

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE HÁBITAT INFRAESTRUCTURA Y

CREATIVIDAD

CARRERA DE: SISTEMAS DE INFORMACIÓN



PROYECTO DE TRABAJO DE TITULACIÓN

TEMA: Implementación de un Servidor para la orquestación en tiempo real de Agentes de IA Asistentes Médicos.

Caso de Estudio: Empresa Zeropapers

AUTOR:

RONALD FRANCISCO RODRIGUEZ CUEVA

TUTOR:

Ing. Charles Escobar

QUITO, OCTUBRE DE 2025

## **DEDICATORIA**

A mis padres, quienes con su amor incondicional y sacrificio constante se convirtieron en el cimiento sobre el cual pude construir este sueño profesional, pues sus palabras de aliento en los momentos de mayor duda fueron la fuerza necesaria para no rendirme jamás. Sin su guía y sus valores, este título no tendría el mismo significado, ya que ellos me enseñaron que la verdadera ingeniería empieza con la perseverancia y la honestidad.

A mis hermanos, por ser mis compañeros de vida y los primeros en celebrar cada pequeño avance en mi formación académica. Gracias por entender mis ausencias durante las largas jornadas de estudio y por recordarme siempre que, más allá de los códigos y servidores, lo más importante es la familia que camina a nuestro lado.

A mis amigos de la carrera, quienes pasaron de ser compañeros de aula a convertirse en pilares fundamentales de mi aprendizaje. Juntos compartimos el estrés de los proyectos, las noches sin dormir configurando redes y la satisfacción de ver un sistema funcionando por primera vez. Gracias por enseñarme el valor del trabajo en equipo, por las discusiones técnicas que me hicieron crecer y por ser esa red de apoyo que convirtió los desafíos de la ingeniería en experiencias inolvidables.

Finalmente, a la vida, por permitirme aprender de cada error y convertirme en la persona y el profesional que soy hoy. Este logro es de todos ustedes, porque cada uno puso una pieza esencial en el código de mi vida para que hoy este proyecto sea una realidad funcional.

## **AGRADECIMIENTO**

En primer lugar, expreso mi más profundo agradecimiento a mis profesores de la carrera de Sistemas de Información, quienes con su dedicación y paciencia compartieron sus conocimientos y experiencias para guiarnos en el complejo camino de la tecnología. Su compromiso académico y sus consejos técnicos fueron fundamentales para formarme no solo como un profesional capaz de enfrentar retos como la orquestación de agentes de inteligencia artificial, sino también como una persona con ética y visión de futuro. Gracias por exigirnos excelencia y por enseñarnos que el aprendizaje en esta área es una evolución constante.

De manera muy especial, agradezco a mis padres, quienes han sido el soporte inquebrantable de mi vida y mi mayor motivación para culminar este proyecto de titulación. Gracias por su apoyo incondicional, por confiar en mi capacidad incluso cuando los desafíos técnicos parecían insuperables y por estar presentes en cada etapa de mi formación profesional. Este logro es el resultado de su esfuerzo y sacrificio, y representa el fruto de todo el amor y los valores que me han brindado desde el primer día.

Finalmente, agradezco a la institución y a todos aquellos que de forma directa o indirecta aportaron al desarrollo de esta investigación, facilitando los recursos y el entorno necesarios para convertir una idea técnica en una solución real para la sociedad.

## Resumen

La presente investigación desarrolla e implementa una arquitectura avanzada de orquestación para Agentes de Inteligencia Artificial mediante el uso del protocolo Model Context Protocol (MCP), con el objetivo de optimizar la interacción entre modelos de lenguaje de gran escala y sistemas de Historia Clínica Electrónica (HCE). El estudio aborda la problemática actual de las alucinaciones en la IA clínica debido a la falta de acceso a datos dinámicos y contextuales del paciente. Para ello, se diseñó un prototipo funcional sobre un servidor Debian 13 virtualizado en Proxmox, utilizando el framework FastAPI para la gestión de servicios asíncronos y una base de datos PostgreSQL para el manejo de datos híbridos.

Los resultados técnicos demostraron la viabilidad y eficiencia de la solución, alcanzando una latencia de respuesta de apenas 766 ms en la recuperación de cargas masivas de 1,002 registros médicos. Durante las pruebas de estrés, el sistema exhibió una robustez notable, manteniendo el uso de la unidad central de procesamiento en un 11.10% y una estabilidad de memoria RAM de 8.64 GiB, sin evidencia de fugas de recursos tras múltiples ciclos de ejecución. Se concluye que la implementación del estándar MCP bajo un transporte de flujo STDIO garantiza un entorno de alta disponibilidad, integridad y confidencialidad, proporcionando una herramienta escalable y segura que potencia el razonamiento clínico asistido por IA en entornos hospitalarios de alta demanda.

Finalmente, la investigación aporta una solución técnica viable a la brecha existente entre la inteligencia artificial generativa y la gestión de datos sensibles en salud, demostrando que es posible orquestar agentes inteligentes que respeten la privacidad del paciente mientras operan con una eficiencia computacional óptima. La arquitectura propuesta no solo reduce la carga operativa del personal médico mediante el acceso instantáneo a información contextualizada, sino que también sienta las bases para futuras implementaciones de medicina de precisión basadas en infraestructuras de código abierto y hardware escalable. Este trabajo valida que el uso de tecnologías como Debian y Proxmox,

en conjunto con el protocolo MCP, constituye un ecosistema robusto capaz de transformar la asistencia médica digital hacia un modelo más confiable, rápido y seguro.

## Table de Contenidos

1.	Capítulo 1: Introducción.....	1
1.1	Planteamiento del Problema .....	1
1.2	Justificación.....	1
1.3	Objetivos .....	3
1.3.1	Objetivo General .....	3
1.3.2	Objetivos Específicos.....	3
1.4	Alcance.....	3
1.5	Contexto de la Empresa.....	4
2.	Capítulo 2: Marco Teórico y Conceptual .....	4
2.1	Antecedentes .....	4
2.2	Marco teórico: Fundamentos y comparativa de Arquitecturas .....	6
2.2.1	Fundamentos de la Inteligencia Artificial Agentiva .....	6
2.2.1.1	Definición del Agente de IA Asistente Médico (Planificación, Razonamiento y Acción) .....	7
2.2.1.2	Requisitos Operacionales Críticos: Acción Transaccional y Datos Vivos en el Entorno Sanitario .....	7
2.2.2	Análisis Teórico de las Arquitecturas de Conectividad Legacy .....	8
2.3	Marco Conceptual .....	9
2.3.1	Servidor MCP (Gateway de Orquestación Segura).....	9
2.3.2	Agente de IA (Cliente MCP).....	10
2.3.3	Primitivas del Servidor MCP .....	10
2.3.4	Datos Vivos Operacionales y Clínicos.....	10
2.3.5	Protocolo de Intercambio JSON RPC .....	11
2.3.6	Conectividad Híbrida .....	11

2.3.7. Middleware.....	12
2.3.8. FASTAPI.....	12
2.3.9. SQLAlchemy.....	13
2.3.10. Endpoint de API.....	13
2.4 Fundamentación técnica y metodológica .....	13
2.4.1. Selección de la Metodología de Desarrollo Técnica.....	13
2.4.1.1. Justificación de eXtreme Programming (XP) para Desarrollo Individual .....	15
2.4.1.2. Principios de XP Aplicados al Desarrollo del Servidor (TDD y Refactorización).....	15
2.4.2. Stack Tecnológico y Herramientas de Implementación.....	16
2.4.2.1. Lenguaje de Programación y Framewor .....	16
2.4.2.2. Entorno de Infraestructura (Proxmox, Debian 13 y Docker).....	16
2.4.2.3. Herramientas para Simulación de Datos Vivos y Conectores.....	17
3. Capítulo 3: Metodología científica y técnica.....	17
3.1 Enfoque y Diseño de la Investigación.....	17
3.1.1. Tipo de Investigación.....	17
3.1.2 Diseño Experimental .....	19
3.2 Fases del Ciclo Desarrollo enfocado en XP.....	20
3.2.1. Fase de Planificación: identificación de las Historias de Usuario .....	20
3.2.2 Fase de Diseño: Definición de Interfaces y Esquemas de Interoperabilidad .....	21
3.2.3 Fase de Codificación y Pruebas (TDD).....	22
3.3. Arquitectura del Sistema e Implementación Técnica.....	23
3.3.1 Configuración del Entorno Virtual.....	23
3.3.2. Orquestación de Servicios con Docker: .....	24
3.3.3. Desarrollo del Middleware de Orquestación.....	26

3.3.4. Estrategia de Conectividad a la Fuente de Datos Híbrida .....	27
3.4. Pruebas y Protocolo de Validación .....	27
3.4.1. Escenarios de Evaluación Clínica .....	27
3.4.2. Métricas de Evaluación .....	28
3.5. Consideraciones Éticas y de Seguridad.....	30
3.5.1. Manejo de Datos de Prueba.....	30
3.5.2. Control de Acceso .....	30
4. Capítulo 4: Implementación del prototipo del Servidor .....	32
4.1. Análisis Comparativo de Arquitecturas de Integración .....	32
4.1.1 Limitaciones de las Arquitecturas Legacy: APIs Ad-hoc y Rigidez Estructural.....	32
4.1.2 Desafíos del RAG (Retrieval-Augmented Generation) en Entornos Clínicos .....	34
4.1.3. Model Context Protocol (MCP) para Entornos Clínicos .....	36
4.1.4. Ventajas del Modelo de Orquestación MCP vs. Métodos Tradicionales... 38	
4.1.5. Justificación de la Selección Arquitectónica del MCP. ....	39
4.1.6. Las Primitivas del MCP: Herramientas (Tools) y Recursos (Resources) .. 40	
4.1.7. El MCP como Única Solución que Cubre Seguridad, Datos Vivos y Acción .....	41
4.2.1. Despliegue de la Infraestructura en Proxmox VE y Debian 13 .....	42
4.2.2. Containerización con Docker .....	43
4.2.3. Implementación de la Base de Datos Híbrida .....	46
4.3.1. Construcción del Servidor MCP con Python y FastAPI .....	49
4.3.2. Implementación de la Capa de Recursos.....	50
4.3.3 Implementación de la Capa de Herramientas.....	51
<i>Diagrama de secuencia herramientas MCP</i> .....	52
4.3.4 Integración del Adaptador de Seguridad y Autenticación .....	53

4.3.5. Interfaz de Comunicación .....	57
4.4. Integración y Pruebas del Agente de IA Médico .....	58
4.4.1 Configuración del Cliente MCP y Conectividad con el LLM .....	61
4.4.2 Ejecución de Casos de Uso.....	64
4.4.3 Análisis de la Interacción JSON-RPC.....	70
4.5. Evaluación de Resultados y Métricas de Desempeño .....	72
4.5.1 Análisis de Latencia en la Orquestación .....	72
4.5.2. Evaluación del Rendimiento y Consumo de Recursos en Entorno Debian 13 .....	76
4.5.3 Validación del Cumplimiento de la Tríada de Seguridad .....	77
5. Capítulo 5: Conclusiones y Recomendaciones.....	79
5.1 Conclusiones .....	79
6. Bibliografía.....	81
7. Anexos.....	84
<b>Figura 1</b> .....	19
<b>Figura 2</b> .....	23
<b>Figura 3</b> .....	29
<b>Figura 4</b> .....	31
<b>Figura 5</b> .....	33
<b>Figura 6</b> .....	35
<b>Figura 7</b> .....	37
<b>Figura 8</b> .....	42
<b>Figura 9</b> .....	43
<b>Figura 10</b> .....	44
<b>Figura 11</b> .....	45
<b>Figura 12</b> .....	45
<b>Figura 13</b> .....	47
<b>Figura 14</b> .....	47
<b>Figura 15</b> .....	50

<b>Figura 16</b> .....	51
<b>Figura 17</b> .....	52
<b>Figura 18</b> .....	54
<b>Figura 19</b> .....	54
<b>Figura 20</b> .....	55
<b>Figura 21</b> .....	56
<b>Figura 22</b> .....	57
<b>Figura 23</b> .....	58
<b>Figura 24</b> .....	59
<b>Figura 25</b> .....	60
<b>Figura 26</b> .....	61
<b>Figura 27</b> .....	62
<b>Figura 28</b> .....	63
<b>Figura 29</b> .....	65
<b>Figura 30</b> .....	65
<b>Figura 31</b> .....	66
<b>Figura 32</b> .....	66
<b>Figura 33</b> .....	67
<b>Figura 34</b> .....	68
<b>Figura 35</b> .....	68
<b>Figura 36</b> .....	69
<b>Figura 37</b> .....	70
<b>Figura 38</b> .....	71
<b>Figura 39</b> .....	71
<b>Figura 40</b> .....	74
<b>Figura 41</b> .....	75
<b>Figura 42</b> .....	75
<b>Figura 43</b> .....	76
<b>Tabla 1</b> .....	38
<b>Tabla 2</b> .....	48
<b>Tabla 3</b> .....	54
<b>Tabla 4</b> .....	78

<b>Tabla 5</b> .....	84
<b>Tabla 6</b> .....	85

## **1. Capítulo 1: Introducción**

### **1.1 Planteamiento del Problema**

Las instituciones de la salud están invirtiendo en Agentes de IA Asistentes Médicos para optimizar la toma de decisiones clínicas y los procesos administrativos. Sin embargo, estos agentes están limitados debido a la fragilidad e ineficiencia de su conectividad actual a los sistemas operacionales de la clínica u hospital es por eso necesitan de conexiones seguras y datos más actualizados de diferentes fuentes de datos como los que se encuentran en sistemas internos y así los agentes realicen sus trabajos de manera eficiente.

En la actualidad estos Agentes Médicos recurren a herramientas que son soluciones creadas a medida como el uso de APIs personalizadas o usan RAG, esto causa una complejidad significativa al momento de hacer mantenimiento ya que no se convierten en integraciones dinámicas y con esto pueden aparecer puntos débiles en la seguridad. Además, estas integraciones personalizadas no garantizan la seguridad y el control de acceso requerido para manejar la información sensible de las Historias Clínicas Electrónicas, lo que impide que el Agente alcance su potencial máximo de autonomía. Por eso se puede apreciar que los agentes de IA no alcanzan su potencial máximo y no son realmente autónomos. Ante esta situación surge la necesidad de implementar un mecanismo más seguro y estandarizado para poder explotar los Agentes Inteligentes.

### **1.2 Justificación**

Las instituciones de salud hoy en día requieren de Agentes Inteligentes como Asistentes Médicos que ayuden a sus procesos internos y críticos es por eso es esencial en pensar la rapidez y la obtención de datos en vivo reales, entonces se crea una necesidad clara que supere el uso de arquitectura como RAG que son buenas para obtener datos estáticos, pero ineficientes para obtener datos vivos y estructurados. La migración a una nueva arquitectura y servidor es crucial para garantizar la rapidez y la actualidad de la información médica.

La importancia de utilizar este Servidor actualizado y eficaz es que nos brindara Habilitación de Agentes, Seguridad/Control, y Agilidad Empresarial con enfoque a la salud:

- **Habilitación de la IA Agentiva y la Acción:** Es crucial implementar un Servidor adaptable a cualquier entorno ya que transforma los de modelos de IA de generadores de texto a verdaderos agentes de acción. El proyecto demuestra como la elección de un servidor potente permite a los Agentes de IA en este caso los asistentes médicos ejecutar tareas medicas operacionales que son cruciales y también sustentar sus tomas de decisiones en datos vivo y actualizados de la historia clínica, proporcionando la autonomía del agente medico el cual le permite descubrir y utilizar funcionalidades del sector de la salud sin tener que requerir intervención humana constante.
- **Seguridad y Control Corporativo:** Un servidor estandarizado y rápido es vital para el cumplimiento normativo como la privacidad de los datos de la salud. Este servidor actúa como un puente auditado que controla que el Agente de IA solo acceda a los sistemas backend como las historias clínicas electrónicas con los permisos específicos del paciente. Se garantiza que los datos sensibles del paciente se mantengan seguros dentro del perímetro de seguridad sanitario, evitando la exposición a APIs externas o bases de datos vectoriales no controladas.
- **Agilidad y Escalabilidad Arquitectónica:** Este proyecto de titulación ofrece un valor estratégico a largo plazo en la infraestructura, ya que al implementar este servidor como protocolo ayudara a reutilizar el mismo servidor para que se puedan conectar múltiples Agentes de IA de la salud a las misma herramientas o sistemas para obtener la información, garantizando una integración eficiente.

“Hemos llegado a un punto de inflexión en el que los avances más significativos no provendrán de modelos de IA marginalmente mejores, sino de resolver este problema fundamental de integración. Aquí entra en juego el Protocolo de Contexto del Modelo (MCP), un estándar abierto destinado a convertirse en el lenguaje universal para la interacción agente-herramienta. Se entiende mejor a través de una analogía simple pero potente: MCP es el USB-C para la IA. Así como el USB-C creó un único puerto estandarizado para conectar un universo de periféricos, MCP proporciona un conector universal para que los agentes de IA se conecten de forma segura y fluida a cualquier herramienta o fuente de datos.” (C, 2025).

Esto será muy factible ya que en este proyecto se implementará un prototipo operativo de este servidor, donde se validará utilizando recursos prototipados de las empresas de la salud como APIs, sistemas externos, base de datos. El valor de esta implementación prototipada es demostrar una infraestructura diseñado para la IA esto asegurara la escalabilidad y la integración segura de futuros agentes de IA.

### **1.3 Objetivos**

#### **1.3.1 Objetivo General**

Implementar un Servidor para la orquestación en tiempo real de Agentes de IA Asistentes Médicos.

#### **1.3.2 Objetivos Específicos**

- Comparar las arquitecturas para la orquestación en tiempo real de Agentes de IA para asistentes médicos.
- Implementar un prototipo para la orquestación en tiempo real para la orquestación de agentes inteligentes Asistente médico.
- Evaluar la arquitectura implementada que cumpla con las necesidades del Asistente Medico (Agente de IA).

### **1.4 Alcance**

Este proyecto se centra únicamente en el desarrollo e implementación de un prototipo funcional de Servidor escalable, estandarizado, eficaz y sus mecanismos de conectividad en el cliente (Agentes de IA) con el servidor y este con los sistemas híbridos orientados al sector de la salud.

- Alcance Incluido:
  - Diseño Arquitectónico del Servidor
  - Implementación del Servidor Prototipo
  - Implementación del Protocolo de Conexión
  - Conectividad Híbrida Mínima
  - Evaluación de Rendimiento:

- Alcance Excluido
  - El desarrollo del cliente será un prototipo otorgado por la empresa que será un asistente médico.
  - Desarrollo de Arquitecturas Alternativas
  - Desarrollo de Sistemas Backend
  - Implementación en producción ya que se trata de un prototipo.

## **1.5 Contexto de la Empresa**

El crecimiento en el uso de agentes de inteligencia artificial en el ámbito de la salud ha evidenciado la necesidad de contar con una arquitectura intermedia capaz de gestionar de forma controlada la interacción entre dichos agentes y los sistemas clínicos. Si bien los agentes de IA poseen capacidades avanzadas de razonamiento y generación de respuestas, su autonomía debe ser limitada cuando se trata de ejecutar acciones sensibles, como la emisión de recetas médicas o el acceso a historiales clínicos. En este contexto, se requiere un servidor de orquestación seguro que actúe como un punto de control, permitiendo explotar la inteligencia del agente sin comprometer la seguridad, la trazabilidad ni la integridad de la información médica.

Asimismo, la naturaleza dinámica de las consultas médicas y la posibilidad de múltiples interacciones concurrentes demandan una arquitectura rápida, escalable y estandarizada, capaz de operar en tiempo real. Un servidor de orquestación permite centralizar las decisiones de ejecución, desacoplar la lógica del agente de los sistemas clínicos y facilitar la integración con distintos recursos del entorno de salud. De esta manera, se garantiza que el agente de IA pueda asistir al profesional médico de forma eficiente, mientras que las acciones críticas permanecen bajo un marco de control, validación y auditoría, alineado con las necesidades actuales del sector sanitario.

## **2. Capítulo 2: Marco Teórico y Conceptual**

### **2.1 Antecedentes**

Una vez definidos los objetivos generales y específicos se empezaron a buscar proyectos o títulos de grados similares que compartan nuestro objetivo. Precisamente no se esperaba encontrar varias investigaciones profundas sobre este tema debido a que es un protocolo nuevo y sorprendió encontrar una titulación de grado cuyo objetivo es automatizar

procesos de agentes Inteligentes con arquitectura RAG (Retrieval Augmented Generation) pero que impulsa a usar un servidor más eficiente y centralizado.

La tendencia actual apunta a arquitecturas agentic, donde múltiples agentes especializados colaboran de forma coordinada para alcanzar objetivos complejos. Este paradigma se apoya en protocolos como el Model Context Protocol (MCP), que busca estandarizar la interoperabilidad entre agentes y aplicaciones externas. En este contexto, las plataformas de workflows (n8n, Zapier, Make) pueden actuar como orquestadores que combinan lo mejor de ambos mundos: la estructura lógica de un flujo predefinido y la adaptabilidad de agentes basados en LLMs. Esto abre la puerta a sistemas híbridos capaces de integrar razonamiento dinámico con procesos repetibles y seguros. (Valderrama, 2025)

Analizando dicho trabajo (Valderrama, 2025), en su trabajo de grado propone Diseñar e implementar un sistema multiagente que automatice la gestión de tareas personales y profesionales, integrando modelos de lenguaje de gran tamaño y agentes especializados lo cual no lo está haciendo enfocado a un ambiente empresarial y cuyo flujo de arquitectura no usara MCP.

“RAG es una técnica extra dentro de los marcos de trabajo de la inteligencia artificial, cuyo objetivo es recuperar hechos de una base de conocimientos externa para garantizar que los modelos de lenguaje devuelven información precisa y actualizada. Un sistema RAG típico (ver imagen) incluye un LLM, una base de datos vectorial (para almacenar convenientemente los datos externos) y una serie de comandos o preguntas. Es decir, de forma simplificada, cuándo hacemos una pregunta en lenguaje natural a un asistente cómo ChatGPT” (Alejandro, 2024)

Existen varios artículos que hablan sobre el futuro y uso de MCP, garantizando la seguridad y un estándar donde los Agentes Inteligentes serán orquestados para obtener la información de diferentes recursos dentro y fuera de su organización. Al respecto, (González Guerra, 2025) en su artículo para Hiberus, afirma: "La inteligencia artificial está evolucionando rápidamente, y con ello surgen nuevas formas de mejorar su eficiencia y conectividad con datos en tiempo real. Uno de los desarrollos más recientes es el Model Context Protocol (MCP), un estándar abierto que permite a los modelos de IA acceder

directamente a archivos, APIs y herramientas sin necesidad de procesos intermedios como *embeddings* o búsquedas vectoriales".

## **2.2 Marco teórico: Fundamentos y comparativa de Arquitecturas**

Es crucial entender la importancia de la arquitectura que se va a implementar para así poder crear un Servidor que pueda orquestar de forma eficiente y rápida recuperando datos vivos de cualquier tipo de fuente de información y no solo de datos no estructurados el cual limita el funcionamiento de nuestro agente de IA (Asistente Medico).

### **2.2.1 Fundamentos de la Inteligencia Artificial Agentiva**

La IA agentiva se basa en técnicas de IA generativa para poder cumplir objetivos de manera autónoma con una intervención humana muy baja para cumplir los objetivos. Es decir que esta IA agentiva se diferencia de la IA generativa ya que no solo realiza imágenes, textos o códigos, sino que puede usarse para cumplir las tareas que se necesitan en un sistema ayudando de herramientas externas para poder obtener más información y así cumplir su autonomía.

Utilizar IA agentiva nos conlleva a resaltar varias ventajas que se deben mencionar como se habla anteriormente es autónoma, proactiva, especializada, adaptable e intuitiva, con estas ventajas esta IA es totalmente garantizada para su aplicación en sistemas grandes donde se requiere de cumplir varios objetivos ya que gracias a su autonomía esta IA generativa puede desarrollar esas tareas a tiempo y de manera eficiente.

“Algunos sistemas de IA agentiva pueden autorreforzarse y escalar comportamientos en una dirección no deseada. Este problema ocurre cuando la IA optimiza demasiado agresivamente para una métrica particular sin salvaguardas. Y como los sistemas agénticos suelen estar compuestos por múltiples agentes autónomos que trabajan juntos, existen oportunidades para el fracaso. Atascos de tráfico, cuellos de botella, conflictos de recursos: todos estos errores tienen el potencial de desencadenarse en cascada. Es importante que los modelos tengan objetivos claramente definidos que puedan medirse, con bucles de feedback establecidos para que los modelos puedan mover cada vez más cerca de la organización con el tiempo.” (Stryker, 2024)

### **2.2.1.1 Definición del Agente de IA Asistente Médico (Planificación, Razonamiento y Acción)**

El objetivo de este Agente Medico es guiar al doctor para tomar una decisión clínica completa ayudándole a diagnosticar al paciente y poder recetar medicina para dicho paciente es aquí crucial el servidor ya que se encargará de obtener la data viva operacional del HCE que es Historia Clínica Especializada de cada paciente, esto nos ayudara a que el agente razone de una manera eficiente para cumplir con el objetivo. Este acceso a través del servidor habilita una acción transaccional mediante las herramientas expuestas por el servidor.

El agente es capaz de crear documentos administrativos y es crucial es uso del servidor de orquestación el cual garantiza al agente conectarse con las herramientas necesarias que el agente asistente medico puede invocar para armar dichos formularios, los cuales no solo arman la estructura de los formularios, sino que también validan que el documento o formulario a crear cumpla con los requisitos exigidos por la documentación sanitaria.

La responsabilidad del agente asistente medico es servir como una ayuda extra casi profesional para el doctor, es decir este agente no tomará la última decisión final, sino que sirve para que el doctor se guie y ejecute una decisión final eficiente y acertada, por lo cual el servidor de orquestación incluirá una trazabilidad de las acciones que aporto este agente asistente medico con el objetivo de garantizar una auditoria y control medico legal.

### **2.2.1.2. Requisitos Operacionales Críticos: Acción Transaccional y Datos Vivos en el Entorno Sanitario**

El requisito operativo crítico para el Agente Asistente Médico es la Acción Transaccional Segura, que implica la escritura y validación de documentos oficiales, como la generación de un auto llenado de formularios de alta. Para manejar esta información sensible, el Servidor debe exponer una Herramienta que no solo estructure la data, sino que también interactúe de forma segura con el sistema de gestión documental sanitario. Esta capa de orquestación es crucial, ya que maneja el flujo de trabajo de llenado y simula la validación electrónica, asegurando la integridad de la data y cumpliendo con las normativas de trazabilidad exigidas por la ley sanitaria.

En segundo lugar, la confianza del diagnóstico depende de la actualidad de los datos vivos. Para que el agente pueda tomar una decisión informada, necesita la información más reciente y sensible al tiempo, como los últimos signos vitales, alergias confirmadas hoy o resultados de exámenes pendientes. La consulta directa al sistema de la verdad HCE (Historia Clínica Electrónica) es obligatoria. El Recurso del servidor garantiza precisamente esto: elimina la dependencia de la indexación estática y obsoleta, asegurando que la consulta a la base de datos se realice al instante, lo cual es vital para evitar errores fatales en la prescripción.

Finalmente, el servidor de orquestación esta alineada a cumplir su requisito de baja latencia, lo cual implica que la interacción del agente Asistente Medico debe tener una conexión instantánea con los sistemas híbridos proporcionando datos vivos operacionales en tiempo real es por eso por lo que la velocidad es un factor de calidad y seguridad clínica garantizando una atención eficiente y rápida para sus pacientes o asistir al médico en su documentación sanitaria.

### **2.2.2. Análisis Teórico de las Arquitecturas de Conectividad Legacy**

Es importante definir cuál es la arquitectura que se debe implementar para cumplir con los objetivos del Agente Asistente Médico. El análisis preliminar muestra que las arquitecturas existentes son insuficientes para los requisitos clínicos de Acción Transaccional y Datos Vivos Operacionales. Por lo tanto, la selección de una arquitectura de conectividad estandarizada y segura se vuelve el paso más crítico del proyecto, pues garantiza la interoperabilidad y la trazabilidad necesarias para la toma de decisiones sanitarias.

