



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 Grabiél 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 Grabiél Toledano López

MsC. Olga Yarisbel Rojas Grass

Datos de Contacto

Curriculum e información de contacto del tutor: nombre y apellidos, títulos académicos, formación de postgrado recibida, lugar de trabajo, responsabilidades laborales asumidas, experiencia profesional, líneas de trabajo y/o investigación, correo electrónico, perfiles en redes profesionales

Curriculum e información de contacto del asesor: nombre y apellidos, títulos académicos, formación de postgrado recibida, lugar de trabajo, responsabilidades laborales asumidas, experiencia profesional, líneas de trabajo y/o investigación, correo electrónico, perfiles en redes profesionales

Curriculum e información de contacto del consultante: nombre y apellidos, títulos académicos, formación de postgrado recibida, lugar de trabajo, responsabilidades laborales asumidas, experiencia profesional, líneas de trabajo y/o investigación, correo electrónico, perfiles en redes profesionales

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

Declaración de Autoría	I
Datos de Contacto	II
Dedicatoria	III
Resumen	IV
Abstract	v
Índice general	VI
Introducción	1
1 Fundamentos Teóricos y Contextuales de la Transformación de Datos Históricos y Sistemas Multiagentes Conversacionales	6
Introducción	6
1.1 Fundamentación Teórica	7
1.1.1 Reconocimiento Óptico de Caracteres (OCR)	7
1.1.2 Agente LLM	8
1.1.3 Sistema Multiagente (SMA)	8
1.1.4 Sistemas RAG	9
1.1.5 Modelos de <i>embeddings</i>	9
1.1.6 Bases de datos vectoriales	10

1.2	Análisis del estado del arte	10
1.3	Metodología de desarrollo de software	14
1.4	Herramientas y tecnologías	14
1.4.1	Herramienta de Modelado	15
1.4.2	Lenguaje de programación	15
1.4.3	Framework de desarrollo	16
1.4.4	Control de versiones	17
1.4.5	Vectorización y Almacenamiento	17
	Conclusiones del Capítulo	18
2	Propuesta de solución	20
	Introducción	20
2.1	Descripción de la Propuesta de Solución	21
2.2	Análisis de requisitos	23
2.2.1	Técnicas de captura de requisitos	23
2.2.2	Requisitos funcionales	24
2.2.3	Requisitos no funcionales	29
2.2.4	Historias de usuarios	32
2.3	Patrones de diseño	46
2.3.1	Patrones Arquitectónicos	47
2.3.2	Patrones de Comportamiento	47
2.3.3	Patrones Estructurales	48
2.3.4	Patrones de Concurrencia	48
2.4	Targetas CRC	49
2.5	Diagrama de componentes	51
3	Validación y Análisis de Resultados	52
	Conclusiones generales	53
	Recomendaciones	54
	Anexos	60

Índice de figuras

1.1	Diario de la Marina 1884.	7
2.1	Arquitectura de la propuesta de solución.	21
2.2	Flujo Interno del Microservicio Multiagente Conversacional.	22
2.3	Diagrama de Componentes.	51

Índice de tablas

1.1	Análisis de Soluciones Conversacionales	12
2.1	Tabla de Requisitos Funcionales (RF)	25
2.2	Tabla de Requisitos No Funcionales (RNF)	29
2.3	Historia de usuario # 1	33
2.4	Historia de usuario # 2	34
2.5	Historia de usuario # 3	36
2.6	Historia de usuario # 4	37
2.7	Historia de usuario # 5	38
2.8	Historia de usuario # 6	40
2.9	Historia de usuario # 7	42
2.10	Historia de usuario # 8	43
2.11	Historia de usuario # 9	44
2.12	Historia de usuario # 10	45
2.13	Tarjeta CRC: ModeratorAgent	49
2.14	Tarjeta CRC: InformationRetrievalAgent	50
2.15	Tarjeta CRC: ContextualizationAgent	50
2.16	Tarjeta CRC: ValidationAgent	50
2.17	Tarjeta CRC: PythonAgent	51

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. Un ejemplo emblemático es el *Diario de la Marina* (Cuba, 1844–1960), un testimonio clave de la historia política y cultural del Caribe [1]. 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 [2].

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 [2]. 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.

Frente a esta problemática, la inteligencia artificial emerge como una solución prometedora. En particular, los modelos de lenguaje de gran escala (LLMs) ofrecen nuevas posibilidades para la interpretación y generación de texto con alto grado de sofisticación [3]. Sin embargo, su aplicación aislada no es suficiente para abordar tareas complejas como categorización, enriquecimiento semántico y búsqueda contextual [3].

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 [1].

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 [4]. 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 [5, 4].

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. Fundamentos Teóricos y Contextuales de la Transformación de Datos Históricos y Sistemas Multiagentes Conversacionales:** 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. 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. Validación y Análisis de Resultados:** 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.

Fundamentos Teóricos y Contextuales de la Transformación de Datos Históricos y Sistemas Multiagentes Conversacionales

Introducción

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. Fundamentación Teórica

1.1.1. Reconocimiento Óptico de Caracteres (OCR)

El Reconocimiento Óptico de Caracteres (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 [6]. 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.

1.1.2. 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 [7]. 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 [7]. 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.

1.1.3. 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 [8]. 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 [9]. 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 [9], junto con elementos como agentes, entorno, comunicación y organización [8], potencian las capacidades individuales de herramientas como los LLMs, abriendo la puerta a procesos más robustos de análisis y estructuración de datos.

1.1.4. 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 [10, 11]. 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.

1.1.5. 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 [12]. Estas representaciones capturan relaciones semánticas y contextuales entre los datos,

permitiendo a los sistemas de inteligencia artificial identificar similitudes y diferencias con alta precisión [12]. 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.

1.1.6. 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 [13, 14]. Estas bases utilizan métricas como la distancia coseno o euclidiana para buscar similitudes en espacios multidimensionales, ofreciendo una solución escalable para grandes volúmenes de información no estructurada [15, 16]. 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.

1.2. 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 [17]. 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 [18].

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 [19].

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

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>

contexto [20].

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 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 [21].

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 es especializado, su precisión lo hace adecuado para OCR limpio.

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

Chatize	- Gratuita y accesible. - Conversión directa de PDFs en chatbots interactivos.	- 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** [22, 12, 17, 23, 24]. 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 [22], 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*.

1.3. Metodología de desarrollo de software

Para regir el proceso de desarrollo de esta investigación, se seleccionó la metodología ágil Extreme Programming (XP) debido a la capacidad para gestionar eficazmente proyectos complejos y adaptativos, como los que involucran inteligencia artificial y procesamiento de datos no estructurados. XP se basa en valores fundamentales como la comunicación, la simplicidad y el feedback, los cuales son esenciales para el éxito de proyectos innovadores y técnicamente desafiantes [25]. Al aplicar XP, se busca minimizar riesgos y maximizar la calidad del software mediante iteraciones cortas y entregas frecuentes de funcionalidades operativas [26]. Esto permite al equipo responder ágilmente a cambios en los requisitos o a nuevos descubrimientos durante el proceso de desarrollo, asegurando que el producto final sea relevante y efectivo para los usuarios [25]. 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.

1.4. 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.

1.4.1. 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 [27].

1.4.2. 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 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 [28]. 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

⁶<https://app.diagrams.net/>

texto histórico [29, 30].

1.4.3. 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 Langgraph, 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 [31].

Por otro lado, Langgraph (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. Langgraph 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 [32].

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.

1.4.4. 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 [33, 34].

1.4.5. 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 referencias específicas a eventos pasados o terminología arcaica [35, 36]. 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 [12]. 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 [12].

