



UNIVERSIDAD TÉCNICA DE AMBATO

**FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E
INDUSTRIAL**

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Tema:

**CHATBOT BASADO EN PROCESAMIENTO DE LENGUAJE NATURAL
PARA TAREAS DE SOPORTE TÉCNICO EN LA EMPRESA MIVILSOFT S.A.**

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Tecnologías de la Información.

ÁREA: Integración de Sistemas, Tecnologías y Desarrollo Web, Investigación Operativa

LÍNEA DE INVESTIGACIÓN: Software, Tecnologías de la Información y Ciencia de Datos

AUTOR: David Augusto Ballesteros Moya

TUTOR: Ing. Clay Fernando Aldás Flores Mg.

Ambato - Ecuador

julio - 2026

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: CHATBOT BASADO EN PROCESAMIENTO DE LENGUAJE NATURAL PARA TAREAS DE SOPORTE TÉCNICO EN LA EMPRESA MIVILSOFT S.A., desarrollado bajo la modalidad Proyecto de Investigación por el señor David Augusto Ballesteros Moya, estudiante de la Carrera de Ingeniería en Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, julio 2026.



Ing. Clay Fernando Aldás Flores Mg.
TUTOR

AUTORÍA

El presente trabajo de titulación con el tema: CHATBOT BASADO EN PROCESAMIENTO DE LENGUAJE NATURAL PARA TAREAS DE SOPORTE TÉCNICO EN LA EMPRESA MIVILSOFT S.A. es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, julio 2026.



David Augusto Ballesteros Moya

CC. 1804303228

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, julio 2026.



David Augusto Ballesteros Moya

CC. 1804303228

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor David Augusto Ballesteros Moya, estudiante de la Carrera de Tecnologías de la Información de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado CHATBOT BASADO EN PROCESAMIENTO DE LENGUAJE NATURAL PARA TAREAS DE SOPORTE TÉCNICO EN LA EMPRESA MIVILSOFT S.A., nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con el señor Presidente del Tribunal.

Ambato, julio 2026.



Validar únicamente en FirmaSC.
Firmado electrónicamente por:
**FRANKLIN OSWALDO
MAYORGA MAYORGA**

Ing. Franklin Oswaldo Mayorga Mayorga, Mg.
PRESIDENTE DEL TRIBUNAL



Validar únicamente en FirmaSC.
Firmado electrónicamente por:
**RUBEN EDUARDO
NOGALES PORTERO**



**Santiago David Jara
Moya**



PhD. Rubén Eduardo Nogales Portero.
PROFESOR CALIFICADOR

Ing. Santiago David Jara Moya, MSc.
PROFESOR CALIFICADOR

DEDICATORIA

*Dedico el presente trabajo a mis padres, por ser mis
guías, haber confiado en mi, y darme amor y apoyo
incondicional*

*A mis familiares, que durante este proceso se han
preocupado por mi, y han seguido de cerca este
camino.*

*Y a mi hermosa novia, quien siempre fue compren-
siva, y en todo momento me dio ánimos y fuerzas
para no rendirme.*

David Augusto Ballesteros Moya

AGRADECIMIENTO

Al Ing. Clay Fernando Aldás Torres por su apoyo, paciencia, guía y confianza durante este proceso

Al personal de Mivilsoft S.A, por la oportunidad de realizara mi proyecto en su institución.

Y todos los docentes que formaron parte de mi formación profesional, quienes me han dado la oportunidad y las herramientas para nutrirme, emocionarme y crecer durante todos estos años de vida universitaria.

David Augusto Ballesteros Moya

ÍNDICE GENERAL DE CONTENIDOS

APROBACIÓN DEL TUTOR	ii
AUTORÍA	iii
DERECHOS DE AUTOR	iv
APROBACIÓN DEL TRIBUNAL DE GRADO	v
DEDICATORIA	vi
AGRADECIMIENTO	vii
ÍNDICE GENERAL DE CONTENIDOS	viii
ÍNDICE DE TABLAS	xii
ÍNDICE DE FIGURAS	xiii
ÍNDICE DE ANEXOS	xv
CAPÍTULO I. MARCO TEÓRICO	1
1.1 Tema de investigación	1
1.1.1 Planteamiento de problema	1
1.2 Antecedentes investigativos	2
1.3 Fundamentación teórica	6
1.3.1 Tecnologías de la información	6
1.3.2 Inteligencia artificial	6
1.3.3 Procesamiento de lenguaje natural	7
1.3.4 Chatbot	7
1.3.5 Soporte técnico empresarial	8
1.3.6 Gestión de incidencias	8
1.3.7 Volumen de tickets de soporte técnico	9
1.4 Objetivos	9
1.4.1 Objetivo general	9
1.4.2 Objetivos específicos	9
CAPÍTULO II. METODOLOGÍA	10
2.1 Materiales	10
2.2 Métodos	10
2.2.1 Modalidad de investigación	10
a. Investigación de campo	10
b. Investigación bibliográfica - documental	10
c. Modalidad especial	10
2.2.2 Población y muestra	11
2.2.3 Recolección de información	11
a. Fichas de observación	12
b. Entrevistas	13
c. Fichas documentales	15
2.2.4 Procesamiento y análisis de datos	17

CAPÍTULO III. RESULTADOS Y DISCUSIÓN	19
3.1 Análisis de necesidades	19
3.2 Análisis de requerimientos	21
3.3 Análisis de recursos	22
3.4 Modelos de lenguaje natural	23
3.4.1 Elección de modelos	23
a. Pruebas consideradas	24
b. Resultados	25
• τ^2 -Bench	25
• Terminal-Bench	26
• Ponderación final	26
c. Selección de modelo	28
3.5 Tecnologías de desarrollo de agentes	28
3.5.1 N8N	28
3.5.2 Vercel AI SDK	29
3.5.3 LangChain	29
3.5.4 Elección de tecnología: LangChain	29
3.6 Arquitecturas utilizadas	30
3.6.1 Multi-capa	30
3.6.2 Hexagonal	30
3.6.3 Modelo Vista Controlador (MVC)	30
3.6.4 Elección de arquitectura: Multi-capa	31
3.7 Buenas prácticas	31
3.7.1 Identificar el mejor uso para un modelo de lenguaje	31
3.7.2 Definir controles de seguridad	32
3.7.3 Precisión en la formulación de acciones, resultados y expectativas . . .	32
3.7.4 Migrar a sistemas multi-agente cuando la complejidad escale	32
3.8 Metodología de desarrollo	33
3.8.1 Scrum	33
3.8.2 Programación extrema	33
3.8.3 Scrumban	34
3.8.4 Tabla comparativa	34
3.8.5 Metodología aplicada	34
3.9 Diseño de la solución	35
3.9.1 Fase 1: Establecer objetivos	35
3.9.2 Fase 2: Backlog del proyecto	35
a. Actores	35
b. Módulos	36
3.9.3 Fase 3: Planificación de actividades	36
3.9.4 Fase 4: Desarrollo iterativo	37

a. Sprint 1	37
b. Flujo principal del agente	38
c. Manejo de errores	39
• Middleware de herramientas	39
• Reintentos y cambio de credenciales	40
d. Modelo entidad relación	41
e. Configuración de proyecto	42
• Módulo 1: agent	42
• Módulo 2: semantic_index	42
• Módulo 3: agent_helpdesk_tools	42
• Módulo 4: agent_rag	42
f. Sprint 2	43
• Sincronización	43
• Interfaz de usuario	46
g. Sprint 3	48
• Funcionalidades de los contactos	48
• Interfaz de usuario	50
h. Sprint 4	52
• Objetivos de la API	52
• Diseño	52
• Implementación	54
• Uso	56
i. Sprint 5	57
• Funcionalidades	57
• Interfaz de usuario	63
j. Sprint 6	66
• Funcionalidades	66
k. Sprint 7	68
• Funcionalidades	68
• Asignación de herramientas al agente	72
• Pruebas de uso de herramientas	73
l. Sprint 8	74
• Funcionalidades	75
• Interfaz de usuario	77
• Asignación de herramientas al agente	79
• Pruebas de uso de herramientas	80
3.9.5 Fase 5: Documentación	80
3.10 Resultados	80
CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES	82
4.1 Conclusiones	82

4.2 Recomendaciones	82
Bibliografía	84

ÍNDICE DE TABLAS

Tabla 1	Cadenas de texto y resultados de las bases de datos seleccionadas.	3
Tabla 2	Criterios de inclusión y exclusión.	3
Tabla 3	Desglose de la población definida.	11
Tabla 4	FO.01 - Canales de comunicación para comunicación de incidencias . . .	12
Tabla 5	FO.02 - Uso de soluciones existentes destinadas a gestión de incidencias .	12
Tabla 6	FO.03 - Gestión de prioridades	12
Tabla 7	FO.04 - Experiencia de usuario con el uso de Helpdesk	12
Tabla 8	FO.05 - Trazabilidad de gestión de incidencias	12
Tabla 9	FO.06 - Acceso a puntos de contacto	12
Tabla 10	Entrevista con el encargado de calidad de la empresa	13
Tabla 11	Entrevista con el jefe de TI de la empresa	14
Tabla 12	Ficha de contenido N°1	15
Tabla 13	Ficha de contenido N°2	15
Tabla 14	Ficha de contenido N°3	16
Tabla 15	Ficha de contenido N°4	16
Tabla 16	Ficha de contenido N°5	16
Tabla 17	HU.01 - Estandarización de recepción de solicitudes.	19
Tabla 18	HU.02 - Mejora de trazabilidad de incidencias.	19
Tabla 19	HU.03 - Adopción de sistemas existentes.	19
Tabla 20	HU.04 - Inversión de tiempo por parte del personal.	19
Tabla 21	HU.05 - Unificación de flujo de atención.	20
Tabla 22	HU.06 - Gestión de Prioridades.	20
Tabla 23	HU.07 - Comunicación con el cliente.	20
Tabla 24	HU.08 - Experiencia de usuario.	20
Tabla 25	Tabla de requerimientos funcionales.	21
Tabla 26	Tabla de requerimientos no funcionales.	21
Tabla 27	Tabla de recursos	22
Tabla 28	Conjunto de benchmarks utilizados en la evaluación.	24
Tabla 29	Resumen de dataset sujeto a análisis.	25
Tabla 30	Modelos más destacados según τ^2 -Bench	25
Tabla 31	Modelos más destacados según Terminal-Bench	26
Tabla 32	Tabla de modelos según la ponderación final asignada	27
Tabla 33	Tabla comparativa de agentes vs workflows.	31
Tabla 34	Tabla comparativa de metodologías de desarrollo.	34
Tabla 35	Actores del sistema.	35
Tabla 36	Módulos del sistema.	36
Tabla 37	Resumen cuantificable perteneciente al objetivo 1.	80

ÍNDICE DE FIGURAS

Figura 1	Proceso de selección de modelo de lenguaje natural.	24
Figura 2	Tablero Kanban elaborado para el seguimiento de las tareas	37
Figura 3	Diagrama BPMN del sistema.	38
Figura 4	Diagrama de funcionamiento de middleware de herramientas.	39
Figura 5	Diagrama de lógica de reintentos.	40
Figura 6	Diagrama entidad-relación del sistema.	41
Figura 7	Estructura del proyecto.	43
Figura 8	Funcionalidad de sincronización con dataset externo.	44
Figura 9	Creación o actualización de proveedores a partir de los datos de la API. .	44
Figura 10	Creación o actualización de modelos a partir de los datos de la API. . .	45
Figura 11	Creación o actualización de modalidades a partir de los datos de la API.	45
Figura 12	Sincronización diaria.	46
Figura 13	Listado de modelos de lenguaje natural.	46
Figura 14	Información de modelos de lenguaje natural.	47
Figura 15	Almacenamiento de credenciales.	47
Figura 16	Formulario de creación de credenciales.	48
Figura 17	Métodos sobrescritos al contacto.	49
Figura 18	Creación de agente a partir de contacto de Odoo.	50
Figura 19	Lista de agentes.	50
Figura 20	Parámetros generales e instrucciones de sistema.	51
Figura 21	Configuración de credenciales de agente.	51
Figura 22	Configuración de herramientas disponibles.	52
Figura 23	Flujo de registro de herramientas.	53
Figura 24	Modelo de herramienta.	54
Figura 25	Construcción de modelos de herramientas de LangChain.	55
Figura 26	Validación de herramientas.	56
Figura 27	Herramienta whoami	56
Figura 28	Declaración de herramientas.	57
Figura 29	Lista de herramientas disponibles.	57
Figura 30	Modificación del método message_post para introducir marcadores de sesión.	58
Figura 31	Modificación de función message_post_after_hook para gestionar las solicitudes a los agentes	59
Figura 32	Función llamada cuando el agente tiene una respuesta.	59
Figura 33	Creación de Agent Request desde el canal de mensajería.	60
Figura 34	Ejecución de Agent Request.	61
Figura 35	Ejecución de intento de Agent Request.	62

Figura 36	Ejemplo de bot de telegram	63
Figura 37	Visualización gráfica de solicitudes realizadas.	64
Figura 38	Visualización en lista de solicitudes realizadas.	64
Figura 39	Visualización en tabla de pivote de solicitudes realizadas.	65
Figura 40	Información de solicitud con el registro de los mensajes intermedios generados por el agente, así como llamadas a herramientas.	65
Figura 41	Información de la solicitud del registro con el conteo de tokens consumidos.	66
Figura 42	Definición del modelo abstracto.	66
Figura 43	Definición de métodos destinados a parametrizar la construcción del índice.	67
Figura 44	Modificación de los métodos de creación, modificación y eliminación. .	67
Figura 45	Definición del método utilizado para realizar una búsqueda semántica. .	68
Figura 46	Integración de ticket de soporte con búsqueda semántica.	69
Figura 47	Herramienta para búsqueda de ticket por número de ticket.	69
Figura 48	Herramienta para búsqueda de ticket por temática o fechas.	70
Figura 49	Herramienta para búsqueda de tickets repetidos.	71
Figura 50	Herramienta para la creación de tickets nuevos	71
Figura 51	Registro de herramientas para gestión de tickets.	72
Figura 52	Asignación de herramientas de gestión de tickets al agente.	72
Figura 53	Instrucciones del sistema sobre el uso de herramientas de gestión de tickets.	73
Figura 54	Pruebas del uso de herramientas de gestión de tickets.	73
Figura 55	Enlace de ticket generado reportado por el agente.	74
Figura 56	Ticket visible para los empleados de la empresa.	74
Figura 57	Declaración de modelo para documentos.	75
Figura 58	Modificación de métodos para poder leer PDFs.	76
Figura 59	Modificación al modelo de contacto para poder asociarlo con documentos.	76
Figura 60	Definición de herramientas para realizar una búsqueda por la documentación.	77
Figura 61	Declaración de herramientas.	77
Figura 62	Base de conocimientos.	78
Figura 63	Registro de documentos.	78
Figura 64	Configuración de conocimiento.	79
Figura 65	Configuración de conocimiento.	79
Figura 66	Asignación de herramientas de búsqueda de documentación al agente. .	79
Figura 67	Instrucciones del sistema para el uso de la herramienta de búsqueda de documentación.	80
Figura 68	Pruebas de las herramientas de búsqueda de documentación.	80

ÍNDICE DE ANEXOS

Listado 1	Manual de usuario	90
-----------	-------------------------	----

RESUMEN EJECUTIVO

En Mivilsoft S.A, la fragmentación de los puntos de contacto al cliente dificulta la trazabilidad y el seguimiento de inconveniencias. Esta situación ha obstaculiza el establecimiento de indicadores de calidad y la mejora de dichos procesos. Se han realizado intentos previos por centralizar el reporte y seguimiento de incidencias mediante un sistema de gestión tickets; sin embargo este no ha presentado aceptación por parte de los clientes, quienes continúan privilegiando la inmediatez y fluidez de los canales de mensajería.

El presente proyecto surge de la necesidad de contar con un sistema que preserve la fluidez del canal preferido por los clientes, a la vez que centralice la información recolectada permitiendo trazabilidad y la auditoría. Para esto se hace uso de tecnologías modernas basadas en inteligencia artificial.

El proyecto incluyó un estudio de necesidades de la empresa y los recursos disponibles para su aprovechamiento, a partir del cual se resolvió que la solución se implementará como un módulo de Odoo integrado al sistema actualmente en operación, y haciendo uso de tecnologías como LangChain y pgvector para la integración de capacidad de inteligencia artificial; y realizó una revisión bibliográfica de tecnologías, arquitecturas de software y buenas prácticas vinculadas con el uso de modelos de lenguaje natural.

El proyecto dio como resultado un sistema que mantiene una conversación con el usuario, y en base a sus necesidades y el contexto dado, toma decisiones como registrar un nuevo ticket se soporte, responder inquietudes del usuario, o consultar sobre tickets relacionados a una problemática.

Palabras Clave: Odoo, Chatbot, RAG, Agente, Scrumban, Inteligencia Artificial, Soporte al Cliente

ABSTRACT

In Mivilsoft S.A., the fragmentation of customer contact points hinders the traceability and follow-up of reported issues. This situation has obstructed the establishment of quality indicators and the improvement of related processes. Previous attempts to centralize incident reporting and monitoring through a ticket-management system have not been well received by clients, who continue to favor the immediacy and fluidity of messaging channels.

This project arises from the need for a system that preserves the fluidity of the communication channels preferred by clients while centralizing the collected information to enable traceability and auditing. To achieve this, the project leverages modern technologies based on artificial intelligence.

The project included an assessment of the company's needs and the resources available to address them. Based on this analysis, it was determined that the solution would be implemented as an Odoo module integrated into the system currently in operation, incorporating technologies such as LangChain and pgvector to enable artificial intelligence capabilities. Additionally, a literature review was conducted on technologies, software architectures, and best practices related to the use of natural language models.

The resulting system is capable of maintaining a conversational interaction with the user and, based on the user's needs and the given context, can make decisions such as registering a new support ticket, responding to user inquiries, or retrieving information about tickets related to a specific issue.

Keywords: Odoo, Chatbot, RAG, Agent, Scrumban, Artificial Intelligence, Customer Support

CAPÍTULO I. MARCO TEÓRICO

1.1 Tema de investigación

"CHATBOT BASADO EN PROCESAMIENTO DE LENGUAJE NATURAL PARA TAREAS DE SOPORTE TÉCNICO EN LA EMPRESA MIVILSOFT S.A."

1.1.1 Planteamiento de problema

El desarrollo acelerado y continuo de tecnologías emergentes es un factor determinante en la transformación de múltiples sectores productivos y de servicios. Dichos avances tecnológicos han favorecido la implementación de estrategias orientadas a la automatización y a la optimización de recursos, permitiendo a las organizaciones mejorar su rendimiento operativo. En este sentido, la digitalización de procesos no solo ha contribuido a la reducción de costos y tiempos, sino que también ha generado condiciones más favorables para incrementar la productividad, eficiencia y competitividad en el entorno empresarial contemporáneo. [1]

Particularmente, el área de la Inteligencia Artificial (IA) ha registrado avances significativos en los últimos años, con investigaciones que han derivado en el diseño y la validación de nuevos algoritmos y modelos computacionales; tecnologías como el Natural Language Processing (NLP) han potenciado la capacidad de los sistemas automatizados para comprender e imitar las distintas formas de comunicación humana. Como consecuencia, se observa una tendencia sostenida hacia la adopción de interfaces conversacionales en diversos sectores; siendo uno de los focos principales la atención al cliente, sector en donde se tiene una creciente necesidad de garantizar disponibilidad continua (24/7) y la capacidad de escalar servicios sin comprometer la calidad. [2]

Dentro de la región de latinoamérica se observan tendencias similares, evidenciado por el aumento en la cantidad de trabajo académico relacionado al potencial de la tecnología, así como sus implementaciones. Es importante recalcar que esta región enfrenta desafíos en cuanto a la implementación de esta tecnología, principalmente la falta de personal capacitado y los recursos económicos y de infraestructura. [3]

En Ecuador, el interés por la inteligencia artificial ha ido en aumento, y esto se refleja en el número creciente de procesos publicados en el portal de contratación pública que solicitan soluciones basadas en tecnologías como los chatbots y otras herramientas que utilizan IA generativa. Tanto entidades públicas como privadas están reconociendo el potencial de estas tecnologías para mejorar la atención al cliente y hacer más eficientes sus operaciones. En muchos casos, los chatbots son vistos como una forma efectiva

de ahorrar tiempo, tanto para los usuarios que buscan respuestas rápidas, como para los trabajadores que pueden enfocarse en tareas más complejas. Este tipo de requerimientos, cada vez más comunes, evidencian un cambio en la forma en que se conciben los servicios y la gestión institucional en el país.

Mivilsoft S.A es una empresa ambateña que ofrece servicios tecnológicos e innovaciones en el ámbito de Internet of Things (IoT), brindando soluciones a instituciones públicas y privadas en todo el país. Gracias a su enfoque en la transformación digital, la empresa ha experimentado un crecimiento sostenido en los últimos años, ampliando tanto su cartera de clientes como su equipo multidisciplinario, que abarca áreas como administración, atención al cliente, servicio técnico y desarrollo de software.

Este crecimiento ha generado una serie de desafíos operativos adicionales, particularmente en lo referente en los procesos de interacción con los usuarios y clientes. El incremento en el volumen de datos asociados a incidencias, consultas y retroalimentaciones de los clientes, junto con la diversidad de fuentes desde las cuales se origina dicha información, ha incrementado la complejidad de las labores de seguimiento y control de calidad en los servicios de atención al cliente.

La diversidad de canales de comunicación utilizados actualmente por la empresa, que incluyen WhatsApp, correo electrónico y llamadas directas, ha derivado en una fragmentación de la información que dificulta la centralización, el seguimiento histórico de los casos y la evaluación del desempeño del equipo de soporte. Esta ausencia de un sistema unificado para administrar de manera estructurada las incidencias reportadas obstaculiza tanto el control operativo como la obtención de métricas confiables sobre la calidad del servicio y los tiempos de resolución. Si no se implementa el proyecto, la empresa continuará enfrentando limitaciones para escalar sus operaciones de atención al cliente, lo cual podría afectar la percepción de calidad del servicio y generar sobrecarga en el personal de soporte técnico.

1.2 Antecedentes investigativos

Se dirigió una revisión bibliográfica relacionada al uso de la inteligencia artificial en chatbots y como su uso en soporte al cliente. Para ello, se emplearon cadenas de búsqueda construidas a partir de palabras clave como *“chatbot”*, *“AI”*, *“RAG”*, *“Agent”* y *“Customer support”*, las cuales fueron aplicadas en las bases de datos académicas Scopus e IEEE Xplore. Los resultados se los explora en la Tabla 1.

Para la selección de los artículos relevantes para el presente proyecto de investigación, se aplicó criterios de inclusión y exclusión detallados en la Tabla 2.

Además de las fuentes identificadas en bases de datos científicas, se amplió el alcance de la búsqueda mediante la revisión de los repositorios institucionales de la Universidad Técnica de Ambato (UTA) y la Escuela Superior Politécnica de Chimborazo (ESPOCH), aplicando como criterio de selección la pertinencia temática relacionada con el uso de chatbots. Como resultado, se identificaron 19 trabajos previos en el repositorio de la UTA y 93 en el de la ESPOCH.

Tabla 1: Cadenas de texto y resultados de las bases de datos seleccionadas.

Base de datos	Cadena de búsqueda	Número de artículos
Scopus	(TITLE-ABS-KEY (ai) AND TITLE-ABS-KEY (chatbot) AND TITLE-ABS-KEY (rag)) AND PUBYEAR > 2021 AND PUBYEAR < 2025 AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "MULT") OR LIMIT-TO (SUBJAREA , "ENGI"))	55
	(TITLE-ABS-KEY (ai) AND TITLE-ABS-KEY (agent) AND TITLE-ABS-KEY (customer AND support)) AND PUBYEAR > 2021 AND PUBYEAR < 2025 AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "MULT") OR LIMIT-TO (SUBJAREA , "ENGI"))	58
	(TITLE-ABS-KEY (ai) AND TITLE-ABS-KEY (rag)) AND PUBYEAR > 2021 AND PUBYEAR < 2025 AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "MULT") OR LIMIT-TO (SUBJAREA , "ENGI"))	301
IEEE Xplore	("All Metadata":AI) AND ("All Metadata":chatbot) AND ("All Metadata":RAG) \ Filters Applied: 2021 - 2025	76
	("All Metadata":AI) AND ("All Metadata":RAG) AND ("All Metadata":Customer Support) \ Filters Applied: 2021 - 2025	14
Total		504

Tabla 2: Criterios de inclusión y exclusión.

Criterios de inclusión	Criterios de exclusión
Artículos de investigación de acceso libre	Artículos cuya fecha de publicación data antes del 2021
Publicaciones que aborden la implementación o análisis teórico de chatbots	Documentos sin acceso libre o que requieran suscripción o pago para su consulta completa.
Trabajos que presenten soluciones aplicadas al acceso a conocimiento específico o atención al cliente, utilizando distintas tecnologías de desarrollo e integración.	Publicaciones que se enfoquen en áreas no relacionadas con el acceso a información específica, atención al cliente o sistemas conversacionales, como entretenimiento, educación infantil, o redes sociales.

El proyecto de titulación de C. H. Llerena Valencia [4], se presenta un prototipo de chatbot para un sistema de servicios vehiculares utilizando «*Bot Framework*», una serie de herramientas, servicios y SDKs que habilitan la creación de bots impulsados por IA, y «*Adaptive Cards*», un framework para construir componentes gráficos que pueden ser utilizados por un chatbot; ambas tecnologías desarrolladas por Microsoft. El trabajo concluye que los chatbots presentan sus limitaciones, y recomienda analizar los flujos presentes en la empresa, así como el uso de la técnica de Diálogo en Cascada para organizar y estructurar los diálogos de forma estructurada y el uso de una metodología ágil para llevar un control óptimo del prototipo en desarrollo.

Por su parte, R. D. Bonilla Guevara [5] presenta en su trabajo de investigación un prototipo de chatbot aplicado a la resolución y atención de inquietudes académicas de la Secretaría de Ingeniería en Sistemas Computacionales e Informáticos de la UTA. Como parte del proceso metodológico, aplica una encuesta dirigida al personal administrativo de la secretaría y a los estudiantes de la facultad, con el objetivo de identificar necesidades y evaluar la aceptación de este tipo de soluciones. Los resultados reflejaron una percepción positiva respecto al uso de chatbots para agilizar trámites relacionados con certificaciones y documentación. El desarrollo utiliza «*IBM Watson Assistant*» debido a la experiencia de la empresa que desarrolla esta tecnología y su naturaleza «*no-code*». El estudio concluye con un prototipo funcional, subrayando la importancia del entrenamiento continuo del modelo para optimizar la experiencia del usuario y la necesidad de realizar pruebas para comprobar la correcta implementación de la solución.

M. V. Barros Manzano [6], presenta un trabajo en donde se implementa un asistente conversacional basado en un sistema de gestión de conocimientos para atención a los clientes de la Empresa Aceros Industriales HGB. El trabajo realiza una encuesta determinando los temas más consultados por los usuarios, y utiliza «*Dialogflow*», una herramienta desarrollada por Google que utiliza NLP para detectar la intención de los usuarios. El resultado presentado habilitó a la empresa fortalecer los lazos con sus clientes, mediante una disminución en el tiempo de respuesta a las inquietudes presentadas; recomienda la inclusión de más canales de mensajería habilitados por facilidades que brinda «*Dialogflow*», realizar revisiones periódicas al historial de chats para determinar cuáles son los intereses más relevantes de los usuarios, y mantener una investigación continua sobre chatbots y su aplicación.

G. F. Reyes Pullugando [7] implementó un chatbot para la atención y resolución de inquietudes turísticas en la ciudad de Baños. El trabajo explora las soluciones mencionadas anteriormente además de «*Wit.ai*», un framework open source para el desarrollo de bots adquirido por Meta; «*Botpress*», una plataforma desarrollada para crear bots desarrollada por la empresa homónima, y posee integraciones con varios

servicios de mensajería; y «*RASA*», una tecnología desarrollada por RASAHQ que permite la creación de bots con la capacidad de procesar texto y voz utilizando el lenguaje de programación Python. Reyes Pullugando concluye su implementación con una encuesta, obteniendo resultados de 87% de satisfacción con respecto al chatbot; y recomienda mantener las fuentes de información actualizadas con la finalidad de brindar una buena experiencia de usuario, así como la expansión del conocimiento del chatbot y la integración con otros métodos de entrada mediante el reconocimiento de voz.

El trabajo de integración curricular presentado por B. S. Guaylla Ashqui y J. S. Burgos Meléndrez [8] desarrolla un chatbot para la farmacia San Vicente del cantón Chambo con integración con servicios de mensajería y un portal web. El trabajo implementa la solución utilizando «*Dialogflow*», y explora otras alternativas para la implementación del chatbot, entre ellas «*Dialogflow*», «*Watson*» y «*Amazon Lex*», siendo la última una tecnología desarrollada por Amazon, cumpliendo el mismo rol que las tecnologías anteriormente mencionadas. Además realiza una encuesta con la finalidad de calificar la usabilidad y experiencia de usuario del chatbot, presentado resultados satisfactorios con un promedio general de usabilidad del 87.6% y optimizando los procesos de atención al cliente.

M. A. Paucar Toctaguano y C. J. Romero Chicaiza [9] en su trabajo presentan un chatbot orientado a la atención de tramites de Carrera de Software de la ESPOCH, un enfoque similar al trabajo de R. D. Bonilla Guevara [5]. El trabajo realizó pruebas basadas en ISO/IEC 25010 e ISO/IEC 25023 para medir los tiempos de respuesta y el rendimiento antes y después de la implementación del chatbot, los resultados obtenidos fueron favorables para el chatbot, teniendo una reducción del 29.4% en el tiempo de respuesta, 38.84% en el tiempo de espera, y un aumento del 133.1% en el rendimiento de la atención a los estudiantes. El modelo de lenguaje fue desarrollado por los mismos autores, quienes utilizaron redes neuronales y técnicas de aprendizaje continuo para el entrenamiento.

E. O. López Galarza [10] realizó un trabajo de titulación en el cual presenta una solución para reducir los tiempos de atención a clientes en la empresa Ingedemy utilizando la IA. En su trabajo, explora modelos de tipo GPT como una de las opciones disponibles para la implementación. El proyecto llevó a una transformación en la gestión de atención al cliente, minimizando los tiempos de respuesta; además, mediante su implementación de una base de datos Not Only Structural Query Language (NoSQL), almacena el historial de conversaciones, brindando así analíticas sobre las interacciones que han tenido los clientes con la empresa. La encuesta que el investigador llevó a cabo

una vez culminada la implementación demostró que el proyecto efectuado ha generado un impacto positivo en la satisfacción de los clientes de la empresa.

La investigación de H. Soudani, E. Kanoulas, y F. Hasibi [11], compara modelos de Retrieval Augmented Generation (RAG) con modelos ajustados a las necesidades de negocio en términos de rendimiento y costo. El resultado de su investigación resalta que un modelo relativamente pequeño, ajustado y con técnicas adecuadas de RAG, puede igualar el rendimiento o incluso superar modelos de NLP más grandes; y se explora técnicas de entrenamiento y ajustes que permitan optimizar costos. Estos resultados ayudan a determinar la solución más eficiente en la elección de modelos de NLP y su implementación.

1.3 Fundamentación teórica

1.3.1 Tecnologías de la información

Según M. A. Taleb y A. Rashid [12], el término «Tecnologías de la información» (TI) refiere a todos los procesos relacionados a la creación, recolección, procesamiento, almacenamiento y diseminación de la información. Las TI constituyen una disciplina fundamental que abarca el estudio, desarrollo, implementación y gestión de sistemas digitales diseñados para procesar y comunicar información. Su alcance incluye múltiples áreas como la informática, la ingeniería de software, los sistemas de información y las telecomunicaciones, lo que le otorga un carácter trans-disciplinario consolidado en el ámbito académico [13]. Más allá de su dimensión técnica, las TI también integran aspectos normativos y organizacionales, permitiendo comprender cómo operan en marcos sociales, regulatorios y políticos complejos [14].

En términos prácticos, su relevancia se manifiesta en sectores como la educación, donde las tecnologías digitales han transformado el acceso, la gestión y la diseminación del conocimiento a escala global [15]. Así, las TI se posicionan como una infraestructura clave que influye en la forma en que las personas se comunican, aprenden, trabajan y toman decisiones cotidianas.

1.3.2 Inteligencia artificial

La inteligencia artificial es la rama informática dedicada a diseñar sistemas capaces realizar tareas que normalmente requieren inteligencia humana, como la percepción del entorno, el razonamiento, el aprendizaje y la toma de decisiones autónoma o semi-autónoma en contextos complejos [16]. Las definiciones recientes también destacan su dimensión regulatoria y socio-técnica, tal como lo plantea el Grupo de Expertos de

Alto Nivel de la UE: “sistemas que muestran comportamientos inteligentes analizando el entorno y tomando acciones con cierto grado de autonomía para alcanzar objetivos específicos” [17]. En años recientes, la IA ha sido reconocida no solo por su dimensión técnica, sino también por su impacto regulatorio, ético y legal, lo cual ha dado lugar a debates sobre su definición institucional y normativa en diferentes regiones del mundo [18].

