Project - High Level Design

on

Rag-Chat-Bot

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.	ADITYA AKOLKAR	EN22CS301047
2.	AMAN GURJAR	EN22CS301105
3.	Dazzele Dhalani	EN22CS301312
4.	AMAN MAKWANA	EN22CS301106
5.	ADITYA RAJ SISODIYA	EN22CS301063

Group Name: 02D1

Project Number: GAI-02

Industry Mentor Name:

University Mentor Name: Prof. Ajaj Khan

Table of Contents

1. Introduction:

1.1 Scope of the document

- Defines the high-level architecture of the Healthcare Generative AI system
- Focuses on AI-based medical document generation using RAG and LLM models
- Covers major components: React frontend, FastAPI services, RAG engine, safety module, and MongoDB
- Describes document generation and retrieval workflows
- Explains overall data flow and system integrations

1.2 Intended Audience

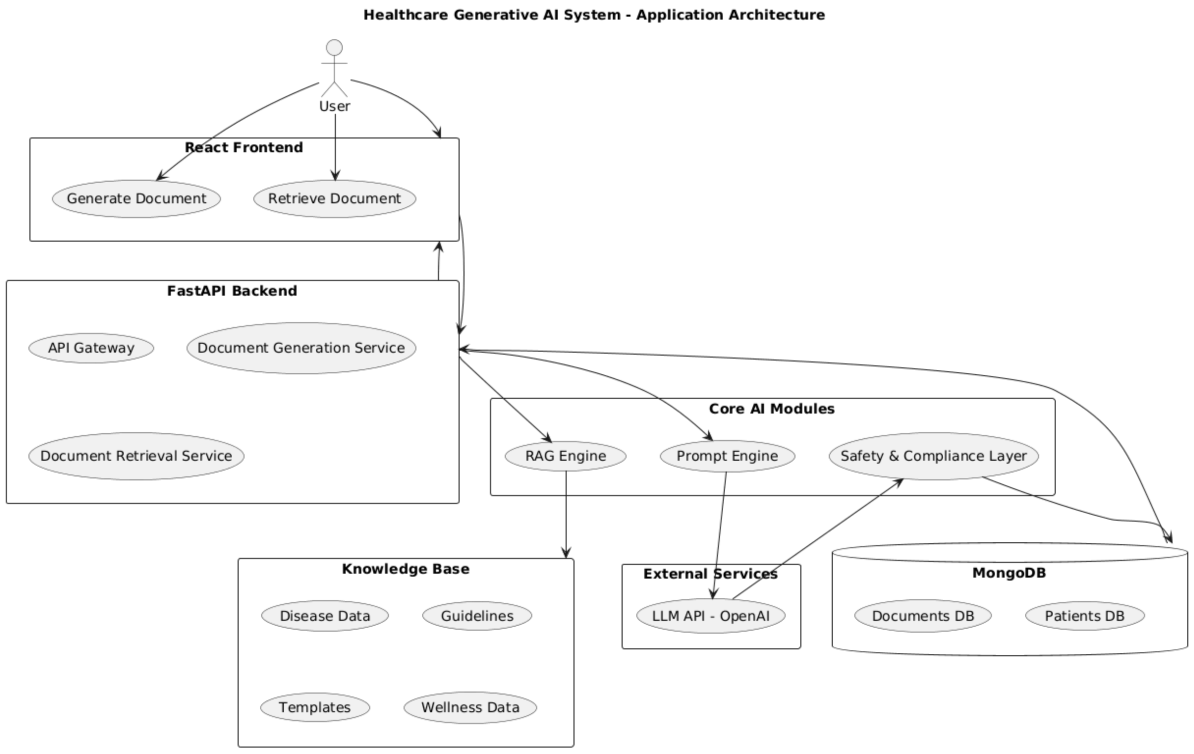
- Healthcare professionals (doctors, clinicians)
- Patients using AI-generated health documents
- Hospital and clinic management
- Organizations adopting Generative AI in healthcare systems

