



UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
VICERRECTORÍA DE INVESTIGACIÓN Y POSTGRADO
FACULTAD DE TECNOLOGIAS LIBRES

**MULTIAGENTE CONVERSACIONAL PARA LA INTERACCIÓN CON LOS
DATOS DEL TRANSPORTE MARÍTIMO RECOGIDOS EN EL DIARIO DE LA
MARINA**

**Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas**

Autor:

Daniel Rojas Grass

Tutores:

Dr.C. Orlando Grabiela Toledano López
MsC. Olga Yarisbel Rojas Grass

La Habana, 2025

Declaración de Autoría

El autor del trabajo de diploma con título “***Multiagente conversacional para la interacción con los datos del transporte marítimo recogidos en el Diario de la Marina***”, concede a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación, con carácter exclusivo. De forma similar se declara como único autor de su contenido.

Y para que así conste, firmo la presente declaración jurada de autoría en La Habana a los ____ días del mes de _____ del año _____.

Daniel Rojas Grass

Dr.C. Orlando Grabiela Toledano López

MsC. Olga Yarisbel Rojas Grass

Dedicatoria

La tercera página (opcional) se utilizará para la dedicatoria y en ella se expondrá a qué personas o entidades se dedica el trabajo

Resumen

Text

Palabras clave: texto

Abstract

Text

Keywords: three to five word

Índice general

Introducción	1
1 Fundamentación Teórica	6
1.1 Reconocimiento Óptico de Caracteres (OCR)	6
1.2 Agente LLM	7
1.3 Sistema Multiagente (SMA)	8
1.4 Sistemas RAG	9
1.5 Modelos de <i>embeddings</i>	9
1.6 Bases de datos vectoriales	10
1.7 Análisis del estado del arte	10
1.8 Enfoque y metodología de desarrollo de software	14
1.8.1 Análisis del enfoque: Estrella de Boehm–Turner	14
1.8.2 Comparativa de metodologías ágiles	14
1.8.3 Justificación de la elección: Extreme Programming	15
1.9 Herramientas y tecnologías	16
1.9.1 Herramienta de Modelado	16
1.9.2 Lenguaje de programación	16
1.9.3 Framework de desarrollo	17
1.9.4 Control de versiones	18
1.9.5 Vectorización y Almacenamiento	18
Conclusiones del Capítulo	19

2 Análisis, diseño e implementación de la propuesta de solución	21
Introducción	21
2.1 Descripción de la Propuesta de Solución	22
2.2 Análisis de requisitos	25
2.2.1 Técnicas de captura de requisitos	25
2.2.2 Requisitos funcionales	26
2.2.3 Requisitos no funcionales	29
2.2.4 Historias de usuarios	31
2.2.5 Tarjetas CRC	36
2.3 Diseño de la propuesta de solución	39
2.3.1 Diseño de la arquitectura	39
2.3.2 Diseño del modelo de datos	40
2.4 Implementación	41
2.4.1 Estándares de codificación en Python y JavaScript	41
2.4.2 Patrones de diseño	42
2.4.3 Interfaz Principal del Sistema	45
2.4.4 Diagrama de despliegue	46
Conclusiones	48
3 Pruebas de software	49
Introducción	49
3.1 Pruebas de software	50
3.2 Estrategia de pruebas	50
3.3 Pruebas unitarias	51
3.4 Pruebas funcionales	54
3.4.1 Método de caja negra	54
3.5 Prueba de seguridad	57
3.6 Prueba de rendimiento	59
Conclusiones	59
Conclusiones generales	60

Recomendaciones	61
Anexos	68
A Historias de Usuario	69
B Targetas CRC	81

Índice de figuras

1.1	Diario de la Marina 1884	7
2.1	Estructura de la propuesta de solución (Fuente: elaboración propia).	22
2.2	Flujo Interno del Microservicio Multiagente Conversacional (Fuente: elaboración propia).	24
2.3	Arquitectura de microservicios del sistema multiagente (Fuente: elaboración propia).	39
2.4	Diseño del modelo de datos (Fuente: elaboración propia).	40
2.5	Extracto de código que implementa el patrón (Fuente: elaboración propia). <i>Singleton</i>	43
2.6	Extracto de código que implementa el patrón <i>Cadena de responsabilidad</i> (Fuente: elaboración propia).	44
2.7	Extracto de código que implementa el patrón <i>Plantilla Abstracta</i> (Fuente: elaboración propia).	44
2.8	Extracto de código que implementa el patrón <i>Fachada</i> (Fuente: elaboración propia).	45
2.9	Interfaz principal del chat con el sistema multiagente (Fuente: elaboración propia)	46
2.10	Representación del modelo de despliegue. (Fuente: elaboración propia).	47
3.1	Resultado de las pruebas unitarias.	54
3.2	Representación del resultado la ejecución de una prueba usando Selenium IDE, del requisito Insertar consulta.	55
3.3	Representación del resultado de las pruebas funcionales (Fuente: Elaboración Propia).	57
3.4	Prueba de seguridad 1ra iteración.	58
3.5	Prueba de seguridad 2da iteración.	59

Índice de tablas

1.1	Análisis de Soluciones Conversacionales	12
2.1	Tabla de Requisitos Funcionales (RF)	26
2.2	Requisitos No Funcionales (RNF)	30
2.3	Historia de usuario # 1	31
2.4	Historia de usuario # 2	33
2.5	Historia de usuario # 3	35
2.6	Tarjeta CRC: Agente Moderador	37
2.7	Tarjeta CRC: Agente recuperador de información (FAISS)	37
2.8	Tarjeta CRC: Agente Contextualizador	37
2.9	Tarjeta CRC: Agente de Validación	38
2.10	Tarjeta CRC: Agente PandasAi	38
3.1	Plan de Pruebas	50
3.2	Cálculo de la complejidad ciclomática del método <code>post</code> de la clase <code>Message_Create_AV</code> .	52
3.3	Cálculo de la complejidad ciclomática del método <code>post</code> de la clase <code>Message_Create_AV</code> .	53
3.4	Caminos del grafo de flujo (Fuente: Elaboración propia).	53
3.5	Tabla de caminos	53
3.6	Caso de Prueba para el camino básico 2 (Fuente: Elaboración propia).	53
3.7	Caso de prueba para la funcionalidad Insertar Consulta (Fuente: Elaboración Propia).	56
3.8	Variables de caso de prueba “Insertar Consulta” (Fuente: Elaboración Propia).	56
A.1	Historia de usuario # 4	69

A.2	Historia de usuario # 5	71
A.3	Historia de usuario # 6	73
A.4	Historia de usuario # 7	74
A.5	Historia de usuario # 8	75
A.6	Historia de usuario # 9	77
A.7	Historia de usuario # 10	78
B.1	Tarjeta CRC: Usuario	81
B.2	Tarjeta CRC: Autenticador	81
B.3	Tarjeta CRC: Conversación	82
B.4	Tarjeta CRC: Historial	82
B.5	Tarjeta CRC: Chat	82
B.6	Tarjeta CRC: Backend	82
B.7	Tarjeta CRC: BaseDeDatos	83
B.8	Tarjeta CRC: MicroservicioMAS	83

Introducción

La digitalización masiva de documentos históricos ha transformado el acceso a fuentes de información que, durante siglos, permanecieron confinadas a archivos físicos. Este proceso ha generado una explosión de datos digitales provenientes de colecciones diversas, como periódicos históricos, manuscritos y registros oficiales. Sin embargo, la transición del medio impreso al digital no ha estado exenta de desafíos. La conversión de estos materiales ha dado lugar a grandes volúmenes de datos no estructurados, caracterizados por su heterogeneidad y la falta de metadatos estandarizados, lo que dificulta su procesamiento automatizado [Margot Note, 2018].

Este fenómeno genera retos de índole técnica, como la conservación a largo plazo y la compatibilidad entre formatos, así como de naturaleza conceptual, asociados a la obtención y contextualización del conocimiento para propósitos académicos y culturales [Margot Note, 2018]. La incapacidad de estructurar y analizar eficientemente estos datos limita su potencial como recurso para la investigación histórica, el análisis lingüístico y el estudio de patrones sociales.

Un ejemplo emblemático sobre esta situación es el *Diario de la Marina*, autodenominado «El decano de la prensa cubana», fue un referente de la prensa en La Habana entre 1844 y 1960. De carácter conservador, documentó eventos políticos y sociales, además de registrar información valiosa sobre actividades económicas, como el transporte marítimo, pilar fundamental de la historia comercial de Cuba [Wikipedia, 2023].

La digitalización de sus páginas ha permitido preservar este patrimonio histórico, aunque con limitaciones significativas. Factores como la diversidad de tipografías antiguas, la calidad variable de impresión y el estado de conservación de los documentos han generado errores de transcripción y desorganización de la información [Yogish Naik G R, 2023]. En el caso específico del transporte marítimo —rutas, cargamentos, puertos y fechas—, esta falta de estructura dificulta su análisis sistemático y su uso en investigaciones. Por

ello, la mera digitalización no basta: es necesario aplicar procesos avanzados de interpretación y contextualización que permitan convertir estos datos en conocimiento útil y accesible para responder a consultas especializadas [Ferro et al., 2023, Yogish Naik G R, 2023].

A partir de este panorama, se identifican tres ejes problemáticos interrelacionados:

- **Desestructuración y heterogeneidad de los datos digitalizados:** La conversión de documentos impresos antiguos —como el *Diario de la Marina*— a formatos digitales ha generado grandes volúmenes de información no estructurada, afectados por errores de reconocimiento óptico de caracteres (OCR), tipografías irregulares y la ausencia de metadatos estandarizados. Esta situación dificulta su análisis computacional y limita su reutilización en entornos académicos o científicos.
- **Falta de contextualización semántica para propósitos analíticos:** Aunque se dispone de los datos digitalizados, estos carecen de mecanismos que permitan comprender su significado en contexto. Información como rutas marítimas, nombres de embarcaciones o fechas relevantes no está categorizada ni vinculada semánticamente, lo que impide su integración en flujos de análisis históricos o culturales más amplios.
- **Limitaciones de las soluciones tradicionales frente a la complejidad del dominio:** Las herramientas clásicas de recuperación de información no logran abordar la riqueza y ambigüedad inherente al lenguaje histórico.

Problema de la investigación

A partir de la situación problemática descrita se identifica el siguiente problema de la investigación: ¿Cómo interpretar consultas en lenguaje natural para realizar análisis estadísticos sobre datos del transporte marítimo recogidos en el *Diario de la Marina*?

Con base en lo anterior, se define como **objeto de estudio** el proceso de análisis estadístico de datos a través de consultas en lenguaje natural , y como **campo de acción** el análisis estadístico de datos mediante el uso de sistemas multiagente conversacionales y modelos neuronales del lenguaje.

Objetivo General de la Investigación

Desarrollar un sistema multiagente conversacional que permita interpretar y contextualizar automáticamente los datos no estructurados del transporte marítimo recogidos en el Diario de la Marina,para su transformación

en conocimiento estructurado que facilite el análisis de patrones comerciales y la comprensión de las dinámicas económicas del Caribe entre 1844 y 1960.

Objetivos Específicos

A partir del planteamiento del objetivo de investigación, se definen los siguientes objetivos específicos:

- Analizar los referentes teóricos y metodológicos sobre la transformación de información no estructurada a lenguaje natural, revisando enfoques de inteligencia artificial y sistemas conversacionales aplicados a datos históricos.
- Identificación de requisitos, análisis y diseño del sistema multiagente conversacional, definiendo sus componentes, interacciones y flujos de trabajo.
- Desarrollar un sistema multiagente conversacional que contribuya a la transformación y el análisis de la información no estructurada extraída del *Diario de la Marina*, enfocándose en los datos del transporte marítimo.
- Validar el correcto funcionamiento del sistema multiagente conversacional, aplicando métricas de evaluación y pruebas de software para garantizar su calidad, precisión y usabilidad.

Métodos de Investigación

Para llevar a cabo esta investigación se aplicaron métodos teóricos y empíricos de la investigación científica, los cuales se relacionan con el desarrollo de un sistema multiagente conversacional para la transformación e interacción con datos históricos. A continuación, se detallan:

Métodos Teóricos

1. **Analítico-sintético.** Permitió analizar, sintetizar y evaluar el proceso de transformación de datos no estructurados provenientes del *Diario de la Marina*, desde un enfoque centrado en la aplicación de técnicas de procesamiento de lenguaje natural y sistemas multiagentes. Con este método se identificó la esencia del problema de investigación, analizando los componentes del proceso de digitalización, las limitaciones de las técnicas actuales de reconocimiento óptico de caracteres (OCR) y las necesidades de contextualización de los datos del transporte marítimo para su uso efectivo.
2. **Hipotético-deductivo.** Se empleó para identificar las variables clave involucradas en la interacción conversacional con datos históricos y sus interrelaciones, especialmente aquellas relacionadas con

el diseño de sistemas multiagentes su implementación y el procesamiento de consultas en lenguaje natural. Este método facilitó la formulación de supuestos sobre cómo los agentes colaborativos pueden mejorar la interpretación y contextualización de la información no estructurada.

3. **Histórico-lógico.** Se aplicó para revisar la evolución de las tecnologías de digitalización de documentos históricos, el desarrollo de sistemas conversacionales basados en inteligencia artificial y el uso de datos del transporte marítimo en contextos históricos. Este enfoque permitió reconocer los avances teórico-prácticos en el área, así como las limitaciones actuales en la gestión de datos no estructurados extraídos de fuentes como el *Diario de la Marina*.

Métodos Empíricos

1. **Análisis documental.** Consistió en la revisión de la literatura relacionada con la transformación de datos no estructurados, el diseño de sistemas multiagentes y las aplicaciones de modelos de lenguaje en la interpretación de textos históricos. Incluyó el estudio de enfoques, algoritmos y herramientas de inteligencia artificial utilizadas en tareas de procesamiento de lenguaje natural y extracción de conocimiento a partir de documentos digitalizados.
2. **Experimental.** Para validar los resultados del sistema multiagente conversacional desarrollado, se evaluó su capacidad para interpretar consultas en lenguaje natural y contextualizar los datos del transporte marítimo provenientes del *Diario de la Marina*. Los resultados obtenidos se compararon mediante un conjunto de métricas de evaluación, incluyendo precisión, tiempo de respuesta, tasa de éxito en la resolución de tareas y coherencia semántica, todas ellas ajustadas a la calidad de las respuestas generadas y a la satisfacción del usuario.

Estructura del Documento

El documento está organizado en introducción, tres capítulos, conclusiones, recomendaciones, bibliografía y anexos. A continuación, se describe el contenido abordado en cada capítulo:

- **Capítulo 1. Fundamentación Teórica:** Se realiza un estudio y análisis de los diferentes métodos y técnicas para el procesamiento de datos no estructurados provenientes de documentos históricos, con énfasis en la digitalización del *Diario de la Marina*. Se analizan los fundamentos teóricos relacionados con la transformación de datos mediante técnicas de inteligencia artificial, centrándose en el uso de sistemas multiagentes y modelos de lenguaje para interpretar textos históricos. De igual manera, se

revisan los principales referentes teóricos sobre el procesamiento de lenguaje natural (PLN), la estructuración de información no estructurada y la interacción conversacional con usuarios. Finalmente, se evalúan los desafíos específicos asociados a los datos del transporte marítimo (rutas, puertos, fechas) extraídos de fuentes digitalizadas, identificando las limitaciones de las técnicas actuales como el Reconocimiento Óptico de Caracteres (OCR).

- **Capítulo 2. Análisis, diseño e implementación de la propuesta de solución:** En este capítulo, se desarrolla un sistema multiagente conversacional diseñado para interpretar y contextualizar automáticamente los datos no estructurados del transporte marítimo del *Diario de la Marina*. Se modelan y describen la arquitectura empleada en el sistema multiagente, requisitos funcionales y no funcionales, así como los patrones de diseño implementados en la solución. Finalmente, se presentan las conclusiones parciales del capítulo, destacando las decisiones de diseño y los resultados preliminares de la implementación.
- **Capítulo 3. Pruebas de software:** Se desarrolla un conjunto de experimentos utilizando una muestra representativa de datos digitalizados del *Diario de la Marina*, enfocándose en registros del transporte marítimo, y se discuten los principales resultados en cuanto a la eficacia del sistema multiagente al procesar consultas en lenguaje natural. Con la arquitectura óptima seleccionada, se evalúa el sistema propuesto en escenarios prácticos, como la respuesta a preguntas sobre rutas marítimas, puertos y fechas históricas. Primero, se valida el sistema con una muestra controlada de datos extraídos del periódico y se comparan los resultados con enfoques tradicionales del estado del arte (por ejemplo, búsquedas manuales o sistemas no conversacionales). Se realiza un análisis exploratorio de los datos históricos, incluyendo preprocesamiento, corrección de errores y extracción de características relevantes para estructurar la información. Finalmente, se presentan los resultados experimentales, evaluando métricas como precisión, recall y satisfacción del usuario, y se discuten las implicaciones para la investigación histórica y la gestión de patrimonios digitales.

CAPÍTULO 1

Fundamentación Teórica

El primer capítulo de la tesis establece el marco teórico, contextual y técnico necesario para el desarrollo de un sistema multiagente conversacional capaz de interactuar con datos históricos del transporte marítimo, extraídos del Diario de la Marina, un periódico cubano de gran relevancia histórica, publicado entre 1844 y 1960. Este capítulo se organiza en torno a cinco pilares fundamentales: la fundamentación teórica del tema, un análisis del estado del arte y del mercado en el ámbito de los sistemas multiagente aplicados a este tipo de contextos, la justificación metodológica del proceso de desarrollo de software, la selección y descripción de las herramientas y tecnologías utilizadas, y la gestión de bases de datos requerida para el procesamiento de la información histórica. Como cierre, se presentan observaciones preliminares que no solo resumen los hallazgos iniciales, sino que también sientan las bases conceptuales y técnicas para los capítulos subsiguientes de la investigación.

1.1. Reconocimiento Óptico de Carácteres (OCR)

El Reconocimiento Óptico de Carácteres (OCR) es una tecnología que permite convertir documentos físicos, como imágenes escaneadas o fotografías de texto, en datos digitales editables. Este proceso emplea algoritmos avanzados para analizar la estructura visual de una imagen, identificando caracteres y palabras a partir de

patrones reconocibles, lo que transforma contenido analógico en un formato susceptible de edición, búsqueda y análisis automatizado [Piryani et al., 2025]. En el ámbito de la digitalización masiva de archivos históricos, como el *Diario de la Marina* (1844–1960), el OCR desempeña un papel crucial como el primer paso para preservar y dar acceso a un patrimonio cultural que, de otro modo, permanecería confinado a soportes físicos deteriorados. Sin embargo, su aplicación a documentos históricos no está exenta de limitaciones.

Factores como la diversidad de tipografías antiguas, el desgaste físico de los originales y la calidad variable de impresión introducen errores significativos en la transcripción, resultando en datos digitales no estructurados que carecen de organización y metadatos estandarizados. En el caso del *Diario de la Marina* (ver Figura 1.1), estas imperfecciones se evidencian en la conversión de información sobre transporte marítimo —rutas, puertos, cargamentos y fechas—, cuya heterogeneidad dificulta su análisis sistemático.



