UIDE
Powered by
Arizona State University

ESCUELA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

TESIS PREVIA A LA OBTENCIÓN DE TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN.

AUTOR: Santiago Michael Jiménez Orellana

TUTOR: Mgs. Ing. Milton Ricardo Palacios Morocho

"Desarrollo e Implementación de un Chatbot Basado en LLM para brindar información a la comunidad universitaria de la Universidad Internacional del Ecuador, campus Loja"

Loja - Ecuador

Octubre 2025

CERTIFICACIÓN DE AUTORÍA

Yo, **Santiago Michael Jiménez Orellana,** declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido presentado anteriormente para la obtención de ningún grado académico ni calificación profesional, y que he consultado la bibliografía que se detalla.

Cedo mis derechos de propiedad intelectual a la **Universidad Internacional del Ecuador**, para que pueda publicarlo y difundirlo en internet, de conformidad con lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás normativas aplicables.

Santiago Michael Jiménez Orellana
Autor

APROBACIÓN DEL TUTOR

Yo, Milton Ricardo Palacios Morocho, certifico que conozco al autor del presente trabajo de titulación **"Desarrollo e Implementación de un Chatbot Basado en LLM para brindar información a la comunidad universitaria de la Universidad Internacional del Ecuador, campus Loja"**, Santiago Michael Jiménez Orellana, siendo el responsable exclusiva tanto de su originalidad y autenticidad, como de su contenido.

Mgs. Ing. Milton Ricardo Palacios Morocho
DIRECTOR DEL TRABAJO DE TITULACIÓN

DEDICATORIA

El presente trabajo está dedicado con todo mi amor y cariño principalmente a mi pilares fundamentales en la vida, mis abuelos maternos. Que me enseñaron lo más preciado y valioso de la vida, los valores, la fortaleza y el amor para subsistir ante las adversidades. A mis familiares que viven el día a día conmigo, gracias por su fraterno y cálido apoyo. A mis amigos y conocidos durante todo el transcurso de mi carrera los cuales influyeron en mi descubrimiento y crecimiento personal. Por todo, gracias totales.

AGRADECIMIENTO

Mi sincero agradecimiento a mis familiares cercanos y lejanos que directa e indirectamente me han dado su apoyo incondicional y moral, sin embargo, mi más cálido agradecimiento es a mis abuelos maternos que son sus esfuerzos diarios en los días buenos y malos estuvieron ahí apoyándome y dándome fuerzas para seguir y no renunciar nunca a pesar de las circunstancias.

De igual manera, agradecer a mi director y tribunal de este trabajo por sus guías y apoyo durante el transcurso de mi trabajo de titulación. También mi agradecimiento a los docentes de la escuela por su esfuerzo en intentar crear profesionales competentes para la sociedad y su orientación en esta área del conocimiento tan extensa que es la informática.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La Universidad Internacional del Ecuador, campus Loja, en los últimos años ha experimentado un crecimiento en su población estudiantil y en la demanda de servicios administrativos y académicos. Este incremento ha generado dificultades en la gestión, acceso y difusión de información hacia su comunidad universitaria y el público en general, ya que los canales tradicionales de atención no siempre logran satisfacer de manera oportuna las múltiples consultas. Entre los principales inconvenientes se evidencian la falta de disponibilidad inmediata de datos actualizados, la duplicación de esfuerzos en procesos administrativos, los retrasos en la entrega de respuestas y, en consecuencia, un nivel considerable de insatisfacción por parte de los usuarios.

Para dar solución a esta problemática, el presente proyecto propone el diseño, desarrollo e implementación de un chatbot inteligente basado en un modelo de lenguaje grande (LLM), capaz de atender consultas frecuentes de manera rápida, precisa y accesible en cualquier momento. El proceso metodológico contempló la recolección de información institucional a través de técnicas de web scraping aplicadas a fuentes digitales oficiales, seguida de un proceso de preprocesamiento, limpieza y organización de los datos para garantizar la calidad de la información utilizada en el entrenamiento del modelo. Una vez completada esta etapa, se procedió al ajuste, entrenamiento y evaluación del LLM, asegurando que cumpliera con parámetros de precisión y pertinencia en las respuestas. Asimismo, se desarrolló una interfaz gráfica amigable, diseñada con criterios de usabilidad y accesibilidad, que facilita la interacción entre los usuarios y el sistema.

Los resultados esperados incluyen una mejora significativa en la eficiencia de la gestión de información institucional, la optimización de los recursos humanos y tecnológicos disponibles, así como un incremento en la satisfacción de la comunidad universitaria frente a los servicios de atención. El chatbot, al operar de manera continua y sin limitaciones horarias, representa un canal alternativo y complementario de comunicación que fortalece el vínculo entre la universidad y sus diferentes actores. Finalmente, se proyecta como una herramienta escalable y adaptable a otros contextos académicos o institucionales, consolidándose como un aporte a la innovación tecnológica y a la modernización de los procesos de gestión de información.

Palabras clave: Chatbot, LLM, gestión de información, scraping, universidad

ABSTRACT

The International University of Ecuador, Loja campus, has experienced significant growth in its student population and in the demand for academic and administrative services in recent years. This increase has created challenges in managing, accessing, and disseminating information to the university community and the general public, as traditional communication channels are not always able to respond promptly to the growing number of queries. Among the main difficulties identified are the lack of immediate access to updated data, duplication of administrative efforts, delays in responses, and, consequently, user dissatisfaction.

To address this issue, the present project proposes the design, development, and implementation of an intelligent chatbot based on a Large Language Model (LLM), capable of providing quick, accurate, and accessible responses at any time. The methodological process included the collection of institutional information through web scraping techniques applied to official digital sources, followed by data preprocessing, cleaning, and organization to ensure the quality of the information used for training the model. Once this stage was completed, the LLM was fine-tuned, trained, and evaluated to ensure accuracy and relevance in its responses. Additionally, a user-friendly graphical interface was developed, designed with usability and accessibility principles, to facilitate seamless interaction between users and the system.

The expected results include a significant improvement in the efficiency of institutional information management, optimization of available human and technological resources, and greater satisfaction among the university community with the services provided. By operating continuously and without time restrictions, the chatbot represents an alternative and complementary communication channel that strengthens the relationship between the university and its stakeholders. Finally, the system is projected as a scalable and adaptable tool for other academic or institutional contexts, contributing to technological innovation and the modernization of information management processes.

Keywords: Chatbot, LLM, information management, scraping, university

INTRODUCCIÓN

En la actualidad, las instituciones educativas se enfrentan al desafío de proporcionar información precisa y oportuna a una audiencia cada vez más extensa y diversificada. La Universidad Internacional del Ecuador, campus Loja, tampoco es una excepción. La necesidad constante de información por parte de sus estudiantes, candidatos y empleados promueve la implantación de enfoques más avanzados capaces no solo de mejorar drásticamente los canales de comunicación, sino también aportar una ventaja significativa a la experiencia del usuario. Una perspectiva bien redondeada es necesaria para cubrir las necesidades de información de toda la comunidad.

Ante esta situación, la creación e implementación de un chatbot basado en LLM de gran modelo de lenguaje se considera una opción efectiva según las tendencias modernas y novedosas. Los chatbots impulsados por AI tienen la oportunidad de trabajar con grandes cantidades de información y producir respuestas naturales en contexto, lo que los convierte en el instrumento ideal para distribuir información. Además, la aplicación de tal plataforma traerá eficacia de comunicación a la institución, con la implementación de respuestas automatizadas a las preguntas frecuentes y la cobertura de 24/7.

La relevancia de esta propuesta se fundamenta en su potencial para transformar los canales de comunicación universitaria, optimizando tanto el acceso como la eficiencia operacional. La automatización de respuestas a consultas rutinarias libera recursos institucionales valiosos y garantiza atención continua, aspecto especialmente relevante en el contexto académico actual, caracterizado por su naturaleza dinámica y su proyección internacional. El empleo de modelos de lenguaje avanzados permite generar respuestas precisas y contextualizadas según los requerimientos particulares de cada usuario, estableciendo así nuevos estándares en la calidad del servicio informativo universitario (Goel et al., 2018).

En síntesis, esta iniciativa atiende requerimientos operativos inmediatos de la universidad y simultáneamente consolida el liderazgo institucional en la adopción de tecnologías emergentes para la gestión informativa y comunicacional. La implementación de esta herramienta refuerza el

perfil innovador de la institución y su dedicación hacia la excelencia en los procesos académicos y administrativos.

PLANTEAMIENTO DEL PROBLEMA

La Universidad Internacional del Ecuador, campus Loja, tiene un problema real con el manejo de información. Todos los días, estudiantes, empleados y visitantes hacen preguntas de todo tipo, y el personal que debe responder está desbordado. Esto causa que las respuestas tarden mucho y la gente se moleste.

Los sistemas que tiene ahora la universidad no están mal, pero no dan lo que la gente busca: respuestas al instante y que alguien esté disponible a cualquier hora. Además, encontrar algo en la página web es un dolor de cabeza. La información muchas veces está vieja, lo que hace todo peor.

El problema se hace más grande porque la universidad ha crecido mucho. Hay más carreras, más estudiantes, y por tanto, mucha más información que manejar. Sin algo que responda automáticamente las preguntas típicas, la universidad no puede trabajar bien ni dar un servicio decente.

Un chatbot podría arreglar todo esto. Daría respuestas al momento, estaría siempre disponible y sería preciso. Lo que queremos ver con este proyecto es si algo así realmente puede hacer que la comunicación en la universidad funcione mejor, que manejar la información sea más fácil, y que la gente quede contenta con el servicio que recibe.

OBJETIVOS

Objetivo General

Desarrollar e implementar un chatbot basado en modelo del lenguaje grande (LLM) para proporcionar información de manera eficiente y oportuna a la comunidad universitaria de la Universidad Internacional del Ecuador, campus Loja.

Objetivos específicos

- Realizar el proceso de extracción de datos utilizando Scrapy en los sitios web de la Universidad Internacional del Ecuador (UIDE) para obtener información relevante y actualizada, y acceder a la data física disponible en la Universidad para almacenarla en un repositorio digital.

- Preprocesar y limpiar los datos extraídos para asegurar su calidad y relevancia, y entrenar un LLM utilizando estos datos, ajustando los hiperparámetros y optimizando el modelo para un rendimiento adecuado.

- Evaluar el modelo entrenado utilizando conjuntos de datos de validación para medir su precisión y exactitud, y realizar los ajustes necesarios en el modelo basándose en los resultados de la evaluación para mejorar su rendimiento.

- Desarrollar una interfaz gráfica de usuario (GUI) amigable e intuitiva que permita a los usuarios interactuar fácilmente con el chatbot, e integrar la interfaz gráfica con el LLM para facilitar la comunicación y la entrega de información.

- Realizar pruebas exhaustivas del sistema completo, incluyendo el chatbot y la interfaz gráfica, para asegurar su correcto funcionamiento, identificar y corregir posibles errores, y garantizar que cumple con los requerimientos de la comunidad universitaria.

CAPÍTULO I: ESTADO DEL ARTE

ANTECEDENTES

En el campo de los agentes conversacionales académicos, los últimos cinco años han estado marcados por un fuerte impulso hacia la automatización de la comunicación y la mejora de la experiencia de usuario mediante inteligencia artificial avanzada. A nivel nacional, Santacruz Sánchez (2022) creó un chatbot para la Unidad de Titulación de la UCE usando metodología ADDIE y entrevistó a ocho docentes para validar su prototipo. Su trabajo mostró que la herramienta recortó bastante los tiempos de respuesta, algo que nos ayudó mucho cuando hicimos nuestro prototipo y las pruebas con usuarios. Puma Quilumba (2023) trabajó con procesamiento de lenguaje natural en un sistema para la biblioteca de la UTN, usando Microsoft Bot Framework. Logró bajar 40% las consultas que se repetían todo el tiempo, y además nos dio ideas valiosas sobre cómo diseñar la interfaz y manejar los datos antes de procesarlos.

En otros países, Zhao et al. (2024) hicieron una revisión completa de frameworks y técnicas para entrenar LLM, incluyendo LoRA y adapters. Esto nos sirvió mucho para ajustar nuestros parámetros y elegir las mejores herramientas de Scrapy y entrenamiento. Chukwuere (2024) investigó el futuro de los chatbots generativos en universidades, encontrando problemas con la integridad académica, privacidad y escalabilidad. Propuso ciclos de evaluación y políticas que adoptamos para nuestro protocolo de validación. Como usé RAG en mi desarrollo, el trabajo de Neupane et al. (2024) fue clave. Ellos construyeron BARKPLUG V.2, un chatbot que accede a los datos de Mississippi State University usando pipelines RAG y mide su eficacia con el índice RAG y el SUS. Obtuvieron un RAG promedio de 0.96 y más del 85% de satisfacción, lo que confirma que RAG era la elección correcta para evitar alucinaciones y dar respuestas precisas.

Estas cinco investigaciones me dieron la base teórica y metodológica para cada etapa de mi tesis: desde extraer y limpiar datos (Scrapy + preprocesamiento), hasta entrenar y ajustar el LLM, construir la interfaz y hacer las pruebas de validación. Todo esto resultó en un chatbot RAG sólido y hecho a la medida de lo que necesita la comunidad de la UIDE, campus Loja.

MARCO TEÓRICO

1.1. Los chatbots en la educación superior.

1.1.1. Cómo han cambiado los agentes conversacionales.

Los primeros chatbots en universidades funcionaban con reglas muy básicas del tipo "si pasa esto, entonces haz aquello", lo que los hacía bastante limitados para entender cómo habla la gente realmente. Cuando aparecieron los modelos de aprendizaje profundo y las arquitecturas Transformer, como BERT y GPT, todo cambió. Estos chatbots empezaron a entender mejor lo que las personas querían decir y el contexto de sus preguntas (Vaswani et al., 2017). Este avance ha hecho que los chatbots puedan dar respuestas más coherentes y se adapten a diferentes áreas, algo muy importante en las universidades donde las preguntas pueden ir desde trámites sencillos hasta dudas complicadas sobre materias específicas.

1.1.2. Qué papel tienen los chatbots en las universidades.

En las universidades, los chatbots trabajan como asistentes virtuales que están disponibles las 24 horas, todos los días. Esto libera a los equipos administrativos de responder lo mismo una y otra vez, y hace que los estudiantes tengan una mejor experiencia con el servicio (Santacruz Sánchez, 2022; Dempere et al., 2024). También ayudan a que más gente pueda acceder a la información, especialmente aquellos que tienen horarios complicados o necesidades diferentes de idioma, cumpliendo con la responsabilidad que tienen las universidades de hacer que la información esté al alcance de todos.

1.2. Grandes modelos de lenguaje grande (LLM).

1.2.1 Arquitectura Transformer.

El Transformer es la base de los LLM que usamos hoy en día. Funciona con algo llamado mecanismos de atención, que básicamente le permiten al modelo decidir qué tan importante es

cada palabra en una oración, sin tener que leer todo en orden estricto como hacían los modelos anteriores. Esto hace que el entrenamiento sea mucho más rápido (Vaswani et al., 2017). Esto hace posible escalar modelos a miles de millones de parámetros, capaz de captar matices semánticos y relaciones complejas en grandes corpus de texto.

1.2.2. Métodos de Fine-Tuning.

El fine-tuning consiste en ajustar los pesos de un LLM preentrenado sobre un conjunto de datos específico. Técnicas como LoRA (Low-Rank Adaptation) y adapters minimizan la cantidad de parámetros que se modifican, reduciendo los requisitos de cómputo y memoria (Zhao et al., 2024). Aunque mejora la especialización del modelo, puede conllevar sobreajuste al dominio y no evita completamente las "alucinaciones" – respuestas plausibles pero incorrectas.

1.2.3. Ajuste Eficiente de parámetros(PEFT).

Además de LoRA y adapters, hay otras formas de hacer PEFT como prefix-tuning, que agrega vectores al principio del input sin tocar el modelo original, y prompt-tuning, que solo aprende tokens de prompt para cambiar cómo se comporta el LLM (Li & Liang, 2021). Con estas técnicas puedes reducir hasta 99% los parámetros que necesitas entrenar, lo que es genial cuando no tienes muchos recursos (Hu et al., 2021).

1.2.4. Aprendizaje continúo y llvido catastrófico.

Cuando entrenas un modelo para algo específico, a veces se le olvida lo que sabía antes, lo que se conoce como olvido catastrófico. Hay métodos como Elastic Weight Consolidation (EWC) que castigan cambios muy bruscos en los parámetros importantes, buscando un equilibrio entre especializarse en algo nuevo y no perder el conocimiento general (Kirkpatrick et al., 2017).

1.2.5. Evaluación y validación del Fine-Tuning.

No basta con usar métricas generales como perplexity. Es importante usar benchmarks específicos del área (como tests de precisión en preguntas frecuentes de universidades) y que personas reales evalúen si las respuestas tienen sentido y son correctas. También hay que revisar qué tan confiado está el modelo en sus respuestas para detectar cuando está inventando cosas (Jiang et al., 2023).

1.2.6. Costos y recursos.

Los costos de GPU, tiempo de entrenamiento y memoria cambian mucho según la técnica que uses: un fine-tuning completo puede necesitar muchas GPUs, mientras que LoRA o adapters funcionan con una sola tarjeta, algo muy importante cuando tienes presupuesto limitado (Raffel et al., 2020).

1.2.7. Ajuste con retroalimentación Humana (RLHF).

El fine-tuning con refuerzo basado en retroalimentación humana (RLHF) usa comentarios de usuarios o evaluadores para que el modelo se alinee mejor con lo que queremos en términos de usabilidad y seguridad, reduciendo respuestas tóxicas o sesgadas (Ouyang et al., 2022).

1.3. Recuperación Aumentada por Generación (RAG).

1.3.1. Principio y flujo de trabajo.

En RAG, primero se indexa un repositorio de documentos (por ejemplo, manuales, políticas, FAQ) en un motor de vectores como FAISS. Ante una consulta, se recuperan los fragmentos más relevantes y luego el LLM genera la respuesta basándose explícitamente en ese contexto. Este enfoque mitiga las alucinaciones al anclar la generación en datos verificables (Lewis et al., 2020).

1.3.2. Indexación y Embeddings.

Se elige un modelo de embeddings (p. ej. SBERT) y se configura la dimensión del vector—comúnmente entre 768 y 1 024—para balancear capacidad de representación y eficiencia de búsqueda en FAISS (Reimers & Gurevych, 2019; Johnson et al., 2019).

1.3.3. Estrategias de Recuperación.

Existen tres enfoques principales:

- **Dense Retrieval:** usa similitud de coseno sobre vectores densos.
- **Sparse Retrieval:** sistemas basados en BM25 o TF-IDF.=
- **Híbrido:** combina ambas para mejorar cobertura y precisión (Xiong et al., 2021).