1.3 System Overview

- Healthcare Generative AI system designed to generate medical and educational documents
- Uses Retrieval-Augmented Generation (RAG) with Large Language Models (LLMs)
- Retrieves verified medical knowledge from structured datasets before generation
- Generates documents such as disease overviews, health suggestions, educational notes, and draft medical certificates
- Applies safety and compliance checks before delivering output
- Stores generated documents securely and allows retrieval using unique Document ID
- Ensures privacy-first access and controlled medical content generation

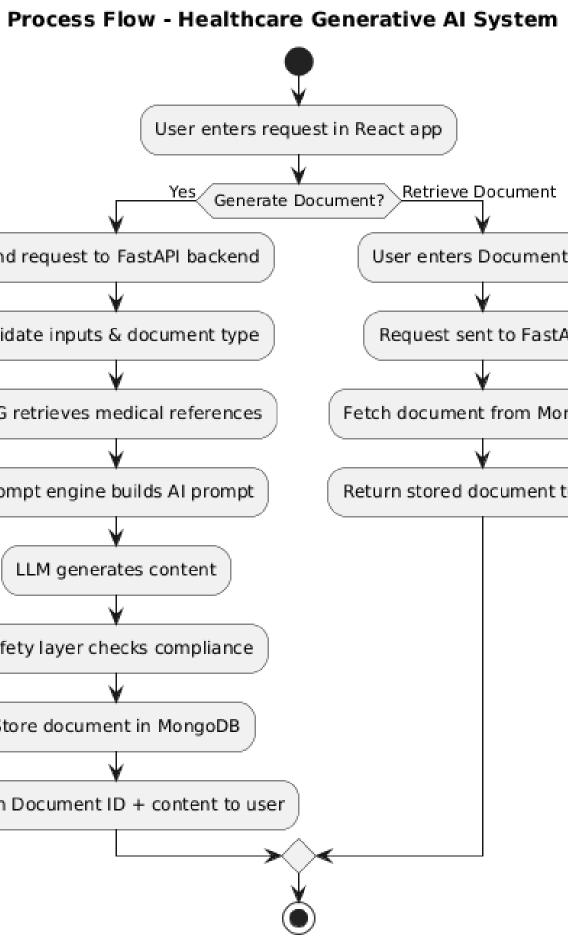
2. System Design:

2.1 Application Design



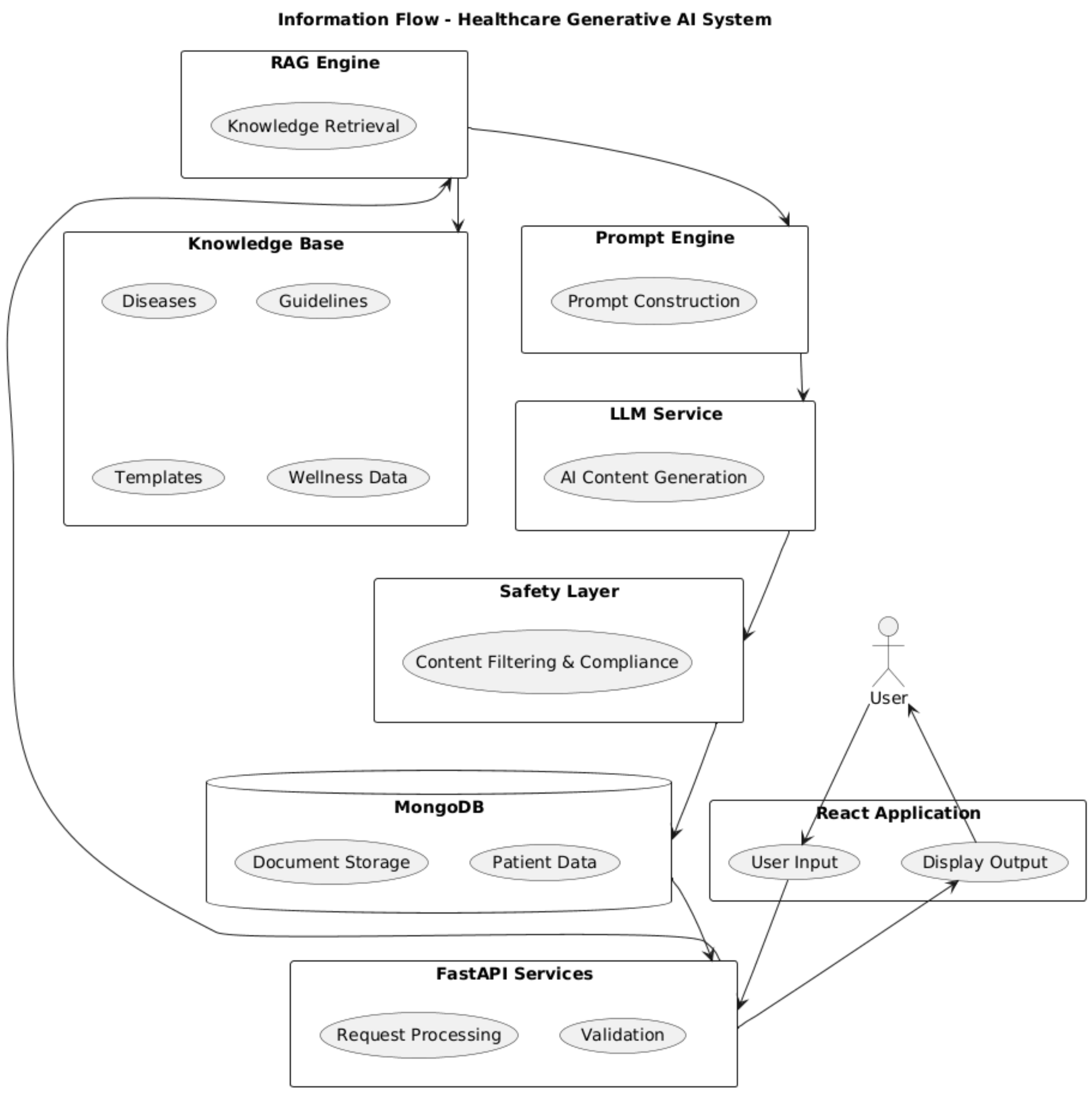
- React-based application used for document generation and retrieval
- FastAPI handles API requests and system logic
- RAG engine retrieves relevant medical knowledge from the dataset
- Prompt engine and LLM generate AI-based medical content
- Safety layer and MongoDB ensure secure storage and compliant output

2.2 Process Flow


Process Flow - Healthcare Generative AI System
User enters request in React app
Generate Document?
Yes
Retrieve Document
Send request to FastAPI backend
Validate inputs & document type
RAG retrieves medical references
Prompt engine builds AI prompt
LLM generates content
Safety layer checks compliance
Store document in MongoDB
Return Document ID + content to user
User enters Document ID
Request sent to FastAPI
Fetch document from MongoDB
Return stored document to user