Uno de los objetivos importantes que tiene el Agente de IA es la baja latencia y la estandarización para poder resolver sus problemas de forma rápida y así ser autónomo. Por lo tanto, estas arquitecturas tradicionales son limitadas para la transferencia de datos crudos, los cuales son cruciales para la toma de decisiones que necesite el Agente de IA frente a las peticiones del doctor. Según (Goodwin, 2023)“Mantener baja la latencia de una red es importante porque afecta directamente a la productividad, la colaboración, el rendimiento de las aplicaciones y la experiencia del usuario. Cuanto mayor es la latencia (y más lentos los tiempos de respuesta), más se resienten estos aspectos. La baja latencia es especialmente

crucial a medida que las empresas persiguen la transformación digital y dependen cada vez más de las aplicaciones y servicios basados en la nube dentro del Internet de las cosas.” Por ende, tener una buena latencia ayuda a que el servidor de orquestación responda a las peticiones del Agente de IA de manera eficiente y segura.

Por otro lado, al implementar un middleware de orquestación, se debe tener en cuenta que el Agente pueda ejecutar acciones transaccionales sin necesidad de conocer la complejidad interna que hoy en día se tiene con las arquitecturas actuales. Por lo cual una arquitectura solida debe permitir que las decisiones del servidor sean detalles que queden en segundo plano y así no afecta la lógica de negocio central. Finalmente, los cambios en los sistemas híbridos no deben afectar al ecosistema implementada en el servidor, esto asegura que el servidor tenga continuidad operativa en todo momento y integridad de los datos híbridos ante cualquier escenario posible.

## **2.3 Marco Conceptual**

### **2.3.1. Servidor MCP (Gateway de Orquestación Segura)**

Es el pilar fundamental de esta implementación el cual es un componente de middleware permitiendo la orquestación segura entre los agentes de IA y los servidores de backend llegando a la conclusión de que es un único Gateway de comunicación. Por otro lado, es el encargado de centralizar la auditoria y control de acceso lo cual es fundamental para garantizar la trazabilidad de cada acción o consulta realizada a los sistemas de backend como el sistema de Historias Clínicas Electrónicas, también tiene una función operacional muy importante el cual es el responsable de exponer las herramientas y recursos que el agente puede consumir.

“En términos más simples, un servidor MCP se puede considerar como un complemento o adaptador para IA. Al igual que un navegador web puede tener complementos para añadir funcionalidad, un modelo de IA puede usar servidores MCP para ampliar sus capacidades de forma segura, ya sea para consultar un documento, una base de datos o incluso controlar un navegador web. Lo que hace especiales a los servidores MCP es que todos utilizan el mismo protocolo estándar. Esto significa que una IA no necesita código personalizado para cada nueva herramienta o fuente de datos; en cambio, se comunica con cualquier servidor MCP mediante un lenguaje común. *Algunos incluso han*

*comparado a MCP con el "USB-C de la IA": un conector universal para modelos de IA y sistemas externos.” (Buzdugan S, 2025)*

### **2.3.2. Agente de IA (Cliente MCP)**

Es el cliente que utiliza el servidor para orquestar sus llamadas, es un sistema basado en LLM (Large Language Model) que tiene varias funciones críticas que son la planificación y el razonamiento. Por lo tanto, tiene una función importante en el proyecto el cual es el encargado de decidir que herramienta o recurso invocar basándose en su razonamiento clínico y así cumplir con las necesidades que tiene el médico y por último usa el protocolo JSON RPC para enviar sus peticiones al servidor MCP.

“Un cliente MCP es una aplicación (aplicación de IA, bot) que solicita de forma segura datos empresariales de un servidor MCP para informar a los LLM, acceder a sistemas y automatizar acciones.” (Ezra, 2025)

### **2.3.3. Primitivas del Servidor MCP**

Son las primitivas esenciales del protocolo MCP y es lo que expone el servidor para que el agente de IA consuma esas primitivas dependiendo de sus objetivos a realizar, por ende, tienen un alto impacto y actúan como adaptadores recibiendo las peticiones del Agente para proceder a ejecutar las acciones o consumir la información que requieren los Agente

Se debe destacar que estas primitivas ayudan al correcto funcionamiento del agente de IA logrando una eficiencia enorme al rendimiento del agente Asistente Medico, además estas primitivas permiten que el servidor MCP este centralizado y pueda ser escalable, es decir es adaptable para nuevos agentes de IA, por lo cual se puede reutilizar código y se genera un costo operativo barato en cuanto la integración de nuevos agentes.

### **2.3.4. Datos Vivos Operacionales y Clínicos**

Los datos vivos operacionales y clínicos se refieren a la información crítica que va cambiando constantemente y cuya actualidad de los datos es sensible al tiempo y es importante para la toma de decisiones del Agente de IA, como los signos vitales, alergias o resultados de exámenes pendientes. Esta data tiene una comparación significativa con los datos indexados los cuales son documentación estática y su acceso en tiempo real a través

de los Recurso MCP es indispensable para la confianza y seguridad en el diagnóstico y el soporte que brinda el agente de IA al médico.

La criticidad de estos datos reside en que tan rápidos son consultados ya que cada segunda cuenta y va a influir en la eficiencia del diagnóstico, lo cual garantiza un diagnóstico preciso para un correcto diagnóstico hacia al paciente. Por lo tanto, el Servidor MCP garantiza que sus recursos consulten al sistema Historia Clínica Electrónica en tiempo real, eliminando el riesgo de tomar una información estática indexada que perjudica a la toma de decisiones y eficiencia en el Agente de IA

### **2.3.5. Protocolo de Intercambio JSON RPC**

Es el estándar de comunicación subyacente que utilizar MCP para mandar peticiones desde el agente al servidor y del servidor a los sistemas híbridos como el HCE. Por lo tanto, es un protocolo ligero que utiliza JSON para la codificación de datos, actuando como la capa de transporte que asegura que las peticiones sean de baja latencia. Esta velocidad es fundamental para garantizar la eficiencia instantánea en el soporte a la decisión clínica.

“En pocas palabras, JSON-RPC es una sucesión de JSON y un protocolo globalmente reconocido para llamadas a procedimientos remotos. A nivel de desarrollo, JSON-RPC implementa diversas tareas de definición de estructuras de datos para aplicaciones. Es un protocolo relativamente nuevo y de alcance limitado.” (Beschokov, 2025)

“Los conjuntos de comandos, la flexibilidad y los escenarios de implementación son limitados. Sin embargo, para los desarrolladores, es la opción ideal cuando se busca un desarrollo rápido y sencillo. Estas limitaciones son una ventaja en casos de uso sencillos y obligan a los desarrolladores a reemplazar REST con JSON-RPC.” (Beschokov, 2025)

### **2.3.6. Conectividad Híbrida**

Se refiere a la capacidad que tiene el Servidor MCP para interactuar simultáneamente con sistemas de información diversos y así obtener todo de tipo de datos que sea importante para la toma de decisiones o acción que debe tomar el agente de IA. Esto incluye tanto la conexión a sistemas locales como el HCE, como a servicios remotes o basados en la nube.

El servidor MCP unifica toda esta complejidad bajo su estándar de seguridad de orquestación.

El resultado de tener una conectividad híbrida es que garantiza que nuestra infraestructura tecnológica sea cohesionada que mantiene un estándar único de seguridad, autenticación y trazabilidad para todas las interacciones, sin importar la ubicación del sistema de destino y también ayuda a que las operaciones críticas que dependen de la baja latencia y la seguridad local se mantengan intactas, mientras se aprovechan las funcionalidades escalables y geográficamente distribuidas de la nube.

## 2.4. Fundamentación Técnica y Metodológica.

### **2.3.7. Middleware**

“Middleware es software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él. Básicamente, funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas. A veces, se le denomina “plumbing” (tuberías), porque conecta dos aplicaciones para que se puedan pasar fácilmente datos y bases de datos por una “canalización”. El uso de middleware permite a los usuarios hacer solicitudes como el envío de formularios en un explorador web o permitir que un servidor web devuelva páginas web dinámicas en función del perfil de un usuario.” (Microsoft Azure, n.d.)

### **2.3.8. FASTAPI**

“FastAPI es un framework moderno de alto rendimiento para Python 3.7+ que destaca por su capacidad para acelerar el desarrollo de APIs entre dos y tres veces más rápido que otros métodos tradicionales. Al estar construido sobre Starlette y utilizar sugerencias de tipos de Python, ofrece una robustez lista para producción, equiparable en velocidad a lenguajes como NodeJS o Go. Su ecosistema facilita enormemente la labor del desarrollador mediante la validación automática de datos, una gestión de errores eficiente y la generación de documentación interactiva basada totalmente en los estándares OpenAPI y JSON Schema, lo que garantiza una integración sencilla y profesional.” (Ali, 2024)

### **2.3.9. SQLAlchemy**

“SQLAlchemy es una biblioteca de mapeo objeto-relacional (ORM) para Python que proporciona un conjunto completo de herramientas para interactuar con bases de datos relacionales de manera eficiente y escalable. Su propósito fundamental es permitir que los desarrolladores operen bases de datos utilizando objetos y lógica de Python en lugar de escribir consultas SQL crudas, facilitando la creación de aplicaciones que son independientes del motor de base de datos utilizado (como PostgreSQL, MySQL o SQLite). Al separar la estructura de los datos de su almacenamiento físico, SQLAlchemy permite gestionar esquemas complejos y relaciones entre tablas mediante un diseño de "Data Mapper", lo cual es ideal para proyectos que requieren un alto nivel de personalización y control sobre el rendimiento de las consultas.” (Bayer, 2025)

### **2.3.10. Endpoint de API**

“Un endpoint de API es una ubicación digital, generalmente representada mediante una URL, donde una interfaz de programación de aplicaciones recibe y gestiona las solicitudes realizadas por los clientes para acceder a recursos específicos alojados en un servidor. Estos endpoints son componentes fundamentales de una API, ya que actúan como puntos de entrada que permiten la comunicación entre distintas aplicaciones de software, funcionando como un intermediario que facilita el intercambio de datos, funcionalidades y servicios bajo un conjunto definido de reglas y protocolos.” (Nosowitz & Goodwin, 2024)

En términos generales, los endpoints se pueden utilizar en como los componentes de acción y consulta que requiere nuestro Agente de IA, ya que permiten interactuar con sistemas externos permitiendo extraer información actualiza, su uso es crucial sin importar el tipo de arquitectura que se use para la orquestación segura del servidor.

## **2.4 Fundamentación técnica y metodológica**

### **2.4.1. Selección de la Metodología de Desarrollo Técnica**

La elección de una metodología robusta es crucial para la implementación de este prototipo de middleware, al ser una implementación individual y el tiempo es limitado se

debe escoger una metodología ágil para poder cumplir con estas condiciones. Por lo cual se escogerá una metodología que este centrado en la disciplina técnica y la calidad del código.

La ventaja de implementar este prototipo es mediante una metodología ágil es importante ya que se sigue un orden robusto y eficiente sobre la gestión de procesos que permite a nuestro desarrollo librarse de errores al ser colocado en producción. Esto garantiza la calidad del código desde el inicio a través de prácticas de ingeniería verificables (como TDD y Refactorización), lo que resulta en un servidor confiable, mantenible y funcionalmente eficiente, capaz de manejar con seguridad los datos críticos del sector salud.

La metodología scrum no sería muy factible para el desarrollo de este prototipo debido a que es creado para realizar entre 2 o mas personas por lo tanto queda descartado al ser una metodología ágil ineficiente para un solo desarrollador limitando los beneficios que tiene esta metodología cuyo enfoque es trabajar en equipo para cumplir con los objetivos que requiere el sistema o servidor a implementar.

“Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.” (Xavier, 2008)

La metodología ágil Xtreme Programming (XP) sirve como un conjunto de prácticas diseñadas para generar código robusto, adaptable y ágil, además esta metodología tiene pilares fundamentales que son la simplicidad en el diseño, tener una retroalimentación continua y el coraje para realizar cambios profundos en el desarrollo. Esto se logra a través de principios técnicos como el desarrollo por pruebas y la refactorización continua, que mantiene el código limpio y mantenible. Estos hábitos ayudaran a que la implementación del servidor sea seguro ya que al desarrollar la complejidad no va a comprometer la seguridad y por eso será la metodología ágil escogida para el desarrollo de este prototipo.

#### **2.4.1.1. Justificación de eXtreme Programming (XP) para Desarrollo Individual**

Se selecciona como el marco metodológico principal porque es la metodología ágil más centrada en la disciplina técnica rigurosa y la alta calidad del código que cumplen con los objetivos del desarrollo de un middleware que tendrá información sensible sobre el área de la salud. Debido a que el desarrollo será individual esta metodología es la indicada ya que transforma los requisitos de seguridad y contabilidad en hábitos de codificación verificable, eliminando la dependencia en las prácticas de gestión de equipo asegurando que el servidor sea robusto, verificable y limpio desde el comienzo de su desarrollo, el cual garantiza que el diseño arquitectónico del Gateway no comprometa su misión.

El entorno de desarrollo individual utilizando XP nos ayuda a optimizar tiempo, ya que, con metodologías tradicionales, donde la retroalimentación y la corrección de errores son procesos bastante largos, en esta metodología ágil la retroalimentación es inmediata. Esto permite aplicar el principio de diseño simple constantemente. Esta agilidad técnica es fundamental para el desarrollo de prototipos, permitiendo una rápida adaptación a los requerimientos del protocolo del servidor.

En conclusión, la elección de XP se justifica ya que se alinea al rigor técnico con la eficiencia atemporal del proyecto de tesis. Por lo tanto, XP es la garantía metodológica de que la solución implementada no solo será funcional, sino que cumplirá con los estándares de calidad y confiabilidad que están exigidos para la decisión clínica.

#### **2.4.1.2. Principios de XP Aplicados al Desarrollo del Servidor (TDD y Refactorización)**

El desarrollo guiado por pruebas (TDD) es el principio técnico fundamental aplicado para el desarrollo del Servidor. Esta técnica trata de escribir primero las pruebas unitarias y de integración para validar la seguridad y la funcionalidad de cada herramienta y recurso antes de escribir el código que lo implementa. Esto garantiza una trazabilidad al código manteniendo un historial por cada código implementado lo cual asegura flexibilidad en la implementación del servidor, asegurando que el gateway de seguridad y las conexiones que tiene que realizar a sistemas de backend funcionen bajo las especificaciones de seguridad esperadas y protegiendo contra la regresión de fallos.

El principio Refactorización Continua se utilizará para mejorar el diseño interno del código del servidor de forma constante sin que existan alteraciones externas en su comportamiento. Esta disciplina es esencial para este prototipo, donde la arquitectura siempre deberá permanecer simple y flexible a medida que se integran nuevas capas de seguridad y lógica de protocolo. Este principio evita que el código este sin un orden o no siga un diseño adecuado y evita la acumulación de complejidad, lo cual es vital para tener un mantenimiento y auditoria eficiente en el futuro del servidor.

## **2.4.2. Stack Tecnológico y Herramientas de Implementación**

### **2.4.2.1. Lenguaje de Programación y Framewor**

Se escoge Python como el lenguaje de programación, justificado por su dominio en el ecosistema de inteligencia Artificial y su robustez en el manejo de las estructuras con JSON y protocolos web, que son perfectos para el protocolo a implementarse. Posteriormente en el backend se escogió FastApi sobre alternativa como Flask o Django. FastAPI se justifica por su alto rendimiento y su diseño el cual es asincrónico, el cual es importante para lograr una baja latencia requerida por el requisito de que los datos deben ser vivo operaciones en el entorno clínico. Esta eficiencia es vital para el protocolo JSON RCP utilizado por los servidores de hoy en día.

La elección De FastAPI se complementa con su integración nativa con Pydantic.Lo cual es indispensable para el proyecto de tesis. Ya que permite la definición rigurosa de los esquemas de las herramientas y recursos de una forma estandarizada. Este rigor en los esquemas de las herramientas y recursos cubre los principios de interoperabilidad, ayudando a que los agentes de IA siempre sepan exactamente que tipos de datos esperar y enviar y así fortalece la calidad en la comunicación sobre soluciones más flexibles, pero menos estructuradas.

### **2.4.2.2. Entorno de Infraestructura (Proxmox, Debian 13 y Docker)**

La infraestructura esta sustentada en una pila de código abierto ya que prioriza la eficiencia y la estabilidad, por esa razón Proxmox Virtual Enviroment es seleccionado como hipervisor sobre soluciones comerciales, debido a que proporciona un control robusto y herramientas sencillas para el aislamiento controlado del entorno. Por otro lado, se

selecciona Debian 13 sobre otras distribuciones Linux como Ubuntu, por su estabilidad superior y ligereza, optimizando los recursos para el servidor de alto rendimiento.

El servidor se implementará utilizando Docker y Docker Compose para empaquetar la aplicación y sus dependencias, sobre una instalación directa del sistema operativo. Esta elección asegura la portabilidad y consistencia del prototipo, lo cual es un requisito clave para el entorno de prueba en cualquier fase del proyecto. El uso de contenedores aísla el servidor de otras aplicaciones, convirtiendo este desarrollo de middleware más independiente y así mantener limpio y minimizando el riesgo de conflictos de dependencias.

#### **2.4.2.3. Herramientas para Simulación de Datos Vivos y Conectores**

La fuente de los datos HCE sería a una base de PostgreSQL con una entidad relación real, pero con la información prototipada debido a temas de seguridad y lo cual es una base de datos muy eficiente debido a su robustez y soporte nativo a JSON sobre otras bases de datos como MySQL o SQLite. La capa de conexión entre el Servidor y PostgreSQL se implementará con el ORM SQLAlchemy para simplificar el mapeo de los recursos a consultas de Datos vivos el cual acelera el desarrollo.

Las Herramientas que invocan sistemas externos como ERPs se simularán con librerías HTTP asíncronas de Python, como httpx. Esta decisión técnica asegura que las llamadas de acción realizadas por el Servidor sean no bloqueantes, manteniendo la baja latencia que es crítica para el rendimiento del gateway. La combinación de estas herramientas provee un entorno realista para demostrar la capacidad del servidor de interactuar con una Conectividad Híbrida diversa y exigente.

### **3. Capítulo 3: Metodología científica y técnica**

#### **3.1 Enfoque y Diseño de la Investigación**

##### **3.1.1. Tipo de Investigación**

La siguiente investigación es de tipo aplicada, debido a que su objetivo general es resolver un problema práctico y actual que existe en el ámbito de la salud el cual es la orquestación segura y eficiente de los agentes de inteligencia artificial sobre los sistemas de backend como la Historia Clínica Electrónica. La diferencia crucial con la investigación

técnica es que esta implementación busca transformar el conocimiento de la interoperabilidad semántica y la seguridad en producto funcional que es capaz de operar con una latencia baja y alta integridad de los datos.

“La interoperabilidad semántica constituye la capacidad de diversos sistemas para intercambiar información y procesar su significado intrínseco de manera precisa, incluso cuando poseen estructuras de datos heterogéneas. Este nivel de integración se logra mediante el empleo de vocabularios compartidos y terminologías comunes que permiten a los sistemas interpretar los datos bajo un marco conceptual único, independientemente de su fuente o formato original. El proceso se apoya fundamentalmente en el uso de metadatos, los cuales funcionan como guías terminológicas para el sistema receptor durante la transferencia de información. En el contexto sanitario, esta capacidad es crítica, ya que permite unificar distintas denominaciones clínicas bajo códigos estandarizados, garantizando que todos los actores de la red de salud interpreten una condición médica de forma unívoca y alineada, eliminando así la ambigüedad en la toma de decisiones clínicas.” (Greg & Matthew, n.d.)

Adicionalmente, el estudio posee un carácter tecnológico, que está fundamentado en la creación de un prototipo que cumpla con las necesidades del Agente Asistente Médico. Este enfoque se centra en el desarrollo de un middleware el cual actúa como un puente estandarizado transaccional que orquesta los agentes de inteligencia artificial con los sistemas híbridos, el cual está integrado con tecnología moderna como Python, FastApi y Docker. El objetivo tecnológico es validar la viabilidad técnica de una arquitectura que permita a los agentes de inteligencia artificial relacionarse con los datos clínicos sin que existan inseguridades que puedan comprometer al sistema anfitrión.

Por último, para validar la fortaleza de los resultados, el desarrollo de este middleware se mantiene en una estrategia de datos híbridos. Esta estrategia permite que la investigación evalúe la robustez del servidor frente a registros obtenidos de la base de datos PostgreSQL que almacena la Historia Clínica Electrónica (HCE) pero están prototipados por seguridad e integridad de los datos y también prototipos y simulación que están diseñadas para probar estos escenarios extremos. De este modo la investigación aplicada logra que el producto

tecnológico pueda desarrollarse ante la complejidad de la data real y los problemas diarios que se tiene como la alta latencia al consumir estos datos.

### 3.1.2 Diseño Experimental

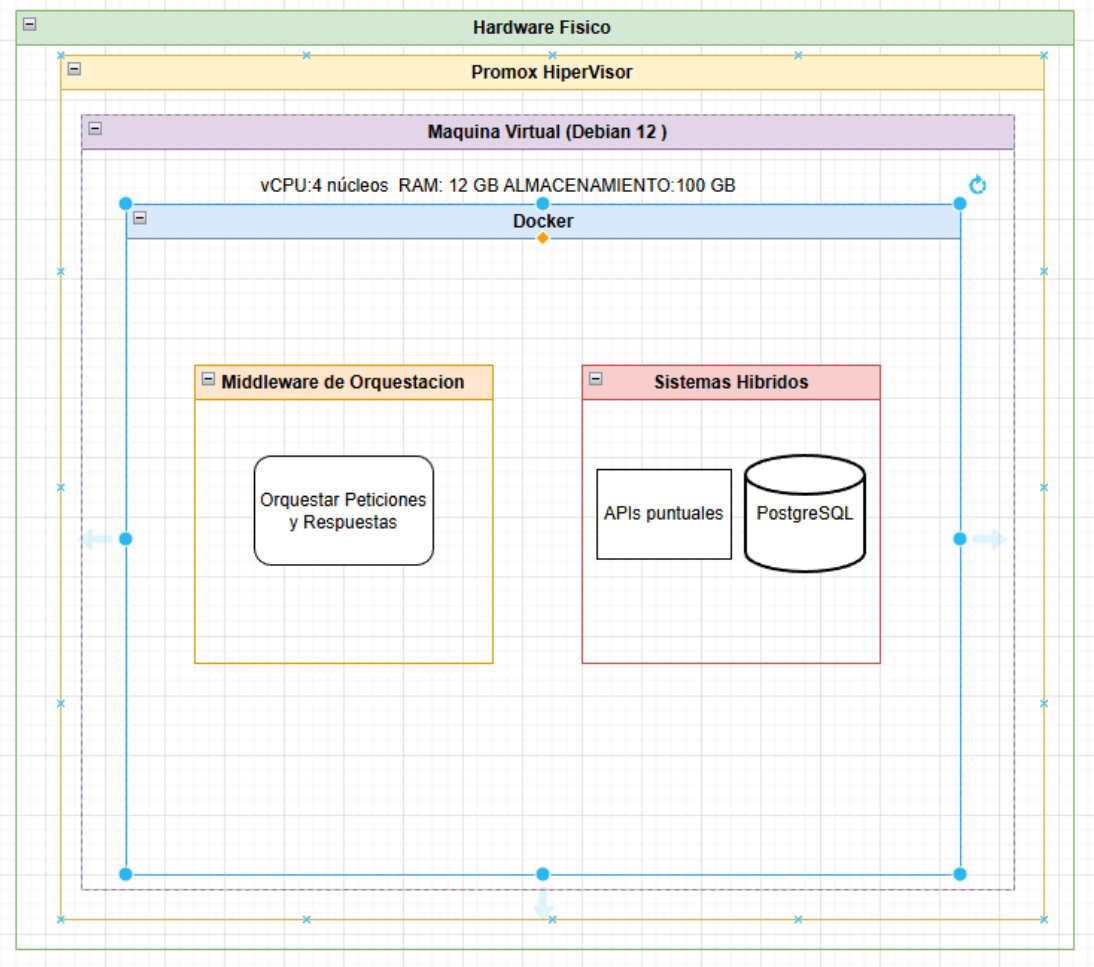
El diseño experimental se fundamente en un entorno de laboratorio virtual robusto, que esta implementado sobre el hipervisor Promox. Este entorno permite el despliegue de una máquina virtual dedicada con Debian 13, cuya configuración es 12 GB en RAM, 100 GB de almacenamiento y 4 núcleos de CPU para garantizar la repetibilidad de los experimentos bajo condiciones controladas. El usar este hipervisor crea un aislamiento absoluto de los componentes y la ejecución de estados de las pruebas consistentes, esto asegura que la latencia y rendimiento no se vean afectadas por factores externos a la infraestructura.

En el ecosistema virtual, la validación crucial se realiza mediante Docker cuya función es albergar el Servidor como los sistemas híbridos, bases de datos que contiene la HCE en PostgreSQL. Por lo cual, la validación importante es medir el procesamiento de las peticiones que realiza el agente de inteligencia artificial. Los datos híbridos se implementarán dentro del Docker de base de datos y sistemas híbridos que se implementaran, lo que permite entender como el sistema gestiona en simultaneo el uso de las herramientas que se encargan de las acciones transaccionales y la recuperación de información clínica real y critica sobre prototipos dinámicos.

Finalmente, el ejercicio experimental obliga al prototipo a someterse a flujos de trabajos clínicos predefinidos como los registros de signos vitales. Se enfatiza que cada iteración se evalúa bajo métricas cuantitativas del éxito transaccional y la integridad de los datos, utilizando la metodología ágil XP lo que nos permite crear ajustes incrementales. Por lo cual, el diseño mencionado provee un marco científico riguroso el cual permite evidenciar que el Servidor es un prototipo capaz de trabajar con baja latencia y alta precisión en un ambiente de salud simulado.

### Figura 1

*Diagrama del entorno del laboratorio virtual*



*Nota: Diagrama del entorno del laboratorio virtual. Autor: Ronald Rodriguez, 2025.*

## 3.2 Fases del Ciclo Desarrollo enfocado en XP

### 3.2.1. Fase de Planificación: identificación de las Historias de Usuario

Para medir el alcance funcional del Servidor se debe hacer mediante Historias de usuarios el cual se adapta a la fase de planificación de la metodología XP. Estas historias reemplazan las documentaciones sencillas, enfocadas en las capacidades que tiene middleware para ofrecer al Agente de IA como la optimización de procesos o la autonomía del Agente como tal. Las historias se centraron en resolver las necesidades que provocaba la arquitectura actual como que el Agente debe ser más autónomo y mejorar su eficiencia consumiendo datos vivos operacionales. Esta simplicidad metodológica permite priorizar enfoques para el desarrollador el cual es construir primero los componentes de mayor riesgo y valor clínico dándole un valor crítico en la implementación de la capa de seguridad y la conectividad a los datos híbridos.

La planificación se define por la velocidad de respuesta del Servidor. Por lo cual, cada historia de usuario estará lleva conectada una métrica que será evaluada constantemente como la baja latencia al realizar las operaciones requeridas por el Agente Asistente Medico, asegurando que la eficiencia sea un objetivo intrínseco al diseño funcional y no quede como una opción secundaria. La implementación del servidor se rige a partir de las historias del usuario como la lectura hacia el sistema híbrido o prescribir recetas o papeleo clínico necesario para procesar pacientes. Por ejemplo, para resolver la historia de leer el sistema híbrido es crucial el uso eficiente de SQLAlchemy para mapear los recursos de la base de datos PostgreSQL.

Finalmente, la naturaleza de los datos híbridos influye directamente en la planificación debido a que estos datos ayudaran a la eficiencia del prototipo del Servidor ante casos clínicos reales. Esto garantiza que la planificación no solo cubra el flujo de trabajo normal, sino también la robustez y el rendimiento de baja latencia que ofrece el servidor frente a escenarios complejos de validación clínica.

### **3.2.2 Fase de Diseño: Definición de Interfaces y Esquemas de Interoperabilidad**

Se define en el diseño arquitectónico de la interfaz de comunicación que tiene el Agente Asistente Medico con los datos vivos operacionales lo cual es una interacción bidireccional, cuyo objetivo es dividir las necesidades que tiene el Agente Asistente Medico en cuanto las capacidades de consulta y las capacidades de acción. El diseño de estos esquemas se rige por tipado estático lo cual garantiza que las peticiones y respuestas en formato JSON sean validas, predecibles y estructuralmente integra.