1.3.4. Generación Guiada.

Tras la recuperación, el LLM incorpora los pasajes como contexto explícito en el prompt (prompt engineering), a menudo en un formato "<contexto>…<pregunta>" para reforzar la veracidad de la respuesta (Lewis et al., 2020).

1.3.5. Beneficios y consideraciones.

La principal ventaja de RAG radica en la precisión y veracidad de las respuestas, pues el modelo debe citar o parafrasear información existente, elevando la confianza de los usuarios. Sin embargo, requiere mantener un pipeline de actualización e indexación constante y un volumen de datos bien estructurado (Neupane et al., 2024). Además, la latencia puede ser mayor que en un fine-tuning puro, dado el paso adicional de recuperación.

1.3.6. Veracidad y Confianza.

Al anclar la generación en pasajes recuperados, RAG reduce las alucinaciones y permite citar textualmente las fuentes, mejorando la experiencia del usuario y la auditabilidad del sistema (Lewis et al., 2020).

1.3.7. Mantenimiento y Actualización.

Es imprescindible diseñar un pipeline de reindexación incremental para incorporar nuevos documentos (manuales UIDE, políticas internas) y migrar viejas versiones sin interrumpir el servicio (Neupane et al., 2024).

1.3.8. Latencia y Optimización.

El paso de recuperación añade latencia; técnicas como caching de embeddings frecuentes y optimización de proximidad de datos pueden mitigar retrasos y escalar consultas concurrentes (Karpukhin et al., 2020).

1.3.9. Evaluación End-to-End.

Se utilizan métricas mixtas: recall@k para la fase de recuperación y Exact Match/F1 para la generación, midiendo cuán precisa es la respuesta respecto a un ground-truth (Karpukhin et al., 2020).

1.3.10. Sesgos y Ética.

Es necesario revisar regularmente los fragmentos recuperados para evitar perpetuar sesgos o información desactualizada, y garantizar que el chatbot no ofrezca contenido inapropiado (Bender et al., 2021).

1.4. Comparativa: RAG vs. Fine-Tuning.

Tabla 1

Comparativa entre RAG y Fine-Tuning

Criterio	RAG (Retrieval-Augmented Generation)	Fine-Tuning
Proceso	1. Indexación de documentos en motor de vectores (p. ej., FAISS). 2. Recuperación de fragmentos relevantes. 3. Generación contextualizada.	Ajuste directo de los pesos del LLM usando datos de entrenamiento específico.
Ventajas	• Respuestas basadas en datos verificados (menos alucinaciones). • Fácil actualización de contenido sin reentrenar todo el modelo.	• Respuestas muy especializadas al dominio. • Menor latencia en generación, al no requerir etapa de recuperación.
Desventajas	• Mayor complejidad en infraestructura (indexación, actualización). • Latencia extra en cada consulta.	• Riesgo de sobreajuste y obsolescencia rápida de información. • No garantiza precisión si el LLM genera “alucinaciones”.
Recursos	• Necesita almacenamiento y computación para índices vectoriales. • La generación es similar a un LLM estándar.	• Requiere GPU/TPU significativos para reentrenar.
Casos de uso ideales	• FAQs extensas y dinámicas. • Documentación técnica que cambia frecuentemente.	• Tareas con vocabulario muy especializado y estático.

Nota. Elaboración propia.

La decisión de emplear Recuperación Aumentada por Generación (RAG) en lugar del enfoque tradicional de fine-tuning se fundamenta en varias consideraciones prácticas y estratégicas. En primer lugar, RAG garantiza una mayor fidelidad de la información al respaldar cada respuesta en fragmentos extraídos directamente de los documentos institucionales de la UIDE, lo que reduce drásticamente el riesgo de “alucinaciones” y refuerza la confianza de los usuarios en la exactitud de las respuestas. Además, este esquema facilita la actualización continua del repositorio de conocimiento: siempre que cambien las políticas, los reglamentos o las guías académicas, basta con reindexar los nuevos documentos en FAISS sin necesidad de volver a

entrenar el modelo completo, lo cual ahorra tiempo y recursos computacionales. Por otro lado, aunque RAG introduce cierta complejidad adicional en la infraestructura —ya que requiere un pipeline de indexación y recuperación—, mantiene el uso de un LLM preentrenado para la generación final, con lo cual no se incurre en los elevados costos de GPU o TPU asociados al reentrenamiento periódico que demanda el fine-tuning. Por último, lo bueno de RAG es que puedes agregar información nueva de manera rápida y fácil, lo que lo hace perfecto para una universidad donde constantemente están cambiando cosas y necesitas mantener el chatbot actualizado. En un lugar donde la información se mueve tan rápido, tener algo que se adapte así de bien es lo que hace que el chatbot realmente sirva y se pueda mantener a largo plazo.

1.5. Análisis de Modelos de Lenguaje de Gran Escala (LLM).

1.5.1. ChatGPT (GPT-4 – OpenAI).

GPT-4 es un modelo autoregresivo multimodal que hizo OpenAI, el que vino después de GPT-3.5. Lo que lo hace especial es que puede hacer tareas complicadas de razonamiento, entiende bien el lenguaje natural y genera texto que tiene sentido. Usa una arquitectura Transformer y lo entrenaron con billones de parámetros. También aplicaron técnicas como RLHF (Reinforcement Learning with Human Feedback) para que el modelo entienda mejor lo que las personas quieren que haga (OpenAI, 2023).

Características clave:

- Entrada multimodal (texto e imágenes).
- Alto rendimiento en benchmarks como MMLU y HellaSwag.
- Adaptabilidad a tareas diversas mediante prompts.
- Fuerte enfoque en seguridad y mitigación de sesgos.

Limitaciones:

- Requiere acceso mediante API propietaria.

- Puede presentar “alucinaciones” en temas específicos si no está anclado a datos externos.
- No permite fácil integración con bases de datos vectoriales sin una arquitectura adicional.

1.5.2. *Claude 2 – Anthropic.*

Claude 2 es un LLM que creó Anthropic, y se enfocan mucho en que sea seguro, responsable y esté bien alineado con lo que queremos. Se basa en una variante del modelo Transformer y ha sido entrenado utilizando Constitutional AI, una técnica que reduce la necesidad de supervisión humana directa al momento de refinar los comportamientos del modelo (Anthropic, 2023).

Características clave:

- Alineación ética mejorada mediante técnicas constitucionales.
- Capacidad para manejar entradas largas (hasta 100K tokens).
- Ideal para tareas de redacción, asistencia legal o educativa.

Limitaciones:

- Acceso limitado en ciertas regiones.
- Menor rendimiento que GPT-4 en tareas de razonamiento matemático complejo.
- No está diseñado específicamente para conectarse con pipelines RAG o embeddings externos.

1.5.3. Mistral 7B / Mixtral – Mistral AI.

Mistral es una familia de modelos de código abierto. El más conocido es Mixtral, que funciona con algo llamado Mixture-of-Experts (MoE), básicamente usa solo una parte de sus

parámetros en cada momento en lugar de todos. Lo hicieron pensando en ser eficiente y que rinda bien, y mucha gente lo usa cuando necesita modelos que no pesen tanto y se puedan adaptar fácil.

Características clave:

- Código abierto y uso sin restricciones.
- Eficiencia de cómputo con bajo consumo de GPU.
- Compatible con arquitecturas personalizadas.

Limitaciones:

- No es multimodal.
- Requiere entrenamiento adicional para tareas específicas.
- Carece de soporte nativo para integración con APIs comerciales.

1.5.4. Gemini 2.0 – Google DeepMind.

Gemini 2.0 lo hizo Google DeepMind, y es un modelo multimodal bastante avanzado que está pensado para funcionar bien en sistemas complicados. Puede trabajar con lenguaje, visión y programación, y lo que más llama la atención es que está hecho especialmente para trabajar con embeddings y RAG. Funciona muy bien cuando lo usas en universidades, instituciones y lugares donde necesitas buscar información confiable.

Características clave:

- Multimodal (texto, imagen, código).
- Integración nativa con herramientas como FAISS, BigQuery y Vertex AI.
- Capacidad de trabajar directamente con vectores y embeddings a través de su API.
- Mayor control en generación de respuestas basadas en contexto externo.

Limitaciones:

- Requiere configuración inicial para conectarse a sistemas de vectores.
- Dependencia del ecosistema Google Cloud si se desea integración total.

1.6. Tabla Comparativa de LLMs Analizados.

Tabla 2

Comparación de los LLM's analizados

Modelo	Context Window	Multimodalidad	RAG/Embeddings nativo	Seguridad/Alineación
GPT-4	≈128 000 tokens	Texto + imagen	Parcial (requiere pipeline)	Muy alta
Claude 2	≈100 000 tokens	Texto	No	Muy alta
Llama 2	4 000 – 16 000 tokens	Texto	No (stack propio)	Media
Gemini 2.0	1 000 000 tokens	Texto, imagen, vídeo, audio	Sí (Vertex AI, FAISS)	Alta

Nota. Elaboración propia.

Cuando comparas estos cuatro modelos de lenguaje, se notan diferencias importantes en qué tan grandes son y qué pueden hacer. GPT-4 y Gemini 2.0 trabajan con billones de parámetros, lo que les permite hacer razonamientos complicados y mantener coherencia en respuestas largas. Claude 2, con unos 100 mil millones de parámetros, está en un punto medio entre capacidad y eficiencia. Llama 2, que va de 7 a 70 mil millones de parámetros, es mucho más liviano y lo puedes usar localmente, pero no tiene la fuerza para manejar contextos muy largos.

En cuanto a la ventana de contexto, Gemini 2.0 se lleva las palmas porque puede manejar hasta un millón de tokens de una sola vez, muy por encima de los 128 000 de GPT-4 y los 100 000 de Claude 2. Esto es clave cuando necesitas procesar documentos completos, conversaciones largas o material académico extenso sin tener que cortarlo en pedazos. Llama 2, con entre 4 000 y 16 000 tokens, se queda corto para estas situaciones.

La capacidad multimodal también hace la diferencia: GPT-4 y Gemini 2.0 pueden trabajar con texto e imágenes, pero mientras GPT-4 se queda ahí, Gemini también maneja video y audio. Esto abre muchas posibilidades como analizar clases grabadas, transcribir charlas y procesar diagramas. Claude 2 y Llama 2 solo entienden texto, lo que los limita bastante en universidades donde podrían usar materiales académicos en diferentes formatos.

Lo que realmente marca la diferencia para hacer un chatbot es cómo se conecta con embeddings y RAG. Solo Gemini 2.0 se conecta directamente con motores vectoriales como FAISS y servicios de Vertex AI, permitiendo que cada respuesta se base automáticamente en pedazos de la documentación sin necesidad de programar capas extra o pipelines complicados. Esto hace que desarrollar sea más simple y las respuestas lleguen más rápido. Los otros modelos necesitan software externo o que hagas adaptaciones manuales cada vez.

Por último, aunque GPT-4, Claude 2 y Gemini 2.0 proporcionan APIs propietarias con altos estándares de seguridad y alineación, y Llama 2 cuenta con la ventaja del código abierto, ésta última sacrifica la integración directa con RAG y la multimodalidad. En conjunto, la combinación de contexto masivo, multimodalidad ampliada, integración nativa de embeddings y altos niveles de seguridad consolida a Gemini 2.0 como la opción más robusta y versátil para el chatbot de la Universidad Internacional del Ecuador, campus Loja.

1.7. Preprocesamiento e Indexación de Datos.

Para garantizar la calidad de la recuperación en RAG, es fundamental realizar tokenización, normalización y eliminación de ruido en los datos. Scrapy permitió extraer de manera sistemática las páginas web de la UIDE, transformando tablas, PDF y HTML en texto plano listo para indexar (Puma Quilumba, 2023). Un pipeline robusto debe incluir:

- Segmentación de documentos en fragmentos semánticos (párrrafos, secciones).
- Vectorización con modelos de embeddings (p. ej., sentence-transformers).
- Indexación en FAISS para búsquedas rápidas de vecinos más cercanos.

1.8. Embeddings de oraciones.

Representaciones vectoriales densas que capturan el significado de frases o párrafos completos. Modelos como Sentence-Transformers usan redes siamesas para generar embeddings semánticos que mejoran notablemente la precisión en tareas de búsqueda y agrupamiento de texto (Reimers & Gurevych, 2022).

1.9. Ventana de contexto (Context Window).

Número máximo de tokens que un LLM puede procesar en una sola pasada. La ampliación de esta ventana (por ejemplo, GPT-4 con hasta 128 k tokens) permite mantener coherencia en diálogos largos o documentos extensos (OpenAI, 2023).

1.10. Prompt Engineering.

Disciplina que diseña y refina las instrucciones (prompts) dadas al modelo para guiar la generación. Técnicas como “chain-of-thought” y few-shot prompting mejoran la exactitud y explicabilidad de las respuestas (Wei et al., 2022).

1.11. Evaluación de recuperación.

Conjunto de métricas orientadas a sistemas de recuperación y generación (RAG). Incluye RAGAS (Retrieval-Augmented Generation Average Score), Mean Reciprocal Rank (MRR) y Recall@k para medir la relevancia de los documentos recuperados y su impacto en la generación final (Neupane et al., 2024).

1.12. Seguridad y privacidad en chatbots.

Conjunto de prácticas que protegen datos sensibles: cifrado de extremo a extremo, políticas de retención mínima, anonimización de PII y cumplimiento de normativas como GDPR y LOPD. Imprescindible para resguardar la información estudiantil (Zhang, Lu, & Xu, 2022).

1.13. Sesgos y ética en IA conversacional.

Los LLM pueden reproducir sesgos presentes en sus datos de entrenamiento. Auditorías periódicas, filtrado de contenido y técnicas de debiasing son esenciales para prevenir respuestas discriminatorias o inapropiadas (Sheng, Chang, & Natarajan, 2021).

1.14. Monitoreo y logging.

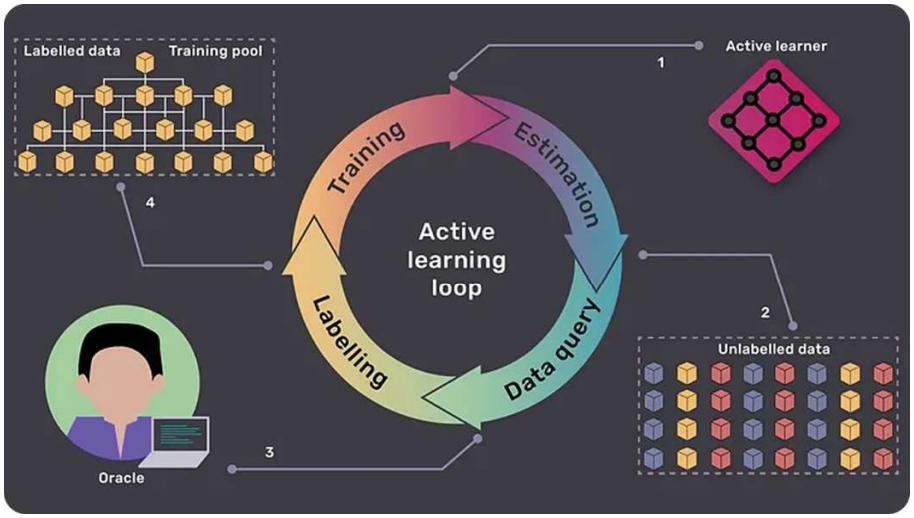
Registro detallado de eventos (consultas, latencias, errores) con herramientas como Prometheus, Grafana o ELK Stack. Facilita la detección temprana de cuellos de botella y la retroalimentación continua al sistema (Burns et al., 2021).

1.15. Aprendizaje continuo y Active Learning.

Estrategias que incorporan feedback de usuarios reales para refinar el modelo. El Active Learning selecciona ejemplos ambiguos para anotación manual, acelerando el proceso de mejora con un menor volumen de datos (Settles, 2021).

Figura 1

Active learning



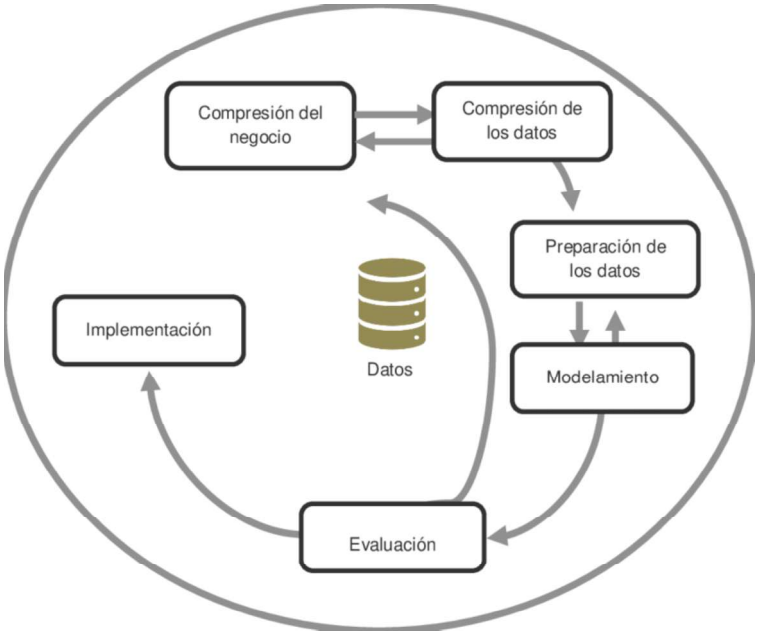
Nota. Elaboración propia.

1.16. Metodología.

1.16.1. Data Mining Methodology for Engineering Applications (DMME).

Figura 2

Proceso de la Metodología DMME



Nota. Adaptado de "Data Mining Methodology for Engineering Applications (DMME)—A Holistic Extension to the CRISP-DM Model" (p. 2407), por H. Wiemer, L. Drowatzky, y S. Ihlenfeldt, 2019, Applied Sciences, 9(12).

Wiemer, Drowatzky e Ihlenfeldt (2019) proponen la metodología DMME como una extensión holística de CRISP-DM, adaptada a los retos de proyectos de ingeniería. Se compone de seis fases:

- **Comprensión del negocio:** definición de objetivos de ingeniería, limitaciones técnicas y recursos disponibles.
- **Análisis de requerimientos de datos:** identificación de fuentes de datos específicas de ingeniería (sensores, CAD, logs de operación) y evaluación de su calidad.

- **Preparación y enriquecimiento:** limpieza de ruido, sincronización de flujos temporales y generación de variables derivadas (por ejemplo, parámetros físicos calculados).
- **Modelado especializado:** uso de algoritmos de machine learning y técnicas de simulación numérica propias de la ingeniería (e.g., modelos de elementos finitos combinados con redes neuronales).
- **Validación multifactor:** evaluación simultánea de métricas técnicas (error de predicción) y criterios de ingeniería (seguridad, normas ISO).
- **Despliegue industrial:** integración en sistemas de control y mantenimiento, con protocolos de actualización y monitorización en tiempo real (Wiemer et al., 2019).

