**Team : 16**

**Deep Learning**

CMPE 258 [FALL 2025]

**Team Member:** Prateek Sharma - [prateek.sharma03@sjsu.edu](mailto:prateek.sharma03@sjsu.edu) - 018228418

Option 2 - LLMs and AI Agent

**Healio.ai - AI Health Care Assistant**

**Final Report**

# Abstract

Healio.ai is an advanced AI-powered telemedicine platform designed to bridge the gap between patient symptoms and professional medical consultation and verification. The system utilizes Large Language Models (LLMs) to conduct intelligent patient interviews, gathering comprehensive medical history and symptoms. It features a dual-interface design: a patient portal for interactive diagnosis and a doctor dashboard for reviewing AI-generated reports of the patients. Key innovations include real-time generation of PDF medical reports for approval from real doctors to satisfy users with LLM’s diagnosis, historical medical references in diagnosis, FHIR (Fast Healthcare Interoperability Resources) compliance for data exchange, a robust role-based authentication system, DuckDuckGo web search if symptoms are extremely rare and complex. The platform demonstrates how AI can streamline clinical workflows, reduce administrative burden especially for routine queries, avoid wait time for the doctor appointments and improve the accessibility of preliminary medical assessments.

# 1. Introduction & Problem Description

Access to timely medical advice is a growing challenge in modern healthcare. Patients often face long wait times for simple consultations. While fast access is one of the problems, doctors spend costly time piecing together history, increasing the risk of missed details. Healio.ai addresses this by automating the preliminary consultation phase. Targeting both patients seeking immediate health guidance and healthcare providers needing efficient workflow tools, Healio.ai serves as an intelligent intermediary. It is significant because it not only improves patient engagement but also ensures that doctors receive structured, standardized clinical data and reports, allowing them to focus on diagnosis and treatment report review.

# 2. Background & Related Work

# Key references

* **Gemini Flash** - low-latency, long-context, multimodal LLMs available via Google Cloud’s Gemini API.
* **Agent frameworks:**
  + **CrewAI** -   CrewAI is a Python framework for building **multi-agent** workflows where each agent has a role (e.g., Researcher, Diagnostician, Scribe), a goal, and access to tools (web search, PDF reader, databases, APIs). You define a task graph—who does what, in what order—and CrewAI handles message passing, hand-offs, retries, and structured outputs. It’s **LLM-agnostic** (you can plug in Gemini/other models), supports memory/context sharing between agents, and makes it easy to constrain agents to produce JSON your backend can validate. CrewAI turns a big problem into coordinated, testable steps run by specialized AI teammates.
* **FHIR** - canonical standard for interoperable clinical data and REST behaviors (resources like Patient, Observation, DiagnosticReport, DocumentReference).