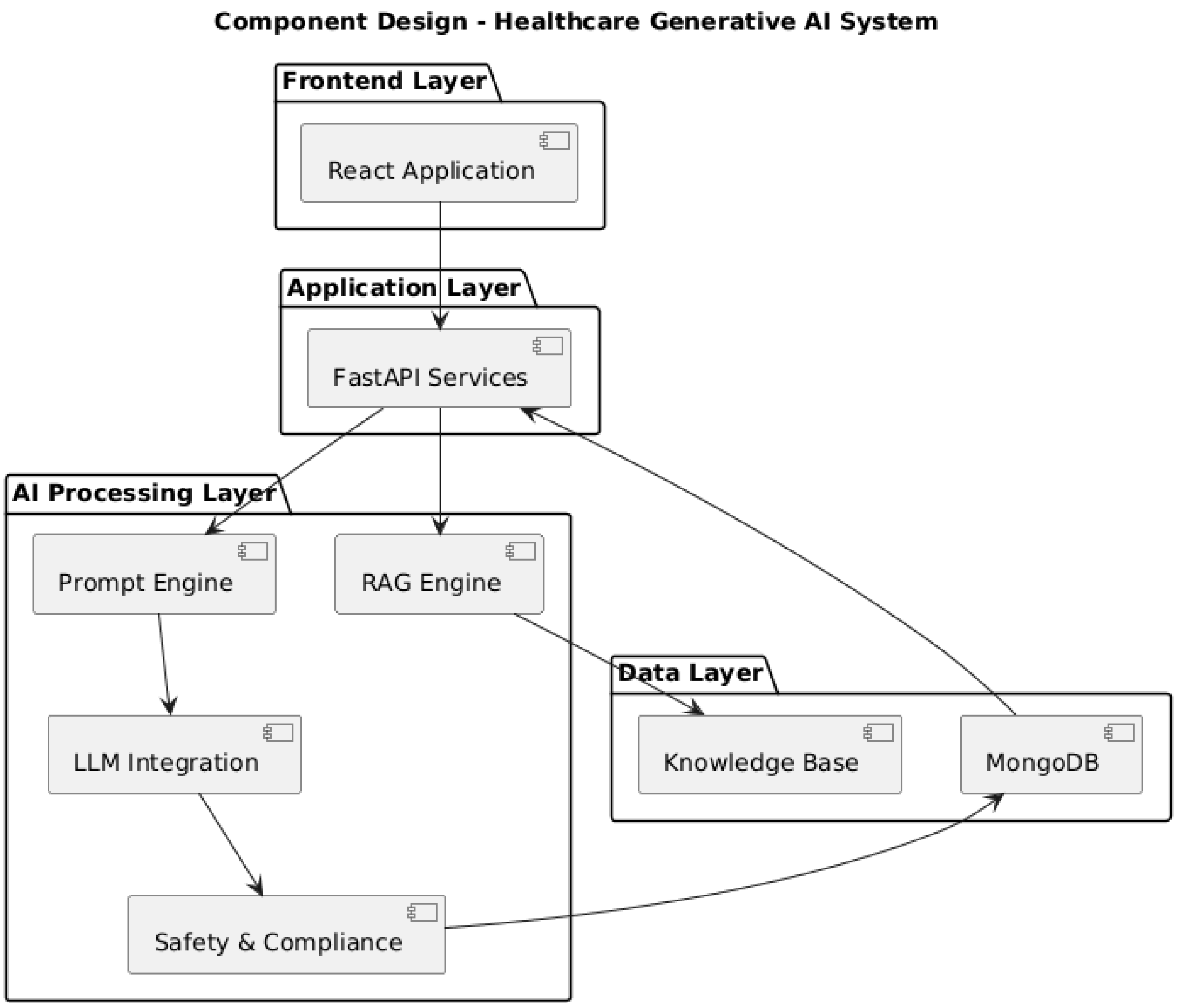
• User submits request to generate or retrieve a document through the React application
• FastAPI processes the request and validates inputs
• RAG engine retrieves relevant medical knowledge from the dataset
• Prompt engine and LLM generate AI-based document content
• Safety layer checks compliance and filters unsafe content
• Generated document is stored in MongoDB with a unique Document ID
• Document is returned to the user or retrieved later using the Document ID

2.3 Information flow



- User request is sent from the React application to FastAPI services
- RAG engine retrieves relevant medical data from the knowledge base
- Prompt engine and LLM generate the required document content
- Safety layer validates and filters the generated information
- Final document is stored in MongoDB and returned to the user

2.4 Components Design



- React application handles user interaction and requests
- FastAPI manages application logic and API processing
- RAG, Prompt Engine, and LLM handle AI-based document generation
- Safety module ensures compliance and secure output
- MongoDB and Knowledge Base manage document and medical data storage