1.16.2. CRISP-ML(Q): Quality-Driven Machine Learning Process Model.

Figura 3

Diagrama de CRISP-ML(Q)

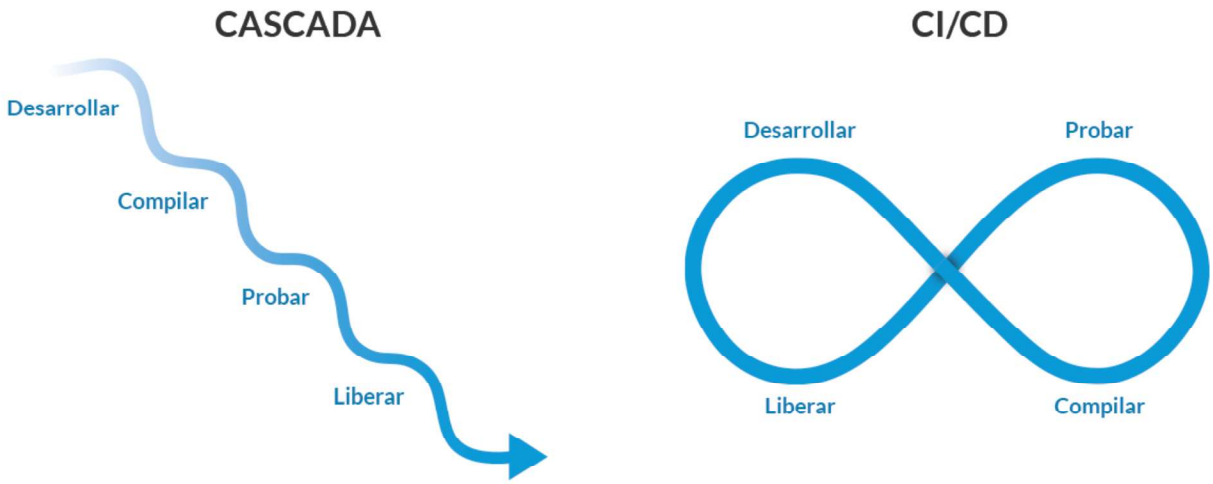
Nota. Adaptado de “CRISP-ML(Q): A Methodical Approach to Quality Assurance and Machine Learning Development” (16 de enero de 2024), por S. Vanga, en Medium.

Studer et al. (2020) presentan CRISP-ML(Q), un proceso en seis fases que adapta CRISP-DM a proyectos de Machine Learning, incorporando aseguramiento de calidad a cada tarea:

- **Definición de alcance y calidad:** unifica negocio y datos en el primer paso, estableciendo métricas de calidad (disponibilidad, completitud, equidad).
- **Diseño de calidad de datos:** contempla validaciones automáticas, detección de sesgos y trazabilidad de origen de datos.
- **Preparación y generación de datasets:** incluye protocolos de versionado y pruebas unitarias sobre transformaciones.
- **Entrenamiento y optimización:** enfatiza tests continuos de regresión sobre métricas de desempeño y fairness.
- **Evaluación de riesgos y pruebas de calidad:** análisis de robustez ante cambios de distribución y verificación de cumplimiento normativo.
- **Despliegue y monitorización continua:** pipelines de CI/CD, alertas de degradación del modelo y reentrenamientos automatizados (Studer et al., 2020)

Figura 4

CI/CD

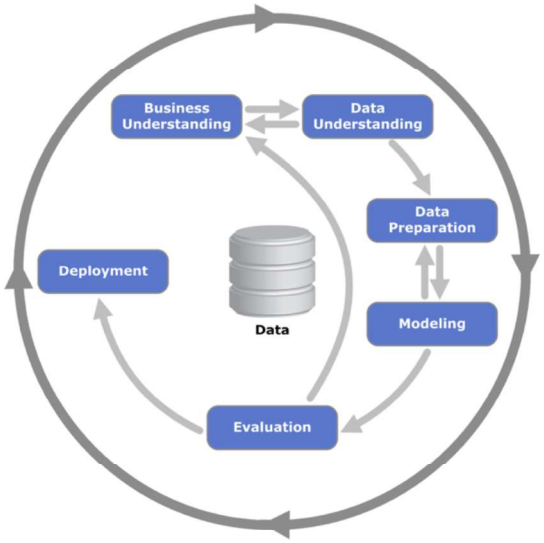


Nota. Elaboración propia.

1.16.3. Revisión Sistemática de CRISP-DM (CENTERIS 2020).

Figura 5

Proceso CRISP-DM



Nota. Adaptado de "CRISP-DM: Cross-Industry Standard Process for Data Mining," por IBM (s. f.)

Schröer, Kruse y Marx Gómez (2020) presentan en el congreso CENTERIS una revisión sistemática de estudios que aplican CRISP-DM, identificando buenas prácticas y áreas de mejora en cada fase:

- **Fase de Business Understanding:** mejor soporte para la definición de KPIs y gestión de stakeholders.
- **Data Understanding:** uso de herramientas de profiling automatizado y análisis de calidad con dashboards.
- **Data Preparation:** plantillas estandarizadas para pipelines ETL y control de versiones de datos.
- **Modeling:** recomendación de frameworks modulares que faciliten la comparación de algoritmos.
- **Evaluation:** integración de métricas de negocio junto a métricas técnicas.

- **Deployment:** mayor énfasis en documentación y plantillas para la transición a producción (Schröer et al., 2020)

1.17. Tabla comparativa de las metodologías estudiadas.

Tabla 3

Comparativa de metodologías adecuadas para el proyecto

Aspecto	DMME	CRISP-ML(Q)	Revisión CRISP-DM
Origen / Publicación	Applied Sciences, 2019	arXiv Mar 2020	CENTERIS 2020
Fases clave	1) Dominio 2) Datos 3) Preparación 4) Modelado 5) Validación 6) Despliegue	1) Alcance/Calidad 2) Calidad 3) Prep 4) Entrenamiento 5) Evaluación 6) Monitorización	1) Bus. 2) Datos 3) Prep 4) Modelado 5) Eval 6) Despliegue
Enfoque principal	Adaptación a proyectos de ingeniería y simulación	Aseguramiento de calidad continuo en ML	Identificación de gaps y mejores prácticas en CRISP-DM
Fortalezas	Enriquecimiento de variables de ingeniería; integración con sistemas de control	Integración CI/CD; tests de regresión; fairness; trazabilidad	Plantillas estandarizadas; énfasis en documentación y KPIs
Limitaciones	Requiere conocimientos de simulación avanzada	Dependencia de pipelines y herramientas de CI/CD	No propone un proceso nuevo, sino mejoras sobre CRISP-DM
Aplicabilidad al chatbot UIDE	Útil para integrar variables extraídas de Scrapy con simulaciones de carga de usuarios en la GUI	Garantiza calidad de datos y modelos LLM durante despliegue y monitorización	Permite identificar brechas en la fase de despliegue e implementación de la GUI

Nota. Elaboración propia.

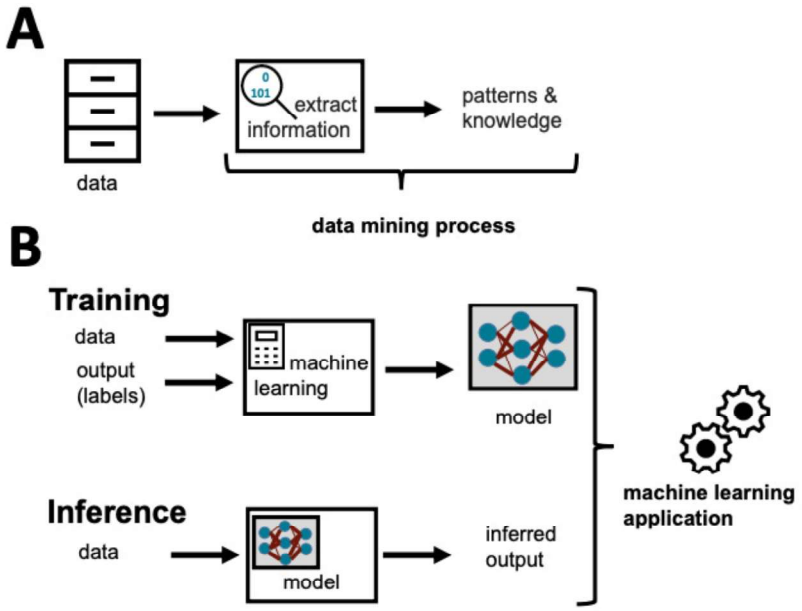
Para el presente proyecto dirigido a la Universidad Internacional del Ecuador, campus Loja, es crítico:

- Asegurar la calidad continua de los datos extraídos (Scrapy, datos físicos) y del modelo de lenguaje durante su ciclo de vida real necesario ante cambios en los sitios web de UIDE o en la información institucional.
- Implementar pipelines de despliegue y monitorización automáticos que detecten degradación de desempeño o sesgos en las respuestas, y permitan reentrenamientos eficientes.
- Trazabilidad y tests de regresión sobre transformaciones y entrenamiento del LLM, garantizando transparencia y reproducibilidad.

CRISP-ML(Q) aborda estos puntos mediante su fase inicial conjunta de negocio y calidad, la incorporación de QA en cada tarea, y la explicitación de despliegue con CI/CD y monitorización. Por ello, frente a DMME (más orientada a ingeniería de variables físicas) y la revisión de CRISP-DM (enfocada en mejoras para CRISP-DM clásico), CRISP-ML(Q) provee el marco más completo para proyectos de LLM que requieren altos estándares de calidad y operación continua.

Figura 6

Proceso de CRISP-ML(Q)



Nota. Elaboración propia.

1.18. Arquitectura Técnica.

1.18.1. Captura de contenido dinámico.

Tabla 4

Comparativa de captura de contenido dinámico: Puppeteer vs. Scrapy

Característica	Puppeteer	Scrapy
Tipo de enfoque	Control de un navegador real (Chrome/Chromium) en modo headless.	Framework de scraping asíncrono puro basado en Twisted.
Renderizado de JavaScript	Total (ejecuta todos los scripts de la página).	Parcial, depende de middlewares o integraciones externas (Splash, Selenium).
Facilidad de configuración	Baja, requiere instalación de binarios de Chrome y gestión de versiones.	Alta, sólo Python y dependencias de Scrapy.
Escalabilidad	Moderada, cada instancia consume memoria de navegador.	Alta, múltiples spiders pueden correr concurrentemente con bajo consumo de memoria.
Uso de recursos	Elevado, por cada pestaña/headless browser.	Bajo, gestión ligera de peticiones HTTP.
Casos de uso	Páginas con heavy JS (SPA), interacciones complejas.	Extracción masiva de sitios estáticos o semi-dinámicos, scraping en paralelo.

Nota. Elaboración propia.

La elección entre Puppeteer y Scrapy depende fundamentalmente de la naturaleza del contenido objetivo. Puppeteer, al controlar un navegador real, garantiza la captura íntegra de cualquier contenido que se cargue dinámicamente mediante JavaScript, eventos DOM o llamadas AJAX complejas. Esta fidelidad es indispensable en sitios modernos como UIDE, donde gran parte de la información puede provenir de frameworks SPA (Single Page Applications). No obstante, su coste en memoria y CPU por instancia lo hace menos adecuado para exploraciones masivas a gran escala. Por otro lado, Scrapy, al estar diseñado para realizar múltiples peticiones concurrentes con un footprint muy ligero, resulta óptimo para rastrear y extraer grandes volúmenes de páginas HTML estáticas o ligeramente dinámicas.

1.19. Extracción y procesamiento de documentos.

Tabla 5

Comparativa de extracción y procesamiento de documentos: pdf-parse vs. Tesseract.js

Característica	pdf-parse	Tesseract.js
Fuente de texto	Texto embebido en el PDF (text layer).	OCR de imágenes incluidas en el PDF.
Precisión en texto digital	Alta, recupera exactamente lo embebido.	N/A, no está pensado para texto embebido.
Necesidad de OCR	Ninguna si el PDF contiene capa de texto.	Necesaria para PDFs escaneados o imágenes embebidas.
Velocidad de procesamiento	Rápida, procesado binario directo.	Lenta, depende de la complejidad de las imágenes y el motor JavaScript.
Dependencias	Sólo Node.js y libvips opcional.	Tesseract + librerías de entrenamiento de idiomas.
Casos de uso	Extracción de catálogos digitalizados, documentos nativamente generados.	Documentos escaneados, imágenes de texto antiguo sin capa de texto.

Nota. Elaboración propia.

Para asegurar una cobertura completa de todos los tipos de PDF encontrados en la UIDE, fue necesario combinar pdf-parse y Tesseract.js. pdf-parse extrae con precisión el texto embebido, lo cual es inmediato y mantiene la estructura de párrafos y tabulaciones originales. Sin embargo, muchos documentos institucionales antiguos están escaneados sin capa de texto, y ahí entra Tesseract.js: a través de OCR se recupera la información presente únicamente como píxeles. Aunque esta etapa es mucho más lenta y puede introducir errores en el reconocimiento de caracteres, garantiza que ningún contenido quede fuera del pipeline de indexación. La combinación de ambas herramientas robusteció tu repositorio de datos y mejoró la confianza en la integridad de la información indexada.

1.20. Fragmentos de código y su explicación en el desarrollo del scraper.

Tabla 6

Fragmentos de código y su explicación en el desarrollo del scraper

Fragmento de código	Explicación
const puppeteer = require('puppeteer');	Se importa la librería Puppeteer, la cual permite automatizar un navegador para extraer información de páginas web.
class ExhaustiveUIDEScraper { ... }	Definición de la clase principal que contiene toda la lógica del scraper..
this.config = { maxDepth: options.maxDepth || 50, ... }	Se establece la configuración general del scraper, como la profundidad máxima de navegación, el número de páginas a procesar, archivos a descargar y otros parámetros de rendimiento.
async initializeDirectories() { ... }	Método encargado de crear las carpetas necesarias para almacenar los resultados, así como inicializar archivos de consolidación y registros de progreso.
async start() { ... }	Método principal que ejecuta el flujo completo: inicializa directorios, abre el navegador, descubre URLs, procesa el contenido y genera reportes finales.
async massiveUrlDiscovery() { ... }	Función para la primera fase del scraping: descubre masivamente URLs a partir de sitemaps y enlaces de la página principal.
async exhaustiveProcessing(browser) { ... }	Método que recorre todas las URLs descubiertas, procesando exhaustivamente cada página para extraer información, archivos y nuevos enlaces.
async extractAllPageData(page, url) { ... }	Función que obtiene todos los datos posibles de una página.
async exhaustiveFileExtraction(page, currentUrl) { ... }	Método que localiza y descarga archivos presentes en cada página (PDF, Word, Excel, etc.), además de intentar extraer su contenido textual.
async generateComprehensiveReports() { ... }	Método encargado de generar un reporte final con estadísticas de la extracción.

Nota. Elaboración propia.

La tabla anterior presenta los fragmentos más relevantes del código que conforma el scraper desarrollado. Cada fragmento se acompaña de una explicación que detalla su propósito y funcionalidad dentro de la arquitectura general de la aplicación. De esta manera, se facilita la comprensión del flujo lógico del sistema, desde la configuración inicial y la creación de directorios hasta el descubrimiento masivo de URLs, la extracción exhaustiva de datos y archivos, y la generación de reportes consolidados. Esta organización permite evidenciar cómo las distintas secciones del código interactúan de manera articulada para lograr un proceso automatizado, escalable y eficiente de recolección de información.

1.21. Almacenamiento y recuperación de vectores.

Tabla 7

Comparativa de almacenamiento y recuperación de vectores: FAISS vs. Elasticsearch.

Característica	FAISS	Elasticsearch
Arquitectura interna	C++ nativo con bindings para Python; índices IVF-PQ, HNSW, Flat.	Java sobre Lucene; índices invertidos y plugin de vectores.
Escalabilidad	Excelente para millones–miles de millones de vectores en un solo nodo con GPU opcional.	Distribuida por diseño, permite sharding y réplicas en un cluster.
Latencia de consulta	Milisegundos a microsegundos en vectores de hasta 1B.	Decenas de milisegundos en clusters medianos.
Facilidad de integración	Sencilla en Python; API limitada a consulta y construcción de índices.	API RESTful robusta, soporte en múltiples lenguajes y ecosistema de plugins.
Casos de uso	Recuperación rápida de embeddings para RAG en un único servicio.	Búsqueda híbrida (texto + vectores), monitoreo y logging integrados.

Nota. Elaboración propia.

FAISS y Elasticsearch cubren escenarios diferentes dentro de un pipeline RAG. FAISS, optimizado para cálculos vectoriales de alta dimensión, brilla cuando se requiere recuperar miles de documentos por consulta en submilisegundos. Su integración directa con Python y frameworks de embeddings hace que sea la opción preferida para prototipos y despliegues de un solo nodo. En cambio, Elasticsearch extiende esta capacidad con un ecosistema de búsqueda textual tradicional,

permitiendo combinar consultas semánticas y de palabras clave en un mismo cluster distribuido. Para tu chatbot, FAISS resultó más sencillo de desplegar y suficiente para la carga estimada de consultas, mientras que Elasticsearch queda como opción futura para un despliegue multiusario georeplicado.

1.22. Streaming de respuestas en tiempo real.

Tabla 8

Comparativa de streaming en tiempo real: Server-Sent Events vs. WebSockets

Característica	SSE (Server-Sent Events)	WebSockets
Direccionalidad	Unidireccional (servidor → cliente).	Bidireccional (cliente ↔ servidor).
Manejo de conexiones	Basado en HTTP/1.1; reconexión automática.	Protocolo dedicado; requiere handshake WebSocket sobre HTTP.
Complejidad	Baja, sólo endpoints HTTP tradicionales.	Alta, necesita gestionar estado de conexión y protocolos personalizados.
Uso de recursos	Ligero, reutiliza conexiones HTTP.	Moderado, mantiene sockets abiertos en ambos extremos.
Casos de uso	Notificaciones de eventos, flujos de texto parcial (streaming de tokens).	Chats interactivos complejos, juegos en tiempo real, colaboración en vivo.

Nota. Elaboración propia.

En escenarios de chat generativo, las respuestas suelen llegar en forma de tokens parciales que el usuario aprecia ver en tiempo real. SSE cumple muy bien esta función al enviar un flujo continuo de datos sobre la conexión HTTP existente, simplificando la implementación en entornos sin estado y reduciendo la sobrecarga de infraestructura (Christie, 2024). Aunque WebSockets ofrece comunicación bidireccional, en un chatbot RAG el canal de cliente→servidor se limita a enviar la consulta in¡”ial,’por lo que SSE resultó ser la opción más eficiente y sencilla para tu frontend Next.js, manteniendo la latencia al mínimo y evitando la complejidad de gestionar estados de socket en el servidor.

