not stored permanently)

2. Intermediate Workflow Data

- conditions_output
- precautions_output
- medications_output
- nutrition_output

3. Final Output

- final_structured_result: Aggregated and formatted response displayed in UI

4. Control Flags

- is_submitted: Tracks whether the user has submitted symptoms
- has_error: Tracks validation/API failures
- error_message: Stores error details for UI display

Session Lifecycle

Session Start

A session begins when the user opens the Streamlit application. Default values for session variables are initialized if not already present.

Session Execution

During processing:

- user inputs are stored in session state
- sequential prompt outputs are stored step-by-step
- the final structured output is stored for rendering

Session End

The session ends when:

- the user closes the browser tab, or
- the application is refreshed/restarted

Session Security Considerations

- API keys are only stored in memory during the active session.
- No session data is saved to disk or database.
- The system avoids storing any personally identifiable information (PII).
- All LLM communication happens via secure HTTPS.

Future Enhancement

In future versions, session management can be extended with:

- user login and authentication
- database-based user history
- encrypted storage for session data
- role-based access control for admin dashboards

6. Caching

The Healthcare Planning Assistant uses external LLM APIs (Google Gemini and Groq) for generating responses. Since LLM calls are time-consuming and cost-based, caching is an important design feature to improve performance, reduce repeated API calls, and enhance user experience.

7. Non-Functional Requirements

7.1 Security Aspects

Since the system deals with health-related inputs, security and privacy are critical even in prototype form. The Healthcare Planning Assistant is designed to minimize risk by avoiding storage of sensitive user information.

Key Security Requirements

- **API Key Protection:**
  API keys entered by the user are used only during the active session and are not stored permanently.
- **No Persistent Storage of Patient Data:**
  The system does not store symptoms, health conditions
- **Secure Communication:**
  All communication with external AI services (Gemini API and Groq API) occurs through secure HTTPS.
- **Session-Based Isolation:**
  User data is stored temporarily using Streamlit session state, ensuring that one user's session data is not shared with another.
- **Input Validation:**
  The system validates symptom input to prevent invalid or empty requests and reduce prompt injection risks.
- **Medical Disclaimer and Safety Warnings:**
  The system displays disclaimers to ensure the user understands that the output is advisory and not a medical diagnosis.

Security Limitations

- No authentication mechanism is implemented.
- No encryption at rest is required since no persistent storage is used.
- The system is not certified for healthcare regulatory compliance.

7.2 Performance Aspects

The system performance is primarily dependent on external LLM API response time. The prototype is optimized for single-user local execution.

Key Performance Requirements

- **LLM Response Time:**
  The expected response time depends on API latency and model complexity.
  Average response time may range from 5 to 20 seconds depending on the selected model.
- **Sequential Execution Efficiency:**
  Since the system runs 4 sequential LLM calls (conditions, precautions, medication, nutrition), the total time is the combined latency of all calls.
- **Session Caching:**
  Intermediate results are stored in st.session_state to prevent repeated calls during UI reruns.
- **Lightweight Processing:**
  Most processing is limited to prompt formatting and response parsing, which is computationally lightweight

Performance Limitations

- Not optimized for multiple concurrent users.
- No background job queue is used.
- API limits and throttling may affect performance.

Future Enhancements for Performance

- Combine multiple steps into fewer LLM calls.
- Add response caching with TTL.
- Use async API calls for faster pipeline execution.
- Deploy on cloud infrastructure for scalability.

8. References

1. **Streamlit Documentation**
   Streamlit Official Docs – UI framework for Python web applications.

2. **LangChain Documentation**
   LangChain Official Docs – Framework for LLM orchestration, prompt templates, and chaining.

3. **Google Gemini API Documentation**
   Google AI Gemini API – Official documentation for Gemini model integration.

4. **Groq LLM API Documentation**
   Groq Cloud API – Official documentation for accessing Llama-based models.

5. **Python Documentation**
   Python Official Docs – Core language reference and standard libraries.