2.5 Key Design Considerations

- Privacy-first approach with document access only through unique Document ID
- Use of RAG to ensure generation from verified medical knowledge sources
- Safety and compliance layer to prevent unsafe or incorrect medical content

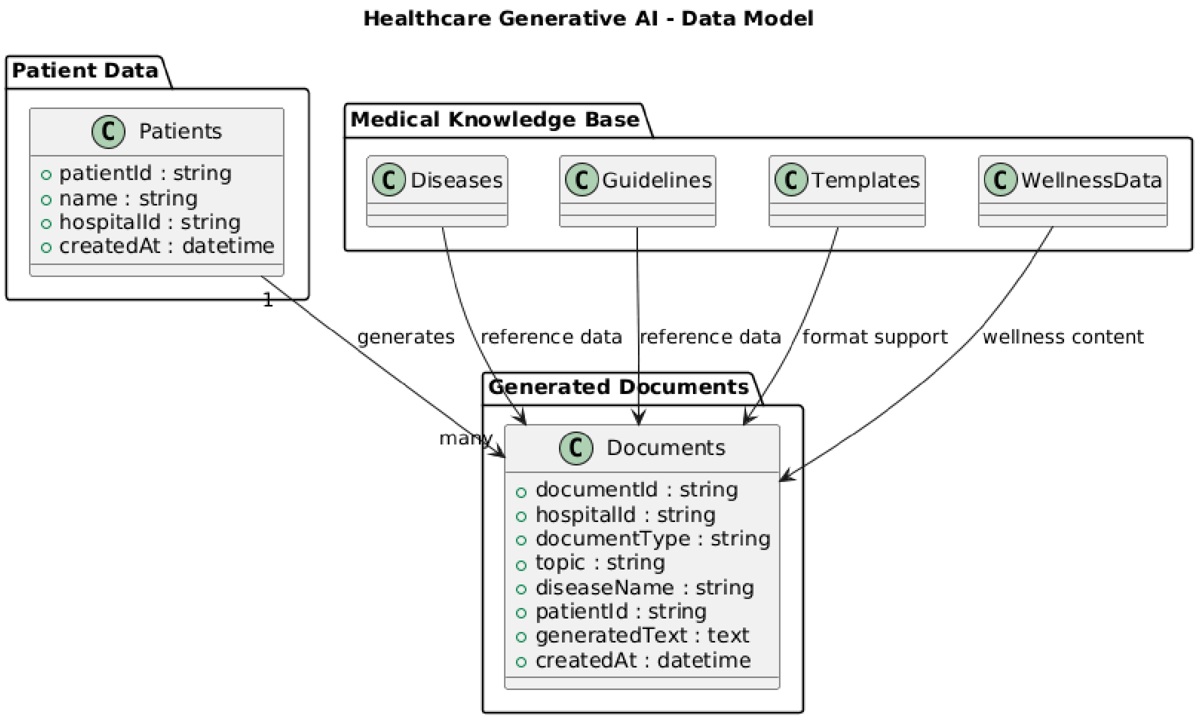
• Modular architecture for scalability and future enhancements
• Secure storage of documents and patient data using MongoDB

2.6 API Catalogue

• **/api/generate** – Generates AI-based medical documents using RAG and LLM
• **/api/document/{document_id}** – Retrieves document using unique Document ID
• **/api/patients** – Creates new patient record in the system
• **/api/patients/{hospital_id}** – Fetches list of patients for a hospital
• **/api/document-types** – Provides available document types and requirements
• **/** – Health check endpoint to verify system status

3. Data Design.

3.1 Data Model



• Data is stored in MongoDB using structured collections for scalability
• Main entities include Patients, Documents, and Medical Knowledge Base
• Each patient can have multiple generated documents
• Knowledge base data (diseases, guidelines, templates, wellness) supports AI document generation
• Document records store generated content, type, hospital ID, and timestamp for tracking and retrieval