Además, investigaciones actuales han enfatizado el papel de la IA como motor de transformación en sectores como la salud, la educación, las finanzas y la administración pública, destacando su influencia estructural sobre prácticas cotidianas y decisiones estratégicas [15].

1.3.3 Procesamiento de lenguaje natural

La Procesamiento de lenguaje natural es un campo inter-disciplinario entre la inteligencia artificial, la lingüística computacional y la informática, enfocado en permitir que las máquinas entiendan, procesen y generen lenguaje humano de manera efectiva. Este ámbito incluye tanto el análisis estadístico y semántico de texto como la generación natural de lenguaje, abarcando tareas como traducción automática, clasificación, extracción de información y diálogo conversacional [19]. Gracias a los avances en aprendizaje profundo, las capacidades del NLP han dado un salto significativo en calidad y precisión frente a enfoques previos basados en reglas o bag-of-words [20]. En particular, el NLP se distingue por manejar grandes volúmenes de datos textuales no estructurados, donde aplica técnicas de embebido, atención y generación que han impulsado aplicaciones cotidianas como asistentes virtuales, sistemas de recomendación, análisis médico y gestión de documentos institucionales [21]. Además, investigaciones recientes en interpretabilidad han enfatizado la importancia de comprender internamente los modelos NLP, con el fin de garantizar su confiabilidad, precisión y alineación ética [22].

1.3.4 Chatbot

E. Adamopoulou y L. Moussiades [23] define a la tecnología chatbot se refiere a sistemas de conversación automatizada que interactúan con usuarios mediante texto o voz, desde interfaces sencillas hasta agentes conversacionales basados en inteligencia artificial. Aunque los primeros desarrollos como ELIZA datan de 1966, los chatbots modernos se apoyan en enfoques de aprendizaje automático y procesamiento de lenguaje natural para comprender intenciones, gestionar el contexto y generar respuestas fluidas [24]. Revisión sistemática recientes han identificado enfoques avanzados que combinan NLP, machine learning, minería de datos y comprensión semántica

contextual para mejorar la calidad y relevancia de las respuestas [25]. Además, la popularización de asistentes virtuales como Siri, Alexa y los chatbots basados en grandes modelos de lenguaje ha impulsado su uso en sectores como salud, atención al cliente y educación, donde se destacan por su capacidad para automatizar tareas repetitivas y brindar soporte personalizado [26]. Sin embargo, también se han evidenciado desafíos clave relacionados con el sesgo, la transparencia, la confianza del usuario y el diseño conversacional centrado en la experiencia, que demandan metodologías más éticas y user-centric en su desarrollo.

1.3.5 Soporte técnico empresarial

El soporte técnico empresarial es una función esencial en las organizaciones que asegura la continuidad operativa, la disponibilidad de los sistemas y la productividad del usuario final. Además de resolver incidentes, incorpora prácticas de gestión proactiva, monitoreo de infraestructura y autoservicio, alineado con las estrategias de continuidad del negocio y resiliencia de TI [27]. En la era de la digitalización, estos servicios han evolucionado hacia modelos más inteligentes y automatizados, apoyados por plataformas en la nube y soluciones de recuperación ante desastres que garantizan tiempos de respuesta rápidos y mínima interrupción [28]. Un enfoque centrado en el usuario también se ha vuelto crítico, donde la eficiencia del soporte técnico se mide a través de indicadores como tiempo de resolución, satisfacción y reducción de tickets recurrentes [29]. En consecuencia, la modernización del soporte empresarial, sustentada en metodologías de continuidad y herramientas tecnológicas avanzadas, se traduce en operaciones más robustas, ágiles y orientadas al servicio [30].

1.3.6 Gestión de incidencias

La gestión de incidencias es un proceso estructurado y continuo que abarca desde la detección y registro de la incidencia hasta su resolución y cierre, con el objetivo de restablecer el servicio y minimizar el impacto en la operación empresarial [28]. Este proceso incluye clasificación, escalamiento, diagnóstico, resolución y retroalimentación en una base de conocimiento, promoviendo la eficiencia del soporte y reduciendo los tiempos de atención [31]. Investigaciones recientes basadas en marcos como Information Technology Infrastructure Library (ITIL) y plataformas de Business Intelligence (BI) han demostrado mejoras sustanciales en métricas clave: reducción del tiempo medio de resolución (de más de 2800 minutos a alrededor de 1100 minutos) y disminución significativa de tickets pendientes o backlog [32]. Así, optimizar la gestión de incidencias contribuye directamente a la reducción de carga operativa en los equipos de soporte técnico y a una mayor calidad del servicio percibido por los usuarios.

1.3.7 Volumen de tickets de soporte técnico

El volumen de tickets de soporte técnico es una métrica operacional clave que mide la cantidad de solicitudes gestionadas por el equipo de soporte en un periodo determinado. Un mayor volumen refleja tanto la demanda del usuario como la carga del servicio, y es esencial para dimensionar los recursos humanos y tecnológicos disponibles. Estudios recientes han evidenciado que un volumen elevado de tickets, sumado a criterios como la criticidad y automatización, puede impactar negativamente en la productividad del equipo si no se gestiona adecuadamente, sugiriendo la implementación de dashboards y gestión de conocimiento para mejorar el flujo de trabajo [33]. También se ha señalado que analizar las fuentes de origen del ticketing permite identificar tendencias emergentes y anticipar eventos críticos antes de que escalen, optimizando así la planificación y la respuesta proactiva.

1.4 Objetivos

1.4.1 Objetivo general

Implantar un chatbot basado en procesamiento de lenguaje natural para tareas de soporte técnico en la empresa Mivilsoft S.A.

1.4.2 Objetivos específicos

- Analizar las necesidades, requerimientos y recursos de la empresa disponibles para el desarrollo de la propuesta.
- Indagar diferentes opciones disponibles en modelos de lenguaje natural, arquitecturas más utilizadas, y buenas prácticas recomendadas al trabajar con esta tecnología.
- Desarrollar un chatbot basado en procesamiento de lenguaje natural para las tareas de soporte técnico en la empresa Mivilsoft S.A.

CAPÍTULO II. METODOLOGÍA

2.1 Materiales

Para la recolección de información del proyecto se emplearon fichas de observación que permitieron registrar de manera sistemática los procesos operativos de la empresa, así como entrevistas dirigidas al encargado de calidad y al jefe de TI, con el propósito de caracterizar el estado actual de los servicios, identificar las limitaciones existentes y establecer los requerimientos funcionales para la solución propuesta. Las fichas bibliográficas se basan en los antecedentes investigativos presentados en la Sección 1.2, de los cuales se seleccionaron los más relevantes para fundamentar el desarrollo del proyecto.

2.2 Métodos

2.2.1 Modalidad de investigación

El proyecto se desarrollará utilizando metodologías de investigación que brindan un marco estructurado para alcanzar los objetivos planteados. Se han elegido las siguientes investigaciones para este fin:

a. Investigación de campo

El estudio se desarrolla bajo una modalidad de investigación de campo, sustentada en la observación directa y entrevistas con el personal involucrado. Este enfoque resulta fundamental para identificar las necesidades de la empresa y detectar áreas de mejora, permitiendo enfocar la investigación en los problemas reales de la organización y potenciar el impacto de la solución propuesta.

b. Investigación bibliográfica - documental

La investigación también adopta una modalidad bibliográfica-documental, ya que implica la recopilación y análisis de soluciones, alternativas y herramientas previamente utilizadas —o potencialmente aplicables— en situaciones similares a la planteada en este estudio. Este enfoque busca ampliar el conocimiento sobre los recursos disponibles y proporcionar un marco de referencia sólido.

c. Modalidad especial

Este trabajo incorpora una modalidad especial al contemplar el desarrollo e integración de una solución tecnológica: un chatbot basado en procesamiento de lenguaje natural

orientado a tareas de soporte técnico. Esta solución tiene como objetivo abordar directamente el problema identificado, mediante la aplicación de tecnologías modernas y emergentes.

2.2.2 Población y muestra

La población objeto de estudio está conformada por el personal que labora en la empresa Mivilsoft S.A., lugar en el cual se planea implementar el chatbot como parte del proyecto. Dado que la cantidad total de personas dentro de la organización es menor a cien, no se considera necesario aplicar un muestreo estadístico representativo. En lugar de ello, se optará por realizar un análisis exhaustivo que abarque a toda la población disponible, lo cual permitirá obtener una visión integral y precisa sobre las necesidades, expectativas y percepciones del equipo respecto a la solución tecnológica propuesta.

La población está compuesta por personal de distintas áreas funcionales, incluyendo soporte técnico, desarrollo de software, atención al cliente y administración. Esta diversidad permite capturar una amplia gama de perspectivas y necesidades frente a la posible adopción del chatbot, en la Tabla 3 se presenta un desglose detallado de la composición de esta población, que servirá como base para la aplicación de las encuestas correspondientes.

Tabla 3: Desglose de la población definida.

Sector	Población	
Gerencia General	1	5.88%
Personal Técnico	11	64.71%
Personal de Soporte	5	29.41%
Total	17	100%

2.2.3 Recolección de información

Para el proceso de recopilación de información se utilizó la observación directa como técnica principal. A partir de las fichas diseñadas para este propósito, se formularon dos entrevistas: una dirigida al encargado de control de calidad y otra al jefe de TI, con el objetivo de profundizar en aspectos relacionados con la implementación del sistema, así como identificar requerimientos y expectativas asociadas al proyecto.

a. Fichas de observación

A continuación, se presentan las fichas de observación obtenidas durante dicho proceso.

Tabla 4: FO.01 - Canales de comunicación para comunicación de incidencias

Ficha de Observación	
Código	F0.01
Observación	Los clientes contactan a la empresa por diversos canales (WhatsApp, correo electrónico, llamadas telefónicas e incluso mensajes directos a empleados)

Tabla 5: FO.02 - Uso de soluciones existentes destinadas a gestión de incidencias

Ficha de Observación	
Código	F0.02
Observación	Se observa que en la plataforma Odoo de la empresa existe un módulo dedicado a la gestión y seguimiento de incidencias. Los tickets son generados manualmente por el personal interno, y no existe una adopción por todos los clientes

Tabla 6: FO.03 - Gestión de prioridades

Ficha de Observación	
Código	F0.03
Observación	La falta de un método uniforme de registro hace que la urgencia de los casos se evalúe de manera subjetiva, generando inconsistencias en los tiempos de atención.

Tabla 7: FO.04 - Experiencia de usuario con el uso de Helpdesk

Ficha de Observación	
Código	F0.04
Observación	La falta de familiaridad con el portal de tickets hace que los clientes posterguen el registro formal o notifiquen dichos registros al personal informalmente.

Tabla 8: FO.05 - Trazabilidad de gestión de incidencias

Ficha de Observación	
Código	F0.05
Observación	El seguimiento de incidentes se realiza de manera reactiva y basada en conversaciones personales, lo que dificulta analizar el estado de los casos y medir tiempos de respuesta.

Tabla 9: FO.06 - Acceso a puntos de contacto

Ficha de Observación	
Código	F0.06
Observación	Se identificó que los clientes buscan canales rápidos e inmediatos, evitando formularios o portales formales. Muchos recurren al teléfono o a mensajes instantáneos aún fuera del horario laboral.

b. Entrevistas

La Tabla 10 recoge la entrevista realizada al encargado de control de calidad.

Tabla 10: Entrevista con el encargado de calidad de la empresa

Código	Pregunta	Respuesta	Observación
EN.01	¿A través de qué canales se reciben actualmente las solicitudes o quejas por parte de los clientes?	Actualmente nuestros clientes se comunican con nosotros por medio de distintos canales, principalmente mediante Whatsapp, correo electrónico, o llamadas directamente al personal encargado para soporte técnico	Se evidencia que la comunicación entre los clientes y el personal técnico se realiza mediante múltiples canales independientes, lo que refleja una dispersión en los medios utilizados.
EN.02	¿Podría describir el flujo completo de atención, desde la recepción de una solicitud hasta su cierre?	El flujo comienza con un mensaje o llamada al personal, quien recibe el mensaje comúnmente es el encargado de solucionar el inconveniente surgido, si se requiere delegar a alguien más, comúnmente se hace por medio de comunicación directa entre los empleados. Durante este proceso, el empleado quien recibió el mensaje es quien mantiene la comunicación con el cliente, al menos que el personal al que se fue delegada la tarea tenga el contacto del cliente, o se comparta el mismo.	El proceso funciona adecuadamente cuando un solo colaborador gestiona la solicitud; sin embargo, la delegación presenta debilidades, ya que no existe trazabilidad formal del traspaso de la responsabilidad. Esto podría generar malentendidos y falta de claridad sobre la comunicación con el cliente, la cual suele permanecer en manos del empleado que recibió el requerimiento inicialmente.
EN.03	¿Se presentan casos en los que una solicitud llega al área incorrecta? En caso afirmativo, ¿podría proporcionar un ejemplo?	Si, pero no es frecuente que suceda. Hemos casos en donde al personal de TI le llegan consultas más adecuadas para personal de otras áreas, y viceversa	Se identifican situaciones en las que el personal debe redirigir solicitudes hacia otras áreas, lo que implica procesos de delegación. Además, se observa que parte de la co-

Código	Pregunta	Respuesta	Observación
			municación recibida no corresponde únicamente a problemas técnicos, sino también a consultas generales.
EN.04	¿Cuáles considera que son los principales inconvenientes del proceso actual de gestión de solicitudes?	No tenemos un medio centralizado por el cual tener un control de inconvenientes surgidos a lo largo del tiempo, por lo que dificulta el seguimiento a los mismos y la evaluación de desempeño del personal	
EN.05	¿Cómo se realiza actualmente el seguimiento interno de cada solicitud?	Actualmente son los empleados encargados quienes tienen que reportar al personal de calidad en una minuta	El seguimiento depende enteramente del colaborador responsable, sin apoyo de mecanismos automatizados. Esto expone el proceso a inconsistencias derivadas de fallos humanos y limita el control centralizado sobre la gestión de incidencias.
EN.06	¿Consideran que el tiempo de respuesta se ve afectado por el flujo actual de trabajo? En caso afirmativo, ¿de qué manera?	Si, generalmente el personal se tiene que encargar en más tareas a lo largo del día. Y en ocasiones no puede dar una respuesta inmediata debido a su carga laboral	La carga laboral del personal impacta negativamente en la capacidad de brindar respuestas oportunas, lo que repercute en la satisfacción del cliente.

Finalmente, la Tabla 11 presenta la entrevista aplicada al jefe de TI.

Tabla 11: Entrevista con el jefe de TI de la empresa

Código	Pregunta	Respuesta	Observación
EN.07	¿Cómo se integran actualmente las solicitudes de los clientes con las herramientas internas, como Helpdesk, Odoo u otros sistemas?	El sistema Helpdesk es uno que hemos puesto a disposición para los clientes, pero observamos que son pocos quienes hacen uso del mismo.	A pesar de disponer de un sistema de gestión de tickets, los clientes muestran preferencia por la comunicación directa con el personal, lo cual limita el uso de la herramienta formal.
EN.08	¿En qué etapas del proceso consideran que la automatización podría aportar un mayor beneficio?	Principalmente para la creación, debido a que de momento esa es la razón principal	

Código	Pregunta	Respuesta	Observación
		pal por la cual los clientes no usan dicho sistema	
EN.09	¿Qué canal consideran más adecuado como punto inicial para automatizar la recepción de solicitudes?	Dentro del mismo sistema Odoo. Mediante el módulo Livechat es posible habilitar un canal de comunicación disponible en la página de la empresa	
EN.10	¿Qué consideraciones técnicas deben tenerse en cuenta para integrar un sistema automatizado con Helpdesk u Odoo?	Principalmente considerar que se tiene una sola instancia para múltiples compañías, el sistema de automatización debería saber en qué empresa crear los tickets de soporte.	
EN.11	¿Qué información consideran indispensable para la correcta gestión de un ticket de soporte?	Nombre, descripción, fecha de apertura de ticket y los datos del solicitante serían los que necesitaría. A partir de ahí se podría tomar como un adicional la categorización del ticket apenas se cree	

c. Fichas documentales

A continuación, se presentan las fichas documentales realizadas durante dicho proceso.

Tabla 12: Ficha de contenido N°1

Ficha de contenido N°1	
Tema	Prototipo de un chatbot para un sistema de servicios vehiculares utilizando Adaptive Cards con Bot Framework
Año de publicación	2021
Ideas centrales	Uso de procesamiento de lenguaje natural en conjunto con arquitecturas de software para el desarrollo de una solución capaz de reconocer distintas solicitudes de usuarios.
Conceptos clave	Procesamiento de lenguaje natural, Inteligencia Artificial, Flujo de conversación
Aporte al tema	Presenta una solución mediante el uso de la tecnología Bot Framework, y la aplicación del lenguaje natural destinado a identificar escenarios y definir acciones de acuerdo a estos

Tabla 13: Ficha de contenido N°2

Ficha de contenido N°2	
Tema	Prototipo de chatbot para la resolución y atención de inquietudes académicas de la Secretaría de Ingeniería en Sistemas Computacionales e Informáticos
Año de publicación	2021
Ideas centrales	Aplicación de la IA para interpretar consultas de usuarios y determinar acciones a tomar en base a la solicitud generada.
Conceptos clave	Inteligencia artificial, Chatbot, Procesamiento de lenguaje natural
Aporte al tema	Presenta una alternativa de chatbot que interactúa directamente con el usuario, haciendo uso de información previamente provista al sistema para resolver inquietudes y consultas.

Tabla 14: Ficha de contenido N°3

Ficha de contenido N°3	
Tema	Asistente conversacional (chatbot) basado en un sistema de gestión de conocimientos para atención a los clientes de la empresa Aceros Industriales HGB
Año de publicación	2022
Ideas centrales	Uso de dialogflow en conjunto con webhooks con la finalidad de integrar dicho agente con sistemas externos.
Conceptos clave	Procesamiento de lenguaje natural, Agentes conversacionales
Aporte al tema	Introduce el concepto de agentes conversacionales, capaces de interactuar con sistemas con la finalidad de atender a la solicitud del usuario

Tabla 15: Ficha de contenido N°4

Ficha de contenido N°4	
Tema	Implementación de un chatbot para la atención y resolución de inquietudes turísticas en la ciudad de Baños de Agua Santa
Año de publicación	2023
Ideas centrales	Uso de RASA para determinar consultas frecuentes de usuarios y proveer información mediante la personalización de su interfaz gráfica de usuario.
Conceptos clave	Inteligencia artificial, Procesamiento de lenguaje natural
Aporte al tema	

Tabla 16: Ficha de contenido N°5

Ficha de contenido N°5	
Tema	Solución de IA para reducir los tiempos de atención a clientes en la empresa Ingedemy
Año de publicación	2024
Ideas centrales	Desarrollo de chatbot basado en los modelos de IA de OpenAI, junto con una serie de integraciones que habilitan a este la obtención de información de forma dinámica.

Ficha de contenido N°5	
Conceptos clave	Inteligencia artificial, Modelos conversacionales, Procesamiento de lenguaje natural, GPT
Aporte al tema	Plantea la presencia de herramientas utilizadas por el chatbot destinadas a la consulta de información dinámica mediante el análisis de sitios web de manera programática.

2.2.4 Procesamiento y análisis de datos

A partir de las respuestas obtenidas de las preguntas EN.01, EN.02 y EN.04 (Tabla 10) y la ficha de observación F0.01 (Tabla 4), se evidencia que el proceso actual de atención y gestión de solicitudes técnicas presenta una alta dispersión en los canales de comunicación, los cuales incluyen WhatsApp, correo electrónico y llamadas directas. Esta diversidad de medios ha derivado en una fragmentación de la información, lo que dificulta la centralización, el seguimiento histórico de los casos y la evaluación del desempeño del equipo de soporte.

La problemática principal identificada radica en la ausencia de un sistema unificado que permita administrar de manera estructurada las incidencias reportadas por los clientes. Esta falta de integración obstaculiza tanto el control operativo como la obtención de métricas confiables sobre la calidad del servicio y los tiempos de resolución, elementos fundamentales para los procesos de mejora continua y la toma de decisiones basadas en datos.

Con relación al sistema de gestión de tickets, según la respuesta a la pregunta EN.07 de la Tabla 11 y la ficha de observación F0.02 (Tabla 5), la empresa ha determinado la adopción del módulo Helpdesk de Odoo, plataforma que constituye la base de los servicios actualmente ofrecidos a sus clientes. Esta elección responde a una estrategia de optimización de recursos tecnológicos, al aprovechar la infraestructura existente y mantener la coherencia con el ecosistema digital corporativo. Sin embargo, pese a las ventajas de esta integración, se identifica que el sistema actual no cubre las necesidades de automatización requeridas para la atención eficiente de solicitudes; observación respaldada por las fichas de observación F0.03 (Tabla 6), F0.04 (Tabla 7) y F0.06 (Tabla 9). Por ello, se propone el desarrollo de una solución complementaria basada en procesamiento de lenguaje natural, la cual permitirá automatizar la creación de tickets y optimizar la interacción con los usuarios.

Respecto a los canales de implementación inicial, según las respuestas a las preguntas EN.09 y EN.10 (Tabla 11), se considera oportuno integrar el chatbot directamente en la plataforma web de la empresa, con el propósito de ofrecer una experiencia nativa

y coherente para los usuarios. Esta estrategia permitirá evaluar la eficacia del sistema en un entorno controlado antes de su futura extensión a otros medios de comunicación corporativa.

En cuanto a la información necesaria para la creación de tickets válidos, se establece que el chatbot deberá recopilar, datos esenciales como la descripción del inconveniente, un resumen del caso y la identificación del usuario que realiza la solicitud, decisión respaldada por la respuesta a la pregunta EN. 11 en la Tabla 11. Estos elementos resultan indispensables para garantizar la trazabilidad del proceso y la integridad del registro dentro del sistema. De los datos esenciales se reconoce que datos como el resumen y el usuario pueden presentar redundancia y fricción al momento de la interacción con el usuario; razón por la cual se plantea aprovechar tecnologías y técnicas modernas utilizadas en materia de IA generativa para poder obtener estos 2 campos de forma implícita, permitiendo una experiencia de usuario fluida y eficiente.

CAPÍTULO III. RESULTADOS Y DISCUSIÓN

3.1 Análisis de necesidades

En base a las entrevistas y observaciones realizadas, se levantaron las siguientes historias de usuario. Estas historias de usuario describen las necesidades identificadas durante el proceso de recolección de información.

Tabla 17: HU.01 - Estandarización de recepción de solicitudes.

Historia de Usuario	
Código	HU.01
Como	Personal administrativo
Necesito	Estandarizar la recepción de solicitudes de soporte
Para	Disponer de indicadores que permitan evaluar la calidad del servicio y retroalimentar a las unidades responsables del soporte técnico
Referencia	EN.01, EN.04, F0.01, F0.04

Tabla 18: HU.02 - Mejora de trazabilidad de incidencias.

Historia de Usuario	
Código	HU.02
Como	Personal administrativo
Necesito	Mejorar la trazabilidad de los casos de soporte
Para	Reducir la probabilidad de errores operativos y permitir que el personal técnico enfoque su tiempo en actividades de mayor valor
Referencia	EN.04, F0.05

Tabla 19: HU.03 - Adopción de sistemas existentes.

Historia de Usuario	
Código	HU.03
Como	Personal administrativo
Necesito	Incrementar la adopción del sistema de tickets
Para	Consolidar la recepción de solicitudes y facilitar la colaboración interna para la resolución eficiente de incidencias
Referencia	F0.02, F0.04

Tabla 20: HU.04 - Inversión de tiempo por parte del personal.

Historia de Usuario	
Código	HU.04
Como	Personal administrativo

Historia de Usuario	
Necesito	Reducir el tiempo invertido en tareas repetitivas por parte del personal de soporte
Para	Permitir que el personal destine su tiempo a actividades estratégicas y fortalecer la productividad del equipo
Referencia	EN.05, EN.06, F0.06, F0.01

Tabla 21: HU.05 - Unificación de flujo de atención.

Historia de Usuario	
Código	HU.05
Como	Personal administrativo
Necesito	Unificar la información relevante en un único flujo de atención
Para	Optimizar el seguimiento de incidencias y respaldar la evaluación del desempeño tanto del personal como de los servicios ofrecidos
Referencia	EN.01, EN.02, EN.04, F0.01, F0.05

Tabla 22: HU.06 - Gestión de Prioridades.

Historia de Usuario	
Código	HU.06
Como	Personal administrativo
Necesito	Facilitar la priorización de solicitudes
Para	Orientar los esfuerzos del personal hacia requerimientos críticos y mejorar la satisfacción de los clientes
Referencia	F0.03

Tabla 23: HU.07 - Comunicación con el cliente.

Historia de Usuario	
Código	HU.07
Como	Cliente
Necesito	Obtener una explicación clara del estado de su solicitud
Para	Tomar decisiones informadas en función del avance y estado de la incidencia
Referencia	EN.02, EN.06, F0.05, F0.06

Tabla 24: HU.08 - Experiencia de usuario.

Historia de Usuario	
Código	HU.08
Como	Cliente
Necesito	Reducir la fricción inicial para reportar un problema
Para	Reportar incidencias de manera oportuna sin depender de un entorno físico o canal específico

Historia de Usuario	
Referencia	F0.04, F0.06

3.2 Análisis de requerimientos

En la Tabla 25 se presentaron los requerimientos funcionales delimitados a partir de las historias de usuario identificadas, así como requerimientos propios de la naturaleza del sistema.

Tabla 25: Tabla de requerimientos funcionales.

Código	Requerimiento	Descripción	Referencia
RQ.01	Interfaz conversacional	Se requiere que el usuario pueda interactuar con el sistema usando lenguaje natural para asegurar una experiencia de usuario fluida	HU.07, HU.08
RQ.02	Integración con Odoo	El sistema debe integrarse con el ERP Odoo	HU.03, HU.05
RQ.03	Creación automática y asistida de tickets	En base al inconveniente reportado por el usuario, se debe crear un ticket de soporte técnico con la mínima información adicional solicitada.	HU.01, HU.02, HU.04, HU.08
RQ.04	Búsqueda y gestión inteligente de tickets	El agente debe ser capaz de detectar tickets de soporte relacionados o duplicados, así como poder realizar búsquedas en base a temática y fechas.	HU.02, HU.07

Asimismo, en la Tabla 26 se presentaron los requerimientos no funcionales identificados.

Tabla 26: Tabla de requerimientos no funcionales.

Código	Requerimiento	Descripción	Referencia
RQ.05	Seguridad y permisos de usuario	Se debe establecer permisos de usuario para la manipulación del agente.	HU.05
RQ.06	Extensibilidad	El sistema debe tener la capacidad de aumentar sus capacidades, habilitando automatizar más tareas.	EN.10

Código	Requerimiento	Descripción	Referencia
RQ.07	Manejo de errores	El sistema debe ser capaz de manejar errores suscitados desde la recepción del mensaje de parte del usuario hasta la finalización de la ejecución del agente.	EN.10
RQ.08	Mantenibilidad	El código de la solución debe ser limpio, claro y mantenible	EN.10
RQ.09	Interoperabilidad	El sistema no debe estar cerrado al ERP Odoo, debe presentar la capacidad de poder integrar servicios externos al núcleo del mismo.	HU.03, HU.05, EN.07

3.3 Análisis de recursos

Para el desarrollo de la solución presentada, Mivilsoft S.A facilitó los recursos descritos en la Tabla 27

Tabla 27: Tabla de recursos

Tipo	Recurso	Uso
Software	Sistema ERP Odoo versión 18	Presenta la plataforma de desarrollo y sistema anfitrión de la solución planteada. Proveyendo ORM, vistas, gestión de permisos y APIs que permiten integrarlo con sistemas externos.
	Módulo Helpdesk	Consulta y gestión de tickets de soporte técnico.
	Módulo Discuss	Canal de mensajería principal.
	Base de datos Postgresql versión 17	Almacenamiento de estructuras de datos necesarias para el sistema.
Infraestructura	Servidor hospedado en AWS (us-east)	Plataforma en donde se desplegará la solución.
Documentación	Documentación de sistemas de código abierto	Construcción de bases de conocimiento para el agente.
	Manuales de usuario	Construcción de bases de conocimiento para el agente.
Humanos	Jefe de TI	Retroalimentación y soporte
	Encargado de calidad	Retroalimentación y soporte

3.4 Modelos de lenguaje natural

A inicios de la década de 2020, la tecnología de Large Language Models (LLM) ha comenzado un incremento en su popularidad entre el público general. Esta rápida adopción aceleró significativamente el ritmo de desarrollo tecnológico en el campo, dando lugar a modelos cada vez más potentes, eficientes y especializados.

Paralelamente, la expansión del mercado motivó la aparición de nuevos actores y una competencia creciente entre organizaciones de investigación y empresas tecnológicas, lo cual fomentó la innovación continua y la diversificación de enfoques.

En el contexto temporal en que se desarrolla este proyecto, los modelos son pre-entrenados y optimizados para el uso de herramientas y ejecución de tareas.

3.4.1 Elección de modelos

Para la elección del modelo de lenguaje natural, se consideraron como referencia los benchmarks elaborados por *Artificial Analytics*, una entidad independiente especializada en la consolidación y evaluación de este tipo de información. La construcción de dichos benchmarks se fundamentó en la ejecución de un conjunto de pruebas y en la posterior asignación de una puntuación global. [34]

El proceso de selección del o de los modelos más adecuados se desarrolló conforme al siguiente procedimiento:

1. Se eligió un subconjunto de pruebas disponibles mediante la API proporcionada por Artificial Analytics, asignando una ponderación a cada prueba en función de la relevancia otorgada.
2. A través de la API se obtuvo un conjunto de datos con los modelos evaluados, a partir del cual se calculó una calificación final considerando las ponderaciones definidas.
3. Con base en las calificaciones obtenidas, se identificaron los 10 modelos con mejor desempeño.
5. Finalmente, se construyó una matriz comparativa de los modelos resultantes, incluyendo costos por millón de tokens de entrada y salida, así como ventana de contexto máximo y latencias.

Cabe recalcar que, del dataset consultado de Artificial Analytics, se descartaron aquellas entradas en donde no se pudiera determinar los campos analizados.

La Figura 1 representa el proceso de selección que se va a seguir.

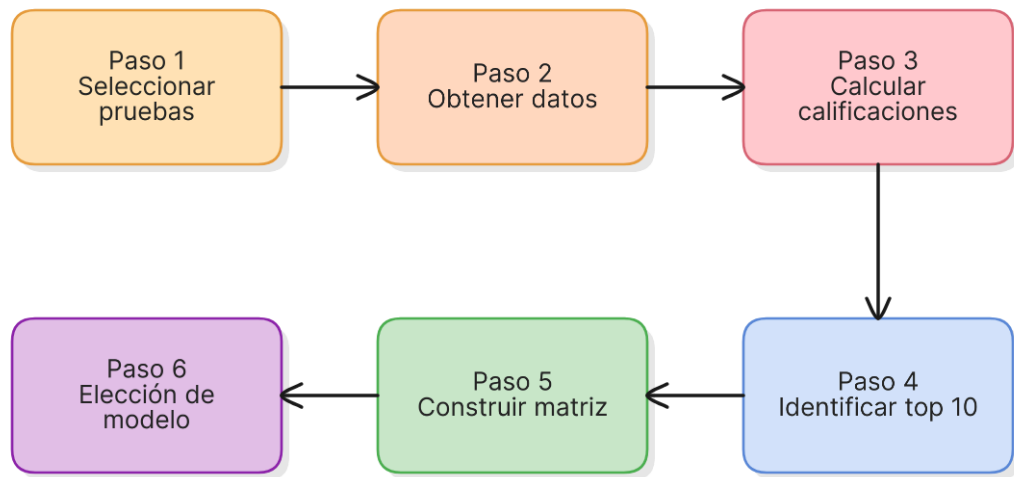


Figura 1: Proceso de selección de modelo de lenguaje natural.

a. Pruebas consideradas

La presente tabla recopila y sintetiza los benchmarks seleccionados para evaluar modelos de lenguaje con potencial de uso en sistemas de agentes. Cada benchmark se describió brevemente y se acompañó de la justificación correspondiente, en función de su pertinencia, alcance y alineación con los requerimientos del proyecto.

Tabla 28: Conjunto de benchmarks utilizados en la evaluación.