Figura 1.1. Diario de la Marina 1884.

La importancia del OCR en esta investigación se deriva de su rol como el primer eslabón en la transformación de estos documentos en datos digitales accesibles, aunque sus limitaciones resaltan la necesidad de enfoques posteriores de inteligencia artificial para refinar y dar sentido a la información resultante.

1.2. Agente LLM

Los datos generados por el OCR, aunque accesibles, presentan desafíos que requieren herramientas más avanzadas para su interpretación. Aquí es donde entran en juego los agentes basados en modelos de lenguaje de gran escala (LLM, por sus siglas en inglés: *Large Language Model*), entidades computacionales que

aprovechan técnicas de inteligencia artificial para comprender, generar e interactuar con el lenguaje humano de forma sofisticada [Guo et al., 2024]. Entrenados con vastos conjuntos de datos textuales, estos agentes son capaces de procesar lenguaje natural con coherencia y contextualidad, interpretando significados implícitos y adaptándose a diferentes estilos de comunicación. En el contexto del Diario de la Marina, los agentes LLM pueden abordar la ambigüedad y los errores de los datos no estructurados producidos por el OCR. Por ejemplo, podrían corregir términos náuticos mal transcritos o inferir el significado de frases incompletas causadas por el deterioro del papel, como una referencia a un cargamento de azúcar en un puerto específico. Además, su capacidad para integrar información externa —mediante herramientas como bases de datos históricas— permite conectar datos aislados con contextos más amplios, como eventos comerciales del Caribe [Guo et al., 2024]. Este concepto es fundamental porque transforma texto crudo en contenido más comprensible, aunque su acción aislada no basta para estructurar sistemáticamente grandes volúmenes de información histórica, lo que lleva a explorar enfoques más coordinados.

1.3. Sistema Multiagente (SMA)

Mientras los agentes LLM ofrecen una poderosa herramienta para interpretar texto, la complejidad y el volumen de los datos del Diario de la Marina demandan un enfoque más estructurado. Un sistema multiagente (SMA) se define como un conjunto de agentes autónomos que operan en un entorno compartido, interactuando entre sí para resolver problemas complejos o cumplir tareas específicas que exceden las capacidades de un solo componente [Zambrano et al., 2020]. Estos agentes trabajan de manera independiente pero también colaboran o compiten según los objetivos establecidos, lo que los hace idóneos para manejar información diversa y dinámica [Pérez-Pons et al., 2021]. La relevancia de los SMA radica en su capacidad para descomponer la heterogeneidad de los datos históricos en tareas manejables. Por ejemplo, un agente podría especializarse en identificar nombres de puertos en textos mal transcritos, mientras otro contextualiza esos datos con rutas comerciales del siglo XIX. Esta modularidad permite procesar eficientemente grandes volúmenes de información y adaptarse a las particularidades de los documentos históricos. Características como la autonomía, la interacción mediante protocolos como el lenguaje ACL, y la adaptabilidad [Pérez-Pons et al., 2021], junto con elementos como agentes, entorno, comunicación y organización [Zambrano et al., 2020], potencian las capacidades individuales de herramientas como los LLMs, abriendo la puerta a procesos más robustos de análisis y estructuración de datos.

1.4. Sistemas RAG

La coordinación de agentes en un SMA amplifica su potencial, pero la precisión y relevancia de la información procesada dependen de acceder a contextos actualizados y verificables. Los sistemas RAG (Retrieval Augmented Generation) representan una arquitectura innovadora que combina modelos de lenguaje avanzados con mecanismos de recuperación de información en tiempo real [Gao, 2024, Lewis et al., 2020a]. A diferencia de los modelos tradicionales, que se limitan a conocimientos preentrenados, los sistemas RAG consultan fuentes externas dinámicamente, integrando datos frescos para generar respuestas más precisas y contextualizadas. En el caso de los datos del Diario de la Marina, un sistema RAG podría enriquecer la interpretación de referencias al transporte marítimo al recuperar información complementaria —como registros de otros periódicos o archivos históricos— que no está presente en el texto original. Esto reduce errores y alucinaciones, un problema común en el procesamiento de textos históricos ambiguos, y asegura que la información sea relevante para análisis económicos o sociales. Su importancia en este marco teórico reside en su capacidad para conectar los datos no estructurados con un contexto más amplio, un paso esencial para transformarlos en conocimiento útil, aunque requiere herramientas adicionales para organizar y almacenar eficientemente esa información recuperada.

1.5. Modelos de *embeddings*

La recuperación y generación de información que ofrecen los sistemas RAG necesitan un mecanismo para representar y comparar datos de manera efectiva. Los modelos de *embeddings* son algoritmos entrenados para transformar datos complejos, como texto o imágenes, en representaciones vectoriales numéricas en un espacio multidimensional [Mikolov et al., 2013]. Estas representaciones capturan relaciones semánticas y contextuales entre los datos, permitiendo a los sistemas de inteligencia artificial identificar similitudes y diferencias con alta precisión [Mikolov et al., 2013]. Para los datos del Diario de la Marina, los *embeddings* pueden convertir menciones dispersas de puertos, fechas o cargamentos en vectores que reflejen su significado subyacente. Por ejemplo, términos como “Puerto de La Habana” y “Habana” en contextos distintos podrían agruparse como similares, facilitando su análisis sistemático. Este concepto es clave porque proporciona una base matemática para procesar y estructurar información no estructurada, vinculando los avances de los

LLMs y RAG con métodos de almacenamiento y búsqueda más avanzados, necesarios para manejar grandes volúmenes de datos históricos.

1.6. Bases de datos vectoriales

Finalmente, las representaciones generadas por los modelos de *embeddings* requieren un sistema optimizado para su almacenamiento y recuperación. Las bases de datos vectoriales son una evolución en la gestión de datos, diseñadas específicamente para manejar vectores numéricos de alta dimensión mediante algoritmos de búsqueda aproximada (ANN) y técnicas de indexación como HNSW, PQ y LSH [Xie et al., 2023, Han and Liu, 2023]. Estas bases utilizan métricas como la distancia coseno o euclídea para buscar similitudes en espacios multidimensionales, ofreciendo una solución escalable para grandes volúmenes de información no estructurada [Azizi et al., 2024, Sun and Guo, 2024]. En el contexto del *Diario de la Marina*, una base de datos vectorial podría almacenar *embeddings* de los datos del transporte marítimo, permitiendo búsquedas rápidas y contextualizadas —como encontrar todas las menciones de un puerto específico en un rango de años—. Su integración con LLMs y sistemas RAG potencia la capacidad de recuperar información semánticamente relevante, cerrando el ciclo desde la generación inicial de datos por OCR hasta su transformación en conocimiento estructurado. Este enfoque es fundamental para manejar la escala y complejidad de los archivos históricos, proporcionando una infraestructura robusta para análisis posteriores.

1.7. Análisis del estado del arte

El análisis del estado del arte busca explorar las soluciones actuales que integran inteligencia artificial (IA) para interactuar con documentos, con el fin de identificar sus alcances, limitaciones y cómo estas carencias justifican la necesidad de un enfoque más especializado para manejar datos históricos no estructurados, como los del Diario de la Marina. Este ejercicio no solo sirve como base para elegir tecnologías adecuadas, sino que también pone en evidencia los vacíos que hacen pertinente un sistema capaz de interpretar y contextualizar información histórica de manera precisa y accesible.

Entre las plataformas que ejemplifican este enfoque se encuentran:

Chatize:¹ Chatize es una herramienta gratuita que transforma documentos PDF en chatbots interactivos, permitiendo a los usuarios hacer preguntas, obtener resúmenes y extraer información de textos como libros, ensayos o contratos legales [Chatize Team, 2023]. Su simplicidad y accesibilidad la convierten en una opción atractiva para documentos modernos, pero su aplicación a textos históricos revela deficiencias significativas. No está diseñada para manejar transcripciones imperfectas ni para integrar información externa que contextualice eventos pasados, como las dinámicas del transporte marítimo en el Caribe entre 1844 y 1960. Esto la hace insuficiente para las necesidades de investigadores que requieren precisión y profundidad histórica.

TextCortex ChatPDF:² Esta plataforma permite cargar archivos en formatos como PDF, PPTX y DOCX, posibilitando interacciones conversacionales con múltiples documentos simultáneamente. Su capacidad para integrarse con diversas fuentes de datos como Google Drive y OneDrive facilita la gestión de información dispersa. Sin embargo, aunque ofrece una interfaz intuitiva para documentos modernos, podría no estar optimizada para manejar las particularidades de textos históricos con problemas de OCR o lenguaje arcaico [TextCortex, 2024].

Vidnoz ChatPDF:³ Esta herramienta gratuita transforma archivos PDF en asistentes interactivos en línea, permitiendo realizar preguntas, obtener resúmenes y analizar documentos en segundos. Soporta más de 50 idiomas y la capacidad de procesar múltiples PDFs simultáneamente. Aunque es eficaz para documentos contemporáneos, su aplicación a textos históricos podría verse limitada por la falta de herramientas especializadas para corregir errores de OCR o interpretar terminología antigua [Vidnoz, 2024].

MyMap.AI ChatPDF:⁴ Además de permitir interacciones conversacionales con documentos PDF, MyMap.AI se especializa en la creación de contenido visual, generando resúmenes visuales, mapas mentales o diagramas de flujo basados en el contenido del PDF. Esta característica puede ser útil para esquematizar información compleja proveniente de documentos históricos, facilitando una comprensión más profunda del contexto [MyMap.AI, 2024].

Adobe Acrobat AI Assistant:⁵ Este asistente de Adobe permite interactuar de forma conversacional con archivos PDF. Ofrece resúmenes automáticos, respuestas a preguntas en lenguaje natural y análisis de textos

¹<https://www.chatize.com/es>

²<https://textcortex.com/es/chatpdf-alternative>

³<https://es.vidnoz.com/chat-pdf.html>

⁴<https://www.mymap.ai/es/chat-pdf>

⁵<https://www.adobe.com/acrobat/generative-ai-pdf.html>

extensos. Su fiabilidad y calidad de extracción lo convierten en una opción sólida, aunque su enfoque generalista puede no adaptarse perfectamente a documentos con errores de escaneo o estructuras textuales no convencionales [Adobe Inc., 2024].

Tabla 1.1. Análisis de Soluciones Conversacionales

Herramienta	Fortalezas	Debilidades	Aplicabilidad en Documentos Históricos
TextCortex ChatPDF	<ul style="list-style-type: none"> - Soporte para múltiples formatos (PDF, DOCX, PPTX). - Interacción con múltiples documentos a la vez. - Integración con plataformas externas. 	<ul style="list-style-type: none"> - No optimizado para errores de OCR o lenguaje antiguo. - No tiene soporte explícito para contexto histórico. 	<i>Moderada:</i> útil para exploración general, pero limitada en precisión con fuentes antiguas.
Vidnoz ChatPDF	<ul style="list-style-type: none"> - Interfaz gratuita, rápida y en múltiples idiomas. - Permite resumen y preguntas directas. 	<ul style="list-style-type: none"> - No adapta bien contenido con estructuras complejas o dañadas. - Poca personalización. 	<i>Baja:</i> apropiada para revisión superficial, pero no para análisis detallado.
MyMap.AI ChatPDF	<ul style="list-style-type: none"> - Generación de mapas conceptuales automáticos. - Buena visualización de relaciones de conceptos. 	<ul style="list-style-type: none"> - No centrado en precisión textual ni en corrección de OCR. 	<i>Moderada:</i> útil para esquematizar estructuras narrativas o comerciales antiguas.
Adobe Acrobat AI Assistant	<ul style="list-style-type: none"> - Potente motor de extracción textual. - Resúmenes automáticos y análisis conversacional preciso. 	<ul style="list-style-type: none"> - Enfoque generalista. - Poca adaptabilidad a contenido histórico con errores de digitalización. 	<i>Moderada:</i> aunque no especializado, su precisión lo hace adecuado para OCR limpio.
Chatize	<ul style="list-style-type: none"> - Gratuita y accesible. - Conversión directa de PDFs en chatbots interactivos. 	<ul style="list-style-type: none"> - No diseñada para textos históricos con errores de OCR. - Incapacidad de integrar contexto histórico o datos externos. 	<i>Baja:</i> útil para textos modernos, pero insuficiente para archivos complejos como el Diario de la Marina.

Análisis general y reflexión:

El panorama actual de soluciones que combinan LLMs con documentos digitales muestra un avance notable en la capacidad de las máquinas para procesar e interactuar con texto. Plataformas como **LangChain** y **Chatize**, junto con otras herramientas similares como **Haystack**, **LlamaIndex**, **TextCortex**, **Vidnoz** y **MyMap.AI**, siguen un flujo común: preprocesamiento (limpieza, tokenización), generación de representaciones vectoriales (embeddings) y almacenamiento en bases de datos vectoriales como **FAISS**, **Pinecone** o **Weaviate** [Lewis et al., 2020b, Mikolov et al., 2013, Chatize Team, 2023, Haystack Team, 2023, LlamaIndex Team, 2023]. Estas herramientas implementan técnicas de recuperación aumentada por generación (RAG), donde el sistema recupera fragmentos relevantes de los documentos y genera respuestas basadas en esa información, a menudo presentadas a través de interfaces conversacionales.

Sin embargo, este enfoque, aunque efectivo para documentos contemporáneos, no satisface las demandas de los archivos históricos. La literatura, como el trabajo de Lewis et al. (2020) sobre RAG, demuestra mejoras en la precisión de las respuestas al integrar recuperación de información en tiempo real [Lewis et al., 2020b], pero no aborda los retos específicos de textos antiguos: lenguajes obsoletos, deterioro físico y falta de metadatos estandarizados.

Por ejemplo, en el caso del *Diario de la Marina*, las menciones a puertos o cargamentos requieren no solo una transcripción precisa, sino también una conexión con eventos históricos externos (como las políticas comerciales cubanas del siglo XIX), algo que las soluciones actuales no priorizan. Mientras **Chatize** y **Vidnoz** permiten exploraciones básicas, carecen de capacidades avanzadas para contextualizar eventos pasados. **Adobe Acrobat AI** ofrece análisis más precisos pero no está orientado al detalle histórico. **MyMap.AI** aporta visualizaciones útiles pero poco aprovechables si el OCR falla. Por su parte, **LangChain** y **Haystack** ofrecen una arquitectura flexible que, si se adapta correctamente, puede enfrentar estos desafíos históricos.

Este análisis pone de manifiesto que las soluciones actuales, aunque avanzadas, carecen de la especificidad necesaria para transformar datos históricos no estructurados en conocimiento útil y accesible. La necesidad de integrar tecnologías como OCR, LLMs, bases de datos vectoriales y técnicas RAG en un marco adaptado a las particularidades de los textos antiguos —con énfasis en la corrección de errores, la contextualización histórica y la interacción dinámica— surge como una respuesta lógica a estas limitaciones. Así, el estado del arte no solo valida la pertinencia de explorar un enfoque más especializado, sino que también orienta la selección de herramientas como LangChain o FAISS, que, combinadas estratégicamente, pueden superar los retos identificados y potenciar el análisis de archivos como los del *Diario de la Marina*.

1.8. Enfoque y metodología de desarrollo de software

1.8.1. Análisis del enfoque: Estrella de Boehm–Turner

Para decidir entre un enfoque tradicional o ágil, se aplica la técnica de la Estrella de Boehm–Turner [Boehm-Turner, 2003].

Los cinco ejes relevantes en el proyecto son:

- **Personas vs. Proceso:** Al ser un equipo de sólo una persona con experiencia previa en metodologías ágiles, se prioriza *People* para maximizar la comunicación y la adaptabilidad.
- **Cambios vs. Plan:** El alto grado de incertidumbre y la necesidad de iteraciones frecuentes inclinan la balanza hacia *Change*.
- **Clientes vs. Contrato:** Dada la baja disponibilidad de stakeholders y pocos requisitos regulatorios, se busca un término medio, pero con capacidad de ajustar en función de feedback real.
- **Competencia técnica vs. Gestión del proyecto:** La introducción de tecnologías nuevas y el tratamiento de datos no estructurados refuerzan la preferencia por la *competencia técnica* y la flexibilidad.
- **Control vs. Flexibilidad:** el bajo nivel de criticidad permite una mayor *flexibilidad*.

En conjunto, este análisis muestra que un enfoque **ágil** es el más adecuado para gestionar la incertidumbre, aprovechar la experiencia previa y permitir ajustes rápidos durante los cuatro meses de desarrollo.

1.8.2. Comparativa de metodologías ágiles

A continuación se describen brevemente cuatro candidatas:

Kanban – Flujo continuo de trabajo sin iteraciones fijas. – Ventajas: visualización sencilla, adaptación instantánea a cambios. – Inconvenientes: puede perderse la cadencia de entrega regular y foco en la calidad de las prácticas de ingeniería.

Lean Software Development – Enfoque en eliminar desperdicios, entrega rápida y aprendizaje continuo. – Ventajas: eficiencia en recursos, mejora continua. – Inconvenientes: requiere madurez en prácticas de mejora y métricas precisas.

Dynamic Systems Development Method (DSDM) – Marco iterativo con fases definidas (Feasibility, Foundations, Exploration, Engineering, Deployment). – Ventajas: estructura híbrida que combina planificación y agilidad, roles y entregables claros. – Inconvenientes: algo más pesado en documentación y governance, puede ralentizarse en proyectos muy cortos.

Extreme Programming (XP) – Iteraciones muy cortas (1–2 semanas), énfasis en prácticas de ingeniería (desarrollo dirigido por pruebas, pair programming, integración continua). – Ventajas: máxima calidad de código, feedback inmediato, reducción de defectos. – Inconvenientes: requiere disciplina en el uso de prácticas y cierta colaboración permanente (pairing).

1.8.3. Justificación de la elección: Extreme Programming

Tras comparar las características:

- **Iteraciones cortas y feedback rápido** de XP encajan con el alto grado de incertidumbre y el plazo de cuatro meses.
- **Prácticas de calidad** (TDD, CI, refactoring) minimizan el riesgo al usar tecnologías nuevas y datos no estructurados.
- La **experiencia previa** en ágil facilita la adopción de XP sin curva de aprendizaje excesiva.
- La **flexibilidad** para ajustar requisitos en cada iteración garantiza que, aunque el involucramiento de stakeholders sea bajo, se puedan incorporar descubrimientos técnicos y cambios de alcance de forma continua.

Por ello, se selecciona *Extreme Programming* como la metodología de desarrollo de este proyecto de tesis, ya que ofrece el equilibrio óptimo entre rapidez, adaptabilidad y calidad del software. [[Alliance, 2017](#)]

En conclusión, la adopción de *Extreme Programming* en este proyecto de tesis proporciona un marco sólido y adaptable que se alinea con las necesidades específicas del desarrollo de sistemas de inteligencia artificial aplicados a la interpretación de datos históricos, garantizando una gestión eficiente del proyecto y la entrega de un producto de alta calidad.