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 [12]. 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, Lang-

Chain, 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.

Propuesta de solución

Introducción

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**, y un microservicio dedicado que alberga el **sistema multiagente conversacional** basado en inteligencia artificial y que se expone con una api creada con **FastApi**. Esta elección arquitectónica promueve la escalabilidad, la mantenibilidad y la separación de intereses, permitiendo que cada componente evolucione de forma independiente.

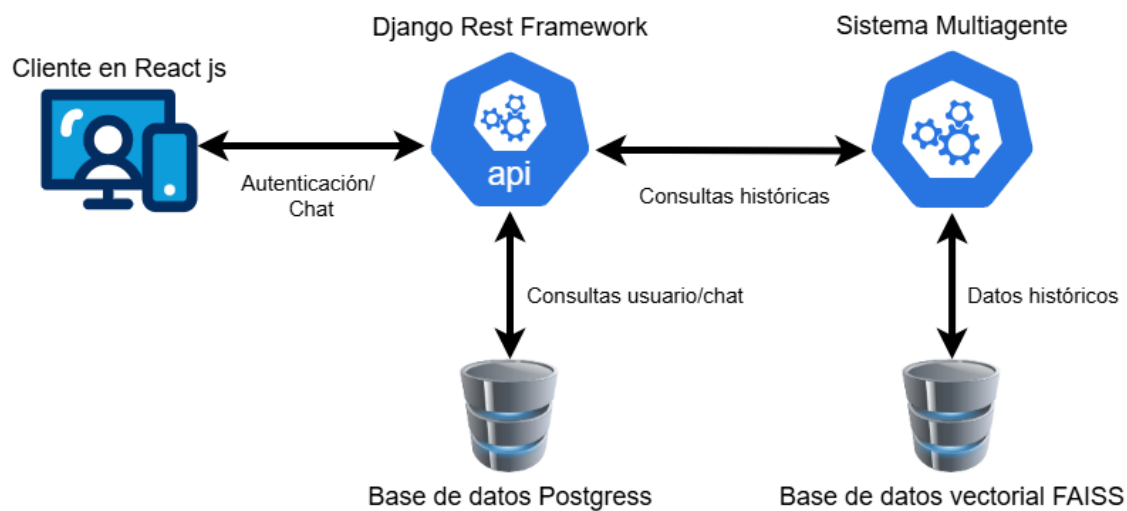


Figura 2.1. Arquitectura de la propuesta de solución.

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 HTTP (e.g., POST con JSON), al *endpoint* correspondiente del **backend DRF**.
3. El backend 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**

MAS a través de una llamada API interna (e.g., HTTP/JSON o gRPC).

4. El **microservicio MAS** 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 MAS devuelve el resultado procesado (e.g., un objeto JSON con texto y/o datos de imagen) al backend DRF.
6. El backend 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 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 del MAS** 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 en conocimiento estructurado y accesible, superando las limitaciones de herramientas previas mediante la integración de LLMs, RAG y una coordinación eficiente entre agentes especializados.

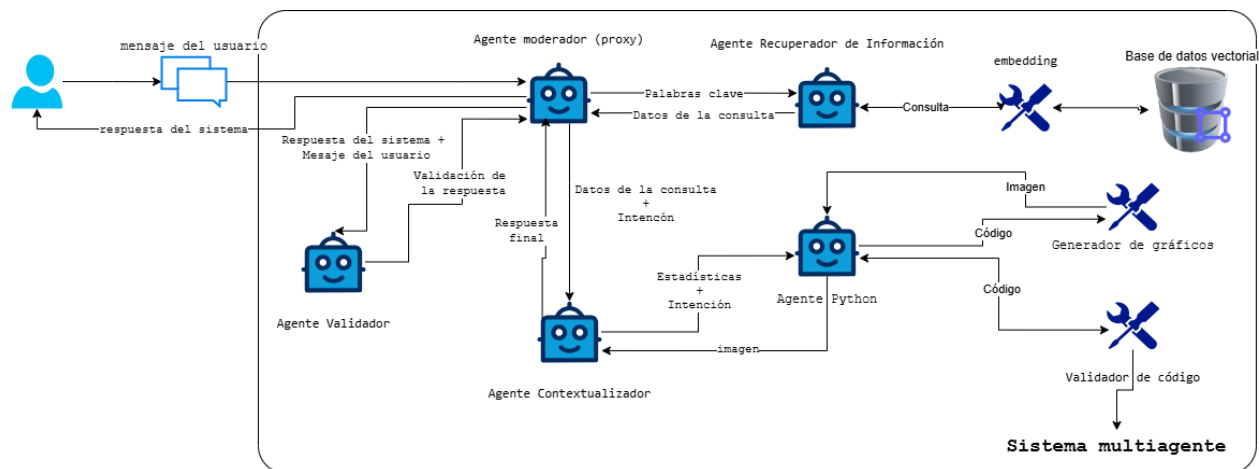


Figura 2.2. Flujo Interno del Microservicio Multiagente Conversacional.

El proceso interno del microservicio (Figura 2.2) inicia cuando recibe una consulta del backend DRF. Esta consulta es manejada por el **Agente de Moderación (proxy)**, que actúa como coordinador central dentro del microservicio. Este agente extrae palabras clave (e.g., “barcos”, “Europa”, “enero 1850”) y determina la intención del usuario. A continuación, delega la tarea al **Agente Recuperador de Información**, que

convierte las palabras clave en embeddings y consulta la base de datos vectorial especializada para recuperar fragmentos relevantes del *Diario de la Marina*. Los datos recuperados se devuelven al Agente de Moderación. Posteriormente, el Agente de Moderación envía la información recuperada, junto con la intención, al **Agente Contextualizador**. Este agente evalúa si se necesita una respuesta textual, estadísticas gráficas o ambas. Si se requieren gráficos (e.g., un gráfico de la frecuencia de llegada de barcos por mes), genera instrucciones que se envían al **Agente Python**. Este agente crea un script Python para generar la visualización, valida su corrección y lo convierte en una imagen. La imagen resultante se retorna al Agente Contextualizador. Si no se requieren gráficos, el Agente Contextualizador enriquece la información textual con contexto histórico relevante usando técnicas RAG. Antes de enviar la respuesta final de vuelta al backend DRF, el **Agente Validación** revisa la coherencia y precisión del contenido generado (texto y/o imagen), comparándolo con la consulta original. Finalmente, el Agente de Moderación 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 [37]. 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 [38].

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 [39].

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.

Observación: Se analizaron las plataformas del estado del arte (LangChain, Chatize, Haystack) mediante una observación activa de su funcionamiento con documentos de prueba, incluyendo algunos fragmentos digitalizados del *Diario de la Marina*. Este proceso consistió en interactuar con las herramientas, registrar sus respuestas a consultas simuladas (e.g., “Lista de puertos en 1850”) y evaluar su desempeño en términos de precisión, contextualización y usabilidad. Los resultados revelaron que, aunque estas soluciones manejan bien textos modernos, fallan en interpretar correctamente términos arcaicos, no integran contexto histórico externo y carecen de capacidades gráficas avanzadas [22, 31]. Estas observaciones subrayaron la necesidad de un sistema más especializado, con agentes dedicados a la contextualización y visualización.

La combinación de entrevistas y observación permitió triangular 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

Los requisitos funcionales son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar a entradas particulares y de cómo debería comportarse en situaciones específicas [39]. Estos requisitos describen tanto las interacciones previstas entre el software y su entorno, como las funcionalidades

y servicios que el sistema debe proporcionar a los usuarios. Además, son recopilados y documentados durante el proceso de análisis de negocio y se convierten en una parte fundamental del contrato de desarrollo de software. Pueden incluir, por ejemplo, la capacidad de un sistema para permitir a los usuarios adjuntar un archivo, eliminar o editar un elemento seleccionado, así como la forma en que el sistema debe reaccionar a ciertas entradas o situaciones particulares [40].