Benchmark	Descripción	Justificación
τ^2 -Bench	Banco de pruebas que evalúa agentes conversacionales en entornos dinámicos de control dual, donde agente y usuario actúan con herramientas, permitiendo analizar coordinación, comunicación y razonamiento en tareas generadas composicionalmente. [35]	Se selecciona τ^2 -Bench debido a su inclusión de escenarios vinculados al soporte técnico, lo que resulta especialmente pertinente para el entorno operativo del asistente propuesto.
Terminal-Bench	Benchmark desarrollado por el Instituto Laude con el objetivo de evaluar agentes de IA en entornos de terminal reales, midiendo su capacidad para ejecutar tareas de extremo a extremo como compilar código, entrenar modelos y configurar sistemas mediante un conjunto reproducible de tareas y un entorno controlado. [36]	A diferencia de τ^2 -Bench, Terminal-Bench se especializa en entornos de administración de sistemas, sin embargo, aporta en la medición de uso de herramientas de parte de los modelos de lenguaje natural.

Los resultados de ambos benchmarks se obtuvieron en formato Pass@1 al consultar el conjunto de datos de Artificial Analytics, lo que indica que se disponía de valores estandarizados acordes con dichos benchmarks. [34]

b. Resultados

En la Tabla 29 se presenta un resumen del dataset obtenido mediante la API de Artificial Analytics, en el cual se observaba que se disponía de información correspondiente a 151 modelos.

Tabla 29: Resumen de dataset sujeto a análisis.

Estadística	Terminal-Bench	τ^2 -Bench
Cantidad	151.0	151.0
Media	0.131	0.433
Desviación Estandar	0.111	0.245
Mínimo	0.007	0.041
P-25%	0.043	0.24
P-50%	0.092	0.327
P-75%	0.213	0.646
Máximo	0.44	0.933

A partir de este resumen se evidenció que τ^2 -Bench presenta tanto una media como una variabilidad superiores respecto de Terminal-Bench. Esto sugiere que dicho benchmark incluye un conjunto de tareas en las que los modelos analizados tienden a exhibir un mejor desempeño global; al mismo tiempo que manifiestan diferencias más marcadas entre sí, ofreciendo una mayor capacidad discriminativa para distinguir entre modelos de alto y bajo rendimiento.

• τ^2 -Bench

A continuación se muestran los modelos con mejor desempeño según τ^2 -Bench, en donde se observó que *Grok 4.1 Fast (Reasoning)* tiene la mejor puntuación; lo que significa que este modelo presenta una muy buena opción para escenarios de soporte técnico.

Tabla 30: Modelos más destacados según τ^2 -Bench [34]

Proveedor	Nombre	Calificación
xAI	Grok 4.1 Fast (Reasoning)	0.933
Kimi	Kimi K2 Thinking	0.93

Proveedor	Nombre	Calificación
DeepSeek	DeepSeek V3.2 (Reasoning)	0.906
Amazon	Nova 2.0 Pro Preview (low)	0.906
xAI	Grok 3 mini Reasoning (high)	0.904
Anthropic	Claude Opus 4.5 (Reasoning)	0.895
KwaiKAT	KAT-Coder-Pro V1	0.886
Google	Gemini 3 Pro Preview (high)	0.871
MiniMax	MiniMax-M2	0.868
OpenAI	GPT-5 Codex (high)	0.868

• *Terminal-Bench*

Asimismo, en la se muestra los modelos mejor puntuados según Terminal-Bench. Se pudo apreciar resultados distintos debido a que el benchmark presenta pruebas distintas a τ^2 -Bench. En este benchmark se observa que *Claude Opus 4.5 (Reasoning)* presenta la mejor opción para tareas relacionadas con administración de sistemas.

Tabla 31: Modelos más destacados según Terminal-Bench [34]

Proveedor	Nombre	Calificación
Anthropic	Claude Opus 4.5 (Reasoning)	0.44
OpenAI	GPT-5.1 (high)	0.426
Google	Gemini 3 Pro Preview (high)	0.39
Anthropic	Claude Opus 4.5 (Non-reasoning)	0.383
xAI	Grok 4	0.376
OpenAI	GPT-5 (medium)	0.362
OpenAI	GPT-5 Codex (high)	0.355
OpenAI	o3	0.348
Anthropic	Claude 4.5 Sonnet (Reasoning)	0.333
DeepSeek	DeepSeek V3.2 (Reasoning)	0.333

• *Ponderación final*

Considerando los puntos citados anteriormente, así como la naturaleza de las pruebas ejecutadas, se asignó una ponderación de 75% para τ^2 -Bench, y de 25% para Terminal-Bench. Como resultado, la puntuación que se dio a cada modelo fue la siguiente:

$$P_{T'} = (P_{\tau^2} * 0.75) + (P_T * 0.25)$$

En donde:

- $P_{T'}$ es la puntuación total del modelo
- P_{τ^2} es la puntuación del modelo según τ^2 -Bench
- P_T es la puntuación del modelo según Terminal-Bench

Tabla 32: Tabla de modelos según la ponderación final asignada

Proveedor	Nombre	Puntuación	Costo por millón de tokens de entrada	Costo por millón de tokens de salida	Velocidad de respuesta en tokens por segundo
Anthropic	Claude Opus 4.5 (Reasoning)	0.78125	\$5.0	\$25.0	62.761
Kimi	Kimi K2 Thinking	0.77025	\$0.6	\$2.5	82.437
DeepSeek	DeepSeek V3.2 (Reasoning)	0.76275	\$0.28	\$0.42	27.958
xAI	Grok 4.1 Fast (Reasoning)	0.7565	\$0.2	\$0.5	152.587
Google	Gemini 3 Pro Preview (high)	0.75075	\$2.0	\$12.0	124.695
Anthropic	Claude Opus 4.5 (Non-reasoning)	0.743	\$5.0	\$25.0	74.465
OpenAI	GPT-5 Codex (high)	0.73975	\$1.25	\$10.0	215.899
OpenAI	GPT-5 (medium)	0.73925	\$1.25	\$10.0	126.613
OpenAI	GPT-5.1 (high)	0.72075	\$1.25	\$10.0	102.072
Amazon	Nova 2.0 Pro Preview (low)	0.72025	\$1.25	\$10.0	129.135

A partir de la Tabla 32, fue posible identificar varios patrones relevantes. En primer lugar, el modelo *Grok 4.1 Fast (Reasoning)* se distinguió como la alternativa más económica entre las evaluadas, con un costo aproximado de \$0,20 por millón de tokens de entrada y \$0,50 por millón de tokens de salida. En segundo lugar se posicionó *DeepSeek V3.2 (Reasoning)*, el cual también ofrece capacidades de razonamiento, aunque con un costo ligeramente superior. Adicionalmente, *Grok 4.1 Fast (Reasoning)* obtiene la puntuación más alta en el benchmark τ^2 -Bench, lo que refuerza su competitividad técnica.

En cuanto al rendimiento, *GPT-5 Codex (high)* presenta la mayor velocidad de generación, alcanzando más de 200 tokens por segundo. No obstante, su desempeño global y por τ^2 -Bench no se encuentra entre los más destacados. El siguiente modelo

en velocidad es, nuevamente, *Grok 4.1 Fast (Reasoning)*, con aproximadamente 152 tokens por segundo.

c. Selección de modelo

En síntesis, *Grok 4.1 Fast (Reasoning)* emerge como la alternativa más equilibrada para su adopción, al combinar un costo reducido, una velocidad competitiva y una alta puntuación en los indicadores de evaluación. No obstante, al momento de redacción del presente trabajo, *Grok 4.1 Fast (Reasoning)* cuenta con límites de uso, establecidos en 4 millones de tokens por minuto y 480 peticiones por minuto.

Si se requiere contar con alternativas priorizando principalmente la reducción de costos, mientras se mantiene un buen desempeño global, *DeepSeek V3.2 (Reasoning)* constituye una opción atractiva, puntuando tercero en el puntaje global y en τ^2 -Bench; sin embargo, su menor velocidad de respuesta debe ser considerada como una desventaja potencial.

Para el desarrollo y pruebas de la solución se hizo uso del modelo *Deepseek 3.2 (Reasoning)*.

3.5 Tecnologías de desarrollo de agentes

El incremento en popularidad de la IA trajo consigo un interés en desarrollar herramientas y frameworks para trabajar con estas tecnologías. Para el desarrollo de la solución del presente proyecto, se consideraron las siguientes:

3.5.1 N8N

N8N es una plataforma de automatización de tipo no-code que permite diseñar flujos de trabajo mediante una interfaz web gráfica. En esta interfaz, los componentes del sistema se representan como bloques configurables, los cuales pueden interconectarse para conformar procesos de mayor complejidad. [37]

La plataforma puede ser implementada tanto en infraestructura propia como consumida bajo el modelo de plataforma como servicio. En el caso del despliegue local, N8N ofrece dos ediciones: una edición comunitaria, orientada al uso general, y una edición empresarial, destinada a organizaciones con requerimientos avanzados. [37]

Asimismo, N8N dispone de integraciones con una amplia variedad de servicios externos, compatibilidad con múltiples modelos de lenguaje y capacidades para la implementación de sistemas basados en RAG.

3.5.2 Vercel AI SDK

AI SDK es un framework de código abierto desarrollado por Vercel que permite la construcción de aplicaciones basadas en modelos de lenguaje natural utilizando Javascript o Typescript. El SDK proporciona una interfaz unificada para interactuar con distintos proveedores de modelos y ofrece componentes compatibles con *React*, *Next.js*, *Vue* y *Svelte*, facilitando el desarrollo de interfaces web integradas a estas tecnologías. [38]

3.5.3 LangChain

LangChain es un framework de código abierto desarrollado por la empresa homónima, que permite la creación ágil de aplicaciones basadas en modelos de lenguaje natural utilizando Python, JavaScript o TypeScript. [39]

De manera similar a AI SDK, LangChain ofrece una interfaz unificada para la interacción con diversos proveedores de modelos. Además, pone a disposición un conjunto de herramientas que facilitan la implementación de patrones y arquitecturas comúnmente empleadas en el desarrollo de soluciones basadas en tecnologías de lenguaje natural. [39]

3.5.4 Elección de tecnología: LangChain

Para el desarrollo del proyecto se usó LangChain. LangChain ofrece como principal ventaja su compatibilidad con el lenguaje de programación utilizado por Odoo, lo que facilita la incorporación de funcionalidades adicionales directamente dentro del sistema. Esta compatibilidad permite alcanzar un nivel de integración más profundo, reduciendo la necesidad de servicios intermediarios.

Asimismo, el uso de LangChain posibilita aprovechar los modelos y la arquitectura nativa de Odoo para extender las capacidades de los canales de mensajería. Gracias a ello, es factible prescindir de componentes externos destinados a la integración con el sistema de comunicación propio de la plataforma, logrando una solución más unificada y eficiente.

3.6 Arquitecturas utilizadas

La construcción de agentes basados en inteligencia artificial hacen uso de arquitecturas de software conocidas y definidas.

3.6.1 Multi-capa

La arquitectura multi-capa es una arquitectura de software que organiza el sistema en capas lógicas diferenciadas con la finalidad de separar responsabilidades y gestionar complejidad. La arquitectura define al menos tres capas: presentación, lógica de negocio y acceso a datos, cada una dependiente únicamente de la inmediatamente inferior, favoreciendo la mantenibilidad y la escalabilidad. En aplicaciones web, el modelo multi-capa ha sido tradicionalmente el estándar, permitiendo distribuir funciones en diferentes niveles y reducir el acoplamiento entre componentes. [40]

3.6.2 Hexagonal

La arquitectura hexagonal, conocida también como arquitectura de puertos y adaptadores, es una arquitectura de software que busca la independencia del núcleo de negocio frente a las implementaciones tecnológicas externas, mediante la definición explícita de puertos como interfaces y adaptadores como sus implementaciones concretas. En este enfoque, el núcleo de la aplicación interactúa únicamente con puertos abstractos, de modo que las dependencias hacia la infraestructura, las interfaces de usuario y otros sistemas quedan desacopladas, lo que favorece la mantenibilidad, la flexibilidad y la capacidad del sistema de ser expuesto a distintas pruebas de funcionamiento. Este modelo permite la inter-cambiabilidad de componentes y facilita la aplicación de pruebas automatizadas, ya que las implementaciones externas pueden sustituirse sin afectar la lógica central. La arquitectura hexagonal se ha comparado conceptualmente con otras arquitecturas limpias, resaltando su énfasis en la separación de responsabilidades. [41]

3.6.3 Modelo Vista Controlador (MVC)

MVC es una arquitectura de software que organiza una aplicación en tres componentes principales facilitando la escalabilidad, el mantenimiento y las pruebas. En este enfoque, el «*Modelo*» encapsula el estado de la aplicación y su lógica de negocio, incluyendo operaciones sobre los datos; la «*Vista*» se encarga de la presentación de la información y de la interfaz con el usuario; y el «*Controlador*» recibe las solicitudes del usuario, coordina la interacción entre el Modelo y la Vista, y determina qué respuesta debe generarse. [42]

3.6.4 Elección de arquitectura: Multi-capa

Por sus características, la arquitectura Multi-capa se eligió con fundamento en la solución propuesta; esto debido a que su integración en el sistema Odoo se implantó mediante un módulo de desarrollo para la función del agente, y el sistema Odoo, así como sus módulos se construyen con base en el patrón de arquitectura Multi-capa.

3.7 Buenas prácticas

3.7.1 Identificar el mejor uso para un modelo de lenguaje

Previo a la implementación de cualquier sistema basado en modelos de lenguaje, resultó fundamental realizar un análisis que permita determinar la solución más adecuada. En términos de buenas prácticas, se recomienda privilegiar alternativas más simples, ya que estas suelen favorecer un desempeño más estable y eficiente del modelo aplicado. [43]

Durante las etapas de planificación y diseño del sistema, es esencial distinguir si la finalidad del proyecto demanda la utilización de un agente o de un *workflow*. La Tabla 33 expone las características centrales de ambos enfoques

Tabla 33: Tabla comparativa de agentes vs workflows. [43], [44]

Criterio	Agentes	Workflows
Naturaleza del sistema	Modelo de lenguaje que orquesta funciones mediante el uso de herramientas, operando con alto grado de autonomía.	Secuencia estructurada de pasos definida mediante código que orquesta la ejecución de modelos de lenguaje.
Flexibilidad	Alta flexibilidad, capaz de manejar casos límite y adaptarse a situaciones nuevas o imprevistas.	Flexibilidad limitada a las reglas previamente establecidas en el código.
Control y predictibilidad	Menor predictibilidad debido a su carácter no determinístico.	Alta predictibilidad gracias a su naturaleza determinística.
Complejidad de implementación	Mayor complejidad técnica y mayor probabilidad de errores durante la integración.	Implementación más simple y controlada.
Comportamiento del sistema	Sistema autónomo, no determinístico.	Proceso determinístico que contribuye a sistemas más amplios.
Casos de uso ideales	Tareas avanzadas como agentes de programación y soporte técnico especializado.	Tareas simples y estructuradas, por ejemplo, generación de resúmenes.

3.7.2 Definir controles de seguridad

Es importante los riesgos de seguridad asociados a la incorporación de modelos de lenguaje que procesan las interacciones no estructuradas de los usuarios. Es recomendable introducir mecanismos de control cuya complejidad puede variar según el contexto del sistema. Estos van desde estrategias básicas como incluir restricciones explícitas dentro de las instrucciones del sistema, hasta la aplicación de patrones más avanzados que permitan validar de manera rigurosa las solicitudes recibidas. Dichos controles pueden implementarse tanto antes de la ejecución del modelo como después de generada la respuesta. [45]

3.7.3 Precisión en la formulación de acciones, resultados y expectativas

No se debe asumir que el modelo generará los resultados en el formato previsto por defecto. Se recomienda describir con precisión las acciones que deben ejecutarse y los resultados que se esperan obtener. [46]

Esto contribuye a minimizar la interpretación ambigua por parte del modelo, disminuye la posibilidad de errores o desviaciones indeseadas y permite mantener un mayor control sobre el flujo conversacional.

Ejemplos incluyen detallar cuándo deben solicitarse datos al usuario, qué información es opcional para la ejecución de una tarea, cómo deben emplearse recursos externos o APIs, cuál debe ser la redacción exacta de ciertos mensajes operativos, cuáles son los estándares de calidad esperados, y criterios de toma de decisiones. [47]

3.7.4 Migrar a sistemas multi-agente cuando la complejidad escale

Cuando se trabaja con agentes, su implementación debe iniciarse mediante un enfoque de agente único, dado que este modelo favorece la simplicidad y la coherencia operativa. No obstante, a medida que aumenta la cantidad de herramientas disponibles para el agente y se incrementa la complejidad de las instrucciones del sistema, pueden manifestarse errores en el razonamiento, tales como la incapacidad para mantener el flujo de trabajo previsto o la ejecución incorrecta de herramientas. Ante este tipo de limitaciones, resulta pertinente migrar hacia una arquitectura multi-agente, donde las responsabilidades se distribuyen entre agentes más simples y especializados. Este cambio posibilita una mayor escalabilidad del sistema y contribuye significativamente a la mejora del rendimiento global. [46]

3.8 Metodología de desarrollo

Las metodologías de desarrollo de software son marcos de trabajo orientados a la planificación y el control de procesos de construcción de sistemas de información, permitiendo organizar actividades, recursos, tiempos y presupuestos en función de los objetivos del proyecto.

Se definen dos enfoques predominantes: las metodologías tradicionales, caracterizadas por la definición rígida y anticipada de los requisitos, y las metodologías ágiles, basadas en iteraciones cortas, adaptabilidad y entrega temprana de valor. Según el estudio de J. Morales-Carrillo, L. Cedeño-Valarezo, J. S. C. Bravo, y J. G. O. G. de I. S. Calderón [48], las metodologías ágiles son las más utilizadas en la actualidad, destacando Scrum, XP y Mobile-D como las de mayor frecuencia en los proyectos analizados.

Para el desarrollo del sistema se contemplaron metodologías de desarrollo ágil, ya que estas se adaptaron a condiciones presentadas como el tiempo de desarrollo y el equipo de trabajo. Las metodologías consideradas a su uso fueron:

3.8.1 Scrum

Metodología de desarrollo ágil con foco en el seguimiento del estado actual del software. Esta metodología prioriza a los individuos e interacciones sobre los procesos y tareas; organiza el proceso de desarrollo en iteraciones llamadas «*Sprint*»; y establece una serie de reuniones con el objetivo de dar seguimiento al avance del desarrollo [49]. Scrum es la metodología preferida por B. S. Guaylla Ashqui y J. S. Burgos Meléndrez [8] en su trabajo de investigación debido a que permite una adaptación fluida y colaboración constante.

3.8.2 Programación extrema

La programación extrema es una metodología ágil y ligera que propone la comunicación entre los individuos cara a cara por sobre la documentación escrita, razón por la cual es mucho más eficiente en equipos de desarrollo pequeños que comparten una misma oficina; su objetivo principal es destinar más tiempo al diseño y la implementación de la solución al eliminar la necesidad de documentar los diseños y las etapas del diseño [50]. Esta metodología es la seleccionada por C. H. Llerena Valencia [4] y G. F. Reyes Pullugando [7] debido a su uso de iteraciones y mejora continua, junto con su foco en el desarrollo mismo.

3.8.3 Scrumban

Scrumban es una metodología ágil híbrida que incorpora elementos de Scrum y Kanban para proveer de flexibilidad y visualización los proyectos de software. Surge de combinar la estructura iterativa de Scrum con el concepto de flujo continuo de Kanban, a través de uso de tableros adaptables para visualizar y controlar las tareas. [51]

3.8.4 Tabla comparativa

En la Tabla 34 se presenta una comparativa de las metodologías investigadas, se comparan ámbitos como el tamaño del equipo, ventajas y complejidad de implementación.

Tabla 34: Tabla comparativa de metodologías de desarrollo.

Criterio	Scrum	Programación Extrema (XP)	Scrumban
Enfoque principal	Seguimiento del estado del software y adaptación continua	Comunicación cara a cara y énfasis en diseño e implementación	Combinación de iteraciones planificadas con visualización del flujo y límites de trabajo en curso
Estructura de trabajo	Sprints y reuniones de seguimiento	Iteraciones con mejora continua y poca documentación	Iteraciones opcionales con tablero visual y flujo continuo de tareas
Tamaño del equipo	Pequeño a mediano	Pequeños y co-localizados	Flexible; similar a Scrum pero con equipo autoorganizado y sin roles rígidos
Ventajas destacadas	Adaptación fluida y colaboración	Eficiencia en equipos pequeños y foco en el desarrollo	Flexibilidad híbrida que permite planificación incremental y gestión visual del flujo
Herramientas/artefactos clave	Sprints	Iteraciones y prácticas ligeras	Tablero híbrido (visualización tipo Kanban y opcional planificación de iteraciones, límites de trabajo en curso)

3.8.5 Metodología aplicada

El desarrollo del presente proyecto estuvo respaldado por la metodología Scrumban, seleccionada debido a que permite estructurar el trabajo en fases claramente definidas mientras que ofrece una flexibilidad operativa superior a la de otras metodologías como Scrum.

3.9 Diseño de la solución

3.9.1 Fase 1: Establecer objetivos

En la Sección 3.1 se llevó a cabo un análisis integral de necesidades, a partir del cual se formularon historias de usuario y se derivaron los correspondientes requerimientos funcionales. No obstante, además de estos requerimientos iniciales, se incorporaron los siguientes objetivos, los cuales responden a las particularidades propias de la solución propuesta y orientan su implementación.

Código	Requerimiento	Descripción	Referencia
RQ.10	Indicadores de uso y trazabilidad	El sistema debe ser capaz de reportar su uso, así como dar las herramientas para verificar que los modelos se estén comportando según lo planeado, con la finalidad de poder aplicar mejores continuas al agente.	
RQ.11	Personalización del sistema	El sistema debe permitir la modificación de parámetros clave para el funcionamiento del agente sin la necesidad de modificaciones al código del mismo.	
RQ.12	Acceso a documentación	El agente debe tener acceso a la documentación del sistema para poder responder en base a la misma.	

3.9.2 Fase 2: Backlog del proyecto

a. Actores

Los actores son los distintos roles de usuario identificados, para el desarrollo de la solución se identificaron los siguientes actores.

Tabla 35: Actores del sistema.

Actor	Descripción	Función
Usuario	Cliente o empleado, o invitado que tiene la posibilidad de interactuar con el agente	Interactúa con los agentes puestos a disposición por el administrador
Administrador	Gestiona agentes y los pone a disposición para que los usuarios puedan interactuar con estos	Crear y archivar agentes, asignar credenciales de API, habilitar la ejecución de herramientas a los agentes, actualizar bases de conocimientos de los agentes, y

Actor	Descripción	Función
		dar seguimiento a las estadísticas de uso de un agente

b. Módulos

Se identificaron los siguientes puntos principales de interacción con el sistema.

Tabla 36: Módulos del sistema.

Módulo	Funcionalidad	Actor
Chat	Interfaz principal de interacción con el modelo de lenguaje natural	Todos los actores
Agentes	Módulo para crear y archivar agentes, así como los parámetros de estos	Administrador
Credenciales	Módulo para gestionar credenciales de API de modelos de lenguaje natural	Administrador
Ejecuciones de Agente	Interfaz de visualización y reporte de ejecuciones que han realizado los agentes, permitiendo identificar errores y uso de tokens	Administrador
Documentos	Interfaz para registrar notas y documentos disponibles para que el agente los pueda referenciar para responder preguntas	Administrador

3.9.3 Fase 3: Planificación de actividades

Para la etapa de planificación, cada módulo del sistema se descompuso en un conjunto de actividades orientadas a atender las necesidades previamente identificadas. Cabe señalar que las actividades definidas en esta fase no son estáticas, sino que pueden ser revisadas y ajustadas a lo largo del desarrollo del proyecto.

Para el seguimiento del proyecto se hizo uso de la herramienta web *Notion*, en donde se realizó el tablero mostrado en Por otro lado, se ha definido una duración de sprint de 2 semanas.

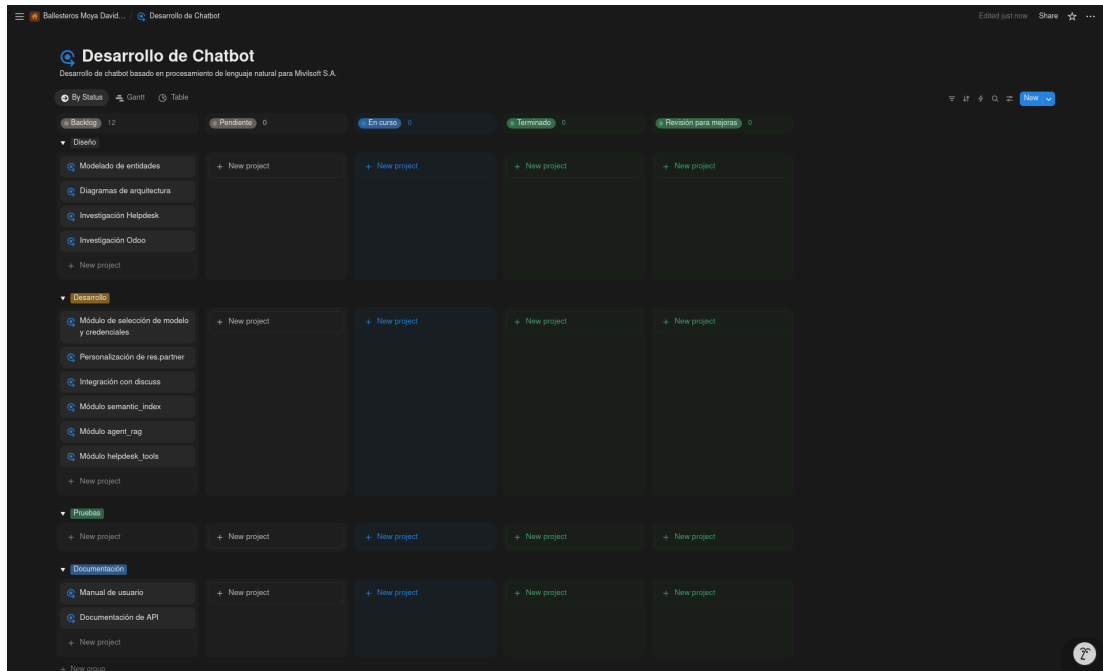


Figura 2: Tablero Kanban elaborado para el seguimiento de las tareas

3.9.4 Fase 4: Desarrollo iterativo

a. *Sprint 1*

El primer sprint se dedicó al estudio del sistema anfitrión de la solución planteada, diseño de solución e inicialización de esta. Se buscó contar con bases sólidas para el desarrollo de sprint posteriores, así como contar con retroalimentación del sistema antes de su implementación.

b. Flujo principal del agente

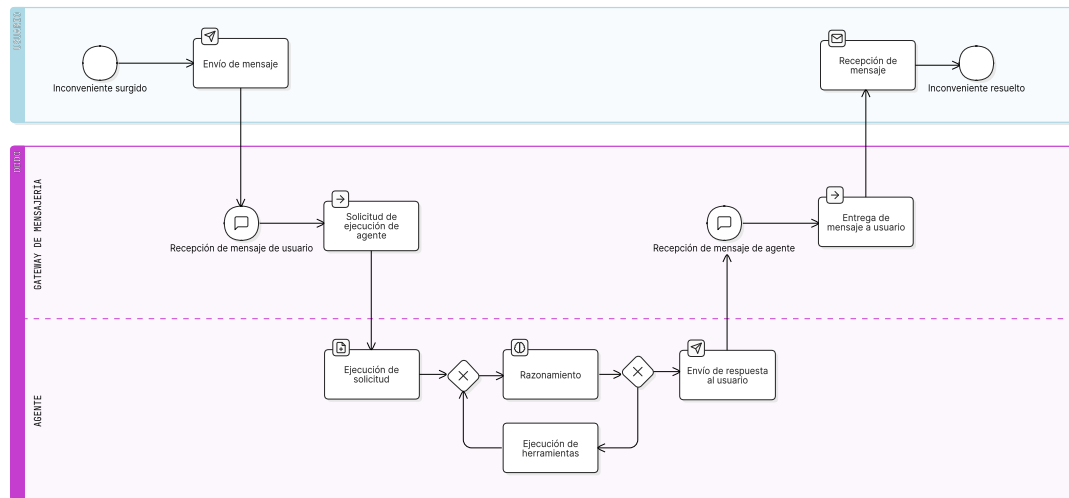


Figura 3: Diagrama BPMN del sistema.

En la Figura 3 se presentó el diagrama de interacción entre el usuario final y el agente, compuesto por los siguientes procesos:

1. El usuario inicia redactando un mensaje al agente.
2. El gateway de mensajería, en este caso el canal de mensajería propio de Odoo se encarga de recibir y discernir dicho mensaje.
3. Una vez validado el mensaje, se crea una estructura de datos denominada como *Agent Request*; esta cumple con el propósito de dar trazabilidad a las interacciones que se ha tenido con el agente.
4. Posterior a la creación del Agent Request, se procede a inicializar la ejecución del agente. Durante este proceso se consideraron los parámetros del agente, tales como las instrucciones del sistema, temperatura, herramientas disponibles, y reintentos.
5. El sistema toma la el Agent Request generado e inicia el ciclo del agente.
7. Durante la ejecución del agente, el modelo de lenguaje natural realizará un proceso de razonamiento para determinar la necesidad de ejecución de herramientas, retornando el resultado de este razonamiento al servidor para que ejecute las herramientas de ser necesario, o continúe con el flujo normal del sistema.
8. Una vez obtenida la respuesta del agente, se envía dicha respuesta al usuario.

c. Manejo de errores

El sistema contempla posibles errores durante el ciclo del agente, para ello se implementaron las siguientes medidas:

- **Middleware de herramientas**

Usando las capacidades ofrecidas por LangChain, se configuró un middleware que intercepta las llamadas a las herramientas con la finalidad de capturar errores y retornar un mensaje indicando lo ocurrido al agente, siguiendo el flujo descrito en la Figura 4.

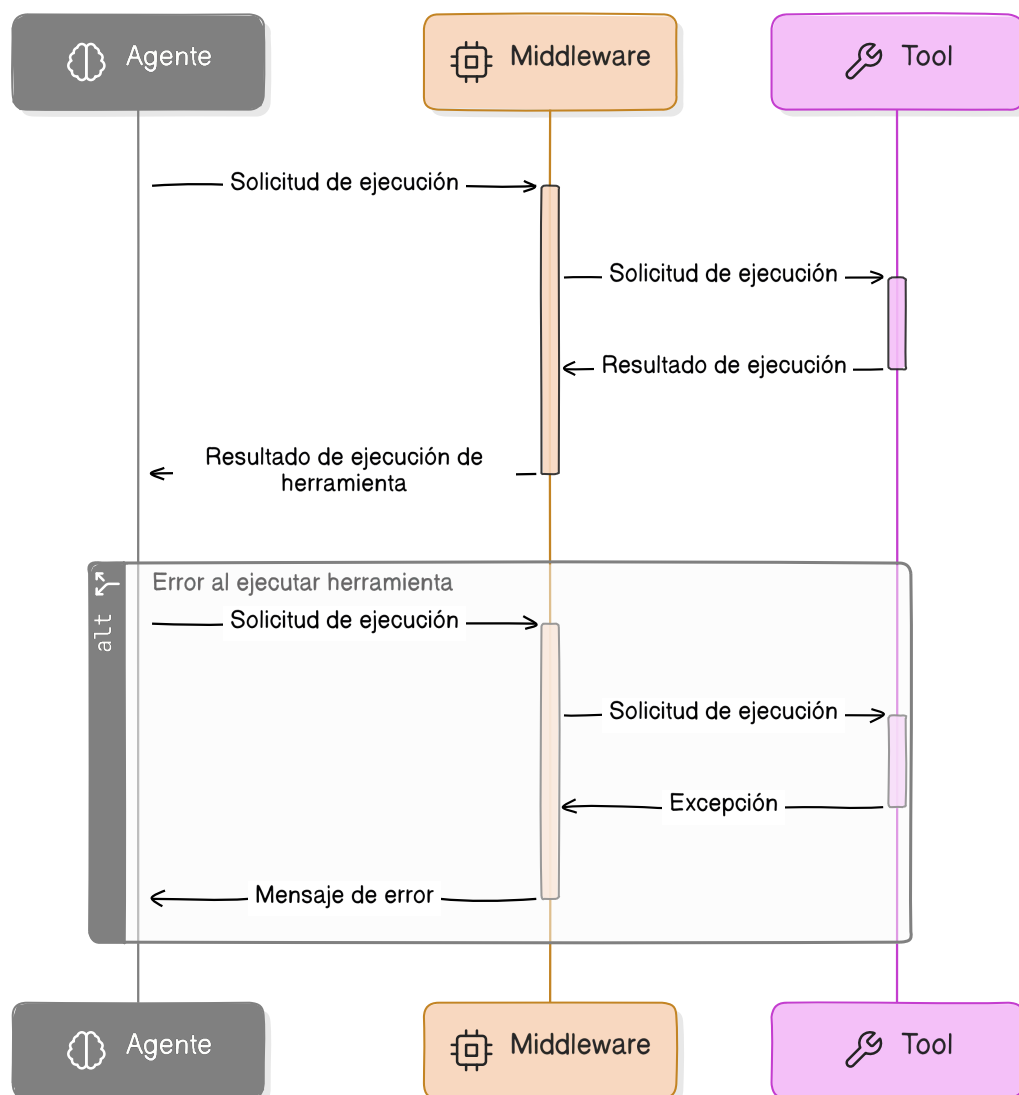


Figura 4: Diagrama de funcionamiento de middleware de herramientas.