1.9. Herramientas y tecnologías

Partiendo de los requerimientos definidos para la implementación del sistema y las características del entorno donde se aplicará la solución propuesta, se realizó un estudio de las tendencias y tecnologías existentes en la actualidad para el desarrollo de sistemas multiagente conversacionales para la interacción con datos no estructurados.

1.9.1. Herramienta de Modelado

Draw.io⁶ es una herramienta de diagramación gratuita y de código abierto que ofrece múltiples ventajas para usuarios que buscan crear diagramas de manera eficiente. Proporciona una amplia gama de plantillas predefinidas y una extensa biblioteca de formas y símbolos, permitiendo la creación de diversos tipos de diagramas, como diagramas de flujo, mapas mentales, organigramas y diagramas UML. Esta versatilidad facilita la adaptación de la herramienta a diferentes necesidades y proyectos. Los diagramas creados en Draw.io pueden ser altamente personalizados en términos de colores, fuentes y estilos, otorgando a cada proyecto una apariencia única y profesional. Además, la herramienta permite exportar los diagramas en diversos formatos, como PNG, JPEG, SVG y PDF, facilitando su incorporación en presentaciones, informes y otros documentos [Ltd, 2025].

1.9.2. Lenguaje de programación

En el desarrollo de sistemas inteligentes para la interpretación y contextualización de datos no estructurados, la elección del lenguaje de programación es un factor crítico que impacta directamente en la eficiencia, flexibilidad y escalabilidad del sistema. En este contexto, Python en su versión 3.12.7 ha sido seleccionado como el lenguaje principal para la implementación del sistema multiagente conversacional destinado a interactuar con los datos del transporte marítimo recogidos en el *Diario de la Marina*. Esta elección se fundamenta en criterios técnicos, científicos y prácticos que garantizan un desarrollo robusto y eficiente. Python es uno de los lenguajes de programación más utilizados en el ámbito de la inteligencia artificial y el procesamiento de lenguaje natural (PLN). Dispone de un ecosistema consolidado de bibliotecas enfocadas en inteligencia

⁶<https://app.diagrams.net/>

artificial, procesamiento de datos y aprendizaje automático. En particular, herramientas como OpenAI API, LangChain y LangGraph ofrecen capacidades avanzadas para la construcción de agentes conversacionales inteligentes, optimizando la interpretación y contextualización de los datos históricos del transporte marítimo [LangChain, 2023]. Dado que la base de datos utilizada en este proyecto contiene información no estructurada extraída mediante OCR de revistas antiguas, es fundamental contar con herramientas especializadas en la manipulación y análisis de texto. Python, a través de bibliotecas como Pandas, NLTK, SpaCy y Transformers, proporciona una infraestructura sólida para la limpieza, estructuración y análisis semántico de grandes volúmenes de texto histórico [Bird et al., 2009, Python, 2023].

1.9.3. Framework de desarrollo

El desarrollo del *Multiagente Conversacional* para la interacción con los datos del transporte marítimo requiere herramientas avanzadas de procesamiento de lenguaje natural (PLN), coordinación de agentes de inteligencia artificial (IA) y una integración eficiente con modelos de lenguaje de gran escala (LLMs). En este contexto, se ha optado por utilizar LangChain como tecnología base, complementada por Langraph, que extiende sus capacidades hacia una arquitectura de orquestación multiagente más robusta.

LangChain (0.3.21) se presenta como una librería fundamental para la creación de aplicaciones basadas en LLMs, ofreciendo una estructura modular y flexible que se adapta perfectamente a los requisitos de este proyecto. Una de sus principales características es el manejo eficiente del contexto y la memoria, lo que permite almacenar y recuperar información relevante de interacciones previas, algo esencial para gestionar consultas en lenguaje natural sobre los datos históricos del transporte marítimo. Además, LangChain facilita la integración con bases de datos y la recuperación de información, mediante técnicas como la recuperación aumentada por generación (RAG), lo que permite enriquecer las respuestas con fragmentos relevantes de documentos digitalizados mediante OCR. Su capacidad para diseñar flujos conversacionales mediante chains y agentes personalizados es otro punto clave, ya que permite estructurar interacciones complejas de manera lógica y controlada. Finalmente, LangChain es compatible con una variedad de modelos de lenguaje de OpenAI y Hugging Face, lo que ofrece la flexibilidad necesaria para experimentar con distintas opciones y optimizar el rendimiento del sistema de agentes [Team, 2023a].

Por otro lado, Langraph (0.3.21) extiende las capacidades de LangChain al proporcionar una capa adicional de

orquestación que gestiona la interacción entre múltiples agentes mediante grafos de estados. Esta herramienta es fundamental para coordinar el flujo de trabajo entre agentes especializados, como los que se encargan de extraer datos históricos y aquellos que interpretan la información contextual. Langraph permite definir de manera clara las transiciones y reglas de decisión entre los distintos agentes, lo que optimiza la eficiencia y evita redundancias o errores en el procesamiento. Además, su diseño visual basado en grafos facilita el control y la visualización del ciclo conversacional, permitiendo una gestión precisa de la interacción entre los diferentes agentes. Esto resulta crucial para asegurar que el sistema sea escalable y eficiente a medida que se integran más agentes con distintas funciones dentro del sistema multiagente. La capacidad de Langraph para gestionar dinámicamente el flujo de decisiones entre agentes es una característica clave que mejora la coordinación y rendimiento global del sistema [Team, 2023b].

En conjunto, LangChain y Langraph ofrecen una solución potente y flexible para el desarrollo del sistema multiagente conversacional, permitiendo una integración eficiente con los datos históricos del transporte marítimo y facilitando la orquestación de agentes especializados en tareas complejas de análisis y respuesta.

1.9.4. Control de versiones

GitHub, basado en el sistema de control de versiones Git, permite un seguimiento detallado de cada modificación realizada en los archivos del proyecto. Esta capacidad es esencial para mantener un historial claro y ordenado de la evolución del trabajo, facilitando la identificación y reversión de cambios cuando sea necesario. Además, proporciona una visión transparente de las contribuciones individuales, lo que es crucial en proyectos colaborativos [git, 2024, git, 2021].

1.9.5. Vectorización y Almacenamiento

El procesamiento de datos en el sistema de multiagentes requiere una representación precisa y semántica de los documentos históricos. Para lograr esto, se utilizarán técnicas de vectorización que transforman los textos en representaciones numéricas mediante embeddings generados por modelos de lenguaje preentrenados, como los disponibles en Hugging Face. Estos embeddings capturan no solo el contenido literal del texto, sino también su significado semántico y las relaciones contextuales entre las palabras, lo que resulta fundamental cuando se trata de interpretar datos históricos, como aquellos del Diario de la Marina, que pueden contener referen-

cias específicas a eventos pasados o terminología arcaica [Devlin et al., 2018, Reimers and Gurevych, 2019]. La capacidad de estos modelos para capturar relaciones contextuales profundas entre palabras y frases es una ventaja clave cuando se trabaja con textos que tienen un vocabulario especializado o en desuso.

Una vez generados los embeddings, es esencial contar con un sistema que permita su almacenamiento y recuperación eficiente. En este caso, se opta por utilizar una base de datos vectorial como FAISS, que está diseñada para manejar grandes volúmenes de vectores y realizar búsquedas de alta eficiencia [Mikolov et al., 2013]. FAISS no solo facilita la indexación de los embeddings, sino que también optimiza la búsqueda por similitud semántica, permitiendo una recuperación rápida de fragmentos relevantes de texto que se alinean con las consultas de los usuarios. Esta capacidad es crucial en el contexto histórico, donde las consultas pueden involucrar la búsqueda de información relacionada con términos que no se encuentran en los documentos de manera directa, sino a través de sus vínculos semánticos [Mikolov et al., 2013].

Por tanto, la integración de estas tecnologías no solo facilita la recuperación de información relevante, sino que también permite mantener la precisión semántica, un factor crítico en el análisis de datos históricos, donde los detalles específicos son clave para la correcta interpretación de los textos [Mikolov et al., 2013]. En este sentido, la vectorización y el almacenamiento en FAISS ofrecen una solución robusta y escalable para abordar los desafíos de la recuperación de información en documentos históricos complejos, garantizando tanto la eficiencia como la precisión en el contexto de grandes volúmenes de datos no estructurados.

Conclusiones del capítulo

Este capítulo ha establecido los fundamentos teóricos y tecnológicos esenciales para el desarrollo de un sistema multiagente conversacional diseñado para procesar y analizar los datos históricos del *Diario de la Marina* (1844–1960). A lo largo de este capítulo, se exploraron diversas tecnologías clave, incluyendo el reconocimiento óptico de caracteres (OCR), los modelos de lenguaje de gran escala (LLMs), sistemas multiagente, Recuperación Aumentada por Generación (RAG), técnicas de *embeddings* y bases de datos vectoriales. Se demostró cómo estas tecnologías tienen el potencial de transformar grandes volúmenes de información no estructurada en conocimiento útil y accesible.

El análisis comparativo de las soluciones existentes, como Chatize, ha puesto de manifiesto que, aunque

efectivas en contextos contemporáneos, estas plataformas no son adecuadas para abordar los desafíos específicos de los documentos históricos. Esto se debe a su incapacidad para manejar de manera efectiva las particularidades de los textos antiguos, como errores de OCR derivados de tipografías obsoletas o la falta de contextualización histórica.

Este enfoque especializado, combinado con la metodología de desarrollo ágil Extreme Programming (XP), permitirá superar estos obstáculos. Las herramientas tecnológicas seleccionadas, como Python 3.12.7, LangChain, LangGraph y FAISS, ofrecen una infraestructura robusta y flexible para la implementación del sistema. Estas herramientas no solo facilitan la construcción de un sistema multiagente eficiente, sino que también permiten una integración efectiva de las tecnologías necesarias para interpretar y contextualizar los datos históricos de manera precisa.

CAPÍTULO 2

Análisis, diseño e implementación de la propuesta de solución

Este capítulo presenta la propuesta de solución para abordar el problema identificado en la investigación: la interpretación y contextualización de datos no estructurados del transporte marítimo extraídos del Diario de la Marina (1844–1960). Aquí se detallan los requisitos funcionales (RF) y no funcionales (RNF) que el sistema debe cumplir, así como una descripción exhaustiva de la solución propuesta, incluyendo las historias de usuario y las tarjetas CRC (Clase-Responsabilidad-Colaboración). Además, se analizan los patrones arquitectónicos y de diseño seleccionados para estructurar la aplicación, junto con las buenas prácticas de codificación asociadas a las tecnologías empleadas. Finalmente, se incluyen el diagrama de clases y el diagrama de componentes, que ilustran los elementos clave que integran esta propuesta. Este capítulo sienta las bases técnicas y conceptuales para la implementación y validación descritas en capítulos posteriores, alineándose con los objetivos específicos de diseño e implementación establecidos en la introducción.

2.1. Descripción de la Propuesta de Solución

La solución propuesta aborda la interacción con los datos históricos del transporte marítimo del *Diario de la Marina* (1844-1960) mediante el desarrollo de una aplicación web moderna, separando claramente las responsabilidades entre la interfaz de usuario, la lógica de negocio y el procesamiento avanzado de lenguaje

natural. Esta arquitectura se compone de tres elementos principales: un *frontend* desarrollado con React js, un *backend* API construido con Django REST Framework(DRF), y un microservicio dedicado que alberga el sistema multiagente conversacional basado en inteligencia artificial y que se expone con una api creada con FastApi.

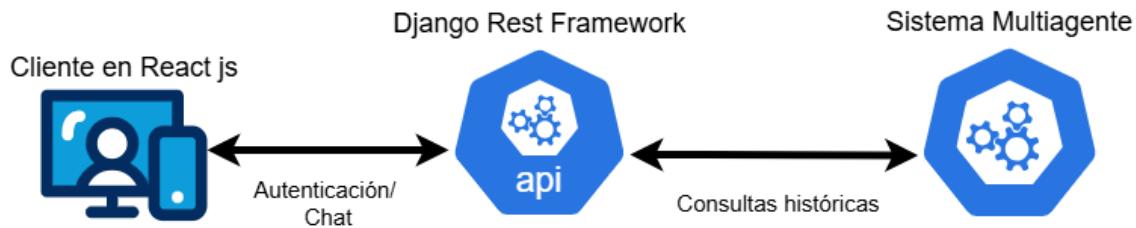


Figura 2.1. Estructura de la propuesta de solución (Fuente: elaboración propia).

La Figura 2.1 ilustra la arquitectura general. El flujo de interacción del usuario es el siguiente:

1. El usuario interactúa con la interfaz web desarrollada en React, donde ingresa sus consultas en lenguaje natural (e.g., “¿Qué barcos llegaron de Europa en enero de 1850?”).
2. El *frontend* React envía la consulta del usuario, típicamente como una petición HTTPS, al *endpoint* correspondiente del *backend* de DRF.
3. El *backend* de DRF actúa como orquestador principal de la lógica de negocio. Puede gestionar autenticación de usuarios (si aplica), almacenar historial de conversaciones, y, crucialmente, procesa la solicitud entrante. Determina que la consulta requiere procesamiento por IA y la reenvía al microservicio del sistema multiagente a través de una llamada API.
4. El microservicio del sistema multiagente recibe la consulta y ejecuta su lógica interna basada en agentes para procesar el lenguaje natural, buscar en la base de datos vectorial del *Diario de la Marina*, contextualizar la información y generar la respuesta (texto y/o imágenes).
5. El microservicio del sistema multiagente devuelve el resultado procesado al *backend* de DRF.
6. El *backend* de DRF recibe la respuesta del microservicio, la formatea si es necesario (e.g., preparándola para la visualización), y la envía de vuelta al frontend React.
7. Finalmente, el *frontend* React recibe la respuesta del *backend* de DRF y la presenta al usuario de forma clara y ordenada en la interfaz de chat, mostrando el texto y/o las imágenes generadas.

Dentro del microservicio sistema multiagente opera la lógica conversacional avanzada, cuyo flujo interno se detalla a continuación y se ilustra en la Figura 2.2. Este microservicio está diseñado específicamente para

transformar los datos históricos no estructurados del *Diario de la Marina* en conocimiento estructurado y accesible, superando las limitaciones de herramientas previas mediante la integración de LLMs¹, RAG² y una coordinación eficiente entre agentes especializados.

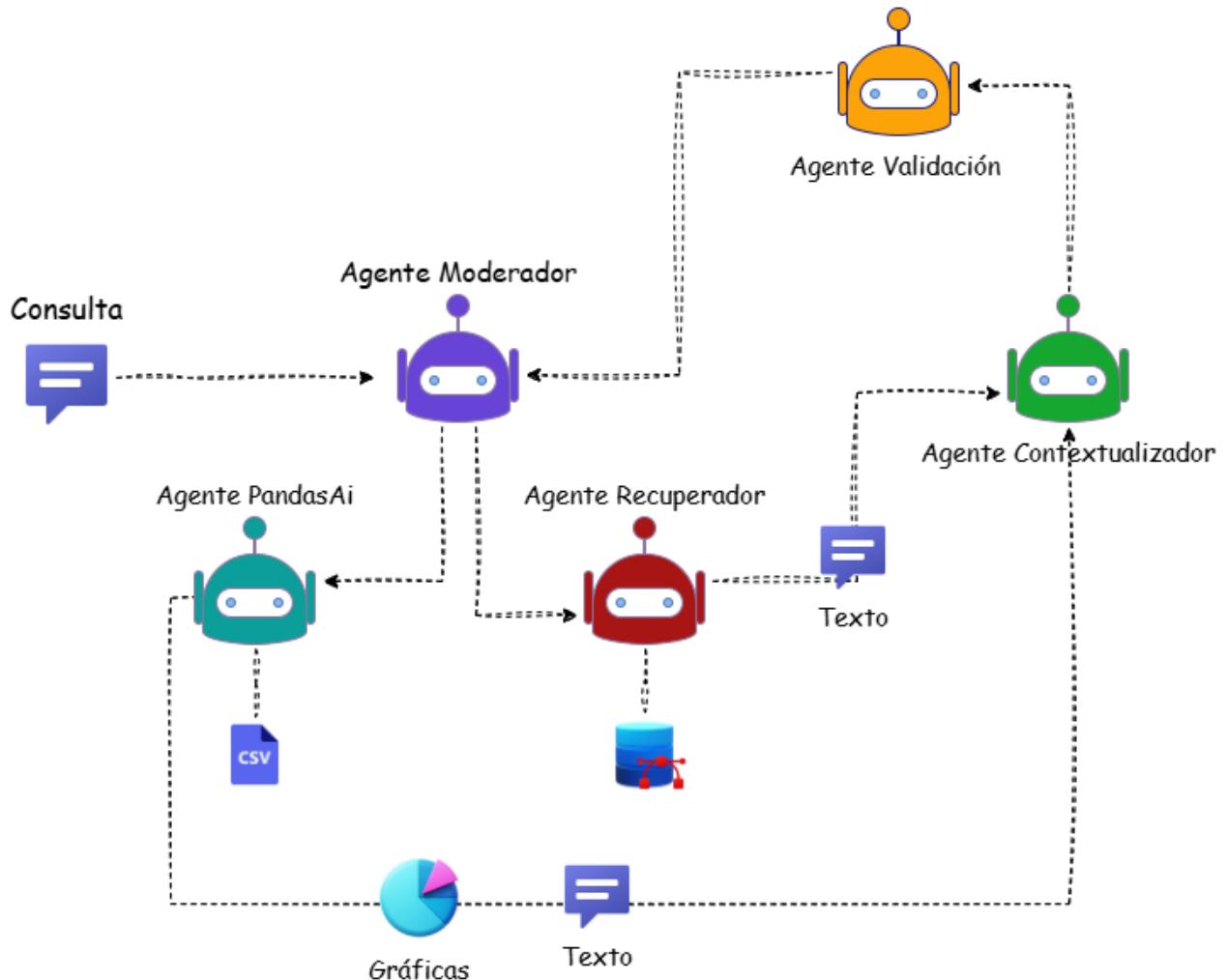


Figura 2.2. Flujo Interno del Microservicio Multiagente Conversacional (Fuente: elaboración propia).

El proceso interno del microservicio (Figura 2.2) inicia cuando recibe una consulta del *backend* DRF. Esta consulta es manejada por el Agente Moderador, que actúa como coordinador central dentro del microservicio. El Agente Moderador extrae palabras clave (e.g., “barcos”, “Europa”, “enero 1850”) y determina la intención

¹LLM (Large Language Model): Modelo de lenguaje de gran tamaño entrenado con enormes volúmenes de datos textuales, capaz de comprender y generar lenguaje natural con un alto nivel de coherencia, usado en tareas como la traducción automática, el resumen de textos o la generación de respuestas conversacionales.