“La tipificación estática es un sistema donde las variables se vinculan a un tipo de datos definido durante la compilación y permanecen inalterables durante toda la ejecución, lo cual garantiza una robusta seguridad de tipos al reducir drásticamente los errores por desajustes de datos. Al permitir que el compilador identifique incoherencias en una etapa temprana del desarrollo, se facilita la creación de software más confiable y predecible; además, esta estructura proporciona ventajas significativas en el rendimiento, ya que el conocimiento previo de los tipos permite optimizaciones de código que aceleran la ejecución del programa.” (BairesDev Editorial Team, 2025)

Las interfaces de acción requieren de un diseño eficiente ya que son un pilar fundamental para la seguridad operativa y los entornos de la salud, por lo cual cada transacción operacional se diseña bajo un esquema de permisos granulares, donde el acceso a las herramientas que el servidor expone al agente asistente médico dependen estrictamente del contexto clínico y las credenciales del operador. Este enfoque garantiza que el Agente Asistente Médico acceda a lo necesario para así tener integridad en los datos, los cuales son muy sensibles. De este modo, el agente solo podrá ejecutar acciones donde se le otorgue el permiso generando una seguridad ante ejecuciones erróneas o accesos no permitidos a información crítica y sensible.

En conclusión, este diseño es vital para el componente Adaptador, ya que es el encargado de orquestar que necesidad tiene el Agente Asistente Médico ya sea para acciones o consultas. Al centralizar la lógica de acceso en el adaptador asíncrono, se establece la base técnica para tener una baja latencia en las respuestas, permitiendo que el servidor gestione de manera eficiente la conectividad híbrida entre los datos sintéticos y los prototipos de simulación clínica.

“La programación asíncrona permite que las tareas se ejecuten de forma independiente, lo que permite procesar múltiples tareas al mismo tiempo sin bloquear el hilo de ejecución principal. A diferencia de las operaciones síncronas, las tareas asíncronas se ejecutan de forma independiente, lo que permite que otras operaciones continúen sin bloquearse. Las tareas que no dependen de otras pueden delegarse y ejecutarse simultáneamente con la operación principal, y luego informar el resultado al finalizar. Es como pedir un café a través de una aplicación móvil mientras se camina hacia la tienda: no hay que detenerse para hacer el pedido.” (Almeida, 2025)

### **3.2.3 Fase de Codificación y Pruebas (TDD)**

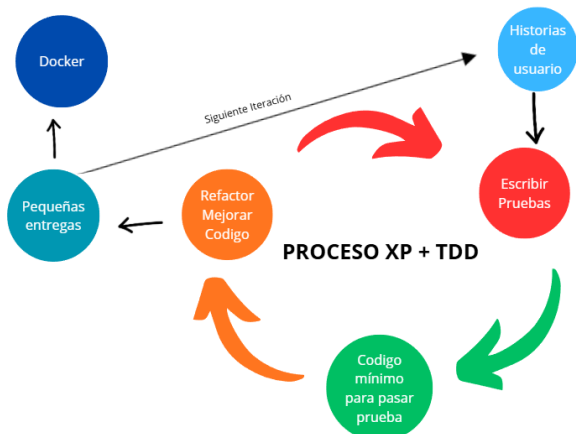
Para asegurar la robustez del servidor se debe seguir el desarrollo guiado por pruebas (TDD). Este enfoque sigue una serie de pasos para cumplir los objetivos, donde primero se requiere escribir la prueba unitaria que falla, posteriormente el código mínimo necesario para que pase la prueba y finalmente se refactoriza el código que se implementará al crear el Servidor de orquestación para el Asistente Médico. En este contexto aplicamos esta técnica que brinda la metodología ágil XP en la lógica de validación de esquemas y a los adaptadores

de datos, asegurando que nuestro servidor sea dinámico al presentar cambios a futuro en la comunicación o en los sistemas híbridos como la base de datos PostgreSQL.

Este proceso TDD se complementa con la integración continua que nos proporciona Docker, donde cada vez que un recurso o herramienta pase las pruebas unitarias y esta codificada se integrara de inmediato en el contenedor Docker y se despliega en el entorno de nuestro hipervisor Promox para pruebas de integración. Por lo tanto, esta integración temprana y continua es crucial para el flujo de trabajo de XP, permitiendo analizar conflictos en librerías o dependencias entre FastAPI o SQLAlchemy en las fases iniciales del desarrollo de este servidor.

**Figura 2**

*Diagrama de flujo del prototipo Servidor MCP*



*Nota:* Diagrama de flujo del prototipo Servidor MCP. Autor: Ronald Rodriguez, 2025.

### 3.3. Arquitectura del Sistema e Implementación Técnica

#### 3.3.1 Configuración del Entorno Virtual

El Servidor fue implementado sobre un entorno de visualización Promox Virtual Enviroment (PVE) lo cual permite establecer un ambiente de pruebas reproducible y controlado, crucial para la metodología XP. Se creo una maquina virtual con el nombre ZPMCP, configurada con el sistema operativo Debian GNU/Linux 12.0 conocido como Bookworm. Se selecciono este sistema operativo debido a su código abierto y su flexibilidad al momento de hospedar como servidor, la estabilidad que tiene frente estos proyectos y su

naturaleza minimalista, lo cual reduce crear una especie de sobrecarga del hipervisor y permite dedicar mas tiempo a consumir la mayor cantidad de recursos de hardware a la ejecución eficiente de la aplicación.

“Linux es un sistema operativo (SO) de código abierto basado en Unix. Funciona en ordenadores, servidores, teléfonos móviles y una amplia gama de otros dispositivos. Linux fue desarrollado por Linus Torvalds para ser una alternativa similar a Unix a Unix, uno de los primeros sistemas operativos jamás creados. Depende en gran medida del kernel de Linux para su funcionalidad. En un sistema operativo, el kernel es un programa informático que permite a los usuarios controlar el hardware y el software del sistema. Además del núcleo, el sistema operativo Linux utiliza varios componentes, como bibliotecas del sistema y servicios, pero todos dependen del kernel para comunicarse y recibir comandos de los usuarios.” (Mesh & Ian, 2022)

Se asignaron recursos pensando en el objetivo crucial que tiene el servidor que es la baja latencia y concurrencia, incluyendo 4 CPUs virtuales, 12 GB de memoria RAM y un almacenamiento de 100 GB. Esta configuración en la maquina virtual fue crucial para simular un ambiente de producción adecuado, el cual era optimo para generar las pruebas de integración y operación simultanea de los contenedores del Servidor y Sistemas híbridos o base de datos como PostgreSQL. La instalación del sistema operativo fue instalada sin entornos gráficos, garantizando un menor consumo de recursos y poder enfocarse en el correcto uso del hardware para operaciones críticas del Servidor.

Esta configuración facilita el trabajo de la Integración Continua, lo que permite que nuevas funcionalidades agregadas a los contenedores sean probadas de inmediato en el Servidor, siempre y cuando estas funcionalidades cumplan con el ciclo de TDD. Promox, juega un papel importante debido a su replicabilidad y aislamiento ya que son pilares que aseguran las métricas de rendimiento y robustez adquiridas durante la experimentación de las funcionalidades permitiendo verificar si son consistentes y validas para el proyecto de titulación.

### **3.3.2. Orquestación de Servicios con Docker:**

La arquitectura del Servidor se implementó bajo un esquema de Dockers lo que garantiza la portabilidad, la reproducibilidad y la separación de responsabilidades entre los

componentes, lo cual refleja un ecosistema ligero y eficiente. Esta orquestación se basa de dos servicios principales interconectado, alojados en la máquina virtual de Promox:

- **Contenedor del Servidor:** Este contenedor aloja el servidor, es decir es el foco central del proyecto de titulación donde existen los componentes como los componentes de acción y consulta, adaptador cuyo rol es consumir y orquestar servicios, además aloja el Gateway FastAPI que se expone para ser consumido por el Agente Asistente Medico.
- **Contenedor de Simulación/Datos Híbridos:** Este contenedor representa la HCE real, sino la capa de datos que el Servidor consume. Su propósito es simular la base de datos PostgreSQL con datos prototipados, lo que nos permite realizar pruebas de lectura, acciones y rendimiento con la estructura de datos que se requiere.

Esta estructura de contenedores es muy eficiente ya que ayuda aislar las dependencias es decir no se cruzan librerías para crear el servidor y los prototipos para representar las acciones y consultas como la base de datos de PostgreSQL la cual es consumida por el agente asistente médico , esto quiere decir que no se desarrollo una base datos para los recursos que necesita nuestro Agente Medico, sin embargo solo se crearan las herramientas para simular las acciones transaccionales que debe cumple el agente de IA.

En definitiva, esta arquitectura de orquestación asegura que el servidor demuestre su capacidad para ejecutar lecturas sobre la base de datos y consumir datos de fuentes híbridas, integrando información real de la HCE con prototipos diseñados para entornos de producción. El sistema es plenamente apto para el uso de librerías asincrónicas como **httpx** al invocar sistemas externos, lo cual, aunque no se refleje en la implementación final de este proyecto de titulación, permite establecer un entorno realista y de alto rendimiento. Esto facilita la medición precisa de los tiempos de respuesta en el flujo de peticiones y respuestas procesadas por el middleware.

“HTTPX se posiciona como el sucesor moderno de librerías clásicas como Requests, destacándose por su capacidad nativa para ejecutar solicitudes tanto sincrónicas como asincrónicas (async/await) bajo una misma interfaz intuitiva. Esta versatilidad es fundamental para el desarrollo de aplicaciones actuales, ya que permite escalar interacciones

con APIs complejas mediante el uso de HTTP/2, gestión avanzada de tiempos de espera (timeouts) y un sistema robusto de autenticación. Al integrar HTTPX, los desarrolladores de Python obtienen una herramienta eficiente que no solo simplifica el consumo de servicios web, sino que también optimiza el rendimiento del sistema a través de la agrupación de conexiones y el manejo fluido de flujos de datos (*streaming*)” (Ulili, 2025)

### 3.3.3. Desarrollo del Middleware de Orquestación

El middleware se desarrolló bajo la librería FastAPI en Python, lo cual exprimimos su eficiencia en la programación asincrónica lo cual es importante para mantener la baja latencia y que se cumplan las peticiones en tiempo real. A continuación, nos podemos fijar como esta desarrollado el contenedor que albergara el Servidor. La lógica del enrutador es recibir y despachar las peticiones en formato JSON-RCP, utilizando Modelos\_Pydantic para validar la estructura del JSON estrictamente de entrada y salida. Esta validación permite que se filtren errores rápidamente y optimizar el tiempo y garantiza que el Agente Asistente Medico reciba respuestas estructuradas y predecibles.

“Pydantic se ha consolidado como la solución líder para la validación de datos en Python al transformar las sugerencias de tipo estándar en reglas de ejecución estrictas y automatizadas. Su principal ventaja radica en la eficiencia del código: en lugar de saturar la lógica con múltiples comprobaciones manuales como `isinstance()` o validaciones personalizadas repetitivas, permite definir la estructura de datos una única vez mediante una sintaxis limpia y nativa. A partir de esa definición, la biblioteca gestiona de forma transparente la coerción de tipos y la limpieza de datos, garantizando que la información entrante sea siempre válida y proporcionando mensajes de error detallados cuando algo falla, lo que acelera drásticamente el ciclo de desarrollo y la depuración.” (Tuychlev, 2025)

La capa de conexión con las fuentes de datos se implemento con el ORM SQLAlchemy dentro del adaptador. Esta técnica nos ayuda a optimizar el mapeo de los recursos a consultas de Datos Vivos, lo cual acelera el desarrollo y este es uno de los principios de la metodología XP y permitiendo al adaptador manejar tanto las consultas y acciones ORM a PostgreSQL. Esta función doble que contiene el adaptador es crucial para demostrar la velocidad y la explotación que hará el servidor al interactuar con una conectividad híbrida diversa y exigente de manera no bloqueante.

### **3.3.4. Estrategia de Conectividad a la Fuente de Datos Híbrida**

En esta parte se integra una base de datos con contenido real la cual es una base de PostgreSQL y por ende es un beneficio debido a su eficiencia con formatos JSON. Para cumplir con los objetivos del proyecto esta base de una Historia clínica electrónica (HCE) estará poblada con los mismos datos, pero mezclados para cumplir con la seguridad de los datos ya que solo será un prototipo de Servidor y que sirven para probar escenarios específicos que pueden pasar en el sector de la salud.

Esta parte sirve mucho para los recursos que necesita el Agente de IA es decir requerimientos del Agente Asistente Medico que solo serán de lectura como por ejemplo revisar la presión arterial, al usar SQLAlchemy permite mantener la lógica de negocio y la validación de esquemas de la aplicación de Python. lo que asila al Servidor de las complejidades directas que se tiene con el lenguaje SQL, lo que ayuda a cumplir uno de los principios de la metodología XP como la refactorización continua.

Aunque en la implementación actual no se consumen servicios externos, la arquitectura fue diseñada considerando la posibilidad de futuras integraciones. La adopción de un modelo asincrónico facilita la incorporación posterior de nuevas fuentes de información o servicios complementarios sin necesidad de modificar la estructura central del sistema, preservando así su escalabilidad y mantenibilidad utilizando utilizando librerías HTTP asincrónicas de Python, como el anteriormente mencionado httpx. La combinación de un backend SQL eficiente con la simulación asincrónica de las llamadas para las herramientas a Apis externas o la misma base de datos Postgresql demuestra la capacidad del Servidor para gestionar la conectividad híbrida de manera segura.

## **3.4. Pruebas y Protocolo de Validación**

### **3.4.1. Escenarios de Evaluación Clínica**

La validación de la arquitectura de orquestación se basó en tres pruebas críticas en escenarios clínicos claves lo cual representan el flujo esencial que se siguen en este entorno de la salud. Estos casos de uso fueron diseñados para probar la eficiencia de los componentes principales que son el Adaptador como al hacer una consulta de paciente ocupa los componentes de consulta y sobre datos híbridos como el registro de los signos vitales que

evalúa la acción de la escritura simulada, y la receta electrónica que es también hecha por el componente de acción que prueba la acción transaccional compleja que puede albergar el servidor.

Cada escenario se ejecuto bajo el principio de TDD que brinda la metodología XP, donde cada paso del caso de uso se mapeo con las pruebas unitarias y de integración, La definición de estos casos de uso es fundamental, debido a que obliga a que se valide la conectividad Híbrida en dos escenarios, la lectura de los datos desde la fuente de la Historia Clínica Electrónica (HCE) y la ejecución asincrónica de acciones a la misma base de datos.

El objetivo principal que tienen estos escenarios es demostrar que tan capaz es el servidor ante casos complejos o simples, lo cual demuestra una robustez y funcionalidad eficiente del Servidor. Al basar las pruebas en flujo clínicos reales, estamos garantizando que la validación no solo sea técnica, sino que cumpla con los objetivos y requisitos definidos para el servidor como Gateway de orquestación segura.

### **3.4.2. Métricas de Evaluación**

Para cuantificar el desempeño de la arquitectura de orquestación propuesta, se definieron métricas orientadas al rendimiento interno del servidor y a la correcta ejecución de las operaciones solicitadas por el Agente Asistente Médico. La métrica principal considerada es la latencia de respuesta, entendida como el tiempo total que transcurre desde que el agente envía una solicitud al endpoint FastAPI hasta que recibe la respuesta final, incluyendo el proceso de decisión, validación RBAC y ejecución de recursos o herramientas en el servidor MCP. Esta métrica permite evaluar la capacidad del servidor para operar en tiempo real y garantizar una experiencia fluida para el personal médico.

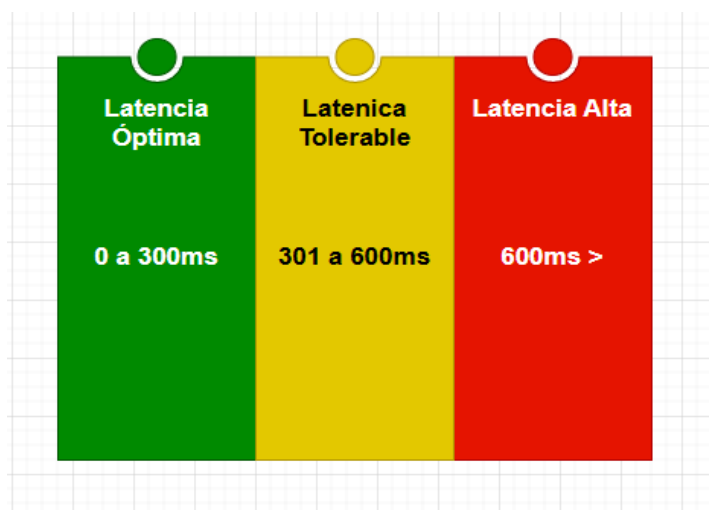
Por lo tanto, las métricas son obligatorias cuando se desea alcanzar un servidor de la calidad. Según (Quiroz, 2025)“Las métricas son medidas específicas de un aspecto del rendimiento del servicio, como la disponibilidad o la latencia. Los indicadores clave de rendimiento (KPI) están vinculados a los objetivos empresariales y se utilizan para juzgar el progreso de un equipo hacia esos objetivos. Los KPI no existen sin objetivos empresariales; son "indicadores" de progreso hacia un objetivo establecido.” Esta práctica complementa el correcto funcionamiento del servidor ante entornos clínicos,

De forma complementaria, se utiliza la tasa de éxito de las operaciones orquestadas, la cual mide el porcentaje de solicitudes que son procesadas correctamente sin errores de validación, ejecución o acceso a datos. Esta métrica se apoya en los registros de auditoría generados por el sistema y permite verificar que las acciones clínicas, como la consulta de historiales o la simulación de recetas, se ejecuten de manera íntegra y consistente. Un valor alto en esta métrica evidencia la estabilidad del servidor y la correcta coordinación entre el agente, la capa de orquestación y la base de datos clínica.

Finalmente, se considera el uso de recursos del servidor como métrica de apoyo para evaluar la capacidad del sistema en escenarios de concurrencia moderada. Este análisis se realiza observando el consumo de CPU, memoria y el número de solicitudes atendidas por segundo en la máquina virtual que ejecuta el servidor bajo Debian 13. Estos indicadores permiten determinar si la arquitectura propuesta es adecuada para entornos clínicos con infraestructura limitada, demostrando que el servidor de orquestación puede operar de forma eficiente sin requerir recursos excesivos.

### Figura 3

*Gráfico de Latencia del Servidor*



*Nota:* Gráfico de Latencia del Servidor. Autor: Ronald Rodriguez, 2025.

### **3.5. Consideraciones Éticas y de Seguridad**

#### **3.5.1. Manejo de Datos de Prueba**

El tratamiento de datos es crucial en el proyecto de un servidor de orquestación seguro ya que se cumple con la integridad de los datos, donde aplicamos una técnica muy importante que es la animación y desidentificación antes de cargar la información en el Docker de sistema híbrido. Este proceso asegura que ningún dato este asociado con el paciente físico. La seguridad no solo es encriptar la data, sino de garantizar que el acceso a datos de la salud este restringido exclusivamente a fines de investigación o atención directa. Por ende, el uso de datos híbridos protege la integridad y la privacidad de las personas mientras permite evaluar el servidor frente a escenarios de alta complejidad.

El diseño del middleware asegura que la información crítica que el Agente Asistente Medico solicita mediante sus peticiones no se guarde en memoria cache, ni en logs, sino que solo da acceso a dicha información en el tiempo que se necesite para completar la operación. Por lo tanto, se minimiza el riesgo de explotar estos datos críticos en una posible vulneración al Docker, asegurando que el middleware actúe como un conducto seguro y no como un repositorio de información privada, donde se establece un control riguroso y robusto sobre la persistencia de los datos de sesión.

En conclusión, el aplicar estas técnicas nos ayudan a tener un servidor seguro y eficiente que cumple con las normativas que se imponen a nivel mundial como la ISO 27001 que nos dice que para proteger la información se deben cumplir con tres factores la confidencialidad, integridad y la disponibilidad de los datos. Por lo cual, la disponibilidad queda totalmente respaldada por la infraestructura que ofrece Promox, que permite la recuperación rápida y un aislamiento de fallos mediante contenedores, mientras que la confidencialidad se logra con el control de accesos creando la capa de seguridad en el servidor y la integridad con la privacidad de la data formando la famosa triada.

#### **3.5.2. Control de Acceso**

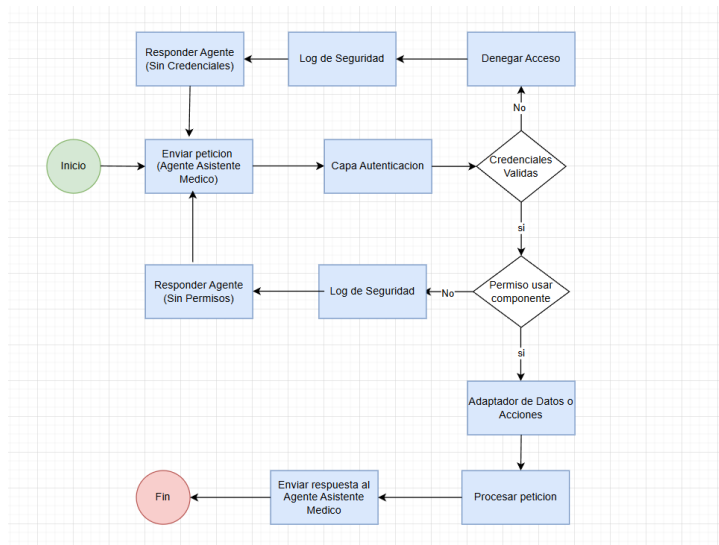
La seguridad del middleware se fundamentó con el módulo de Control de Acceso Basado en Roles y en la validación del contexto. Por lo tanto, cada petición del Agente Asistente Medico debe ser validado con autenticación y validación de permisos sobre los componentes de acción o consulta. Esta validación permite que la orquestación sea segura y

eficiente evitando ejecuciones no deseadas o vulnerabilidades que afecten a la integridad y disponibilidad de los datos y el servidor, permitiendo que el Agente Asistente Medico actúe de manera autónoma, pero bajo controles estrictos del administrador del servidor clínico.

Al seguir esta técnica permite una estricta auditoria permitiendo saber que está pasando con el servidor en todo momento. Según (Red Hat, 2024), “Este modelo concede o deniega el acceso a los usuarios según las funciones y las responsabilidades que tengan asignadas. Estas funciones determinan los recursos a los que pueden acceder los usuarios, de modo que los administradores del sistema puedan gestionar directamente los derechos de acceso conforme a ellas. El RBAC admite las jerarquías de funciones, por lo que aquellos usuarios que tienen una función de nivel superior también pueden heredar los permisos de acceso de otras inferiores, lo cual simplifica la gestión. Por ejemplo, un usuario con una función de "Personal" puede tener permisos limitados, como de solo lectura o de uso compartido de archivos, mientras que un "Gerente" dispondrá de todos los permisos de un usuario de "Personal" más derechos de acceso adicionales. El RBAC se rige por el principio del mínimo privilegio (PoLP), que constituye la base de la seguridad con confianza cero.” Por ende, no es común que existan ataques que puedan quitar la disponibilidad del servidor con los doctores, garantizando un servidor eficiente y activo en todo momento.

**Figura 4**

*Diagrama de flujo de la seguridad del Servidor*



*Nota:* Diagrama de flujo de la seguridad del Servidor. Autor: Ronald Rodriguez, 2025.

Como se puede observar en la **Figura 4** se siguen dos tipos de controles de acceso donde el primero validamos credenciales es decir si el doctor o el paciente que ejecuta la acción este en la base de datos HCE que es la que nos brinda los datos reales y va cambiando minuto a minuto por eso MCP es fundamental aquí ya que expone sus herramientas y recursos en todo momento que el Agente ejecute una pregunta y el otro validar los permisos aplicando RBAC que es uno de los principios más útil y usado cuando se tratan de sistemas complejos o mínimos. Por último, cada petición se agrega en la respectiva tabla de auditoria de cada base de datos recuerda que hay una del sistema y la otra es donde queremos explotarla con MCP.

## **4. Capítulo 4: Implementación del prototipo del Servidor**

### **4.1. Análisis Comparativo de Arquitecturas de Integración**

#### **4.1.1 Limitaciones de las Arquitecturas Legacy: APIs Ad-hoc y Rigidez Estructural**

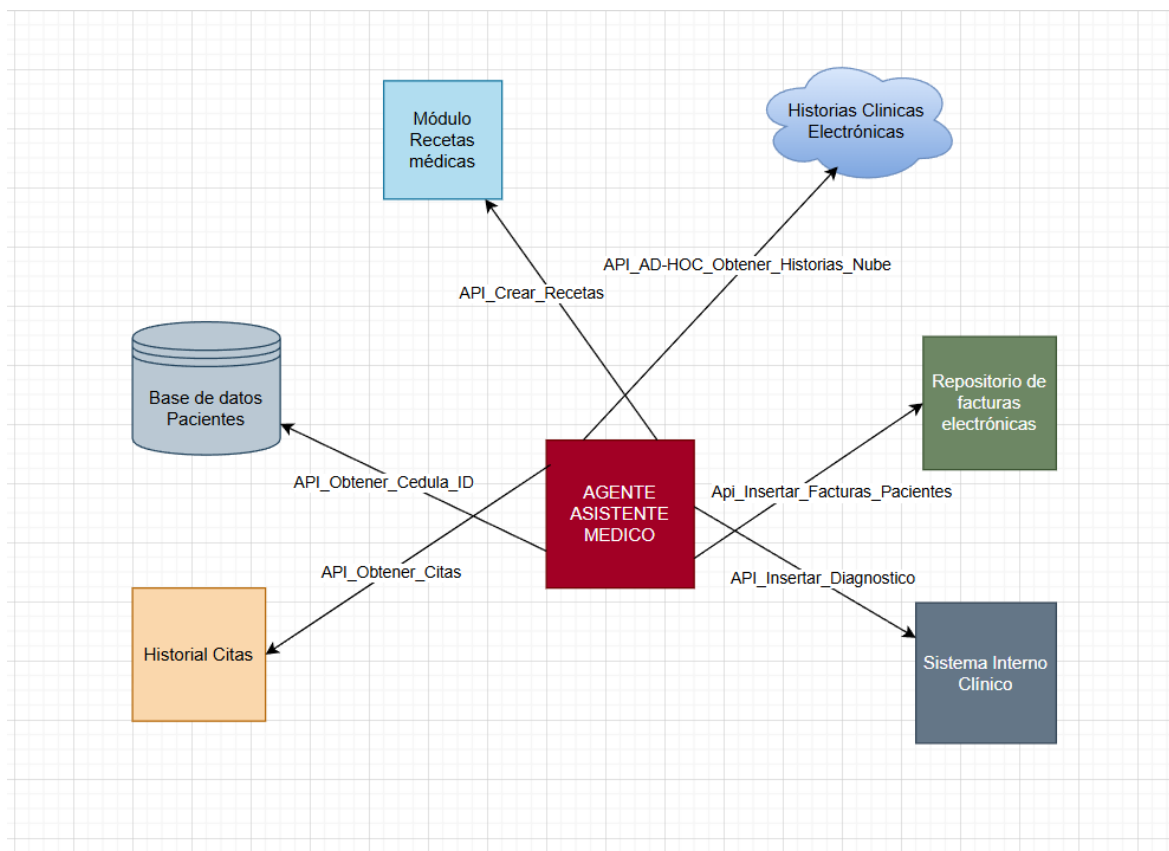
La dependencia de arquitecturas basadas en APIs Ad-hoc son muy limitadas para la eficiencia y autonomía del Agente de IA ya que es una solución no escalable y maneja un concepto de punto a punto son APIs a medida que van surgiendo, dependiendo de los requerimientos de Agente Asistente Medico. Según (Sommerville, 2011) “Cuanto más estrechamente acoplados estén los componentes de un sistema, más probable es que un cambio en uno de ellos provoque fallos en cadena en los demás, lo que incrementa exponencialmente los costos de mantenimiento y reduce la fiabilidad general”. Esta rigidez estructural que provoca esta arquitectura es pésima para el contexto del proyecto de titulación, debido a que no esta estandarizado lo que genera deficiencia en la orquestación del servidor.

Además, en estas soluciones Ad-hoc el desarrollador debe generar mapeos manuales lo cual pierde dinamismo y depender de endpoints estáticos estanca la autonomía del Agente Asistente Medico. Por ende, cada vez que el agente necesite de algún recurso nuevo, se debe desarrollar el endpoint, probarlo y desplegarlo, lo que ralentiza la iteración y aumenta la deuda técnica, por lo cual en un entorno clínico donde los esquemas de datos de la HCE son complejos y vastos el uso de esta solución basada en Ad-hoc queda obsoleta.

Finalmente, se debe comprender que los Agentes de IA, tienen un límite de memoria conocido como ventana de contexto. Según (Bergmann, 2024)“La ventana de contexto de un modelo de lenguaje grande (LLM) se refiere a la cantidad de texto, medida en tokens, que el modelo puede procesar y mantener presente en un momento dado, funcionando de manera similar a una memoria de trabajo. Una ventana de contexto más amplia permite al modelo comprender conversaciones más largas, analizar documentos o fragmentos de código extensos y generar respuestas más coherentes y precisas, reduciendo errores y alucinaciones. No obstante, cuando la información supera este límite, es necesario truncarla o resumirla para que el modelo continúe operando, y aunque ampliar la ventana mejora el rendimiento, también incrementa los costos computacionales y puede aumentar la exposición a ataques adversarios.” Por lo tanto, es crucial pensar en el costo operacional que con lleva la arquitectura selecciona y que tan eficiente, escalable y adaptable es para la explotación correcta del Agente de IA.