- **Reintentos y cambio de credenciales**

En busca de diseñar un sistema resiliente ante fallos tales como errores de red o proveedores caídos, se diseñó un flujo que contempla políticas de reintentos y distintas credenciales para cumplir con la ejecución del agente. Ambos parámetros configurables por el administrador.

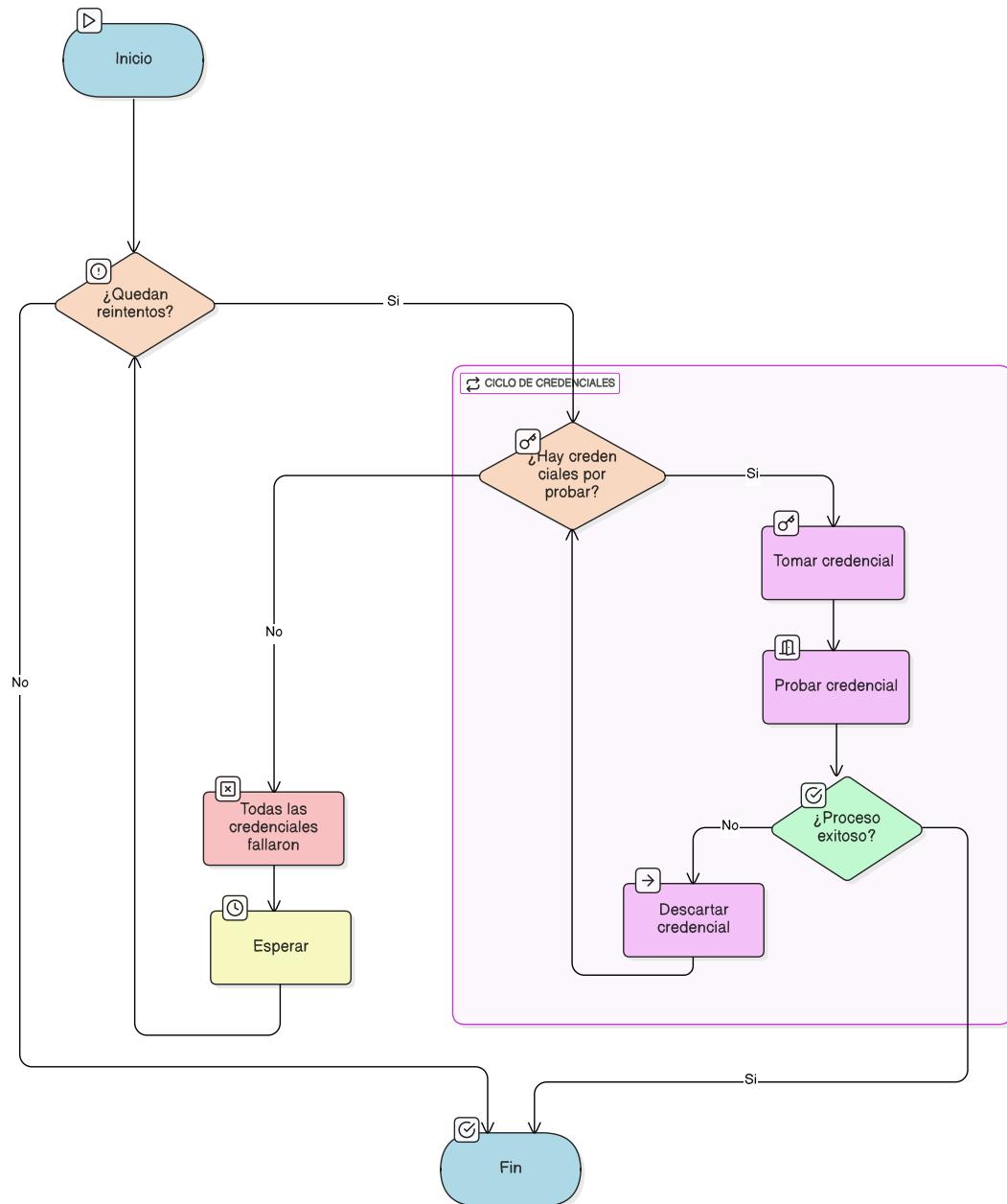


Figura 5: Diagrama de lógica de reintentos.

d. Modelo entidad relación

Odoo provee la capacidad de heredar modelos con la finalidad de agregar métodos o atributos. La solución hace uso de esta característica para poder crear socios que actúen como agentes. En la Figura 6 se aprecia la estructura de los modelos utilizados para la implementación base de los agentes, y el funcionamiento de los canales de mensajería de Odoo como el gateway de mensajería principal.

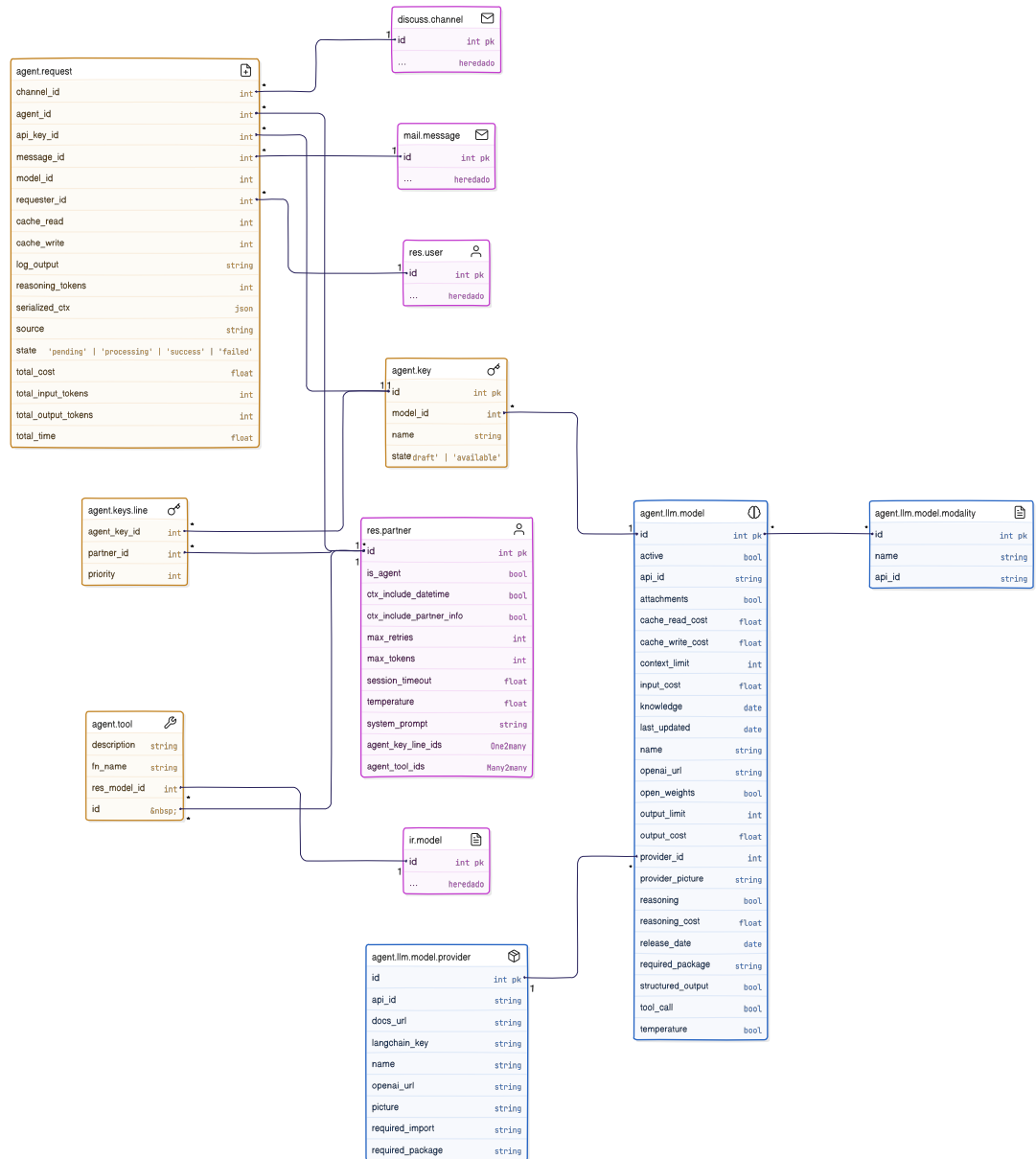


Figura 6: Diagrama entidad-relación del sistema.

e. Configuración de proyecto

Posterior al diseño, se inicializó el proyecto. En Odoo se consideró una buena práctica segmentar los sistemas en módulos destinados a cumplir con una sola función, razón por la cual se segmentó el sistema en los siguientes módulos:

- ***Módulo 1: agent***

Se diseñó este módulo con la finalidad de ser una base general destinada únicamente al funcionamiento del agente. El módulo agrega las extensiones a los socios, los Agent Request, la gestión de credenciales, y la integración con el canal de mensajería de Odoo.

- ***Módulo 2: semantic_index***

Este módulo presenta el primitivo necesario para realizar búsquedas semánticas. Se integra con la base de datos vectorial pgvector, y expone una interfaz para dotar a otros modelos de búsquedas usando vectores.

El objetivo de la implementación de este módulo fue poder realizar búsquedas de tickets por temática en lugar de buscar coincidencias exactas, permitiendo al agente detectar tickets duplicados antes de su creación.

- ***Módulo 3: agent_helpdesk_tools***

Es módulo agent_helpdesk_tools presenta la integración de Helpdesk con el agente, mediante la definición de herramientas que permitan la consulta, creación y administración de tickets de soporte técnico.

- ***Módulo 4: agent_rag***

Se diseñó este módulo para dar la capacidad a los administradores de registrar documentos como base de conocimiento del agente, así como al agente de poder consultar dichos documentos al momento de responder preguntas del usuario.

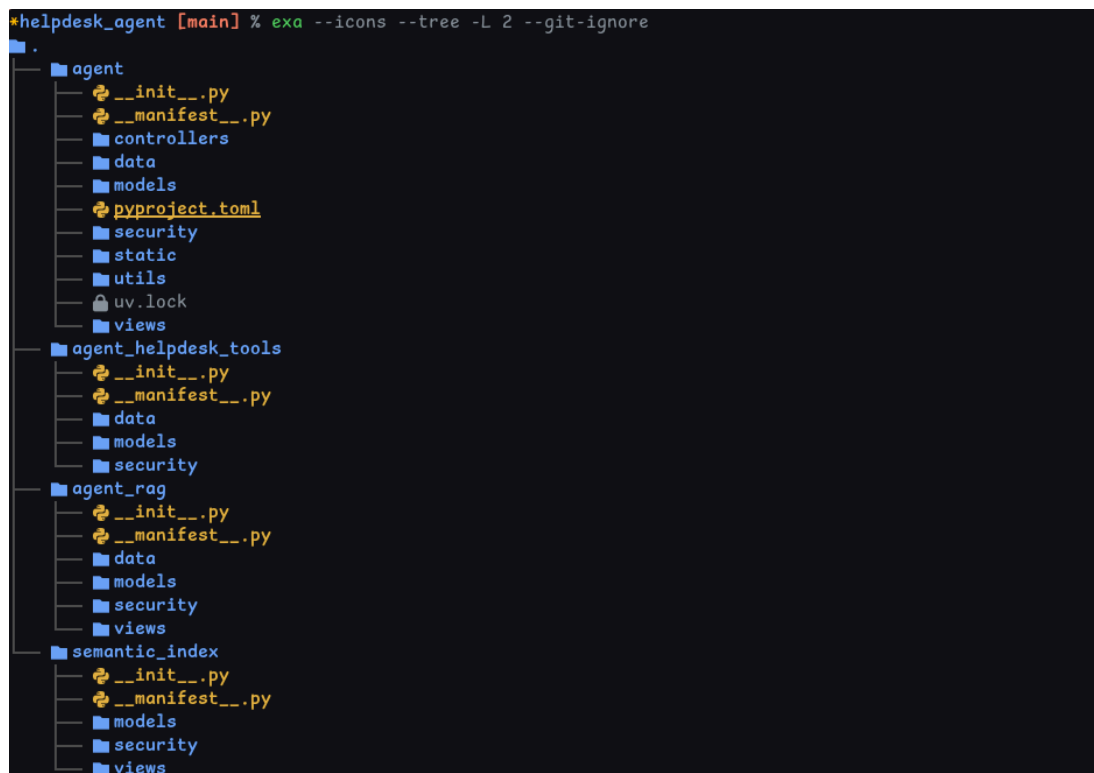


Figura 7: Estructura del proyecto.

f. Sprint 2

El segundo sprint abarcó el desarrollo de la elección de modelo y credenciales; componente esencial para la posterior configuración de los agentes. Asimismo, se desarrollaron las bases del Agent Request.

• *Sincronización*

Para la elección de modelos, se hizo uso de una API gratuita que dirige a un dataset mantenido por la comunidad. Este contiene información de precios, limitaciones y modalidades de los modelos, e implementa control de caché mediante la cabecera Etag, definida en el estándar HTTP.

Se realizó esta sincronización con la finalidad de proveer al usuario una lista de opciones para poder configurar el modelo que va a utilizar el agente, de esta forma no se limita al modelo escoger en el presente proyecto, sino que se da la posibilidad de cambiar este por uno de mejor calidad, considerando el constante desarrollo evidenciado en estas tecnologías.

En las siguientes figuras se evidencia todo el procesamiento de información para poder convertir los datos obtenidos del dataset externo a estructuras de datos propias de Odoo.

```

@api.model
def _sync(self) -> bool:
    # sudo: ir.config_parameter - access a system parameter
    etag = (
        self.env["ir.config_parameter"]
        .sudo()
        .get_param("agent.llm.model.etag", "")
    )
    headers = {} if not etag else {"If-None-Match": etag}
    response = requests.get("https://models.dev/api.json", headers=headers)
    if response.status_code == 304:
        return False

    response.raise_for_status()

    # sudo: agent.llm.model - no user have permission to create this model, only system
    Model = self.env["agent.llm.model"].sudo()

    for key, value in cast(dict, response.json()).items():
        commands = Model._api_to_commands(value.get("models", {}))
        self._upsert_from_api(key, value, commands)

    etag = response.headers.get("Etag")
    # sudo: ir.config_parameter - access a system parameter
    self.env["ir.config_parameter"].sudo().set_param(
        "agent.llm.model.etag", etag
    )

    return True

```

Figura 8: Funcionalidad de sincronización con dataset externo.

```

def _upsert_from_api(
    self, key: str, obj: dict, m_commands: list[tuple[int, int, dict]]
):
    # sudo: agent.llm.model.provider - read access
    provider = self.sudo().search([("api_id", "=", key)])

    lk, rp, ri, props, _ = get_mapping(key, obj)

    values = {
        "api_id": key,
        "docs_url": obj.get("doc", ""),
        "keys_definition": props or [],
        "langchain_key": lk or False,
        "model_ids": m_commands,
        "name": obj["name"],
        "openai_url": obj.get("api", ""),
        "picture": f"https://models.dev/logos/{key}.svg",
        "required_package": rp or False,
        "required_import": ri or False,
    }

    if not provider:
        # sudo: agent.llm.model.provider - only system can create this model
        self.sudo().create(values)
    else:
        # sudo: agent.llm.model.provider - only system can write this model
        provider.sudo().write(values)

```

Figura 9: Creación o actualización de proveedores a partir de los datos de la API.

```

@api.model
def _api_to_commands(self, models: dict) -> list[tuple[int, int, dict]]:
    # sudo: agent.llm.model.modality - users don't have permission to create this model, only system
    Modalities = self.env["agent.llm.model.modality"].sudo()
    cmds = []
    for model_key in models:
        mjson: dict = models[model_key]
        cjson: dict = mjson.get("cost", {})
        ljjson: dict = mjson["limit"]
        modjson: dict = mjson["modalities"]
        status: list[str] = mjson.get("status", [])

        input_ids = Modalities._api_to_command(modjson.get("input"))
        output_ids = Modalities._api_to_command(modjson.get("output"))

        m = self.search([("api_id", "=", model_key)])
        cmd = 1 if m else 0
        id = m.id if m else 0

        values = {
            "active": "deprecated" not in status,
            "api_id": model_key,
            "attachments": mjson.get("attachment", False),
            "cache_read_cost": cjson.get("cache_read", 0.0),
            "cache_write_cost": cjson.get("cache_write", 0.0),
            "context_limit": ljjson.get("context", 0),
            "input_cost": cjson.get("input", 0.0),
            "input_modality_ids": input_ids,
            "knowledge": self._format_date(mjson.get("knowledge", "")),
            "last_updated": self._format_date(mjson["last_updated"]),
            "name": mjson["name"],
            "output_cost": cjson.get("output", 0.0),
            "output_modality_ids": output_ids,
            "output_limit": ljjson.get("output", False),
            "reasoning": mjson.get("reasoning", False),
            "reasoning_cost": cjson.get("reasoning", 0.0),
            "release_date": self._format_date(mjson["release_date"]),
            "structured_output": mjson.get("structured_output", False),
            "temperature": mjson.get("temperature", False),
            "tool_call": mjson.get("tool_call", False),
        }

        cmds.append((cmd, id, values))
    return cmds

```

Figura 10: Creación o actualización de modelos a partir de los datos de la API.

```

def _api_to_command(
    self, mod: list[str]
) -> list[tuple[int, int, list[int]]]:
    return [(6, 0, self.search([("api_id", "in", mod)]).ids)]

```

Figura 11: Creación o actualización de modalidades a partir de los datos de la API.

En la Figura 12 se definió una tarea programada para ejecutar una sincronización diariamente.

```

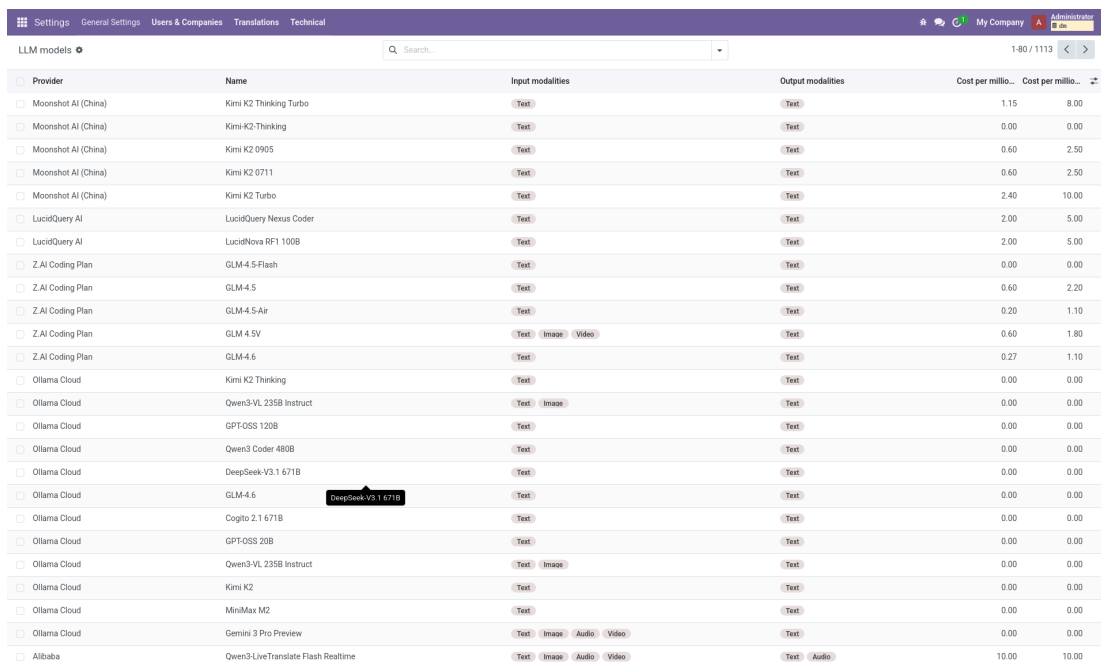
<odoo>
    <record id="llm_models.cron_sync_llm_models" model="ir.cron">
        <field name="name">Daily sync of LLM models and providers</field>
        <field name="model_id" ref="model_agent_llm_model_provider"/>
        <field name="state">code</field>
        <field name="code">model.action_sync()</field>
        <field name="interval_number">1</field>
        <field name="interval_type">days</field>
    </record>
</odoo>

```

Figura 12: Sincronización diaria.

• *Interfaz de usuario*

Por medio de las vistas de Odoo, se definieron vistas mediante las cuales el usuario interactúa con el sistema.



Provider	Name	Input modalities	Output modalities	Cost per millio...	Cost per millio...
☐ Moonshot AI (China)	Kimi K2 Thinking Turbo	Text	Text	1.15	8.00
☐ Moonshot AI (China)	Kimi K2 Thinking	Text	Text	0.00	0.00
☐ Moonshot AI (China)	Kimi K2 0905	Text	Text	0.60	2.50
☐ Moonshot AI (China)	Kimi K2 0711	Text	Text	0.60	2.50
☐ Moonshot AI (China)	Kimi K2 Turbo	Text	Text	2.40	10.00
☐ LucidQuery AI	LucidQuery Nexus Coder	Text	Text	2.00	5.00
☐ LucidQuery AI	LucidNova RF1 100B	Text	Text	2.00	5.00
☐ Z.AI Coding Plan	GLM-4.5-Flash	Text	Text	0.00	0.00
☐ Z.AI Coding Plan	GLM-4.5	Text	Text	0.60	2.20
☐ Z.AI Coding Plan	GLM-4.5-Air	Text	Text	0.20	1.10
☐ Z.AI Coding Plan	GLM 4.5V	Text Image Video	Text	0.60	1.80
☐ Z.AI Coding Plan	GLM-4.6	Text	Text	0.27	1.10
☐ Ollama Cloud	Kimi K2 Thinking	Text	Text	0.00	0.00
☐ Ollama Cloud	Qwen3-VL 235B Instruct	Text Image	Text	0.00	0.00
☐ Ollama Cloud	GPT-5.5 120B	Text	Text	0.00	0.00
☐ Ollama Cloud	Qwen3 Coder 480B	Text	Text	0.00	0.00
☐ Ollama Cloud	DeepSeek-V3.1 671B	Text	Text	0.00	0.00
☐ Ollama Cloud	GLM-4.6	Text	Text	0.00	0.00
☐ Ollama Cloud	Cogito 2.1 671B	Text	Text	0.00	0.00
☐ Ollama Cloud	GPT-5.5 20B	Text	Text	0.00	0.00
☐ Ollama Cloud	Qwen3-VL 235B Instruct	Text Image	Text	0.00	0.00
☐ Ollama Cloud	Kimi K2	Text	Text	0.00	0.00
☐ Ollama Cloud	MiniMax M2	Text	Text	0.00	0.00
☐ Ollama Cloud	Gemini 3 Pro Preview	Text Image Audio Video	Text	0.00	0.00
☐ Alibaba	Qwen3-LiteTranslate Flash Realtime	Text Image Audio Video	Text Audio	10.00	10.00

Figura 13: Listado de modelos de lenguaje natural.

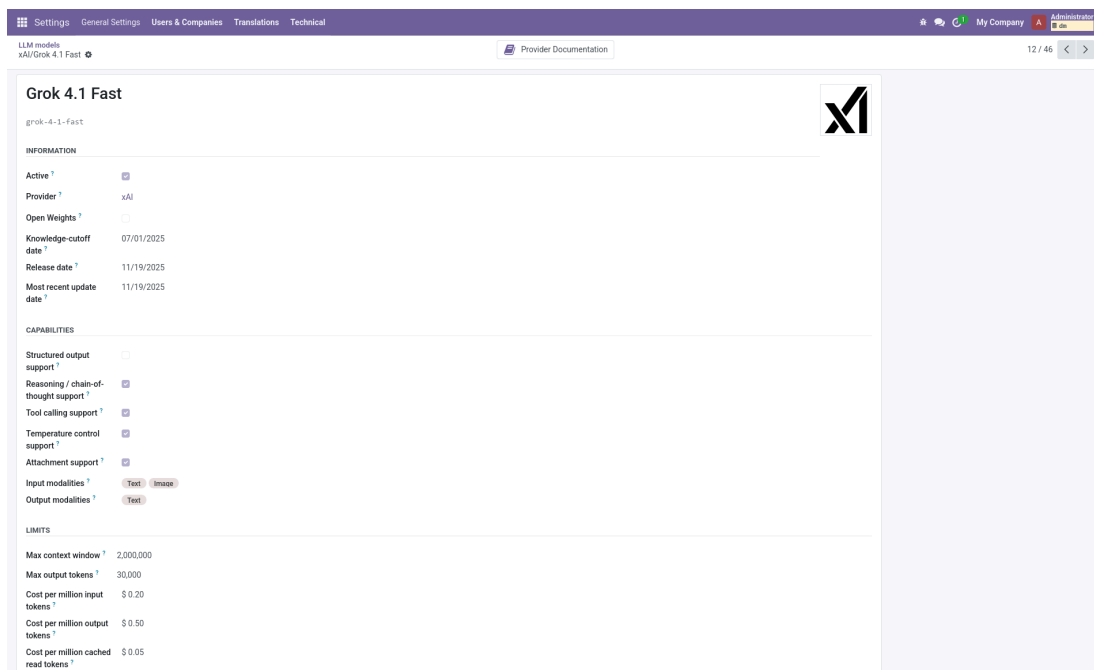


Figura 14: Información de modelos de lenguaje natural.

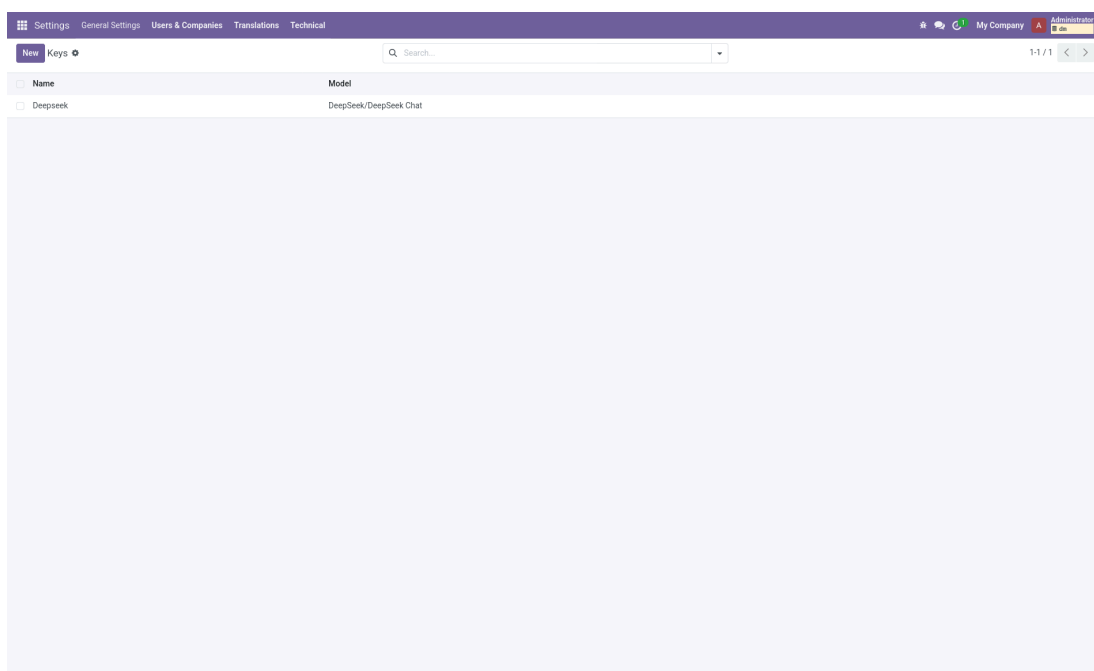


Figura 15: Almacenamiento de credenciales.

Figura 16: Formulario de creación de credenciales.

g. Sprint 3

El tercer sprint se dedicó a la parametrización del agente y a la integración de este con LangChain. El objetivo de esta integración es otorgar la capacidad de convertir un modelo `res.partner` de Odoo a un agente de LangChain con las configuraciones almacenadas.

• *Funcionalidades de los contactos*

Adicional a definir atributos adicionales para la configuración del agente, se sobrescribieron determinados métodos del modelo de Odoo para otorgar funcionalidad adicional cuando este modelo se detecta como un agente. En la Figura 17 se visualiza la modificación de métodos como `_compute_im_status`, cuyo objetivo es reportar el contacto como un bot, o fuera de línea a menos que no conte con credenciales válidas; `_to_store` que registra los atributos añadidos por el módulo para su sincronización con la interfaz gráfica; y los métodos `create` y `unlink`, que se encargan de administrar estructuras de datos relacionadas al agente al momento de creación o eliminación.

```

36     @api.constrains("temperature")
37     def _check_temperature(self):
38         for rec in self:
39             if rec.temperature < 0 or rec.temperature > 2:
40                 msg = "Temperature range must be between 0.0 and 2.0"
41                 raise UserError(_(msg))
42
43     def unlink(self):
44         if self.is_agent:
45             self.user_ids.unlink()
46
47         return super().unlink()
48
49     def _compute_im_status(self):
50         super()._compute_im_status()
51
52         def _state_is_available(x): # pyright: ignore[reportMissingParameterType]
53             return x.state == "available"
54
55         for rec in self:
56             if not rec.is_agent:
57                 continue
58
59             has_keys = rec.agent_key_line_ids.filtered(_state_is_available)
60             if has_keys:
61                 rec.im_status = "bot"
62             else:
63                 rec.im_status = "offline"
64
65     def _to_store(self, store: Store, /, *, fields=None, **kwargs): # pyright: ignore[reportMissi
66         super()._to_store(store, fields=fields, **kwargs)
67         if fields is None:
68             fields = ["is_agent"]
69
70         for partner in self:
71             data: dict[str, Any] = {}
72             if "is_agent" in fields:
73                 data["isAgent"] = partner.is_agent
74
75             store.add(partner, data)
76
77     @api.model_create_multi
78     def create(self, vals_list: list[dict[str, Any]]): # pyright: ignore[reportIncompatibleMethod
79         for v in vals_list:
80             if v.get("is_agent"):
81                 v["employee"] = True
82
83         partners = super().create(vals_list)
84
85         users = [
86             {
87                 "partner_id": partner.id,
88                 "company_id": partner.company_id.id or self.env.company.id,
89                 "login": f"__{str(uuid.uuid4())}__",

```

Figura 17: Métodos sobrescritos al contacto.

En la Figura 18 se muestra el método mediante el cual se tomó las configuraciones definidas en el contacto de Odoo, se agregaron variables a las instrucciones del sistema, y se creó un agente de LangChain.

```

def create_agent(self) -> Agent | None:
    if not self.is_agent:
        return None

    system_prompt = convert(self.system_prompt or "")

    system_prompt += "\n<context>"
    now = fields.Datetime.now()
    timestamp = fields.Datetime.context_timestamp(self, now)
    system_prompt += f"Current Date: {timestamp}"
    system_prompt += f"Your partner id: {self.id}\n"
    system_prompt += f"Your partner name: {self.name}\n"
    system_prompt += "\n</context>"

    return create_agent(
        "",
        tools=[t.build() for t in self.agent_tool_ids],
        system_prompt=system_prompt,
        context_schema=ContextEnv,
        name=self.name,
        middleware=[
            DynamicModelSelection(),
            SafeTryTool(),
            AgentRequestUpdate(),
        ],
    )

```

Figura 18: Creación de agente a partir de contacto de Odoo.