²RAG (Retrieval-Augmented Generation): Técnica que combina modelos generativos con mecanismos de recuperación de información, permitiendo al modelo acceder a fuentes externas relevantes durante la generación de respuestas, mejorando así la precisión y actualidad de los resultados producidos.

del usuario. En la mayoría de los casos, delega la tarea de búsqueda al Agente PandasAi, que realiza la extracción de datos estructurados (como fechas, nombres de capitanes, puertos) desde el archivo CSV que contiene los datos procesados del *Diario de la Marina*. Solo en casos específicos, cuando la consulta requiere información detallada sobre el contenido de la carga de los barcos, el Agente Moderador coordina con el Agente Recuperador, que convierte las palabras clave en *embeddings* y consulta la base de datos vectorial especializada para recuperar fragmentos relevantes. Los datos recuperados, ya sea por el Agente PandasAi o el Agente Recuperador, se devuelven al Agente Moderador. Posteriormente, el Agente Moderador envía la información recuperada, junto con la intención del usuario, al Agente Contextualizador. Este agente evalúa si se necesita una respuesta textual, una visualización gráfica (e.g., un gráfico de la frecuencia de llegada de barcos por mes), o ambas. Si se requieren gráficos, el Agente Contextualizador genera instrucciones que se envían al Agente PandasAi. Este agente crea un script Python para generar la visualización, valida su corrección y genera una imagen (e.g., PNG). La imagen resultante se retorna al Agente Contextualizador. Si no se requieren gráficos, el Agente Contextualizador enriquece la información textual. Antes de enviar la respuesta final al *backend DRF*, el Agente Validacion revisa la coherencia y precisión del contenido generado (texto y/o imagen), comparándolo con la consulta original. Finalmente, el Agente Moderador ensambla la respuesta validada y la retorna como salida del microservicio. Esta arquitectura desacoplada, combinando una interfaz web moderna, un backend robusto y un microservicio especializado en IA, no solo facilita la interacción del usuario y la gestión de datos, sino que también optimiza la investigación histórica, contribuye a la preservación del patrimonio documental y permite escalar los componentes de IA de forma independiente.

2.2. Análisis de requisitos

El análisis de requisitos da como resultado la especificación de las características operativas del software. Indica la interfaz de este y otros elementos del sistema, y establece las restricciones que limitan al software [Pressman et al., 2010]. Es una fase crucial en el proceso de desarrollo de software. Se trata de una etapa inicial en la cual un analista busca entender las necesidades del cliente y traducirlas en un conjunto de requisitos claros y bien definidos [Palli, 2023].

2.2.1. Técnicas de captura de requisitos

La definición de los requisitos del sistema se fundamenta en un proceso sistemático de captura, basado en dos técnicas ampliamente reconocidas en ingeniería de software: la entrevista y la observación, aplicadas en el contexto de los ejemplos analizados en el estado del arte (Capítulo 1). Estas técnicas, permitieron identificar las expectativas de los usuarios potenciales y las limitaciones de las soluciones existentes, asegurando que los requisitos reflejen tanto las demandas prácticas como las carencias técnicas observadas [Sommerville, 2011].

Entrevista: Se realizaron entrevistas semiestructuradas con historiadores y académicos especializados en historia del Caribe, quienes serían usuarios finales del sistema. Las preguntas se diseñaron para explorar sus necesidades al interactuar con documentos históricos digitalizados, como el *Diario de la Marina*. Por ejemplo, se les consultó: “¿Qué tipo de información busca con mayor frecuencia en archivos históricos?” y “¿Qué dificultades encuentra al analizar datos marítimos de textos antiguos?”. Las respuestas destacaron la importancia de obtener respuestas contextualizadas (e.g., vinculadas a eventos históricos), la necesidad de visualizaciones gráficas para patrones comerciales y la frustración con errores de transcripción que dificultan el análisis. Estas aportaciones guiaron la definición de requisitos como la corrección de datos y la generación de gráficos.

Análisis de Sistema Homólogos: Mediante el análisis de sistemas existentes es posible estudiar aplicaciones similares a la que se necesita obtener. Cuando se tiene la concepción del funcionamiento de un software similar en cuanto a funcionalidades y características es más sencillo identificar los requisitos del sistema que se necesita implementar. Durante la investigación se realizó un estudio de aplicaciones similares a la solución a desarrollar, en las cuales se observaron los diseños de sus interfaces, las funcionalidades que ofrecen, el grado de dificultad a la hora de interactuar con la aplicación, entre otros rasgos importantes que contribuyeran a obtener un producto con la mejor calidad posible [Sommerville, 2011]. Como fuente importante para la obtención de requisitos principales del sistema a desarrollar, se encuentra el análisis a fondo de los sistemas homólogos estudiados y antes descritos en el epígrafe 1.2.

La combinación de entrevistas y observación permitió comprender las necesidades de los usuarios con las deficiencias técnicas de las soluciones actuales, proporcionando una base sólida para los requisitos funcionales y no funcionales que se detallan a continuación.

2.2.2. Requisitos funcionales

Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer [Sommerville, 2011].

A continuación, se describen los requisitos funcionales específicos para el sistema multiagente conversacional propuesto:

Tabla 2.1. Tabla de Requisitos Funcionales (RF)

ID	Nombre	Descripción	Complejidad	Prioridad
RF 1	Registrar usuario	Permitir a un visitante registrarse proporcionando datos válidos (e.g., nombre de usuario, email, contraseña), que serán almacenados de forma segura por el backend.	Media	Alta
RF 2	Autenticar usuario	Permitir a un usuario registrado iniciar sesión proporcionando credenciales válidas, verificadas por el backend, generando una sesión activa.	Media	Alta
RF 3	Cerrar sesión	Permitir al usuario autenticado finalizar su sesión activa en el sistema.	Baja	Media
RF 4	Iniciar conversación	Permitir al usuario autenticado crear una nueva sesión de chat independiente de las anteriores.	Baja	Alta
RF 5	Listar conversación	Mostrar al usuario autenticado una lista de sus conversaciones previas para poder seleccionarlas y revisarlas.	Media	Media
RF 6	Visualizar conversación	Permitir al usuario seleccionar una conversación de su historial para visualizarla y continuarla.	Media	Media
RF 7	Eliminar conversación	Permitir al usuario eliminar permanentemente una conversación específica de su historial.	Media	Baja

Continúa en la siguiente página...

Tabla 2.1. Tabla de Requisitos Funcionales (RF) (Continuación)

ID	Nombre	Descripción	Complejidad	Prioridad
RF 8	Insertar consulta	Permitir al usuario escribir y enviar una consulta en lenguaje natural dentro de la conversación activa.	Baja	Alta
RF 9	Visualizar mensajes	Mostrar de forma clara y ordenada el diálogo (consultas del usuario y respuestas del sistema) dentro de la conversación activa.	Media	Alta
RF 10	Recibir consulta	El microservicio MAS debe recibir la consulta para iniciar el flujo de agentes.	Media	Alta
RF 11	Analizar consulta	El Agente Moderador debe analizar la consulta para identificar términos clave relevantes.	Alta	Alta
RF 12	Determinar acción	El Agente Moderador debe interpretar el propósito principal de la consulta del usuario para determinar la próxima acción del sistema (información, análisis, gráficas estadísticas etc.).	Alta	Alta
RF 13	Generar embeddings	El Agente Recuperador debe convertir las palabras clave en representaciones vectoriales (embeddings) adecuadas para la búsqueda semántica.	Alta	Alta
RF 14	Consultar base de datos	El Agente Recuperador debe buscar y obtener fragmentos de texto relevantes del <i>Diario de la Marina</i> desde la base de datos vectorial, basándose en los embeddings.	Alta	Alta
RF 15	Consultar CSV	El Agente PandasAi debe buscar y obtener fragmentos de texto relevantes del <i>Diario de la Marina</i> desde la base de datos en formato csv.	Alta	Alta

Continúa en la siguiente página...

Tabla 2.1. Tabla de Requisitos Funcionales (RF) (Continuación)

ID	Nombre	Descripción	Complejidad	Prioridad
RF 16	Contextualizar respuesta	El Agente Contextualizador debe generar una respuesta coherente en lenguaje natural, integrando la información recuperada y añadiendo contexto histórico si es pertinente.	Alta	Alta
RF 17	Identificar estadísticas	El Agente Contextualizador debe determinar si la consulta o los datos recuperados justifican la creación de una representación gráfica.	Media	Media
RF 18	Formular instrucciones para gráfico	Si se requiere un gráfico, el Agente Contextualizador debe generar las especificaciones (tipo de gráfico, datos a usar) para el Agente PandasAi.	Alta	Alta
RF 19	Generar script	El Agente PandasAI debe crear un script ejecutable que produzca la visualización solicitada a partir de los datos y especificaciones.	Alta	Alta
RF 20	Validar script	El Agente PandasAi debe verificar que el script generado es sintácticamente correcto y no contiene errores obvios antes de ejecutarlo.	Alta	Alta
RF 21	Generar imagen	El Agente PandasAi debe ejecutar el script validado para generar la visualización como un archivo de imagen (e.g., PNG, JPG).	Alta	Alta
RF 22	Combinar resultados	El Agente Contextualizador (o Moderador) debe combinar la respuesta textual y/o la imagen generada en una estructura de respuesta unificada.	Alta	Alta
RF 23	Validar respuesta	El Agente de Validación debe revisar la respuesta preliminar para asegurar su precisión, coherencia con la consulta y ausencia de información errónea.	Alta	Alta

Continúa en la siguiente página...

Tabla 2.1. Tabla de Requisitos Funcionales (RF) (Continuación)

ID	Nombre	Descripción	Complejidad	Prioridad
RF 24	Enviar respuesta	El microservicio MAS debe enviar la respuesta final ensamblada al backend a través de su API.	Media	Alta
RF 25	Visualizar respuesta	El backend debe enviar la respuesta (texto y/o referencia a la imagen) al frontend para su visualización.	Media	Alta

2.2.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades de este como fiabilidad, tiempo de respuesta y la capacidad de almacenamiento. Incluyen además restricciones de tiempo, sobre el proceso de desarrollo y estándares [Sommerville, 2011]. A continuación, se definen los requisitos no funcionales que debe cumplir la aplicación basándose en los establecido por las normas ISO 25000 Calidad del Producto de Software, específicamente la ISO/IEC 25010 que define las características de calidad que se tienen en cuenta al evaluar las propiedades de un producto de software [iso, 2023]. A continuación, se listan los requisitos no funcionales identificados:

Tabla 2.2. Requisitos No Funcionales (RNF)

ID	Descripción
RNF 1: Rendimiento	
RNF 1.1	Tiempo de respuesta para consultas simples (búsqueda de barcos por nombre o fecha) no superior a los 30 segundos para 20 usuarios concurrentes.
RNF 1.2	Tiempo de respuesta para consultas complejas con generación de gráficos no superior a un minuto para 20 usuarios concurrentes.
RNF 2: Seguridad	
RNF 2.1	Almacenamiento seguro de contraseñas de usuarios
RNF 2.2	Rastreabilidad de decisiones tomadas por los agentes IA

Tabla 2.2. Requisitos No Funcionales (RNF) (continuación)

ID	Descripción
RNF 2.3	El acceso a la información debe estar restringido por usuario, contraseña.
RNF 3: Usabilidad	
RNF 3.1	Retroalimentación visual durante el procesamiento de consultas
RNF 3.2	Mensajes de error claros y comprensibles para usuarios
RNF 4: Fiabilidad	
RNF 4.1	Alta disponibilidad del sistema completo
RNF 4.2	El sistema debe ser tolerante a fallos, y mostrar solo la información necesaria para orientar al usuario
RNF 5: Mantenibilidad	
RNF 5.1	El software estará bien documentado de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarse.
RNF 5.2	Se debe hacer uso de los estándares de codificación definidos para el sistema multi-agente
RNF 5.3	Gestión controlada de dependencias

2.2.4. Historias de usuarios

Las historias de usuario (HU) son descripciones breves y simples de los requerimientos de un cliente o usuario, que facilitan la comunicación con los desarrolladores del proyecto. Estas historias permiten expresar las expectativas y necesidades de los usuarios de una manera clara y comprensible, evitando ambigüedades y malentendidos que podrían llevar a pérdidas de tiempo y recursos [Menzinsky et al., 2018].

Se realiza una HU por cada RF del componente, a continuación, se mostrarán las HU más relevantes. Para consultar el resto de las historias de usuario, se remite al **Anexo A**, donde se presentan de forma completa y organizada. Estas historias complementan la visión general del sistema y permiten una comprensión más exhaustiva de los requerimientos funcionales detallados.

Tabla 2.3. Historia de usuario # 1

Historia de usuario	
Número: 1	Nombre: Interactuar con el chat activo (Enviar Consulta y Ver Respuesta)
Usuario: Usuario autenticado	
Prioridad en negocio: Alta	Riesgo en desarrollo: Medio
Puntos estimados: 2 semanas	Iteración asignada: 1
Programador responsable: Daniel Rojas Grass	
Descripción: Como usuario autenticado, quiero poder escribir una consulta en lenguaje natural en la interfaz de chat, enviarla al sistema, y ver tanto mi consulta como la respuesta del sistema (texto y/o imagen) mostradas de forma clara y ordenada en el área de diálogo de la conversación activa. (Corresponde principalmente a RF9-Input, RF10, RF26, RF27, RF28)	
Precondiciones:	
<ul style="list-style-type: none"> • El usuario tiene una sesión activa. • El usuario tiene una conversación activa (nueva HU:04 o cargada HU:06). • El <i>frontend</i>, el <i>backend</i> y el microservicio MAS están operativos y pueden comunicarse entre sí. 	
Flujo de acción:	
<ol style="list-style-type: none"> 1. Usuario escribe una consulta en el campo de texto del chat. 2. Usuario envía la consulta (click en botón 'Enviar' o presiona 'Enter'). 3. El <i>frontend</i> muestra inmediatamente la consulta del usuario en el área de diálogo (marcada como del usuario). 4. El <i>frontend</i> envía la consulta y el ID de la conversación activa (si existe) al <i>backend</i>. 5. (Flujo cubierto en HU:09 y HU:10) El <i>backend</i> recibe la consulta, la envía al microservicio MAS para procesamiento, recibe la respuesta del MAS (texto y/o referencia a imagen) y la almacena en la BD asociada a la conversación (RF9-Persistencia). 6. El <i>backend</i> envía la respuesta procesada (texto y/o URL/datos de imagen) al <i>frontend</i>. 7. El <i>frontend</i> recibe la respuesta del backend. 8. El <i>frontend</i> muestra la respuesta del sistema (texto y/o imagen) en el área de diálogo (marcada como del sistema). 	
Observaciones: La interfaz debe indicar visualmente cuándo el sistema está procesando la respuesta. El manejo de errores (fallos de red, errores del MAS) debe ser robusto, mostrando mensajes apropiados al usuario. La visualización del chat debe permitir scroll para ver mensajes antiguos.	

Continúa en la próxima página

Tabla 2.3. Continuación de la página anterior

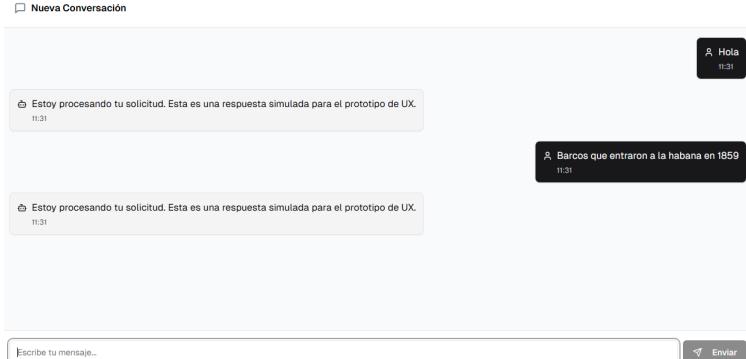
Interfaz:	Interfaz principal del Chat en el sitio web mostrando diálogo:
	

Tabla 2.4. Historia de usuario # 2

Historia de usuario	
Número: 2	Nombre: Obtener respuesta textual relevante del MAS
Usuario: Usuario autenticado (indirectamente, a través del sistema)	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alto
Puntos estimados: 4 semanas	Iteración asignada: 2
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla 2.4. Continuación de la página anterior

<p>Descripción: Como sistema (actuando en nombre del usuario), quiero que el microservicio MAS procese una consulta recibida del <i>backend</i>, la analice (palabras clave, intención), recupere información relevante del Diario de la Marina desde la BD vectorial, sintetice una respuesta textual coherente y contextualizada, la valide y la devuelva al <i>backend</i>, para que el usuario final reciba información precisa. (Corresponde al flujo interno del MAS: RF11, RF12, RF13, RF14, RF15, RF16, RF22, RF23, RF24, RF25)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El microservicio MAS (FastAPI) está operativo. • Todos los agentes internos (Moderador, Pandasai, Contextualizador, Validación) están implementados y disponibles. • La base de datos vectorial (Faiss) está cargada con los datos del *Diario de la Marina* y es accesible. • El csv con los datos del <i>Diario de la Marina</i> esta cargado y es accesible. • El LLM subyacente está configurado y accesible. • El MAS recibe una solicitud válida del <i>backend</i> a través de su API. <p>Flujo de acción (interno del MAS):</p> <ol style="list-style-type: none"> 1. MAS recibe la consulta vía API (RF11). 2. Agente Moderador extrae palabras clave y determina intención (RF12, RF13). 3. Agente Recuperador genera <i>embeddings</i> y busca en BD vectorial (RF14, RF15). 4. Agente PandasAi genera tanto gráficas como búsquedas de análisis en el csv. 5. Agente Contextualizador recibe fragmentos relevantes y genera respuesta textual inicial, añadiendo contexto (RF16). 6. (Si no se requiere gráfico) Agente Contextualizador (o Moderador) prepara la respuesta textual preliminar (RF22). 7. Agente Validación revisa coherencia, relevancia y posible toxicidad/error (RF23). 8. Agente Moderador ensambla la respuesta textual validada en formato JSON (RF24). 9. MAS retorna la respuesta JSON al backend DRF (RF25). <p>Observaciones: La calidad de los embeddings y la estrategia de recuperación son críticas (RF14, RF15). La capacidad del LLM para sintetizar y contextualizar sin ^alucinar_{es} fundamental (RF16). El agente de validación es clave para la fiabilidad (RF23). La latencia del proceso completo debe ser aceptable (considerar RNF).</p> <p>Interfaz:</p> <p>[N/A - Proceso interno del Microservicio MAS]</p>

Tabla 2.5. Historia de usuario # 3

Historia de usuario	
Número: 3	Nombre: Recibir visualización gráfica cuando sea pertinente
Usuario: Usuario autenticado	
Prioridad en negocio: Media	Riesgo en desarrollo: Moderado
Puntos estimados: 2 semanas	Iteración asignada: 3
Programador responsable: Daniel Rojas Grass	
<p>Descripción: Como usuario autenticado, quiero que cuando mi consulta o la información recuperada sugieran la necesidad de una visualización (e.g., análisis de tendencias, comparación de datos), el sistema (específicamente el MAS) genere un gráfico apropiado (e.g., barras, líneas) y me lo presente como una imagen dentro de la respuesta del chat, junto con el texto explicativo, para facilitar mi comprensión. (Corresponde principalmente a RF17, RF18, RF19, RF20, RF21, RF22-integración, RF28-visualización)</p>	
<p>Precondiciones:</p> <ul style="list-style-type: none"> • El flujo de HU:09 está en progreso. • El Agente Contextualizador identifica que la consulta/datos justifican un gráfico (RF17). • Los datos necesarios para el gráfico están disponibles y estructurados. • El Agente PandasAi esta disponible para la generación de gráficas. 	
<p>Flujo de acción (continuación de HU:09):</p> <ol style="list-style-type: none"> 1. Agente Contextualizador determina necesidad de gráfico y formula instrucciones (tipo, datos) (RF17, RF18). 2. Agente Contextualizador invoca al Agente PandasAi con las instrucciones. 3. Agente PandasAi genera el script de Pandas para crear el gráfico (RF19). 4. Agente PandasAi valida la sintaxis y lógica básica del script (RF20). 5. Agente PandasAi ejecuta el script validado para generar la imagen del gráfico (e.g., PNG) (RF21). 6. Agente PandasAi devuelve la imagen (o una referencia a ella) al Agente Contextualizador/Moderador. 7. Agente Contextualizador/Moderador integra la imagen junto con la respuesta textual en la estructura preliminar (RF22). 8. (Continúa flujo de HU:09) Validación (RF23), Ensamblaje (RF24), Retorno al <i>backend</i> (RF25). 9. (Flujo de HU:08) <i>Backend</i> envía URL/datos de imagen al <i>frontend</i>. 10. <i>Frontend</i> muestra la imagen recibida dentro del chat (RF28). 	