3.2 Data Access Mechanism

• Data access is handled through FastAPI-based API endpoints
• MongoDB is used for storing and retrieving patient and document data
• RAG engine reads structured knowledge base files for content generation
• Prompt engine and AI modules access retrieved data for processing
• Secure read and write operations are performed before storing documents
• Document retrieval is allowed only through unique Document ID
• Access control ensures privacy and protection of medical information

3.3 Data Retention Policies

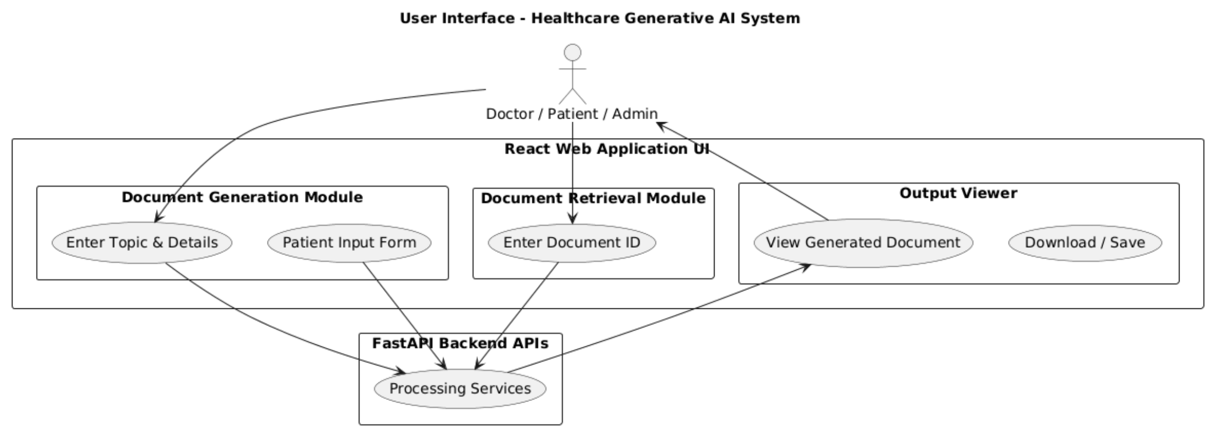
• Generated documents are stored securely in MongoDB for future retrieval
• Documents are accessible only through unique Document ID to maintain privacy
• Patient-related data is stored only for system functionality and tracking
• No unnecessary personal medical information is retained in the system
• Data retention follows privacy-focused and secure storage practices
• Old or unused data can be archived or deleted as per organizational policies

3.4 Data Migration

• Data migration allows transfer of records when system upgrades or database changes occur
• Supports movement of patient and document data between database versions
• Ensures data consistency and integrity during migration process
• Backup mechanisms are maintained before performing migration
• Migration supports scalability and future system expansion
• Enables integration with new storage systems or cloud platforms if required