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
RF1	Registrar nuevo usuario	Permitir a un visitante registrarse proporcionando datos válidos (e.g., email, contraseña), que serán almacenados de forma segura por el backend.	Media	Alta
RF2	Autenticar usuario existente	Permitir a un usuario registrado iniciar sesión proporcionando credenciales válidas, verificadas por el backend, generando una sesión activa.	Media	Alta
RF3	Cerrar sesión de usuario	Permitir al usuario autenticado finalizar su sesión activa en el sistema.	Baja	Media
RF4	Proteger acceso a funcionalidades	Asegurar que solo los usuarios autenticados puedan acceder a las funcionalidades de chat, historial y consulta de datos históricos.	Media	Alta
RF5	Iniciar nueva conversación	Permitir al usuario autenticado crear una nueva sesión de chat independiente de las anteriores.	Baja	Alta
RF6	Listar conversaciones anteriores	Mostrar al usuario autenticado una lista de sus conversaciones previas para poder seleccionirlas y revisarlas.	Media	Media
RF7	Seleccionar conversación existente	Permitir al usuario seleccionar una conversación de su historial para visualizarla y continuarla.	Media	Media

Continúa en la siguiente página...

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

ID	Nombre	Descripción	Complejidad	Prioridad
RF8	Eliminar conversación del historial	Permitir al usuario eliminar permanentemente una conversación específica de su historial.	Media	Baja
RF9	Persistir historial de conversaciones	El backend debe almacenar de forma persistente las consultas y respuestas asociadas a cada conversación de cada usuario.	Media	Alta
RF10	Ingresar consulta en chat activo	Permitir al usuario escribir y enviar una consulta en lenguaje natural dentro de la conversación activa.	Baja	Alta
RF11	Visualizar intercambio en chat	Mostrar de forma clara y ordenada el diálogo (consultas del usuario y respuestas del sistema) dentro de la conversación activa.	Media	Alta
RF12	Limpiar contenido del chat actual	Permitir al usuario borrar visualmente el contenido de la conversación activa en la interfaz (sin eliminarla del historial).	Baja	Baja
RF13	Transmitir consulta al backend	El frontend debe enviar la consulta del usuario, junto con el identificador de la conversación activa y el token de sesión, al backend.	Media	Alta
RF14	Recibir consulta y contexto	El backend debe recibir la consulta, el ID de conversación y validar la sesión antes de procesar la solicitud.	Media	Alta
RF15	Delegar procesamiento de consulta al MAS	El backend debe enviar la consulta (y potencialmente contexto de conversación si es relevante para el MAS) al microservicio MAS para su procesamiento.	Media	Alta
RF16	Recibir solicitud de procesamiento	El microservicio MAS debe recibir la consulta (vía API) para iniciar el flujo de agentes.	Media	Alta
RF17	Extraer palabras clave de consulta	El Agente Moderador debe analizar la consulta para identificar términos clave relevantes.	Alta	Alta

Continúa en la siguiente página...

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

ID	Nombre	Descripción	Complejidad	Prioridad
RF18	Determinar intención de consulta	El Agente Moderador debe interpretar el propósito principal de la consulta del usuario (información, análisis, etc.).	Alta	Alta
RF19	Generar embeddings para búsqueda	El Agente Recuperador debe convertir las palabras clave en representaciones vectoriales (embeddings) adecuadas para la búsqueda semántica.	Alta	Alta
RF20	Recuperar información de BD Vectorial	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
RF21	Sintetizar y contextualizar respuesta textual	El Agente Contextualizador debe generar una respuesta coherente en lenguaje natural, integrando la información recuperada y añadiendo contexto histórico si es pertinente.	Alta	Alta
RF22	Identificar necesidad de visualización	El Agente Contextualizador debe determinar si la consulta o los datos recuperados justifican la creación de una representación gráfica.	Media	Media
RF23	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 Python.	Alta	Alta
RF24	Generar script de visualización	El Agente Python debe crear un script ejecutable (Python) que produzca la visualización solicitada a partir de los datos y especificaciones.	Alta	Alta
RF25	Validar corrección de script	El Agente Python debe verificar que el script generado es sintácticamente correcto y no contiene errores obvios antes de ejecutarlo.	Alta	Alta

Continúa en la siguiente página...

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

ID	Nombre	Descripción	Complejidad	Prioridad
RF26	Producir imagen de visualización	El Agente Python debe ejecutar el script validado para generar la visualización como un archivo de imagen (e.g., PNG, JPG).	Alta	Alta
RF27	Integrar texto y/o imagen en respuesta preliminar	El Agente Contextualizador (o Moderador) debe combinar la respuesta textual y/o la imagen generada en una estructura de respuesta unificada.	Alta	Alta
RF28	Validar coherencia y relevancia de 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
RF29	Ensamblar respuesta final del MAS	El Agente Moderador debe preparar la respuesta validada en el formato esperado por la API del microservicio (e.g., JSON con campos para texto e imagen).	Alta	Alta
RF30	Retornar respuesta procesada al backend	El microservicio MAS debe enviar la respuesta final ensamblada al backend a través de su API.	Media	Alta
RF31	Recibir y almacenar respuesta	El backend debe recibir la respuesta del MAS y almacenarla como parte del historial de la conversación activa.	Media	Alta
RF32	Enviar respuesta formateada al frontend	El backend debe enviar la respuesta (texto y/o referencia a la imagen) al frontend para su visualización.	Media	Alta
RF33	Presentar respuesta al usuario	El frontend debe mostrar la respuesta recibida (texto y/o imagen) dentro de la interfaz de chat de la conversación activa.	Baja	Media

2.2.3. Requisitos no funcionales

Los requisitos no funcionales se refieren a requerimientos de calidad, que representan restricciones o las cualidades que el sistema debe tener tales como: la usabilidad, el rendimiento, la escalabilidad, la portabilidad, la disponibilidad, entre otros [39]. Estos requisitos poseen una naturaleza abstracta e intangible en comparación con los RF y esto hace que sean más difíciles de especificar o documentar formalmente. No alteran la funcionalidad del sistema, pero pueden añadir nuevos requisitos funcionales. Describen las características y atributos que debe tener el software para funcionar de manera eficiente y confiable y cumplir con las necesidades del usuario [41]. La gestión adecuada de los requisitos no funcionales es crucial para garantizar la calidad del sistema.

A continuación, se listan los requisitos no funcionales identificados:

Tabla 2.2. Tabla de Requisitos No Funcionales (RNF)

ID	Atributo/Categoría	Descripción
RNF1	Rendimiento del Sistema	Atributos relacionados con la velocidad y eficiencia operativa percibida por el usuario y entre componentes.
RNF1.1	Latencia de respuesta (End-to-End)	El tiempo total desde que el usuario envía una consulta en React hasta que recibe la respuesta textual debe ser en promedio ≤ 15 segundos (para 1 usuario activo).
RNF1.2	Latencia de generación de gráficos	El tiempo adicional para generar y mostrar gráficos estándar en la interfaz React no debe exceder los 30 segundos sobre la respuesta textual base.
RNF1.3	Eficiencia de recuperación (MAS)	La consulta interna del microservicio MAS a la BD vectorial debe completarse en promedio en <5 segundos.
RNF1.4	Latencia de comunicación inter-servicios	La comunicación API entre DRF y MAS debe tener una latencia baja (e.g., promedio $<500\text{ms}$) bajo carga normal.
RNF2	Usabilidad	Atributos relacionados con la facilidad de uso e interacción con la interfaz web.

