Informe Desarrollo de Agente

**Evaluación Parcial*: [2]***

Integrantes: Aaron Gorigoitia, Xavier Blanco

Profesor: Giocrisrail Godoy Bonillo

Asignatura: Ingeniería de soluciones con inteligencia artificial

Fecha: 29/10/2025

### **Introducción**

### Este proyecto corresponde a la segunda parte del proyecto iniciado en la evaluación parcial N°1, donde tuvimos que crear un agente funcional para la clínica "Sonrisa Saludable". En esta etapa, se profundizó en la implementación, donde se realizó la integración y orquestación avanzada de herramientas de consultar (cómo buscar pacientes o disponibilidad), escritura (registrar notas, enviar confirmaciones) y razonamiento (clasificar urgencia, recomendar especialista), donde en este caso trabajaremos principalmente con el framework Langchain para lograr un flujo de trabajo organizacional automatizado.

### **Diseño e implementación del agente**

### Se creó un repositorio en GitHub, donde se encuentra todo el código fuente del proyecto. Este se trabajó con el framework de Langchain (versión 0.1.0 o superior), el cual nos permite integrar de forma modular los componentes necesarios: módulos de recuperación de información (desde bases de datos o historiales), razonamiento (usando el LLM) y generación de respuestas. La elección de Langchain se justifica por su robustez para definir cadenas (chains) y agentes, facilitando la orquestación entre distintos módulos y herramientas, lo cual garantiza una mejor escalabilidad si se necesitaran añadir más funciones a futuro y compatibilidad con diversas fuentes de datos y modelos.

### Para la integración de herramientas, se aprovechó el mecanismo de Function Calling de OpenAI, donde las funciones python que definen las herramientas (como buscar paciente, agendar cita, etc.) Se describen con esquemas claros (usando pydantic), permitiendo que el LLM las invoque de forma precisa y estructurada.

### Las respuestas del agente son resueltas basándose en embeddings vectoriales para la búsqueda semántica en la memoria y utilizando el modelo gpt-4-turbo. Se optó por GPT-4-Turbo sobre modelos anteriores debido a su superior capacidad de seguimiento de instrucciones complejas, razonamiento avanzado y su ventana de contexto más amplia (128k tokens), crucial para mantener la coherencia en conversaciones largas o con mucha información. El sistema incluye módulos específicos de escritura (para registrar información en bases de datos o enviar notificaciones) y validación (para asegurar que los datos como fechas, ruts, entre otras, y sean correctos antes de usarlos), donde el agente genera las respuestas finales basándose en el razonamiento aplicado al contexto clínico específico de "Sonrisa Saludable".

### **Configuración de memoria y recuperación de contexto**

### Para este apartado, se implementaron mecanismos de memoria que combinan **corto y largo plazo** para asegurar la continuidad y coherencia en flujos prolongados.

* **Memoria de Corto Plazo:** Se configuró una memoria de Langchain, la cual almacena los últimos intercambios de la conversación actual (por ejemplo, los últimos 10 mensajes). Esto permite responder a preguntas inmediatas sobre lo recién hablado y evita el desborde de contexto en prompts muy largos.
* **Memoria de Largo Plazo:** Se utilizó una base de datos vectorial, en este caso ChromaDB, donde se almacenan *embeddings* de conversaciones pasadas y notas clínicas relevantes. ChromaDB se eligió por ser de código abierto, fácil de configurar localmente para el proyecto y tener buena integración con Langchain. Este sistema es el que permite al agente mantener la coherencia a través de diferentes sesiones o en tareas muy prolongadas.
* **Recuperación de Contexto Semántico:** Cuando el agente necesita recordar información pasada relevante para la consulta actual, utiliza un proceso de recuperación semántica. La consulta actual se convierte en un embedding y se busca en ChromaDB los fragmentos de conversaciones o notas anteriores cuyos embeddings sean más cercanos (usando similitud coseno). Esta técnica, a menudo parte de un sistema RAG (Retrieval-Augmented Generation), permite encontrar información conceptualmente relacionada, aunque no use exactamente las mismas palabras, asegurando así la continuidad del flujo de trabajo.

### **Planificación y toma de decisiones**

### El agente incorpora un módulo de **planificación de tareas** que secuencia las actividades necesarias para responder a una solicitud del usuario, como análisis de la petición, búsqueda de información, ejecución de herramientas y generación de la respuesta final. Se emplea una estrategia basada en el enfoque **ReAct (Reason + Act)**, el cual es un patrón soportado por los agentes de Langchain. Este enfoque permite que el agente "piense" (Reason) qué acción tomar a continuación, ejecute esa acción (Act) - que puede ser usar una herramienta o responder al usuario - y observe el resultado para decidir el siguiente paso.

### Esto permite que el agente ajuste su comportamiento ante consultas complejas o condiciones cambiantes.

* **Ejemplo 1 (Información Faltante):** Si un paciente consulta por un tratamiento no registrado en la base de conocimiento (buscar\_protocolo\_clinico no devuelve resultados), el agente razona que falta información, decide no inventar una respuesta y en su lugar sugiere una cita presencial para evaluación.
* **Ejemplo 2 (Cambio de Solicitud):** Si el usuario pide una cita para el lunes a las 10 am, y luego dice "mejor el martes por la tarde", el agente detecta el cambio, razona que debe descartar la búsqueda anterior y realizar una nueva consulta de agenda con los nuevos parámetros (consultar\_agenda con fecha=martes, horario=tarde), manteniendo el resto de la información del paciente.

Viendo estos ejemplos, se demuestra cómo la planificación y toma de decisiones implementadas permiten al agente manejar tareas con múltiples etapas y adaptarse a condiciones cambiantes.

### **Documentación técnica**

### La documentación detallada del diseño e implementación del agente, incluyendo las decisiones de arquitectura, justificación de componentes y las instrucciones precisas para ejecutar el sistema, se encuentran dentro del archivo README.md adjunto en la raíz del proyecto de GitHub. Dicho README se complementa con diagramas de orquestación y flujo de trabajo que ilustran visualmente la interacción entre los componentes del agente y su relación con el flujo automatizado.

### **Referencias bibliográficas**

### ChromaDB Team. (2024). *ChromaDB Documentation*. Chroma. <https://docs.trychroma.com>

### LangChain AI. (2024). *LangChain Python Documentation*. LangChain.<https://python.langchain.com/docs/>

### OpenAI. (2024). *GPT-4 Technical Report* (arXiv:2303.08774). arXiv.<https://doi.org/10.48550/arXiv.2303.08774>