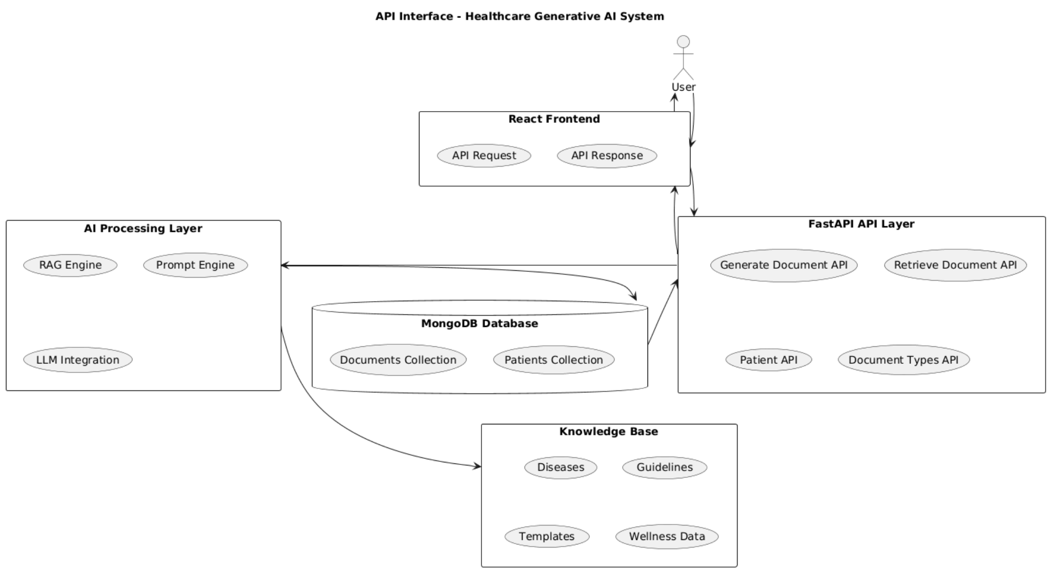
4. Interfaces

4.1 User interface:



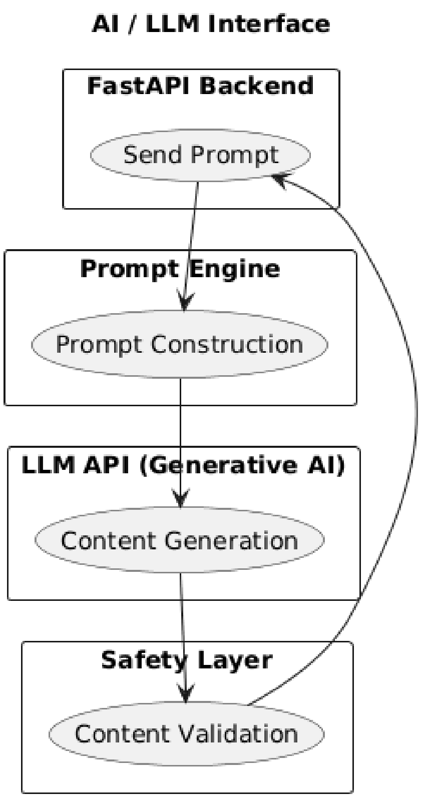
- React web application provides an easy and interactive user experience
- Separate modules for document generation and document retrieval
- Patient details and document topics collected through structured forms
- Generated documents displayed in a clear output viewer interface
- UI communicates with FastAPI backend through secure API requests

4.2 API interface



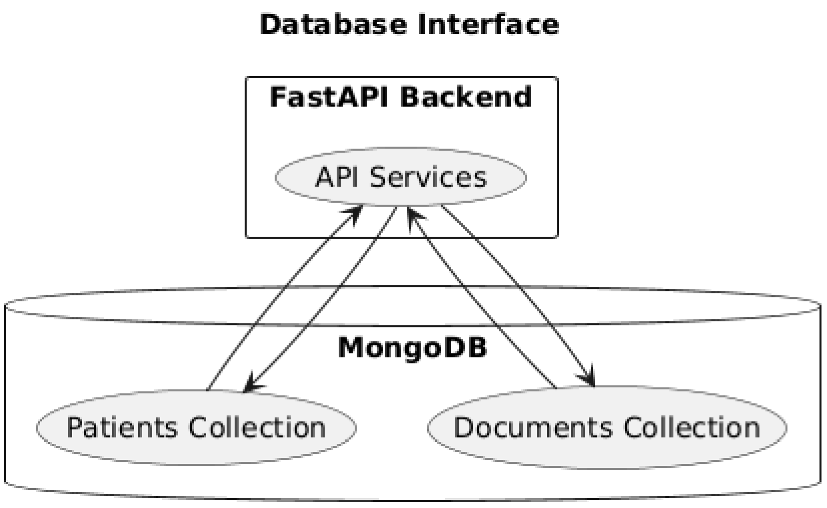
- React frontend communicates with backend through REST APIs
- FastAPI acts as the central API handling layer
- APIs interact with AI modules for document generation
- MongoDB stores and retrieves patient and document data
- Knowledge base is accessed through RAG during API processing