## Existing products

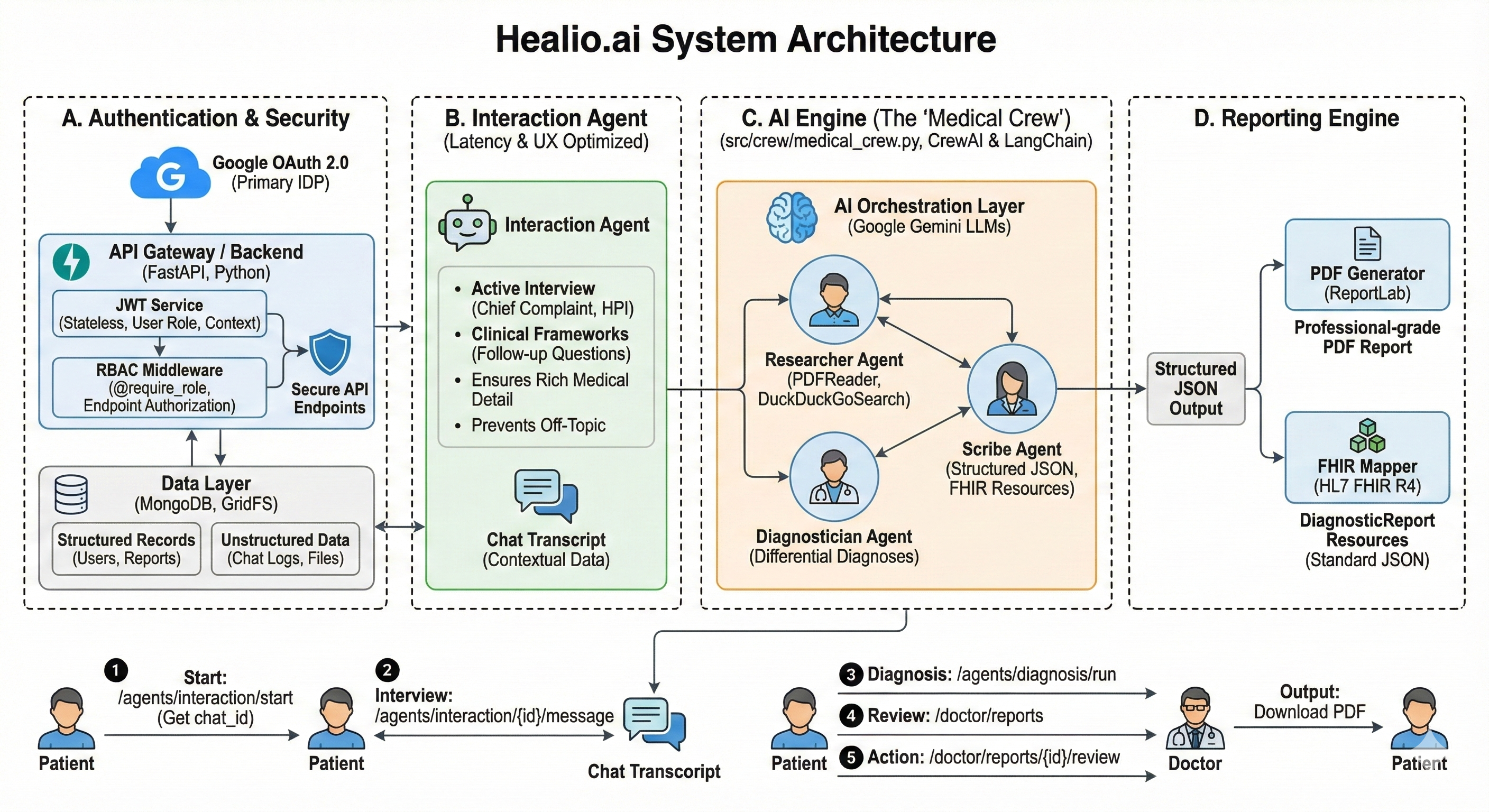
* **Telemedicine / AI triage chat:**
  + **K Health** — consumer symptom checker + clinician handoff/virtual care.

## ****Ambient clinical documentation at scale:****

* + **Nuance DAX Copilot (Microsoft)** — ambient note generation integrated with major EHRs; clinician review/attestation workflow.
  + **Abridge** — AI documentation used by health systems (e.g., UChicago Medicine), turning conversations into EHR notes.

These show that LLMs are already used for chat-based triage and clinical note drafting in production settings.

# 3. System Design



# 1. High-Level Overview

Healio.ai follows a modern client-server architecture designed for scalability, security, and interoperability. It separates the concerns of user interface, business logic/AI processing, and data persistence.

## Layers:

* Frontend (Client): A React-based Single Page Application (SPA) powered by Vite, providing a responsive interface for Patients and Doctors made using Loveable.ai.
* API Gateway / Backend: A FastAPI (Python) server that acts as the central orchestrator. It handles authentication, request validation, and routing.
* AI Orchestration Layer: A dedicated module using CrewAI and LangChain to manage autonomous agents (Medical Researcher, Diagnostician, Scribe) powered by Google Gemini LLMs.
* Data Layer: MongoDB for flexible storage of unstructured medical data (chat logs) and structured records (reports, users).