Continúa en la próxima página

Tabla 2.5. Continuación de la página anterior

Observaciones: La detección de la necesidad de un gráfico debe ser fiable (RF17). La generación del script debe ser segura (evitar ejecución de código arbitrario). La validación del script (RF20) es importante para evitar errores en tiempo de ejecución. Los gráficos deben ser claros y legibles. Considerar formatos de imagen web-friendly (PNG, JPG, SVG).
Interfaz: Visualización de imagen (gráfico) dentro del Chat en el sitio web, junto al texto explicativo.

Las historias de usuario presentadas detallan los requerimientos funcionales del sistema desde la perspectiva del usuario, abarcando desde la gestión de usuarios y autenticación hasta la interacción con el chat y la generación de respuestas textuales y gráficas por parte del microservicio MAS. Estas historias no solo especifican el comportamiento esperado del sistema, sino que también establecen una base clara para el diseño y desarrollo del software, asegurando que las necesidades del usuario se traduzcan en funcionalidades concretas.

2.2.5. Tarjetas CRC

Las tarjetas CRC (Clase-Responsabilidad-Colaboración) son una herramienta de diseño de software orientado a objetos, creada por Kent Beck y Ward Cunningham. Estas tarjetas se utilizan para identificar las clases, sus responsabilidades y cómo colaboran con otras clases para cumplir tareas específicas en un sistema [Beck and Cunningham, 1989]. La representación del sistema multiagente y su interfaz gráfica mediante tarjetas CRC permite estructurar de forma clara y concisa las clases que lo componen, sus responsabilidades específicas y las colaboraciones necesarias para cumplir sus objetivos. Esta técnica facilita la comprensión del comportamiento de cada agente dentro del sistema —como el coordinador, el buscador de información o el generador de respuestas—, promoviendo un diseño orientado a objetos coherente, reutilizable y fácil de mantener. Además, al emplearse en etapas tempranas del desarrollo, las tarjetas CRC fortalecen la comunicación entre desarrolladores y respaldan la validación del modelo antes de su implementación definitiva.

Tabla 2.6. Tarjeta CRC: Agente Moderador

Tarjeta CRC	
Clase	Agente Moderador
Responsabilidades:	Colaboración:
Recibir la consulta del usuario	Agente PandasAi
Extraer palabras clave e identificar la intención	Agente recuperador de información (FAISS)
Coordinar el flujo de información entre agentes	Agente Contextualizador
Ensamblar y entregar la respuesta final al usuario	Agente de Validación

Tabla 2.7. Tarjeta CRC: Agente recuperador de información (FAISS)

Tarjeta CRC	
Clase	Agente recuperador de información (FAISS)
Responsabilidades:	Colaboración:
Convertir las palabras clave en <i>embeddings</i>	Agente Moderador
Consultar la base de datos vectorial para recuperar la información relevante	Base de datos vectorial
Devolver los resultados al Agente Moderador	

Tabla 2.8. Tarjeta CRC: Agente Contextualizador

Tarjeta CRC	
Clase	Agente Contextualizador
Responsabilidades:	Colaboración:
Recibir la información relevante y la intención del usuario	Agente Moderador
Generar indicaciones de estadísticas o contextualizar la información según la petición del usuario	Agente PandasAi
Enviar la información procesada al Agente Moderador	

Tabla 2.9. Tarjeta CRC: Agente de Validación

Tarjeta CRC	
Clase	Agente de Validación
Responsabilidades: Revisar la coherencia de la respuesta generada por el sistema Comparar la respuesta con la entrada original Validar o rechazar la respuesta antes de enviarla al usuario	Colaboración: Agente Moderador Agente Contextualizador

Tabla 2.10. Tarjeta CRC: Agente PandasAi

Tarjeta CRC	
Clase	Agente PandasAi
Responsabilidades: Generar un <i>script</i> de Pandas para representar estadísticas gráficas cuando sea necesario Validar el <i>script</i> y asegurarse de que sea correcto Convertir el gráfico generado en una imagen que pueda ser integrada en la respuesta final Hacer consultas al CSV cargado para hacer análisis profundos de datos.	Colaboración: Agente Contextualizador

En este documento se incluyen ejemplos representativos de algunas de las tarjetas CRC más relevantes para el desarrollo de la solución propuesta. Sin embargo, debido a su extensión, se remite al *Anexo B* para consultar el conjunto completo de las tarjetas CRC, donde se detallan todas las clases identificadas, sus responsabilidades y los colaboradores correspondientes.

2.3. Diseño de la propuesta de solución

2.3.1. Diseño de la arquitectura

La arquitectura propuesta adopta el estilo de *microservicios*, en el cual la aplicación se descompone en un conjunto de servicios independientes, cada uno ejecutándose en su propio proceso y comunicándose mediante protocolos API REST [Fowler, 2014c, Wikipedia, 2025]. Cada microservicio se orienta a una capacidad de negocio específica, lo que facilita la comprensión y el mantenimiento aislado de los componentes [Richardson, 2025, Fowler, 2014c]. La modularidad inherente a este enfoque permite el despliegue automatizado e independiente de cada servicio, mejorando la agilidad operativa y reduciendo el tiempo de inactividad asociado a las actualizaciones [Fowler, 2014a, Newman and Fowler, 2021]. La escalabilidad horizontal se ve potenciada, dado que cada servicio puede replicarse de forma autónoma según la demanda, optimizando el uso de recursos y posibilitando un dimensionamiento granular [Newman and Fowler, 2021, Microsoft, 2021]. El desacoplamiento entre servicios incrementa la tolerancia a fallos, ya que la interrupción de un componente no compromete la disponibilidad del sistema global [Fowler, 2014b, Richardson, 2025]. La heterogeneidad tecnológica está plenamente soportada, puesto que cada microservicio puede implementarse con los lenguajes y frameworks más adecuados para su responsabilidad particular [Richardson, 2025, Microsoft, 2022].

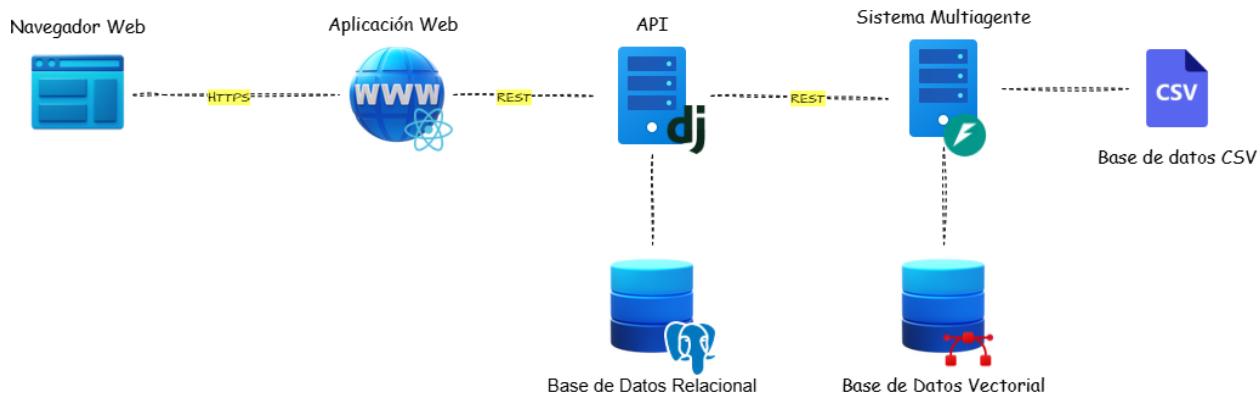


Figura 2.3. Arquitectura de microservicios del sistema multiagente (Fuente: elaboración propia).

La presente configuración arquitectónica se materializa en tres componentes principales como se puede visualizar en la Figura 2.3:

1. Un frontend desarrollado con React, encargado de la interacción con el usuario.

2. Un backend construido con Django REST Framework, que actúa como orquestador de la lógica de negocio y gestor de peticiones.
3. Un microservicio especializado en procesamiento de lenguaje natural, implementado con FastAPI, que alberga el sistema multiagente conversacional.

Esta elección garantiza la independencia en el ciclo de vida de cada servicio, promoviendo la mantenibilidad y facilitando la incorporación de nuevas tecnologías o la sustitución de componentes sin afectar al sistema global [GraphApp.ai, 2025, Microsoft, 2024].

2.3.2. Diseño del modelo de datos

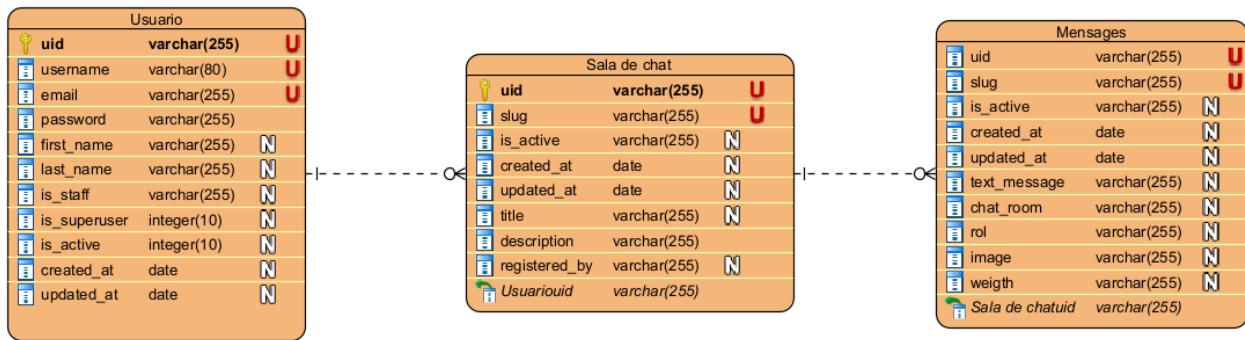


Figura 2.4. Diseño del modelo de datos (Fuente: elaboración propia).

El modelo de datos propuesto en la Figura 2.4 articula de manera coherente la estructura relacional necesaria para soportar un sistema de chat en tiempo real [Codd, 1970]. Se fundamenta en los principios del modelo relacional que establecen las bases para el almacenamiento y la recuperación eficiente de datos, garantizando la unicidad de las tuplas y la ausencia de redundancia innecesaria [Silberschatz et al., 2019].

La definición de la entidad Usuario contempla un identificador único uid, atributos de autenticación y perfil, así como indicadores de estado y marcas temporales de creación y modificación, satisfaciendo los requerimientos de integridad y auditoría en el dominio de usuario [Elmasri and Navathe, 2010]. La elección de tipos de datos VARCHAR para los campos característicos responde a la flexibilidad necesaria para posibles extensiones en los identificadores y datos personales [Date, 2003].

La entidad Sala de chat permite agrupar conversaciones de manera lógica y filtrable [Connolly and Begg, 2014]. Al establecer una relación de uno a muchos con la entidad Usuario a través de la clave foránea registered_by, se refuerza la trazabilidad de la creación y administración de espacios de comunicación [Harrington, 2015].

La entidad Mensajes está diseñada para almacenar cada mensaje asociado a una sala de chat, con su propio identificador, estado de actividad, contenido textual, atributos de rol y metadatos adicionales que facilitan la moderación y la representación multimodal [Silberschatz et al., 2019]. La implementación de la clave foránea chat_room garantiza la integridad referencial, evitando mensajes huérfanos y asegurando la coherencia de las interacciones [IBM Documentation Contributors, 2025].

El diseño enfatiza la normalización hasta la tercera forma normal, minimizando anomalías de actualización y duplicación de datos [Silberschatz et al., 2019]. Asimismo, la adopción de métodos ágiles para el modelado de datos respalda iteraciones rápidas y adaptativas durante el ciclo de vida del desarrollo [Ambler, 2003]. La adherencia al estándar SQL definido en ISO/IEC 9075 certifica la interoperabilidad en entornos heterogéneos de bases de datos [ISO, 2016]. Finalmente, la consideración de criterios de escalabilidad y rendimiento se inspira en lecciones históricas sobre arquitecturas de sistemas de datos [Stonebraker and Hellerstein, 2005].

2.4. Implementación

2.4.1. Estándares de codificación en Python y JavaScript

Los estándares de codificación son esenciales para garantizar la mantenibilidad, colaboración y calidad del software [van Rossum et al., 2001]. En este epígrafe se analiza comparativamente las convenciones principales para Python y JavaScript, destacando sus similitudes y diferencias fundamentales.

Estándares para Python

1. **Convenciones de Estilo:** Python cuenta con una guía oficial de estilo denominada PEP 8 [van Rossum et al., 2001], que establece directrices precisas para la escritura de código. La indentación de 4 espacios (prohibiendo el uso de tabuladores) y el límite de 79 caracteres por línea son dos de sus reglas más características. El espacio alrededor de operadores y la organización de imports en tres grupos (bibliotecas estándar, terceros y locales) promueven la consistencia visual. La nomenclatura sigue patrones específicos: `snake_case` para funciones y variables, `PascalCase` para clases, y `MAYÚSCULAS` para constantes. La documentación mediante docstrings (siguiendo PEP 257 [Goodger, 2001]) facilita la generación automática de documentación técnica.

2. **Herramientas y Prácticas:** El ecosistema Python ofrece herramientas como `flake8` para verificación de estilo y `black` para formateo automático. El uso de *type hints* (desde Python 3.5) mejora la seguridad de tipos en proyectos complejos. El manejo de excepciones debe ser explícito, evitando cláusulas `except` genéricas que puedan ocultar errores.

Estándares para JavaScript

1. **Guías de Referencia:** JavaScript carece de un estándar oficial único, pero guías como el Airbnb JavaScript Style Guide [[Air, 2022](#)] y Google JavaScript Style Guide [[Goo, 2022](#)] se han consolidado como referencias. Estas enfatizan el uso de ES6+, con preferencia por `const/let` sobre `var`, y arrow functions para callbacks.

La convención de nombres utiliza `camelCase` para variables/funciones y `PascalCase` para clases. Los literales de plantilla (template strings) son preferidos sobre concatenación manual, y el punto y coma sigue siendo opcional pero debe usarse consistentemente.

2. **Ecosistema Moderno:** Herramientas como ESLint (configurable con reglas específicas) y Prettier automatizan la aplicación de estándares. El sistema de módulos ES6 (`import/export`) ha reemplazado ampliamente a `require` en proyectos nuevos. Para proyectos complejos, TypeScript ofrece tipado estático, reduciendo errores en tiempo de ejecución [[Cherny, 2019](#)].

Ambos lenguajes comparten principios fundamentales: modularización del código, documentación clara, y uso de linters. Python muestra mayor uniformidad gracias a PEP 8, mientras JavaScript permite más flexibilidad mediante guías configurables. En rendimiento, JavaScript prioriza la compatibilidad con navegadores, mientras Python enfatiza la legibilidad como principio filosófico [[Peters, 2004](#)]. La adopción de estándares debe adaptarse al contexto del proyecto y equipo, utilizando herramientas automatizadas para garantizar cumplimiento. Tanto Python como JavaScript han desarrollado ecosistemas maduros que, cuando se usan consistentemente, elevan sustancialmente la calidad del código.

2.4.2. Patrones de diseño

Los patrones de diseño de software representan soluciones probadas y estandarizadas para problemas recurrentes en el desarrollo, encapsulando mejores prácticas y promoviendo la creación de software modular,

legible, flexible y robusto [Gavilánez Álvarez et al., 2022]. En el desarrollo de esta aplicación web y su microservicio asociado, se han aplicado conscientemente varios patrones para abordar los desafíos inherentes a su arquitectura distribuida y su flujo de trabajo basado en IA. A continuación, se analizan algunos de los patrones de diseño a implementar en el sistema multiagente conversacional y en el *backend* (desarrollado con Django REST Framework), destacando su relevancia y su impacto en el procesamiento de datos históricos del *Diario de la Marina*.

Patrón Singleton se utiliza para gestionar recursos críticos compartidos en el sistema multiagente, asegurando que componentes como la base de datos históricos, el índice FAISS para la base de datos vectorial, y los agentes tengan una única instancia en toda la aplicación. Este patrón es fundamental para optimizar el uso de recursos, ya que evita la creación redundante de instancias pesadas, lo que podría impactar negativamente el rendimiento del sistema, especialmente al procesar grandes volúmenes de datos históricos del *Diario de la Marina*. La Figura 2.5 muestra un extracto de este código, evidenciando la implementación del patrón. La importancia del patrón *Singleton* radica en su capacidad para centralizar el acceso a recursos compartidos, reduciendo la sobrecarga computacional y asegurando consistencia en los datos procesados. Esto es particularmente relevante en un sistema multiagente, donde múltiples agentes (como *AgenteRecuperador* y *AgentePandasAi*) necesitan acceder a las mismas estructuras de datos para generar respuestas textuales o gráficas.

```
51  def get_faiss_db() → Optional[FAISS]:  
52      """Devuelve la instancia cargada de la base de datos FAISS."""  
53      if _vector_store is None:  
54          load_faiss_index() # Intenta cargar si no lo está  
55      return _vector_store
```

Figura 2.5. Extracto de código que implementa el patrón (Fuente: elaboración propia). *Singleton*.