4.3 AI/ML INTERFACE



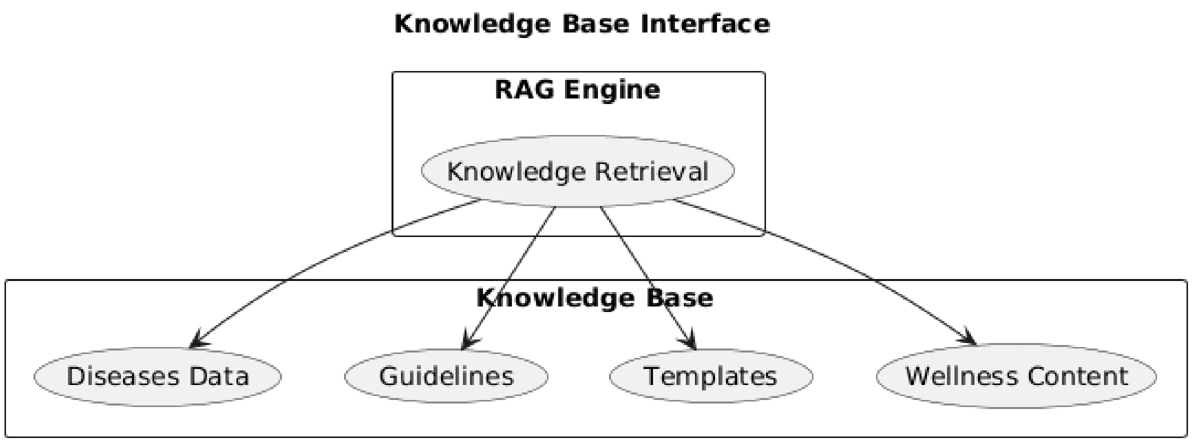
- Integration with external Generative AI (LLM API)
- Prompt engine prepares structured input for AI model
- LLM generates medical and educational content
- Safety layer validates generated output
- Ensures reliable and controlled AI-based document generation

4.4 Database interface



- MongoDB used for storing patient and document data
- Backend APIs perform read and write operations
- Supports secure and scalable data storage
- Enables document retrieval using Document ID
- Maintains structured storage for system records

4.5 Knowledge Base Interface



• Provides structured medical information for document generation
• Includes diseases, guidelines, templates, and wellness content
• Accessed by RAG engine during retrieval process
• Ensures content generation from verified knowledge sources
• Supports accurate and domain-specific AI output

5. State and session management

• Application manages state during document generation and retrieval processes
• React frontend maintains temporary user input and interaction state
• Backend processes each request independently using stateless APIs
• Document ID is used to maintain session reference for generated documents
• MongoDB stores persistent data such as patients and generated documents
• No long-term user session is stored to ensure privacy and security

6. Caching

• Caching helps reduce repeated processing and improves system response time
• Frequently accessed medical knowledge data can be cached for faster retrieval
• Generated document responses may be temporarily cached to avoid regeneration
• Backend services manage temporary storage of request and retrieval data
• Caching reduces load on AI models and database operations
• Ensures efficient and scalable system performance

7. Non-Functional Requirements

7.1 Security Aspects

• Privacy-first design with document access only through unique Document ID
• Secure API communication between frontend and backend services
• Controlled access to patient and document data in MongoDB
• Safety layer prevents unsafe or misleading medical content generation
• No sensitive personal medical data exposed through the system

7.2 Performance Aspects

• Fast document generation using optimized RAG and LLM processing
• Efficient data retrieval from MongoDB for quick response time
• Caching reduces repeated processing and improves system speed
• Scalable architecture supports multiple users simultaneously
• Optimized API communication ensures smooth system performance

References

• FastAPI Documentation – for backend API development
• MongoDB Documentation – for database design and data management
• React Documentation – for frontend application development
• FAISS and Sentence Transformers – for vector search and RAG implementation
• Generative AI / LLM API Documentation – for AI-based content generation
• Healthcare content guidelines and medical reference materials used in knowledge base