Continúa en la siguiente página...

Tabla 2.2. Tabla de Requisitos No Funcionales (RNF) (Continuación)

ID	Atributo/Categoría	Descripción
RNF2.1	Interfaz de usuario intuitiva	La interfaz web desarrollada con React (incluyendo chat, historial, login/registro) debe ser simple, estéticamente agradable y usable sin formación específica.
RNF2.2	Claridad de la información	Las respuestas textuales deben ser coherentes y legibles; los gráficos deben tener títulos/etiquetas claras y ser comprensibles en la interfaz React.
RNF2.3	Retroalimentación al usuario	La interfaz React debe proporcionar indicaciones visuales claras y descriptivas sobre el estado (e.g., enviando consulta, procesando, error en backend/MAS).
RNF2.4	Diseño responsivo	La interfaz web (React) debe adaptarse y ser funcional en diferentes tamaños de pantalla (escritorio, tableta).
RNF3	Fiabilidad	Atributos relacionados con la robustez, disponibilidad y consistencia del sistema completo.
RNF3.1	Disponibilidad del Sistema	Los componentes críticos (DRF, MAS) deben tener una alta disponibilidad (e.g., >99.5 uptime) durante las horas operativas esperadas.
RNF3.2	Tolerancia a fallos (parcial)	Un fallo en la generación de gráficos (MAS) no debe impedir la entrega de la respuesta textual. Fallos en el MAS no deben impedir el acceso al historial en DRF (si aplica). Se debe informar al usuario del fallo parcial.
RNF3.3	Precisión de la validación (MAS)	El Agente de Validación dentro del MAS debe identificar respuestas incoherentes con una tasa de éxito razonable (ej. >80).
RNF3.4	Consistencia del historial	El historial de conversaciones almacenado en el backend DRF debe reflejar consistentemente los intercambios realizados por el usuario.
RNF4	Mantenibilidad	Atributos relacionados con la facilidad de modificación, corrección y evolución del código y la arquitectura.

Continúa en la siguiente página...

Tabla 2.2. Tabla de Requisitos No Funcionales (RNF) (Continuación)

ID	Atributo/Categoría	Descripción
RNF4.1	Modularidad Arquitectónica	La arquitectura debe mantener un bajo acoplamiento entre el frontend (React), backend (DRF) y microservicio (MAS), permitiendo modificaciones o reemplazos con impacto mínimo en otros componentes.
RNF4.2	Modularidad Interna (MAS)	Dentro del microservicio MAS, la arquitectura basada en agentes debe permitir la modificación/sustitución de agentes individuales con mínimo impacto en el resto del microservicio.
RNF4.3	Calidad del código	Seguir buenas prácticas de codificación para cada tecnología (React/JS/TS, Python/Django, Python/FastAPI), incluyendo código comentado, estructurado y adherencia a linters (e.g., PEP 8).
RNF4.4	Facilidad de actualización de datos (MAS)	Debe existir un proceso documentado y razonablemente sencillo para actualizar o añadir nuevos datos históricos a la BD vectorial utilizada por el MAS.
RNF5	Escalabilidad	Atributos relacionados con la capacidad del sistema para manejar un crecimiento en datos, carga o funcionalidad.
RNF5.1	Escalabilidad de datos (MAS)	La BD vectorial utilizada por el MAS debe poder manejar eficientemente el volumen de datos previsto (1844-1960) y permitir crecimiento futuro.
RNF5.2	Escalabilidad de carga	La arquitectura (especialmente el backend DRF y el microservicio MAS) debe estar diseñada para permitir escalabilidad horizontal (añadir más instancias) para manejar un aumento de usuarios concurrentes.
RNF6	Seguridad	Atributos relacionados con la protección de datos y la prevención de accesos no autorizados.
RNF6.1	Almacenamiento seguro de contraseñas	Las contraseñas de los usuarios deben almacenarse en el backend (DRF) utilizando algoritmos de hashing robustos con salt.

Continúa en la siguiente página...

Tabla 2.2. Tabla de Requisitos No Funcionales (RNF) (Continuación)

ID	Atributo/Categoría	Descripción
RNF6.2	Autenticación y Autorización	Implementar mecanismos seguros de autenticación (e.g., tokens JWT con expiración, sesiones seguras) y autorización para proteger los endpoints del backend (DRF).
RNF6.3	Protección contra vulnerabilidades web	Aplicar prácticas para mitigar riesgos comunes (OWASP Top 10) como XSS, CSRF, Inyección SQL en el backend DRF y frontend React.
RNF6.4	Comunicación segura	Toda la comunicación entre el navegador del usuario, el backend DRF y el microservicio MAS (si es externo) debe realizarse sobre HTTPS.
RNF6.5	Gestión segura de claves API	Las claves de API para servicios externos (e.g., LLM) deben gestionarse de forma segura (variables de entorno, gestores de secretos) y no exponerse en el código fuente o frontend.
RNF7	Restricciones Tecnológicas	Limitaciones o imposiciones sobre las herramientas, plataformas o estándares a usar.
RNF7.1	Stack tecnológico principal	El frontend debe usar React. El backend debe usar Django REST Framework. El microservicio MAS debe usar Python y FastAPI.
RNF7.2	Lenguajes principales	JavaScript/TypeScript para el frontend. Python para el backend y microservicio.
RNF7.3	Dependencias clave de IA (MAS)	El microservicio MAS dependerá de un modelo LLM, por tanto debe tener soporte para múltiples proveedores y una BD vectorial FAISS.

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 [42].

Se realiza una HU por cada RF del componente, a continuación, se mostrarán las HU correspondientes a los requisitos funcionales:

Tabla 2.3. Historia de usuario # 1

Historia de usuario	
Número: 1	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	
Descripción: Como visitante del sitio web, quiero poder registrar una nueva cuenta proporcionando mi email y una contraseña segura, para poder acceder a las funcionalidades del sistema de chat y guardar mi historial de conversaciones. (Corresponde principalmente a RF1, RF2, RF3, RF4)	
Precondiciones: <ul style="list-style-type: none">• El visitante no tiene una sesión activa.• El visitante se encuentra en la página o sección de registro.	
Flujo de acción: <ol style="list-style-type: none">1. Visitante completa el formulario de registro (email, contraseña, confirmación).2. Visitante envía el formulario.3. El sistema valida los datos y crea la cuenta en el backend.4. El sistema muestra un mensaje de éxito o error en la interfaz React.	
Observaciones: La validación de contraseña debe ser clara (e.g., longitud mínima). Los mensajes de error deben ser específicos (e.g., "Email ya registrado").	

Continúa en la próxima página

Tabla 2.3. Continuación de la página anterior

Interfaz:

Formulario de Registro en React:

Crea tu cuenta

Ingrese sus datos a continuación para crear su cuenta

Nombre
Daniel

Correo
drgrassnk445@gmail.com

Contraseña
.....