1.23. Django 5.1.4.

Django es un framework de desarrollo web de alto nivel para Python que implementa el patrón Model–Template–View (MTV). En su versión 5.1.4, Django ha reforzado la compatibilidad con ASGI, lo que facilita la ejecución de código asíncrono y la integración con WebSockets y canales, sin renunciar a su conocido sistema de ORM, gestión de migraciones y panel de administración automático. Gracias a su filosofía “baterías incluidas”, proporciona de forma nativa herramientas para autenticación, manejo de sesiones, enrutamiento de URLs y validación de formularios, acelerando significativamente el desarrollo de APIs seguras y escalables. La comunidad de djangueros y documentación oficial garantizan buenas prácticas y actualizaciones constantes, vitales para mantener la estabilidad de un servicio de chatbot en producción (Django Software Foundation, 2024).

1.24. Django REST Framework (DRF) 3.15.2.

DRF le da a Django todo lo que necesitas para hacer APIs RESTful de forma rápida y fácil de mantener. Lo que más llama la atención es su sistema de serializadores, que básicamente convierten los modelos de Django a JSON de manera automática, validan los datos que llegan y manejan las relaciones complicadas entre objetos sin que te duela la cabeza. Las vistas basadas en clases (CBV) de DRF hacen que sea súper fácil abstraer cosas comunes como listar, crear, actualizar y borrar, y puedes agregar mixins para personalizar permisos y autenticación como quieras.

Desde la versión 3.12, DRF empezó a soportar vistas asíncronas de manera nativa, lo que hace que todo vaya más rápido cuando tienes operaciones pesadas de I/O. En 2021 mejoraron bastante la generación automática de esquemas OpenAPI, algo que hace mucho más simple documentar y probar la API (Christie, 2021).

1.25. djangorestframework-simplejwt 5.4.0.

Simple JWT te da todo un sistema de autenticación con JSON Web Tokens (JWT) que sigue las mejores prácticas de OAuth 2.1. Lo bueno es que puede rotar refresh tokens y tiene una lista negra (blacklist) que te permite invalidar tokens que se hayan comprometido sin tener que guardar sesiones en el servidor. Su configuración admite definir tiempos de expiración granulares para access y refresh tokens, elegir algoritmos de firma (HS256, RS256) y customizar reclamaciones (claims) según requisitos de seguridad institucional. Al no consumir estado en el backend, facilita el despliegue horizontal y evita cuellos de botella de sesión, lo que resulta especialmente útil en arquitecturas de chatbot donde cada consulta se autentica de manera independiente (Jones, Bradley, & Sakimura, 2021).

1.26. LangChain 0.3.14.

LangChain es un framework modular diseñado para orquestar aplicaciones basadas en grandes modelos de lenguaje (LLM). Sus "chains" te permiten conectar pasos como generar embeddings, recuperar documentos (RAG), ejecutar prompts y procesar las respuestas. Esta idea de trabajar con componentes hace que puedas cambiar proveedores de embeddings (OpenAI, Cohere, HuggingFace) o motores de búsqueda (FAISS, Elasticsearch) sin tener que tocar el resto del sistema. LangChain también trae herramientas para evaluar automáticamente qué tan buenas son las respuestas y hacer experimentos A/B con prompts, lo que hace más rápido mejorar los modelos y la experiencia del usuario. En las versiones más nuevas agregaron soporte para pipelines asíncronos y orquestación con flujos de trabajo, convirtiéndolo en el centro de tu backend RAG (Mann & Wang, 2023).

1.27. Google Cloud Firestore 2.20.0.

Firestore es una base de datos NoSQL de documentos que maneja Google Cloud. Crece horizontalmente y tiene replicación en varias regiones con poca latencia para leer y escribir. Sus colecciones y documentos te dejan organizar los fragmentos indexados y los logs de interacción del chatbot de manera que tenga sentido, mientras que las transacciones atómicas y las reglas de

seguridad basadas en Firebase Authentication te aseguran que todo sea consistente y puedas controlar quién accede a cada documento.

En la versión 2.20 mejoraron bastante el rendimiento de consultas compuestas y cómo se integra con Cloud Functions, lo que hace más fácil crear triggers que pueden, por ejemplo, re-indexar documentos nuevos automáticamente o mandarte alertas cuando detecten patrones de uso raros. Todo esto te da una capa de gestión de datos confiable y que se puede extender para tu arquitectura RAG (Google Cloud, 2022).

CAPITULO II: ANÁLISIS Y DISEÑO

2.1. Analizando el problema.

2.1.1. Contexto Institucional.

La Universidad Internacional del Ecuador (UIDE), campus Loja, es parte de una institución educativa reconocida que atiende a estudiantes muy diversos y cada vez son más. Como universidad, tiene que manejar muchísima información: trámites administrativos, carreras, calendarios, reglamentos y políticas específicas del campus. Toda esta información tiene que estar disponible cuando la gente la necesite y ser correcta para estudiantes actuales, los que quieren entrar, profesores y personal administrativo.

El campus Loja tiene sus propios problemas por dónde está ubicado y el tipo de estudiantes que tiene. La gente siempre está preguntando cosas diferentes: sobre admisiones, qué necesitan para estudiar, trámites, qué servicios hay, horarios, cuánto cuesta estudiar y un montón de procedimientos que necesitan respuestas rápidas y correctas.

2.1.2. Identificación de la Problemática.

Como está la situación Actual.

Actualmente, en la UIDE campus Loja manejan la información y atienden las consultas de varias maneras tradicionales, pero que tienen bastantes problemas:

Canales de atención que existen:

- **Atención presencial:** Ventanillas de servicio al estudiante con horarios limitados (8:00 - 17:00, lunes a viernes)
- **Comunicación telefónica:** Call center con capacidad limitada de agentes.
- **Correo electrónico institucional:** Respuestas con demoras de 24-48 horas.
- **Sitio web institucional:** Información estática que requiere navegación compleja.
- **Redes sociales:** Atención reactiva sin capacidad de respuesta inmediata.

Problemáticas Identificadas:

Sobrecarga del Personal Administrativo: El análisis inicial muestra que más o menos el 70% de las consultas diarias son las mismas preguntas de siempre: qué necesito para entrar, cómo me matriculo, cuánto cuesta estudiar, cuándo son las fechas importantes, y trámites básicos. Esto hace que el personal administrativo se la pase respondiendo lo mismo todo el día, cuando podrían estar haciendo cosas más importantes.

Limitaciones de Horario: La atención normal solo funciona en horario de oficina, lo que complica las cosas para estudiantes que trabajan, viven lejos en el campo, o que tienen clases justo cuando están abiertos los departamentos administrativos. Esta limitación es particularmente crítica durante períodos de alta demanda como procesos de admisión, períodos de matrícula, y fechas límite para trámites académicos.

Inconsistencia en la Información: La dispersión de información en múltiples plataformas y la dependencia de diferentes personas para proporcionar respuestas genera inconsistencias en la información entregada. Un mismo procedimiento puede ser explicado de manera diferente por distintos funcionarios, lo que crea confusión y potenciales errores en los procesos estudiantiles.

Accesibilidad y Experiencia de Usuario: Los estudiantes, especialmente los de nueva generación (nativos digitales), esperan respuestas inmediatas y acceso 24/7 a la información institucional. La navegación compleja del sitio web actual y la falta de herramientas de búsqueda inteligente dificultan el acceso rápido a información específica.

Escalabilidad Limitada: Durante períodos pico (inicio de semestres, procesos de admisión, fechas de pago), la demanda de información supera significativamente la capacidad de respuesta del personal, generando colas de espera, saturación telefónica y demoras en respuestas por correo electrónico.

Impacto de la Problemática

En los Estudiantes:

- Frustración por demoras en obtener información crítica.
- Pérdida de tiempo en desplazamientos innecesarios al campus.
- Riesgo de pérdida de fechas límite importantes.
- Experiencia de usuario deficiente que afecta la percepción institucional.

En el Personal Administrativo:

- Sobrecarga laboral por consultas repetitivas.
- Reducción del tiempo disponible para tareas estratégicas.
- Estrés durante períodos de alta demanda.
- Dificultad para mantener consistencia en las respuestas.

En la Institución:

- Ineficiencias operativas que incrementan costos.
- Riesgo de errores en información crítica.
- Imagen institucional afectada por calidad de servicio.
- Dificultad para escalar servicios durante crecimiento estudiantil.

2.1.3. Justificación de la Solución Tecnológica

Necesidad de Automatización Inteligente

La implementación de un chatbot basado en tecnología de Large Language Models (LLM) con arquitectura RAG (Retrieval-Augmented Generation) se presenta como una solución integral que aborda simultáneamente múltiples aspectos de la problemática identificada. Esta tecnología permite crear un asistente virtual capaz de:

- **Procesamiento de Lenguaje Natural:** Interpretar consultas en español coloquial sin requerir comandos específicos.
- **Recuperación Inteligente:** Acceder a la base de conocimiento institucional de manera precisa.
- **Respuestas Contextuales:** Generar respuestas naturales y específicas para cada consulta.
- **Disponibilidad Continua:** Operar 24/7 sin limitaciones de horario.
- **Escalabilidad:** Atender múltiples consultas simultáneamente sin degradación del servicio.

Ventajas de la Arquitectura RAG

La elección de RAG sobre otras aproximaciones tecnológicas se fundamenta en:

Precisión y Veracidad: RAG ancla las respuestas en documentos verificados de la institución, reduciendo significativamente el riesgo de "alucinaciones" típicas de los LLM y garantizando que la información proporcionada sea oficial y actualizada.

Actualización Dinámica: Permite incorporar cambios en políticas, procedimientos o información institucional sin necesidad de reentrenar el modelo completo, simplemente actualizando la base de documentos indexada.

Trazabilidad: Cada respuesta puede ser rastreada a su fuente documental original, facilitando la auditoría y verificación de la información proporcionada.

Eficiencia de Recursos: Utiliza un LLM preentrenado para generación, evitando los costos computacionales asociados con el fine-tuning completo de modelos grandes.

2.1.4. Alcance del Problema

Usuarios Primarios:

- **Estudiantes Actuales:** Pregrado del campus Loja.
- **Estudiantes Prospectivos:** Personas interesadas en programas académicos.

- **Docentes:** Personal académico que requiere información administrativa.
- **Personal Administrativo:** Para consultas internas y verificación de procedimientos.

Usuarios Secundarios:

- **Padres de Familia:** Información sobre procedimientos y requisitos.
- **Empresas Colaboradoras:** Información sobre programas de vinculación.

Dominios de Información

El chatbot debe manejar información correspondiente a:

Área Académica:

- Programas de pregrado y posgrado disponibles.
- Mallas curriculares y pensum académico.
- Requisitos de admisión y proceso de postulación.
- Calendario académico y fechas importantes.
- Procedimientos de matrícula y registro de materias.

Área Administrativa:

- Trámites estudiantiles (certificados, constancias, cartas).
- Información financiera (aranceles, formas de pago, becas).
- Procedimientos de graduación y titulación.
- Servicios complementarios (biblioteca, laboratorios, cafetería).

Área Normativa:

- Reglamentos académicos y disciplinarios.
- Políticas institucionales específicas.
- Derechos y deberes estudiantiles.
- Procesos de apelación y recursos.

Limitaciones Identificadas

Técnicas:

- Dependencia de la calidad y actualización de la información fuente.
- Limitaciones del procesamiento de consultas altamente específicas o personalizadas.
- Necesidad de escalamiento a atención humana para casos complejos.

Organizacionales:

- Requerimiento de procesos de actualización coordinados con las diferentes áreas.
- Necesidad de capacitación para personal que supervisará el sistema.
- Establecimiento de protocolos de escalamiento y seguimiento.

Éticas y Legales:

- Protección de datos personales según normativas vigentes.
- Manejo responsable de información estudiantil sensible.
- Transparencia en el uso de inteligencia artificial para atención al usuario.

2.2. Requerimientos.

La fase de levantamiento de requerimientos constituye una etapa clave dentro del análisis, pues define las funcionalidades esenciales que debe cubrir el sistema para responder eficazmente a las necesidades informativas de la comunidad universitaria. El enfoque utilizado se basó en entrevistas semiestructuradas con coordinadores académicos de cada una de las carreras, complementado con una entrevista especializada con el departamento de Marketing para definir los requerimientos técnicos y funcionales del sistema.

2.2.1. Fuentes de Recolección de Información.

Fase 1: Identificación de Consultas Frecuentes.

La primera fase del levantamiento se realizó a través de entrevistas dirigidas con los coordinadores académicos de cada una de las carreras ofertadas en el campus Loja de la UIDE. Esta metodología permitió obtener una perspectiva directa y autorizada sobre las consultas más frecuentes que reciben de parte de los estudiantes, así como identificar los principales puntos de fricción en los procesos de comunicación institucional.

Coordinadores que formaron parte de las entrevistas:

- **Msc. Mercy Namicela - Coordinadora de Negocios Internacionales**: Proporcionó insights sobre consultas relacionadas con programas de intercambio, comercio internacional y oportunidades laborales globales.
- **Ing. Lorena Conde - Coordinadora de Ingeniería en Tecnologías de la Información:** Aportó perspectiva sobre consultas técnicas, laboratorios especializados, certificaciones y proyectos de desarrollo tecnológico.
- **Msc. Viviana Torres - Coordinadora de Administración de Empresas**: Contribuyó con información sobre consultas administrativas, emprendimiento, prácticas profesionales y vinculación empresarial.
- **Psic. Domenica Burneo - Coordinadora de Psicología:** Facilitó comprensión sobre consultas de orientación académica, prácticas preprofesionales, procesos de titulación y áreas de especialización.
- **Ab. Raquel Veintimilla - Coordinadora de Derecho:** me contó sobre las preguntas que recibe sobre mallas académicas y cómo funcionan las cosas por dentro.

Fase 2: Definición de Requerimientos Técnicos.

La segunda fase se centró en una entrevista especializada con el departamento de Marketing institucional para establecer los requerimientos funcionales, no funcionales y de experiencia de usuario del sistema de chatbot.

Stakeholder Clave:

Lcda. Janine Rojas - Departamento de Marketing, se encarga de definir qué necesita el sistema desde el punto de vista de la universidad, incluyendo objetivos de comunicación, imagen corporativa, cómo medir si está funcionando bien y cómo conectarlo con las estrategias de marketing digital de la universidad.

¿Cómo se realizaron las entrevistas?

Diseño del Instrumento de Entrevista

Para conseguir la información que necesitábamos, armamos entrevistas semiestructuradas que mezclaban preguntas generales para todos con preguntas específicas para los coordinadores académicos. De esta manera pudimos entender tanto las necesidades de comunicación de toda la universidad como los detalles particulares de cada carrera. Los detalles completos de la entrevista están en el Anexo A.

Entrevista con el Departamento de Marketing

Para saber qué necesitábamos técnicamente y cómo debía funcionar el sistema, hicimos una entrevista específica con Janine Rojas del departamento de Marketing, centrándonos en estos puntos:

Objetivos de la Universidad y Experiencia del Usuario:

- Definir qué quiere lograr la universidad con su comunicación digital.
- Características de diseño que muestren la identidad de la UIDE.

Lo que necesita el Sistema:

- Funcionalidades específicas requeridas según necesidades institucionales.
- Diferenciación de información para estudiantes actuales vs. prospectivos.
- Protocolos de escalamiento y manejo de consultas complejas.
- Características de personalización y adaptabilidad del sistema.

¿Qué resultados se obtuvieron en las entrevistas?

Resultados de Preguntas Generales

Cuando analicé las respuestas de las preguntas que les hice a todos los coordinadores académicos, encontré que todos tenían necesidades de comunicación parecidas:

¿Qué Preguntas Llegan Más Seguido?

Los cinco coordinadores estuvieron de acuerdo en que las consultas que más les llegan tienen que ver con:

- Información sobre requisitos de admisión y procesos de matrícula.
- Dudas sobre horarios de clases y modalidades de estudio disponibles.
- Consultas sobre aranceles, formas de pago y opciones de becas.
- Información sobre calendario académico y fechas importantes.
- Procedimientos para trámites estudiantiles (certificados, constancias).

Canales de Comunicación Actuales

En las entrevistas identifiqué los siguientes canales que usan los estudiantes:

- Consultas presenciales en las coordinaciones académicas (60% del tiempo).
- Comunicación vía correo electrónico institucional (25% del tiempo).
- Consultas telefónicas al campus (10% del tiempo).
- Redes sociales oficiales de la universidad (5% del tiempo).

Información Indispensable para el Chatbot

Según lo que me dijeron los coordinadores, el chatbot debe incluir:

- Información académica actualizada por de cada escuela.
- Procedimientos administrativos paso a paso.
- Directorio de contactos de coordinaciones y departamentos.

- Calendario académico con fechas relevantes.
- Información sobre servicios estudiantiles disponibles.

Evaluación de Herramientas Actuales

Los coordinadores evaluaron las herramientas actuales como "parcialmente efectivas" debido a:

- Limitaciones de horario para atención presencial.
- Demoras en respuestas por correo electrónico.
- Dificultad para mantener información actualizada en múltiples canales.
- Sobrecarga de consultas repetitivas que consumen tiempo administrativo.

Recomendaciones para el Chatbot

- Disponibilidad 24/7 para consultas básicas.
- Interfaz intuitiva y de fácil navegación.
- Capacidad de escalamiento a atención humana cuando sea necesario.
- Actualización automática de información institucional mediante archivos.

Requerimientos Institucionales del Departamento de Marketing

Definición de Requerimientos con Janine Rojas

La entrevista realizada con Janine Rojas del departamento de Marketing se enfocó en establecer los requerimientos institucionales y técnicos del sistema desde la perspectiva de comunicación corporativa y experiencia de usuario. Los principales hallazgos se categorizaron en las siguientes áreas:

Requerimientos de Imagen e Identidad Corporativa:

- El chatbot debe reflejar los valores institucionales de excelencia académica, innovación tecnológica y compromiso social de la UIDE.

- Debe mantener un tono de comunicación profesional pero cercano, adaptado al perfil de estudiantes jóvenes y adultos jóvenes.
- Las respuestas deben ser consistentes con la línea editorial y comunicacional de la universidad.
- Debe incluir elementos visuales que refuercen la identidad corporativa (colores, logo, tipografía institucional).

Requerimientos de Experiencia de Usuario:

- Interfaz intuitiva y moderna que refleje la calidad tecnológica de la institución.
- Tiempo de respuesta optimizado (máximo 5 segundos) para mantener el engagement del usuario.
- Capacidad de procesamiento de consultas en español coloquial y formal.
- Funcionalidad responsive que se adapte a diferentes dispositivos (móvil, tablet, desktop).

Requerimientos de Calidad de Servicio:

- Precisión mínima del 85% en respuestas sobre información institucional oficial.
- Sistema de escalamiento automático a personal humano para consultas complejas.
- Registro y análisis de todas las interacciones para mejora continua.
- Cumplimiento de normativas de protección de datos personales.