Patrón Cadena de responsabilidad permite que el sistema multiagente implemente un flujo de trabajo estructurado, que orqueste la ejecución de agentes a través de un grafo de estados. Este patrón permite que cada agente (como el moderador, el contextualizador, el ejecutor PandasAI, y el validador) procese y modifique un estado compartido, pasando el control al siguiente agente en la cadena. Esta estructura es esencial para manejar consultas complejas que requieren el procesamiento conjunto de datos estructurados (del archivo CSV) y no estructurados (de la base vectorial), como se describe en la Figura 2.6.

La relevancia de este patrón radica en su capacidad para modularizar el flujo de procesamiento, permitiendo que cada agente se especialice en una tarea específica. Esto no solo facilita el mantenimiento y la extensibilidad

del sistema, sino que también asegura que las consultas se procesen de manera eficiente, cumpliendo con los requisitos funcionales relacionados con la generación de respuestas.

```

14 |     # --- Funciones Nodo para Langraph ---
15 |     # --- NODO EJECUTOR MODERADOR ---
16 | > def run_moderator(state: GraphState) → Dict[str, Any]: ...
17 |
18 |     # --- NODO EJECUTOR PANDASAI ---
19 | > def run_pandasai_executor(state: GraphState) → Dict[str, Any]: ...
20 |
21 |     # --- NODO EJECUTOR CONTEXTUALIZADOR ---
22 | > def run_contextualizer(state: GraphState) → Dict[str, Any]: ...
23 |
24 |     # --- NODO EJECUTOR VALIDADOR ---
25 | > def run_validator(state: GraphState) → Dict[str, Any]: ...
26

```

Figura 2.6. Extracto de código que implementa el patrón *Cadena de responsabilidad* (Fuente: elaboración propia).

Patrón Plantilla Abstracta permite estandarizar comportamientos comunes en modelos, serializadores y paginaciones. Este patrón permite definir una estructura general que puede ser heredada y personalizada por clases específicas, promoviendo la reutilización de código y facilitando el mantenimiento del sistema. La Figura 2.7 muestra un extracto de este código.

La importancia del patrón *Plantilla Abstracta* en este contexto radica en su capacidad para reducir la duplicación de código y garantizar consistencia en la estructura de los datos manejados por el *backend*. Esto es crucial en un sistema que orquesta múltiples microservicios, ya que asegura que las respuestas enviadas al *frontend* sean uniformes y que los modelos de datos sean fácilmente extensibles para futuras funcionalidades.

```

13 class BaseModel(AbstractBaseModel):
14     """
15         model that represents the basic fields for all instances of the database
16     """
17     uid = models.UUIDField(primary_key=True, editable=False, default=uuid.uuid4)
18     slug = models.SlugField(unique=True, blank=True, null=True, verbose_name=_('Slug'))
19     is_active = models.BooleanField(default=True, verbose_name=_('Is Active'))
20
21     class Meta:
22         abstract = True
23
24     def soft_delete(self):
25         """Logically remove the object by changing is_active to False."""
26         self.is_active = False
27         self.save()
28

```

Figura 2.7. Extracto de código que implementa el patrón *Plantilla Abstracta* (Fuente: elaboración propia).

Patrón Fachada permite simplificar la interacción con subsistemas complejos, encapsulando la lógica de manejo de mensajes y respuestas en clases específicas. Este patrón permite abstraer operaciones complejas,

como el formateo de mensajes o la gestión de respuestas de un modelo de lenguaje, en una interfaz sencilla que puede ser utilizada por otras partes del sistema. La Figura 2.8 muestra ejemplo de la implementación de este patrón.

La relevancia del patrón *Facade/Service* radica en su capacidad para reducir la complejidad del backend, permitiendo que los controladores de Django REST Framework se centren en la lógica de negocio mientras las clases de servicio manejan las operaciones más técnicas. Esto mejora la modularidad y facilita la integración con el sistema multiagente, asegurando que las respuestas generadas por el Microservicio MAS se procesen y entreguen al frontend de manera eficiente.

```
22 class Formatted_Messages_Manager:
23     """ Class that is responsible for preparing messages to pass them through context. """
24
25     def __get_mimetype(self, image_path): ...
26
27     def __convert_path(self, image_path): ...
28
29     def __encode_image_to_base64(self, image_path): ...
30
31     def create_formated_message(self, chat:Chat): ...
32
33
34 class GPT_Response_Manager:
35     """ Class that is responsible for returning the LLM response. """
36     image_service = GenerateImageTool()
37     tools_list = [ ... ]
38
39     def __handler_image(self, image_url:str, prompt_openai:str): ...
40
41     def generate_response(self, formated_messages:list, chat:Chat): ...
```

Figura 2.8. Extracto de código que implementa el patrón *Fachada* (Fuente: elaboración propia).

El uso de estos patrones de diseño en el sistema multiagente y el backend no solo asegura una implementación robusta y escalable, sino que también facilita la transformación de datos históricos del *Diario de la Marina* en conocimiento estructurado, cumpliendo con los objetivos del proyecto.

2.4.3. Interfaz Principal del Sistema

La figura 2.9 muestra la interfaz principal del sistema multiagente, diseñado para permitir la interacción en lenguaje natural con datos históricos del *Diario de la Marina*. A continuación, se detalla cada uno de los componentes numerados de la interfaz:

- 1. Botón de Nuevo Chat:** ubicado en la parte superior izquierda, este botón permite al usuario iniciar una nueva conversación. Al activarse, se borra la conversación actual y se prepara el sistema para



Figura 2.9. Interfaz principal del chat con el sistema multiagente (Fuente: elaboración propia)

recibir una nueva consulta.

- 2. Panel de Historial de Conversaciones:** muestra una lista cronológica de conversaciones anteriores. Cada elemento incluye el título de la conversación (asignado automáticamente o manualmente) y un menú contextual para acciones como renombrar o eliminar. Esta sección permite retomar diálogos previos de forma eficiente.
- 3. Información del Usuario:** en la parte inferior del panel lateral se muestra el correo electrónico del usuario autenticado. Incluye un botón para copiar fácilmente la dirección, facilitando su reutilización o verificación de sesión.
- 4. Área de Entrada de Consulta:** permite al usuario redactar y enviar preguntas o mensajes. El campo de texto cuenta con una sugerencia que invita a interactuar: *¿Quéquieres saber hoy?*. Incluye un botón de envío que ejecuta la solicitud hacia el sistema conversacional.

2.4.4. Diagrama de despliegue

Los diagramas de despliegue muestran cómo los componentes de software se despliegan físicamente en los procesadores; es decir, el diagrama de despliegue muestra el hardware y el software en el sistema, así como el middleware³ usado para conectar los diferentes componentes en el sistema. En esencia, los

³Middleware, también conocido como lógica de intercambio de información entre aplicaciones o agente intermedio, es un sistema de software que ofrece servicios y funciones comunes para las aplicaciones.

diagramas de despliegue se pueden considerar como una forma de definir y documentar el entorno objetivo [Sommerville, 2011].

A continuación, la figura 2.10. muestra el diagrama correspondiente al sistema propuesto.

Nodos: elementos de procesamiento con al menos un procesador, memoria, y posiblemente otros dispositivos.

Dispositivos: nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.

Conectores: expresa el tipo de conector o protocolo utilizado entre el resto de los elementos del modelo.

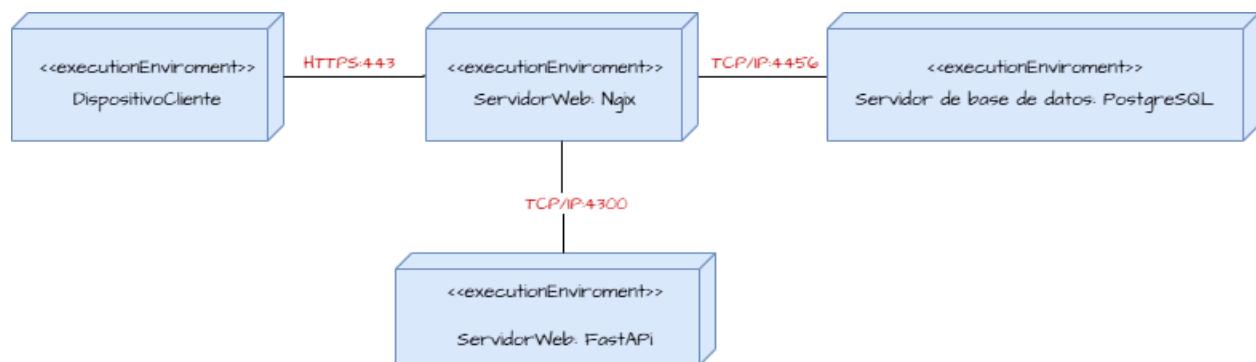


Figura 2.10. Representación del modelo de despliegue. (Fuente: elaboración propia).

Dispositivo del cliente: Se refiere a el conjunto de todos los clientes que consumirán el software desde sus computadoras. La máquina cliente necesita de muy pocas prestaciones; teniendo un navegador web (Chrome, Mozilla Firefox, Internet Explorer, Opera), una RAM mínima de 4 GB, una tarjeta de red y un procesador mínimo de 3.3 GHz podrá acceder al sistema y realizar las operaciones necesarias

Servidor Web: Representa el servidor que se comunica con la PC Cliente mediante el protocolo HTTPS y además realiza peticiones al servidor de Bases de Datos mediante el protocolo TCP/IP, es el encargado de la presentación del repositorio, debe estar compuesto de 32 GB de RAM, 1 TB de almacenamiento, un procesador de 3.3 GHz o superior, una tarjeta de red, servidor web Nginx o Trafic con Docker.

Servidor de BD: Elemento de cómputo, dedicado a almacenar y proveer datos necesarios para el funcionamiento de la aplicación web. es el encargado de almacenar la información generada del sistema, para el correcto funcionamiento del repositorio es necesario que posea PostgreSQL, una RAM mínima de 4 GB, un procesador mínimo de 3.3 GHz y un disco duro de 1 TB.

Conclusiones del capítulo

El capítulo 2 presenta un enfoque metodológico y técnico para abordar la interpretación de datos históricos no estructurados del Diario de la Marina, integrando metodologías ágiles, arquitecturas multiagente y tecnologías emergentes. La aplicación de Extreme Programming permitió adaptar el desarrollo a las necesidades identificadas en usuarios especializados, como historiadores, priorizando la corrección de errores de OCR y la contextualización de respuestas. La arquitectura propuesta, basada en microservicios y agentes especializados (Moderador, Recuperador, Contextualizador), mostró potencial para gestionar la complejidad de los datos mediante la coordinación de tareas como la recuperación semántica con FAISS y la generación de visualizaciones dinámicas.

La selección de herramientas como LangChain, modelos de embeddings y bases de datos vectoriales contribuyó a optimizar la precisión en el procesamiento de consultas, mientras que patrones de diseño como Singleton y Cadena de Responsabilidad reforzaron la mantenibilidad del sistema. La interfaz, diseñada para facilitar la interacción con usuarios no técnicos, combinó un historial de conversaciones y visualizaciones gráficas, evidenciando la viabilidad de transformar datos históricos en conocimiento accesible.

Este enfoque sugiere un avance en el campo de la informática histórica, al proponer un marco replicable que integra sistemas multiagente con IA generativa, adaptable a otros contextos de patrimonio digital. Las decisiones técnicas, respaldadas por un proceso iterativo y centrado en el usuario, reflejan una solución equilibrada entre innovación y aplicabilidad práctica, sentando bases para futuras validaciones experimentales y ampliaciones del sistema.

CAPÍTULO 3

Pruebas de software

El presente capítulo tiene como objetivo validar el correcto funcionamiento del sistema multiagente conversacional desarrollado para interpretar y contextualizar los datos no estructurados del transporte marítimo recogidos en el *Diario de la Marina*. A través de un conjunto de pruebas cuidadosamente diseñadas, se evalúa si el sistema cumple con los requisitos funcionales y no funcionales previamente definidos, así como con los criterios de calidad establecidos por el modelo ISO/IEC 25010 [[iso, 2023](#)].

Para ello, se ejecutan pruebas unitarias, de integración y funcionales sobre los distintos componentes del sistema, prestando especial atención al comportamiento del microservicio de inteligencia artificial encargado del procesamiento conversacional. Asimismo, se emplean métricas como el tiempo de respuesta, la coherencia semántica de las respuestas generadas y la capacidad de recuperación de información relevante. Además, se incluyen pruebas empíricas orientadas a medir la usabilidad desde la perspectiva del usuario final, validando si la interfaz permite una interacción fluida y si el sistema entrega respuestas precisas y contextualizadas ante consultas históricas reales.

Los resultados obtenidos en esta fase de verificación y validación permiten no solo comprobar la calidad del producto, sino también identificar oportunidades de mejora y establecer la base para una futura evolución del sistema en contextos reales de explotación académica o investigativa.

3.1. Pruebas de software

Las pruebas tratan de demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo. Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa [Sommerville, 2011]. A fin de encontrar los errores del sistema y garantizar un nivel aceptable de calidad y confianza, se realizaron pruebas de software, de caja negra, tanto manuales como estáticas, haciendo uso de las principales técnicas existentes, y aprovechando las pruebas automatizadas, apoyándose en la estrategia de pruebas recomendada por el CMS, Desarrollo dirigido por pruebas (*Test Diver Development*, por sus siglas en inglés TDD).

3.2. Estrategia de pruebas

La elaboración de todo producto de software implica la posibilidad de introducción de errores que provocan fallos en el sistema desarrollado. Por este motivo debe existir una vía para garantizar su calidad y correcto funcionamiento. La realización de pruebas es una actividad que permite verificar el producto bajo ciertas condiciones y en base a los requerimientos identificados para la construcción del mismo, los resultados son observados y registrados para su corrección.

Tabla 3.1. Plan de Pruebas

Prueba	Método	Herramienta	Alcance
Unitaria	Caja blanca con la técnica del camino básico	Módulo TestCase de Django para la realización de pruebas automatizadas	Se automatizarán pruebas para las unidades de código separadas por módulos.
Funcional	Caja negra con particiones equivalentes	SeleniumIDE	Se probará el funcionamiento del 100 % de los requisitos.

Prueba	Método	Herramienta	Alcance
Rendimiento	Pruebas de carga y estrés	Apache JMeter	Se aplicará sobre un entorno de pruebas con prestaciones similares a las de despliegue establecidas en los requisitos no funcionales. Se probará la aplicación con 20 usuarios concurrentes buscando tiempos de respuesta menores a 5 segundos.
Seguridad		Acunetix Web Vulnerability Scanner 9.5	Se aplicará para detectar vulnerabilidades: <ul style="list-style-type: none"> • Inyección SQL • Programación Cross-Site (XSS) • Ataques de fuerza bruta a las credenciales • Redirecciones y reenvíos no validados
Aceptación	Pruebas de alfa y beta		Se aplicará la prueba a los encargados del evento como miembros del comité científico y del comité organizador.

3.3. Pruebas unitarias

Se aplican a un componente del software. Podemos considerar como componente, a una función, una clase, una librería, etc. Estas pruebas las ejecuta el desarrollador, cada vez que va probando fragmentos de código o scripts para ver si todo funciona como se desea. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito, que el fragmento de código debe satisfacer. El método utilizado para realizar este tipo de prueba se denomina caja blanca [Sommerville, 2011].

Método de caja blanca

Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.

- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Para la realización de las pruebas unitarias, se le aplicó la técnica de prueba del camino básico a las unidades código que responden a funcionalidades críticas del software, lo cual permitió generar el grafo de flujo, calcular la Complejidad Ciclomática (CC) para determinar los caminos linealmente independientes y el número mínimo de escenarios de los casos de prueba para forzar la ejecución de cada camino del conjunto básico. Luego en apoyo a las pruebas se usó el módulo TestCase que ofrece el framework, django, para la automatización de las pruebas unitarias. Con él se probó cada módulo desarrollado, y gracias a la aplicación de la técnica de camino básico, en aquellas funcionalidades críticas, se pudieron automatizar pruebas para cada uno de los escenarios o caminos posibles, garantizando probar todo el código en cuestión.

Entre los elementos de código que fueron probadas se encuentra el referente al método *post* de la clase *Message_Create_AV*, que se encarga de crear los mensajes tanto del usuario como de las respuestas del sistema al usuario en una conversación asociada.

Tabla 3.2. Cálculo de la complejidad ciclomática del método post de la clase *Message_Create_AV*

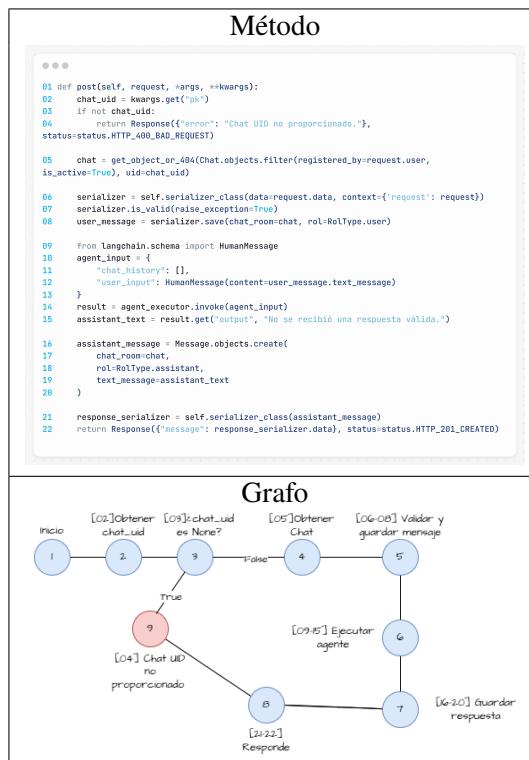


Tabla 3.3. Cálculo de la complejidad ciclomática del método post de la clase Message_Create_AV

Complejidad Ciclomática:	$V(G) = A - N + 2$	$V(G) = P + 1$
$V(G) = \# \text{ de regiones}$	$V(G) = 9 - 9 + 2$	$V(G) = 1 + 1$
$V(G) = 2$	$V(G) = 2$	$V(G) = 2$

Luego de la determinación de los nodos y flujos de control del código se obtuvo el grafo de flujo y se calculó la complejidad ciclomática del algoritmo. Como resultado se obtuvo que la CC es igual a 2, lo que significa que existen dos posibles caminos linealmente independientes y hay que diseñar un mínimo de dos casos de prueba para el algoritmo. La Tabla 3.6 muestra los caminos existentes.

Tabla 3.4. Caminos del grafo de flujo (Fuente: Elaboración propia).

No.	Camino
1	1, 2, 3, 9
2	1, 2, 3, 4, 5, 6, 7, 8

Tabla 3.5. Tabla de caminos

Los casos de prueba para las pruebas de caja blanca por la técnica de camino básico se ejecutan por cada camino independiente que se determine en un algoritmo específico. A continuación, se muestra el caso de prueba para el camino básico independiente 2 del algoritmo.

Tabla 3.6. Caso de Prueba para el camino básico 2 (Fuente: Elaboración propia).