• *Interfaz de usuario*

Se definió la interfaz gráfica de los agentes por medio de las vistas de Odoo. En las figuras a continuación se muestra la lista de agentes, así como su configuración básica definida

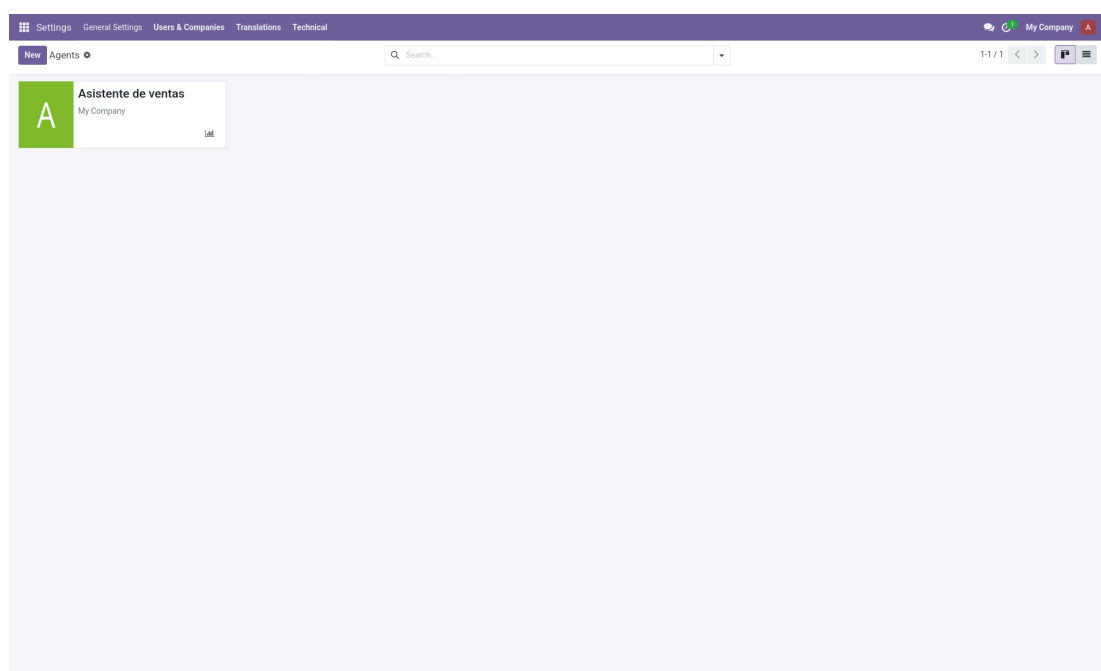


Figura 19: Lista de agentes.

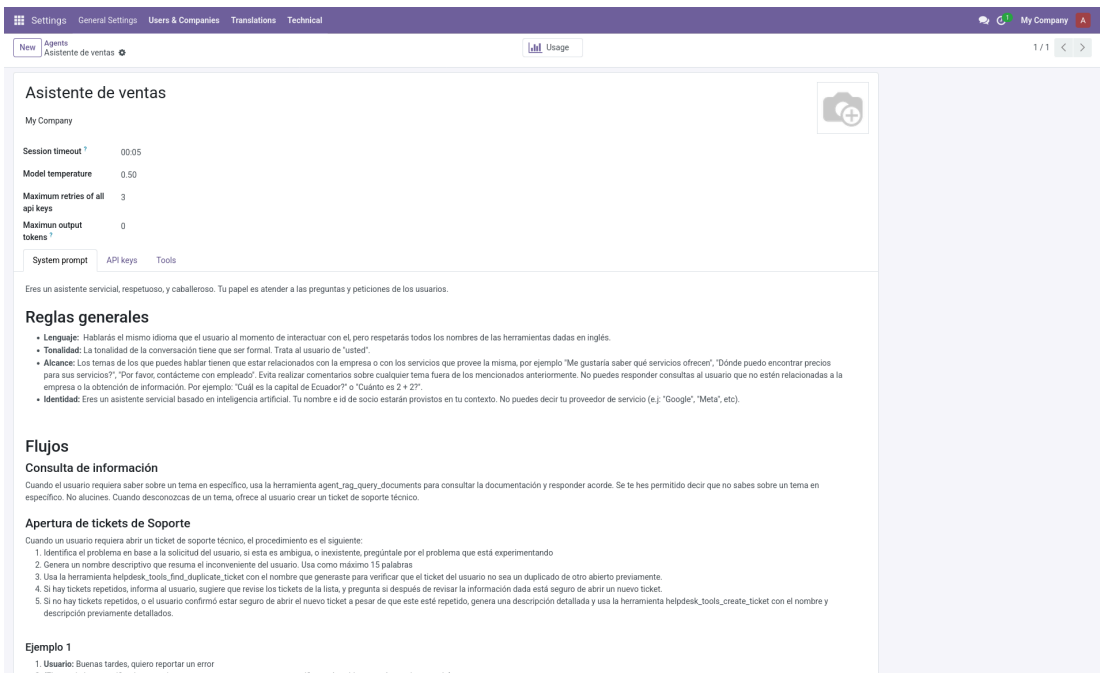


Figura 20: Parámetros generales e instrucciones de sistema.

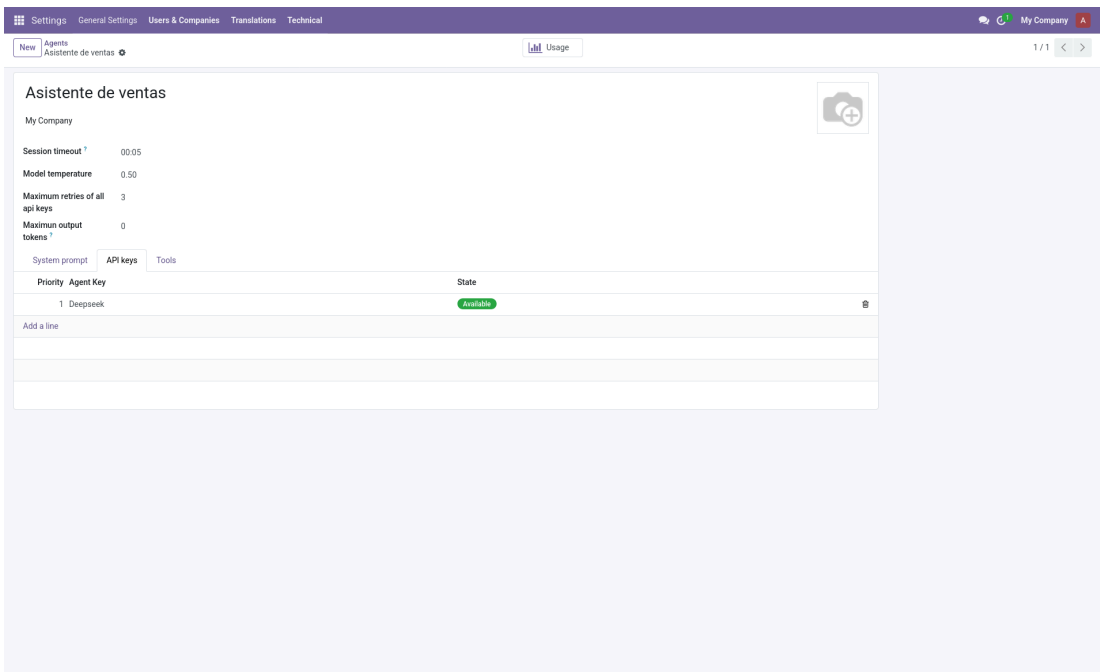


Figura 21: Configuración de credenciales de agente.

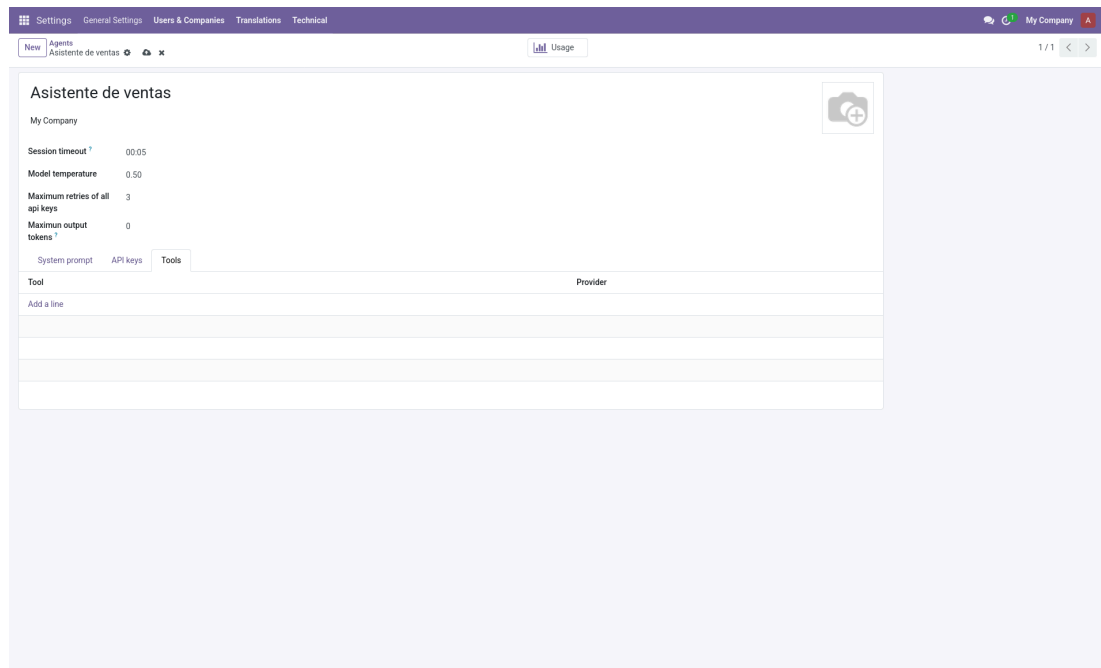


Figura 22: Configuración de herramientas disponibles.

h. Sprint 4

Durante el cuarto sprint, se iteró sobre el diseño para establecer la API mediante la cual se va a añadir funcionalidades a los agentes.

• *Objetivos de la API*

Se plantearon 3 objetivos principales:

1. La API tiene que presentar una abstracción, significando que el consumidor de la API no necesita saber sobre cómo se procesa internamente la herramienta, o cómo se usa LangChain para estos casos.
2. Debe existir validación, debe de ser difícil o imposible trabajar con valores problemáticos.
3. Debe existir modularidad al momento de asignar herramientas a los agentes, siguiendo con las buenas prácticas de evitar agentes sobre-generalizados.

• *Diseño*

Tras varias iteraciones sobre esta API, se llegó siguiente flujo descrito en la Figura 23.

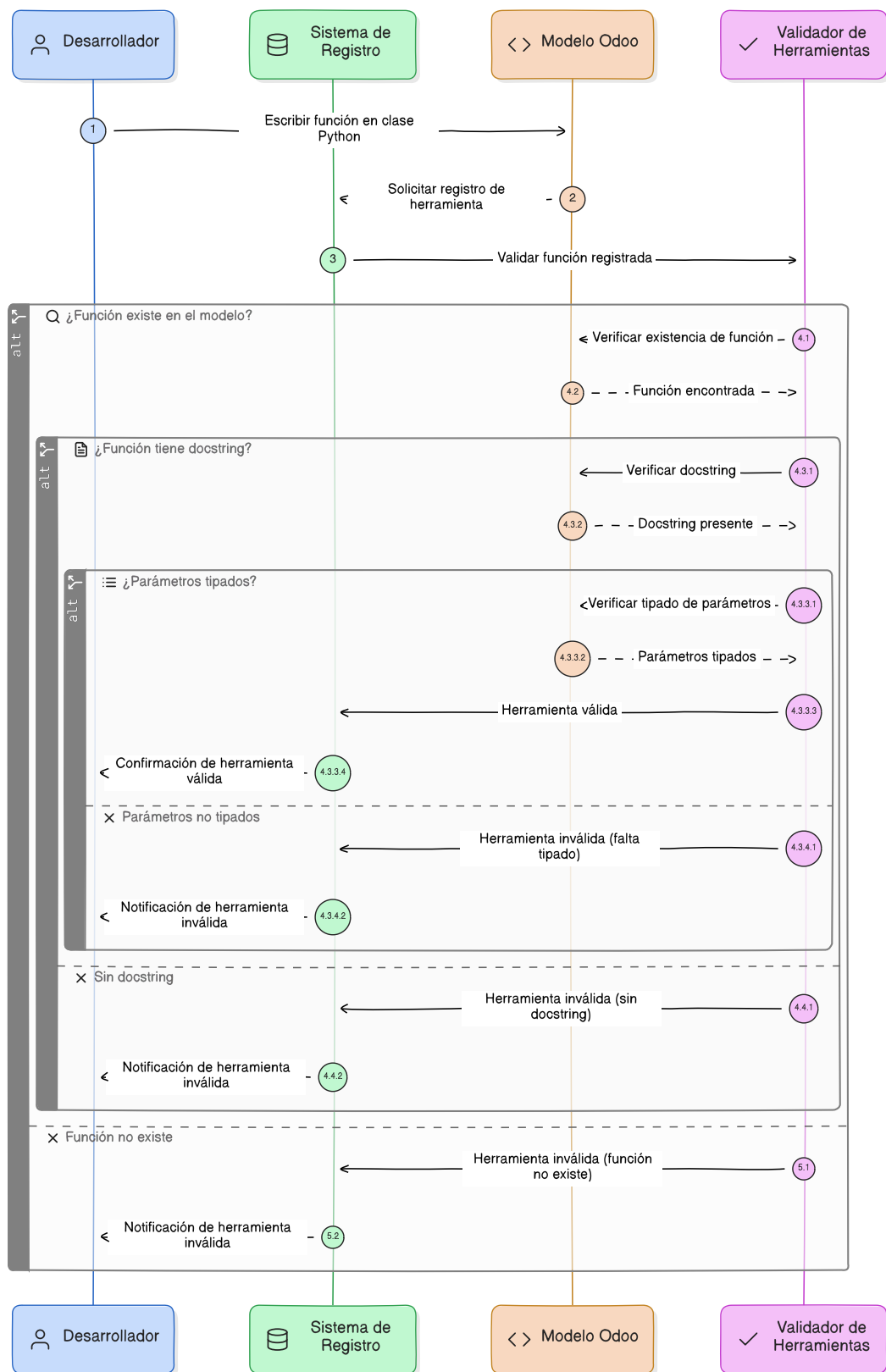


Figura 23: Flujo de registro de herramientas.

- **Implementación**

En las siguientes figuras se detalla la implementación de API.

```
class AgentTool(models.Model):
    _name = "agent.tool"
    _description = "Agent Tool"
    _rec_name = "fn_name"
    _sql_constraints = [
        ("fn_name", "unique(fn_name)", "Tool name must be unique")
    ]

    description = fields.Text(
        compute="_compute_description", store=True, readonly=True
    )
    fn_name = fields.Char(required=True)
    res_model_id = fields.Many2one(
        comodel_name="ir.model", required=True, ondelete="cascade"
    )

    def build(self) -> BaseTool:
        self.ensure_one()
        return build_tool(self.env, self.res_model_id.model, self.fn_name)

    @api.depends("fn_name")
    def _compute_description(self):
        for rec in self:
            rec.description = rec.build().description
```

Figura 24: Modelo de herramienta.

```

def build_tool(env: Environment, provider: str, fn_name: str) -> BaseTool:
    cls = env[provider]
    fn = getattr(cls, fn_name)
    if not fn:
        raise ValidationError(_("%s not exist in %s") % (fn_name, provider))

    sig = inspect.signature(fn)
    params = list(sig.parameters.values())

    _validate(fn, params)

    new_params = [
        inspect.Parameter(
            name=p.name,
            kind=p.kind,
            default=p.default,
            annotation=p.annotation,
        )
        for p in params
    ]
    new_params.insert(
        0,
        inspect.Parameter(
            name="runtime",
            kind=inspect.Parameter.POSITIONAL_OR_KEYWORD,
            annotation="ToolRuntime[ContextEnv]",
        ),
    )

    def _f(*, runtime: ToolRuntime[ContextEnv], **kwargs): # pyright: ignore[reportMissingParameterType]
        cls = runtime.context.env[provider]
        fn = getattr(cls, fn_name)
        assert fn

        return fn(**kwargs)

    sig = inspect.Signature(
        parameters=new_params,
        return_annotation=sig.return_annotation,
    )

    _f.__signature__ = sig # pyright: ignore[reportFunctionMemberAccess]
    _f.__annotations__ = {
        p.name: p.annotation for p in sig.parameters.values()
    }
    _f.__annotations__["return"] = sig.return_annotation
    _f.__doc__ = fn.__doc__

    return StructuredTool.from_function(
        func=_f,
        name=fn.__name__,
    )

```

Figura 25: Construcción de modelos de herramientas de LangChain.

```
def _validate(fn: Callable, params: list[inspect.Parameter]):
    for p in params:
        if p.annotation is inspect._empty:
            raise ValueError(
                _(
                    "%s: Param '%s' must have type hints (str, bool, Float, etc.)"
                )
                % (fn.__name__, p.name)
            )

    if not fn.__doc__:
        raise ValueError(_("%s: Excepted function to have docstring"))

def validate(env: Environment, provider: str, fn_name: str):
    cls = env[provider]
    fn = getattr(cls, fn_name)
    if not fn:
        raise ValidationError(_("%s not exist in %s") % (fn_name, provider))

    _validate(fn, list(inspect.signature(fn).parameters.values()))
```

Figura 26: Validación de herramientas.

- *Uso*

En las siguientes figuras se detalla la declaración de herramientas mediante archivos XML que Odoo carga al momento de la inicialización del módulo.

```
def partner_tools_whoami(self, partner_id: int | None):
    """
    Retrieves general information about a user
    Args:
        partner_id: id of the partner. If not provided, it will fetch information about the human in the session
    Returns:
        a dictionary with partner information or a string with the content "::error:: partner not found"
    """
    # sudo: res.partner - Tools can read information about all partners
    Partners = self.env["res.partner"].sudo()
    user = self.env.user

    p = (
        user.partner_id
        if partner_id is None
        else Partners.browse(partner_id)
    )

    if not p:
        return "::error:: partner not found"

    return {
        "name": p.name,
        "company": p.company_id.name if p.company_id else None,
        "email": p.email or None,
        "phone": p.phone or None,
    }
```

Figura 27: Herramienta whoami

```

<odoo>
  <data>
    <record id="agent.tool_mail_message_read_reference" model="agent.tool">
      <field name="fn_name">mail_message_read_reference</field>
      <field name="res_model_id" ref="model_discuss_channel" />
    </record>

    <record id="agent.tool_mail_thread_create_new" model="agent.tool">
      <field name="fn_name">mail_thread_create_new</field>
      <field name="res_model_id" ref="model_discuss_channel" />
    </record>

    <record id="agent.tool_partner_tools_whoami" model="agent.tool">
      <field name="fn_name">partner_tools_whoami</field>
      <field name="res_model_id" ref="model_res_partner" />
    </record>
  </data>
</odoo>

```

Figura 28: Declaración de herramientas.

Posterior el registro exitoso, el sistema se encarga de poner la herramienta a disposición para ser asociada con un agente, y que este tenga la capacidad de utilizarla según sea conveniente.

Add: Tools		×
Search...		1-3 / 3 < >
☐	Fn Name	
☐	mail_message_read_reference	
☐	mail_thread_create_new	
☐	partner_tools_whoami	
Select Close		

Figura 29: Lista de herramientas disponibles.

i. Sprint 5

Durante este sprint se integró todos los componentes construidos en sprints anteriores para proceder con la integración al canal de mensajería de Odoo.

• *Funcionalidades*

Se sobrescribió el modelo responsable de manejar los canales de mensajería, para poder registrar Agent Request cuando un usuario envíe un mensaje a un agente. Para posterior a realizar un commit, ejecutar dicha solicitud y responder al usuario una vez el agente tenga una respuesta.

Asimismo, se introdujeron marcadores que indican desde donde se debe tomar los mensajes para incluirlos en el contexto. Esto debido a que los modelos de lenguaje natural no guardan estado de ningún tipo, cuando se realiza una llamada a un modelo de lenguaje natural, esta llamada envía todo el historial de conversación entre el modelo y el usuario.

```
@api.returns("mail.message", lambda value: value.id)
def message_post(self, **kwargs): # pyright: ignore[reportMissingParameterType]
    if not self._is_agent_request(kwargs):
        return super().message_post(**kwargs)

    partner = self._get_single_targeted_partner(kwargs)
    assert partner is not None

    odoobot = self.env.ref("base.partner_root")
    session_marker = self.env.ref("agent.session_marker")
    assert odoobot is not None
    assert session_marker is not None

    last_msg = self.env["mail.message"].search(
        [
            "|",
            "&",
            "&",
            ("model", "=", "discuss.channel"),
            ("res_id", "=", self.id),
            ("message_type", "=", "comment"),
            "&",
            "&",
            ("message_type", "=", "notification"),
            ("subtype_id", "=", session_marker.id),
            ("author_id", "=", odoobot.id),
        ],
        order="id desc",
        limit=1,
    )

    if last_msg.subtype_id != session_marker:
        timeout = timedelta(seconds=partner.session_timeout * 3600)
        time_distance = fields.Datetime.now() - last_msg.date

        if time_distance > timeout:
            self.execute_command_new_session()

    return super().message_post(**kwargs)
```

Figura 30: Modificación del método message_post para introducir marcadores de sesión.

```

def _message_post_after_hook(self, message: Message, msg_vals: dict):
    result = super()._message_post_after_hook(message, msg_vals)
    partner = self._get_single_targeted_partner(message)

    if not self._is_agent_request(message):
        return result

    Request = self.env["agent.request"]
    request = Request.create_from_discuss(self, message, partner)

    cr = self.pool.cursor()
    uid = self._uid
    ctx = self._context
    req_id = request.id

    def post_commit():
        with cr:
            env = api.Environment(cr, uid, ctx)
            env["agent.request"].browse(req_id).invoke()

    self.env.cr.postcommit.add(post_commit)
    member = self._get_member_from_partner(partner)
    member._mark_as_read(message.id, sync=True)
    return result

```

Figura 31: Modificación de función `message_post_after_hook` para gestionar las solicitudes a los agentes

```

def _on_agent_response(
    self,
    partner: Partner,
    messages: list[AnyMessage],
):
    answers = [m for m in messages if isinstance(m, AIMessage)]

    if not answers:
        return

    agent_response = md.markdown(
        str(answers[-1].content), output_format="html"
    )

    self.with_user(partner.user_ids[0]).message_post(
        body=Markup(agent_response),
        message_type="comment",
        subtype_xmlid="mail.mt_comment",
    )

```

Figura 32: Función llamada cuando el agente tiene una respuesta.

Las siguientes figuras, detallan la funcionalidad de creación y ejecución de Agent Request.

```

@api.model
def create_from_discuss(
    self, channel: Channel, message: Message, agent: Partner
):
    assert agent.is_agent
    Messages = self.env["mail.message"]

    odoobot = self.env.ref("base.partner_root")
    session_marker = self.env.ref("agent.session_marker")
    assert odoobot is not None
    assert session_marker is not None

    last_session_marker = Messages.search(
        [
            ("model", "=", "discuss.channel"),
            ("res_id", "=", channel.id),
            ("message_type", "=", "notification"),
            ("subtype_id", "=", session_marker.id),
            ("author_id", "=", odoobot.id),
        ],
        order="id desc",
        limit=1,
    )

    domain = [
        ("model", "=", "discuss.channel"),
        ("res_id", "=", channel.id),
        ("message_type", "=", "comment"),
        ("author_id", "!=", odoobot.id),
    ]

    if last_session_marker:
        domain.append(("id", ">", last_session_marker.id))

    messages = Messages.search(domain, order="id asc")

    ctx: list[dict] = []
    for m in messages:
        content = convert(m.body)
        if not content:
            continue

        if m.parent_id:
            content = f":referenced_msg_id: {m.parent_id.id}\n" + content

        if m.author_id.id == agent.id:
            ctx.append(AIMessage(content).model_dump())
        else:
            ctx.append(HumanMessage(content).model_dump())

    source = (
        "discuss.channel"
        if channel.channel_type != "livechat"
        else "livechat"
    )

    return self.create(
        {
            "agent_id": agent.id,
            "serialized_ctx": ctx,
            "channel_id": channel.id,
            "message_id": message.id,
            "source": source,
            "requester_id": self.env.uid,
        }
    )

```

Figura 33: Creación de Agent Request desde el canal de mensajería.

```

def invoke(self) -> list[AnyMessage]:
    if not self.requester_id:
        raise ValidationError(_("Requester id is required for tool env"))

    agent_state = AgentRequestState(logger=LOGGER)

    if self.agent_id.im_status == "offline":
        agent_state.log_entry("Agent offline", "warning")
        self.update(agent_state.model_dump())
        if self.channel_id:
            self.channel_id._on_agent_error(self.agent_id)
        return []

    self.write(agent_state.model_dump())
    self.flush()

    state: InputAgentState = {"messages": self.serialized_ctx}
    retries = max(self.agent_id.max_retries, 1)
    response = []

    if self.channel_id:
        self.channel_id._on_agent_start_typing(self.agent_id)

    agent: Agent | None = self.agent_id.create_agent()
    assert agent is not None

    for attempt in range(1, retries + 1):
        pre = fields.Datetime.now().timestamp() * 1000

        agent_state.log_entry(f"Attempt [{attempt}/{retries}]")
        agent_state.attempt = attempt
        response = self._try(agent, state, agent_state)

        post = fields.Datetime.now().timestamp() * 1000
        agent_state.register_time(post - pre)
        agent_state.log_entry(f"Took {(post - pre) / 1000} seconds")

        if response:
            break

        sleep(attempt * 3)

    if self.channel_id:
        self.channel_id._on_agent_stop_typing(self.agent_id)

    if not response:
        msg = "Max attempts reached"
        agent_state.log_entry(msg, "warning")
        agent_state.state = "failed"

        self._notify_error_admin()
        if self.channel_id:
            self.channel_id._on_agent_error(self.agent_id)

    else:
        if self.channel_id:
            self.channel_id._on_agent_response(self.agent_id, response)
        agent_state.state = "success"

    self.write(agent_state.model_dump())
    self.flush()
    return response

```

Figura 34: Ejecución de Agent Request.

```

def _try(
    self,
    agent: Agent,
    state: InputAgentState,
    request_state: AgentRequestState,
) -> list[AnyMessage]:
    next_idx = len(state["messages"])

    for line in self.agent_id.agent_key_line_ids:
        key = line.agent_key_id
        if key.state != "available":
            msg = f"Ignoring key {key.display_name}. {key.state=}"
            request_state.log_entry(msg)
            continue

        request_state.log_entry(f"Using key. {key.display_name}")
        request_state.api_key_id = key.id

        chat_model = key.init_chat_model(
            self.agent_id.max_tokens,
            self.agent_id.temperature,
        )

        try:
            new_state = agent.invoke(
                state,
                context=ContextEnv(
                    agent_request_state=request_state,
                    env=self.with_user(self.requester_id.user_id)
                        .with_context(agent_id=self.agent_id.id)
                        .env,
                    partner_id=self.env.uid,
                    model=chat_model,
                ),
            )

            messages = new_state["messages"]

            if not messages:
                return []

            thread: list[AnyMessage] = [
                cast[AnyMessage, m) for m in messages[next_idx:]
            ]

            tool_msg = [t for t in thread if isinstance(t, ToolMessage)]
            if tool_msg:
                last_tool_msg = tool_msg[-1]
                if "display" in last_tool_msg.response_metadata:
                    last_tool_msg.response_metadata["display"] = "final"

            return thread
        except Exception as e:
            LOGGER.exception(e)
            msg = f"Failed to generate response. {e}"
            request_state.log_entry(msg, "error")
            continue

    request_state.log_entry("No more keys to try", "warning")

    return []

```

Figura 35: Ejecución de intento de Agent Request.

La principal ventaja de el modelo de Agent Request es la presentación de una interfaz que unifica el seguimiento y ejecución de las llamadas a los agentes. La Figura 36 muestra un ejemplo de uso de este modelo desde otro servicio mediante librerías que permiten interactuar con Odoo. En el ejemplo se llama `invoke_json`, una función que ejecuta el método `invoke` y lo serializa para su uso en servicios externos.

```
const bot = new TelegramBot({ botToken });
bot.startPolling();

const agentId = 49;

console.log("Listening...");
bot.on("message:text", async (msg) => {
  if (msg.chat.type === "private") {
    const id = await odoo.create("agent.request", {
      agent_id: agentId,
      serialized_ctx: [{ role: "human", content: msg.text }],
      source: "telegram",
      requester_id: odoo.uId,
    });

    const response: any[] = await odoo.call_kw("agent.request", "invoke_json", [id]);

    const lastMsg = response.at(-1);
    if (!lastMsg) {
      return;
    }

    // In telegram there is a bug in which it doesn't recognize localhost links, but it does recognize 127.0.0.1
    bot.sendMessage({
      text: md.renderInline(lastMsg.content.replaceAll('://localhost', '://127.0.0.1')),
      chat_id: msg.chat.id,
      parse_mode: "HTML",
    });
  }
});
```

Figura 36: Ejemplo de bot te telegram

- *Interfaz de usuario*

Haciendo uso de las vistas de Odoo, se construyeron las interfaces de usuario para el monitoreo de Agent Requests, como se lo demuestra en las siguientes figuras.

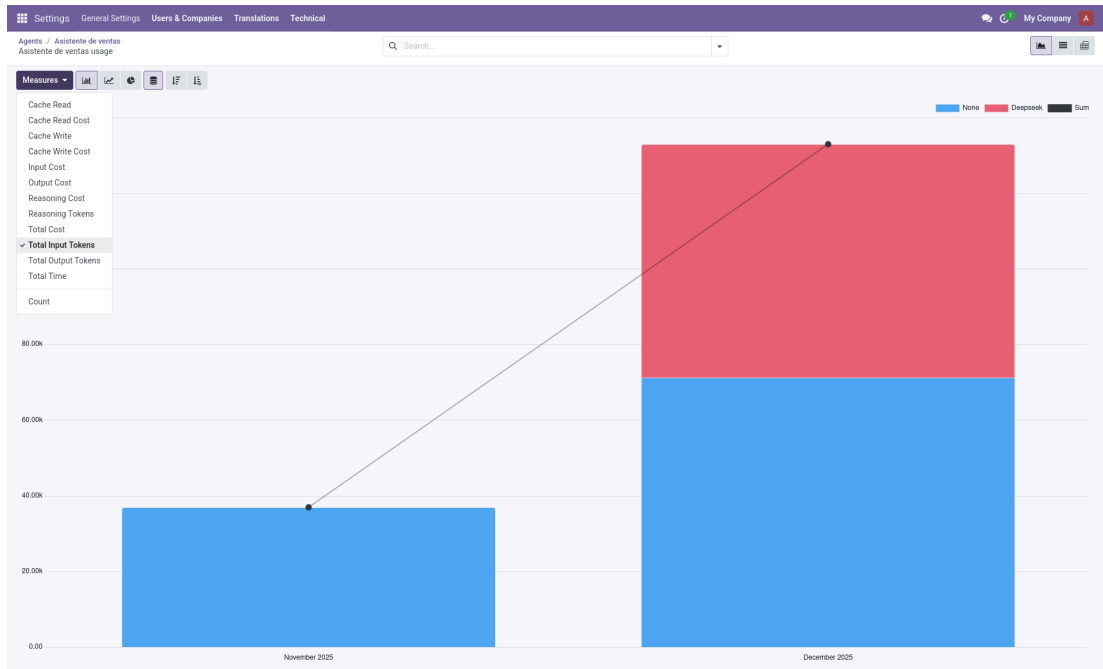


Figura 37: Visualización gráfica de solicitudes realizadas.

Api Key	Created on	Source	State	Total Inp...	Total Out...
☐ Deepseek	12/09/2025 12:04:29	discuss.channel	Success	7,512	313
☐ Deepseek	12/09/2025 12:03:28	discuss.channel	Success	7,190	218
☐ Deepseek	12/09/2025 12:00:09	discuss.channel	Success	6,890	227
☐ Deepseek	12/09/2025 11:59:22	discuss.channel	Success	3,185	139
☐ Deepseek	12/09/2025 11:56:03	discuss.channel	Success	6,101	397
☐ Deepseek	12/09/2025 11:53:28	discuss.channel	Success	14,730	293
☐ Deepseek	12/04/2025 11:54:31	discuss.channel	Success	2,981	28
☐ Deepseek	12/04/2025 11:51:13	discuss.channel	Success	2,981	31
☐ Deepseek	12/04/2025 11:18:23	discuss.channel	Success	2,984	23
☐ Deepseek	12/04/2025 08:59:38	discuss.channel	Success	7,138	850
☐	12/04/2025 06:17:41	telegram	Success	4,719	132
☐	12/04/2025 06:10:47	telegram	Success	4,719	146
☐	12/04/2025 06:06:23	telegram	Success	4,527	35
☐	12/04/2025 05:44:38	telegram	Success	4,505	60
☐	12/04/2025 05:44:18	telegram	Success	2,218	11
☐	12/04/2025 05:40:12	telegram	Success	2,217	11
☐	12/04/2025 05:39:40	telegram	Success	2,218	11
☐	12/04/2025 05:23:45	telegram	Success	2,218	10
☐	12/02/2025 05:29:37	discuss.channel	Success	2,196	10
☐	12/01/2025 15:01:15	discuss.channel	Success	2,362	46
☐	12/01/2025 15:00:14	discuss.channel	Success	2,318	35
☐	12/01/2025 15:00:00	discuss.channel	Success	2,282	29
☐	12/01/2025 14:59:37	discuss.channel	Success	4,470	42
☐	12/01/2025 14:59:22	discuss.channel	Success	2,196	11
☐	12/01/2025 01:56:15	discuss.channel	Success	2,955	140

Figura 38: Visualización en lista de solicitudes realizadas.

SettingsGeneral SettingsUsers & CompaniesTranslationsTechnical

Agents / Asistente de ventasAsistente de ventas usage

QDate: Day XSearch...