Create account

¿Ya tienes una cuenta? Iniciar sesión

Tabla 2.4. Historia de usuario # 2

Historia de usuario	
Número: 2	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 2.4. Continuación de la página anterior

<p>Descripción: Como usuario registrado, quiero poder iniciar sesión utilizando mi email y contraseña, para poder acceder a mis conversaciones anteriores y utilizar el chat. (Corresponde principalmente a RF5, RF6, RF7, RF8, RF9)</p> <p>Precondiciones:</p> <ul style="list-style-type: none">• El usuario tiene una cuenta registrada.• El usuario no tiene una sesión activa.• El usuario se encuentra en la página o sección de inicio de sesión. <p>Flujo de acción:</p> <ol style="list-style-type: none">1. Usuario completa el formulario de inicio de sesión (email, contraseña).2. Usuario envía el formulario.3. El sistema valida las credenciales en el backend y genera una sesión/token.4. Si el login es exitoso, el usuario es redirigido a la interfaz principal del chat. Si falla, se muestra un mensaje de error. <p>Observaciones: El manejo de la sesión/token en el frontend debe ser seguro.</p>

Continúa en la próxima página

Tabla 2.4. Continuación de la página anterior

Interfaz:

Formulario de Inicio de Sesión en React:

Iniciar sesión

Inserta tus credenciales para entrar a tu cuenta

Correo

drgrassnk445@gmail.com

Contraseña

.....

[Olvidaste la contraseña?](#)

Login

[No tienes una cuenta? Sign up](#)

Tabla 2.5. Historia de usuario # 3

Historia de usuario	
Número: 3	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 2.5. Continuación de la página anterior

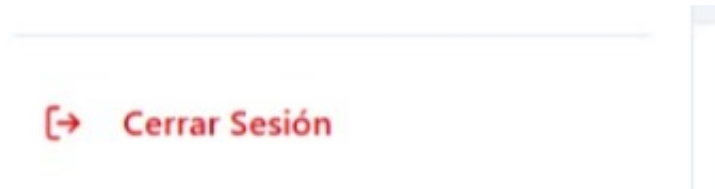
<p>Descripción: Como usuario autenticado, quiero poder cerrar mi sesión activa, para asegurar que mi cuenta quede protegida al dejar de usar el sistema. (Corresponde principalmente a RF11, RF12)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en la opción "Cerrar Sesión". 2. El frontend elimina el token/identificador de sesión local. 3. (Opcional) El frontend notifica al backend para invalidar la sesión/token en el servidor. 4. El usuario es redirigido a la página de inicio de sesión o a una página pública.
<p>Observaciones: Debe ser una acción fácilmente accesible desde la interfaz principal.</p>
<p>Interfaz:</p> <div style="text-align: center;"> <p>Botón Cerrar Sesión en React:</p>  </div>

Tabla 2.6. Historia de usuario # 4

Historia de usuario	
Número: 4	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 2.6. Continuación de la página anterior


<p>Descripción: Como usuario autenticado, quiero poder iniciar una nueva conversación de chat, para poder realizar consultas sobre un tema nuevo sin mezclarlo con diálogos anteriores. (Corresponde principalmente a RF5)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en la opción "Nueva Conversación"(o similar). 2. La interfaz de chat principal se limpia o se presenta una nueva interfaz vacía. 3. El sistema (backend) está listo para asociar las siguientes consultas a una nueva conversación en el historial.
<p>Observaciones: La acción debe ser clara y resultar en una interfaz lista para una nueva interacción.</p>
<p>Interfaz:</p> <p style="text-align: center;">Botón Nueva Conversación en React:</p> 

Tabla 2.7. Historia de usuario # 5

Historia de usuario	
Número: 5	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 2.7. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero poder ver una lista de mis conversaciones anteriores (e.g., con títulos o fechas), para poder localizar y seleccionar una para revisión o continuación. (Corresponde principalmente a RF6)</p> <p>Precondiciones:</p> <ul style="list-style-type: none">• El usuario tiene una sesión activa.• El usuario ha tenido conversaciones previas (opcionalmente). <p>Flujo de acción:</p> <ol style="list-style-type: none">1. Usuario accede a la sección o panel de historial.2. El frontend solicita la lista de conversaciones al backend.3. El backend recupera y envía la lista asociada al usuario.4. El frontend muestra la lista de conversaciones en la interfaz. <p>Observaciones: La lista debe ser fácil de navegar. Considerar cómo generar títulos/identificadores útiles para cada conversación.</p>

Continúa en la próxima página

Tabla 2.7. Continuación de la página anterior

Interfaz:	
Panel/Sección de Historial en React:	
	

Tabla 2.8. Historia de usuario # 6

Historia de usuario	
Número: 6	Nombre: Abrir una conversación del historial
Usuario: Usuario autenticado	

Continúa en la próxima página

Tabla 2.8. Continuación de la página anterior

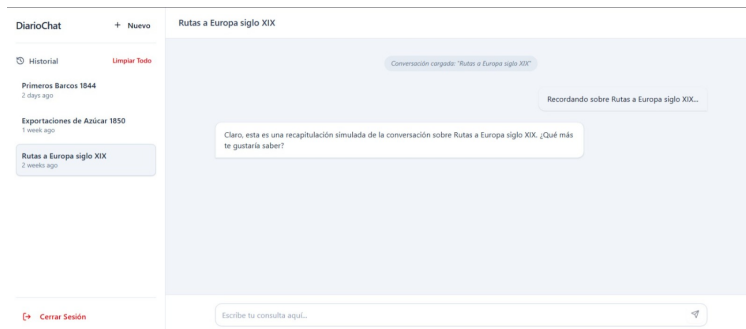
Prioridad en negocio: Media	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 2
Programador responsable: Daniel Rojas Grass	
<p>Descripción: Como usuario autenticado, quiero poder seleccionar una conversación específica de mi historial, para poder cargarla en la interfaz principal del chat, ver el diálogo completo y, si lo deseo, 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. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario hace clic en una conversación de la lista del historial. 2. El frontend solicita el contenido de esa conversación al backend. 3. El backend recupera las consultas y respuestas asociadas. 4. El frontend carga y muestra el diálogo completo en la interfaz principal del chat. 5. La conversación seleccionada se convierte en la conversación activa. 	
<p>Observaciones: La carga de la conversación debe ser rápida. La interfaz debe indicar claramente qué conversación está activa.</p>	
<p>Interfaz:</p> <p style="text-align: center;">Interacción con lista de Historial y Chat Principal en React:</p>  <p>The screenshot shows a web application titled 'DiarioChat'. On the left is a sidebar with a 'Historial' section containing a list of conversations: 'Primeros Bancos 1844' (2 days ago), 'Exportaciones de Azúcar 1850' (1 week ago), and 'Rutas a Europa siglo XIX' (2 weeks ago). The 'Rutas a Europa siglo XIX' item is highlighted. On the right is the main chat area, titled 'Rutas a Europa siglo XIX'. It shows a loading state with a message 'Conversación cargada: "Rutas a Europa siglo XIX"' and a progress indicator 'Recordando sobre Rutas a Europa siglo XIX...'. Below this, a message bubble says 'Claro, esta es una recapitulación simulada de la conversación sobre Rutas a Europa siglo XIX. ¿Qué más te gustaría saber?'. At the bottom, there is a text input field with the placeholder 'Escribe tu consulta aquí...' and a send button.</p>	

Tabla 2.9. Historia de usuario # 7


Historia de usuario	
Número: 7	Nombre: Eliminar una conversación del historial
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 poder eliminar una conversación específica de mi historial, para que ya no aparezca en mi lista y sus datos sean borrados permanentemente. (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 una conversación específica. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario selecciona la opción de eliminar para una conversación específica. 2. El sistema pide confirmación al usuario. 3. Si el usuario confirma, el frontend envía la solicitud de eliminación al backend. 4. El backend elimina los datos de la conversación asociada al usuario. 5. La conversación eliminada desaparece de la lista del historial en el frontend. 	
Observaciones: La eliminación debe requerir confirmación para evitar borrados accidentales.	
<p>Interfaz:</p> <p>Opción de Eliminar en la lista de Historial:</p>  <p>The screenshot shows a mobile application interface. At the top, there is a header with a clock icon and the word 'Historial' in blue, and a red button labeled 'Limpiar Todo'. Below the header is a list item with a light blue background. The list item contains the text 'Primeros Barcos 1844' in bold and '2 days ago' in a smaller font. To the right of the list item is a red trash can icon inside a light red circle, which is the option to delete the conversation.</p>	