Este análisis integral fue determinante para establecer el alcance funcional del sistema y priorizar el desarrollo de características como el procesamiento de lenguaje natural contextualizado, la recuperación inteligente de información especializada por carrera, la generación de respuestas que mantengan la identidad institucional, y la implementación de métricas que permitan evaluar tanto la satisfacción estudiantil como el impacto en los objetivos de comunicación institucional.

El levantamiento de requerimientos se basó entonces en una combinación de necesidades académicas específicas que me dijeron los coordinadores y los objetivos estratégicos institucionales que definió el área de Marketing. Esto me dio una base sólida para desarrollar un

sistema de chatbot que realmente responda a todo lo que necesita la comunidad universitaria de la UIDE campus Loja.

2.2.2. Requerimientos Funcionales.

Tabla 9

Requerimiento funcional 1

Número de requisito	RF01
Nombre de requisito	El sistema debe permitir login con correo y clave o con Gmail
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 10

Requerimiento funcional 2

Número de requisito	RF02
Nombre de requisito	El sistema debe permitir al administrador visualizar diagrama de pastel con estadísticas de Like y Dislike
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 11

Requerimiento funcional 3

Número de requisito	RF03
Nombre de requisito	El sistema debe permitir guardar cada conversación del usuario
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 12

Requerimiento funcional 4

Número de requisito	RF04
Nombre de requisito	El sistema debe permitir crear cuenta con nombre, apellido, correo y clave
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 13

Requerimiento funcional 5

Número de requisito	RF05
Nombre de requisito	El sistema debe permitir la recuperación de contraseña
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 14

Requerimiento funcional 6

Número de requisito	RF06
Nombre de requisito	El sistema debe permitir al chatbot aprender a través de archivos PDF cargados desde el panel de administrador
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 15

Requerimiento funcional 7

Número de requisito	RF07
Nombre de requisito	El sistema debe permitir al administrador maestro crear más administradores
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 16

Requerimiento funcional 8

Número de requisito	RF08
Nombre de requisito	El sistema debe permitir cerrar sesión
Tipo	Requisito
Fuente del requisito	Documento base
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

2.2.3. Requerimientos No Funcionales.

Tabla 17

Requerimiento no funcional 1

Número de requisito	RNF01
Nombre de requisito	El sistema debe garantizar un tiempo de respuesta inferior a 2 segundos para las consultas más comunes.
Tipo	Requisito No Funcional
Fuente del requisito	Buenas prácticas de usabilidad
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 18

Requerimiento no funcional 2

Número de requisito	RNF02
Nombre de requisito	El sistema debe estar disponible al menos el 99% del tiempo anual (alta disponibilidad).
Tipo	Requisito No Funcional
Fuente del requisito	Buenas prácticas de confiabilidad
Prioridad del requisito	Alta/Esencial

Nota. Elaboración propia.

Tabla 19

Requerimiento no funcional 3

Número de requisito	RNF03
Nombre de requisito	La interfaz del sistema debe ser intuitiva y usable.
Tipo	Requisito No Funcional
Fuente del requisito	Buenas prácticas de usabilidad.
Prioridad del requisito	Media

Nota. Elaboración propia.

2.3. Metodología empleada: crisp-ml(q).

El presente proyecto se desarrolló bajo la metodología CRISP-ML(Q) (Cross-Industry Standard Process for Machine Learning with Quality assurance), una evolución especializada de CRISP-DM específicamente adaptada para proyectos de aprendizaje automático y sistemas basados en inteligencia artificial. Esta metodología resulta especialmente pertinente en contextos donde el desarrollo del sistema requiere no solo de programación y diseño, sino de un proceso analítico basado en datos y modelos de lenguaje, desde su comprensión hasta su validación en escenarios reales con garantías de calidad continua.

2.3.1. Justificación del Uso de CRISP-ML(Q) en Proyectos de IA Conversacional.

Elegí CRISP-ML(Q) como metodología para este proyecto porque se adapta perfectamente a los ciclos de desarrollo de sistemas de IA conversacional que usan Large Language Models (LLM) y arquitecturas RAG (Retrieval-Augmented Generation). Las metodologías tradicionales de ingeniería de software no entienden bien el problema desde el análisis de datos y procesamiento de lenguaje natural, y CRISP-ML(Q) sí me permite hacer esto bien para entrenar, ajustar y validar modelos conversacionales.

Esta metodología también conecta de manera natural las etapas de identificar el problema de comunicación, recopilar y preparar los datos de la universidad, aplicar modelos de lenguaje avanzados (como Gemini 2.0) y evaluar constantemente qué tan buenas son las respuestas. Como el sistema tiene que procesar preguntas en lenguaje natural y buscar información precisa en repositorios de documentos, CRISP-ML(Q) me da el marco que mejor garantiza que todo sea robusto, adaptable y técnicamente confiable en el contexto universitario.

2.3.2. Fases del Modelo Aplicadas al Sistema de Chatbot RAG.

La metodología CRISP-ML(Q) consta de seis fases integradas que fueron adaptadas al desarrollo del chatbot institucional de la UIDE campus Loja, como se detalla aa continuación:

Tabla 20

Fases aplicadas de la metodología CRISP-ML(Q)

Fase CRISP-ML(Q)	Aplicación en el Proyecto de Chatbot UIDE
Business & Data Understanding	Identificación del problema comunicacional con base en entrevistas a directores de carrera y análisis de consultas frecuentes estudiantiles. Definición de objetivos institucionales de mejora en atención al usuario.
Data Engineering	Extracción sistemática de contenido del sitio web UIDE mediante Scrapy, procesamiento de documentos PDF institucionales con pdf-parse y Tesseract.js, estructuración de información para indexación vectorial.
ML Engineering	Implementación de pipeline RAG con Sentence-Transformers para embeddings, indexación en FAISS, integración con Gemini 2.0 para generación de respuestas, desarrollo de sistema de prompt engineering contextual.
ML Operations & Monitoring	Despliegue del sistema en arquitectura Django + Next.js, implementación de métricas de calidad (RAG), monitoreo de satisfacción de usuario mediante sistema like/dislike, registro de conversaciones para análisis continuo.
Quality Assurance	Validación de respuestas contra documentación oficial, pruebas de coherencia y relevancia, evaluación de experiencia de usuario mediante métricas SUS, análisis de casos edge y manejo de consultas fuera de dominio.
Governance & Communication	Establecimiento de protocolos de actualización de contenido, definición de políticas de privacidad para datos estudiantiles, documentación técnica completa, capacitación de administradores del sistema.

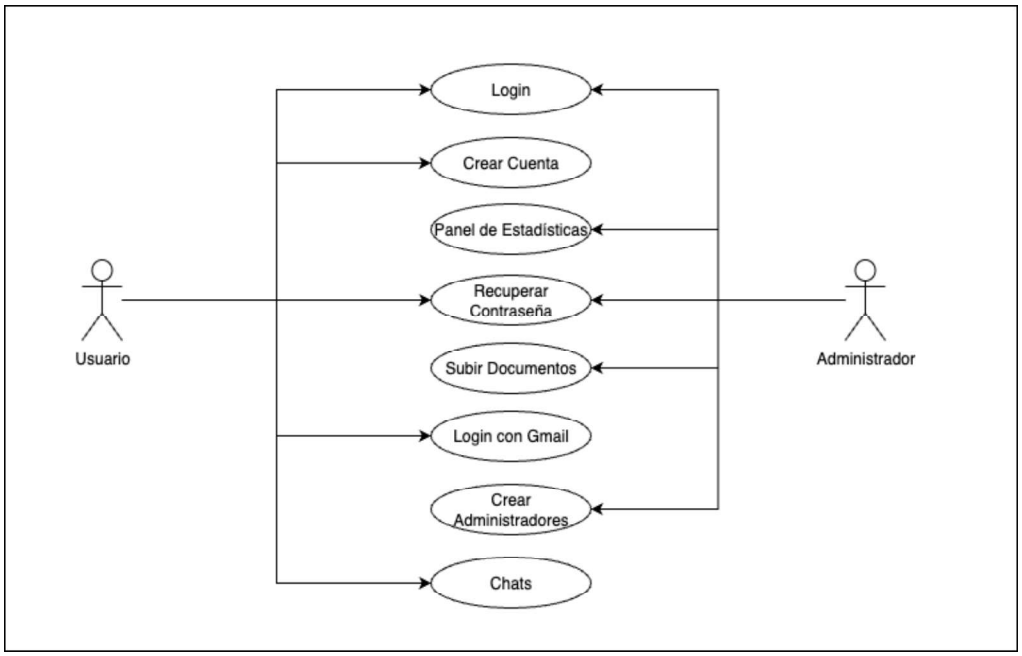
Nota. Elaboración propia.

Esta estructura permitió avanzar de forma sistemática desde la definición del problema comunicacional hasta la entrega de una solución funcional validada. Asimismo, la naturaleza iterativa del modelo facilitó ajustes continuos durante las etapas de desarrollo y pruebas, optimizando el desempeño del sistema en condiciones reales de consulta estudiantil.

2.4. Caso de Uso.

Figura 7

Caso de uso del sistema en general



Nota. Elaboración propia.

2.4.1. Descripción del Diagrama de Casos de Uso.

El diagrama de casos de uso del sistema muestra las interacciones entre dos actores principales y los diferentes casos de uso que pueden ejecutar. El sistema está diseñado para atender tanto a usuarios finales como a administradores, cada uno con diferentes niveles de acceso y funcionalidades específicas.

Actores del Sistema

Actor Primario: Usuario

- Representa a estudiantes actuales, prospectivos, docentes y personal de la UIDE campus Loja.

- Tiene acceso a funcionalidades básicas de autenticación, registro y uso del chatbot.
- Puede interactuar con el sistema a través de diferentes métodos de autenticación.

Actor Secundario: Administrador

- Representa al personal técnico y administrativo encargado de gestionar el sistema.
- Tiene privilegios elevados para administración, configuración y monitoreo.
- Puede realizar todas las funciones de un usuario más las funciones administrativas específicas.

2.4.2. Especificación Detallada de Casos de Uso.

CU01 – Login

Tabla 21

Descripción de caso de uso 1

Campo	Descripción
ID del Caso de uso	CU01
Nombre	Login
Actor Principal	Usuario
Actores Secundarios	Sistema de Autenticación
Descripción	Permite al usuario autenticarse en el sistema utilizando credenciales previamente registradas
Precondiciones	• Usuario debe tener cuenta registrada en el sistema • Credenciales válidas
Postcondiciones	• Usuario autenticado exitosamente • Sesión activa establecida • Acceso a funcionalidades según rol
Flujo Principal	**1.** Usuario accede a la página de login **2.** Sistema muestra formulario de autenticación **3.** Usuario ingresa credenciales (email y contraseña) **4.** Sistema valida credenciales **5.** Sistema establece sesión activa **6.** Sistema redirige a página principal del chatbot

Nota. Elaboración propia.

CU02 - Crear Cuenta

Tabla 22

Descripción de caso de uso 2

Campo	Descripción
ID del Caso de uso	CU02
Nombre	Crear Cuenta
Actor Principal	Usuario
Actores Secundarios	Sistema de Registro, Servicio de Email
Descripción	Permite a nuevos usuarios registrarse en el sistema proporcionando información personal básica
Precondiciones	• Acceso a formulario de registro • Email válido no registrado previamente
Postcondiciones	• Nueva cuenta de usuario creada • Email de confirmación enviado • Usuario puede autenticarse
Flujo Principal	**1.** Usuario accede a página de registro **2.** Sistema muestra formulario de registro **3.** Usuario completa campos obligatorios (nombre, apellido, email, contraseña) **4.** Sistema valida datos ingresados **5.** Sistema verifica unicidad del email **6.** Sistema crea nueva cuenta **7.** Sistema envía email de confirmación **8.** Sistema muestra mensaje de éxito

Nota. Elaboración propia.

CU03 - Recuperar Contraseña

Tabla 23

Descripción de caso de uso 3

Campo	Descripción
ID del Caso de uso	CU03
Nombre	Recuperar Contraseña
Actor Principal	Usuario
Actores Secundarios	Servicio de Email
Descripción	Permite a usuarios registrados restablecer su contraseña cuando la han olvidado
Precondiciones	• Usuario registrado en el sistema • Acceso al email asociado a la cuenta
Postcondiciones	• Nueva contraseña establecida • Usuario puede acceder con nueva contraseña
Flujo Principal	**1.** Usuario accede a opción "Olvidé mi contraseña" **2.** Sistema muestra formulario de recuperación **3.** Usuario ingresa email registrado **4.** Sistema valida existencia del email **5.** Sistema genera token temporal de recuperación **6.** Sistema envía email con enlace de recuperación **7.** Usuario accede al enlace desde su email **8.** Sistema valida token y muestra formulario de nueva contraseña **9.** Usuario ingresa nueva contraseña **10.** Sistema actualiza contraseña **11.** Sistema confirma cambio exitoso

Nota. Elaboración propia.

CU04 - Login con Gmail

Tabla 24

Descripción de caso de uso 4

Campo	Descripción
ID del Caso de uso	CU04
Nombre	Login con Gmail
Actor Principal	Usuario
Actores Secundarios	Google OAuth Service
Descripción	Permite autenticación rápida utilizando credenciales de Google mediante OAuth 2.0
Precondiciones	• Usuario con cuenta Gmail válida • Servicio Google OAuth disponible
Postcondiciones	• Usuario autenticado via Google • Perfil básico sincronizado • Sesión establecida
Flujo Principal	**1.** Usuario selecciona opción "Iniciar sesión con Gmail" **2.** Sistema redirige a página de autenticación de Google **3.** Usuario autoriza acceso a la aplicación **4.** Google retorna token de autenticación **5.** Sistema valida token con Google **6.** Sistema obtiene información básica del perfil **7.** Sistema crea o actualiza cuenta local **8.** Sistema establece sesión activa **9.** Sistema redirige a página principal

Nota. Elaboración propia.

CU05 – Chats

Tabla 25

Descripción de caso de uso 5

Campo	Descripción
ID del Caso de uso	CU05
Nombre	Chats
Actor Principal	Usuario
Actores Secundarios	Sistema RAG, Gemini 2.0
Descripción	Funcionalidad principal que permite al usuario interactuar con el chatbot para obtener información institucional
Precondiciones	• Usuario autenticado en el sistema • Sistema RAG operativo • Base de conocimiento indexada
Postcondiciones	• Conversación almacenada en historial • Respuesta relevante proporcionada • Métricas de satisfacción registradas
Flujo Principal	**1.** Usuario accede a interfaz de chat **2.** Sistema muestra historial previo (si existe) **3.** Usuario escribe consulta en lenguaje natural **4.** Sistema procesa consulta usando NLP **5.** Sistema busca información relevante en base de conocimiento **6.** Sistema envía contexto y consulta a Gemini 2.0 **7.** Sistema recibe respuesta del LLM **8.** Sistema presenta respuesta en interfaz **9.** Usuario evalúa respuesta (like/dislike) **10.** Sistema registra evaluación y almacena conversación

Nota. Elaboración propia.

CU06 - Login Admin

Tabla 26

Descripción de caso de uso 6

Campo	Descripción
ID del Caso de uso	CU06
Nombre	Login Admin
Actor Principal	Adminsitrador
Actores Secundarios	Sistema de Autentificación
Descripción	Autenticación específica para usuarios administradores con validación de privilegios elevados
Precondiciones	• Cuenta de administrador válida • Credenciales administrativas
Postcondiciones	• Administrador autenticado • Acceso a panel administrativo • Permisos elevados activados
Flujo Principal	**1.** Administrador accede a portal administrativo **2.** Sistema muestra formulario de login administrativo **3.** Administrador ingresa credenciales **4.** Sistema valida credenciales y permisos **5.** Sistema verifica rol de administrador **6.** Sistema establece sesión administrativa **7.** Sistema redirige a dashboard administrativo

Nota. Elaboración propia.

CU07 - Panel de Estadísticas

Tabla 27

Descripción de caso de uso 7

Campo	Descripción
ID del Caso de uso	CU07
Nombre	Panel de Estadísticas
Actor Principal	Adminsitrador
Actores Secundarios	Sistema de Analytics
Descripción	Visualización de métricas y estadísticas de uso del chatbot para análisis y toma de decisiones
Precondiciones	• Administrador autenticado • Datos estadísticos disponibles
Postcondiciones	• Métricas visualizadas • Reportes generados • Insights disponibles para decisiones
Flujo Principal	**1.** Administrador accede al panel de estadística **2.** Sistema carga datos de uso y satisfacción **3.** Sistema genera gráficos estadísticos (diagrama de pastel like/dislike **4.** Administrador puede visualizar estadísticas de las respuestas del bot

Nota. Elaboración propia.

CU08 - Subir Documentos

Tabla 28

Descripción de caso de uso 8

Campo	Descripción
ID del Caso de uso	CU08
Nombre	Subir Documentos
Actor Principal	Administrador
Actores Secundarios	Sistema de Procesamiento de Documentos, Pipeline RAG
Descripción	Permite a administradores cargar documentos PDF que serán procesados e integrados a la base de conocimiento
Precondiciones	• Administrador autenticado • Archivo PDF válido • Sistema de procesamiento operativo
Postcondiciones	• Documento procesado y indexado • Base de conocimiento actualizada • Nuevo contenido disponible para consultas
Flujo Principal	**1.** Administrador accede a sección de gestión de documentos **2.** Sistema muestra interfaz de carga **3.** Administrador selecciona archivo PDF **4.** Sistema valida formato y tamaño **5.** Administrador confirma carga **6.** Sistema procesa archivo (extracción de texto) **7.** Sistema genera embeddings del contenido **8.** Sistema indexa en base vectorial FAISS **9.** Sistema actualiza base de conocimiento **10.** Sistema notifica procesamiento exitoso

Nota. Elaboración propia.

CU09 - Crear Administradores

Tabla 29

Descripción de caso de uso 9

Campo	Descripción
ID del Caso de uso	CU09
Nombre	Crear Adminsitradores
Actor Principal	Adminsitrador (Maestro)
Actores Secundarios	Sistema de Gestión de Usuarios
Descripción	Permite al administrador principal crear nuevas cuentas administrativas con permisos específicos
Precondiciones	• Administrador maestro autenticado • Permisos de creación de usuarios
Postcondiciones	• Nueva cuenta administrativa creada • Permisos asignados correctamente • Notificación enviada al nuevo administrador
Flujo Principal	**1.** Administrador maestro accede a gestión de usuarios **2.** Sistema muestra interfaz de creación de administradores **3.** Administrador maestro completa formulario (email, clave) **4.** Sistema valida datos ingresados **5.** Sistema verifica permisos del administrador maestro **6.** Sistema crea nueva cuenta administrativa

Nota. Elaboración propia.