**Figura 5**

*Diagrama de Integración Punto a Punto*



*Nota:* Diagrama de Integración Punto a Punto. Autor: Ronald Rodriguez, 2025.

Como se ilustra en **Figura 5** esta arquitectura Ad-Hoc tiende a crecer de forma desordenada y pierde la escalabilidad como arquitectura, donde cada nueva funcionalidad que requiere el Agente se debe crear un puente de código único. Esta condición, implica la deficiencia ante entornos clínicos complejos, debido a que el Agente de IA debe ser reprogramado ante cualquier cambio estructural, lo que evidencia la rigidez estructural que motiva al cambio hacia una arquitectura centralizada.

#### **4.1.2 Desafíos del RAG (Retrieval-Augmented Generation) en Entornos Clínicos**

La arquitectura RAG es una de las más usadas al implementar soluciones en entornos clínicos. Según (Red Hat, 15) “La generación aumentada por recuperación (RAG) es una técnica que mejora la calidad y la relevancia de las respuestas de una aplicación de inteligencia artificial generativa. Funciona vinculando recursos externos al conocimiento obtenido por un modelo de lenguaje de gran tamaño (LLM) en su entrenamiento previo. Específicamente, estos recursos externos se segmentan, se indexan en una base de datos vectorial y se utilizan como material de referencia para que el LLM los extraiga y ofrezca respuestas más precisas.” El uso de esta arquitectura frente al objetivo de nuestro proyecto de titulación no es la mejor debido a que solo puede interactuar con datos estáticos. (Belcic & Stryker, 2025) nos dice “Los sistemas de RAG tradicionales son herramientas de recuperación de datos reactivas que encuentran información relevante en respuesta a consultas específicas. El sistema RAG no tiene capacidad para adaptarse a contextos cambiantes ni para acceder a otros datos. Los resultados óptimos a menudo requieren una extensa ingeniería rápida.” Por lo tanto, se debe priorizar las necesidades del Agente de IA para que pueda ser un asistente eficiente y autónomo que cumpla con las necesidades del usuario.

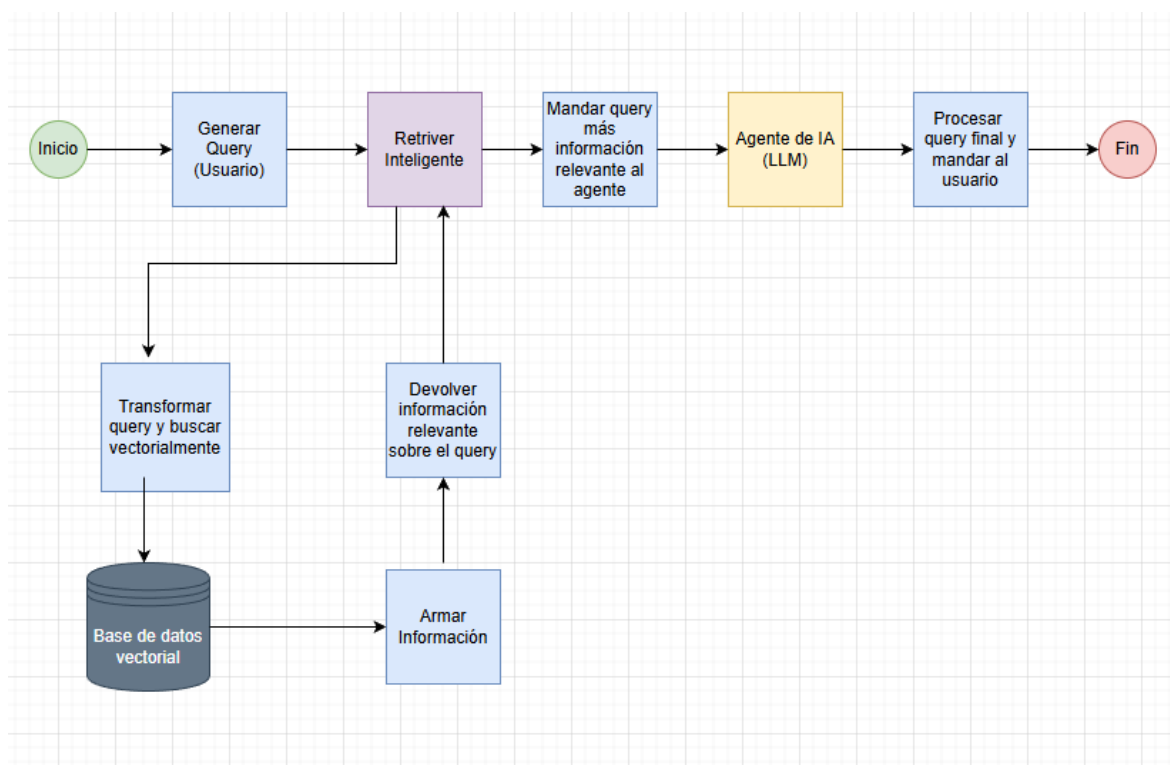
Además, la latencia es crucial para el entorno clínico, por lo cual RAG no es eficiente frente a otras arquitecturas que están basadas en el conocimiento debido a que RAG utiliza la indexación para guardar vectores que pasaron por un proceso de extracción, fragmentación e incrustación. Según (Jain, 2024) “La indexación, en términos de RAG, consiste en organizar una gran cantidad de datos textuales de forma que el sistema RAG encuentre rápidamente la información más relevante para una consulta determinada. Es como crear una biblioteca super eficiente para tu LLM, donde, en lugar de tener que hojear páginas, se

pueden identificar al instante las secciones exactas que satisfacen las necesidades del usuario”. Por lo tanto, estas técnicas internas usadas por la arquitectura RAG no son eficientes para el propósito del Agente de IA que es la latencia y poder comunicarse con un servidor de orquestación seguro y centralizado.

En conclusión, RAG es buena en entornos donde la información no cambie constantemente como el uso de chatbots inteligentes, ya que en escenarios clínicos la información es crucial y se debe trabajar de forma rápida. Existen varias fuentes de información que ayudan a la lógica del negocio, no solo se usan datos estáticos por eso es indispensable encontrar una arquitectura que se acople a la recuperación de datos vivos operacionales con una latencia baja, permitiendo que el servidor sea seguro, eficiente y centralizado.

**Figura 6**

*Diagrama de flujo RAG*



*Nota:* Diagrama de flujo RAG. Autor: Ronald Rodriguez, 2025.

### 4.1.3. Model Context Protocol (MCP) para Entornos Clínicos

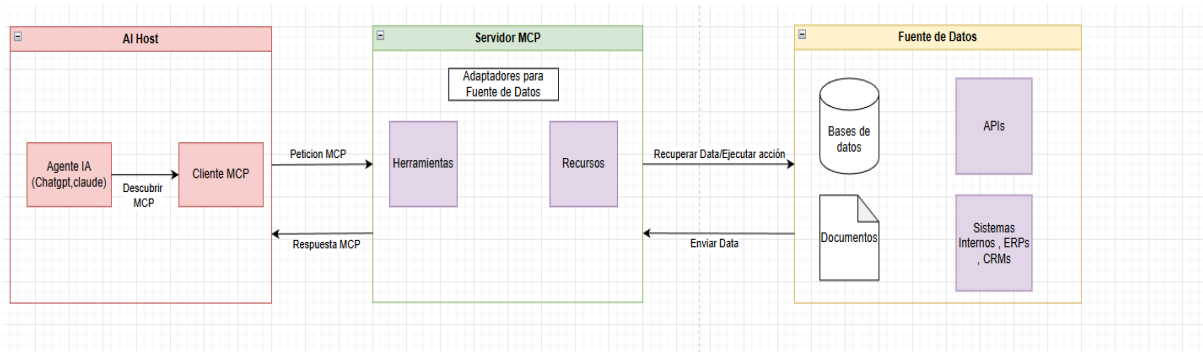
La arquitectura MCP es adaptable y escalable lo cual es fundamental para los entornos clínicos ya que permite resolver los problemas sobre otras arquitecturas que es el acceso a datos en tiempo real o realizar acciones complejas como crear recetas médicas. Según (Google Cloud, n.d.) “El protocolo de contexto de modelo (MCP) es un estándar abierto diseñado para resolver este problema. Presentado por Anthropic en noviembre del 2024, MCP proporciona un "lenguaje" seguro y estandarizado para que los LLM se comuniquen con datos, aplicaciones y servicios externos. Actúa como un puente que permite que la IA vaya más allá del conocimiento estático y se convierta en un agente dinámico que puede recuperar información actualizada y tomar medidas, lo que la hace más precisa, útil y automatizada.” Por lo tanto, esta solución es óptima para la complejidad que existe en los entornos clínicos donde la información se actualiza de manera continua y cada vez existen mas necesidades que el Agente de IA debe realizar para alcanzar su autonomía y eficiencia.

Es indispensable saber cómo funciona el Model Context protocol para poder aplicar en entornos clínicos y satisfacer las necesidades del entorno convirtiendo al Agente de IA como un asistente autónomo y sofisticado capaz de ayudar en la toma de decisiones del doctor. (Google Cloud, n.d.) nos explica que “En esencia, el protocolo de contexto del modelo (MCP) permite que un LLM solicite apoyo de herramientas externas para responder consultas o completar tareas que por sí solo no puede realizar, como acceder a bases de datos o enviar correos electrónicos; por ejemplo, ante una petición como “Busca el último informe de ventas en nuestra base de datos y envíaselo por correo a mi jefe”, el LLM primero descubre las herramientas disponibles mediante el cliente MCP, identificando una herramienta de consulta de base de datos y otra de envío de correos, luego invoca dichas herramientas generando solicitudes estructuradas (primero a `database_query` para obtener el informe), el servidor MCP ejecuta de forma segura la acción externa, traduce la solicitud a una consulta SQL, recupera y devuelve los datos al LLM, posteriormente el modelo realiza una segunda invocación a la herramienta `email_sender` para enviar el informe al destinatario indicado, recibe la confirmación de que la acción se completó correctamente y, finalmente, entrega una respuesta al usuario confirmando que el informe fue encontrado y enviado por correo.”.

En conclusión, el servidor MCP sirve como un puente centralizado que es consumido por el Agente Asistente Médico mediante el cliente MCP, el cual nos ayuda a mantener seguridad y baja latencia al ocupar herramientas y recursos que son requeridos por la petición del Agente, se debe recalcar que esta arquitectura contiene ventajas claras como menos alucinaciones por parte del Agente, conexiones más sencillas y optimizadas para el agente lo que ayuda en automatización de los procesos internos que realiza el Agente a diario en los entornos clínicos. Finalmente, (Google Cloud, n.d.) dice que “MCP fomenta un ecosistema dinámico de implementaciones de código abierto. Los desarrolladores pueden encontrar marcos de trabajo y bibliotecas de servidores de MCP predefinidos en varios lenguajes de programación, lo que puede acelerar significativamente el desarrollo. Adoptar componentes de MCP de código abierto fomenta la interoperabilidad entre distintos modelos y servicios de IA, evita la dependencia de un proveedor y permite que tu organización se beneficie de la innovación y la asistencia impulsadas por la comunidad.”

**Figura 7**

*Diagrama de flujo del MCP general*



*Nota:* Diagrama de flujo del prototipo Servidor MCP. Autor: Ronald Rodriguez, 2025.

La **Figura 7** explica de forma general el funcionamiento del Model Context Protocol (MCP), en el cual el servidor MCP aloja las herramientas y los recursos necesarios para atender las peticiones realizadas por el Agente de IA que se conecta con el cliente MCP. Las herramientas se utilizan para ejecutar acciones específicas, mientras que los recursos permiten recuperar la información solicitada en cada petición. Es importante destacar que el Agente de IA no conoce previamente las herramientas disponibles en el servidor, lo que permite un descubrimiento dinámico de capacidades y facilita la integración y explotación de múltiples fuentes de datos de manera flexible.

#### 4.1.4. Ventajas del Modelo de Orquestación MCP vs. Métodos Tradicionales

La selección de la arquitectura de integración es crucial para conseguir una viabilidad operativa y seguridad para el Agente Asistente Médico. En la presente sección, se ejecutará una comparación exhaustiva entre las arquitecturas mencionadas para lograr un servidor de orquestación seguro y adaptable para el entorno clínico ofreciendo escalabilidad para el Agente. Además, este es el punto crítico del proyecto de titulación, debido lo importante que es entender de una forma clave los beneficios que trae cada arquitectura y así poder ejecutar una decisión precisa frente a los entornos clínicos, los cuales son muy complejos debido a los datos que manejan. A continuación, se presenta la comparación entre las arquitecturas legacy, los sistemas RAG y MCP.

**Tabla 1**

*Comparativa de Arquitecturas legacy, RAG y MCP*

Dimensión Técnica	APIs Ad-hoc (Legacy)	Arquitectura RAG	Model Context Protocol (MCP)
<b>Estado del Dato</b>	Estático/Rígido	Indexado (dato viejo)	Dato vivo (Tiempo Real)
<b>Latencia</b>	Variable (depende del endpoint)	Alta (por vectorización)	Baja (Consulta directa a las fuentes de datos)
<b>Autonomía del Agente</b>	Nula (Muy tipiado)	Media (Solo lectura)	Alta (Auto – descubrimiento)
<b>Integridad Transaccional</b>	Difícil de auditar	No posee capacidad de acción	Nativa (Adaptador de herramientas)
<b>Seguridad de Contexto</b>	Basada en Tokens Simples	Filtros de búsqueda	RBAC y Validación Clínica

*Nota:* Comparativa de Arquitecturas legacy, RAG y MCP. Autor: Ronald Rodriguez, 2025.

Como se puede apreciar en la tabla comparativa, la principal ventaja de MCP es la rapidez en que podemos consultar los datos y los cuales son datos en tiempo real que es un punto muy importante para el entorno clínico frente arquitecturas como RAG donde la indexación provoca una latencia alta y debido a que solo es eficiente para recuperar datos y no ejecutar acciones. Además, frente a las APIs Ad-hoc, el MCP reduce el riesgo de que el agente ejecute acciones riesgosas, es decir reduce alucinaciones en los agentes de IA ya que el protocolo MCP estandariza sus herramientas y recursos.

#### **4.1.5. Justificación de la Selección Arquitectónica del MCP.**

El análisis teórico y comparativo de las arquitecturas de conectividad Legacy y sus limitaciones, sumado a la definición rigurosa de los requisitos operacionales críticos establece que la elección de la plataforma de middleware es el factor más determinante para el éxito del proyecto. La incapacidad de RAG para ejecutar Acción Transaccional y el riesgo de dispersión de seguridad de las APIs Ad-Hoc demuestran que ninguna solución existente cumple con el umbral de confiabilidad y control exigido por el sector salud. Por lo tanto, esta sección finaliza el debate teórico, concluyendo que el Model Context Protocol (MCP) es la única arquitectura diseñada para satisfacer integralmente las necesidades de un Agente de IA Asistente Médico.

Además, varios artículos dicen que MCP es el protocolo que ha ayudado a resolver problemas provocados por arquitecturas antiguas. Según (Kharod, 2025) “El Protocolo de Contexto de Modelo (MCP) se ha consolidado como un estándar fundamental para lograr una integración fluida entre los modelos de IA y herramientas o fuentes de datos externas, por lo que la correcta selección de servidores MCP resulta clave para los desarrolladores que desean crear aplicaciones de IA inteligentes y contextuales. Esta guía va más allá de una simple enumeración de opciones, ya que profundiza en los criterios esenciales de evaluación para elegir el servidor MCP adecuado, analiza casos prácticos y brinda consejos de implementación que facilitan la toma de decisiones informadas, siendo de utilidad tanto para desarrolladores e ingenieros de datos como para entusiastas de la inteligencia artificial que buscan construir aplicaciones más efectivas y preparadas para IA.” Por lo tanto, MCP es un protocolo nuevo que ha ido revolucionando el mundo con su capacidad potente de estandarización.

Finalmente, la elección de MCP se justifica por su capacidad de ofrecer interoperabilidad semántica esto quiere decir que MCP entrega capacidades que serán explotadas por el Agente de IA a diferencia de las APIs Ad-hoc que entregan datos que no son útiles ya que son datos sin contextos. Lo cual, reduce errores médicos que son ocasionados por alucinaciones del Agente siguiendo un contrato de comunicación estricto y estandarizado. Esta solidez ayuda a que el prototipo se pueda explotar en escenarios reales como centros hospitalarios reales.

#### 4.1.6. Las Primitivas del MCP: Herramientas (Tools) y Recursos (Resources)

Es indispensable entender la importancia de las herramientas y recursos que nos ofrece el servidor MCP ya que son fundamentales para el funcionamiento eficaz entre los Agentes de IA y los sistemas externos o fuentes de información siendo cruciales para cumplir con la lógica del negocio y así explotar la autonomía que se quiere llegar a cumplir en los Agentes de IA permitiéndoles incorporar nuevas fuentes de información en sus procesos de toma de decisiones y en la generación de contenido.

La herramienta definida como una primitiva diseñada para ejecutar una acción transacción o una función específica de lógica de negocio convirtiéndole vital en MCP. Según (Kubaski, 2025) “Los recursos MCP permiten a los servidores MCP compartir datos que proporcionan contexto a los modelos de lenguaje, como archivos, esquemas de bases de datos o información específica de la aplicación”. Estas acciones garantizan que el Agente Asistente Medico puede modificar los estados de los sistemas híbridos de manera segura y controlada.

El elemento recurso es una primitiva construida para obtener datos vivos y contexto, que solo son de lectura directa de la fuente de la verdad. Según (Kubaski, 2025) “Los recursos MCP permiten a los servidores MCP compartir datos que proporcionan contexto a los modelos de lenguaje, como archivos, esquemas de bases de datos o información específica de la aplicación”. Al aplicar esta primitiva garantiza que se consulte la información directamente de la fuente operacional, el recurso elimina el riesgo de utilizar información obsoleta y garantiza la confianza necesaria para que el agente pueda razonar de forma segura sobre el diagnóstico y la prescripción.

Finalmente, estas interacciones eficientes están respaldadas por el protocolo JSON RPC, el estándar de comunicación que usa MCP. JSON RPC sirve como la capa de transporte que asegura que las peticiones del Agente Asistente Medico (Cliente MCP) y las respuestas devueltas por el servidor viajen de manera ligera y con baja latencia. Esta velocidad es indispensable para poder entregar datos vivos operacionales para los Agentes de IA y así ofrecer eficiencia en la toma de decisiones clínicas y así garantizar que el Agente ofrezca asistencia instantánea a los requerimientos del médico durante una consulta con los pacientes. El conjunto de estas primitivas ayuda a que los Agentes de IA sean eficientes y

además convierten al MCP en la caja de herramientas universal para la integración de IA en entornos empresariales críticos como en el sector de la salud.

#### **4.1.7. El MCP como Única Solución que Cubre Seguridad, Datos Vivos y Acción**

La cobertura integral de los requisitos que necesita el Agente Asistente Medico es la razón fundamental para la selección del Model Context Protocol (MCP), ya que MCP es la única arquitectura capaz de satisfacer simultáneamente los tres pilares críticos para la operación del Agente Asistente Medico que es la acción, datos vivos y control. Mientras que el análisis previo demostró que las arquitecturas legacy fallan al cumplir estos puntos. Se reitera que la implementación de las dos primitivas mencionadas permiten la acción transaccional y garantizan datos vivos de los sistemas externos en tiempo real ayudando al Agente Asistente Medico completar su misión.

Uno de los argumentos a favor de la elección de MCP es lo eficiente que es en la seguridad y control en el entorno sanitario. El servidor MCP resuelve la principal debilidad de las arquitecturas mencionadas como RAG al imponer un punto de control y auditoría centralizado. Esta centralización es esencial ya que garantiza la trazabilidad completa y el control corporativo de todo acceso a la historia clínica electrónica, lo cual no es negociable para el cumplimiento normativo y la seguridad de la información sensible del paciente. Al actuar como filtro único, el MCP asegura que el Agente Asistente Medico solo pueda invocar las capacidades para las que ha sido autorizado.

Adicionalmente, MCP se posiciona como el habilitador de la escalabilidad, resolviendo falencias o desventajas generadas por las soluciones de conectividad personalizadas. La estandarización del protocolo permite que el hospital crezca en la adopción de IA sin tener que invertir en integraciones para nuevos agentes debido a que es estandarizado. La interoperabilidad semántica del MCP y el uso de esquemas reutilizables ayudan a que futuros agentes de IA puedan explotar su capacidad de autonomía, optimizando tiempo e inversión y asegurando un crecimiento modular que se adapta a la infraestructura tecnológica.

En conclusión, la selección del Model Context Protocol (MCP) es una decisión técnica necesaria validada por el marco teórico. Habiendo demostrado la incapacidad de las

arquitecturas legacy para el contexto de la seguridad, la estandarización, tiempo real y acción transaccional, por eso se declara que el MCP es la única solución capaz de soportar la complejidad y la criticidad del Agente Asistente Médico.

## **4.2. Configuración del Laboratorio Virtual y Entorno de Experimentación**

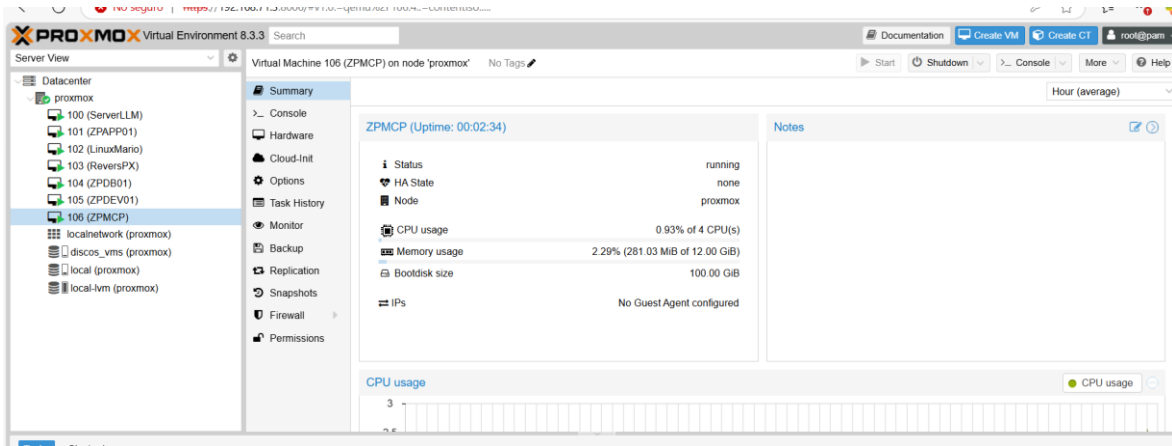
### **4.2.1. Despliegue de la Infraestructura en Proxmox VE y Debian 13**

Para la ejecución del prototipo, como antes mencionado se implementa en un entorno virtualizado debido a que responde a la necesidad de gestionar el servidor MCP de una forma aislada, lo cual garantiza una eficiencia al usar los recursos, permitiendo que el consumo de los recursos de los modelos de IA no se comprometa con la estabilidad de los servicios de la base de datos. Por otro lado, la virtualización tiene una ventaja importante que es el uso de puntos de restauración conocidos como snapshots. Según (Siebert, 2011) “Un *snapshot* (o imagen instantánea) de un disco es una copia del archivo de disco de la máquina virtual (VMDK) en un momento concreto. Conserva el sistema de archivos del disco y la memoria del sistema de nuestra VM, permitiéndonos volver a esa imagen guardada o snapshot en el caso de que algo vaya mal. Los snapshots pueden ser auténticos salvavidas cuando actualizamos o parcheamos aplicaciones y servidores. Este artículo repasa todo lo que hay que saber sobre el uso de los snapshots con VMware, incluyendo su descripción, funcionamiento y técnicas avanzadas.” Lo que asegura la continuidad del desarrollo ante posibles fallos de configuración o anomalías inesperadas.

Además, se agregó una instancia de máquina virtual utilizando el sistema operativo Debian 13 tal y como se propuso en secciones anteriores, este sistema operativo tiene una reputación excelente cuando se habla de entornos de producción por su alta estabilidad y su estricta política de seguridad, factores esenciales al manejar información. La máquina tuvo una configuración excelente que cumple con los objetivos del servidor MCP lo cual proporciona la potencia necesaria para la orquestación de servicios y el procesamiento de las peticiones JSON-RPC del Agente Asistente Médico.

## **Figura 8**

*Configuración del Servidor MCP en Promox*



*Nota:* Configuración del Servidor MCP Promox. Autor: Ronald Rodriguez, 2025.

## Figura 9

### *Sistema Operativo Configurado Servidor MCP*

```
ronald@debian:~/mcp_server/app$ uname -a
Linux debian 6.12.57+deb13-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.57-1 (2025-11-05) x86_64 GNU/Linux
ronald@debian:~/mcp_server/app$ free -h
              total        used          free      shared  buff/cache   available
Mem:           11Gi         574Mi         9.1Gi         584Ki         2.4Gi         11Gi
Swap:          5.0Gi           0B           5.0Gi
```

*Nota:* Sistema Operativo Configurado Servidor MCP. Autor: Ronald Rodriguez, 2025.

Finalmente, podemos evidenciar la configuración del servidor montado sobre la máquina virtual con 12 GB en RAM, 100 GB de disco duro y la instalación del sistema operativo Debian 13. Por lo tanto, esta configuración nos ayudara a integrar con el asistente médico y los sistemas híbridos sin ningún problema, ya que es suficiente para el prototipo que se implementó. Además, es importante saber que Promox nos brinda un monitoreo grafico muy sofisticado el cual nos ayuda a saber la saturación y el uso de los recursos de nuestro servidor como el tráfico de red o el uso de la memoria RAM en cada momento, permitiendo una eficiencia en las métricas que se realizaran al servidor MCP.

### 4.2.2. Containerización con Docker

Una vez configurado el servidor con el sistema operativo base, procedemos a crear la capa de containerización mediante Docker sobre el sistema operativo Debian 13, esta técnica permite que el servidor MCP y sus dependencias FastAPI se puedan ejecutar en un entorno

ligero y reproducible, lo cual lo vuelve independiente del sistema host. Según (Docker Inc, n.d.) “Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y fiable entre entornos informáticos. Una imagen de contenedor Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, entorno de ejecución, herramientas del sistema, bibliotecas del sistema y configuración.” Por ende, nuestro servidor queda completamente independiente de dependencias externas que podrían afectar el funcionamiento si no se usara la técnica de contenedores.

En ámbitos de seguridad el encapsular el servidor MCP nos ayuda que los atacantes no tengan acceso directo a los datos los cuales son críticos y representan un papel muy importante en el entorno clínico, por eso para poder consumir los datos o recursos del servidor se deben tener permisos para hacerlo y así impedir accesos no deseados hacia los contenedores como el de los datos híbridos. Por otro lado, Docker permite que la portabilidad lo cual facilita su implementación en ambientes de desarrollo, pruebas y producción, evitando errores de configuración entre entornos, también Docker permite escalar horizontalmente el Servidor MCP de forma controlada, facilitando la replicación de servicios ante incrementos de carga sin afectar la estabilidad del sistema.