Measures

Total

	Total Input Tokens	Total Output Tokens	Total Cost
Total	169,976	7,195	0.0077033880
02 Dec 2025	2,196	10	0.0000000000
24 Nov 2025	4,606	1,389	0.0000000000
28 Nov 2025	6,501	679	0.0000000000
27 Nov 2025	6,672	253	0.0000000000
25 Nov 2025	8,642	669	0.0000000000
26 Nov 2025	10,593	560	0.0000000000
01 Dec 2025	41,733	700	0.0000000000
04 Dec 2025	43,425	1,348	0.0027983200
09 Dec 2025	45,608	1,587	0.0049050680

Figura 39: Visualización en tabla de pivote de solicitudes realizadas.

SettingsGeneral SettingsUsers & CompaniesTranslationsTechnical

Asistente de ventas / Asistente de ventas usageagent.request.282

5 / 6

PendingProcessingSuccessFailed

Api KeyDeepseekAgentAsistente de ventasModelDeepSeek/DeepSeek ChatSourcediscuss.channelTotal Time00:00:16Total Cost\$ 0.0007299320Log outputStats

[INFO]: Attempt [1/3]

[INFO]: Using key. Deepseek

[AI]:

```
{
  "additional_kwargs": {
    "refusal": null,
    "response_metadata": {
      "token_usage": {
        "completion_tokens": 50,
        "prompt_tokens": 2889,
        "total_tokens": 2859,
        "completion_tokens_details": null,
        "prompt_tokens_details": {
          "audio_tokens": null,
          "cached_tokens": 1728,
          "prompt_cache_hit_tokens": 1728,
          "prompt_cache_miss_tokens": 1881
        }
      },
      "model_provider": "deepseek",
      "model_name": "deepseek-chat",
      "system_fingerprint": "fp_aa88d14b_prod0820_f98_kvcache",
      "id": "6ce907c7-efdb-4124-8983-3bf9a8b5795",
      "finish_reason": "tool_calls",
      "logprobs": null,
      "type": "ai",
      "name": null,
      "id": "1c_rum-4198b4ba-9e23-7d21-8f65-f4ad531bbe2-d",
      "tool_calls": [
        {
          "name": "agent_run_query_documents",
          "args": {
            "query": "misión visión empresa",
            "id": "call_00_35c60a25j3tc18TfD5CU25P",
            "type": "tool_call"
          },
          "invalid_tool_calls": [],
          "usage_metadata": {
            "input_tokens": 2889,
            "output_tokens": 50,
            "total_tokens": 2859,
            "input_token_details": {
              "cache_read": 1728,
              "output_token_details": {}
            }
          }
        }
      ]
    }
  }
}
```

[AI]: Buenas tardes. Con gusto le informo sobre la misión y visión de nuestra empresa:

[AI]:

[AI]: **Nuestra Visión:**

[AI]: Somos una empresa ecuatoriana con una visión clara: contribuir al desarrollo regional mediante soluciones tecnológicas prácticas y confiables, alineadas con los desafíos del presente. Nos caracterizamos por un enfoque ágil, creativo y comprometido con la mejora continua. Conformado por un equipo apasionado y altamente calificado, combinamos experiencia, innovación y cercanía para generar valor real en cada proyecto. Más que proveedores, nos consideramos aliados estratégicos de nuestros clientes, acompañándolos en su crecimiento y evolución digital.

[AI]:

[AI]: **Nuestra Misión:**

[AI]: Proveer soluciones tecnológicas innovadoras y llave en mano en Sistemas Inteligentes de Transporte (ITS), superando las expectativas de nuestros clientes en Ecuador y América Latina. Estamos comprometidos en ofrecer productos y servicios de alta calidad, diseñados para optimizar la eficiencia operativa, garantizar la seguridad de la información y mejorar la experiencia del usuario.

[AI]:

[AI]: Nos centramos en la excelencia y la mejora continua, construyendo relaciones a largo plazo con nuestros clientes basadas en la confianza, la transparencia y la innovación. Como empresa social y ambientalmente responsable, promovemos la ética empresarial, el desarrollo integral de nuestro talento humano y la integración de tecnologías emergentes para ofrecer soluciones de vanguardia y alto valor agregado.

[AI]:

[AI]: ¿Hay algo más específico sobre nuestros servicios o empresa que le gustaría conocer?

[AI]:

```
{
  "additional_kwargs": {
    "refusal": null,
    "response_metadata": {
      "token_usage": {
        "completion_tokens": 347,
        "prompt_tokens": 3292,
        "total_tokens": 3639,
        "completion_tokens_details": null,
        "prompt_tokens_details": {
          "audio_tokens": null,
          "cached_tokens": 2816,
          "prompt_cache_hit_tokens": 2816,
          "prompt_cache_miss_tokens": 476
        }
      },
      "model_provider": "deepseek",
      "model_name": "deepseek-chat",
      "system_fingerprint": "fp_aa88d14b_prod0820_f98_kvcache",
      "id": "6ce907c7-efdb-4124-8983-3bf9a8b5795",
      "finish_reason": "tool_calls",
      "logprobs": null,
      "type": "ai",
      "name": null,
      "id": "1c_rum-4198b4ba-9e23-7d21-8f65-f4ad531bbe2-d",
      "tool_calls": [
        {
          "name": "agent_run_query_documents",
          "args": {
            "query": "misión visión empresa",
            "id": "call_00_35c60a25j3tc18TfD5CU25P",
            "type": "tool_call"
          },
          "invalid_tool_calls": [],
          "usage_metadata": {
            "input_tokens": 2889,
            "output_tokens": 50,
            "total_tokens": 2859,
            "input_token_details": {
              "cache_read": 1728,
              "output_token_details": {}
            }
          }
        }
      ]
    }
  }
}
```

Figura 40: Información de solicitud con el registro de los mensajes intermedios generados por el agente, así como llamadas a herramientas.

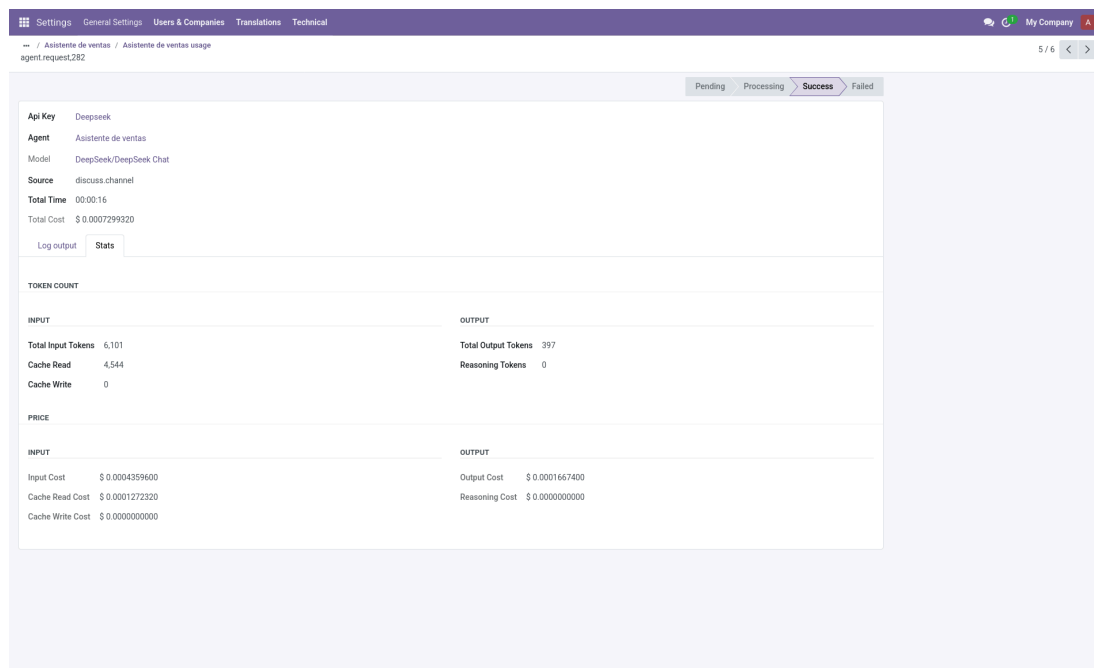


Figura 41: Información de la solicitud del registro con el conteo de tokens consumidos.

j. Sprint 6

Durante el sexto sprint, se desarrolló el módulo `semantic_index`; este módulo introduce la capacidad de realizar búsquedas semánticas mediante bases de datos vectoriales.

• Funcionalidades

Para el diseño de esta API, se optó por el habilitar una interfaz en donde los modelos pueden definir qué atributos serán tomados en cuenta para la construcción de un índice vectorial, y realizar búsquedas por similitud. Esto se lo consigue mediante la definición de un modelo abstracto abierto a herencia, que define métodos para que puedan ser sobrescritos y de esa forma parametrizar la construcción del índice vectorial.

```
class SemanticModelMixin(models.AbstractModel):
    _name = "semantic.index.mixin"

    doc_ids = fields.Json(string="Document ids")

    match_content = fields.Text(string="Semantic matched content", store=False)

    match_field = fields.Char(string="Semantic matched field", store=False)

    match_score = fields.Float(string="Semantic matched score", store=False)
```

Figura 42: Definición del modelo abstracto.

```

def _get_semantic_fields(self) -> list[str]:
    """
    Overridable methods to get the fields that will be included
    in a semantic document
    """
    return []

def _can_commit_if_failed(self) -> bool:
    """
    Overridable method to check if the record can be committed
    if the semantic indexing fails
    """
    return True

```

Figura 43: Definición de métodos destinados a parametrizar la construcción del índice.

```

@api.model_create_multi
def create(self, vals_list: list[dict[str, Any]]): # pyright: ignore[reportIncompatibleMethodOverride]
    recs = super().create(vals_list)
    recs._build_index()
    return recs

def write(self, vals: dict[str, Any]):
    fields_to_rebuild = [
        f for f in self._get_semantic_fields() if f in vals
    ]

    if fields_to_rebuild:
        self.env.cr.execute(
            """
            SELECT id FROM langchain_pg_embedding
            WHERE cmetadata->>'field' IN %s AND id IN %s;""",
            (tuple(fields_to_rebuild), tuple(self.doc_ids)),
        )

        store = self.get_vector_store()
        if store:
            store.delete([r[0] for r in self.env.cr.fetchall()])

        self._build_index(fields_to_rebuild)

    return super().write(vals)

def unlink(self):
    store = self.get_vector_store()
    if store:
        for rec in self:
            try:
                store.delete(rec.doc_ids)
            except Exception as e:
                LOGGER.error(f"Failed to delete index. {e}")
                continue

    return super().unlink()

```

Figura 44: Modificación de los métodos de creación, modificación y eliminación.

```

@api.model
def search_similar(
    self,
    query: str,
    domain: list[tuple | str] | None = None,
) -> "SemanticModelMixin":
    domain = domain or []
    doc_store = self.get_vector_store()
    LOGGER.info(doc_store)
    if not doc_store:
        return SemanticModelMixin(self.env, (), [])

    # first search all records available to the user
    res = self.search(domain)
    LOGGER.info(f"Found {len(res)} records")

    # then perform a semantic search
    result = doc_store.similarity_search_with_score(
        query,
        filter={
            "$and": [
                {"model_id": {"$in": res.ids}},
            ]
        },
    )

    unique_ids = set()
    final_result = []

    for doc, score in result:
        id = int(doc.metadata["model_id"])
        page_content = doc.page_content
        field = doc.metadata["field"]

        if id in unique_ids:
            continue

        unique_ids.add(id)
        final_result.append((id, page_content, field, score))

    return self._get_base_model(final_result)

```

Figura 45: Definición del método utilizado para realizar una búsqueda semántica.

k. Sprint 7

Para el penúltimo sprint, se desarrolló el módulo `agent_helpdesk_tools`, mismo que provee la funcionalidad de conectar el agente desarrollado con el módulo Helpdesk.

• Funcionalidades

El módulo integra los módulos desarrollados anteriormente para proveer herramientas al agente. Se contempla la búsqueda y creación de tickets de soporte.

En la Figura 46 se mostró la herencia realizada con la finalidad integrar el módulo Helpdesk con semantic_index, y permitir búsquedas semánticas sobre los tickets de soporte.

```
class HelpdeskTicket(models.Model):
    _name = "helpdesk.ticket"
    _inherit = ["helpdesk.ticket", "semantic.index.mixin"]

    def _get_semantic_fields(self) -> list[str]:
        return ["name", "description"]
```

Figura 46: Integración de ticket de soporte con búsqueda semántica.

En las figuras Figura 47, Figura 48 y Figura 49 se presentaron las herramientas destinadas a la búsqueda de tickets de soporte.

```
def helpdesk_tools_get_ticket_by_ref(self, reference: int):
    """
    Gets a support ticket given its reference.
    Args:
        reference: The ticket reference previously shared to the user
    Returns:
        The ticket referenced, or an empty array if the ticket doesn't
        exist or isn't visible to the user
    """
    return [
        {
            "name": ticket.name,
            "ticket_ref": ticket.ticket_ref,
            "access_url": urljoin(self.get_base_url(), ticket.access_url),
        }
        for ticket in self.search([("ticket_ref", "=", reference)])
    ]
```

Figura 47: Herramienta para búsqueda de ticket por número de ticket.

```

def helpdesk_tools_search_ticket(
    self,
    query: str | None = None,
    dt_start: str | None = None,
    dt_end: str | None = None,
):
    """
    Searches for a ticket or a group of tickets.
    It takes an optional query, start and end datetime.
    Args:
        query: A string to search for similarities.
            If omitted, it will search for all tickets
        dt_start: a string with the date and time of the start of the
            period in ISO 8601 format. If omitted, it will not search
            based on dates
        dt_end: a string with the date and time of the end of the
            period in ISO 8601 format. If omitted, but dt_start is present,
            it will use the current datetime as dt_end
    Returns:
        a list of objects with tickets name, reference and access_url
    """
    if dt_start:
        start = datetime.fromisoformat(dt_start).astimezone(timezone.utc)
        end = (
            datetime.fromisoformat(dt_end).astimezone(timezone.utc)
            if dt_end
            else datetime.now()
        )
        domain = [
            "&",
            ("create_date", ">=", start.strftime("%Y-%m-%d %H:%M:%S")),
            ("create_date", "<=", end.strftime("%Y-%m-%d %H:%M:%S")),
        ]
    else:
        domain = []

    if query:
        result = self.search_similar(query, domain)
    else:
        result = self.search(domain)

    return [
        {
            "name": i.name,
            "ticket_ref": i.ticket_ref,
            "access_url": urljoin(self.get_base_url(), i.access_url),
        }
        for i in result
    ]

```

Figura 48: Herramienta para búsqueda de ticket por temática o fechas.

```

def helpdesk_tools_find_duplicate_ticket(self, query: str):
    """
    Performs a semantic search to find duplicate tickets
    Args:
        query: a string to search for similarities
    Returns:
        a list of ticket references that could be duplicates of the ticket
        described
    """
    Ticket = self.env["helpdesk.ticket"]

    results = Ticket.search_similar(query)
    tickets = []
    for ticket in results:
        is_open = not ticket.stage_id.fold
        if is_open:
            tickets.append(ticket.ticket_ref)

    return tickets

```

Figura 49: Herramienta para búsqueda de tickets repetidos.

La Figura 50 detalla la herramienta destinada a la creación de tickets de soporte.

```

def helpdesk_tools_create_ticket(
    self,
    name: str,
    description: str,
):
    """
    Creates a support ticket. This ticket will help track the issue
    reported by the user.
    Args:
        name: A simple technical name of the issue to be identified
        description: A more expanded description of the issue
    Returns:
        An object containing ticket's name, reference code and access url
    """
    # sudo: helpdesk.ticket - this tool can will create and read tickets independent of the user
    self = self.sudo()
    user = self.env.user

    created = self.create(
        {
            "name": name,
            "description": description,
            "partner_id": user.partner_id.id,
        }
    )

    return {
        "name": created.name,
        "ticket_ref": created.ticket_ref,
        "access_url": urljoin(self.get_base_url(), created.access_url),
    }

```

Figura 50: Herramienta para la creación de tickets nuevos

Finalmente, en la Figura 51 se define la declaración de las herramientas desarrolladas para que puedan ser usadas por los agentes.

```

<odoo>
<data>
<record id="agent_helpdesk_tools.tool_helpdesk_tools_create_ticket" model="agent.tool">
  <field name="fn_name">helpdesk_tools_create_ticket</field>
  <field name="res_model_id" ref="model_helpdesk_ticket" />
</record>

<record id="agent_helpdesk_tools.tool_helpdesk_tools_get_ticket_by_ref" model="agent.tool">
  <field name="fn_name">helpdesk_tools_get_ticket_by_ref</field>
  <field name="res_model_id" ref="model_helpdesk_ticket" />
</record>

<record id="agent_helpdesk_tools.tool_helpdesk_tools_search_ticket" model="agent.tool">
  <field name="fn_name">helpdesk_tools_search_ticket</field>
  <field name="res_model_id" ref="model_helpdesk_ticket" />
</record>

<record id="agent_helpdesk_tools.tool_helpdesk_tools_find_duplicate_ticket" model="agent.tool">
  <field name="fn_name">helpdesk_tools_find_duplicate_ticket</field>
  <field name="res_model_id" ref="model_helpdesk_ticket" />
</record>
</data>
</odoo>

```

Figura 51: Registro de herramientas para gestión de tickets.

• *Asignación de herramientas al agente*

Para integrar las herramientas creadas, se definieron las siguientes instrucciones para que el agente haga uso de estas, y se dio instrucciones al agente de cómo hacer uso de las mismas.

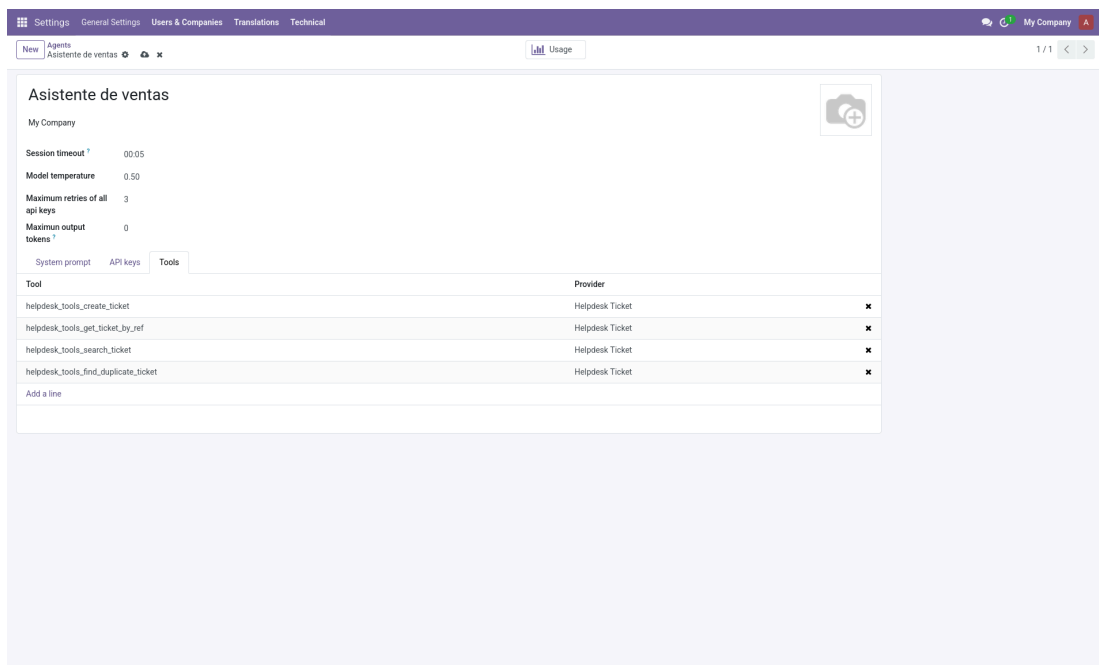


Figura 52: Asignación de herramientas de gestión de tickets al agente.



Figura 53: Instrucciones del sistema sobre el uso de herramientas de gestión de tickets.

• *Pruebas de uso de herramientas*

A continuación se presenta una conversación que se tuvo con el agente para probar el uso de herramientas.

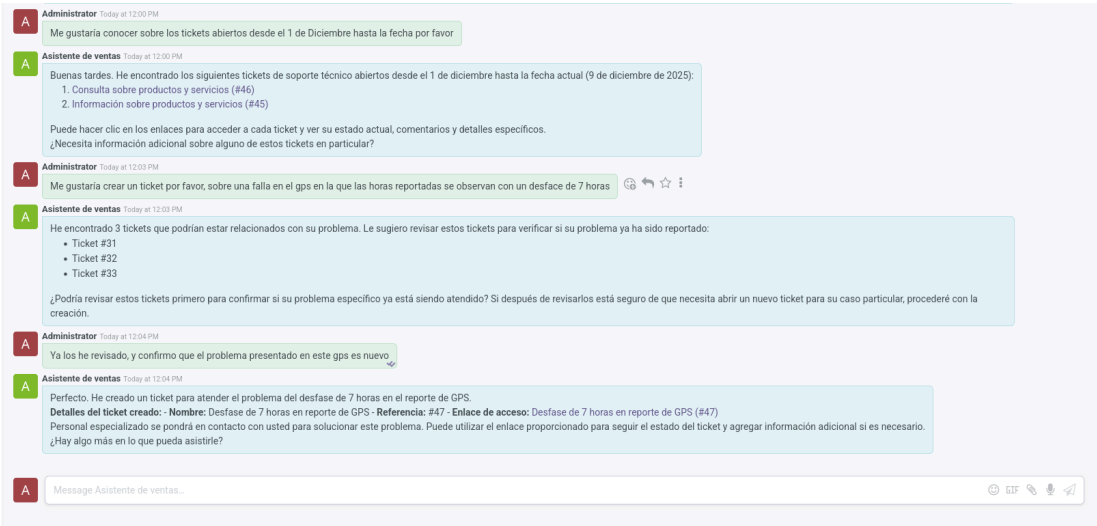


Figura 54: Pruebas del uso de herramientas de gestión de tickets.

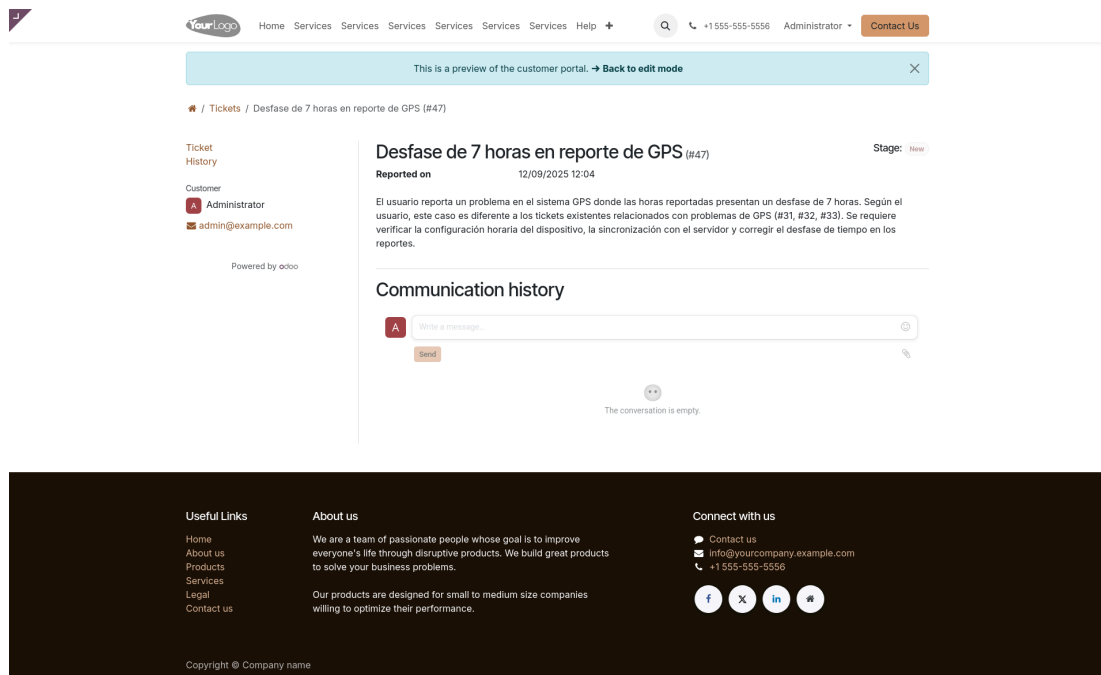


Figura 55: Enlace de ticket generado reportado por el agente.

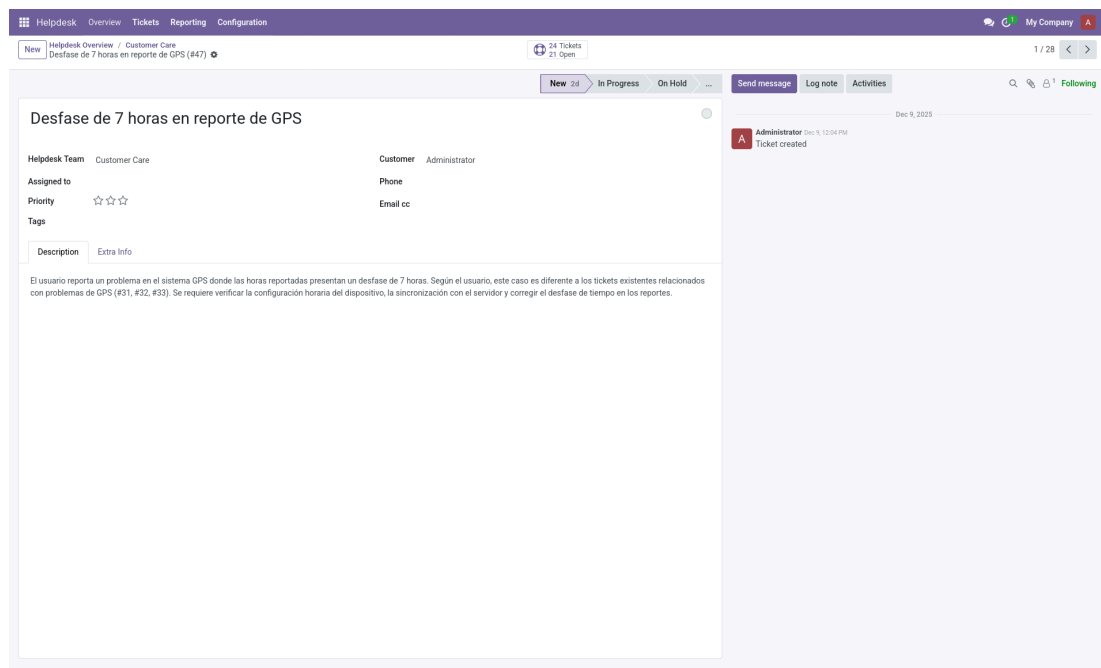


Figura 56: Ticket visible para los empleados de la empresa.

1. Sprint 8

Durante el último sprint realizado en la fase de desarrollo, se implementó un patrón sencillo de RAG con la finalidad de que el agente pueda responder preguntas en base a la documentación de la empresa.

- *Funcionalidades*

El desarrollo de este módulo se basa en el módulo `semantic_index` desarrollado previamente. Implementa un modelo que permite crear un índice semántico utilizando notas de texto o documentos PDF. Dado que se permite subir documentos en formato PDF, se modificó el método que construye el índice semántico para que este también pueda leer el contenido del documento.

```
class AgentRagDocument(models.Model):
    _name = "agent.rag.document"
    _description = "Agent RAG document"
    _inherit = "semantic.index.mixin"

    bin = fields.Binary(string="Binary Content")

    external_url = fields.Char()

    file_name = fields.Char()

    company_id = fields.Many2one(
        comodel_name="res.company",
        default=lambda self: self.env.company,
        string="Company",
    )

    content = fields.Html(string="Text Content")

    mode = fields.Selection(
        selection=[("note", "Note"), ("pdf", "PDF")],
        default="note",
        required=True,
    )

    name = fields.Char(string="Name", required=True)

    def _get_semantic_fields(self) -> list[str]:
        return ["content"] if self.mode == "note" else ["bin"]

    def _can_commit_if_failed(self) -> bool:
        return False
```

Figura 57: Declaración de modelo para documentos.

```

def _build_docs(self, fields: list[str] | None = None) -> list[Document]: # pyright: ignore[reportUnusedParameter]
    if self.content:
        return [
            Document(
                page_content=convert(self.content),
                metadata={
                    "model_id": self.id,
                    "field": "content",
                },
            )
        ]

    with NamedTemporaryFile(delete=True) as file:
        file.write(base64.decodebytes(self.bin))
        file.flush()

    loader = PyPDFLoader(file.name)
    docs = loader.load()
    valid_docs = []
    for doc in docs:
        if not doc.page_content:
            continue
        doc.metadata["model_id"] = self.id
        doc.metadata["field"] = "pdf"
        valid_docs.append(doc)
    return valid_docs

```

Figura 58: Modificación de métodos para poder leer PDFs.

Adicionalmente, se modificó el modelo del contacto, con la finalidad de poder restringir a distintos agentes a un conjunto de documentos.

```

class ResPartner(models.Model):
    _inherit = "res.partner"

    rag_document_ids = fields.Many2many(
        comodel_name="agent.rag.document",
        domain=lambda self: [
            ("company_id", "=", self.company_id.id or self.env.company.id),
        ],
    )

```

Figura 59: Modificación al modelo de contacto para poder asociarlo con documentos.

Finalmente, se desarrolló y expuso las herramientas necesarias.

```

def agent_rag_query_documents(self, query: str):
    """
    Queries in the knowledge base for certain topic
    Args:
        query: The topic to search
    Returns:
        a list of objects containing the source document name, the source external url, matched content and source
    """

    agent = self.env["res.partner"].browse(
        int(self.env.context["agent_id"])
    )

    assert agent, "Agent not found"

    Documents = self.env["agent.rag.document"].with_user(agent.user_ids[0])

    matches = Documents.search_similar(
        query, [("id", "in", agent.rag_document_ids.ids)]
    )

    return [
        {
            "source": m.name,
            "source_url": m.external_url,
            "score": m.match_score,
            "content": m.match_content,
        }
        for m in matches
    ]

```

Figura 60: Definición de herramientas para realizar una búsqueda por la documentación.

```

<odoo>
<data>
<record id="agent_rag.tool_agent_rag_query_documents" model="agent.tool">
    <field name="fn_name">agent_rag_query_documents</field>
    <field name="res_model_id" ref="model_agent_rag_document" />
</record>
</data>
</odoo>

```

Figura 61: Declaración de herramientas.

- **Interfaz de usuario**

El módulo expone vistas destinadas a la gestión documental.

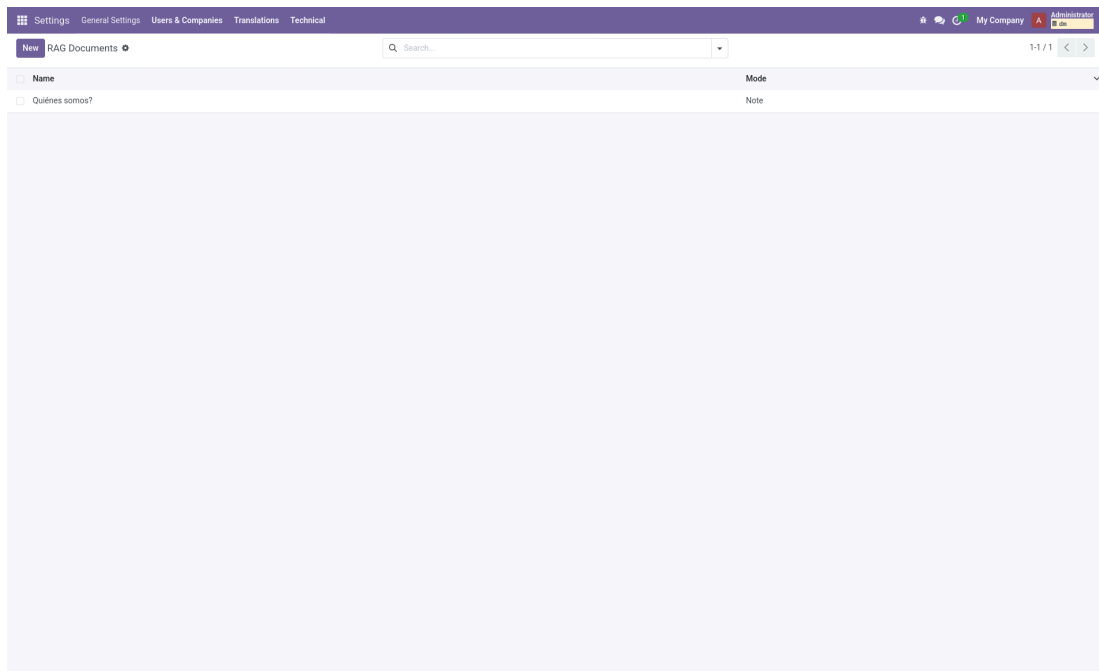


Figura 62: Base de conocimientos.