Proceso	
Caso de prueba	Recibir consulta. Escenario 1.1
Camino independiente	1, 2, 3, 4, 5, 6, 7, 8
Entradas	<ul style="list-style-type: none"> Consulta: Lista los barcos que entraron al puerto de la Habana en 1851
Resultados esperados	<ul style="list-style-type: none"> Lista de barcos que cumplan las condiciones. En la UI mostrar el mensaje generado por el sistema multiagente.
Condiciones de ejecución	<ul style="list-style-type: none"> El usuario debe estar autenticado. Debe estar una conversación creada.

Con la realización de los casos de prueba diseñados se probó la ejecución de cada sentencia del código al menos una vez, teniendo en cuenta todas las condiciones lógicas en sus variantes verdaderas y falsas. La obtención de la CC de valor 2 del método post exemplificado, permitió determinar que existen 2 caminos linealmente independientes, suficientes para probar el código al menos una vez. Los resultados del método de caja blanca fueron satisfactorios. Se automatizaron un total de 25 casos de prueba con el uso de la biblioteca TestCase, de los cuales a 5 se le aplicó la técnica del camino básico, permitiendo que su automatización garantice probar todos los caminos con un mínimo de escenarios diseñados, y obteniendo 0 errores como se aprecia en la Figura 3.2.

```
Ran 25 tests in 132.912s
OK
Destroying test database for alias 'default'...
(env) D:\DanRo\1Workspace\proyectos-tesis\tesis-drf-admin>
```

Figura 3.1. Resultado de las pruebas unitarias.

3.4. Pruebas funcionales

Este tipo de prueba se realiza sobre el sistema funcionando, comprobando que cumpla con la especificación. Para estas pruebas, se utilizan las especificaciones de casos de prueba. Las pruebas basadas en requerimientos son pruebas de validación más que de defecto: se intenta demostrar que el sistema implementó adecuadamente sus requerimientos [Sommerville, 2011].

3.4.1. Método de caja negra

Las pruebas de caja negra, también llamadas pruebas de comportamiento, se enfocan en los requerimientos funcionales del software. Las técnicas de prueba de caja negra permiten derivar conjuntos de condiciones de entrada que revisarán los requerimientos funcionales para un programa [Pressman et al., 2010]. El método de caja negra presenta varias técnicas de prueba como son: partición de equivalencia, análisis de valores límites y grafos de causaefecto. En la presente investigación se utilizará específicamente dentro del método

de caja negra la técnica de partición de equivalencia generando los casos de pruebas de dicha técnica sobre las diferentes interfaces que responden a los requisitos funcionales. Para la aplicación de pruebas de regresión sobre los casos de prueba definidos se usará la herramienta Selenium IDE, que permite grabar todas las interacciones de un usuario con el navegador y posibilita ejecutar de forma automática las mismas, reduciendo el tiempo y los costos de las pruebas funcionales.

Index	Command	Target	Value
6	✓ mouse over	css=.text-yellow-900 > path	
7	✓ mouse out	css=.text-yellow-900 > path	
8	✓ mouse over	css=.text-yellow-900 > path	

Figura 3.2. Representación del resultado la ejecución de una prueba usando Selenium IDE, del requisito Insertar consulta.

A continuación, la Tabla 3.7 muestra el diseño de caso de pruebas del requisito “Insertar consulta” donde se analizarán las variables y condiciones que puedan determinar la respuesta del sistema.

Tabla 3.7. Caso de prueba para la funcionalidad Insertar Consulta (Fuente: Elaboración Propia).

Escenario	Descripción	Variables (Mensaje)	Respuesta Esperada	Respuesta
EC 1.1. Enviar consulta correctamente	El usuario debe escribir la consulta y dar clic en el botón de enviar	¿Cuantos barcos entraron al puerto de La Habana en 1851?	567	530
EC 1.2. Enviar consulta incorrecta(con un contexto fuera de la información conocida por el sistema)	El usuario debe escribir la consulta y dar clic en el botón de enviar	¿Cuánto tiempo duró la 2da guerra mundial?	Lo lamento no tengo esa información disponible	La información solicitada no se encuentra. Por favor reformule la consulta.
EC 1.3. Solicitar un consulta cuya respuesta sea una gráfica	El usuario debe escribir la consulta y dar clic en el botón de enviar	Genera una gráfica con el % de los barcos que entraron a La Habana con arroz en diferentes años.	La respuesta debe ser una gráfica	Responde con una gráfica correspondiente acorde a la consulta
EC 1.4. Enviar consulta vacía	El usuario debe dar clic en el botón de enviar	-	No se efectúa el envío de consultas vacías	No se envía la consulta vacía

No.	Variable	Valor Nulo	Descripción
1	Consulta	No	Es un campo de texto que permite al usuario escribir una consulta al sistema

Tabla 3.8. Variables de caso de prueba “Insertar Consulta” (Fuente: Elaboración Propia).

Las pruebas de caja negra se aplicaron con el objetivo de evaluar las interfaces de comunicación con el usuario, las que demostraron coherencia y funcionalidad, así como probar todas aquellas funcionalidades directamente relacionadas con los requisitos funcionales del sistema. La técnica de partición de equivalencia es aplicada para evaluar los diferentes escenarios que pueden tener lugar ante la ejecución de una acción.

Como resultado de la aplicación de estas pruebas se ejecutan las posibles variantes que posee una interfaz de comunicación con el usuario, resolviendo las no conformidades arrojadas y perfeccionando lo obtenido. Durante la realización de las pruebas se detectaron un conjunto de no conformidades relacionadas con errores de validación y funcionalidad. Los resultados se muestran en la Figura 3.3, donde se evidencia la cantidad de casos de prueba ejecutados, los casos de prueba con no conformidades. Se realizaron tres iteraciones, durante la primera iteración se analizaron 25 casos de prueba, de los cuales 15 resultaron no conformidades. En la segunda y tercera iteración a través de las pruebas de regresión, con el uso del software Selenium IDE, se verificó que las no conformidades anteriores estuviesen solucionadas, y de estas pruebas se obtuvieron 6 nuevas no conformidades en la segunda iteración, quedando resuelta en la tercera iteración y cumpliéndose correctamente los requisitos funcionales.

PRUEBAS FUNCIONALES

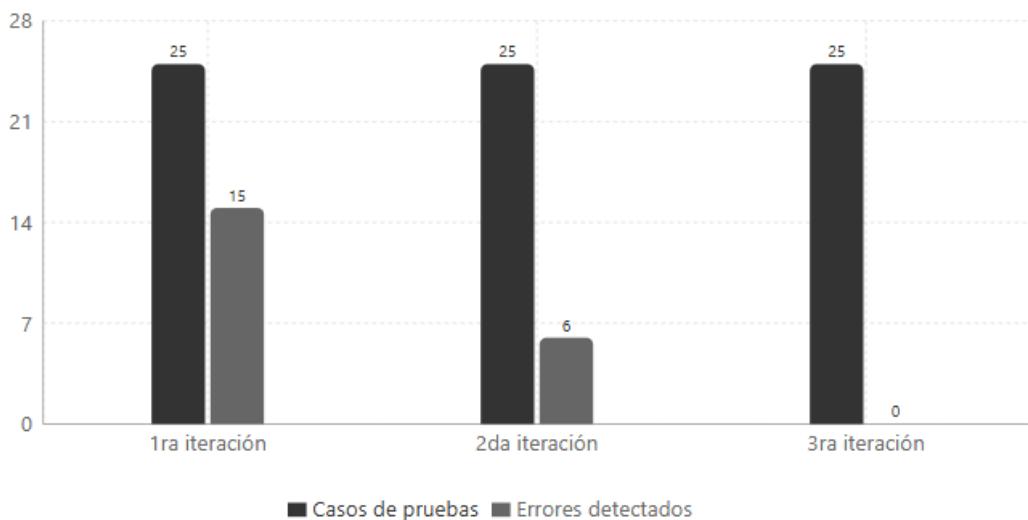


Figura 3.3. Representación del resultado de las pruebas funcionales (Fuente: Elaboración Propia).

3.5. Prueba de seguridad

Las pruebas de seguridad se diseñan para sondear las vulnerabilidades del entorno del lado del cliente, las comunicaciones de red que ocurren conforme los datos pasan de cliente a servidor y viceversa, y el entorno del lado servidor. Cada uno de estos dominios puede atacarse, y es tarea del examinador de seguridad

descubrir las debilidades que puedan explotar quienes tengan intención de hacerlo [Pressman et al., 2010]. Las pruebas de seguridad se aplicaron con ayuda de la herramienta Acunetix Web Vulnerability Scanner 9.5 que establece alertas de tipo: alta, media, baja e informativo, realizándose en dos iteraciones durante el desarrollo de la propuesta solución. En una primera iteración se obtuvo un total de 22 alertas de seguridad, de las cuales 4 clasifican de nivel medio, 1 de nivel bajo y 17 informativas.

De las de nivel medio, se destacaron el uso de protocolo no seguro para el envío de datos, así como los mensajes de error que se muestra en el modo DEBUG de Django para el desarrollo y se detectaron problemas para la protección de contra ataques de fuerza bruta en el formulario de autenticación.

La de nivel bajo, consistía en vistas del sitio que se podían acceder directamente sin pasar la autenticación y el sistema de roles establecido. De carácter informativo fueron detectadas posibles cuentas de usuario en ficheros, así como presencia de directorios desprotegidos y la existencia de etiquetas iframe de HTML5.

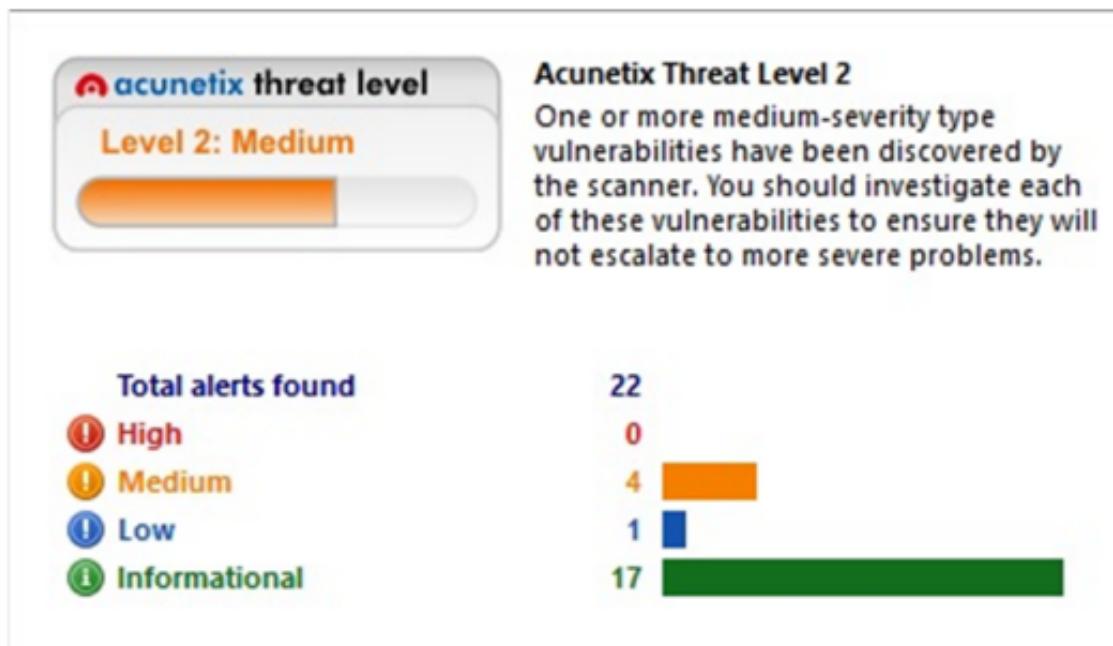


Figura 3.4. Prueba de seguridad 1ra iteración.

Después de aplicar refactorización del código y realizar las validaciones correspondientes, se aplicó la segunda iteración en búsqueda de vulnerabilidades al sistema, arrojando como resultado que todas las que se habían detectado en la primera iteración, habían sido corregidas.

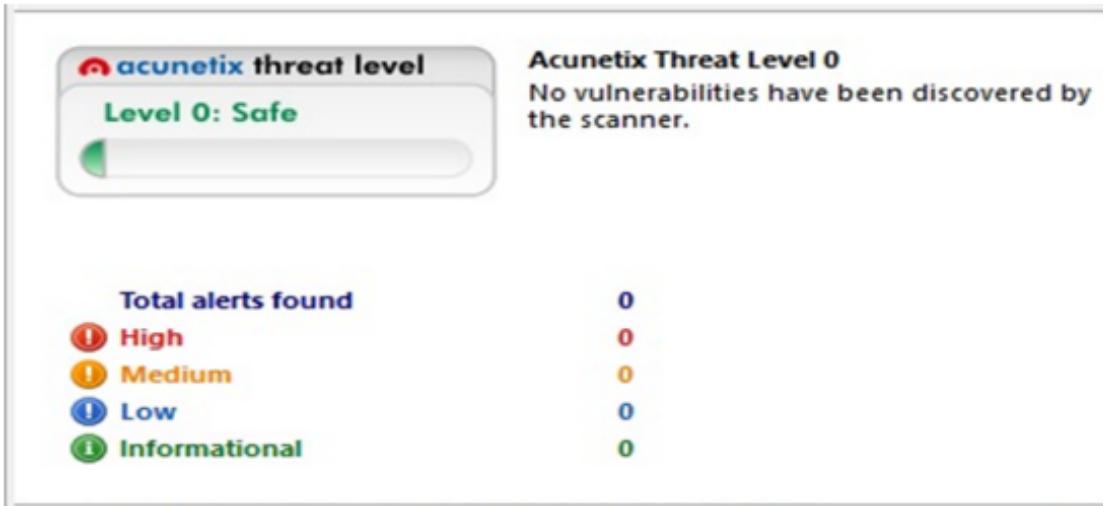


Figura 3.5. Prueba de seguridad 2da iteración.

3.6. Prueba de rendimiento

Las pruebas de rendimiento deben diseñarse para garantizar que el sistema procese su carga pretendida. Esto implica efectuar una serie de pruebas donde se aumenta la carga, hasta que el rendimiento del sistema se vuelve inaceptable. Las pruebas de rendimiento se preocupan tanto por demostrar que el sistema cumple con sus requerimientos, como por descubrir problemas y defectos en el sistema [Sommerville, 2011].

Conclusiones del capítulo

Conclusiones generales

Como resultados de la investigación y dando cumplimiento al objetivo general, se arribó a las siguientes conclusiones:

Recomendaciones

Bibliografía

[ISO, 2016] (2016). Information technology – database languages – sql.

[git, 2021] (2021). Implementing version control with git and github as a learning tool.

[Air, 2022] (2022). Airbnb javascript style guide. <https://github.com/airbnb/javascript>.

[Goo, 2022] (2022). Google javascript style guide. <https://google.github.io/styleguide/jsguide.html>.

[iso, 2023] (2023). ISO/IEC 25010:2023 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. Standard ISO/IEC 25010:2023, International Organization for Standardization, Geneva, Switzerland.

[git, 2024] (2024). What is version control?

[Adobe Inc., 2024] Adobe Inc. (2024). Adobe acrobat ai assistant. Consultado en abril de 2025.

[Alliance, 2017] Alliance, A. (2017). What is extreme programming (xp)?

[Ambler, 2003] Ambler, S. W. (2003). *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. Wiley.

[Azizi et al., 2024] Azizi, I., Echihabi, K., and Palpanas, T. (2024). Vector search on billion-scale data collections. *Proceedings of the VLDB PhD Workshop*.

[Beck and Cunningham, 1989] Beck, K. and Cunningham, W. (1989). A laboratory for teaching object-

oriented thinking. Disponible en: <https://c2.com/doc/oopsla89/paper.html>.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

[Boehm-Turner, 2003] Boehm-Turner (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, Boston, MA.

[Chatize Team, 2023] Chatize Team (2023). Chatize website.

[Cherny, 2019] Cherny, B. (2019). *Programming TypeScript: Making Your JavaScript Applications Scale*. O'Reilly Media.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

[Connolly and Begg, 2014] Connolly, T. and Begg, C. (2014). *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson, 6th edition.

[Date, 2003] Date, C. J. (2003). *An Introduction to Database Systems*. Pearson, 8th edition.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Elmasri and Navathe, 2010] Elmasri, R. and Navathe, S. B. (2010). *Fundamentals of Database Systems*. Addison-Wesley, 6th edition.

[Ferro et al., 2023] Ferro, S., Pelillo, M., and Traviglia, A. (2023). Ai-assisted digitalisation of historical documents. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-M-2-2023:557–563.

[Fowler, 2014a] Fowler, M. (2014a). Microservice testing strategies.

[Fowler, 2014b] Fowler, M. (2014b). Microservice trade-offs.

[Fowler, 2014c] Fowler, M. (2014c). Microservices.

[Gao, 2024] Gao, Y. (2024). Retrieval-augmented generation for large language models: A survey. *arXiv*

preprint arXiv:2312.10997.

[Gavilánez Álvarez et al., 2022] Gavilánez Álvarez, O. D., Layedra Larrea, N. P., and Ramos Valencia, M. V. (2022). Análisis comparativo de patrones de diseño de software. *Polo del Conocimiento: Revista científico - profesional*, 7(7):2146–2165.

[Goodger, 2001] Goodger, D. (2001). Pep 257 – docstring conventions. <https://peps.python.org/pep-0257/>.

[GraphApp.ai, 2025] GraphApp.ai (2025). Martin fowler’s insights on microservices: A comprehensive guide.

[Guo et al., 2024] Guo, T. et al. (2024). Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

[Han and Liu, 2023] Han, Y. and Liu, C. (2023). A comprehensive survey on vector database. *arXiv preprint arXiv:2310.11703*.

[Harrington, 2015] Harrington, J. L. (2015). *Relational Database Design and Implementation: Clearly Explained*. Morgan Kaufmann, 4th edition.

[Haystack Team, 2023] Haystack Team (2023). Haystack documentation.

[IBM Documentation Contributors, 2025] IBM Documentation Contributors (2025). Referential integrity. Accessed: 2025-05-07.

[LangChain, 2023] LangChain (2023). Langgraph.

[Lewis et al., 2020a] Lewis, P. et al. (2020a). Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*.

[Lewis et al., 2020b] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., and Riedel, S. (2020b). Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint*.

[LlamaIndex Team, 2023] LlamaIndex Team (2023). Llamaindex website.