### 2. Component Breakdown

## A. Authentication & Security

* JWT (JSON Web Tokens): Stateless session management. Tokens contain user role (patient, doctor), sub (user\_id), and necessary context.
* RBAC (Role-Based Access Control): Middleware decorators (@require\_role) ensure users can only access endpoints authorized for their specific role.
* Google OAuth 2.0: Primary IDP for secure user onboarding.

## B. Interaction Agent

We kept the Interaction Agent separate from CrewAI for latency and user experience reasons.

* It actively interviews the patient to gather the "Chief Complaint" (what's wrong) and the "History of Present Illness" (HPI).
* It follows clinical frameworks to ask relevant follow-up questions (e.g., "How long has it hurt?", "Does anything make it better?").
* It ensures the chat transcript contains enough rich medical detail so that when the Medical Crew finally runs, they have high-quality data to make a diagnosis.
* It prevents the user from going off-topic or asking non-medical questions during the intake phase.
* Context Fetching: The system queries `db.medical\_histories` for the patient's static background. For historical context, it uses an **LLM-Powered Keyword Retrieval System :** the LLM analyzes the current query to select relevant tags (e.g., "migraine") and fetches specific past conversations linked to those keywords from the database.

## C. AI Engine (The "Medical Crew")

## Located in src/crew/medical\_crew.py, this is the core intelligence using multiple CrewAI agents:

* Researcher Agent: Analyses chat history and attachments using tools like PDFReader and DuckDuckGoSearch when needed.
* Diagnostician Agent: Formulates differential diagnoses based on findings and histories.
* Scribe Agent: Formats the final output into a structured JSON and FHIR resources.

## D. Reporting Engine

* PDF Generator: Uses ReportLab to programmatically create professional-grade PDF reports from JSON data.
* FHIR Mapper: Converts internal JSON data models into HL7 FHIR R4 standard resources (DiagnosticReport) for interoperability.

# 3. API Endpoint Reference

### Authentication (/auth)

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /auth/google/verify | Verifies Google ID Token, creates/updates user account, issues JWT. |
| POST | /auth/dev/login | Dev Only: Bypass Google Auth for local testing. |

### Agents & AI (/agents)

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /agents/interaction/start | Initializes a new diagnostic chat session. |
| POST | /agents/interaction/{chat\_id}/message | Sends user message to AI. Triggers real-time agent response. |
| POST | /agents/diagnosis/run | Triggers the full Medical Crew to analyze chat & generate report. Automatically includes analysis of any uploaded files (images/PDFs) in the diagnosis. |

### Chat Management (/chats)

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| GET | /chats | List all chat sessions for the current user. |
| GET | /chats/{chat\_id} | Retrieve full transcript of a specific chat session. |

### Patient Portal (/patients)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Endpoint | | | Description |
| GET | /patients/{id}/reports | | | List all medical reports for a specific patient. |
| GET | /patients/{id}/reports/{report\_id} | | | View details of a specific report. |
| GET | /patients/{id}/reports/{report\_id}/pdf | | | Detailed PDF download of the report. |
| PUT | /patients/{id}/profile | | | Update patient demographics (Age, Name, Sex). |
|  | |  |  | |
| POST | | /patients/{id}/uploads | Upload medical images/PDFs. Files are analyzed using Gemini Vision and summaries are stored for use in chat and diagnosis. | |
| GET | | /patients/{id}/uploads/{file\_id} | Download a specific uploaded file. | |
|  |  | | |  |

### Doctor Portal (/doctor)

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| GET | /doctor/reports | Dashboard: Global list of all patient reports needing review. |
| POST | /doctor/reports/{id}/review | Submit review (Approve/Reject) and add clinical comments. |
| PUT | /doctor/profile | Update doctor's professional details (Specialty, Bio). |
| GET | /doctor/fhir/DiagnosticReport | Interoperability: Fetch reports in standard HL7 FHIR R4 JSON format. |

### File Management (/upload)

|  |  |  |
| --- | --- | --- |
| Method | Endpoint | Description |
| POST | /upload | Upload a file to GridFS. Returns a file\_id for AI analysis. |