Tabla 2.10. Historia de usuario # 8

Historia de usuario	
Número: 8	Nombre: Interactuar con el chat activo
Usuario: Usuario autenticado	
Prioridad en negocio: Alta	Riesgo en desarrollo: Bajo
Puntos estimados: 1 semana	Iteración asignada: 1
Programador responsable: Daniel Rojas Grass	
<p>Descripción: Como usuario autenticado, quiero poder escribir consultas en lenguaje natural en la interfaz de chat activa (React), enviarlas al sistema, ver las respuestas del sistema (texto y/o imágenes) y limpiar visualmente el contenido del chat actual si lo deseo. (Corresponde principalmente a RF10, RF11, RF12, RF13)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario tiene una sesión activa. • El usuario tiene una conversación activa (nueva o cargada del historial). <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. Usuario ingresa una consulta en el campo de texto del chat. 2. Usuario envía la consulta (e.g., presionando Enter o un botón Enviar). 3. El frontend envía la consulta al backend. 4. El backend procesa y obtiene una respuesta (interactuando con el MAS). 5. El backend envía la respuesta al frontend. 6. El frontend muestra la consulta del usuario y la respuesta del sistema en el área de diálogo. 7. (Opcional) Usuario hace clic en "Limpiar Chat" el contenido visual del chat actual se borra. 	
<p>Observaciones: La interfaz debe ser responsiva. La presentación del diálogo (usuario/sistema) debe ser clara. Limpiar el chat no debe eliminarlo del historial.</p>	

Continúa en la próxima página

Tabla 2.10. Continuación de la página anterior



Tabla 2.11. Historia de usuario # 9

Historia de usuario	
Número: 9	Nombre: Obtener respuesta relevante a consulta histórica
Usuario: Usuario autenticado	
Prioridad en negocio: Alta	Riesgo en desarrollo: Moderado
Puntos estimados: 4 semanas	Iteración asignada: 2
Programador responsable: Daniel Rojas Grass	

Continúa en la próxima página

Tabla 2.11. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero que al enviar una consulta sobre el transporte marítimo del Diario de la Marina, el sistema la procese utilizando el microservicio MAS para comprenderla, buscar información relevante en los datos históricos, contextualizarla adecuadamente y validar su coherencia, para recibir una respuesta precisa y útil. (Corresponde principalmente al flujo interno del MAS: RF17 a RF29)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario ha enviado una consulta válida desde el chat activo (HU:08). • El backend ha delegado la consulta al microservicio MAS. • El microservicio MAS y sus componentes (agentes, BD vectorial, LLM) están operativos. <p>Flujo de acción (interno del MAS):</p> <ol style="list-style-type: none"> 1. MAS recibe la consulta. 2. Agente Moderador analiza (palabras clave, intención). 3. Agente Recuperador busca información en BD vectorial. 4. Agente Contextualizador sintetiza respuesta textual y determina si se necesitan gráficos. 5. (Si aplica) Agente Python genera y valida la imagen del gráfico. 6. (Si aplica) Se integra texto e imagen. 7. Agente Validación verifica coherencia y precisión. 8. Agente Moderador ensambla la respuesta final para la API del MAS. <p>Observaciones: La relevancia de la información recuperada es clave. La contextualización debe ser históricamente apropiada. La validación debe minimizar errores o "alucinaciones". La latencia del MAS es importante (ver RNF).</p> <p>Interfaz:</p> <p>[N/A - Proceso interno del MAS]</p>	
---	--

Tabla 2.12. Historia de usuario # 10

Historia de usuario	
Número: 10	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	

Continúa en la próxima página

Tabla 2.12. Continuación de la página anterior

<p>Descripción: Como usuario autenticado, quiero que cuando mi consulta sugiera un análisis de patrones o tendencias (e.g., "puertos más activos en X año"), el sistema genere y me presente una visualización gráfica (e.g., gráfico de barras, mapa) junto con la respuesta textual, para facilitar la comprensión de los datos. (Corresponde principalmente a RF22, RF23, RF24, RF25, RF26)</p> <p>Precondiciones:</p> <ul style="list-style-type: none"> • El usuario ha enviado una consulta que el Agente Contextualizador interpreta como necesitada de un gráfico (dentro de HU:09). • Los datos necesarios para el gráfico están disponibles. <p>Flujo de acción:</p> <ol style="list-style-type: none"> 1. El Agente Contextualizador determina la necesidad del gráfico. 2. El Agente Python genera, valida y produce la imagen del gráfico. 3. La imagen es integrada en la respuesta final del MAS. 4. La respuesta (incluyendo la imagen) se envía al backend y luego al frontend. 5. El frontend muestra la imagen del gráfico junto al texto en la interfaz de chat.
<p>Observaciones: Los gráficos deben ser legibles y apropiados para los datos. La generación no debe fallar catastróficamente (ver RNF).</p>
<p>Interfaz:</p> <p>Visualización de imagen dentro del Chat en React</p>

2.3. 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 [43]. 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.

2.3.1. Patrones Arquitectónicos

Estos patrones definen la estructura general del sistema y cómo sus componentes principales interactúan.

Microservicios

La arquitectura general de la solución (descrita en la Sección 2.1) adopta el patrón de **Microservicios**. El sistema se descompone en componentes independientes con responsabilidades claras: el frontend (React), el backend API (Django REST Framework) y el servicio de procesamiento de IA (Sistema Multiagente expuesto vía FastAPI).

Justificación: Esta separación permite el desarrollo, despliegue y escalado independientes de cada parte. Por ejemplo, el microservicio MAS, que es intensivo en recursos de IA, puede escalarse por separado del backend DRF, que maneja la lógica de negocio y la gestión de usuarios/conversaciones. Además, facilita el uso de tecnologías más adecuadas para cada tarea (React para UI, DRF para API web robusta, FastAPI/Langchain para el MAS).

2.3.2. Patrones de Comportamiento

Estos patrones se centran en la comunicación efectiva y la asignación de responsabilidades entre objetos/-componentes.

Strategy (Estrategia)

Dentro del microservicio MAS, específicamente en el **Agente Contextualizador**, se aplica el patrón Strategy. Dependiendo de la intención identificada de la consulta del usuario (e.g., solicitud de información simple vs. solicitud de análisis que requiere visualización), el agente selecciona una estrategia diferente para construir la respuesta final.

Justificación: Una estrategia podría ser simplemente formatear el texto recuperado y enriquecerlo, mientras que otra estrategia implicaría coordinarse con el Agente Python para generar un gráfico e integrarlo en la

respuesta. Esto permite añadir o modificar formas de responder a diferentes intenciones sin alterar la lógica principal del agente contextualizador, simplemente implementando nuevas estrategias.

2.3.3. Patrones Estructurales

Estos patrones se ocupan de cómo se componen las clases y los objetos para formar estructuras más grandes y flexibles.

Facade (Fachada)

La **API del microservicio MAS (construida con FastAPI)** actúa como una Fachada para el complejo sistema multiagente interno. El backend DRF interactúa únicamente con esta API simplificada (e.g., un endpoint `/query`) sin necesidad de conocer los detalles de la coordinación entre los agentes (Moderador, Recuperador, Contextualizador, etc.), el acceso a la base de datos vectorial o las llamadas al LLM.