- [Ltd, 2025] Ltd, J. (2025). draw.io - herramienta de diagramación en línea. Accedido: 6 de abril de 2025.
- [Margot Note, 2018] Margot Note (2018). Metadata for archival collections: Challenges and opportunities.
- [Menzinsky et al., 2018] Menzinsky, A., López, G., Palacio, J., Sobrino, M. Á., Álvarez, R., and Rivas, V. (2018). Historias de usuario. *Ingeniería de requisitos ágil*.
- [Microsoft, 2021] Microsoft (2021). Why a microservices architecture?
- [Microsoft, 2022] Microsoft (2022). What are microservices?
- [Microsoft, 2024] Microsoft (2024). Microservices assessment and readiness.
- [Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.
- [MyMap.AI, 2024] MyMap.AI (2024). Mymap.ai chatpdf. Consultado en abril de 2025.
- [Newman and Fowler, 2021] Newman, S. and Fowler, M. (2021). When to use microservices: Sam newman and martin fowler share their knowledge.
- [Nguyen et al., 2025] Nguyen, H. D., Nguyen, T.-H. A., and Nguyen, T. B. (2025). A proposed large language model-based smart search for archive system. arXiv preprint.
- [Palli, 2023] Palli, A. (2023). La importancia del análisis de requerimientos en el desarrollo de software. *PROEFEX*.
- [Peters, 2004] Peters, T. (2004). Pep 20 – the zen of python. <https://peps.python.org/pep-0020/>.
- [Piryani et al., 2025] Piryani, B., Mozafari, J., Abdallah, A., Doucet, A., and Jatowt, A. (2025). Multiocr-qa: Dataset for evaluating robustness of llms in question answering on multilingual ocr texts. *arXiv preprint arXiv:2502.16781*.
- [Pressman et al., 2010] Pressman, R. S. et al. (2010). A practitioner's approach. *Software Engineering*, 2:41–42.
- [Python, 2023] Python, R. (2023). Natural language processing with spacy in python.

- [Pérez-Pons et al., 2021] Pérez-Pons, M. E., Parra, J., Corchado, J. M., Durán, R., Queiroz, J., and Leitão, P. (2021). A brief review on multi-agent system approaches. ResearchGate.
- [Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.
- [Richardson, 2025] Richardson, C. (2025). Microservice patterns.
- [Silberschatz et al., 2019] Silberschatz, A., Korth, H. F., and Sudarshan, S. (2019). *Database System Concepts*. McGraw-Hill, 7th edition.
- [Sommerville, 2011] Sommerville, I. (2011). Software engineering (ed.). *America: Pearson Education Inc.*
- [Stonebraker and Hellerstein, 2005] Stonebraker, M. and Hellerstein, J. M. (2005). What goes around comes around. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR)*.
- [Sun and Guo, 2024] Sun, P. and Guo, R. (2024). Soar: New algorithms for even faster vector search with scann. Google Research Blog.
- [Team, 2023a] Team, L. (2023a). Langchain documentation. Accedido: 2023-04-05.
- [Team, 2023b] Team, L. (2023b). Langraph documentation. Accedido: 2023-04-05.
- [TextCortex, 2024] TextCortex (2024). Textcortex chatpdf. Consultado en abril de 2025.
- [van Rossum et al., 2001] van Rossum, G., Warsaw, B., and Coghlan, N. (2001). Pep 8 – style guide for python code. <https://peps.python.org/pep-0008/>.
- [Vidnoz, 2024] Vidnoz (2024). Vidnoz chatpdf. Consultado en abril de 2025.
- [Wikipedia, 2023] Wikipedia (2023). Diario de la marina.
- [Wikipedia, 2025] Wikipedia (2025). Microservices.
- [Xie et al., 2023] Xie, X., Liu, H., Hou, W., and Huang, H. (2023). A brief survey of vector databases. In *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*, pages 364–371.
- [Yogish Naik G R, 2023] Yogish Naik G R (2023). Enhancing degraded historical document images. Ac-

ceso: 5 de abril de 2025.

[Zambrano et al., 2020] Zambrano, J. et al. (2020). Multi-agent systems for the management of resources and activities in smart classrooms. *IEEE Latin America Transactions*, 18(5):1511–1518.

Anexos

ANEXO A

Historias de Usuario

Tabla A.1. Historia de usuario # 4

Historia de usuario	
Número: 4	Nombre: Registrar una nueva cuenta de usuario
Usuario: Visitante del sitio web	
Prioridad en negocio: Alta	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 1
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla A.1. Continuación de la página anterior

<p>Descripción: Como visitante del sitio web, quiero poder registrar una nueva cuenta proporcionando mi dirección de correo electrónico, nombre de usuario y una contraseña segura, para crear una cuenta personal que me permita acceder a las funcionalidades de consulta y visualización del archivo histórico y guardar mi historial. (Corresponde principalmente a RF1)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El visitante no tiene una sesión activa. • El visitante se encuentra en la página o sección de registro del sitio web. • El <i>backend</i> está operativo y accesible. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Visitante ingresa dirección de correo electrónico, nombre de usuario, contraseña y confirmación de contraseña en el formulario de la sección registrar cuenta. 2. Visitante envía el formulario. 3. El sitio web valida formato básico (e.g., dirección de correo electrónico válido, contraseñas coinciden, nombre de usuario único). 4. El sitio web envía los datos al endpoint de registro del <i>backend</i>. 5. El <i>backend</i> valida los datos (e.g., dirección de correo electrónico no existente, nombre de usuario no existente, complejidad de contraseña) y crea el usuario en la base de datos de forma segura. 6. El <i>backend</i> retorna una respuesta de éxito o error al frontend. 7. El <i>frontend</i> muestra un mensaje apropiado al usuario (éxito o error específico). <p>Observaciones: Implementar validación de fortaleza de contraseña en <i>frontend</i> y <i>backend</i>. Asegurar almacenamiento seguro de contraseñas (hashing). Los mensajes de error deben ser claros (e.g., 'El correo electrónico ya está registrado', 'Las contraseñas no coinciden').</p>

Continúa en la próxima página

Tabla A.1. Continuación de la página anterior

Interfaz:	<p>Formulario de registro del sitio web:</p> 
------------------	--

Tabla A.2. Historia de usuario # 5

Historia de usuario	
Número: 5	Nombre: Iniciar sesión en el sistema
Usuario: Usuario registrado	
Prioridad en negocio: Alta	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 1
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla A.2. Continuación de la página anterior

<p>Descripción: Como usuario registrado, quiero poder iniciar sesión utilizando mi correo electrónico y contraseña previamente registrados, para acceder a mi perfil, mi historial de conversaciones y utilizar las capacidades completas del sistema de chat. (Corresponde principalmente a RF2, RF4)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una cuenta previamente registrada en el sistema. • El usuario no tiene una sesión activa. • El usuario se encuentra en la página o sección de inicio de sesión del sitio web. • El <i>backend</i> está operativo y accesible. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario ingresa su dirección de correo electrónico y contraseña en el formulario del sitio web. 2. Usuario envía el formulario. 3. El sitio web envía las credenciales al <i>endpoint</i> de autenticación del <i>backend</i>. 4. El <i>backend</i> verifica las credenciales contra la base de datos. 5. Si las credenciales son válidas, el <i>backend</i> genera un token de sesión (e.g., JWT) y lo retorna al <i>frontend</i>. 6. Si las credenciales son inválidas, el <i>backend</i> retorna un error de autenticación. 7. El <i>frontend</i> almacena el token de sesión de forma segura (e.g., localStorage, sessionStorage o cookie HttpOnly). 8. Si el inicio de sesión es exitoso, el <i>frontend</i> redirige al usuario a la interfaz principal del chat. Si falla, muestra un mensaje de error ("Credenciales incorrectas"). <p>Observaciones: Utilizar HTTPS para la comunicación. Implementar medidas contra ataques de fuerza bruta (e.g., límites de intentos). El manejo del token en el <i>frontend</i> debe seguir las mejores prácticas de seguridad.</p>
--

Continúa en la próxima página

Tabla A.2. Continuación de la página anterior

Interfaz:	<p>Formulario de inicio de sesión del sitio web:</p> <p style="text-align: center;">Bienvenido de nuevo</p> <p style="text-align: center;">Ingresa tus credenciales para acceder a tu cuenta</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content; margin: auto;"> <p style="text-align: center;">Iniciar Sesión</p> <p style="text-align: center;">Ingresa tu correo y contraseña para iniciar sesión</p> <p>Correo Electrónico</p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content; margin: auto; text-align: center;">nombre@ejemplo.com</div> <p>Contraseña</p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content; margin: auto; text-align: center;">.....</div> <div style="background-color: black; color: white; padding: 5px; text-align: center; margin-top: 10px;">Iniciar Sesión</div> <p style="text-align: center; margin-top: 10px;">¿No tienes una cuenta? Registrarse</p> </div>
------------------	--

Tabla A.3. Historia de usuario # 6

Historia de usuario	
Número: 6	Nombre: Cerrar sesión del sistema
Usuario: Usuario autenticado	
Prioridad en negocio: Media	Riesgo en desarrollo: Bajo
Puntos estimados: 0.5 semanas	Iteración asignada: 1
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla A.3. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero disponer de una opción clara para cerrar mi sesión activa, para asegurar la privacidad de mi cuenta y finalizar mi interacción con el sistema de forma segura. (Corresponde principalmente a RF3)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa (posee un token válido). • El usuario está en la interfaz principal del sistema. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en el botón o enlace 'Cerrar Sesión'. 2. El <i>frontend</i> elimina el token de sesión almacenado localmente. 3. (Recomendado) El <i>frontend</i> envía una solicitud al <i>backend</i> para invalidar el token en el servidor (si se usa una blacklist de tokens). 4. El <i>frontend</i> redirige al usuario a la página de inicio de sesión o a una página pública principal. 5. Cualquier intento posterior de acceder a rutas protegidas con el token antiguo debe fallar. <p>Observaciones: El botón de cerrar sesión debe ser fácilmente accesible en la interfaz de usuario autenticado.</p> <p>Interfaz:</p> <p>Botón cerrar sesión en el sitio web:</p>

Tabla A.4. Historia de usuario # 7

Historia de usuario	
Número: 7	Nombre: Iniciar una nueva conversación
Usuario: Usuario autenticado	
Prioridad en negocio: Alta	Riesgo en desarrollo: Bajo
Puntos estimados: 0.5 semanas	Iteración asignada: 2
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla A.4. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero poder iniciar una nueva conversación de chat en cualquier momento, para realizar consultas sobre temas distintos sin mezclar las interacciones o para empezar de cero si lo necesito. (Corresponde principalmente a RF5)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. • El usuario se encuentra en la interfaz principal del chat. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en la opción "Nueva Conversación"(o ícono '+'). 2. El <i>frontend</i> limpia el área de visualización del chat actual. 3. El <i>frontend</i> establece un estado interno que indica que la próxima consulta pertenece a una nueva conversación (puede no requerir llamada inmediata al <i>backend</i>). 4. Opcionalmente, el <i>frontend</i> puede asignar un ID temporal a la nueva conversación hasta que se envíe el primer mensaje. 5. La interfaz se muestra lista para recibir la primera consulta de la nueva conversación. <p>Observaciones: La acción debe ser visualmente clara y distingible de seleccionar una conversación existente. El estado de "nueva conversación" debe manejarse correctamente hasta el primer envío.</p> <p>Interfaz:</p> <p>Botón nueva conversación en el sitio web:</p> 
--

Tabla A.5. Historia de usuario # 8

Historia de usuario	
Número: 8	Nombre: Ver historial de conversaciones
Usuario: Usuario autenticado	
Prioridad en negocio: Media	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 2
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla A.5. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero ver una lista organizada de mis conversaciones anteriores (por ejemplo, con un título autogenerado o fecha), para poder identificar y acceder fácilmente a interacciones pasadas. (Corresponde principalmente a RF6)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. • El <i>backend</i> y la base de datos que almacena el historial están operativos. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. El <i>frontend</i> (al cargar la interfaz principal o al interactuar con un panel de historial) solicita la lista de conversaciones del usuario al <i>backend</i>. 2. El <i>backend</i> consulta la base de datos para obtener los metadatos de las conversaciones asociadas al usuario autenticado (ID, título/fecha, última actualización). 3. El <i>backend</i> retorna la lista de conversaciones al <i>frontend</i>. 4. El <i>frontend</i> muestra la lista en un panel lateral o sección designada, permitiendo al usuario ver los identificadores de cada conversación. <p>Observaciones: Considerar paginación si el historial puede ser muy largo. La generación de títulos/identificadores debe ser útil (e.g., basado en la primera consulta). La lista debe actualizarse si se crea o elimina una conversación.</p>
--

Continúa en la próxima página

Tabla A.5. Continuación de la página anterior

Interfaz:	Panel/Sección de Historial en el sitio web:
	<p>Nueva Conversación Sin mensajes aún Ahora mismo</p>
	<p>Cómo mejorar la productividad Prueba la técnica Pomodoro... hace 2 horas</p>
	<p>Recomendaciones de libros Te recomendaría 'Hábitos Atómicos'... Ayer</p>
	<p>Consejos para planificar viajes Para viajes económicos, considera... hace 3 días</p>

Tabla A.6. Historia de usuario # 9

Historia de usuario	
Número: 9	Nombre: Abrir una conversación del historial
Usuario: Usuario autenticado	
Prioridad en negocio: Media	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 2
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla A.6. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero poder seleccionar una conversación específica de mi historial listado, para cargar su contenido completo (consultas y respuestas) en la interfaz principal del chat y, opcionalmente, continuarla. (Corresponde principalmente a RF7)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. • El usuario está viendo la lista de su historial de conversaciones (HU:05). • El <i>backend</i> y la base de datos que almacena el historial están operativos. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en una conversación específica en la lista del historial. 2. El <i>frontend</i> envía una solicitud al <i>backend</i> pidiendo el contenido completo de la conversación seleccionada (pasando su ID). 3. El <i>backend</i> recupera todas las consultas y respuestas asociadas a esa conversación para ese usuario. 4. El <i>backend</i> retorna el historial completo de mensajes de esa conversación al <i>frontend</i>. 5. El <i>backend</i> limpia el área de chat actual y muestra los mensajes recuperados en el orden correcto. 6. El <i>backend</i> establece la conversación seleccionada como la conversación activa.^aactual. 7. La interfaz permite al usuario añadir nuevas consultas a esta conversación activa.
<p>Observaciones: La carga debe ser eficiente, especialmente para conversaciones largas. La interfaz debe indicar claramente cuál conversación del historial está activa.</p> <p>Interfaz:</p> <p>Interacción con lista de historial y chat principal en el sitio web:</p>  <p>El prototipo muestra una lista lateral de conversaciones y un área central para el chat. Una conversación específica es seleccionada y mostrada en el centro. Un diálogo emergente informa: "Estoy procesando tu solicitud. Esta es una respuesta simulada para el prototipo de UX." El área de chat muestra un mensaje entrante de "A hola" y un campo para escribir un mensaje.</p>

Tabla A.7. Historia de usuario # 10

Historia de usuario	
Número: 10	Nombre: Eliminar una conversación del historial

Continúa en la próxima página

Tabla A.7. Continuación de la página anterior

Usuario: Usuario autenticado	
Prioridad en negocio: Baja	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 3
Programador responsable: Daniel Rojas Grass	
<p>Descripción: Como usuario autenticado, quiero tener la opción de eliminar permanentemente una conversación específica de mi historial, para mantener mi historial limpio y relevante. (Corresponde principalmente a RF8)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. • El usuario está viendo la lista de su historial o tiene una conversación cargada que desea eliminar. • El <i>backend</i> y la base de datos que almacena el historial están operativos. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en la opción <i>.Eliminar.</i> asociada a una conversación en la lista del historial (o en la conversación activa). 2. El <i>frontend</i> muestra un diálogo de confirmación ("¿Estás seguro de que quieres eliminar esta conversación? Esta acción no se puede deshacer."). 3. Si el usuario confirma la eliminación: <ol style="list-style-type: none"> a) El frontend envía una solicitud al backend DRF para eliminar la conversación (pasando su ID). b) El backend verifica que la conversación pertenece al usuario y la elimina de la base de datos. c) El backend retorna una respuesta de éxito al frontend. d) El frontend elimina la conversación de la lista visible en el historial. e) Si la conversación eliminada era la activa, el frontend limpia el área de chat o carga una conversación por defecto/nueva. 4. Si el usuario cancela, no se realiza ninguna acción. <p>Observaciones: La confirmación es crucial para prevenir eliminaciones accidentales. La eliminación debe ser lógicamente completa en el backend (borrado permanente).</p>	

Continúa en la próxima página

Tabla A.7. Continuación de la página anterior

Interfaz:

Opción de Eliminar en la lista de Historial o Chat activo:



ANEXO B

Targetas CRC

Tabla B.1. Tarjeta CRC: Usuario

Tarjeta CRC	
Clase	Usuario
Responsabilidades: Proporcionar datos para registro (email, username, contraseña) Almacenar credenciales de forma segura Mantener información de sesión activa Asociar conversaciones al usuario	Colaboración: Autenticador BaseDeDatos Conversación

Tabla B.2. Tarjeta CRC: Autenticador

Tarjeta CRC	
Clase	Autenticador
Responsabilidades: Validar datos de registro (email único, contraseña fuerte) Autenticar credenciales de inicio de sesión Generar y gestionar tokens de sesión Finalizar sesiones activas	Colaboración: Usuario BaseDeDatos Backend

Tabla B.3. Tarjeta CRC: Conversación

Tarjeta CRC	
Clase	Conversación
Responsabilidades:	Colaboración:
Iniciar una nueva sesión de chat	Usuario
Almacenar consultas y respuestas	Historial
Permitir continuación de una conversación existente	Backend
<u>Eliminar una conversación del historial</u>	BaseDeDatos

Tabla B.4. Tarjeta CRC: Historial

Tarjeta CRC	
Clase	Historial
Responsabilidades:	Colaboración:
Listar todas las conversaciones de un usuario	Usuario
Proporcionar metadatos de conversaciones (título, fecha)	Conversación
Permitir selección de una conversación específica	Backend
	BaseDeDatos

Tabla B.5. Tarjeta CRC: Chat

Tarjeta CRC	
Clase	Chat
Responsabilidades:	Colaboración:
Mostrar la interfaz de chat activo	Conversación
Permitir ingreso de consultas en lenguaje natural	Backend
Visualizar consultas y respuestas (texto e imágenes)	MicroservicioMAS
Indicar estado de procesamiento	

Tabla B.6. Tarjeta CRC: Backend

Tarjeta CRC	
Clase	Backend

Responsabilidades:	Colaboración:
Gestionar endpoints REST para autenticación y chat	Usuario
Coordinar comunicación entre frontend y MicroservicioMAS	Autenticador
Almacenar y recuperar datos de conversaciones	Conversación
Proteger rutas con autenticación	Historial
	Chat
	MicroservicioMAS
	BaseDeDatos

Tabla B.7. Tarjeta CRC: BaseDeDatos

Tarjeta CRC	
Clase	BaseDeDatos
Responsabilidades: Almacenar datos de usuarios (credenciales, sesiones) Persistir conversaciones y su historial Proveer acceso a datos del <i>Diario de la Marina</i> (vectorial y CSV)	Colaboración: Usuario Autenticador Conversación Historial Backend Agente Recuperador (FAISS) Agente PandasAi

Tabla B.8. Tarjeta CRC: MicroservicioMAS

Tarjeta CRC	
Clase	MicroservicioMAS
Responsabilidades: Recibir consultas del backend Coordinar agentes internos para procesar consultas Devolver respuestas procesadas (texto e imágenes) al backend	Colaboración: Backend Agente Moderador Agente Recuperador (FAISS) Agente Contextualizador Agente de Validación Agente PandasAi