## Figura 10

### *Elaboración del Contenedor del Servidor MCP*

```

ronald@debian:~/mcp_server/app$ sudo docker compose up -d
[+] Building 30.0s (13/13) FINISHED
=> [internal] load local bake definitions          0.0s
=> => reading from stdin 536B                    0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 342B              0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 0.7s
=> [internal] load .dockerignore                  0.0s
=> => transferring context: 2B                    0.0s
=> [1/6] FROM docker.io/library/python:3.11-slim@sha256:1dd3dca85e22 2.3s
=> => resolve docker.io/library/python:3.11-slim@sha256:1dd3dca85e22 0.0s
=> => sha256:654a090213c05af530d397771fd09de5fe6e66c2355 249B / 249B 0.3s
=> => sha256:7cc13cb22d92060ccb92030698fd1bb23d1e7 14.36MB / 14.36MB 0.8s
=> => sha256:1473863bb010703cc5da49db594d480fa6a17a3 1.29MB / 1.29MB 0.4s
=> => sha256:02d7611c4eae219af91448a4720bdba036575 29.78MB / 29.78MB 1.2s
=> => extracting sha256:02d7611c4eae219af91448a4720bdba036575d3bc035 0.5s
=> => extracting sha256:1473863bb010703cc5da49db594d480fa6a17a3a7df0 0.1s
=> => extracting sha256:7cc13cb22d92060ccb92030698fd1bb23d1e7890328d 0.3s
=> => extracting sha256:654a090213c05af530d397771fd09de5fe6e66c2355c 0.0s
=> [internal] load build context                  0.0s
=> => transferring context: 149B                  0.0s
=> [2/6] WORKDIR /app                            0.5s
=> [3/6] RUN apt-get update && apt-get install -y libpq-dev gcc && 10.5s
=> [4/6] COPY app/requirements.txt .              0.1s
=> [5/6] RUN pip install --no-cache-dir -r requirements.txt 7.2s
=> [6/6] COPY app/ .                              0.1s
=> => exporting to image                          8.5s
=> => exporting layers                            0.9s
=> => exporting manifest sha256:9f7de28aacf5f3d07e9339b29cd4699108f6d 0.0s
=> => exporting config sha256:80016b7c760bb26f73c19dc28b6090688c7054 0.0s
=> => exporting attestation manifest sha256:f0d3ac112c8847266b9ad60c 0.0s
=> => exporting manifest list sha256:f19345b0a711d84b400070cd981d57c 0.0s
=> => naming to docker.io/library/mcp_server-mcp-middleware:latest 0.0s
=> => unpacking to docker.io/library/mcp_server-mcp-middleware:lates 1.5s
=> => resolving provenance for metadata file      0.0s
[+] up 3/3
✔Image mcp_server-mcp-middleware Built 30.0s
✔Network mcp_server_default Created 0.0s
✔Container mcp_logic_server Created 1.1s

```

*Nota:* Elaboración del contenedor del servidor MCP. Autor: Ronald Rodriguez, 2025

Como se muestra en la **Figura 10** se elabora el contenedor que tendrá nuestro servidor MCP, se usó el motor Docker compose, Python para poder instanciar el contenedor con lo mínimo que se debe tener para poder integrarlo y que sea portable en ambientes distintos. Según (Docker Inc, n.d.) “Docker Compose es una herramienta para definir y ejecutar aplicaciones multicontenedor. Es la clave para lograr una experiencia de desarrollo e implementación optimizada y eficiente. Compose simplifica el control de toda la pila de aplicaciones, facilitando la gestión de servicios, redes y volúmenes en un único archivo de configuración YAML. Con un solo comando, se crean e inician todos los servicios desde el archivo de configuración.” Por lo cual, este motor es poderoso si se trata de ser un servidor escalable y practico que se necesita integrar en diferentes entornos.

## Figura 11

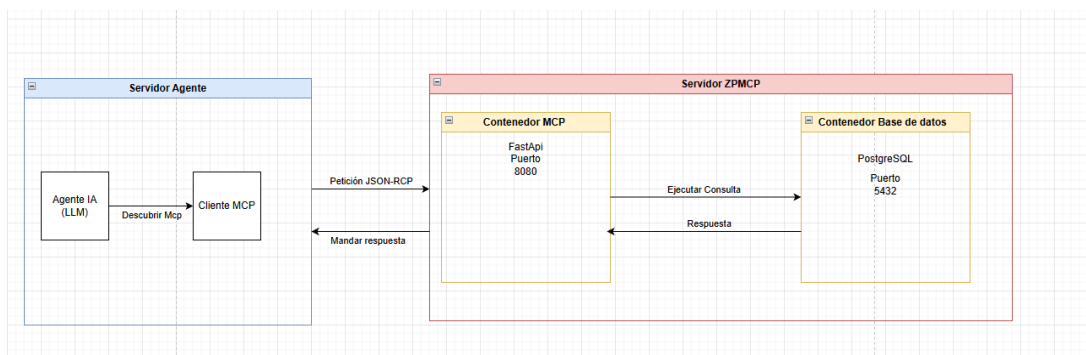
### Estado Contenedor MCP

```
ronald@debian:~/mcp_server$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED
STATUS        PORTS                                NAMES
b7e9beac5680   mcp_server-mcp-server               "python main.py"                      18 seconds ago
Up 17 seconds  0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp  mcp_logic_server
5f3e0b1dfbe9   postgres:15                          "docker-entrypoint.s..."            2 minutes ago
Up 17 seconds  0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp  mcp_db_container
ronald@debian:~/mcp_server$
```

*Nota:* Estado Contenedor del Servidor MCP. Autor: Ronald Rodriguez, 2025.

## Figura 12

### Diagrama de Comunicación Servidor MCP



Nota: Diagrama de Comunicación Servidor MCP. Autor: Ronald Rodriguez, 2025.

La arquitectura del sistema se diseñó bajo un modelo de microservicios desacoplados, separando la lógica de orquestación de la capa de persistencia mediante dos contenedores independientes. El primero, contenedor MCP, gestiona el protocolo MCP y la comunicación con el Agente de IA; el segundo, contenedor PostgreSQL, actúa como un repositorio dedicado para los datos híbridos. Esta separación no solo optimiza el uso de recursos en el servidor Debian 13, sino que también establece un perímetro de seguridad donde la base de datos es accesible únicamente para el middleware a través de una red virtual privada, cumpliendo con los estándares de diseño de software modular para entornos clínicos

Finalmente, la interacción entre los contenedores se rige por un esquema de orquestación donde el Middleware actúa como el núcleo de procesamiento. El stack tecnológico compuesto por FastAPI permite una comunicación asíncrona y eficiente, fundamental para procesar las consultas de la inteligencia artificial sin bloqueos de memoria. Aunque la capa de persistencia (PostgreSQL) se encuentra definida en la infraestructura, su integración final está supeditada a la carga de los esquemas híbridos, garantizando que el Middleware sea capaz de discernir entre recursos de lectura real y herramientas de escritura prototipada una vez que el entorno de datos sea desplegado.

#### **4.2.3. Implementación de la Base de Datos Híbrida**

La arquitectura de persistencia se fundamenta en un motor PostgreSQL 15 orquestado mediante Docker, lo que garantiza el aislamiento de los datos y la portabilidad del sistema en el entorno Debian 13. La base de datos, denominada `hce_clinica`, actúa como un repositorio híbrido que integra un esquema relacional complejo diseñado para la Historia Clínica Electrónica (HCE). Este esquema incluye tablas críticas para el seguimiento de enfermedades crónicas, como `blood_pressure` para el control de hipertensión y `blood_sugar` para el monitoreo glucémico. Es esencial dejar en claro que la entidad relación es de un ambiente real en los entornos clínicos mientras que los datos son genéricos y no necesitan ser anonimizados ya que no se altera la integridad de los datos.

Para asegurar la integridad y la trazabilidad de los datos, esta base de datos viene con disparadores automatizados que gestionan las marcas de tiempo en cada registro de pacientes y médicos. Esta capa de datos permite que el Agente de IA realice consultas dinámicas y

registros de auditoría en tiempo real a través de la tabla `mcp_audit_logs`. La naturaleza híbrida permite combinar registros médicos de grado real con perfiles de prototipos para pruebas de estrés del algoritmo de diagnóstico, exponiendo el servicio a través del puerto 8080 para la interacción con el middleware MCP.

La gestión de la persistencia se realiza mediante volúmenes de Docker, vinculando el directorio físico `postgres_data` del host con el sistema de archivos del contenedor. Según (IONOS, 2022) “Un volumen de Docker es un mecanismo que permite persistir y gestionar datos de forma independiente al ciclo de vida de los contenedores, de modo que la información se conserva incluso cuando un contenedor es eliminado. Los volúmenes facilitan la transferencia y el intercambio de datos tanto entre el sistema host y uno o varios contenedores, como directamente entre contenedores. A diferencia del sistema de archivos en capas (Union File System), los volúmenes existen fuera de la capa de solo lectura y la capa de escritura del contenedor, funcionando como una carpeta compartida accesible desde el host y los contenedores. Gracias a esto, se convierten en una solución eficiente y segura para almacenar información persistente, reutilizar datos y mantener la integridad de la información en entornos basados en Docker.” Por lo tanto, estrategia asegura que la información de los mil registros generados para la fase de pruebas permanezca resiliente ante fallos o reinicios del servidor. La implementación concluye con una interfaz de red virtual que permite al servidor MCP comunicarse de forma segura con PostgreSQL, validada mediante un estado `healthy` en el orquestador de contenedores.

### Figura 13

#### *Orquestación de Servicios con Docker*

```

=> resolving provenance for metadata file
[+] up 4/4
✓ Image mcp_server-mcp-server Built
✓ Network mcp_server_default Created
✓ Container mcp_db_container Healthy
✓ Container mcp_logic_server Created
ronald@debian:~/mcp_server$ docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
586a31b27737	mcp_server-mcp-server	"python main.py"	36 seconds ago	Up Less than a second	0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp	mcp_logic_server
943fb7c671e0	postgres:15	"docker-entrypoint.s..."	36 seconds ago	Up 35 seconds (healthy)	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp	mcp_db_container

```

ronald@debian:~/mcp_server$ █
21687188 @ 1 1 0 1 2

```

*Nota:* Orquestación de Servicios con Docker. Autor: Ronald Rodriguez, 2025.

### Figura 14

## Verificación del Esquema Relacional

```
ronald@debian:~/mcp_server$ sudo docker exec -it mcp_db_container psql -U admin_mcp -d hce_clinica -c "\dt"
[sudo] password for ronald:
List of relations
Schema |          Name          | Type | Owner
-----+-----+-----+-----
public | alcohol_consumption   | table | admin_mcp
public | appointments           | table | admin_mcp
public | blood_pressure         | table | admin_mcp
public | blood_sugar            | table | admin_mcp
public | cie10                  | table | admin_mcp
public | customers              | table | admin_mcp
public | dietary_habits         | table | admin_mcp
public | doctors                | table | admin_mcp
public | forms                  | table | admin_mcp
public | forms_data             | table | admin_mcp
public | frecuency              | table | admin_mcp
public | hipertension_arterial_form | table | admin_mcp
public | mcp_audit_logs         | table | admin_mcp
public | meditation_generic     | table | admin_mcp
public | objetives              | table | admin_mcp
public | perceived_stress_level | table | admin_mcp
public | physical_activity      | table | admin_mcp
public | sleep_quality          | table | admin_mcp
public | smoking_habits         | table | admin_mcp
public | symptoms_catalog       | table | admin_mcp
public | upload_photo           | table | admin_mcp
(21 rows)
```

*Nota:* Verificación del Esquema Relacional. Autor: Ronald Rodriguez, 2025.

**Tabla 2**

### *Detalle de componentes del servidor MCP*

Componente	Detalle Técnico	Propósito
<b>Motor DB</b>	PostgreSQL 15.15	Persistencia relacional clínica.
<b>Volumen</b>	./postgres_data	Persistencia persistente en el host Debian.
<b>Puerto MCP</b>	8080 (FastAPI)	Comunicación con el Agente de IA.
<b>Puerto DB</b>	5432 (TCP)	Acceso interno y externo para auditoría.

*Nota:* Detalle de componentes del servidor MCP. Autor: Ronald Rodriguez,

Finalmente, se describen los pilares fundamentales empleados en la configuración del contenedor, donde se incorporó una capa de seguridad basada en trazabilidad. Esta se materializa en la tabla `mcp_audit_logs`, la cual registra de forma automática cada interacción que un médico o paciente realiza con el Agente de IA a través del servidor MCP. La estructura de entidad-relación implementada refleja un entorno clínico real, permitiendo que el prototipo sea validado bajo escenarios auténticos del sector salud. Esta aproximación no solo garantiza la persistencia de los datos híbridos, sino que impulsa la fiabilidad técnica del sistema ante las demandas operativas de una institución médica.

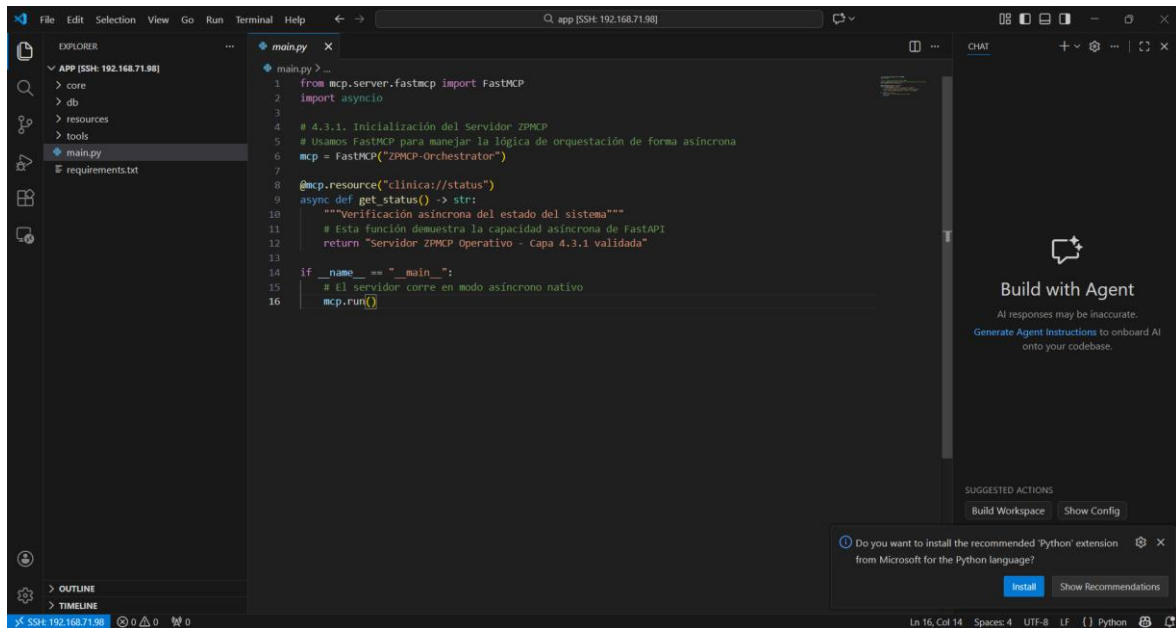
#### **4.3.1. Construcción del Servidor MCP con Python y FastAPI**

La construcción del middleware ZPMCP se basa en el lenguaje de Python, utilizando el framework FastAPI el cual es el núcleo de orquestación del proyecto de titulación, permitiendo la integración nativa con el protocolo MCP, facilitando la exposición de servicios médicos hacia el agente de IA. La estructura de este proyecto fue realizada de forma modular, que consiste en distribuir las responsabilidades en directorios especializados para la gestión de base de datos, recurso y herramientas y así seguir un mantenimiento eficiente siguiendo los estándares que provee la metodología XP.

Uno de los componentes críticos que existen en entornos clínicos son las consultas masivas extensas por lo cual la implementación de lógica asíncrona mediante `async` y `await` es fundamental para que el servidor ZPMCP procese múltiples peticiones simultáneas del Agente de IA sin bloquear el hilo principal de ejecución. Por lo tanto, la asíncronía de FastAPI asegura que el sistema mantenga una alta disponibilidad y tiempos de respuestas optimizados, incluso bajo cargas de trabajo elevadas.

Finalmente, el archivo `main.py` actúa como el punto de entrada y orquestador del sistema, inicializando la clase `FastMCP` que gestiona el ciclo de vida de los contenedores Docker previamente configurados. Esta capa de software funciona como un puente inteligente que traduce las peticiones de lenguaje natural del Agente en operaciones estructuradas de datos. Al utilizar un servidor de puerta de enlace asíncrono, el middleware ZPMCP logra una comunicación fluida entre la capa de presentación de la IA y la capa de persistencia de datos híbridos.

Figura 15

*Visual Studio Code Estructura de archivos*

*Nota:* Visual Studio Code Estructura de archivos. Autor: Ronald Rodriguez, 2025.

### 4.3.2. Implementación de la Capa de Recursos

El flujo comienza cuando el Agente de IA identifica la necesidad de obtener contexto clínico y emite una solicitud estructurada hacia el middleware utilizando el protocolo JSON-RPC. A diferencia de una base de datos tradicional, el Agente no realiza consultas directas, sino que solicita un recurso mediante una URI semántica. Este método garantiza una capa de abstracción total, donde el Agente solo conoce la interfaz de datos médicos, mientras que la complejidad de la infraestructura permanece oculta tras el servidor ZPMCP.

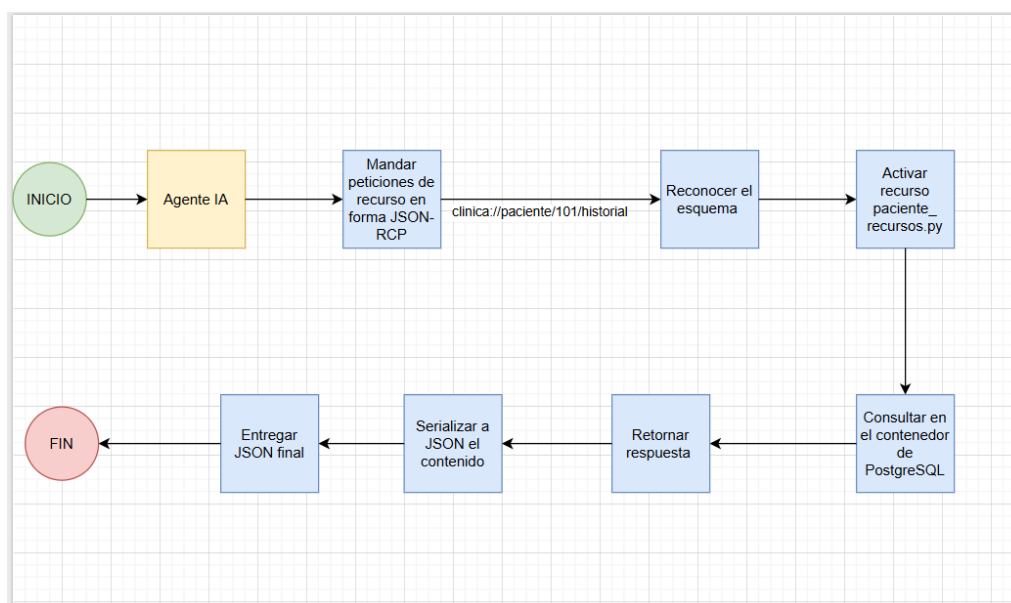
Una vez recibida la petición, el orquestador activa la lógica asíncrona definida en el módulo de recursos. El middleware realiza una conexión interna hacia el contenedor de persistencia para extraer la información necesaria. Durante este paso, se aplica una capa de enriquecimiento donde se etiquetan los registros según su origen (real o simulado). Esta capacidad de procesar datos híbridos en tiempo real es lo que permite al sistema ofrecer un contexto clínico veraz y, al mismo tiempo, seguro para la experimentación con prototipos.

Finalmente, el middleware consolida la información en un objeto JSON que incluye tanto los datos clínicos como los metadatos de auditoría. Esta respuesta es enviada de vuelta

al Agente de IA, quien integra el recurso en su ventana de contexto de trabajo. Este ciclo de consulta asegura que el razonamiento de la inteligencia artificial esté siempre anclado a la evidencia disponible en el servidor, minimizando las alucinaciones y maximizando la precisión de las recomendaciones médicas proporcionadas al usuario.

**Figura 16**

*Diagrama de flujo del uso de recurso MCP*



*Nota:* Diagrama de flujo del uso de recursos MCP. Autor: Ronald Rodriguez, 2025

### 4.3.3 Implementación de la Capa de Herramientas

La capa de herramienta constituye la capacidad operativa del middleware ZPMCP, permitiendo al Agente de IA pasar de la observación a la acción dentro del entorno clínico. Mientras que los recursos proporcionan el contexto, las herramientas ejecutan funciones transaccionales que pueden alterar el estado de la base de datos híbrida. Estas funciones están diseñadas bajo un esquema de ejecución segura, donde cada acción solicitada por la inteligencia artificial es validada por el servidor antes de ser procesada en el contenedor de persistencia, garantizando que la generación de documentos o alertas siga las reglas de negocio médico predefinidas.

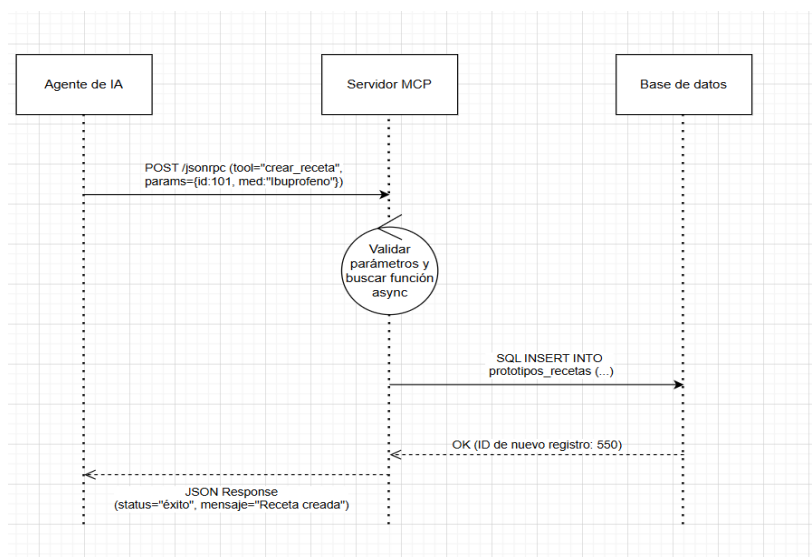
En el marco de esta investigación, se desarrollaron herramientas específicas para la creación de prototipos clínicos y la automatización de documentos. La herramienta de

generación de recetas médicas, por ejemplo, permite que la IA estructure prescripciones basadas en el historial del paciente, las cuales son almacenadas con un marcador de simulación para propósitos de la tesis. Asimismo, se implementó una lógica de generación de alertas transaccionales que notifica sobre posibles riesgos detectados en el análisis del Agente. Esta capacidad de escritura es fundamental para validar cómo un sistema basado en MCP puede asistir activamente en la carga administrativa del personal de salud mediante la creación de borradores médicos precisos.

Técnicamente, cada herramienta se implementó como una función asíncrona decorada por el protocolo MCP, lo que facilita una respuesta inmediata hacia el Agente tras la ejecución de una tarea. Al interactuar con datos híbridos, las herramientas de escritura aplican una lógica de segregación: los datos prototipados provenientes del HCE se mantienen protegidos, mientras que los prototipos de simulación se insertan en tablas dedicadas o se marcan con metadatos específicos de auditoría. Este enfoque permite realizar pruebas de estrés y simulaciones de atención médica complejas sin comprometer la base de datos real del proveedor, cumpliendo con los objetivos de seguridad e integridad planteados en la metodología.

**Figura 17**

*Diagrama de secuencia herramientas MCP*



*Nota:* Diagrama de secuencia herramientas MCP. Autor: Ronald Rodriguez, 2025.

Como se muestra en la Figura 16, se evidencia el uso práctico de las herramientas del servidor MCP, donde se aprecia como el orquestador valida y decide que funciones asincrónicas son efectivas para realizar la petición estructurada con JSON-RCP del agente de IA. Por lo tanto, se cumplen todos los objetivos mencionados como la baja latencia y la autonomía de la agente de IA mediante el servidor MCP que es capaz de tomar las decisiones más acertadas, además el uso de herramientas asincrónicas ayuda al servidor cumplir ejecuciones rápidas y sí que se corten los hilos de cada transacción.

#### **4.3.4 Integración del Adaptador de Seguridad y Autenticación**

La integración del adaptador de seguridad se fundamenta en un modelo de Control de Acceso Basado en Roles (RBAC) de naturaleza dinámica, el cual condiciona la ejecución de las herramientas del servidor MCP a la validación de credenciales en tiempo real contra la tabla doctors. Este módulo realiza una verificación de integridad sobre el identificador y el perfil del profesional antes de autorizar el acceso a la información sensible almacenada en la Historia Clínica Electrónica. Este mecanismo asegura que la interacción del Agente de IA con más de 1,000 registros de pacientes se realice bajo un marco estricto de seguridad clínica y autenticación persistente.

La validación de contexto clínico se programó mediante una capa de intermediación que analiza la legitimidad de cada consulta antes de procesarla en el motor PostgreSQL. El servidor MCP cruza la identidad del solicitante con los registros de la tabla appointments o mcp\_audit\_logs, asegurando que el flujo de datos entre el contenedor mcp\_logic\_server y mcp\_db\_container responda exclusivamente a necesidades médicas documentadas. Este adaptador actúa como un cortafuegos lógico que previene fugas de información y garantiza que el acceso a biomarcadores, como la presión arterial o niveles de glucosa, esté plenamente justificado por el rol del usuario.

Finalmente, el sistema de RBAC se vinculó de forma intrínseca a las herramientas del protocolo MCP, donde cada función exige un doctor\_id verificado como parámetro de entrada. Cada acceso a los recursos del sistema es capturado por el motor de auditoría, generando un rastro histórico en la tabla mcp\_audit\_logs que vincula al médico con la acción específica y el paciente consultado. Esta trazabilidad dinámica permite auditar el comportamiento del agente en entornos de alta demanda, transformando la arquitectura de

contenedores en un ecosistema de salud auditable, seguro y alineado con los estándares de protección de datos médicos.

**Tabla 3**

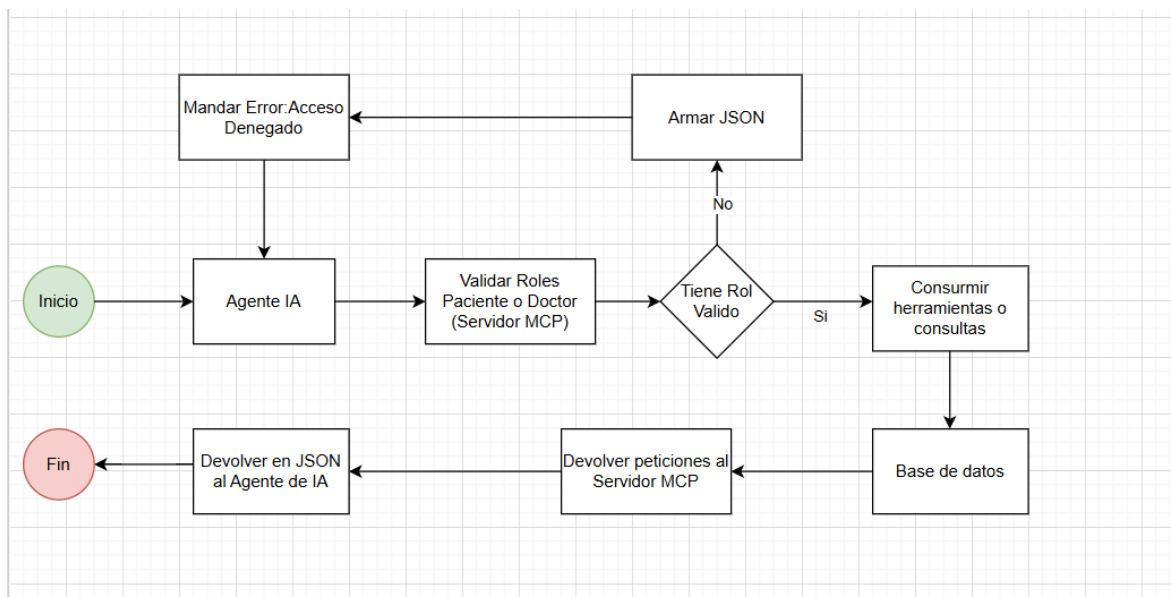
*Matriz de Control de Acceso (RBAC)*

Rol de Usuario	Recurso	Herramienta	Nivel de Acceso
<b>Doctor</b>	Completo (HCE)	Recetas / Alertas	Administrativo
<b>Paciente</b>	Solo personal	Ninguna	Consultas básicas

*Nota:* Matriz de Control Acceso (RBAC). Autor: Ronald Rodriguez, 2025.

**Figura 18**

*Diagrama de flujo acceso a herramientas/recursos del Servidor MCP*



*Nota:* Diagrama de flujo acceso a herramientas/recursos del Servidor MCP. Autor: Ronald Rodriguez, 2025.

**Figura 19**

*Código Fuente Autenticación en Sistema*

```

@app.post("/mcp/read-resource")
async def read_resource(req: ReadResourceRequest):
    usuario = validar_usuario_por_cedula(req.cedula)

    if not usuario:
        raise HTTPException(
            status_code=403,
            detail="Usuario no existe en el sistema"
        )

    # aquí NO validas HCE, eso lo hace MCP
    return await read_resource_from_mcp(req.uri)

class CallToolRequest(BaseModel):
    tool: str
    arguments: dict
    rol: str          # doctor | paciente
    cedula: str       # viene del sistema