2.4.3. Matriz de casos de uso vs requerimientos.

Tabla 30

Matriz de caso de uso vs requerimientos

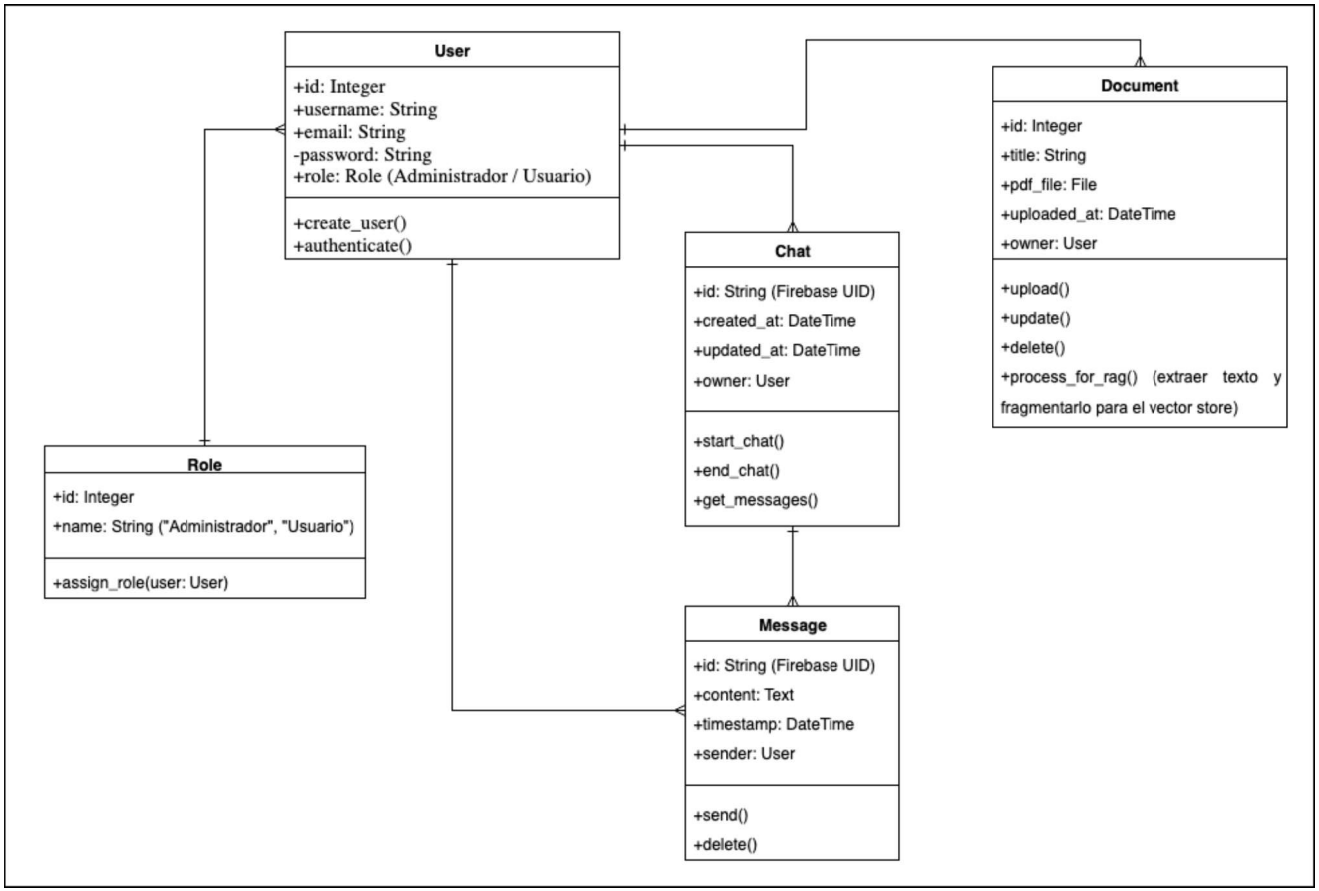
Caso de Uso	RF01	RF02	RF03	RF04	RF05	RF06	RF07	RF08
CU01 - Login	✓							✓
CU02 - Crear Cuenta				✓				
CU03 - Recuperar Contraseña					✓			
CU04 - Login Gmail	✓							✓
CU05 - Chats		✓						
CU06 - Login Admin	✓							
CU07 - Panel Estadísticas			✓					
CU08 - Subir Documentos						✓		
CU09 - Crear Administradores							✓	

Nota. Elaboración propia.

2.5. Diagrama de clases.

Figura 8

Diagrama de clases



Nota. Elaboración propia.

Este diagrama representa la arquitectura de datos de un sistema de chatbot universitario con gestión de documentos. La estructura está diseñada para manejar usuarios, conversaciones y documentos PDF que alimentan la base de conocimiento del sistema.

2.5.1. Entidades Principales

Clase User Representa a los usuarios del sistema con cosas básicas de identificación (id, username, email, password) y un campo role que diferencia entre administradores y usuarios normales. Esta clase tiene métodos básicos para crear usuarios y hacer la autenticación, estableciendo cómo se controla el acceso al sistema.

Clase Role Implementa un sistema de roles simple que me permite asignar permisos específicos a los usuarios. Solo tiene dos roles predefinidos: "Administrador" y "Usuario", lo que da un mecanismo de control de acceso básico pero que funciona bien.

Clase Document Se encarga de los archivos PDF que forman la base de conocimiento del chatbot. Guarda información importante como título, archivo, fecha de carga y quién lo subió. El método process_for_rag() es clave para procesar documentos, ya que extrae el texto y lo divide en pedazos para después indexarlo en el vector store que usa el sistema RAG (Retrieval-Augmented Generation).

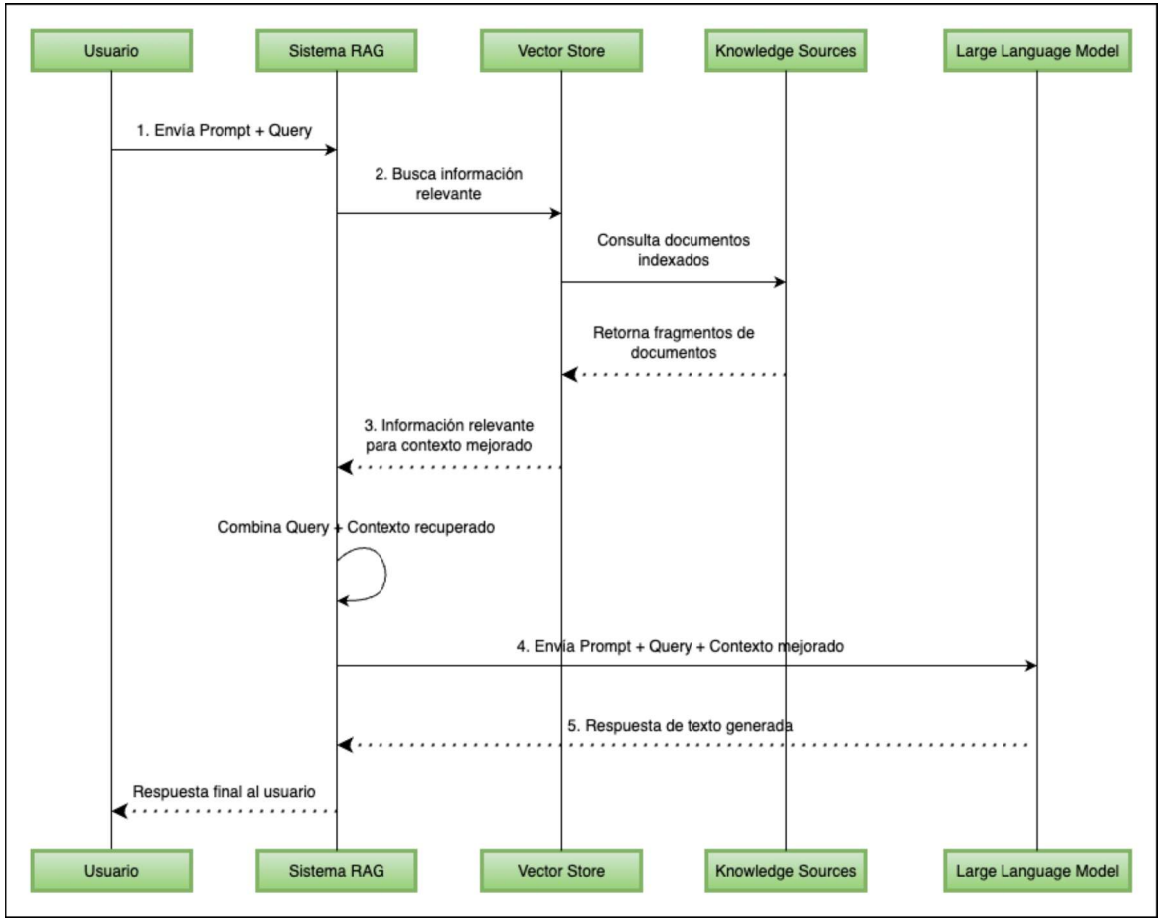
Clase Chat Representa las sesiones de conversación entre usuarios y el chatbot. Usa el ID de Firebase como identificador único y mantiene referencias de cuándo se creó y actualizó. Esta clase funciona como un contenedor para organizar los mensajes por sesión de usuario.

Clase Message Guarda los mensajes individuales dentro de cada conversación. Cada mensaje incluye el contenido del texto, timestamp y referencia a quién lo mandó. Esta estructura me permite mantener un historial completo de las interacciones entre usuarios y el sistema.

2.6. Diagrama de secuencia usuario - chatbot.

Figura 9

Diagrama de secuencia usuario - chatbot

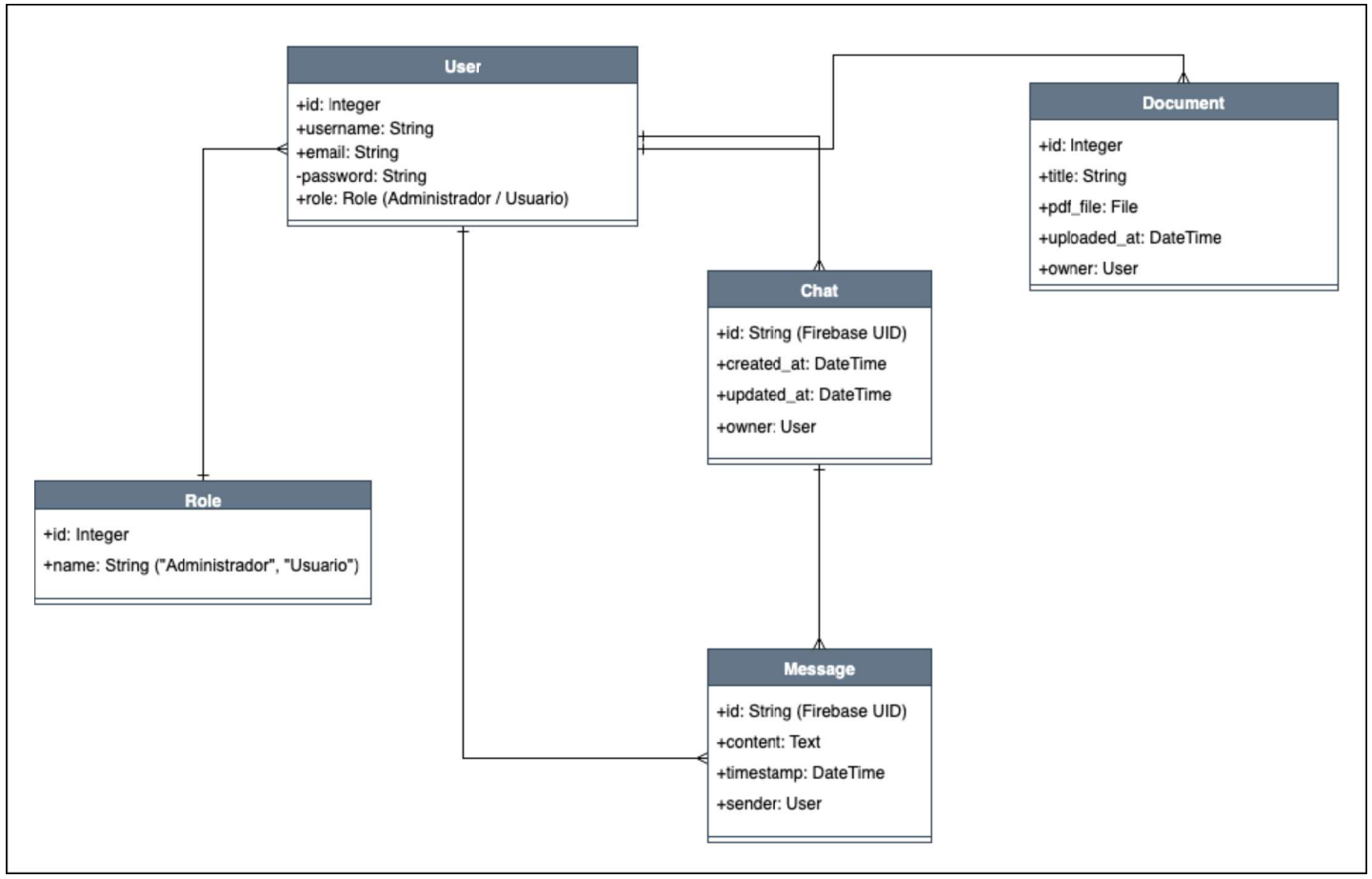


Nota. Elaboración propia.

2.7. Modelo entidad – relación del sistema.

Figura 10

Diagrama Entidad-Relación

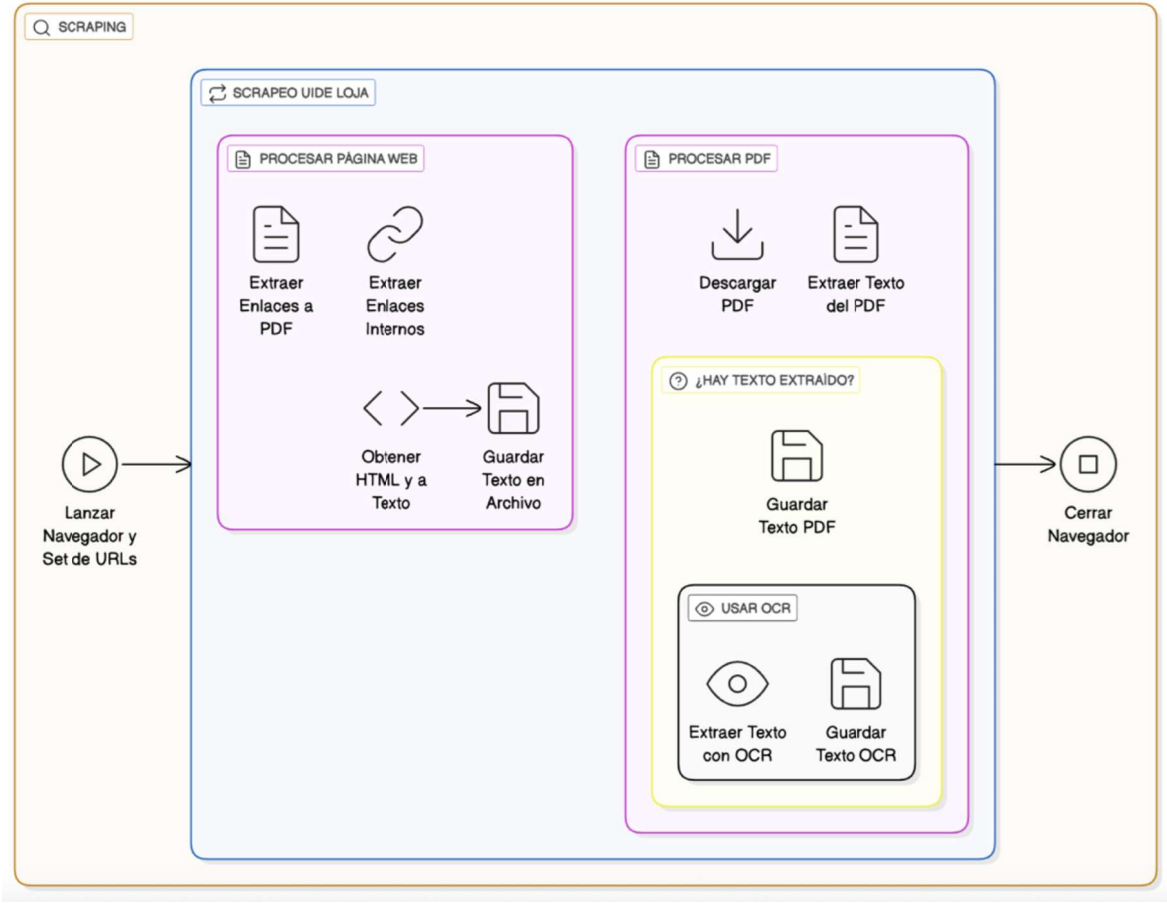


Nota. Elaboración propia.

2.8. Diagrama de scrapeo al portal de la UIDE, campus Loja.

Figura 11

Proceso de scrapeo al portal UIDE, campus Loja

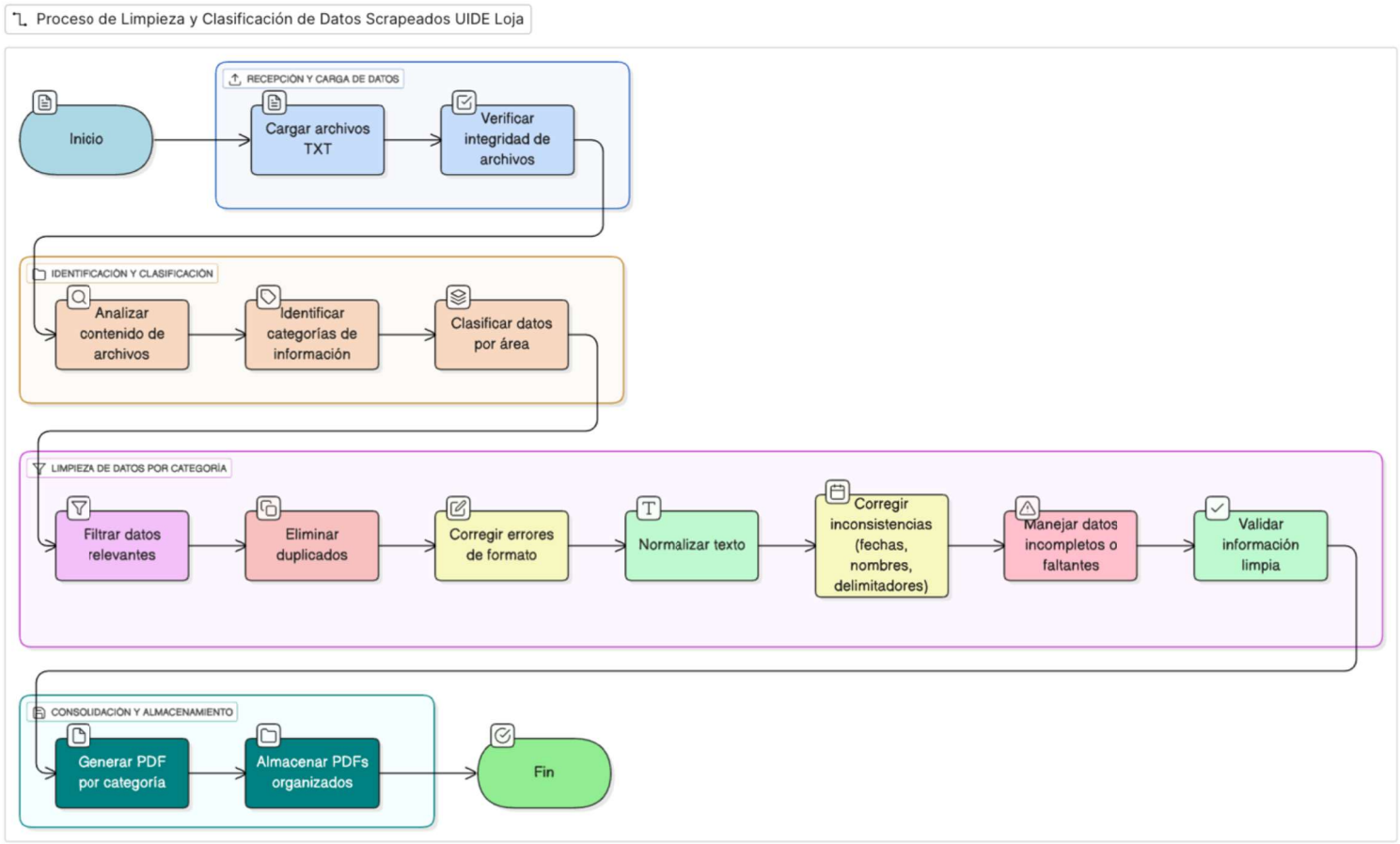


Nota. Elaboración propia.