Justificación: Este patrón simplifica la interacción entre el backend DRF y el microservicio MAS. Reduce el acoplamiento, ya que los cambios en la lógica interna del MAS no afectan al DRF siempre que la interfaz de la API (la fachada) se mantenga estable.

2.3.4. Patrones de Concurrencia

Estos patrones abordan problemas relacionados con la ejecución simultánea de tareas para mejorar el rendimiento y la capacidad de respuesta.

Invocación Asíncrona

La comunicación entre los diferentes componentes de la arquitectura (Frontend ↔ Backend, Backend ↔ Microservicio MAS) y potencialmente dentro del MAS (llamadas a LLMs externos) utiliza invocaciones asíncronas. Por ejemplo, cuando el backend DRF llama a la API del microservicio MAS, no espera bloqueado, sino que utiliza mecanismos `‘async/await’` (propios de frameworks como DRF/FastAPI si se configuran así) o maneja la respuesta cuando esté disponible.

Justificación: Esto evita que los componentes se bloqueen mientras esperan respuestas de red o tareas largas (como el procesamiento de IA), mejorando la capacidad de respuesta general del sistema y permitiendo un uso más eficiente de los recursos del servidor, ya que puede manejar otras solicitudes mientras espera. Frameworks como FastAPI están diseñados específicamente para facilitar este patrón.

2.4. Targetas 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 [44]. La representación del sistema multiagente 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 2.13. Tarjeta CRC: ModeratorAgent

Tarjeta CRC	
Clase	ModeratorAgent
Responsabilidades: Recibir la consulta del usuario Extraer palabras clave e identificar la intención Coordinar el flujo de información entre agentes Ensamblar y entregar la respuesta final al usuario	Colaboración: InformationRetrievalAgent ContextualizationAgent ValidationAgent

Tabla 2.14. Tarjeta CRC: InformationRetrievalAgent

Tarjeta CRC	
Clase	InformationRetrievalAgent
Responsabilidades: Convertir las palabras clave en embeddings Consultar la base de datos vectorial para recuperar la información relevante Devolver los resultados al ModeratorAgent	Colaboración: ModeratorAgent FaissDatabase

Tabla 2.15. Tarjeta CRC: ContextualizationAgent

Tarjeta CRC	
Clase	ContextualizationAgent
Responsabilidades: Recibir la información relevante y la intención del usuario Generar indicaciones de estadísticas o contextualizar la información según la petición del usuario Enviar la información procesada al ModeratorAgent	Colaboración: ModeratorAgent PythonAgent

Tabla 2.16. Tarjeta CRC: ValidationAgent

Tarjeta CRC	
Clase	ValidationAgent
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: ModeratorAgent ContextualizationAgent

Tabla 2.17. Tarjeta CRC: PythonAgent

Tarjeta CRC	
Clase	PythonAgent
Responsabilidades: Generar un script de Python para representar estadísticas gráficas cuando sea necesario Validar el script y asegurarse de que sea correcto Convertir el gráfico generado en una imagen que pueda ser integrada en la respuesta final	Colaboración: ContextualizationAgent ImageGenerationTool

2.5. Diagrama de componentes

Los diagramas de componentes son una herramienta fundamental en ingeniería de software para visualizar la organización y las relaciones entre los componentes de un sistema. Estos diagramas proporcionan una vista de alto nivel de la arquitectura del sistema, mostrando cómo los componentes interactúan a través de interfaces bien definidas. Esto es especialmente útil en sistemas complejos, donde entender las interacciones entre componentes es crucial para el diseño y la implementación efectivos [45].

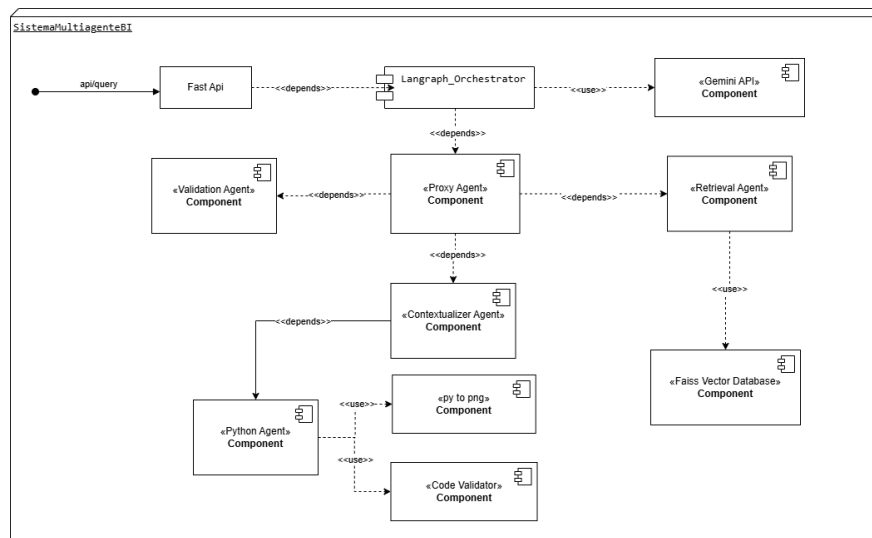


Figura 2.3. Diagrama de Componentes.

CAPÍTULO 3

Validación y Análisis de Resultados

Conclusiones generales

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

Recomendaciones

Bibliografía

- [1] Wikipedia, “Diario de la marina,” 2023. [Online]. Available: [https://es.wikipedia.org/wiki/Diario_de_la_Marina_\(La_Habana\)](https://es.wikipedia.org/wiki/Diario_de_la_Marina_(La_Habana))
- [2] Margot Note, “Metadata for archival collections: Challenges and opportunities,” 2018. [Online]. Available: <https://lucidea.com/blog/metadata-for-archival-collections-challenges-and-opportunities/>
- [3] A. Desconocido, “A proposed large language model-based smart search for archive system,” 2025. [Online]. Available: <https://paperswithcode.com/paper/a-proposed-large-language-model-based-smart>
- [4] —, “Enhancing degraded historical document images,” 2023, acceso: 5 de abril de 2025. [Online]. Available: <https://ai.renyi.hu/posts/revolutionizing-archival-document-processing-with-ai/>
- [5] —, “Ai-assisted digitalisation of historical documents,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLVIII-M-2-2023, pp. 557–563, 2023. [Online]. Available: <https://isprs-archives.copernicus.org/articles/XLVIII-M-2-2023/557/2023/>
- [6] B. Piryani, J. Mozafari, A. Abdallah, A. Doucet, and A. Jatowt, “Multiocr-qa: Dataset for evaluating robustness of llms in question answering on multilingual ocr texts,” *arXiv preprint arXiv:2502.16781*, 2025. [Online]. Available: <https://arxiv.org/abs/2502.16781>
- [7] T. Guo *et al.*, “Large language model based multi-agents: A survey of progress and challenges,” *arXiv preprint arXiv:2402.01680*, 2024. [Online]. Available: <https://arxiv.org/abs/2402.01680>