@app.post("/mcp/call-tool")
async def call_tool(req: CallToolRequest):
    # 🚩 Validar que el usuario exista en el sistema
    usuario = validar_usuario_por_cedula(req.cedula)

    if not usuario:
        raise HTTPException(
            status_code=403,
            detail="Usuario no existe en el sistema"
        )

    rol_db = usuario["usuarioCargo"].lower()

```

Nota: Código Fuente Autenticación en Sistema. Autor: Ronald Rodriguez,2025

## Figura 20

### Código de integración de seguridad por RBAC

```

def autorizar_mcp(
    rol: str,
    nombre: str,
    cedula_doctor: str | None = None,
    dni_paciente: str | None = None
):
    rol = rol.lower()

    if nombre in RECURSOS_PUBLICOS:
        return True

    if rol == "doctor":
        if cedula_doctor and not existe_doctor_hce(cedula_doctor):
            raise PermissionError("Doctor no existe en HCE")

        # Tools
        if nombre.startswith("tool_") and nombre not in TOOLS_DOCTOR:
            raise PermissionError("Tool no permitida para doctor")

        # Resources
        if nombre.startswith("hce://") and nombre not in RECURSOS_DOCTOR:
            raise PermissionError("Resource no permitida para doctor")

    # =====
    # 🚩 PACIENTE
    # =====
    elif rol == "paciente":
        if dni_paciente and not existe_paciente_hce(dni_paciente):
            raise PermissionError("Paciente no existe en HCE")

        # Tools
        if nombre.startswith("tool_") and nombre not in TOOLS_PACIENTE:

```

Nota: Código de integración de seguridad por RBAC. Autor: Ronald Rodriguez,2025

Como se puede apreciar en las Figuras 18 y 19, se detalla la integración de la capa de seguridad diseñada para satisfacer los requerimientos de RBAC (Control de Acceso Basado en Roles) y autenticación de usuarios. La primera fase de seguridad se implementa a nivel de FastAPI en el servidor MCP, en esta etapa, la conexión hacia el motor de base de datos PostgreSQL se gestiona mediante cadenas de conexión robustecidas con cifrado simétrico, empleando librerías de estándar criptográfico. Finalmente, se aplica una validación jerárquica por roles para regular el acceso a cada herramienta y recurso del sistema. Este flujo garantiza la integridad de los datos mediante el registro mandatorio de logs de auditoría, asegurando la trazabilidad total de las interacciones entre el Agente de IA y los registros clínicos.

**Figura 21**

### *Prueba Unitaria de Roles con Herramienta*

The figure displays two screenshots of the Postman interface, illustrating a unit test for role-based access control (RBAC) on the MCP server.

**Left Screenshot (Successful Request):**

- Method:** POST
- URL:** http://192.168.71.98:8080/mcp/call-tool
- Body (JSON):**

```

1 {
2   "tool": "tool_solicitar_cita",
3   "arguments": {
4     "dni_paciente": "1700000001",
5     "rol": "paciente"
6   },
7   "rol": "paciente",
8   "cedula": "1700000001"
9 }
10

```
- Response (JSON):**

```

1 {
2   "_meta": null,
3   "content": [
4     {
5       "type": "text",
6       "text": "{\r\n \"estado\": \"Solicitud de cita registrada\"\n}",
7       "annotations": null,
8       "_meta": null
9     }
10  ],
11   "structuredContent": null,
12   "isError": false
13 }

```

**Right Screenshot (Unauthorized Request):**

- Method:** POST
- URL:** http://192.168.71.98:8080/mcp/call-tool
- Body (JSON):**

```

1 {
2   "tool": "tool_solicitar_cita",
3   "arguments": {
4     "dni_paciente": "1700000001"
5   },
6   "rol": "doctor",
7   "cedula": "1700000001"
8 }
9

```
- Response (JSON):**

```

1 {
2   "detail": "Rol no autorizado"
3 }

```
- Status:** 403 Forbidden (172 ms, 162 B)

*Nota:* Prueba Unitaria de Roles con Herramienta. Autor: Ronald Rodriguez,2025

Finalmente, se realizaron validaciones mediante la herramienta Postman, tal como se muestra en la Figura 20, donde se ejecutaron pruebas sobre las herramientas expuestas por el servidor MCP utilizando distintos roles de usuario. Estas pruebas unitarias permiten verificar la integridad lógica del sistema, asegurando que cada herramienta solo pueda ser utilizada por los perfiles autorizados según las reglas de control de acceso definidas. Por otro lado, la librería Pydantic es fundamental aquí ya que su rol es revisar los argumentos que estén todos los que espera la herramienta y que sean del tipo de dato especificado.

#### 4.3.5. Interfaz de Comunicación

La interfaz de comunicación entre el Agente de IA Médico y el servidor MCP se implementó mediante un endpoint desarrollado con FastAPI, el cual permite el intercambio directo de información entre ambos componentes. Esta configuración facilitó una comunicación fluida dentro del sistema, manteniendo separada la lógica del agente respecto a los servicios internos que gestionan los recursos y herramientas médicas.

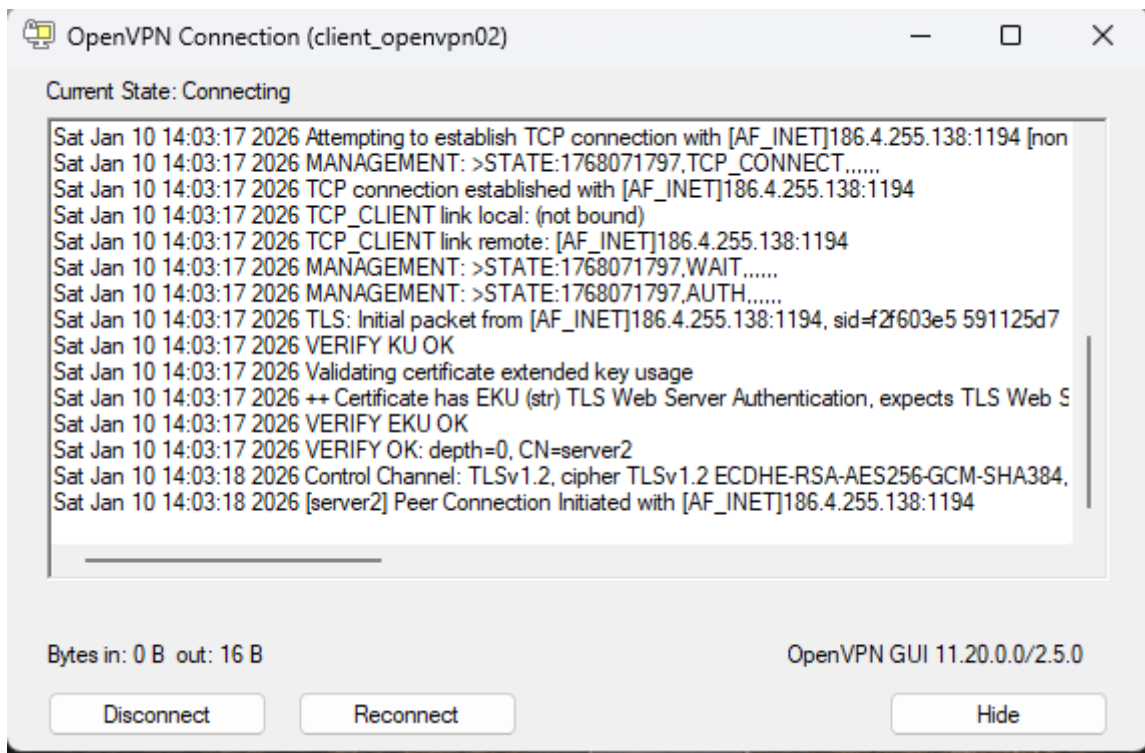
El agente de IA y el servidor MCP se encuentran desplegados dentro de la misma red privada, lo que permite que la comunicación entre ambos se realice de forma directa sin necesidad de mecanismos adicionales de tunelización. Al operar en un entorno de red controlado, los servicios no se exponen públicamente a Internet, reduciendo el riesgo de accesos no autorizados y simplificando la arquitectura de comunicación.

En caso de requerir acceso al sistema desde ubicaciones externas, se definió el uso de una red privada virtual VPN como una capa adicional de seguridad. Según (FORTINET, n.d.) “Una VPN, es decir, una red privada virtual enmascara su dirección de protocolo de Internet (IP), creando una conexión privada desde una conexión Wi-Fi pública. Una VPN es una de las mejores herramientas de privacidad y anonimato para un usuario conectado a cualquier servicio público de Internet porque establece conexiones seguras y cifradas.” Esta medida garantiza que únicamente usuarios autenticados puedan integrarse a la red interna y acceder a los endpoints del servidor FastAPI, manteniendo protegidos los datos clínicos y los servicios críticos del sistema.

Una vez establecida la conectividad dentro de la red privada, el agente de IA envía las solicitudes al endpoint FastAPI del servidor MCP, el cual procesa las peticiones, interactúa con la base de datos y devuelve las respuestas correspondientes. Este diseño permite conservar los datos clínicos dentro del entorno del servidor, asegurando la integridad de la información y la correcta trazabilidad de las operaciones realizadas por el sistema.

#### **Figura 22**

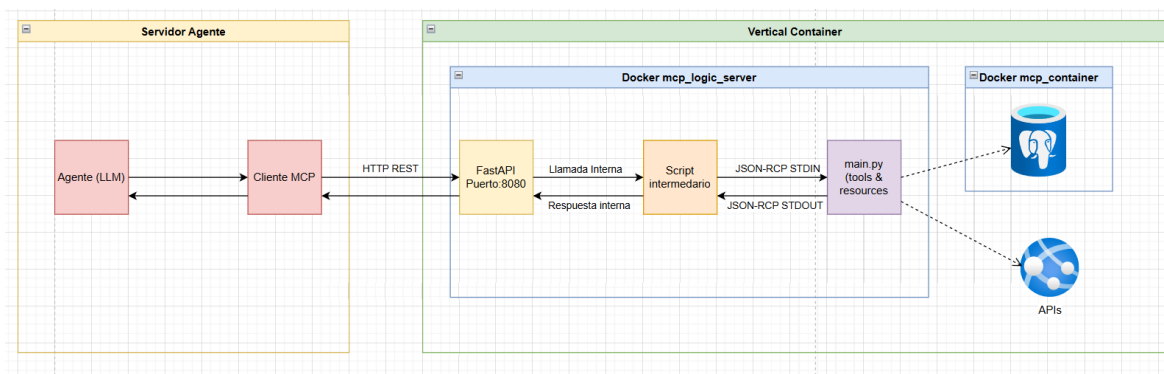
*Conexión de VPN OPENVPN*



*Nota: Conexión de VPN OPENVPN. Autor: Ronald Rodriguez,2025*

**Figura 23**

*Diagrama de conexión interfaz de comunicación*



*Nota: Diagrama de conexión interfaz de comunicación. Autor: Ronald Rodriguez,2025*

#### 4.4. Integración y Pruebas del Agente de IA Médico

La integración del Agente de IA Médico con el servidor MCP se abordó siguiendo los principios de la metodología XP (eXtreme Programming), priorizando iteraciones cortas, validación temprana y pruebas continuas. Durante las primeras iteraciones se identificaron problemas de conectividad relacionados principalmente con la comprensión del protocolo

de comunicación utilizado por MCP y su desacoplamiento respecto a los servicios HTTP tradicionales.

Inicialmente, se asumió que el servidor MCP podía ser consumido directamente como una API REST, lo cual generó errores recurrentes de comunicación y respuestas inesperadas. A partir de pruebas exploratorias (TDD), se determinó que MCP opera sobre el protocolo JSON-RPC utilizando un transporte basado en entrada y salida estándar (stdio), y no mediante endpoints HTTP directos. Este hallazgo permitió replantear la arquitectura de integración, introduciendo una capa intermedia tipo gateway desarrollada con FastAPI, cuya función es traducir solicitudes HTTP del agente a llamadas JSON-RPC compatibles con MCP.

El ajuste mínimo necesario para restablecer la conectividad consistió en separar claramente las responsabilidades de cada componente. El agente de IA se mantuvo como un servicio HTTP, mientras que el servidor MCP se ejecutó como un proceso independiente bajo el modelo stdio. La comunicación entre ambos se logró mediante una clase cliente (MCPClient), encargada de invocar de forma asíncrona los métodos de descubrimiento (/mcp/discover), lectura de recursos y ejecución de herramientas, encapsulando los detalles del protocolo subyacente.

Siguiendo los principios de TDD, cada corrección fue validada mediante pruebas manuales controladas, utilizando herramientas como Postman para verificar la correcta propagación de las solicitudes desde el agente hasta el servidor MCP. Una vez establecida la conexión, se confirmó que el agente podía consultar dinámicamente las capacidades disponibles del servidor MCP y, con base en reglas de decisión definidas en el backend, seleccionar la acción más adecuada sin necesidad de codificar de forma rígida los nombres de las herramientas.

## **Figura 24**

*Integración del Agente con MCP*

```

@app.post("/ask_agentic_mcp", response_model=RagResponse)
async def ask_agentic_mcp(request: RagRequestCombinado):
    try:
        t0 = time.perf_counter()
        validate_user_query(request.pregunta, source="/ask_agentic_mcp")
        _guard_dependency("qdrant")
        _guard_dependency("openai")
        metrics.record_event("request_total", {"endpoint": "ask_agentic_mcp"})

        agent = _get_agent_or_503()
        history = get_session_history(request.session_id, request.identificador)
        full_query = f"(Paciente: {request.identificador}) {request.pregunta}"
        chunks = retrieve_chunks(full_query, request.identificador)
        context_text = _build_context_text(chunks)

        mcp_prompt = request.pregunta
        mcp_result = await agent_decide_mcp_action(
            mcp_prompt,
            mcp_client,
            dni_paciente=dni_paciente,
            id_doctor=id_doctor,
        )
    )

```

*Nota:* Integración del Agente con MCP. Autor: Ronald Rodriguez, 2025

## Figura 25

### *Evidencia de Conexión entre el Agente de IA y Servidor MCP*

```

2026-01-08 00:04:40,526 - httpx - INFO - HTTP Request: POST http://192.168.71.98:8080/mcp/discover "HTTP/1.1 200 OK"
Conectado correctamente
tools: [{"name": "tool_emitir_prescripcion", "description": "", "input_schema": {"properties": {"id_doctor": {"title": "Id Doctor", "type": "integer"}, "diagnostico": {"title": "Diagnostico", "type": "string"}, "tratamiento": {"title": "Tratamiento", "type": "string"}}, "required": ["id_doctor", "dni_paciente", "diagnostico"]}, "output_schema": {"properties": {"id_doctor": {"title": "Id Doctor", "type": "integer"}, "dni_paciente": {"title": "Dni Paciente", "type": "string"}, "tratamiento": {"title": "Tratamiento", "type": "string"}}, "required": ["id_doctor", "dni_paciente", "tratamiento"]}, "title": "tool_emitir_prescripcion", "type": "object"}, {"name": "tool_triage_urgencias", "description": "", "input_schema": {"properties": {"id_doctor": {"title": "Id Doctor", "type": "integer"}, "dni_paciente": {"title": "Dni Paciente", "type": "string"}, "urgencias": {"title": "Urgencias", "type": "string"}}, "required": ["id_doctor", "dni_paciente", "urgencias"]}, "output_schema": {"properties": {"id_doctor": {"title": "Id Doctor", "type": "integer"}, "dni_paciente": {"title": "Dni Paciente", "type": "string"}, "urgencias": {"title": "Urgencias", "type": "string"}}, "required": ["id_doctor", "dni_paciente", "urgencias"]}, "title": "tool_triage_urgencias", "type": "object"}, {"name": "tool_triage_urgenciasArguments", "description": "", "input_schema": {"properties": {"id_doctor": {"title": "Id Doctor", "type": "integer"}, "dni_paciente": {"title": "Dni Paciente", "type": "string"}, "urgencias": {"title": "Urgencias", "type": "string"}}, "required": ["id_doctor", "dni_paciente", "urgencias"]}, "output_schema": {"properties": {"id_doctor": {"title": "Id Doctor", "type": "integer"}, "dni_paciente": {"title": "Dni Paciente", "type": "string"}, "urgencias": {"title": "Urgencias", "type": "string"}}, "required": ["id_doctor", "dni_paciente", "urgencias"]}, "title": "tool_triage_urgenciasArguments", "type": "object"}]
resources: [{"uri_template": "hce://paciente/{dni}", "description": "", "mimeType": "text/plain"}]

```

*Nota:* Integración del Agente con MCP. Autor: Ronald Rodriguez, 2025

La **Figura 24** muestra el flujo final de integración. En una primera etapa, la solicitud del usuario es recibida por el agente de IA. Posteriormente, el agente evalúa la intención de la consulta y determina si es necesario interactuar con servicios externos. Cuando la consulta lo requiere, el agente consulta al servidor MCP mediante el mecanismo de descubrimiento para identificar los recursos y herramientas disponibles. Finalmente, la **Figura 25** muestra la conexión que existe entre el Agente de IA y el Servidor MCP donde el agente recibe las herramientas y recursos disponibles para usar, además la ejecución se realiza de forma controlada a través del gateway, garantizando que el agente no tenga conocimiento previo ni acceso directo a la lógica interna del MCP.

Este enfoque permitió resolver los problemas iniciales de conectividad con un impacto mínimo en la base del sistema, manteniendo una arquitectura desacoplada, extensible y alineada con prácticas ágiles. Además, la aplicación de XP y TDD facilitó la

detección temprana de errores conceptuales, reduciendo el retrabajo y asegurando que cada incremento funcional fuese validado antes de avanzar a la siguiente iteración.

#### **4.4.1 Configuración del Cliente MCP y Conectividad con el LLM**

La configuración del cliente MCP se diseñó como un componente intermedio que permite al Agente de IA comunicarse de manera controlada con el servidor de orquestación, sin exponer directamente la lógica interna ni los recursos clínicos. Este cliente actúa como un puente entre el razonamiento del modelo de lenguaje (LLM) y la capa de ejecución del sistema, permitiendo que las decisiones generadas por el agente sean procesadas de forma estructurada. De esta manera, el LLM se limita a interpretar la intención de la consulta, mientras que la ejecución efectiva de recursos y herramientas queda bajo el control del servidor MCP.

La conectividad entre el cliente MCP y el LLM se estableció mediante un flujo desacoplado, en el cual el modelo de lenguaje no interactúa directamente con la base de datos ni con los servicios clínicos. En su lugar, el LLM genera una representación contextual de la solicitud, la cual es evaluada por el cliente MCP junto con las reglas definidas en el servidor. Este enfoque permite mantener un control preciso sobre las acciones que pueden ejecutarse, evitando que el modelo de lenguaje tenga autonomía total sobre operaciones sensibles como el acceso a historiales clínicos o la generación de acciones médicas.

Este esquema de comunicación favorece una arquitectura modular y escalable, ya que el cliente MCP puede adaptarse a distintos modelos de lenguaje sin modificar la lógica central del sistema. Asimismo, permite realizar pruebas y ajustes de manera incremental, alineándose con los principios de la metodología XP y facilitando la validación progresiva del comportamiento del agente. Como resultado, se logra una integración eficiente entre el LLM y el servidor de orquestación, donde la inteligencia artificial aporta valor en el análisis de la consulta, mientras que el control operativo permanece en una capa segura y auditable.

#### **Figura 26**

*Implementación Cliente MCP*

```

main.py IM, M  guardrails.py  mpc_router.py U  Codex Diff  mcp_client.py X
mcp_client.py > MCPClient > read_resource
1  import httpx
2
3  class MCPClient:
4      def __init__(self, base_url: str):
5          self.base_url = base_url
6
7      async def discover(self):
8          async with httpx.AsyncClient() as client:
9              r = await client.post(f"{self.base_url}/mcp/discover")
10             r.raise_for_status()
11             return r.json()
12
13     async def read_resource(self, uri: str):
14         async with httpx.AsyncClient() as client:
15             r = await client.post(
16                 f"{self.base_url}/mcp/read-resource",
17                 json={"uri": uri}
18             )
19             r.raise_for_status()
20             return r.json()
21
22     async def call_tool(self, tool_name: str, args: dict):
23         async with httpx.AsyncClient() as client:
24             r = await client.post(
25                 f"{self.base_url}/mcp/call-tool",
26                 json={
27                     "tool": tool_name,
28                     "arguments": args
29                 }
30             )
31             r.raise_for_status()
32             return r.json()
33

```

Nota: Implementación Cliente MCP. Autor: Ronald Rodriguez, 2025

Figura 27

### Implementación Enrutador MCP

```

mpc_router.py > agent_decide_mcp_action
1  from mcp_client import MCPClient
2
3  async def validarConexionMCP(mcp: MCPClient):
4      discover = await mcp.discover()
5      tools = discover.get("tools", [])
6      resources = discover.get("resource_templates", [])
7      if not resources:
8          resources = discover.get("resources", [])
9
10     print("Conectado correctamente")
11     print(f"tools: {tools}")
12     print(f"resources: {resources}")
13     return discover
14
15     async def get_mcp_capabilities(mcp: MCPClient):
16         discover = await mcp.discover()
17         return {
18             "tools": discover.get("tools", []),
19             "resource_templates": discover.get("resource_templates", [])
20         }
21
22     async def agent_decide_mcp_action(
23         prompt: str,
24         mcp: MCPClient,
25         *,
26         dni_paciente: str | None = None,
27         id_doctor: int | None = None,
28     ):
29         prompt_lower = prompt.lower()
30         capabilities = await get_mcp_capabilities(mcp)
31
32         # ---- RESOURCES ----
33         for r in capabilities["resource_templates"]:

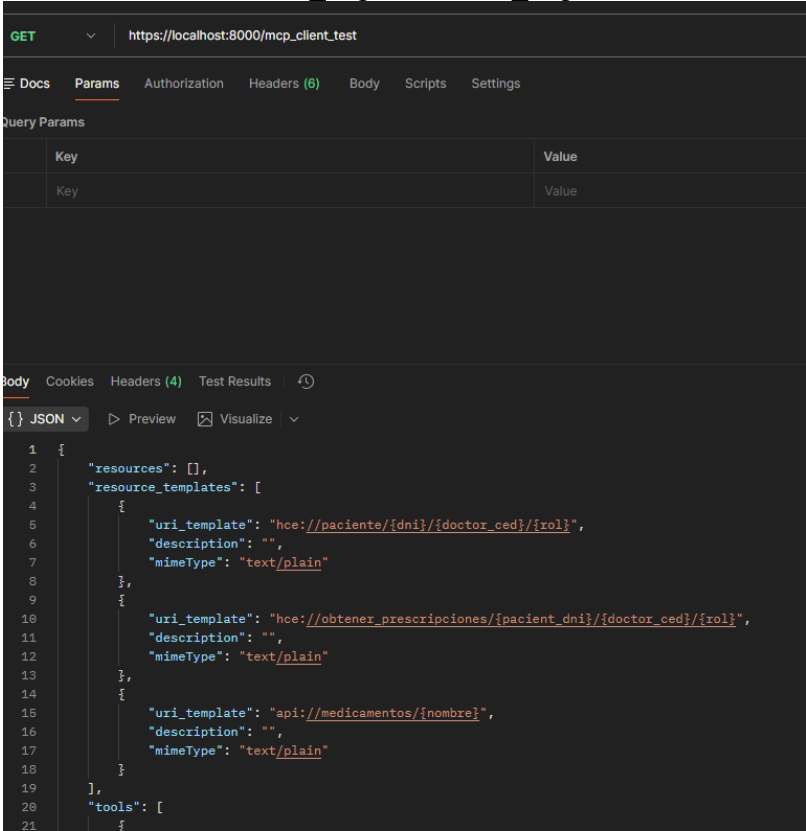
```

Nota: Implementación Enrutador MCP. Autor: Ronald Rodriguez, 2025

Como podemos observar en la **Figura 26** y **Figura 27** es la manera de realizar el cliente MCP que esta dentro de la solución del Agente Asistente Medico , donde con el cliente creamos los métodos básicos y útiles para ejecutar una herramienta , recurso y listar los recursos o herramientas disponibles para posteriormente mientras el agente realiza una conversación siempre consume el método `agent_decide_mcp_action` el cual le sirve para evaluar si la pregunta requiere de MCP o simplemente el LLM puede resolver por su cuenta, donde el agente descubre MCP , es decir no tiene idea de las herramientas solo las usa dependiendo de la pregunta.

**Figura 28**

*Prueba Unitaria cliente mcp - servidor mcp*



```

1  {
2    "resources": [],
3    "resource_templates": [
4      {
5        "uri_template": "hce://paciente/{idni}/{idocctor_ced}/{irol}",
6        "description": "",
7        "mimeType": "text/plain"
8      },
9      {
10       "uri_template": "hce://obtener_prescripciones/{ipacient_dni}/{idocctor_ced}/{irol}",
11       "description": "",
12       "mimeType": "text/plain"
13     },
14     {
15       "uri_template": "api://medicamentos/{inombre}",
16       "description": "",
17       "mimeType": "text/plain"
18     }
19   ],
20   "tools": [
21     {

```

*Nota:* Prueba Unitaria cliente\_mcp - servidor\_mcp. Autor: Ronald Rodriguez, 2025

Como demuestra la **Figura 28**, ejecutamos una prueba unitaria usando la herramienta Postman , donde comprobamos la comunicación entre el cliente que esta de lado del servidor a (Agente) con el servidor b (MCP). Durante esta prueba se puede visualizar el listado de las herramientas y recursos que hay en el servidor. Por otro lado, verificamos las peticiones a nuestro servidor por medio de Uvicorn. Según (GeeksforGeeks, 2025)“Uvicorn es el

servidor ASGI ligero recomendado para FastAPI, optimizado para programación asíncrona. Conocido por su excepcional rendimiento al gestionar código asíncrono, es el favorito de los usuarios de FastAPI por su diseño intuitivo, ideal tanto para entornos de desarrollo como de producción. Uvicorn también ofrece la práctica opción `--reload`, que reinicia automáticamente el servidor al detectar cambios en el código, lo que resulta especialmente útil en la fase de desarrollo.” Por esta razón es optimo usar uvicorn para implementar las pruebas unitarias e integradas ya que ofrece una baja latencia y alta eficiencia en el manejo de procesos asincrónicos.

#### 4.4.2 Ejecución de Casos de Uso

Se ejecutaron cuatro casos de uso validándolas con pruebas unitarias e integrales utilizando Postman como cliente HTTP. Según (QAlified, 2023) “Postman es una herramienta de pruebas de API utilizada por más de 20 millones de usuarios. Ha ganado una inmensa popularidad en el sector de IT entre desarrolladores y testers. La interfaz sencilla y fácil de usar ayuda a la hora de documentar, diseñar y probar las APIs. Puedes crear peticiones de puntos finales de API, enviar varios tipos de datos y evaluar respuestas sin esfuerzo con Postman. Su interfaz gráfica de usuario, sencilla e informativa facilita incluso a los principiantes menos experimentados la interacción con las APIs sin necesidad de profundizar en un código complejo.” Esta herramienta permitió ejecutar los casos de uso de manera óptima y así poder medir la latencia que existe por cada ejecución, hay que recalcar que se hizo uso del Agente Asistente Medico, donde se observa como la respuesta es fusionada con el razonamiento de LLM y MCP, también se verifico las capas de seguridad por RBAC. La interfaz grafica no es prioritaria ya que no esta en el alcance del middleware MCP, sin embargo, he integrado una interfaz gráfica para poder validar la primera capa de seguridad que es la autenticación de los usuarios.

Para garantizar un servidor MCP eficiente y escalable hemos escogido las siguientes historias de usuario:

- **Caso A:** “Consultar el historial clínico del paciente X para su respectivo análisis y así poder tomar decisiones”
- **Caso B:** “Generar una prescripción médica para un paciente X con base a su diagnóstico clínico”
- **Caso C:** “Quiero consultar como paciente la información de un medicamento X y para qué sirve”

- **Caso D:** “Revisar la prescripción medicas que tiene el paciente X para dar seguimiento al tratamiento del paciente”

Siguiendo la práctica TDD de la metodología XP, antes de integrar cada caso de uso con el agente, se realizaron pruebas unitarias y funcionales a nivel de API sobre cada tool o resource, ya que estas se encuentran implementadas como endpoints en FastAPI y se comunican directamente con el servidor MCP. Para cada herramienta se definieron cuatro pruebas, las cuales permitieron validar su comportamiento de manera aislada utilizando Postman. Una vez superadas estas pruebas, se procedió con la integración de las tools con el agente.

**Figura 29**

### Prueba Unitaria Caso A

The screenshot shows a Postman REST client interface. The request is a POST to `http://192.168.71.98:8080/mcp/read-resource`. The body is a JSON object with the following content:

```

1 {
2   "uri": "hce://paciente/1700000001/DOC0005/doctor",
3   "cedula": "1700000001",
4   "rol": "doctor"
5 }

```

The response status is 200 OK, with a time of 619 ms and a size of 903 B. The response body is a JSON object with the following content:

```

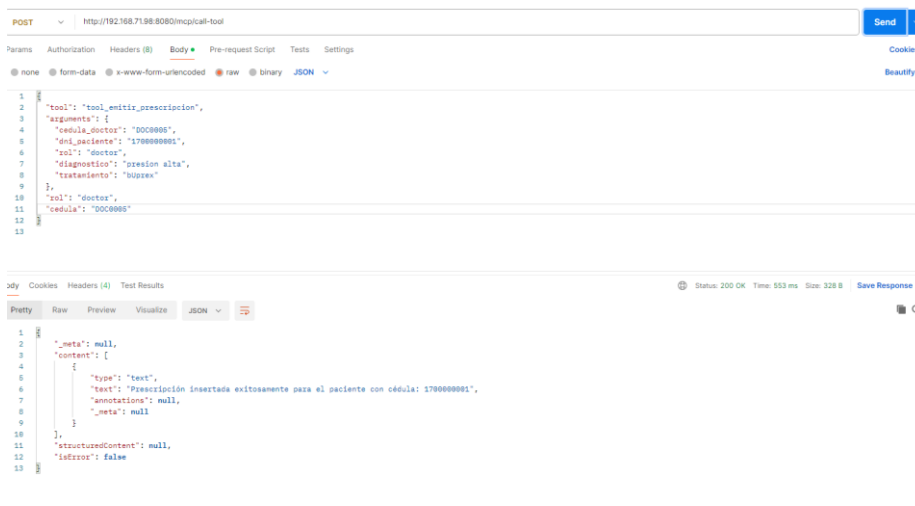
1 {
2   "uri": "hce://paciente/1700000001/DOC0005/doctor",
3   "content": {
4     {
5       "uri": "hce://paciente/1700000001/DOC0005/doctor",
6       "mimeType": "text/plain",
7       "_meta": null,
8       "text": "{\n  \"id\": 1005,\n  \"first_name\": \"Paciente1\",\n  \"last_name\": \"Apellido1\",\n  \"email\": null,\n  \"phone\":\n    null,\n  \"identification\": \"1700000001\",\n  \"birth_date\": null,\n  \"gender\": null,\n  \"address\": null,\n  \"emergency_contact\": null,\n  \"blood_type\": null,\n  \"allergies\": \"Ninguna\",\n  \"medical_history\": \"Sin\n    antecedentes\",\n  \"insurance_provider\": null,\n  \"insurance_number\": null,\n  \"created_at\": \"2026-01-08T19:57:53.\n    584815\",\n  \"updated_at\": \"2026-01-08T19:57:53.584815\",\n  \"is_active\": 1,\n  \"systolic_pressure\": 122,\n  \"diastolic_pressure\": 84\n}"
9     }
10   }
11 }

```

*Nota: Prueba Unitaria Caso A. Autor: Ronald Rodriguez,2025*

**Figura 30**

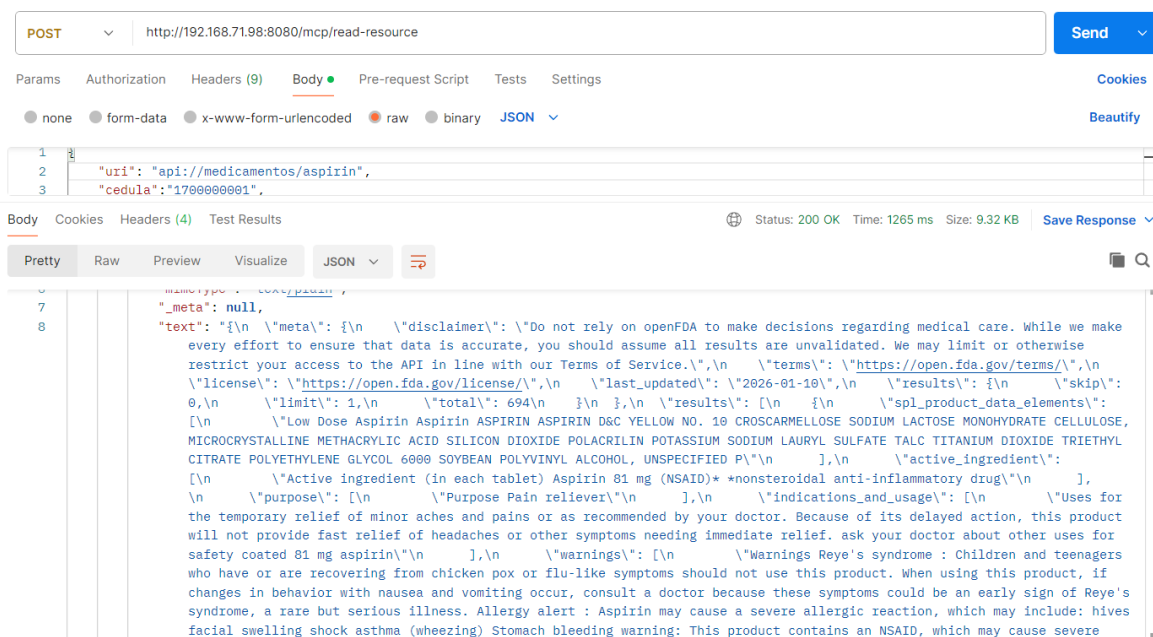
### Prueba Unitaria Caso B



*Nota: Prueba Unitaria Caso B. Autor: Ronald Rodriguez,2025*

### Figura 31

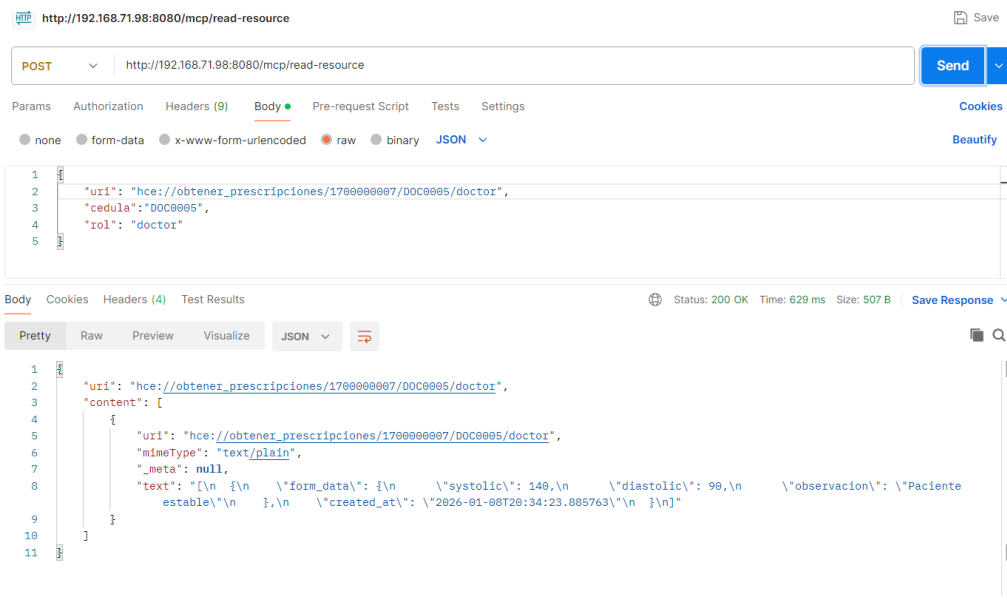
#### Prueba Unitaria Caso C



*Nota: Prueba Unitaria Caso C. Autor: Ronald Rodriguez,2025*

### Figura 32

#### Prueba Unitaria Caso D

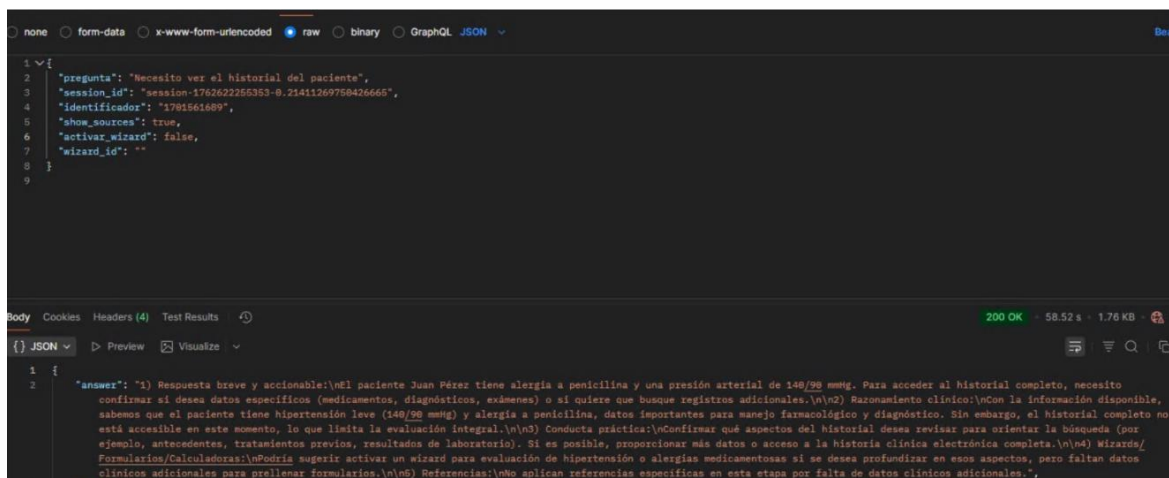


*Nota: Prueba Unitaria Caso D. Autor: Ronald Rodriguez,2025*

Como se puede observar en las **Figuras 29, 30, 31 y 32**, se realizaron pruebas sobre cada herramienta o recurso que cumple con el objetivo definido para cada caso de uso. Estas pruebas permitieron validar de manera aislada la lógica y el comportamiento de cada recurso, garantizando su correcto funcionamiento antes de su integración con el agente. De esta forma, se cumple con la fase TDD propuesta por la metodología XP, asegurando un desarrollo incremental, confiable y alineado a los requerimientos establecidos.

### Figura 33

#### Implementación de Agente + MCP Caso A



*Nota: Implementación de Agente + MCP Caso A. Autor: Ronald Rodriguez,2025*

En base la **Figura 29** nos podemos fijar lo poderoso que es el LLM y aun mas cuando se le agrega MCP debido a que su respuesta fue muy eficiente y correcta e incluso se anima a recetar según el historial obviamente pidiendo el permiso del doctor para ejecutar, al realizar la pregunta el Agente Asistente Medico consumió un recurso debido a que solo era una petición de consulta usando la siguiente herramienta **hce://paciente/{dni}**. Por lo tanto, se aprecia funcionalidad de MCP donde expuso las herramientas y recursos y el Agente tomo la decisión más adecuada para cumplir, además como nos indica la Figura 25 podemos observar el método que se encuentra en el cliente MCP usado por el Agente Asistente Medico, es aquí donde se exponen los recursos y herramientas.

## Figura 34

### Implementación de Agente + MCP Caso B

```

POST https://localhost:8000/ask_agentic_mcp

{
  "pregunta": "Generame una prescripción médica cuyo diagnóstico es presión alta y el tratamiento es lisinopril cada 15 horas y predice en que tiempo se sanara 100% con el tratamiento",
  "session_id": "session-1762622256363-0.214112697700426660",
  "identificador": "1700000001",
  "show_sources": true,
  "activar_wizard": false,
  "wizard_id": "",
  "dni_paciente": "1700000001",
  "rol": "doctor"
}

200 OK - 21.20 s - 1.88 KB

{
  "answer": "1) La prescripción para hipertensión con lisinopril debe ajustarse a dosis estándar (habitualmente una vez al día); no se recomienda cada 15 horas. La hipertensión no se cura 100%, sino que se controla crónicamente. La prescripción ya fue insertada para el paciente con cédula 1700000001. El control de la presión arterial con IECA como lisinopril es efectivo para reducir riesgos cardiovasculares, pero no elimina la enfermedad. La dosificación cada 15 horas no es estándar y puede afectar adherencia y seguridad. La hipertensión requiere seguimiento y ajuste continuo, no un plazo definido para 'cura total'. Mantener tratamiento con lisinopril en dosis adecuada (usualmente diaria), controlar presión arterial periódicamente, evaluar efectos adversos y factores de riesgo. No se puede predecir un tiempo de curación total; el objetivo es control sostenido a largo plazo. No es necesario generar nueva prescripción, pero si se desea, activar wizard_id: tool_emitir_prescripcion con paciente cédula prellenada (1700000001), diagnóstico y tratamiento ajustado (lisinopril 10-20 mg una vez al día). Falta confirmar dosis exacta y frecuencia correcta. Referencias: ESC/ESH Guidelines for hypertension management [1]. UpToDate: ACE inhibitors in hypertension [2]. American Heart Association: Hypertension control [3].",
  "refs": [

```

*Nota:* Implementación de Agente + MCP Caso B. Autor: Ronald Rodriguez,2025

A partir de la **Figura 28**, se evidencia el potencial de los LLM al integrarse con el protocolo MCP. Esta combinación permite obtener respuestas altamente eficientes y precisas. Un aspecto destacable es la capacidad del modelo para sugerir prescripciones basadas en el historial clínico del paciente, manteniendo siempre el rigor ético al solicitar la validación y autorización del facultativo antes de cualquier ejecución. En este flujo se evidencio una petición única de acción el cual es `tool_emitir_prescripcion`.

## Figura 35

### Implementación de Agente + MCP Caso C

```

1 {
2   "pregunta": "Quiero saber para que sirve el medicamento aspirin ",
3   "session_id": "session-176262255353-0.21411269758426665",
4   "identificador": "DOC0085",
5   "show_sources": true,
6   "activar_wizard": false,
7   "wizard_id": "",
8   "dni_paciente": "1700000001",
9   "rol": "paciente"
10 }

```

```

1 {
2   "answer": "1) La aspirina (ácido acetilsalicílico) se usa principalmente como analgésico para aliviar dolores leves a moderados y como antiinflamatorio. Además, en dosis bajas, se emplea para la prevención primaria y secundaria de eventos cardiovasculares ateroscleróticos, trombosis, y en ciertas condiciones específicas como preeclampsia y prevención de complicaciones en intervenciones cardiovasculares.\n\n2) La aspirina inhibe la ciclooxigenasa, reduciendo la producción de prostaglandinas y la agregación plaquetaria, lo que la hace útil para prevenir coágulos sanguíneos. Su uso abarca desde el alivio temporal de dolores hasta la profilaxis en enfermedades cardiovasculares y trombóticas. Sin embargo, tiene riesgos importantes como sangrado gastrointestinal, reacciones alérgicas y contraindicaciones en ciertos grupos (embarazo avanzado, niños con infecciones virales, pacientes con riesgo de sangrado).\n\n3) Se debe usar con precaución en pacientes con antecedentes de úlceras gástricas, trastornos de sangrado, insuficiencia hepática o renal, y evitar en el último trimestre del embarazo salvo indicación médica. La dosis baja (81 mg) es común para prevención cardiovascular, mientras que dosis mayores se usan para analgesia. Siempre consultar al médico antes de iniciar tratamiento.\n\n4) No es necesario activar wizard o formulario para esta consulta general.\n\n5) Referencias:\n- Información FDA sobre aspirina 81 mg [1].\n- Guías clínicas de prevención cardiovascular [2].\n- Recomendaciones en embarazo y uso de aspirina [3].\n- Advertencias sobre efectos adversos y contraindicaciones [4].\n\nContexto utilizado:\n-aspirin--drug-information: Aspirina: Las reacciones adversas pueden incluir sangrado en diversos sitios, siendo dependiente de la dosis, uso concomitante de otros agentes que alteran la hemostasia y susceptibilidad del paciente. Reacciones cardiovasculares incluyen arritmia, hipotensión y taquicardia. A ni_\n-aspirin--drug-information: Este texto describe el uso off-label de Aspirin en adultos para diversas condiciones. Las indicaciones incluyen la prevención primaria de enfermedades cardiovasculares ateroscleróticas, aterosclerosis carotídea, prevención del riesgo de cáncer colorrectal, tratamiento agudo de mi_",
3   "refs": [

```

*Nota:* Implementación de Agente + MCP Caso C. Autor: Ronald Rodriguez

**Figura 36**

### Implementación de Agente + MCP Caso D

```

1 {
2   "pregunta": "Obtener las prescripciones del paciente y explicar el diagnostico",
3   "session_id": "session-176262255353-0.21411269758426665",
4   "identificador": "DOC0085",
5   "show_sources": true,
6   "activar_wizard": false,
7   "wizard_id": "",
8   "dni_paciente": "1700000001",
9   "rol": "paciente"
10 }

```

```

1 {
2   "answer": "1) El paciente con identificación 1700000001 tiene múltiples prescripciones registradas para el diagnóstico de '\npresión alta', con tratamientos principalmente con '\nbiprex' y una prescripción con '\nlisinopril cada 15 horas'. El diagnóstico corresponde a hipertensión arterial.\n\n2) La hipertensión arterial es una condición crónica caracterizada por elevación sostenida de la presión arterial (en este paciente, registros recientes muestran cifras como 140/90 mmHg). Es un factor de riesgo importante para enfermedades cardiovasculares, accidente cerebrovascular y daño renal. El diagnóstico se basa en mediciones repetidas de presión arterial elevadas. El tratamiento busca controlar la presión para reducir riesgos asociados. El medicamento '\nbiprex' no es un nombre farmacológico estándar reconocido, por lo que se debe verificar su composición o si es un nombre comercial local. Lisinopril es un inhibidor de la enzima convertidora de angiotensina (IECA) comúnmente usado para tratar hipertensión. La pauta '\ncada 15 horas' no es estándar y debe revisarse para asegurar adherencia y eficacia.\n\n3) Se recomienda confirmar la medición de presión arterial en consulta, evaluar factores de riesgo y comorbilidades, y ajustar tratamiento con medicamentos antihipertensivos con dosis y frecuencia adecuadas. Es importante seguimiento periódico para evaluar control y efectos adversos. Se debe aclarar la identidad y dosis del medicamento '\nbiprex' para evitar confusión y asegurar tratamiento correcto.\n\n4) Se puede activar wizard_id: tool_emitir_prescripcion para actualizar o corregir prescripción, prellenando diagnóstico '\npresión alta' y tratamiento con lisinopril, pero ajustando frecuencia a dosis diaria estándar. Faltan datos precisos de dosis y composición de '\nbiprex'.\n\n5) Referencias:\n- 2023 ESC/ESH Guidelines for the management of arterial hypertension [1].\n- UpToDate: Hypertension diagnosis and treatment [2].\n- American Heart Association: Hypertension management [3].\n- Farmacología de IECA y pautas de dosificación [4].\n\nContexto utilizado:\n- felsemate--drug-information: No se menciona un fármaco específico en este fragmento. Se indica que se debe completar y firmar un formulario de consentimiento informado por el paciente y el médico para la prescripción, con copias disponibles a través de MEDA Pharmaceuticals.\n- medical brain 01: Este tema presenta la perspectiva de un paciente individual para proporcionar a los clínicos una visión de su experiencia, mejorando así la práctica clínica y la atención sanitaria. No busca ser exhaustivo ni ofrecer recomendaciones diagnósticas, terapéuticas o farmacológicas. Ta_",
3   "refs": [

```

*Nota:* Implementación de Agente + MCP Caso D. Autor: Ronald Rodriguez

Finalmente, se puede apreciar lo bueno que es MCP para que los agentes tomen decisiones, al ser un caso de estudio este servidor esta adaptado para tomar información en tiempo real del Historial Clínico Electrónico, sin embargo, puede conectarse a sistemas externos y buscar información que refuerza la respuesta de la pregunta de usuario, permitiendo que el LLM razone con información precisa y en tiempo real. Al final, logramos que el agente asistente medico deje de adivinar o alucinar y empiece a razonar con evidencia clínica real, entregando respuestas

#### 4.4.3 Análisis de la Interacción JSON-RPC

En esta fase de implementación, se analiza la forma en que se transporta las llamadas desde el agente hacia el Servidor MCP como antes mencionado el agente asistente medico mediante su razonamiento por LLM escoge MCP para la petición del doctor donde envía una solicitud http post y la traduce internamente a una trama de comunicación mediante el estándar JSON-RCP. Este proceso permite que el middleware utilice una interfaz externa moderna (REST) mientras aprovecha la eficiencia de la comunicación STUDIO para interactuar con el motor de la base de datos que contiene el HCE o APIs externas de ser el caso de la pregunta que realiza el Agente Asistente Medico.

#### Figura 37

*Código Fuente conexión para recurso JSON RCP STUDIO+MCP*

```
async def read_resource_from_mcp(uri: str):
    server = StdioServerParameters(
        command="python",
        args=["/home/ronald/mcp_server/app/main.py"],
    )

    async with stdio_client(server) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()

            content = await session.read_resource(uri)