2.9. Proceso de preprocesamiento y limpieza de datos extraídos mediante scrapeo.

Figura 12

Preprocesamiento y limpieza de datos



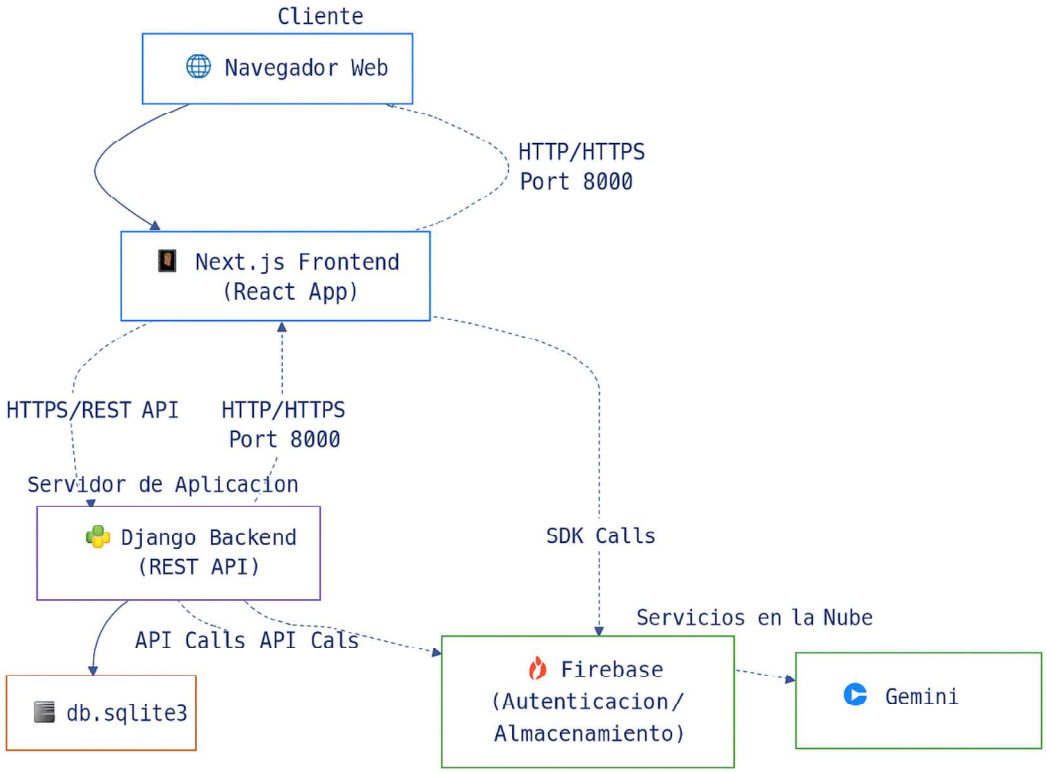
Nota. Elaboración propia.

El diagrama ilustra el pipeline completo de preprocesamiento de información extraída del sitio web universitario, comenzando con la recepción y carga de archivos de texto, seguido de verificación de integridad, análisis de contenido para identificación y clasificación por áreas temáticas académicas, continuando con un proceso sistemático de limpieza que incluye filtrado de datos relevantes, eliminación de duplicados, corrección de errores de formato, normalización de texto, corrección de inconsistencias en fechas y nombres, manejo de datos incompletos y validación final, concluyendo con la consolidación y almacenamiento de documentos PDF organizados por categorías para su posterior indexación en el sistema de recuperación de información del chatbot universitario.

2.10. Diseño de la arquitectura del sistema.

Figura 13

Arquitectura del sistema



Nota. Elaboración propia.

El diseño arquitectónico del sistema de chatbot universitario responde a un enfoque distribuido, modular y escalable, basado en el paradigma Retrieval-Augmented Generation (RAG) y el modelo cliente-servidor. La solución combina procesamiento inteligente de consultas en lenguaje natural con recuperación semántica de información institucional, servicios centralizados de autenticación, gestión de conversaciones y almacenamiento de documentos académicos, lo que permite una operación contextualmente precisa, segura y trazable para responder consultas universitarias específicas.

2.10.1. Arquitectura RAG distribuida.

La arquitectura que implementé separa claramente la parte visual (Next.js), el procesamiento de conocimiento (Django con sistema RAG) y los servicios de infraestructura (Firebase y Gemini 2.0). Esta separación hace que el procesamiento de embeddings y la búsqueda vectorial corran en el backend mientras la interfaz de chat se mantiene fluida en tiempo real, mejorando tanto la experiencia del usuario como la precisión de las respuestas basadas en documentos oficiales de la universidad.

2.10.2. Capa de presentación Conversacional.

El frontend hecho en Next.js me da una interfaz de chat fácil de usar que maneja la interacción usuario-sistema, conecta la autenticación con Gmail a través de Firebase, guarda automáticamente las conversaciones y sincroniza todo en tiempo real. Esta capa esconde la complejidad del procesamiento RAG y le presenta al usuario una experiencia de chat natural, pero manteniendo acceso completo al historial de conversaciones y funciones administrativas para manejar documentos.

2.10.3. Motor de procesamiento RAG.

El backend Django implementa el corazón del sistema RAG conectando FAISS para búsqueda vectorial semántica, Google Embeddings para vectorizar contenido y Gemini 2.0 para generar respuestas con contexto. Esta capa procesa los PDFs de la universidad, los divide en

pedazos que tengan sentido semánticamente, genera embeddings y hace búsquedas de similitud coseno para encontrar información relevante que enriquece el contexto de las preguntas antes de generar respuestas.

2.10.4. Infraestructura de Servicios.

Firebase me da autenticación sólida con login social (Gmail), recuperación de contraseñas y almacenamiento NoSQL para conversaciones mediante Firestore, garantizando que todo se guarde de manera distribuida y se sincronice entre dispositivos. La base de datos SQLite3 maneja metadatos estructurados de documentos y configuraciones del sistema, mientras que Gemini 2.0 actúa como motor de procesamiento de lenguaje natural, dando capacidades avanzadas de comprensión semántica y generación de texto apropiado para el contexto universitario.

2.10.5. Flujo de Procesamiento RAG.

El sistema implementa un pipeline completo de RAG donde las preguntas de los usuarios se procesan mediante búsqueda semántica en la base vectorial de documentos universitarios, encontrando los fragmentos más relevantes que se combinan con la pregunta original para formar un prompt enriquecido que se manda a Gemini 2.0. Esto genera respuestas basadas en información institucional verificable y guarda tanto la pregunta como la respuesta en Firebase para mantener continuidad en la conversación y poder rastrear las interacciones.

Esta arquitectura RAG distribuida asegura que cada respuesta del chatbot esté basada en documentación oficial de la Universidad Internacional del Ecuador Campus Loja, eliminando alucinaciones del modelo de lenguaje y dando un sistema de información universitaria preciso, escalable y fácil de mantener.

CAPÍTULO III: DESARROLLO E IMPLEMENTACIÓN

3. Descripción del desarrollo.

3.1. Entorno de desarrollo y herramientas tecnológicas.

Para desarrollar el sistema de chatbot universitario basado en RAG usé un conjunto amplio y especializado de herramientas tecnológicas, que elegí por qué tan sólidas son, cómo se llevan con sistemas de procesamiento de lenguaje natural y porque tienen comunidades activas que dan soporte. Organicé estas herramientas según el papel que tienen dentro de la arquitectura distribuida del sistema, cubriendo desde el frontend conversacional hasta el backend de procesamiento RAG, servicios de autenticación y almacenamiento en la nube.

3.1.1. Lenguajes, frameworks y bibliotecas utilizadas.

La Tabla 31 resume las tecnologías empleadas, detallando su categoría, versión implementada y propósito específico dentro del proyecto. Esta clasificación permite visualizar de forma estructurada los componentes clave que hicieron posible la construcción del sistema completo de chatbot universitario:

Tabla 31

Herramientas tecnológicas utilizadas en el desarrollo del sistema

Categoría	Tecnología	Versión	Propósito
Lenguajes de Programación	Python	3.9+	Lenguaje principal para backend y procesamiento RAG
	TypeScript	4.9+	Lenguaje tipado para frontend
	JavaScript	ES2022	Funcionalidades dinámicas del frontend
Frontend/Cliente Web	Next.js	13.5+	Framework React para SSR y optimización
	React	18.2+	Biblioteca para construcción de interfaces
	Tailwind CSS	3.3+	Framework CSS utilitario para diseño responsivo
	Lucide React	0.263.1	Biblioteca de iconos moderna
Backend/Servidor	Django	4.2+	Framework web principal para APIs
	Django REST Framework	3.14+	Construcción de APIs RESTful
	python-dotenv	1.0+	Gestión de variables de entorno
Procesamiento de Lenguaje Natural	LangChain	0.1+	Framework para aplicaciones LLM
	LangChain Google GenAI	1.0+	Integración con modelos Gemini
	LangChain Community	0.0.20+	Extensiones de la comunidad LangChain
Embeddings y Búsqueda Vectorial	Google Generative AI	0.3+	API para embeddings y generación
	FAISS	1.7+	Búsqueda de similitud vectorial eficiente
	sentence-transformers	2.2+	Modelos de embeddings semánticos
Procesamiento de Documentos	PyPDF2	3.0+	Extracción de texto de archivos PDF
	python-docx	0.8+	Procesamiento de documentos Word
	beautifulsoup4	4.12+	Parsing y extracción de contenido HTML
	requests	2.31+	Cliente HTTP para web scraping
Cálculo Científico y Análisis	NumPy	1.24+	Operaciones matemáticas y arrays
	SciPy	1.10+	Algoritmos científicos para similitud
	Pandas	2.0+	Manipulación y análisis de datos
Autenticación y Base de Datos	Firebase Admin SDK	6.2+	Autenticación y Firestore backend
	firebase-admin	6.2+	SDK de administración Firebase

	SQLite3	3.40+	Base de datos local para metadatos
Servicios de IA	Google Gemini 2.0 API	-	Modelo de lenguaje para generación
	Google AI Studio	-	Interfaz de configuración de modelos
	Vertex AI	-	Plataforma de ML de Google Cloud
Optimización y Rendimiento	asyncio	3.9+	Programación asíncrona en Python
	concurrent.futures	3.9+	Ejecución paralela de tareas
	cachetools	5.3+	Sistemas de caché para optimización
Procesamiento de Texto	NLTK	3.8+	Toolkit de procesamiento de lenguaje natural
	spaCy	3.6+	Procesamiento avanzado de texto en español
	regex	2023+	Expresiones regulares avanzadas
Comunicación y Redes	urllib3	2.0+	Cliente HTTP de bajo nivel
	certifi	2023+	Certificados SSL/TLS
	charset-normalizer	3.2+	Detección de codificación de caracteres
Desarrollo y Testing	pytest	7.4+	Framework de testing
	black	23.7+	Formateador de código Python
	flake8	6.0+	Linter para calidad de código
	ESLint	8.47+	Linter para TypeScript/JavaScript
Web Scraping	Selenium	4.11+	Automatización de navegador web
	ChromeDriver	115+	Driver para Google Chrome
	scrapy	2.10+	Framework de web scraping
Almacenamiento y Hosting	Vercel	-	Plataforma de hosting para Next.js
	Railway/Render	-	Hosting para backend Django
	Firebase Storage	-	Almacenamiento de archivos en la nube
	GitHub	-	Control de versiones y CI/CD
Monitoreo y Logging	python-logging	3.9+	Sistema de logging nativo
	winston	3.10+	Logging para aplicaciones Node.js
	sentry-sdk	1.32+	Monitoreo de errores en producción

Nota. Elaboración propia.

Como se evidencia en la tabla, el sistema integra múltiples niveles tecnológicos especializados en procesamiento de lenguaje natural: en el frontend se utiliza Next.js con TypeScript para construcción de interfaces conversacionales modernas y responsivas, mientras que el backend emplea Django REST Framework junto con LangChain y Google Generative AI para implementar la arquitectura RAG completa.

El núcleo del sistema RAG se construye sobre FAISS para búsqueda vectorial eficiente, Google Embeddings para vectorización semántica de documentos, y Gemini 2.0 para generación de respuestas contextualizadas. La integración con Firebase proporciona autenticación robusta, almacenamiento de conversaciones en tiempo real y gestión de archivos, mientras que las herramientas de web scraping (Selenium, BeautifulSoup) facilitan la recolección automatizada de información institucional.

Además, se incorporan librerías especializadas en procesamiento de texto en español (spaCy, NLTK), análisis numérico para cálculos de similitud semántica, herramientas de desarrollo para testing y calidad de código, y servicios de hosting modernos que garantizan una solución integral, escalable y optimizada para el procesamiento de consultas universitarias en lenguaje natural.

3.1.2. Justificación técnica de la pila de desarrollo.

La selección de las tecnologías empleadas en el desarrollo del sistema de chatbot universitario basado en RAG se fundamentó en una evaluación técnica rigurosa que contempló criterios como capacidades de procesamiento de lenguaje natural, escalabilidad para sistemas conversacionales, disponibilidad de APIs de inteligencia artificial, experiencia previa del equipo desarrollador y cumplimiento estricto de los requisitos funcionales y no funcionales del sistema de recuperación de información universitaria.

A continuación, explico por qué elegí cada componente tecnológico dentro de la arquitectura del sistema, organizándolo por categorías:

Lenguaje Principal: Python 3.9+

- **Ecosistema de NLP/IA:** Python es el estándar de facto para procesamiento de lenguaje natural y machine learning, con bibliotecas maduras como LangChain, FAISS y transformers.
- **Compatibilidad:** Multiplataforma (Windows, Linux, macOS) sin modificaciones de código.
- **Disponibilidad de recursos:** Amplia documentación, comunidad activa especializada en NLP y abundantes tutoriales sobre RAG.
- **Eficiencia de desarrollo:** Sintaxis clara que acelera el desarrollo de pipelines de procesamiento de texto.
- **Lo que necesita el sistema:** Perfecto para integrar IA generativa, embeddings semánticos y desarrollo web en un solo lugar.

Framework RAG: LangChain + Google Generative AI

- **Abstracción especializada:** LangChain simplifica la construcción de aplicaciones RAG con componentes predefinidos.
- **Integración nativa:** Soporte directo para Google Gemini y embeddings de Google.
- **Modularidad:** Permite intercambiar componentes (vectorstores, LLMs, retrievers) sin refactoring completo.
- **Cadenas de procesamiento:** Facilita la construcción de pipelines complejos de recuperación y generación.
- **Requisitos específicos:** Sistema RAG requiere orquestación precisa entre búsqueda vectorial y generación de texto.

Motor de Búsqueda Vectorial: FAISS

- **Eficiencia en tiempo real:** FAISS ofrece búsqueda de similitud coseno optimizada para conjuntos de datos grandes.
- **Escalabilidad:** Maneja millones de vectores con latencia sub-segundo.

- **Algoritmos avanzados:** Implementa índices aproximados (IVF, HNSW) para búsqueda rápida.
- **Compatibilidad con embeddings:** Soporte nativo para vectores de alta dimensionalidad (768, 1024, 1536).
- **Requisitos del sistema:** Búsqueda semántica en documentos universitarios requiere precisión y velocidad.

Frontend Conversacional: Next.js + React + TypeScript.

- **Server-Side Rendering:** Next.js optimiza la carga inicial y SEO para aplicaciones conversacionales.
- **Tipado estático:** TypeScript reduce errores en desarrollo y mejora mantenibilidad.
- **Componentes reactivos:** React facilita la construcción de interfaces de chat dinámicas.
- **Optimizaciones automáticas:** Code splitting, lazy loading, y optimización de imágenes incluidas.
- **Requisitos del sistema:** Interfaz conversacional requiere actualizaciones en tiempo real y experiencia de usuario fluida.

Backend API: Django + Django REST Framework.

- **Desarrollo rápido:** "Batteries included" - ORM, autenticación, panel de admin ya incluidos.
- **Seguridad:** Protecciones integradas contra CSRF, SQL injection, XSS.
- **Arquitectura MVT:** Hace más fácil separar la lógica RAG, APIs y modelos de datos.
- **API REST sólida:** DRF te da serialización, autenticación por tokens y documentación automática**.**
- **Experiencia previa:** Framework familiar que reduce el tiempo de desarrollo de endpoints especializados.

Servicios de Autenticación: Firebase Authentication

- **Login social:** Se conecta fácil con Gmail sin tener que configurar OAuth de manera complicada.
- **Gestión de usuarios:** Sistema completo de registro, login, recuperación de contraseñas.
- **Tokens JWT:** Autenticación stateless que escala horizontalmente.
- **Seguridad:** Encriptación end-to-end y gestión automática de certificados.
- **Lo que necesita el sistema:** Necesidad de autenticación universitaria segura y manejo de sesiones.

Almacenamiento de Conversaciones: Firebase Firestore

- **Tiempo real:** Sincronización automática de conversaciones entre dispositivos.
- **Escalabilidad:** Base de datos NoSQL que maneja millones de documentos de chat.
- **Estructura jerárquica:** Organización natural usuarios → chats → mensajes.
- **Consultas complejas:** Soporte para filtros, ordenamiento y paginación de historial.
- **Backup automático:** Replicación y recuperación sin configuración manual.