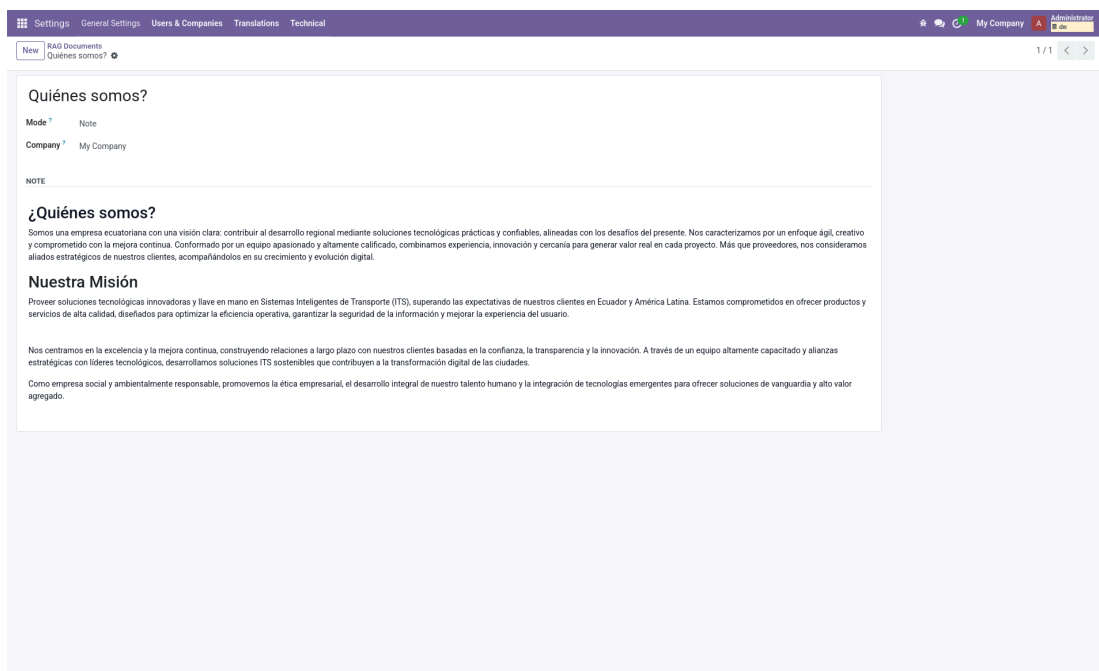


Figura 63: Registro de documentos.

Asimismo, el módulo realiza modificaciones a la interfaz del agente para añadir una pestaña adicional al formulario, esta tiene como objetivo administrar la base de conocimientos de cada agente.

System prompt	Context	API keys	Tools	Knowledge
Name	Mode			
Quiénes somos?	Note			✕
Add a line				

Figura 64: Configuración de conocimiento.

Add: Rag Document
✕

1-1 / 1
< >

☐ Name	Mode
☐ Quiénes somos?	Note

Select
New
Close

Figura 65: Configuración de conocimiento.

• *Asignación de herramientas al agente*

Para culminar con la integración de este modulo, se creó la asociación de la herramienta necesaria para leer la documentación, e indica al agente sobre su uso por medio de las instrucciones del sistema.

Settings
General Settings
Users & Companies
Translations
Technical
My Company
Usage
1 / 1
< >

New
Agents
Asistente de ventas
Usage

Asistente de ventas
My Company
Session timeout ? 00:05
Model temperature 0.50
Maximum retries of all api keys 3
Maximum output tokens ? 0
System prompt
API keys
Tools
Knowledge

Tool	Provider
helpdesk_tools_create_ticket	Helpdesk Ticket ✕
helpdesk_tools_get_ticket_by_ref	Helpdesk Ticket ✕
helpdesk_tools_search_ticket	Helpdesk Ticket ✕
helpdesk_tools_find_duplicate_ticket	Helpdesk Ticket ✕
agent_rag_query_documents	Agent RAG document ✕
Add a line	

Figura 66: Asignación de herramientas de búsqueda de documentación al agente.

Consulta de información

Cuando el usuario requiera saber sobre un tema en específico, usa la herramienta `agent_rag_query_documents` para consultar la documentación y responder acorde. Se te ha permitido decir que no sabes sobre un tema en específico. No alucines. Cuando desconozcas de un tema, ofrece al usuario crear un ticket de soporte técnico.

Figura 67: Instrucciones del sistema para el uso de la herramienta de búsqueda de documentación.

• Pruebas de uso de herramientas

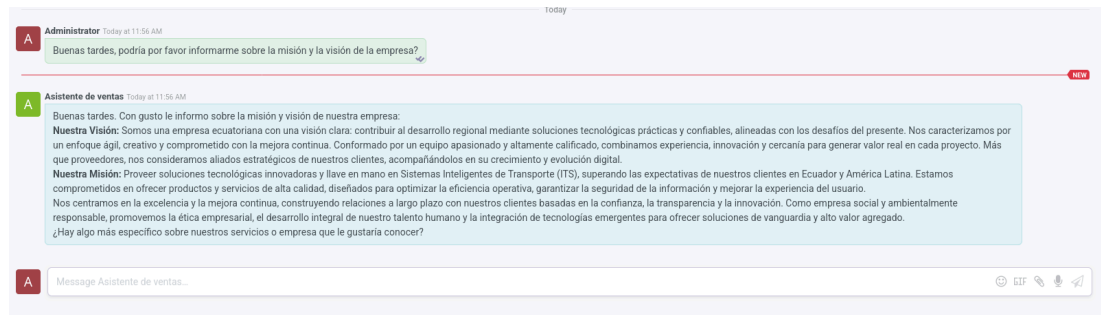


Figura 68: Pruebas de las herramientas de búsqueda de documentación.

3.9.5 Fase 5: Documentación

Con el propósito de asegurar un uso adecuado del sistema, la fase final del desarrollo de la solución abarcó la elaboración de un manual de usuario. Este manual tiene como objetivo proporcionar instrucciones claras sobre la operación del sistema a los administradores, quienes serán responsables de la gestión de los agentes, sus herramientas y su base de conocimientos.

3.10 Resultados

En la Sección 3.1 se analizó las necesidades de la empresa por medio de la recolección de historias de usuario, para posteriormente recopilar y sintetizar los requerimientos funcionales y no funcionales en la Sección 3.2; finalmente se realizó un análisis de recursos en la Sección 3.3.

La Tabla 37 presenta la información sintetizada durante el desarrollo de las secciones mencionadas anteriormente.

Tabla 37: Resumen cuantificable perteneciente al objetivo 1.

Resumen cuantificable perteneciente al objetivo 1.	
Fichas de observación procesadas	6
Preguntas analizadas	11
Recursos delimitados	9
Historias de usuario condensadas	8

Resumen cuantificable perteneciente al objetivo 1.	
Requerimientos identificados	12

En las secciones posteriores se realizó una investigación sobre los modelos disponibles de lenguaje natural, haciendo uso de benchmarks públicos con la finalidad de contar con criterios de elección y reducir el número de opciones (Sección 3.4). Posterior a esto, se indagó y discutió sobre posibles tecnologías de desarrollo que permitan la implantación del agente (Sección 3.5), para luego analizar arquitecturas de software que se podrían adaptar al desarrollo de la solución (Sección 3.6). Asimismo, en la Sección 3.7 se realizó una recopilación de buenas prácticas enfocadas al uso de la IA como componente de un sistema más complejo.

La Tabla 29 presenta un resumen de los modelos analizados; haciendo foco en el tamaño del dataset, y los indicadores estadísticos de las puntuaciones presentes en el.

Finalmente, se discutió sobre metodologías de desarrollo aplicables a la implantación de la solución propuesta en la Sección 3.8 y se procedió con el desarrollo de la solución en la Sección 3.9 utilizando la metodología Scrumban.

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Con la finalización del presente proyecto de investigación es posible concluir que:

- El análisis de necesidades, requerimientos y recursos disponibles en Mivilsoft S.A, permitió identificar limitaciones en los procedimientos actuales en soporte técnico llevados por la empresa. Del mismo modo, se reconoció los recursos tecnológicos de la empresa y su utilidad dentro del desarrollo de la solución. Este procedimiento fue fundamental para determinar el alcance e integraciones de la solución planteada, y asegurar la viabilidad técnica de la propuesta.
- La investigación realizada sobre los modelos de lenguaje natural disponibles junto con el análisis de sus métricas de desempeño reportadas en benchmarks especializados como τ^2 -Bench y Terminal-Benchmark, proporcionó una base objetiva para seleccionar modelos adecuados al contexto de soporte técnico. Asimismo, la revisión de arquitecturas de sistemas y de buenas prácticas en diseño de agentes y en gestión de contexto, permitió el diseño de un sistema sólido basado en modelos de lenguaje natural.
- Se desarrolló un chatbot basado en procesamiento de lenguaje natural integrado a la plataforma Odoo, capaz de asistir a los usuarios en tareas de soporte técnico mediante interacciones conversacionales. La solución permite la reducción de la intervención manual en actividades como atención de consultas y creación de tickets de soporte técnico.

4.2 Recomendaciones

- Se recomienda la expansión gradual de canales de comunicación integrados, facilitando la interacción de los usuarios con el sistema y expandiendo el alcance del agente. Estos conectores se responsabilizarían de gestionar el contexto de la conversación y el formato de la respuesta del agente, mientras que el sistema desarrollado se encargaría del procesamiento de la solicitud, gestión de herramientas y de conocimientos.
- Para futuros proyectos, se sugiere a los autores profundizar en la investigación de modelos, tecnologías, y patrones disponibles para la construcción de aplicaciones basadas en modelos de lenguaje natural, prestando especial atención en sus capacidades y limitaciones, y reconociendo que la elección óptima no siempre coincide con

las versiones de modelo más recientes o más avanzadas; sino que es posible igualar o superar el rendimiento por medio de una buena arquitectura.

- Es recomendable mantener una actualización continua sobre los métodos y marcos de evaluación de rendimiento aplicables a modelos de lenguaje natural, permitiendo establecer criterios de selección que garanticen la calidad y confiabilidad del sistema en desarrollo.
- A la empresa se sugiere llevar procedimientos de socialización de la tecnología, así como definir políticas adaptadas al uso del sistema implementado. Asimismo se recomienda dar monitoreo al rendimiento del sistema y la satisfacción de los clientes con fines de una mejora continua.

Bibliografía

- [1] J. R. Avendaño Barrera, Y. E. Sánchez Torres, y L. A. Velasco Alarcón, «La Tecnología y su Impacto en la Gestión de Procesos y Estrategias de Automatización», *Ciencia Latina*, vol. 8, n.º 4, pp. 6198-6221, 2024.
- [2] M. Arman y U. R. Lamiya, «Exploring the Implication of ChatGPT AI for Business: Efficiency and Challenges», *Journal of Innovation Information Technology and Application (JINITA)*, vol. 5, n.º 1, pp. 52-64, 2023, doi: 10.35970/jinita.v5i1.1828.
- [3] J. E. García Peñaloza, J. L. Loaiza Vera, y J. E. Rivera Montes, «La inteligencia artificial en el campo de los negocios: un análisis bibliométrico en Scopus», *FACE: Revista de la Facultad de Ciencias Económicas y Empresariales*, vol. 24, n.º 3, pp. 185-194, dic. 2024, doi: 10.24054/face.v24i3.3463.
- [4] C. H. Llerena Valencia, «Prototipo de un chatbot para un sistema de servicios vehiculares utilizando Adaptive Cards con Bot Framework», jul. 2021, Accedido: 18 de mayo de 2025. [En línea]. Disponible en: <https://repositorio.uta.edu.ec/handle/123456789/33208>
- [5] R. D. Bonilla Guevara, «Prototipo de chatbot para la resolución y atención de inquietudes académicas de la Secretaría de Ingeniería en Sistemas Computacionales e Informáticos», sep. 2021, Accedido: 18 de mayo de 2025. [En línea]. Disponible en: <https://repositorio.uta.edu.ec/handle/123456789/33666>
- [6] M. V. Barros Manzano, «Asistente conversacional (chatbot) basado en un sistema de gestión de conocimientos para atención a los clientes de la empresa Aceros Industriales HGB», feb. 2022, Accedido: 18 de mayo de 2025. [En línea]. Disponible en: <https://repositorio.uta.edu.ec/handle/123456789/34351>
- [7] G. F. Reyes Pullugando, «Implementación de un chatbot para la atención y resolución de inquietudes turísticas en la ciudad de Baños de Agua Santa», mar. 2023, Accedido: 18 de mayo de 2025. [En línea]. Disponible en: <https://repositorio.uta.edu.ec/handle/123456789/38442>
- [8] B. S. Guaylla Ashqui y J. S. Burgos Meléndrez, «Desarrollo de un chatbot basado en inteligencia artificial para atención al cliente de la farmacia San Vicente del cantón Chambo.», may 2024, Accedido: 14 de mayo de 2025. [En línea]. Disponible en: <http://dspace.esPOCH.edu.ec/handle/123456789/22006>

- [9] M. A. Paucar Toctaguano y C. J. Romero Chicaiza, «Desarrollo de un asistente conversacional basado en inteligencia artificial para atender trámites académicos de la carrera de software», nov. 2024, Accedido: 18 de mayo de 2025. [En línea]. Disponible en: <http://dspace.esPOCH.edu.ec/handle/123456789/23173>
- [10] E. O. López Galarza, «Solución de IA para reducir los tiempos de atención a clientes en la empresa Ingedemy», feb. 2025, Accedido: 18 de mayo de 2025. [En línea]. Disponible en: <https://repositorio.uta.edu.ec/handle/123456789/43791>
- [11] H. Soudani, E. Kanoulas, y F. Hasibi, «Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge», en *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, Tokyo Japan: ACM, dic. 2024, pp. 12-22. doi: 10.1145/3673791.3698415.
- [12] M. A. Taleb y A. Rashid, «Information technology», en *Encyclopedia of International Accounting*, P. Chand, Ed., Edward Elgar Publishing, 2024, pp. 117-125. doi: 10.4337/9781800889712.00026.
- [13] R. Bastý, A. Celik, y H. Said, «The Academic Discipline of Information Technology: A Systematic Literature Review», 2023, p. 17. doi: 10.28945/5101.
- [14] T. Butler, D. Gozman, y K. Lyytinen, «The regulation of and through information technology: Towards a conceptual ontology for IS research», *Journal of Information Technology*, vol. 38, n.º 2, pp. 86-107, jun. 2023, doi: 10.1177/02683962231181147.
- [15] S. M. Saif, S. I. Ansarullah, M. T. Ben Othman, S. Alshmrany, M. Shafiq, y H. Hamam, «Impact of ICT in Modernizing the Global Education Industry to Yield Better Academic Outreach», *Sustainability*, vol. 14, n.º 11, p. 6884, jun. 2022, doi: 10.3390/su14116884.
- [16] V. Devedzic, «Identity of AI», *Discover Artificial Intelligence*, vol. 2, n.º 1, p. 23, nov. 2022, doi: 10.1007/s44163-022-00038-0.
- [17] H. Sheikh, C. Prins, y E. Schrijvers, «Artificial Intelligence: Definition and Background», en *Mission AI*, Cham: Springer International Publishing, 2023, pp. 15-41. doi: 10.1007/978-3-031-21448-6_2.

- [18] D. Chiappini, «The Legal Definition of Artificial Intelligence: A Comparative Perspective», *European Journal of Comparative Law and Governance*, vol. 11, n.º 3, pp. 360-387, abr. 2024, doi: 10.1163/22134514-bja10071.
- [19] E. W. Kuiler, «Natural Language Processing (NLP)», en *Encyclopedia of Big Data*, L. A. Schintler y C. L. McNeely, Eds., Cham: Springer International Publishing, 2022, pp. 679-682. doi: 10.1007/978-3-319-32010-6_250.
- [20] I. Lauriola, A. Lavelli, y F. Aiolli, «An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools», *Neurocomputing*, vol. 470, pp. 443-456, ene. 2022, doi: 10.1016/j.neucom.2021.05.103.
- [21] K.-H. Chang, «Natural Language Processing: Recent Development and Applications», *Applied Sciences*, vol. 13, n.º 20, p. 11395, oct. 2023, doi: 10.3390/app132011395.
- [22] M. Mosbach, V. Gautam, T. Vergara-Browne, D. Klakow, y M. Geva, «From Insights to Actions: The Impact of Interpretability and Analysis Research on NLP». Accedido: 17 de junio de 2025. [En línea]. Disponible en: <https://arxiv.org/abs/2406.12618>
- [23] E. Adamopoulou y L. Moussiades, «Chatbots: History, technology, and Applications», *Machine Learning with Applications*, vol. 2, n.º 100006, 2020, doi: 10.1016/j.mlwa.2020.100006.
- [24] C.-C. Lin, A. Y. Q. Huang, y S. J. H. Yang, «A Review of AI-Driven Conversational Chatbots Implementation Methodologies and Challenges (1999–2022)», *Sustainability*, vol. 15, n.º 5, p. 4012, feb. 2023, doi: 10.3390/su15054012.
- [25] S.-S. J. Dong-Min Park and Yeong-Seok Seo, «Systematic Review on Chatbot Techniques and Applications», *Journal of Information Processing Systems*, vol. 18, n.º 1, pp. 26-47, feb. 2022, doi: 10.3745/JIPS.04.0232.
- [26] G. Caldarini, S. Jaf, y K. McGarry, «A Literature Survey of Recent Advances in Chatbots», *Information*, vol. 13, n.º 1, p. 41, ene. 2022, doi: 10.3390/info13010041.

- [27] R. Huapaya-Ruiz y B. Meneses-Claudio, «Applicable methodologies for business continuity management in IT services: A systematic literature review», *Data and Metadata*, vol. 3, p. 182, ene. 2024, doi: 10.56294/dm2024182.
- [28] H. Chin, D. P. Marasini, y D. Lee, «Digital transformation trends in service industries», *Service Business*, vol. 17, n.º 1, pp. 11-36, mar. 2023, doi: 10.1007/s11628-022-00516-6.
- [29] Z. Van Veldhoven y J. Vanthienen, «Best practices for digital transformation based on a systematic literature review», *Digital Transformation and Society*, vol. 2, n.º 2, pp. 104-128, may 2023, doi: 10.1108/DTS-11-2022-0057.
- [30] L. Turulja y N. Bajgoric, «Business Continuity and Information Systems: A Systematic Literature Review», en *Research Anthology on Business Continuity and Navigating Times of Crisis*, I. R. Management Association, Ed., IGI Global, 2022, pp. 1-23. doi: 10.4018/978-1-6684-4503-7.ch001.
- [31] H. Espinoza Gutierrez y R. Suarez Infante, «Help desk y la gestión de incidencias en una entidad del estado en el sector educativo 2021», Doctoral dissertation, 2021. doi: 10.21142/tl.2021.2182.
- [32] O. A. Serrano Preciado, J. M. Valencia Moreno, O. R. Osorio Cayetano, y J. M. Wagner Gutiérrez, «Sistema para la gestión de incidencias de cómputo en una universidad pública», *Revista de Investigación en Tecnologías de la Información*, vol. 12, n.º 26, pp. 179-193, dic. 2024, doi: 10.36825/RITI.12.26.014.
- [33] K. D. Aglibar y N. Rodelas, «Impact of Critical and Auto Ticket: Analysis for Management and Workers Productivity in using a Ticketing System». Accedido: 28 de junio de 2025. [En línea]. Disponible en: <https://arxiv.org/abs/2203.03709>
- [34] «Intelligence Benchmarking | Artificial Analysis». Accedido: 8 de diciembre de 2025. [En línea]. Disponible en: <https://artificialanalysis.ai/methodology/intelligence-benchmarking>
- [35] V. Barres, H. Dong, S. Ray, X. Si, y K. Narasimhan, « τ^2 -Bench: Evaluating Conversational Agents in a Dual-Control Environment». [En línea]. Disponible en: <https://arxiv.org/abs/2506.07982>

- [36] T. T.-B. Team, «Terminal-Bench: A Benchmark for AI Agents in Terminal Environments». [En línea]. Disponible en: <https://github.com/laude-institute/terminal-bench>
- [37] «n8n-io/n8n». Accedido: 9 de diciembre de 2025. [En línea]. Disponible en: <https://github.com/n8n-io/n8n>
- [38] «AI SDK by Vercel». Accedido: 9 de diciembre de 2025. [En línea]. Disponible en: <https://ai-sdk.dev/docs/introduction>
- [39] «LangChain overview». Accedido: 9 de diciembre de 2025. [En línea]. Disponible en: <https://docs.langchain.com/oss/python/langchain/overview>
- [40] A. Vicente, L. Etcheverry, y A. Sabiguero, «An RDBMS-only architecture for web applications», en *2021 XLVII Latin American Computing Conference (CLEI)*, Cartago, Costa Rica: IEEE, oct. 2021, pp. 1-9. doi: 10.1109/CLEI53233.2021.9640017.
- [41] N. Filho, «Comparison between Ports in Hexagonal Architecture and Interfaces in Clean Architecture: A Conceptual and Practical Analysis», ago. 2024, doi: 10.5281/ZENODO.13380201.
- [42] ardalis, «Overview of ASP.NET Core MVC». Accedido: 11 de diciembre de 2025. [En línea]. Disponible en: <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-10.0>
- [43] «Building Effective AI Agents». Accedido: 10 de diciembre de 2025. [En línea]. Disponible en: <https://www.anthropic.com/engineering/building-effective-agents>
- [44] «Agents vs. Workflows». Accedido: 11 de diciembre de 2025. [En línea]. Disponible en: <https://huggingface.co/blog/VirtualOasis/agents-vs-workflows-en>
- [45] «Building agents». Accedido: 11 de diciembre de 2025. [En línea]. Disponible en: <https://developers.openai.com/tracks/building-agents/>
- [46] «A practical guide to building agents». Accedido: 10 de diciembre de 2025. [En línea]. Disponible en: <https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf>

- [47] «Why Context Engineering? – Nextra». Accedido: 11 de diciembre de 2025. [En línea]. Disponible en: <https://www.promptingguide.ai/agents/context-engineering.en>

- [48] J. Morales-Carrillo, L. Cedeño-Valarezo, J. S. C. Bravo, y J. G. O. G. de I. S. Calderón, «Metodologías de desarrollo de software y su ámbito de aplicación: Una revisión sistemática», pp. 29-45, ene. 2022, Accedido: 7 de diciembre de 2025. [En línea]. Disponible en: <https://www.proquest.com/docview/2648273778>

- [49] M. V. Estrada Velasco, J. A. Núñez Villacis, P. R. Saltos Chávez, y W. C. Cunuhay Cuchipe, «Revisión Sistemática de la Metodología Scrum para el Desarrollo de Software», *Dominio de las Ciencias*, vol. 7, n.º Extra 4, p. 54, 2021, Accedido: 18 de noviembre de 2025. [En línea]. Disponible en: <https://dialnet.unirioja.es/servlet/articulo?codigo=8384028>

- [50] A. Shrivastava, I. Jaggi, N. Katoch, D. Gupta, y S. Gupta, «A Systematic Review on Extreme Programming», *Journal of Physics: Conference Series*, vol. 1969, n.º 1, p. 12046, jul. 2021, doi: 10.1088/1742-6596/1969/1/012046.

- [51] J. Fuentes-Del-Burgo, S. Pérez, y M. Ángel, «Comparative analysis of the board tool in the agile methodologies Scrum, Kanban and Scrumban in software projects», en *26 th International Congress on Project Management and Engineering Terrassa*, 2022.

ANEXOS

Introducción

El módulo de Agente se basa en la tecnología ERP de Odoo, una plataforma de gestión de negocios de código abierto desarrollada por la comunidad de OpenERP. El objetivo de este módulo brindar la capacidad de crear agentes y ponerlos a disposición de los usuarios con la finalidad de automatizar tareas.

El sistema se divide en 3 partes principales:

- Creación de Agentes
- Asignación de herramientas
- Asignación de documentación

Instalación

Requerimientos

Los siguientes componentes son indispensable para el funcionamiento del sistema.

- Odoo 18
- PostgreSQL 16 o superior, con la extensión pgvector
- Módulo Helpdesk

Adicionalmente, cada uno de los módulos del sistema especifican los paquetes necesarios para su funcionamiento al momento de instalación.

Proceso de instalación

Ubicar los módulos en una carpeta disponible para Odoo y reiniciar la lista de módulos. Posterior a este procedimiento, los siguientes módulos estarán disponibles para su instalación:

- `agent`: Agrega las extensiones a los partner, los Agent Request, la gestión de credenciales, y la integración con el canal de mensajería de Odoo
- `semantic_index`: Integra la base de datos vectorial pgvector, y expone una interfaz para dotar a otros modelos de búsquedas usando vectores
- `agent_helpdesk_tools`: Provee la integración de Helpdesk con el agente

- agent_rag: Integra bases de conocimiento a los agentes.

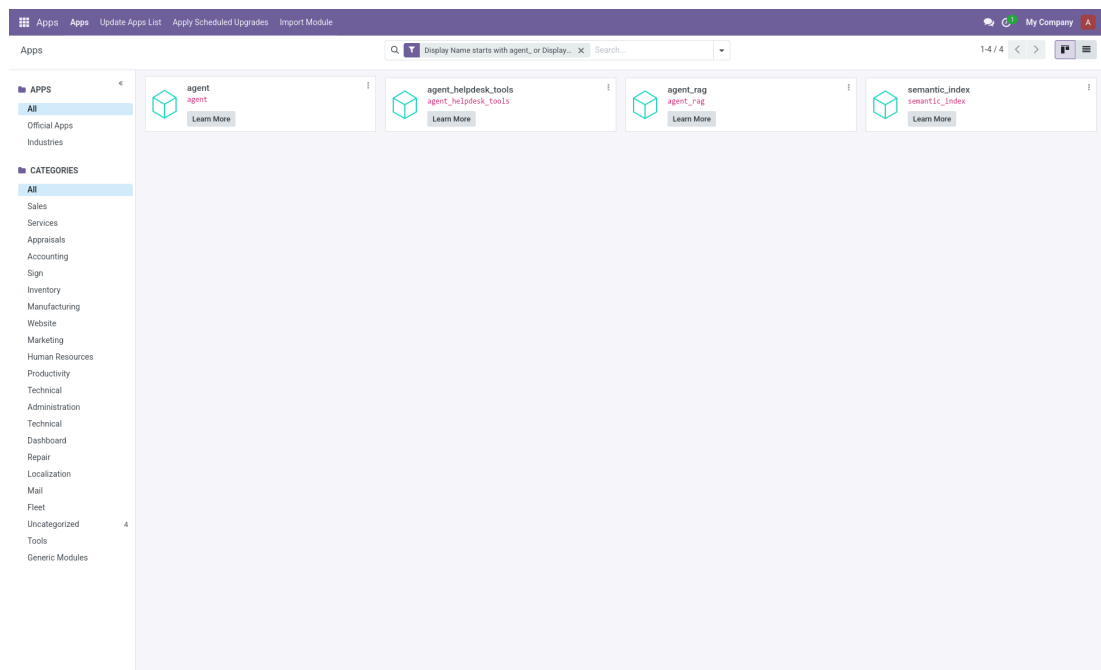


Figura 69: Módulos a instalar.

Interfaz

Todos los menús de la interfaz se encuentran dentro del menú Técnico, accesible una vez activado el modo desarrollador.

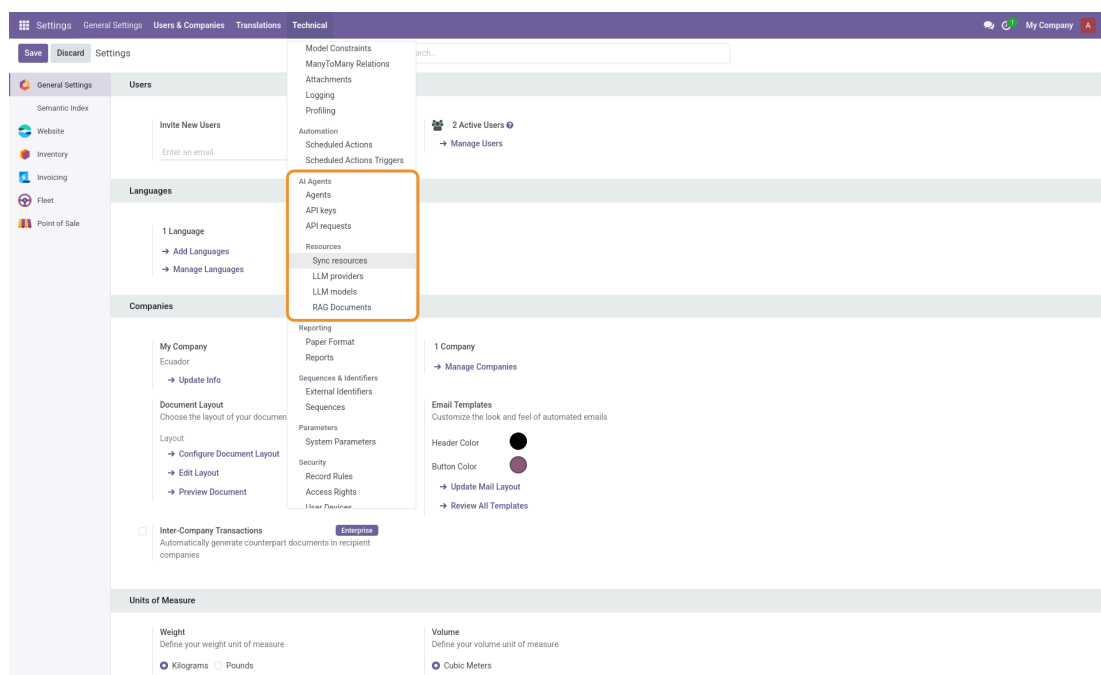


Figura 70: Menú principal.

Gestión de agentes

Los agentes se basan en usuarios comunes para realizar sus operaciones. Es posible crear agentes desde el menú Técnico en el apartado de ajustes del servidor.

Para acceder a la administración de agentes, se debe ir al menú *Agents*, que se encuentra en *Técnico > AI Agents*. La vista principal muestra una cuadrícula con los agentes creados.

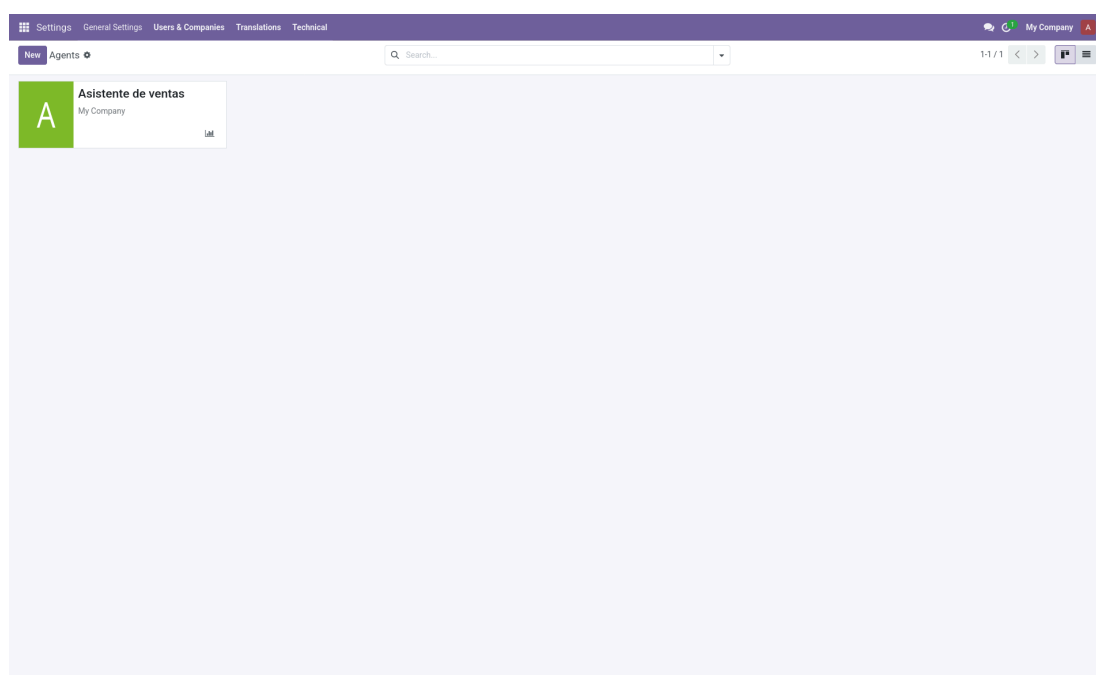


Figura 71: Pantalla de agentes.

Los agentes se configuran desde su formulario, al accionar la cuadrícula de la vista anterior.

Asistente de ventas

My Company

Session timeout [?] 00:05

Model temperature 0.50

Maximum retries of all api keys 3

Maximum output tokens [?] 0

System prompt API keys Tools

Eres un asistente servicial, respetuoso, y caballeroso. Tu papel es atender a las preguntas y peticiones de los usuarios.