### 4. Data Flow Example: "Start to Finish Diagnosis"

1. Start: Patient logs in and calls /agents/interaction/start to get a chat\_id.
2. Interview: Patient sends symptoms via /agents/interaction/{id}/message. The Interaction Agent replies, building history\_str context.
3. Diagnosis: Patient clicks “Generate Report”. Frontend calls /agents/diagnosis/run.  
    • Backend calls Medical Crew.  
    • Agents analyze context + files -> Produce JSON Diagnosis.  
    • Backend saves Report object to MongoDB.
4. Review: Doctor logs in, sees report in /doctor/reports.
5. Action: Doctor reviews report (/doctor/reports/{id}/review).

Output: Patient downloads final PDF

# 4. Implementation Details

* Languages & Frameworks:
  + **Core:** Python 3.11, FastAPI, Pydantic, Motor (MongoDB)
  + **AI:** CrewAI, LangChain, Google Gemini API
  + **Tools:** ReportLab, PyPDF, Pandas, DuckDuckGo Search, PyJWT
* Key Decisions:
  + Asked agent to ask multiple questions instead of single individual questions to cut token cost.
  + We chose FHIR R4 as the data standard to ensure the system is future-proof and compatible with Electronic Health Records (EHR).
  + We generated and stored keywords and summary for each chat and medical history and used LLM to fetch only relevant data in decision making that matches the required keywords.
  + Separate interaction agent for fast response to users.
  + Filter for critical and urgent queries.
  + Asynchronous processing was implemented for report generation to ensure a smooth user experience during chat sessions.

**Code Repository**: https://github.com/prateeks7/healio.ai

# 5. Evaluation & Testing Results

We evaluated the system based on functional correctness and user experience. We tested the application using user interface by implementing multiple scenarios. For some important functionalities check we created evaluation notebook to check and score if the flow was working as expected.

We tested 4 functionalities –

* Medical History Reference – If the agent takes in related medical history in conversation.
* Chat History Reference – If the agent takes related chat history in reference.
* Image Reading – If the app can read user uploaded image.
* Document Reading – If the app can read user’s uploaded document.

We created model\_evaluation\_full with following details -

* We use a Hybrid Scoring System implemented directly in Python - a custom script using two layers of logic:
* AI Judge: We call in a LLM Judge to evaluate our model.
  + Tool: Gemini Flash. Logic: We send a prompt to Gemini: "Scenario: Diabetes. User said: 'Foot sore'. Agent said: [Agent's Response]. Did the agent correctly explicitly use the context of Diabetes? Return 0.0 to 1.0."
* Diabetes: Judge Asked: "Does the agent acknowledge the patient has Type 2 Diabetes?" Result: Yes (Agent correctly retrieved this from patient history.)
* Fracture (Memory): Judge Asked: "Does the agent recall the specific injury (Right Index Finger Fracture) from the past chat?" Result: Yes (Agent retrieved this from vector memory).
* PDF Analysis: Judge Asked: "Does the agent correctly extract the value WBC 15.0 from the lab report?" Result: Yes (Agent read the text content we injected).
* Vision Analysis: Judge Asked: "Does the agent correctly describe the image as a Chat Interface?" Result: Yes (Agent generated a summary using the new Vision Service).

# 5.1 Scores

Confirmed working of 4 critical scenarios (Medical History, Memory Recall, PDF Analysis, Vision Analysis).

# 5.2 FHIR Evaluation

A screenshot of a computer

AI-generated content may be incorrect.

We also evaluated FHIR Format using <https://validator.fhir.org> which ended up showing no error in format.

# 6. References

[1] HL7 FHIR Standard, Release 4. Available: http://hl7.org/fhir/

[2] CrewAI Documentation. Available: <https://docs.crewai.com/>

[3] React Documentation. Available: <https://react.dev/>

[4] FastAPI Documentation. Available: <https://fastapi.tiangolo.com/>

[5] Google. “Gemini API — Overview.” <https://ai.google.dev/gemini-api/docs>

[6] Google. “Gemini API — Vision.” <https://ai.google.dev/gemini-api/docs/vision>