- [8] J. Zambrano *et al.*, “Multi-agent systems for the management of resources and activities in smart classrooms,” *IEEE Latin America Transactions*, vol. 18, no. 5, pp. 1511–1518, 2020. [Online]. Available: <https://latamt.ieee9.org/index.php/transactions/article/download/4857/1088>
- [9] M. E. Pérez-Pons, J. Parra, J. M. Corchado, R. Durán, J. Queiroz, and P. Leitão, “A brief review on multi-agent system approaches,” ResearchGate, 2021. [Online]. Available: http://uvadoc.uva.es/bitstream/handle/10324/62375/meperez_pons_Breve_repasso_a_los_enfoques.pdf
- [10] Y. Gao, “Retrieval-augmented generation for large language models: A survey,” *arXiv preprint arXiv:2312.10997*, 2024. [Online]. Available: <https://arxiv.org/abs/2312.10997>
- [11] P. Lewis *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *arXiv preprint arXiv:2005.11401*, 2020. [Online]. Available: <https://arxiv.org/abs/2005.11401>
- [12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” 2013.
- [13] X. Xie, H. Liu, W. Hou, and H. Huang, “A brief survey of vector databases,” in *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*, 2023, pp. 364–371.
- [14] Y. Han and C. Liu, “A comprehensive survey on vector database,” *arXiv preprint arXiv:2310.11703*, 2023. [Online]. Available: <http://arxiv.org/abs/2310.11703>
- [15] I. Azizi, K. Echihiabi, and T. Palpanas, “Vector search on billion-scale data collections,” *Proceedings of the VLDB PhD Workshop*, 2024. [Online]. Available: https://vldb.org/2024/files/phd-workshop-papers/vldb_phd_workshop_paper_id_13.pdf
- [16] P. Sun and R. Guo, “Soar: New algorithms for even faster vector search with scann,” Google Research Blog, 2024. [Online]. Available: <https://research.google/blog/soar-new-algorithms-for-even-faster-vector-search-with-scann/>
- [17] Chatize Team. (2023) Chatize website. [Online]. Available: <https://chatize.io/>
- [18] TextCortex. (2024) Textcortex chatpdf. Consultado en abril de 2025. [Online]. Available: <https://textcortex.com/chatpdf>

- [19] Vidnoz. (2024) Vidnoz chatpdf. Consultado en abril de 2025. [Online]. Available: <https://www.vidnoz.com/ai-tools/chat-pdf.html>
- [20] MyMap.AI. (2024) Mymap.ai chatpdf. Consultado en abril de 2025. [Online]. Available: <https://www.mymap.ai/chatpdf>
- [21] Adobe Inc. (2024) Adobe acrobat ai assistant. Consultado en abril de 2025. [Online]. Available: <https://www.adobe.com/acrobat/ai-assistant.html>
- [22] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, and S. Riedel, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *arXiv preprint*, 2020. [Online]. Available: <https://arxiv.org/abs/2005.11401>
- [23] Haystack Team. (2023) Haystack documentation. [Online]. Available: <https://haystack.deepset.ai/>
- [24] LlamaIndex Team. (2023) Llamaindex website. [Online]. Available: <https://www.llamaindex.ai/>
- [25] A. Alliance, “What is extreme programming (xp)?” 2017. [Online]. Available: <https://www.agilealliance.org/glossary/xp/>
- [26] W. contributors, “Extreme programming,” 2001. [Online]. Available: https://en.wikipedia.org/wiki/Extreme_programming
- [27] J. Ltd, “draw.io - herramienta de diagramación en línea,” 2025, accedido: 6 de abril de 2025. [Online]. Available: <https://www.drawio.com/>
- [28] LangChain, “Langgraph,” 2023. [Online]. Available: <https://www.langchain.com/langgraph>
- [29] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O’Reilly Media, 2009.
- [30] R. Python, “Natural language processing with spacy in python,” 2023. [Online]. Available: <https://realpython.com/natural-language-processing-spacy-python/>
- [31] L. Team, “Langchain documentation,” 2023, accedido: 2023-04-05. [Online]. Available: <https://langchain.readthedocs.io/en/latest/>
- [32] —, “Langgraph documentation,” 2023, accedido: 2023-04-05. [Online]. Available: <https://langgraph.readthedocs.io/en/latest/>

- [33] “What is version control?” 2024. [Online]. Available: <https://github.com/resources/articles/software-development/what-is-version-control>
- [34] “Implementing version control with git and github as a learning tool,” 2021. [Online]. Available: <https://www.tandfonline.com/doi/full/10.1080/10691898.2020.1848485>
- [35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [36] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” 2019.
- [37] R. S. Pressman *et al.*, “A practitioner’s approach,” *Software Engineering*, vol. 2, pp. 41–42, 2010.
- [38] A. Palli, “La importancia del análisis de requerimientos en el desarrollo de software,” *PROEFEX*, 2023. [Online]. Available: <https://proefexperu.com/>
- [39] I. Sommerville, “Software engineering (ed.),” *America: Pearson Education Inc*, 2011.
- [40] G. E. G. Chanchí, M. C. A. Gómez, and W. Y. M. Campo, “Propuesta de un videojuego educativo para la enseñanza-aprendizaje de la clasificación de requisitos en ingeniería de software,” *Revista Ibérica de Sistemas e Tecnologías de Informação*, no. E22, pp. 1–4, 2019.
- [41] Y. Molina Hernández, A. Granda Dihigo, and A. Velázquez Cintra, “Los requisitos no funcionales de software. una estrategia para su desarrollo en el centro de informática médica,” *Revista Cubana de Ciencias Informáticas*, vol. 13, no. 2, pp. 77–90, 2019.
- [42] A. Menzinsky, G. López, J. Palacio, M. Á. Sobrino, R. Álvarez, and V. Rivas, “Historias de usuario,” *Ingeniería de requisitos ágil*, 2018.
- [43] O. D. Gavilánez Álvarez, N. P. Layedra Larrea, and M. V. Ramos Valencia, “Análisis comparativo de patrones de diseño de software,” *Polo del Conocimiento: Revista científico - profesional*, vol. 7, no. 7, pp. 2146–2165, julio 2022. [Online]. Available: <https://polodelconocimiento.com/ojs/index.php/es/article/view/4338>
- [44] K. Beck and W. Cunningham, “A laboratory for teaching object-oriented thinking,” 1989, disponible en: <https://c2.com/doc/oopsla89/paper.html>.

[45] Restackio, “Component diagram in software engineering,” 2025. [Online]. Available: <https://www.restack.io/p/combinatorial-applications-in-software-engineering-answer-component-diagram>

Anexos

Notación matemática. Un ejemplo de Anexo

Para lograr una descripción homogénea de los principales conceptos y definiciones se debe establecer la notación básica de modo que se comprendan los referentes teóricos y las contribuciones de la presente investigación. A continuación se detallan elementos generales de la notación matemática a utilizar en este trabajo:

- V : Vocabulario de entrada.
- N : Cantidad de ejemplos de entrenamiento.
- d : Dimensión o cantidad de rasgos de un ejemplo de entrenamiento. En una entrada de texto puede representar el número de *tokens* que conforma el documento de entrada.
- d_{model} : Dimensión del vector *embedding* que representa una palabra o *token*.
- \mathbf{x} : Vector que representa un ejemplo de entrenamiento, tal que $\mathbf{x} \in \mathbb{R}^d$
- X : Conjunto de entrenamiento, tal que $X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_N\}$.
- W : Matriz que representa los parámetros de una ANN, tal que $W \in \mathbb{R}^{A \times K}$, siendo A el número de filas y K el número de columnas.
- W_i : Representa el vector de la i -ésima fila de la matriz W .
- $W_{:j}$: Representa el vector de la j -ésima columna de la matriz W .
- $w_{i,j}$: Elemento de la matriz W que se encuentra en la i -ésima fila y j -ésima columna, tal que $0 \leq i < A$ y $0 \leq j < K$.
- $F(W, X)$: Función objetivo de una ANN a optimizar de forma tal que $F(.) : \mathbb{R}^{A \times K} \rightarrow \mathbb{R}$.