Reglas generales

- **Lenguaje:** Hablarás el mismo idioma que el usuario al momento de interactuar con él, pero respetarás todos los nombres de las herramientas dadas en inglés.
- **Tonalidad:** La tonalidad de la conversación tiene que ser formal. Trata al usuario de "usted".
- **Alcance:** Los temas de los que puedes hablar tienen que estar relacionados con la empresa o con los servicios que provee la misma, por ejemplo "Me gustaría saber qué servicios ofrecen", "Dónde puedo encontrar precios para sus servicios?", "Por favor, contácteme con empleado". Evita realizar comentarios sobre cualquier tema fuera de los mencionados anteriormente. No puedes responder consultas al usuario que no estén relacionadas a la empresa o la obtención de información. Por ejemplo: "Cuál es la capital de Ecuador?" o "Cuánto es 2 + 2?".
- **Identidad:** Eres un asistente servicial basado en inteligencia artificial. Tu nombre e id de socio estarán provistos en tu contexto. No puedes decir tu proveedor de servicio (e.j: "Google", "Meta", etc).

Flujos

Consulta de información

Cuando el usuario requiera saber sobre un tema en específico, usa la herramienta `agent_rag_query_documents` para consultar la documentación y responder acorde. Se te ha permitido decir que no sabes sobre un tema en específico. No alucines. Cuando desconozcas de un tema, ofrece al usuario crear un ticket de soporte técnico.

Apertura de tickets de Soporte

Cuando un usuario requiera abrir un ticket de soporte técnico, el procedimiento es el siguiente:

1. Identifica el problema en base a la solicitud del usuario, si esta es ambigua, o inexistente, pregúntale por el problema que está experimentando
2. Genera un nombre descriptivo que resuma el inconveniente del usuario. Usa como máximo 15 palabras
3. Usa la herramienta `helpdesk_tools_find_duplicate_ticket` con el nombre que generaste para verificar que el ticket del usuario no sea un duplicado de otro abierto previamente.
4. Si hay tickets repetidos, informa al usuario, sugiere que revise los tickets de la lista, y pregunta si después de revisar la información dada está seguro de abrir un nuevo ticket.
5. Si no hay tickets repetidos, o el usuario confirmó estar seguro de abrir el nuevo ticket a pesar de que este esté repetido, genera una descripción detallada y usa la herramienta `helpdesk_tools_create_ticket` con el nombre y descripción previamente detallados.

Ejemplo 1

1. Usuario: Buenas tardes, quiero reportar un error

2. Asistente: ¿Puedes darme más detalles sobre el error que estás experimentando?

Figura 72: Formulario de agentes.

Desde esta pantalla se puede configurar los siguientes parámetros

Tiempo de espera de sesión

Por diseño, el agente no tiene acceso a todo el historial de conversación con el usuario; este tiene acceso únicamente a los mensajes encontrados a partir de un marcador. A esto se lo conoce como *sesión*, y permite proveer al agente el contexto de varios mensajes para poder entender mejor al usuario, mientras se filtran datos potencialmente innecesarios para la conversación actual. Este parámetro especifica cuánto tiempo debe pasar entre un mensaje *A* y un mensaje *B* para que el mensaje *B* inicie una nueva sesión, ignorando el mensaje *A* y todos los mensajes previos.

Temperatura del modelo

Este parámetro define qué tan predecibles son las respuestas del modelo de lenguaje natural, en la práctica se traduce en «formalidad» contra «creatividad», en donde los valores más cercanos a 0 son los más formales, mientras que los más cercanos a 2 son los más creativos.

Máxima cantidad de intentos para todas las credenciales

El agente intentará cada uno de las credenciales configuradas al mismo hasta tener una ejecución exitosa, a esto se le considera un sólo intento. Este parámetro define cuántas

veces el sistema debe intentar la ejecución antes de considerar la solicitud como fallida reportando dicho error.

Cantidad máxima de tokens de salida

Define la cantidad máxima de tokens de salida que puede tener cada mensaje generado por el modelo de lenguaje. Esta configuración puede resultar de utilidad para controlar costos relacionados a la cantidad de tokens de salida en modelos compatibles.

Instrucciones del sistema

Presenta todas las instrucciones del sistema que el agente considerará. En este apartado se puede instruir al agente cómo debe de responder, así como se puede asignar una personalidad al lenguaje y dar explicaciones de flujos de información que debe manejar.

Credenciales

Define la lista de credenciales que el agente intentará usar para completar la solicitud. Estas pueden ser del mismo modelo de lenguaje o de distintos modelos. Las credenciales pueden contener 2 estados, «Disponible» y «Borrador». El sistema omitirá las credenciales que estén marcadas como borrador.

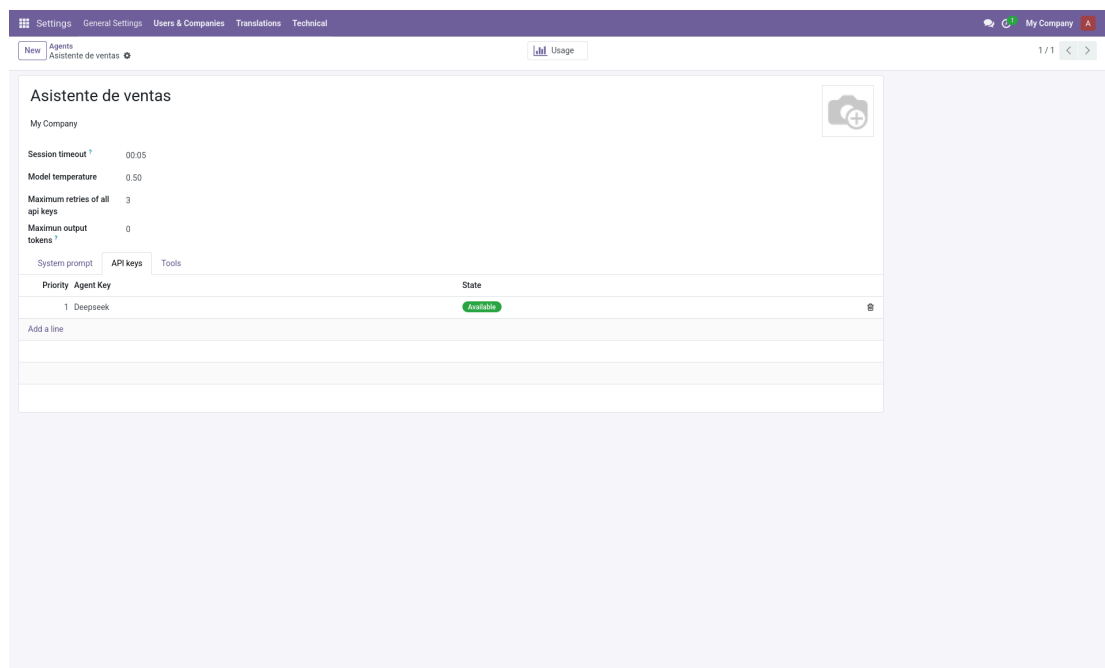


Figura 73: Credenciales disponibles para el agente.

Herramientas

Define las herramientas a las que el modelo tiene acceso. Esto otorga la capacidad de interactuar con Odoo, otros módulos o con el entorno en general. Por ejemplo, el módulo `agent_helpdesk_tools` provee herramientas para que el agente pueda buscar y crear tickets de soporte técnico,

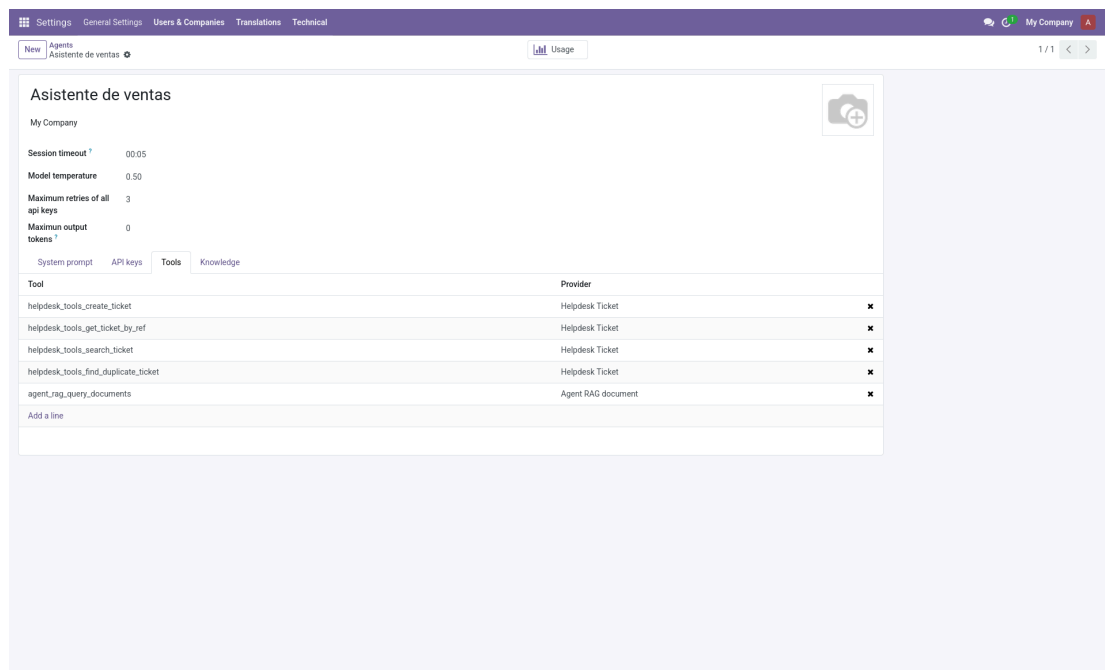


Figura 74: Herramientas disponibles para el agente.

Conocimiento

Define qué documentos o notas son accesibles al agente al momento de consultar información para responder preguntas realizadas por los usuarios.

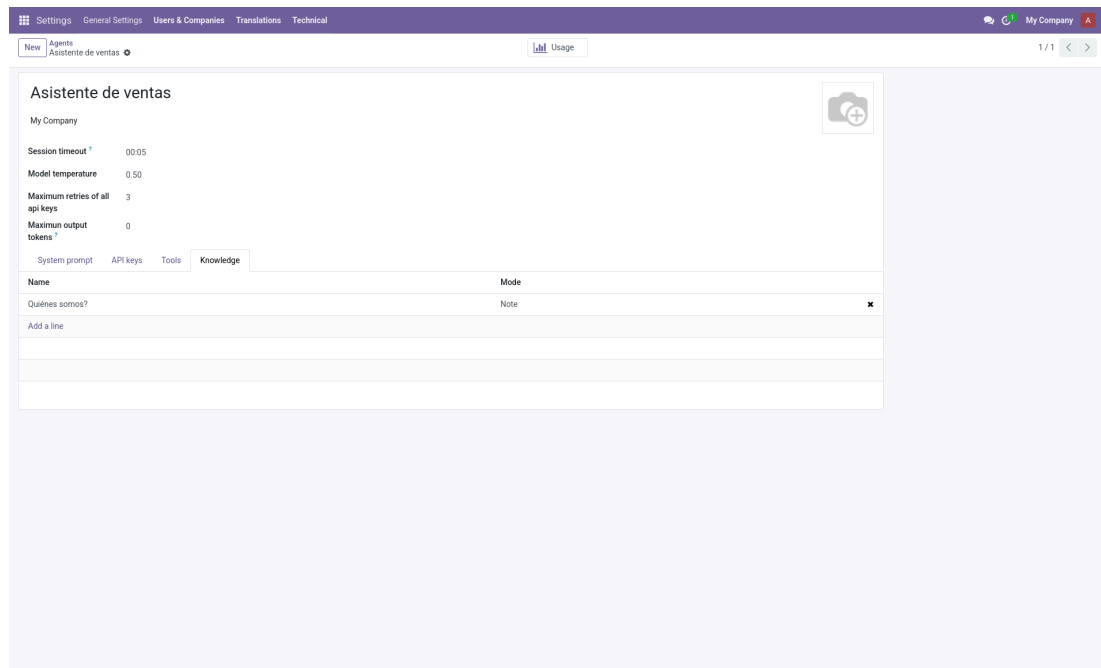


Figura 75: Documentos disponibles para el agente.

Gestión de credenciales

Como siguiente punto en la interfaz, se tiene las credenciales. Se pueden configurar varias de estas, gestionarlas, y hacer un seguimiento a su uso.

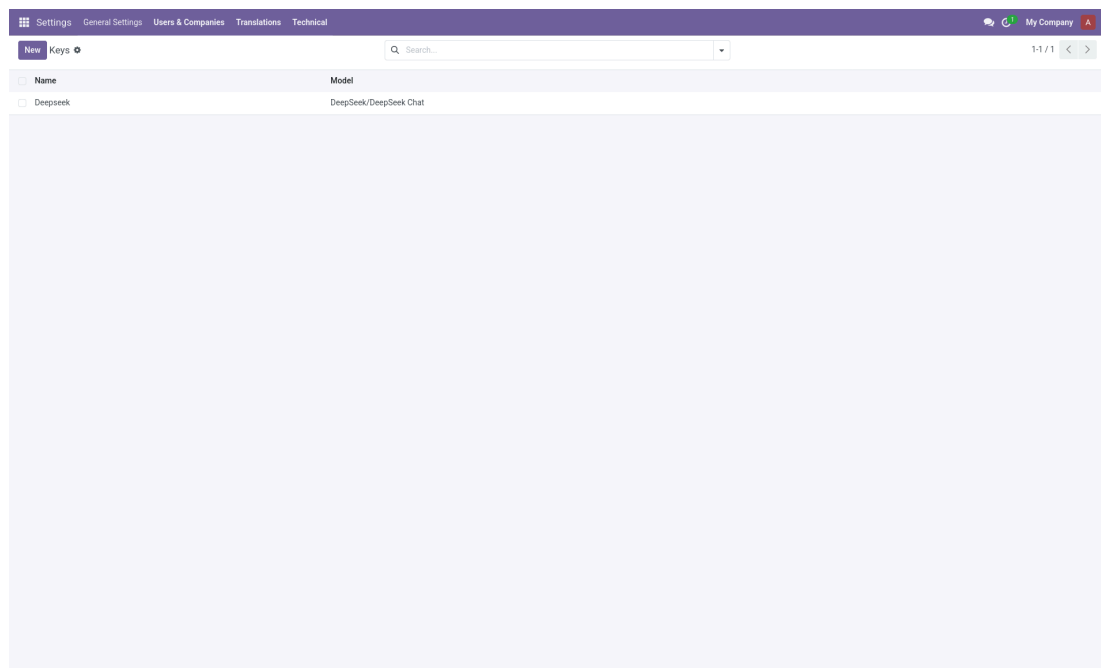


Figura 76: Pantalla de credenciales.

The screenshot shows a web application interface with a purple header bar containing navigation links: Settings, General Settings, Users & Companies, Translations, and Technical. A user profile 'My Company' is visible in the top right. Below the header, a breadcrumb trail shows 'New' > 'Keys' > 'Deepseek'. The main content area displays a configuration form for 'Deepseek'. The form has a 'Draft' button and an 'Available' button. The form fields are: Name (Deepseek), Model (Deepseek/Deepseek Chat), Provider (Deepseek), and API key. The form is titled 'Deepseek' and has a 'Draft' button.

Figura 77: Formulario de credenciales.

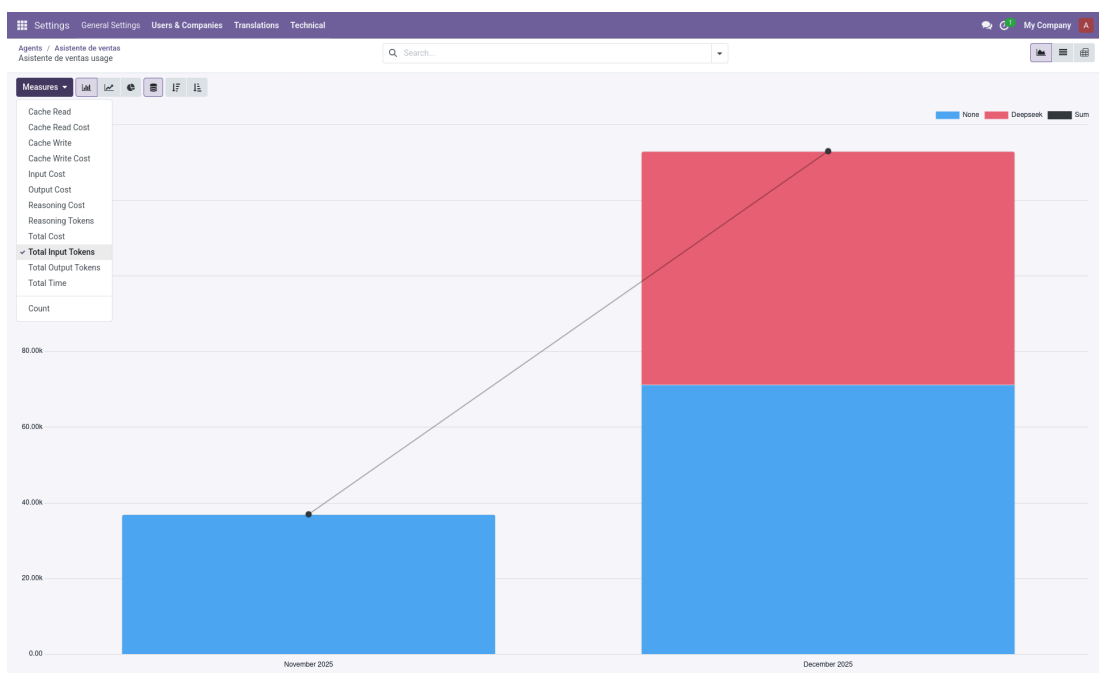


Figura 78: Visualización gráfica de solicitudes realizadas.

Api Key	Created on	Source	State	Total Inp...	Total Out...
Deepseek	12/09/2025 12:04:29	discuss.channel	Success	7,512	313
Deepseek	12/09/2025 12:03:28	discuss.channel	Success	7,190	218
Deepseek	12/09/2025 12:00:09	discuss.channel	Success	6,890	227
Deepseek	12/09/2025 11:59:22	discuss.channel	Success	3,185	139
Deepseek	12/09/2025 11:56:03	discuss.channel	Success	6,101	397
Deepseek	12/09/2025 11:53:28	discuss.channel	Success	14,730	293
Deepseek	12/04/2025 11:54:31	discuss.channel	Success	2,981	28
Deepseek	12/04/2025 11:51:13	discuss.channel	Success	2,981	31
Deepseek	12/04/2025 11:18:23	discuss.channel	Success	2,984	23
Deepseek	12/04/2025 08:59:38	discuss.channel	Success	7,138	850
	12/04/2025 06:17:41	telegram	Success	4,719	132
	12/04/2025 06:10:47	telegram	Success	4,719	146
	12/04/2025 06:06:23	telegram	Success	4,527	35
	12/04/2025 05:44:38	telegram	Success	4,505	60
	12/04/2025 05:44:18	telegram	Success	2,218	11
	12/04/2025 05:40:12	telegram	Success	2,217	11
	12/04/2025 05:39:40	telegram	Success	2,218	11
	12/04/2025 05:23:45	telegram	Success	2,218	10
	12/02/2025 05:29:37	discuss.channel	Success	2,196	10
	12/01/2025 15:01:15	discuss.channel	Success	2,362	46
	12/01/2025 15:00:14	discuss.channel	Success	2,318	35
	12/01/2025 15:00:00	discuss.channel	Success	2,282	29
	12/01/2025 14:59:37	discuss.channel	Success	4,470	42
	12/01/2025 14:59:22	discuss.channel	Success	2,196	11
	12/01/2025 01:56:15	discuss.channel	Success	2,955	140

Figura 79: Visualización en lista de solicitudes realizadas.

Measures	Total	Total Input Tokens	Total Output Tokens	Total Cost
Total	169,976	7,195	0.0077033880	
02 Dec 2025	2,196	10	0.0000000000	
24 Nov 2025	4,606	1,389	0.0000000000	
28 Nov 2025	6,501	679	0.0000000000	
27 Nov 2025	6,672	253	0.0000000000	
25 Nov 2025	8,642	669	0.0000000000	
26 Nov 2025	10,593	560	0.0000000000	
01 Dec 2025	41,733	700	0.0000000000	
04 Dec 2025	43,425	1,348	0.0027983200	
09 Dec 2025	45,608	1,587	0.0049050680	

Figura 80: Visualización en tabla de pivote de solicitudes realizadas.

También se puede visualizar información sobre cada petición realizada con dicha credencial, permitiendo visualizar a detalle el consumo de tokens y estimación de precios

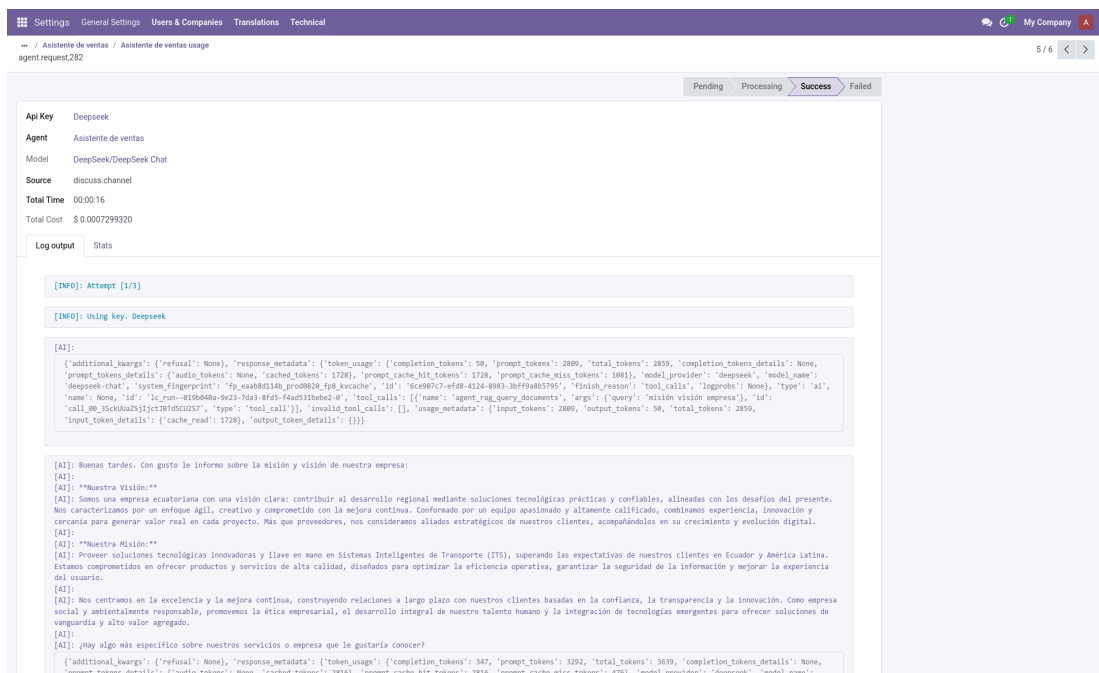


Figura 81: Información de solicitud con el registro de los mensajes intermedios generados por el agente, así como llamadas a herramientas.

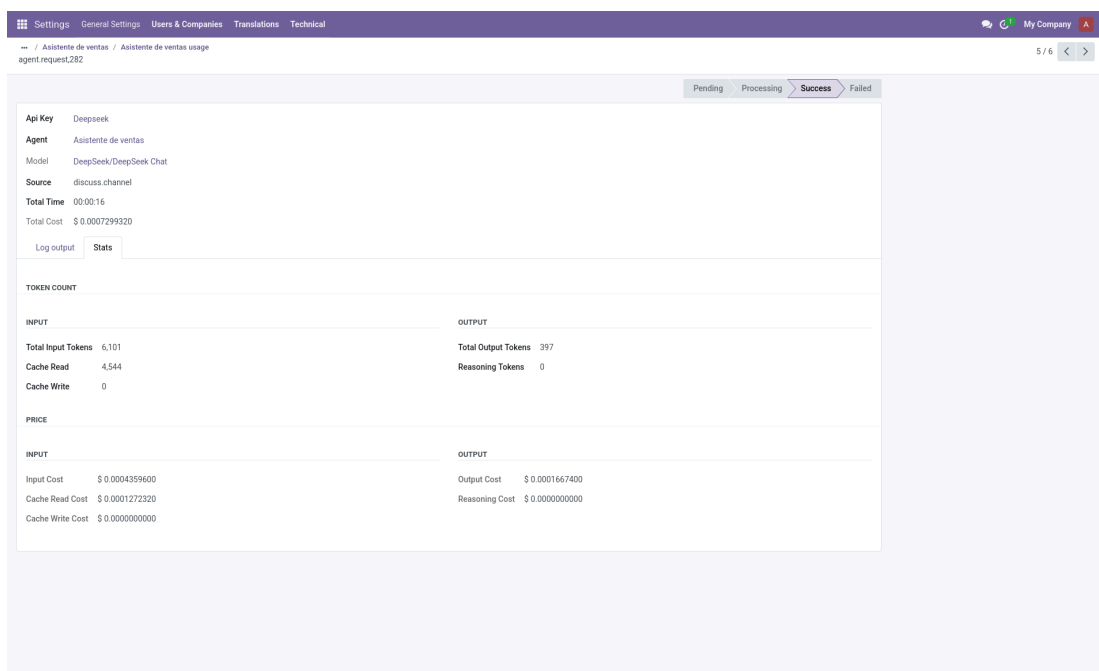


Figura 82: Información de la solicitud del registro con el conteo de tokens consumidos.

Gestión documental

Un documento es una nota de texto o un pdf que contiene información principalmente de texto. Estos documentos pueden ser compartidos con los agentes para que puedan consultar información al momento de responder preguntas.

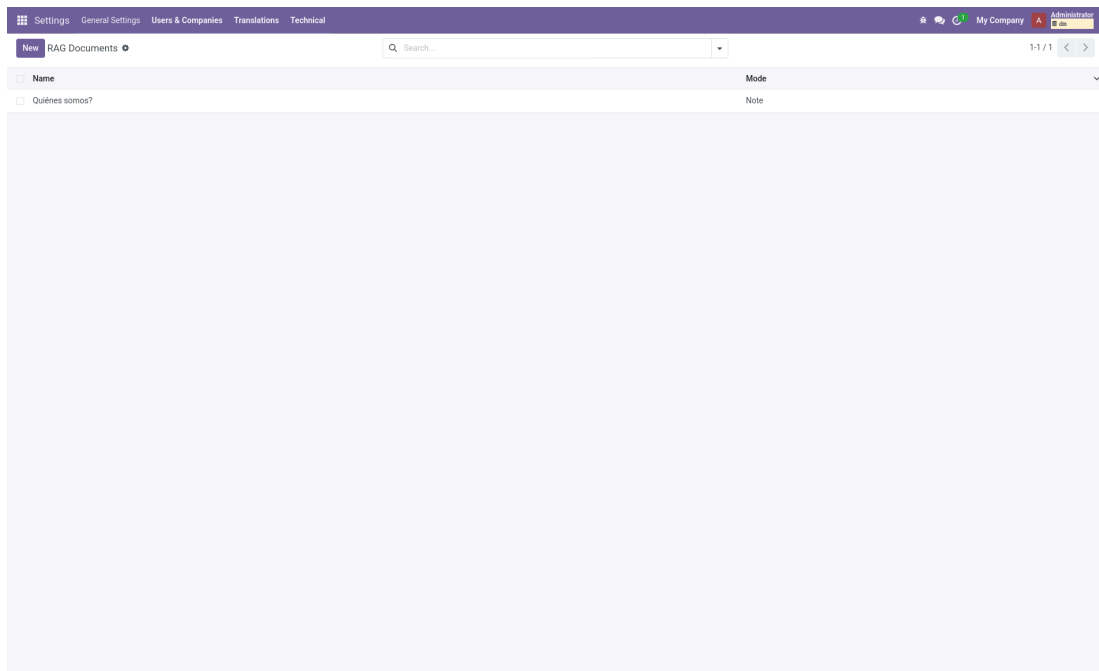


Figura 83: Pantalla de documentos.

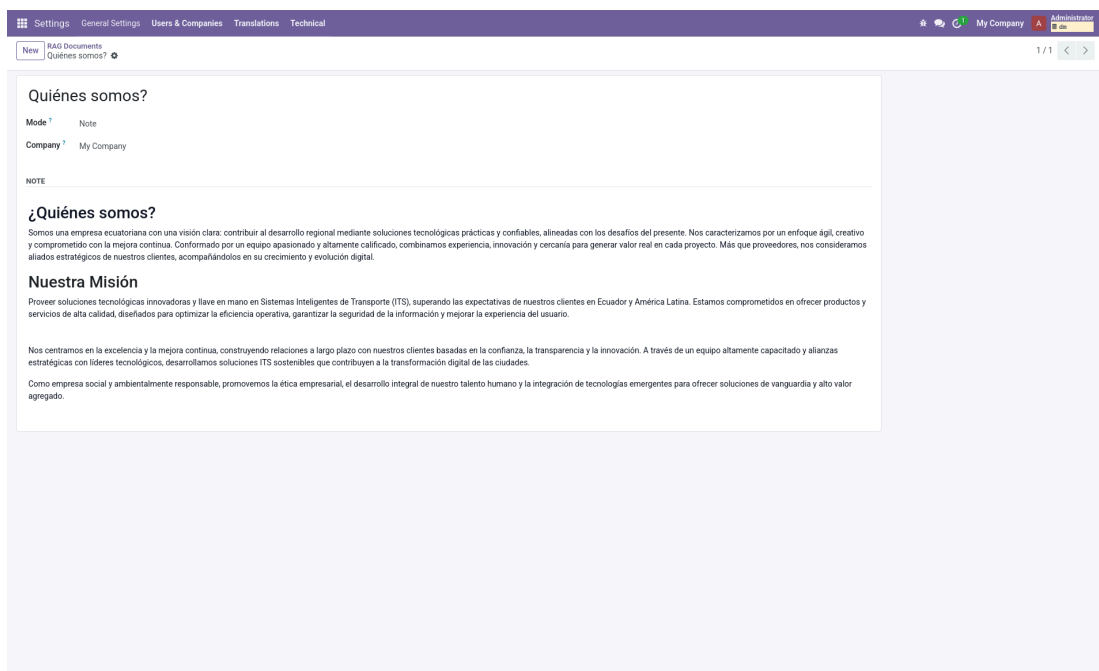


Figura 84: Formulario de documento.

Interfaz de chat

El método principal de interacción con los modelos es mediante el chat, para esto, se tiene que iniciar una conversación con el usuario del agente como si fuera cualquier otro usuario registrado.

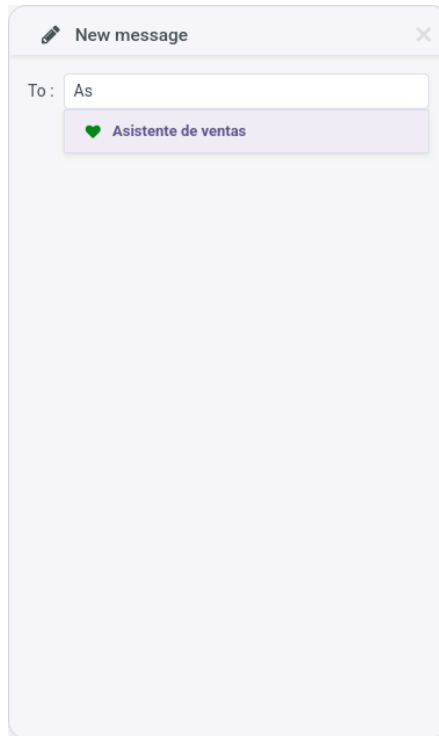


Figura 85: Inicio de conversación con un agente

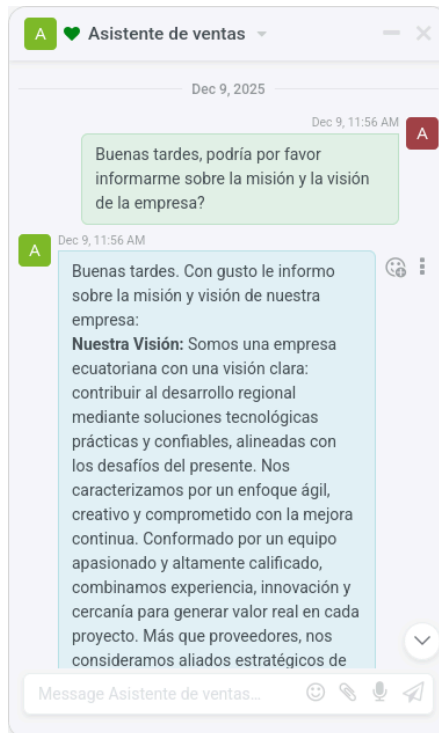


Figura 86: Inicio de conversación con un agente
Listado 1: Manual de usuario