            return {
                "uri": uri,
                "content": content.contents
            }
```

*Nota:* Código Fuente conexión JSON RCP STUDIO+MCP. Autor: Ronald Rodriguez,2025.

El dialogo técnico entre el middleware y el servidor MCP se basa en el intercambio de objetos JSON estructurados. Como se puede observar en la **Figura 37** la función se conecta con STUDIO para obtener el recurso y es ahí donde usas los métodos que tienen ClientSession que pertenece a la librería MCP donde se transforma el JSON RCP para obtener la respuesta del recurso en específico. Esta metodología garantiza que el Agente reciba netamente la información necesaria para procesar las peticiones, eliminando el riesgo de alucinaciones al basar su respuesta final en una fuente de verdad validada.

### Figura 38

*Evidencia tabla de Auditoria nivel*

	auditorialId	usuarioNombre	observacionAuditorialId	auditoriaObservacion	auditoriaFecha
1	1845	DOC20256789	1	Ejecución de tool MCP: tool_emitir_prescripcion	2026-01-08 17:06:22.460
2	1846	DOC20256789	1	Ejecución de tool MCP: tool_emitir_prescripcion	2026-01-08 17:12:18.293
3	1847	DOC20256789	1	Ejecución de tool MCP: tool_emitir_prescripcion	2026-01-08 17:34:28.367
4	1848	DOC20256789	1	Ejecución de tool MCP: tool_emitir_prescripcion	2026-01-08 17:59:37.913
5	1851	PAC-0002	1	Lectura de recurso MCP: api://medicamentos/paracetamol	2026-01-08 18:07:32.627
6	1852	PAC-0002	1	Lectura de recurso MCP: api://medicamentos/acetaminophen	2026-01-08 18:08:35.577
7	1853	PAC-0002	3	Rol no autorizado. Rol DB: paciente, Rol enviado: doctor	2026-01-08 18:12:15.570
8	1854	PAC-0002	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 18:14:02.600
9	1855	PAC-0002	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 18:15:49.103
10	1856	PAC-0002	3	Rol no autorizado. Rol DB: paciente, Rol enviado: doctor	2026-01-08 21:11:31.710
11	1857	PAC-0002	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 21:11:37.140
12	1858	PAC-0002	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 21:37:40.987
13	1859	PAC-0002	3	Rol no autorizado. Rol DB: paciente, Rol enviado: doctor	2026-01-08 21:39:12.417
14	1860	PAC-0002	3	Rol no autorizado. Rol DB: paciente, Rol enviado: doctor	2026-01-08 21:42:51.077
15	1861	PAC-0002	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 21:43:06.220
16	1862	PAC-0001	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 21:48:13.597
17	1863	PAC-0001	1	Ejecución de tool MCP: tool_solicitar_cita	2026-01-08 21:48:18.863
18	1864	PAC-0001	3	Rol no autorizado. Rol DB: paciente, Rol enviado: doctor	2026-01-08 21:48:24.973
19	1867	INTRUSO_API	2	Intento de uso de tool sin usuario válido. Tool: tool_solicitar_cita	2026-01-08 22:25:27.413

*Nota:* Evidencia tabla de Auditoria. Autor: Ronald Rodriguez,2025.

### Figura 39

*Evidencia de Auditoria en HCE*

id	doctor_id	accion	detalle	timestamp
1	1006	VALIDACION	Doctor existe en HCE	2026-0
1-09 04:03:51.98739				
2	2002	ERROR	Doctor no existe en HCE: ID-DOC-1008	2026-0
1-09 04:03:52.012025				
3	1006	VALIDACION	Doctor existe en HCE	2026-0
1-09 04:11:37.237951				
4	2002	ERROR	Doctor no existe en HCE: ID-DOC-1008	2026-0
1-09 04:11:37.265786				
5	1006	RESOURCE	hce://paciente/{dni}/{doctor_ced}/{rol}   Consulta historial paciente 1700000001	2026-0
1-09 04:11:37.274977				
6	1006	VALIDACION	Doctor existe en HCE	2026-0
1-09 04:13:38.902829				
7	2002	VALIDACION	Paciente existe en HCE: 1700000001	2026-0
1-09 04:13:38.927588				
8	1006	RESOURCE	hce://paciente/{dni}/{doctor_ced}/{rol}   Consulta historial paciente 1700000001	2026-0
1-09 04:13:38.936543				
9	1006	VALIDACION	Doctor existe en HCE	2026-0
1-09 04:15:48.506748				
10	2002	VALIDACION	Paciente existe en HCE: 1700000001	2026-0
1-09 04:15:48.530446				
11	1006	RESOURCE	hce://paciente/{dni}/{doctor_ced}/{rol}   Consulta historial paciente 1700000001	2026-0
1-09 04:15:48.541769				
12	1006	VALIDACION	Doctor existe en HCE	2026-0
1-09 04:16:14.987395				
13	1006	RESOURCE	hce://paciente/{dni}/{doctor_ced}/{rol}   Consulta historial paciente 1700000001	2026-0

*Nota:* Evidencia de Auditoria en HCE. Autor: Ronald Rodriguez.2025

Finalmente, se puede evidenciar en la **Figura 38** de manera dinámica los logs se guardan en la base SQL Server del sistema para validar que va ejecutando cada doctor o paciente cuando se ejecutan las herramientas o recursos es decir todo esta auditable. Por último, tenemos la otra capa de seguridad RBAC que está a nivel de MCP, validando que no se ejecuten las herramientas o recursos que no pertenecen al rol, toda esta validación es almacena en la tabla de auditoría del HCE como evidencia la **Figura 39**. Además, se incorporó un usuario señuelo en la base de datos de SQL Server que es el sistema que alberga el Agente para detectar y auditar accesos no autorizados desde sistemas externos. La integración de estos logs garantiza que el intercambio de mensajes de entre el Agente y el Middleware dejen huellas imborrables, cumpliendo con los estándares de trazabilidad exigidos en el entorno de la salud.

## 4.5. Evaluación de Resultados y Métricas de Desempeño

### 4.5.1 Análisis de Latencia en la Orquestación

Para evaluar la eficiencia del flujo de datos, se midió el Round Trip Time (RTT) de las peticiones realizadas desde el Agente de IA hacia el servidor MCP. Según “El tiempo de ida y vuelta (RTT) es la duración en milisegundos (ms) que tarda una solicitud de red en ir desde un punto de origen a un destino y regresar al punto de origen. El RTT es una métrica importante para determinar el estado de una conexión en una red local o en Internet, y los administradores de red la utilizan habitualmente para diagnosticar la velocidad y la fiabilidad de las conexiones de red.” Por ende, esta medición contempla un ciclo completo de la salida del mensaje desde el agente desarrollado en Windows, hasta el Gateway de FastApi alojado

en el servidor MCP realizado en Debian 13 todo esto en la red local, debido a que el servidor MCP y agente se albergan en la misma red, pero diferentes servidores, ayudando aún más a tener una latencia baja y optima. Se definieron procesos como la carga de 1000 registros híbridos en una sola petición, permitiendo identificar el impacto del middleware en la experiencia del usuario con rol de doctor o paciente.

Los resultados obtenidos indican que la mayor parte de tiempo del procesamiento se concentra en la lógica del razonamiento del Agente Asistente Medico, específicamente en la fase de descubrimiento y elección de herramientas o recursos ofrecidos por el servidor MCPo otros recursos ofrecidos por el servidor como consultas a APIs externas con un promedio de 200 ms. Por otro lado, la seguridad y auditoria centrada en la FastApi agrega una latencia marginal de 40 ms, además la inicialización de la sesión MCP a través del transporte STDIO agrega un promedio de 85 ms, mientras que la ejecución de la consulta SQL, sobre la tabla customers agrego un promedio de 50 ms. La latencia acumulada es de 380 ms en condiciones normales, cumpliendo con los estándares de interactividad de soporte a la decisión medica en tiempo real.

La experimentación técnica reveló una notable resiliencia en la latencia al procesar un conjunto de datos íntegramente prototipado que asciende a un volumen de 15,000 entradas estructuradas bajo un esquema relacional específico. Al ejecutar consultas masivas sobre entidades compuestas por aproximadamente 1,000 filas, se constató que la arquitectura orquestada mediante Docker junto al protocolo JSON-RPC optimizan el flujo de paquetes para que los tiempos de respuesta permanezcan invariables. Este comportamiento evidencia una gestión de recursos eficiente por parte del Agente al interactuar con el motor PostgreSQL, asegurando que la integridad del servicio y la fluidez se mantengan constantes frente a las demandas de un entorno clínico simulado. La estabilidad observada valida la escalabilidad del sistema diseñado, garantizando que el incremento en la densidad de la información procesada no degrade el rendimiento percibido durante la interacción en escenarios hospitalarios de alta exigencia.

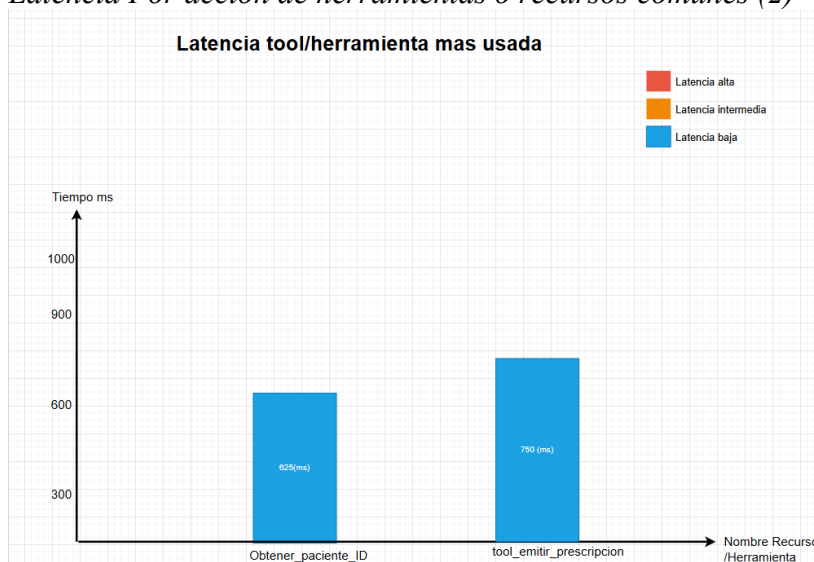
Tabla 3  
*Desglose de Tiempos de Respuesta Promedio*

Fase del Proceso	Tiempo Promedio (ms)	
<b>Petición Agente (Windows -&gt; Debian)</b>	10000 ms	Procesamiento cognitivo para generar diagnósticos y peticiones médicas y procesos internos como seguridad.
<b>Autenticación y Auditoría (FastAPI)</b>	350 ms	Validación de identificador: DOC0005 y registro en la tabla de auditoría.
<b>Apertura de Túnel STUDIO (MCP Server)</b>	200 ms	Inicialización del proceso secundario en Debian para la sesión MCP.
<b>Consulta SQL (PostgreSQL - 1002 registros(customers))</b>	766 ms	Tiempo real de recuperación de un payload de <b>587.45 KB</b>

*Nota:*

## Figura 40

*Latencia Por acción de herramientas o recursos comunes (2)*

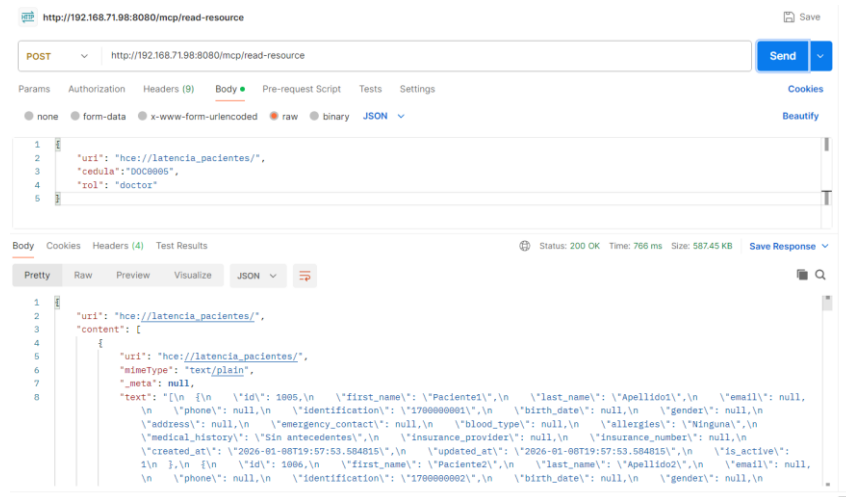


*Nota:* Latencia Por acción de herramientas o recursos comunes (2). Autor: Ronald Rodriguez,2025

Tal como se evidencia en la **Figura 40**, la ejecución de una herramienta conlleva generalmente un tiempo de respuesta superior en comparación con el acceso a un recurso estándar, sin embargo, dicha latencia dependerá estrictamente de la naturaleza del método implementado y la complejidad del recurso creado. La implementación de herramientas destinadas a la inserción de datos demanda un ciclo de ejecución extendido debido a la gestión de transacciones atómicas y a la validación rigurosa de restricciones de integridad dentro del motor relacional. Este proceso exige que el sistema coordine bloqueos de escritura y actualizaciones de índices en PostgreSQL, lo cual incrementa significativamente la carga computacional frente a las operaciones de consulta convencionales. Dicha complejidad estructural resulta indispensable para asegurar la consistencia de la información prototipada, aunque introduce una latencia adicional que debe ser evaluada minuciosamente durante el diseño de la arquitectura del agente clínico.

**Figura 41**

*Prueba de Latencia Postman Pacientes Totales*



*Nota: Prueba de Latencia Postman Pacientes Totales. Autor: Ronald Rodriguez,2025*

**Figura 42**

*Evidencia de uso de auditoria al registrar*

auditoriaId	usuarioNombre	observacionAuditoriaId	auditoriaObservacion	auditoriaFecha
1916	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:17:23.260
1917	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:17:36.417
1918	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:17:38.183
1919	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:17:39.533
1920	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:18:43.203
1921	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:19:12.920
1922	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:20:46.070
1923	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:20:55.613
1924	Doc03032	1	Lectura de recurso MCP: hce://latencia_pacientes/	2026-01-11 03:22:12.317

*Nota:* Evidencia de uso de auditoria al registrar. Autor: Ronald Rodriguez,2025

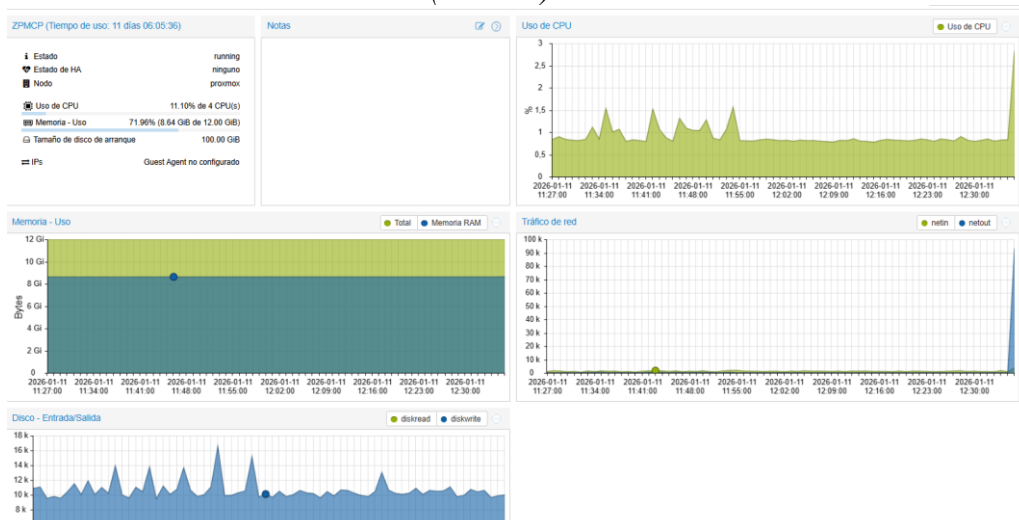
Finalmente, Para evaluar el límite operativo del prototipo, se realizó una prueba de carga masiva recuperando los datos prototipados de pacientes. Como se aprecia en la **Figura 41**, la latencia total registrada fue de 766 ms para un volumen de datos de 587.45 KB. Este tiempo incluye la validación de seguridad de la cédula 'DOC0005', la auditoría del evento como en la **Figura 42**, la apertura del proceso MCP por STUDIO y la transferencia de miles de registros desde PostgreSQL. Considerando la complejidad de la orquestación y el procesamiento del Agente.

#### 4.5.2. Evaluación del Rendimiento y Consumo de Recursos en Entorno Debian 13

Una vez validados los tiempos de respuesta y la lógica de orquestación, es imperativo analizar cómo el hardware asume esta carga de trabajo de manera sostenida. No basta con que el sistema responda una vez; en un entorno hospitalario, la concurrencia y la repetición son constantes. Por esta razón, se sometió al servidor a una prueba de estrés consistente en la ejecución de 20 ciclos iterativos de carga masiva, procesando en cada uno 1,002 registros.

#### Figura 43

##### Prueba de Estrés Servidor MCP (Promox)



*Nota:* Prueba de Estrés Servidor MCP (Promox). Autor: Ronald Rodriguez, 2025

Este escenario de prueba permite validar el comportamiento del middleware operando sobre Debian 13, lo cual garantiza que el consumo de recursos no se degrade con el tiempo ni comprometa la integridad de la historia clínica, manteniendo la disponibilidad incluso en escenarios de mucha concurrencia. En base a la **Figura 43**, analizamos tres aspectos importantes que ayudan a cumplir las expectativas por usar métricas, las cuales son:

- **Resiliencia del CPU:** El servidor mostró una alta capacidad de respuesta al pasar de un uso base del 0.79% a un pico del 11.10% durante la carga masiva, regresando a la normalidad en segundos sin presentar bloqueos. Esto confirma que el hardware asume picos de demanda clínica sin comprometer la estabilidad del sistema.
- **Consistencia de Memoria:** La RAM se mantuvo estable en 8.64 GiB durante las 20 repeticiones de la prueba, lo que demuestra que no existen fugas de memoria al procesar datos extensos. Este comportamiento plano en Proxmox garantiza que el servidor puede operar por largos periodos sin necesidad de reinicios técnicos.
- **Eficiencia de E/S (Input/Output):** El flujo de datos en red y disco fue constante, permitiendo que el payload de 1,002 registros se transmitiera sin generar latencia acumulada. Esta fluidez facilita el manejo de datos híbridos en tiempo real, asegurando una comunicación eficiente entre el agente y el backend en Debian.

Finalmente, los datos arrojados por el monitoreo en Proxmox nos dan la tranquilidad de que la arquitectura no solo es funcional, sino extremadamente robusta para el día a día de un hospital. Resulta impresionante ver cómo, tras procesar 20 veces seguidas una carga masiva de más de mil registros, el servidor Debian 13 ni siquiera se inmuta, manteniendo un uso de CPU que apenas toca el 11.10% en su punto más alto y regresando al 0.79% casi de inmediato.

#### **4.5.3 Validación del Cumplimiento de la Tríada de Seguridad**

La preservación de la Confidencialidad dentro del ecosistema propuesto se fundamenta en la implementación estricta de un middleware desarrollado en FastAPI que gestiona el control de acceso basado en roles mediante validaciones criptográficas de identidad. Durante las fases de experimentación se constató que la infraestructura restringe el acceso a los datos sensibles del historial clínico únicamente a los facultativos que poseen credenciales legítimas como el registro DOC0005 verificado en la base de datos. Esta capa de seguridad impide que cualquier entidad externa o usuario sin los privilegios adecuados

pueda invocar recursos críticos del servidor mediante el uso de decoradores asíncronos que interceptan y auditan cada solicitud en tiempo real para mitigar vulnerabilidades de exposición de información.

En lo que respecta a la Integridad de los datos circulantes se utilizó el protocolo Model Context Protocol operando sobre un transporte de flujo STDIO para asegurar que la comunicación entre el Agente y el servidor de base de datos se mantenga inalterada durante el tránsito masivo. Las pruebas de estrés que involucraron la recuperación iterativa de mil dos registros permitieron verificar que la estructura del payload permaneció idéntica desde su origen en PostgreSQL hasta su interpretación final por parte del modelo de lenguaje. Este mecanismo de transmisión directa elimina los riesgos asociados a la interceptación o degradación de paquetes en redes convencionales garantizando así que las decisiones clínicas tomadas por la inteligencia artificial se sustenten en información fidedigna y completa.

La Disponibilidad del servicio se encuentra garantizada por la resiliencia del entorno virtualizado en Proxmox el cual demostró una estabilidad operativa absoluta incluso bajo escenarios de alta demanda computacional y peticiones concurrentes. El monitoreo técnico reveló que el servidor Debian 13 gestiona picos de carga del once por ciento en la unidad central de procesamiento sin experimentar retardos significativos ni agotamiento de la memoria física disponible en el sistema. Esta capacidad de recuperación inmediata tras la ejecución de tareas complejas asegura que la herramienta permanezca accesible para el personal de salud en cualquier instante crítico de la práctica médica evitando caídas del sistema que podrían comprometer la continuidad asistencial en entornos hospitalarios de alta exigencia tecnológica.

**Tabla 4**

*Resumen Controles de Seguridad*

Pilar	Mecanismo
<b>Confidencialidad</b>	Validación de Roles/Cédulas en FastAPI.
<b>Integridad</b>	Transporte STDIO y validación de registros masivos.
<b>Disponibilidad</b>	Monitoreo de recursos en Proxmox y gestión de hilos asíncronos.

*Nota:* Resumen Controles de Seguridad. Autor: Ronald Rodriguez,2025

## **5. Capítulo 5: Conclusiones y Recomendaciones**

### **5.1 Conclusiones**

Respecto a la comparación de arquitecturas, se determinó que el estándar Model Context Protocol aventaja significativamente a las arquitecturas REST tradicionales en tareas de orquestación para agentes inteligentes debido a su capacidad intrínseca de sostener un túnel de comunicación mediante transporte STDIO que suprime la latencia por negociación de red en cada consulta realizada. Esta infraestructura técnica permitió la consolidación de datos híbridos provenientes de registros reales y prototipados bajo un esquema de seguridad unificado lo cual facilita que el modelo de lenguaje ejecute procesos de razonamiento clínico basados en evidencia actualizada y no simplemente en su base de conocimiento estática preentrenada.

En cuanto a la implementación del prototipo, se alcanzó el desarrollo de un orquestador operativo sobre el sistema Debian 13 con la facultad de centralizar recursos alojados en PostgreSQL y servicios externos representados por la API de la FDA dentro de una misma lógica de ejecución. El sistema demostró una eficiencia notable al registrar una latencia de respuesta de apenas setecientos sesenta y seis milisegundos para la recuperación masiva de mil registros clínicos garantizando que el flujo de trabajo del profesional de la salud no experimente interrupciones por tiempos de espera prolongados durante la consulta de la información del paciente.

Sobre la evaluación de la arquitectura, los indicadores obtenidos en el entorno de virtualización Proxmox ratificaron que el diseño propuesto satisface íntegramente las necesidades del asistente médico al exhibir una estabilidad sistémica absoluta frente a condiciones de alta demanda operativa. Tras la ejecución de veinte ciclos de estrés continuos el servidor mantuvo un techo de utilización de la unidad central de procesamiento de apenas el once por ciento y una ocupación de memoria RAM constante de ocho gigabytes asegurando la escalabilidad de la solución en instituciones sanitarias con elevados volúmenes de concurrencia.

En términos de Seguridad y Trazabilidad, la solución implementada garantiza la integridad de la tríada de seguridad de la información mediante la integración de middlewares específicos que gestionan la autorización de acceso y la generación automática de registros de auditoría para cada transacción realizada. Al vincular de forma unívoca la identidad del facultativo con cada recurso consultado o herramienta ejecutada se establece un marco técnico y legal robusto para el tratamiento de datos sensibles de salud cumpliendo con los estándares de confidencialidad exigidos en la gestión de historias clínicas electrónicas.

## **5.2 Recomendaciones**

Se sugiere la incorporación estratégica de un módulo de persistencia en memoria como Redis para gestionar el almacenamiento temporal de datos provenientes de la API externa que poseen baja volatilidad con la finalidad de mitigar el impacto de la latencia de red internacional en las consultas de vademécums médicos.

Resulta imperativo considerar la aplicación de protocolos de cifrado asimétrico para proteger el transporte de datos por STDIO en escenarios donde el servidor deba operar sobre segmentos de red no privados reforzando así los mecanismos de confidencialidad para alinearse con las regulaciones internacionales de privacidad de datos sanitarios.

Para el desarrollo de investigaciones futuras se propone la integración de una capa de validación humana obligatoria en el flujo de ejecución de herramientas de escritura para asegurar que el agente de inteligencia artificial funcione exclusivamente como un soporte de decisión clínica y que toda modificación en la base de datos sea previamente validada por el criterio del médico responsable.

## 6. Bibliografía

- Alejandro, A. (24 de 01 de 2024). *RAG - Retrieval Augmented Generation: La llave que abre la puerta de la precisión a los modelos del lenguaje*. Obtenido de datos.gob.es: <https://datos.gob.es/es/blog/rag-retrieval-augmented-generation-la-llave-que-abre-la-puerta-de-la-precision-los-modelos-del>
- Ali, M. (03 de 05 de 2024). *Tutorial de FastAPI: Introducción al uso de FastAPI*. Obtenido de datacamp: <https://www.datacamp.com/es/tutorial/introduction-fastapi-tutorial>
- Almeida, J. (12 de 06 de 2025). *Programación sincrónica vs. asincrónica: en qué se diferencian y cuándo usar cada una*. Obtenido de DistantJob: <https://distantjob.com/blog/synchronous-vs-asynchronous-programming/>
- BairesDev Editorial Team. (04 de 06 de 2025). *Escritura estática vs. dinámica: una comparación detallada*. Obtenido de BairesDev: <https://www.bairesdev.com/blog/static-vs-dynamic-typing/>
- Bayer, M. (22 de 06 de 2025). *SQLAlchemy Documentation (v2.0)*. Obtenido de SQLAlchemy.org.: <https://www.sqlalchemy.org/>
- Belcic, I., & Stryker, C. (03 de 03 de 2025). *¿Qué es RAG agéntica?* Obtenido de IBM: <https://www.ibm.com/mx-es/think/topics/agent-rag#:~:text=El%20sistema%20RAG%20no%20tiene,requieren%20una%20extensa%20ingenier%C3%ADa%20r%C3%A1pida.>
- Bergmann, D. (07 de 11 de 2024). *¿Qué es una ventana de contexto?* Obtenido de IBM: <https://www.ibm.com/think/topics/context-window>
- Beschokov, M. (22 de 07 de 2025). *¿Qué es JSON RPC?* Obtenido de wallarm: <https://www.wallarm.com/what/what-is-json-rpc>
- Buzdugan S. (08 de 03 de 2025). *¿Qué son los servidores MCP? La nueva tendencia de IA explicada para todos*. Obtenido de Medium: <https://medium.com/@sebuzdugan/what-are-mcp-servers-the-new-ai-trend-explained-for-everyone-8936489c561f>
- C, J. (10 de Julio de 2025). *Tengo*. Obtenido de La Caja de Herramientas Universal: La Integración Revolucionaria de Herramientas de IA con MCP y Salesforce Agentforce 3: <https://www.tengo.co/es/2025/07/10/la-caja-de-herramientas-universal-la-integracion-revolucionaria-de-herramientas-de-ia-con-mcp-y-salesforce-agentforce-3/>

- Cano Carbada, F. (1 de Julio de 2025). *Protocolo de contexto del modelo (MCP): análisis de los riesgos y los controles de seguridad*. Obtenido de Blog de Red Hat: <https://www.redhat.com/es/blog/model-context-protocol-mcp-understanding-security-risks-and-controls>
- Cloudflare. (s.f.). *¿Qué es el protocolo de Contexto Modelo (MCP)?* Obtenido de Cloudflare: <https://www.cloudflare.com/es-la/learning/ai/what-is-model-context-protocol-mcp/>
- Docker Inc. (s.f.). *What is a container?* Obtenido de Docker: <https://www.docker.com/resources/what-container/>
- Ezra, O. (14 de septiembre de 2025). *¿Qué es un cliente MCP y cómo encaja en el protocolo MCP?* Obtenido de K2VIEW: <https://www.k2view.com/blog/mcp-client/#What-is-an-MCP-client>
- FORTINET. (s.f.). *¿Qué es una VPN? Explicación de redes privadas virtuales*. Obtenido de FORTINET: <https://www.fortinet.com/lat/resources/cyberglossary/what-is-a-vpn>
- GeeksforGeeks. (23 de 07 de 2025). *GeeksforGeeks*. Obtenido de API rápida: Unicorn vs. Uvicorn: <https://www.geeksforgeeks.org/python/fast-api-unicorn-vs-uvicorn/>
- González Guerra, R. S. (10 de Marzo de 2025). El futuro de la IA conectada: Qué es un servidor MCP y por qué podría reemplazar los sistemas RAG. *Hiberus Blog*, 1. Obtenido de <https://www.hiberus.com/crecemos-contigo/el-futuro-de-la-ia-conectada-que-es-un-servidor-mcp/#:~:text=MCP%20permite%20conectar%20cualquier%20modelo,plataformas%20y%20bases%20de%20datos>
- Goodwin, M. (15 de 08 de 2023). *¿Por qué es importante la latencia de una red?* Obtenido de IBM: <https://www.ibm.com/es-es/think/topics/latency>
- Google Cloud. (s.f.). *¿Qué es el MCP y cómo funciona?* Obtenido de Google Cloud: <https://cloud.google.com/discover/what-is-model-context-protocol#understanding-the-model-context-protocol>
- Greg, L., & Matthew, K. (s.f.). *¿Qué es la interoperabilidad?* Obtenido de IBM: <https://www.ibm.com/mx-es/think/topics/interoperability>
- IONOS. (13 de 04 de 2022). *Comprender y gestionar los Docker volumes*. Obtenido de IONOS: <https://www.ionos.com/es-us/digitalguide/servidores/know-how/docker-container-volumes/>

- Jain, M. (5 de 04 de 2024). *RAG: Parte 4: Indexación*. Obtenido de Medium: <https://medium.com/@j13mehul/rag-part-4-indexing-1985f4000f72>
- Kharod, S. (27 de 05 de 2025). *La guía definitiva de servidores MCP: las mejores opciones para crear aplicaciones compatibles con IA*. Obtenido de trebble: <https://trebble.com/blog/mcp-servers-guide>
- Kubaski, L. (01 de 08 de 2025). *Recursos de MCP explicados (y en qué se diferencian de las herramientas de MCP)*. Obtenido de MEDIUM: <https://medium.com/@laurentkubaski/mcp-resources-explained-and-how-they-differ-from-mcp-tools-096f9d15f767>
- Mesh, F., & Ian, S. (14 de 08 de 2022). *IBM*. Obtenido de ¿Qué es Linux?: <https://www.ibm.com/es-es/think/topics/linux>
- Microsoft Azure. (s.f.). *¿Qué es middleware?* Obtenido de Microsoft: <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-middleware>
- Nosowitz, D., & Goodwin, M. (5 de 08 de 2024). *¿Qué es un endpoint de API?* Obtenido de IBM: <https://www.ibm.com/es-es/think/topics/api-endpoint>
- Pawar, P. (16 de abril de 2025). *¿Qué es MCP en IA y sus beneficios?* Obtenido de AALPHA: <https://www.aalpha.net/blog/what-is-mcp-in-ai/>
- QALified. (09 de 08 de 2023). *Una guía completa para pruebas de API con Postman*. Obtenido de QALified : <https://qalified.com/es/blog/postman-para-api-testing/>
- Quiroz, C. (06 de 02 de 2025). *Cinco métricas de SLA que debe monitorizar*. Obtenido de IBM: <https://www.ibm.com/es-es/think/topics/sla-metrics>
- Raeburn, A. (13 de 02 de 2025). *asana*. Obtenido de La programación extrema (XP) produce resultados, pero ¿es la metodología adecuada para ti?: <https://asana.com/es/resources/extreme-programming-xp>
- Red Hat. (2024 de 05 de 15). *¿Qué es la generación aumentada por recuperación?* Obtenido de Red Hat: <https://www.redhat.com/es/topics/ai/what-is-retrieval-augmented-generation>
- Red Hat. (23 de 12 de 2024). *¿Qué es el control de acceso?* Obtenido de Red Hat: <https://www.redhat.com/es/topics/security/what-is-access-control>
- Siebert, E. (04 de 05 de 2011). *Los snapshots de VMware: Cómo funcionan*. Obtenido de ComputerWeekly: <https://www.computerweekly.com/es/consejo/Los-snapshots-de-VMware-Como-funcionan>

- Sommerville, I. (2011). *Ingeniería de Software. (9.ª ed.)*. Lancaster: Pearson Educación.
- Stryker, C. (15 de 08 de 2024). *IBM*. Obtenido de ¿Qué es la IA agentiva?: <https://www.ibm.com/es-es/think/topics/agentive-ai>
- Susnjara, S., & Smalley, I. (s.f.). *¿Qué es la monitorización de servidores?* Obtenido de IBM: <https://www.ibm.com/es-es/think/topics/server-monitoring>
- Tuychev, B. (25 de 06 de 2025). *Pydantic: Una guía con ejemplos prácticos*. Obtenido de datacamp: <https://www.datacamp.com/es/tutorial/pydantic>
- Ulili, S. (14 de 03 de 2025). *Introducción a HTTPX: el cliente HTTP moderno de Python*. Obtenido de Better Stack: <https://betterstack.com/community/guides/scaling-python/httpx-explained/>
- Valderrama, J. P. (2025). *Uso de Agentes de Inteligencia*. Escola Tècnica Superior, Ingeniería en Tecnologías Industriales. Barcelona: Universitat Politècnica de Catalunya. Obtenido de <https://upcommons.upc.edu/entities/publication/e025d013-a1b9-4a3a-80a1-741afd70bec0>
- Xavier, A. (4 de 08 de 2008). *proyectos agiles*. Obtenido de Qué es SCRUM: <https://proyectosagiles.org/que-es-scrum/>

## 7. Anexos

**Tabla 5**

*Matriz de Endurecimiento del Servidor Debian para el Entorno MCP*

Categoría	Medida de Seguridad Implementada	Aplicación específica para el Entorno MCP
<b>Control de Red</b>	Configuración de Firewall	Se restringe el tráfico entrante, solo al servicio de FastAPI puerto 8080 está habilitada para no comprometer al servidor con servicios internos.
<b>Privilegios</b>	Principio de Menor Privilegio (PoLP)	El servidor MCP se ejecuta bajo un usuario sin privilegios de sudo, asegurando que, si el Agente de IA es comprometido, no tenga acceso al kernel del sistema.
<b>Integridad</b>	Aislamiento de Entornos (Virtualenv)	Se utiliza un entorno virtual de Python para que las dependencias del MCP no interfieran con las librerías críticas del sistema operativo Debian 13.

*Nota:* Matriz de Endurecimiento del Servidor Debian para el Entorno MCP. Autor: Ronald Rodriguez,2025

**Tabla 6**

*Planificación de iteraciones XP*

Iteración	Periodo	Objetivos de Desarrollo y Redacción	Entregables / Artefactos
<b>Iteración 1</b>	Noviembre	<p><b>Planificación y Análisis de Arquitectura:</b> Definición de Historias de Usuario iniciales y configuración del entorno de virtualización en Proxmox. Redacción de los Capítulos 1 y 2.</p>	<p>Historias de Usuario, Capítulos 1 y 2, Nodo Proxmox configurado.</p>
<b>Iteración 2</b>	Diciembre	<p><b>Diseño y Desarrollo:</b> Implementación del servidor MCP en Debian 13. Desarrollo del orquestador en FastAPI y conexión a la base de datos PostgreSQL con datos híbridos. Redacción de los Capítulos 3 y 4.</p>	<p>Código fuente main.py, Base de Datos, Capítulos 3 y 4.</p>
<b>Iteración 3</b>	Enero	<p><b>Pruebas y Lanzamiento:</b> Ejecución de pruebas de carga masiva y estrés. Optimización de la latencia a 766 ms. Análisis de la Tríada de Seguridad. Redacción del Capítulo 5, Conclusiones y Resumen.</p>	<p>Reportes de Proxmox, Capítulos Finales, Tesis Completa.</p>

*Nota:* Planificación de iteraciones XP. Autor: Ronald Rodriguez,2025