Modelo de Lenguaje: Google Gemini 2.0

- **Capacidades multimodales:** Procesamiento de texto y documentos en una sola API.
- **Contexto extendido:** Soporte para prompts largos necesarios en aplicaciones RAG.
- **Calidad en español:** Optimizado para comprensión y generación en español latinoamericano.
- **Integración nativa:** API directa con LangChain sin adaptadores adicionales.
- **Costo-efectividad:** Pricing competitivo para aplicaciones educativas.

Procesamiento de Documentos: PyPDF2 + BeautifulSoup + Selenium

- **PyPDF2:** Extracción confiable de texto de documentos PDF universitarios.
- **BeautifulSoup:** Parsing eficiente de contenido HTML del sitio web institucional.

- **Selenium:** Automatización de navegación para contenido dinámico y JavaScript.
- **Compatibilidad:** Soporte para múltiples formatos de documentos académicos.
- **Requisitos específicos:** Necesidad de procesar documentos PDF oficiales y contenido web universitario.

Fragmentación de Texto: RecursiveCharacterTextSplitter

- **Preservación semántica:** Mantiene coherencia de párrafos y secciones.
- **Configurabilidad:** Ajuste de tamaño de chunks según contexto del LLM.
- **Overlapping:** Superposición entre fragmentos para preservar contexto.
- **Optimización para embeddings:** Chunks de tamaño ideal para vectorización.

Hosting y Despliegue: Vercel + Railway/Render

Vercel (Frontend):

- Deploy automático desde Git.
- CDN global para latencia mínima.
- Optimizaciones automáticas para Next.js.
- SSL/HTTPS integrado.

Railway/Render (Backend):

- Escalabilidad automática según demanda.
- Base de datos PostgreSQL managed.
- Variables de entorno seguras.
- Integración continua.

3.1.3. Criterios de decisión consolidados.

A partir de la evaluación de las herramientas seleccionadas, se destacan los siguientes criterios que guiaron la construcción de la pila tecnológica del sistema:

Tabla 32

Criterios de decisión

Criterio	Justificación
Especialización en NLP	Soporte nativo para embeddings, vectorstores y modelos de lenguaje
Escalabilidad semántica	Capacidad de manejar grandes volúmenes de documentos y consultas simultáneas
Calidad en español	Optimización para procesamiento de texto académico en español
Tiempo real	Latencia mínima en búsqueda vectorial y generación de respuestas
Integración API	Conectividad fluida entre componentes RAG y servicios de terceros
Experiencia conversacional	Interfaces especializadas para aplicaciones de chat y diálogo
Seguridad universitaria	Autenticación robusta y protección de datos académicos sensibles

Nota. Elaboración propia.

3.1.4. Arquitectura RAG resultante.

Gracias a esta selección tecnológica cuidadosamente estructurada, el sistema ofrece:

- Búsqueda semántica precisa con latencia sub-segundo para documentos universitarios.
- Generación contextualizada basada en información institucional verificable.
- Conversaciones persistentes con sincronización en tiempo real entre dispositivos.
- Escalabilidad horizontal para múltiples usuarios y documentos simultáneos.
- Alta disponibilidad y seguridad en producción con backup automático.
- Mantenibilidad y extensibilidad para futuras mejoras en el modelo RAG.

3.1.5. Prototipado y análisis de código.

Prototipos.

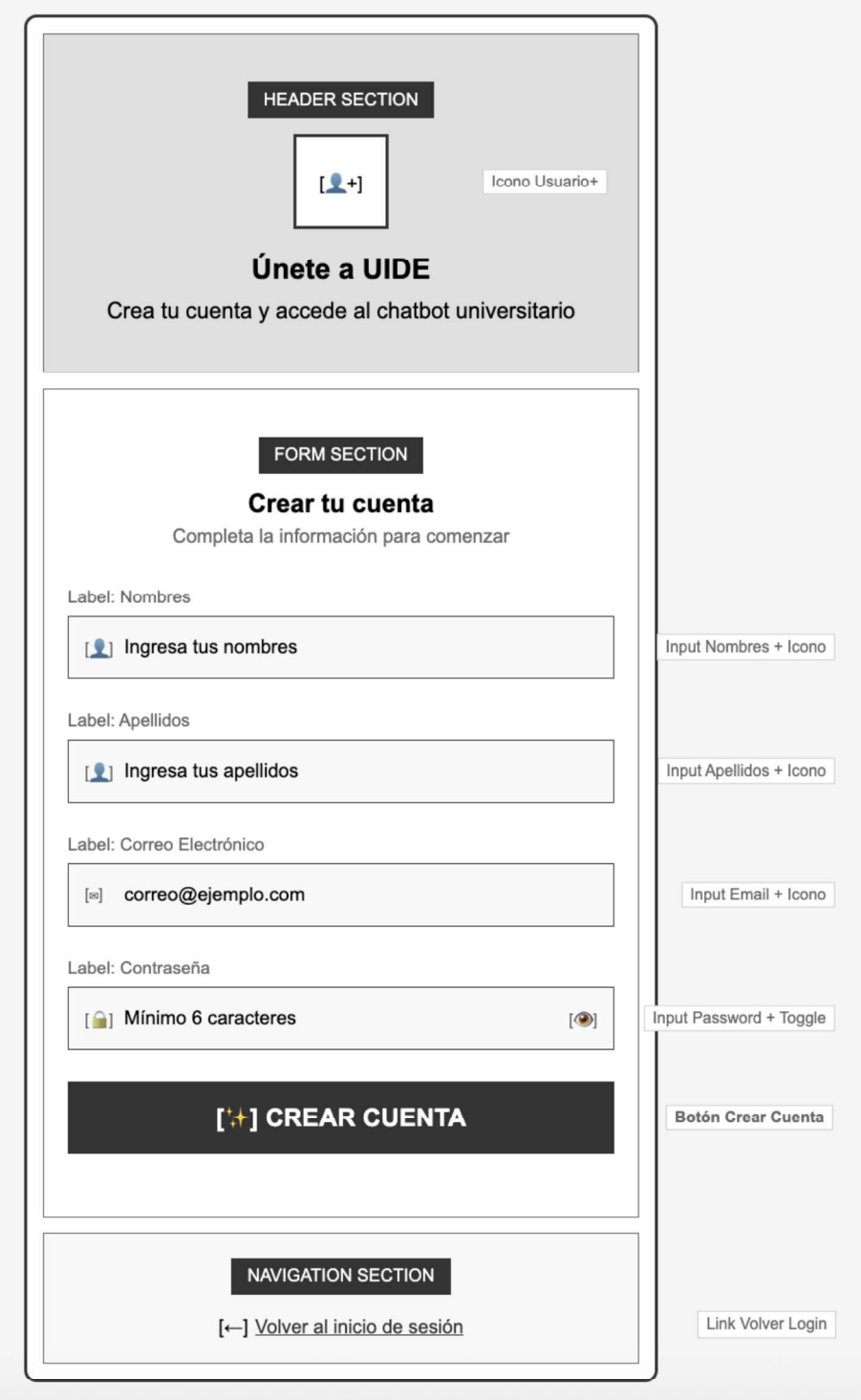
Figura 14

Prototipo pantalla de login

Nota. Elaboración propia.

Figura 15

Prototipo pantalla crear cuenta



Nota. Elaboración propia.

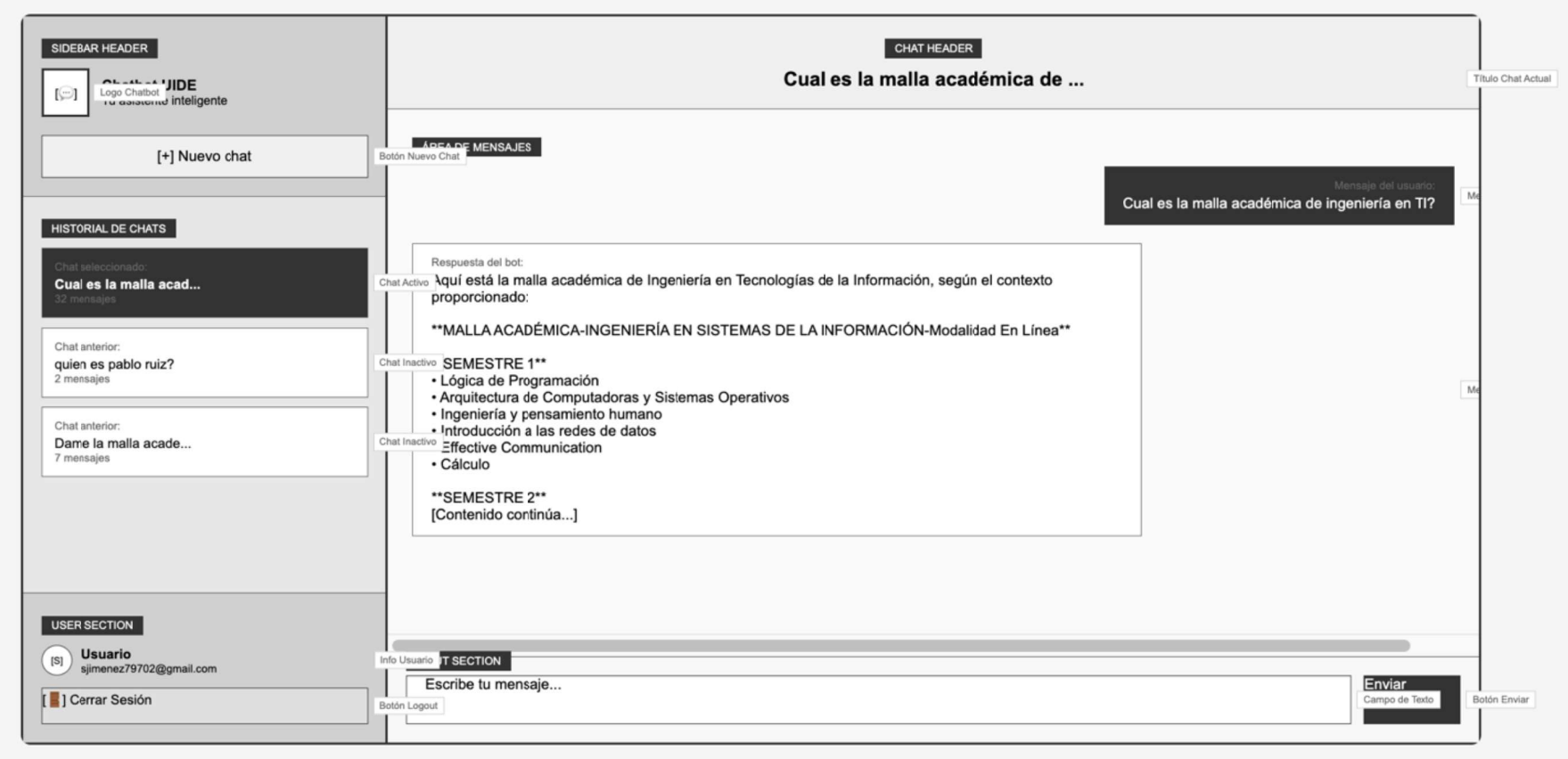
Figura 16

Prototipo de pantalla para recuperar contraseña

Nota. Elaboración propia.

Figura 17

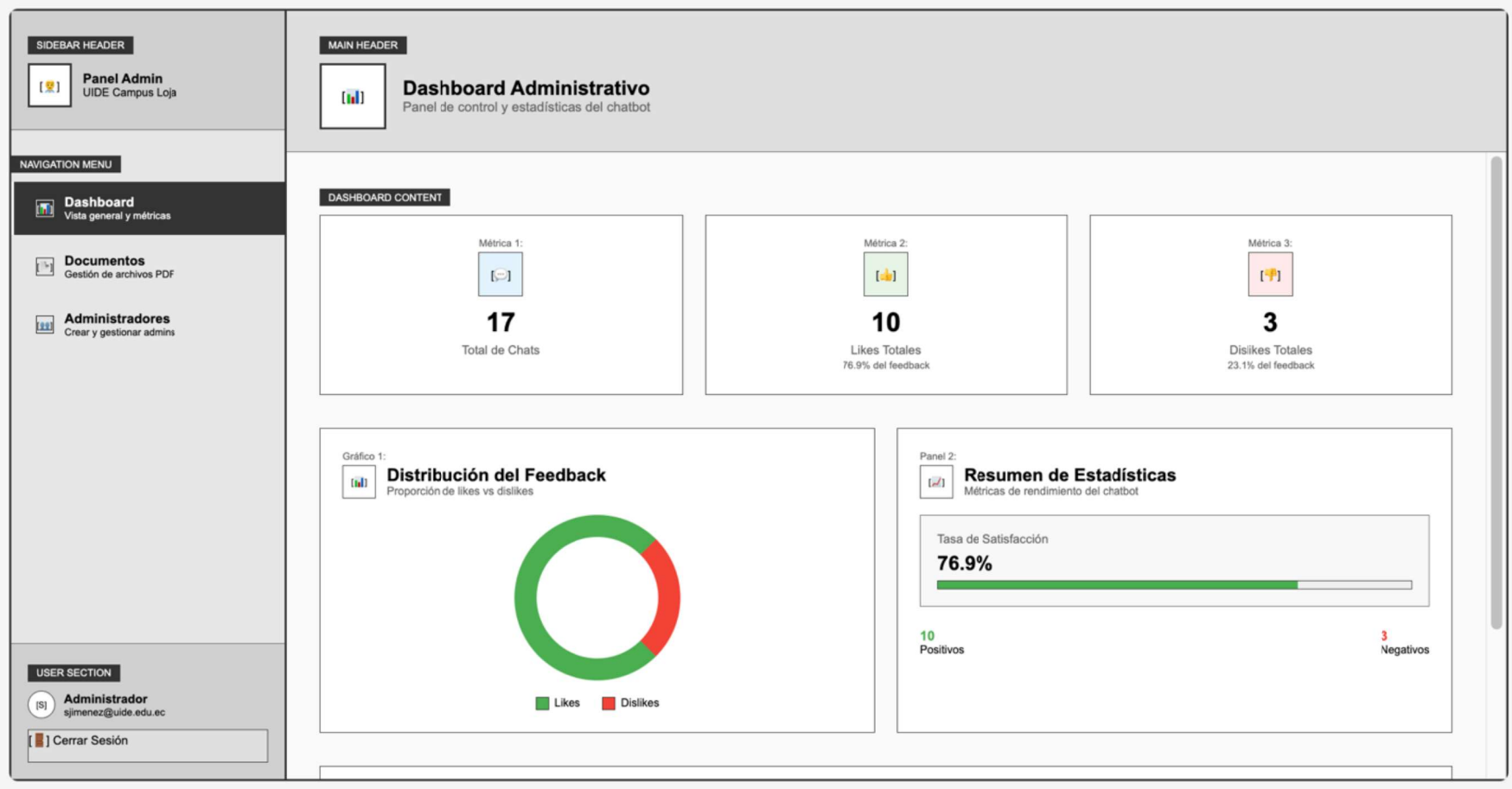
Prototipo pantalla de chat usuario



Nota. Elaboración propia.

Figura 18

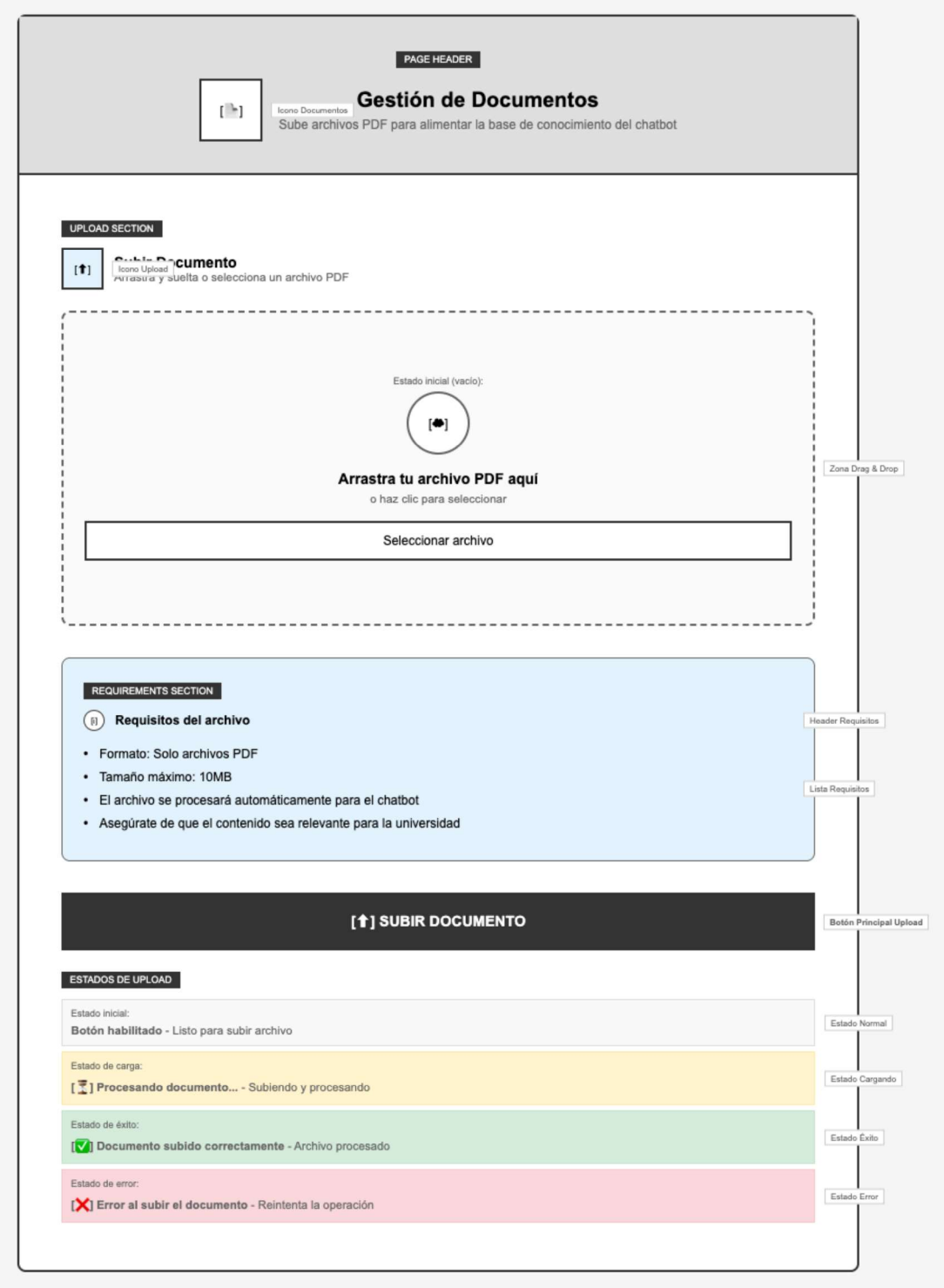
Prototipo dashboard administrativo



Nota. Elaboración propia.

Figura 19

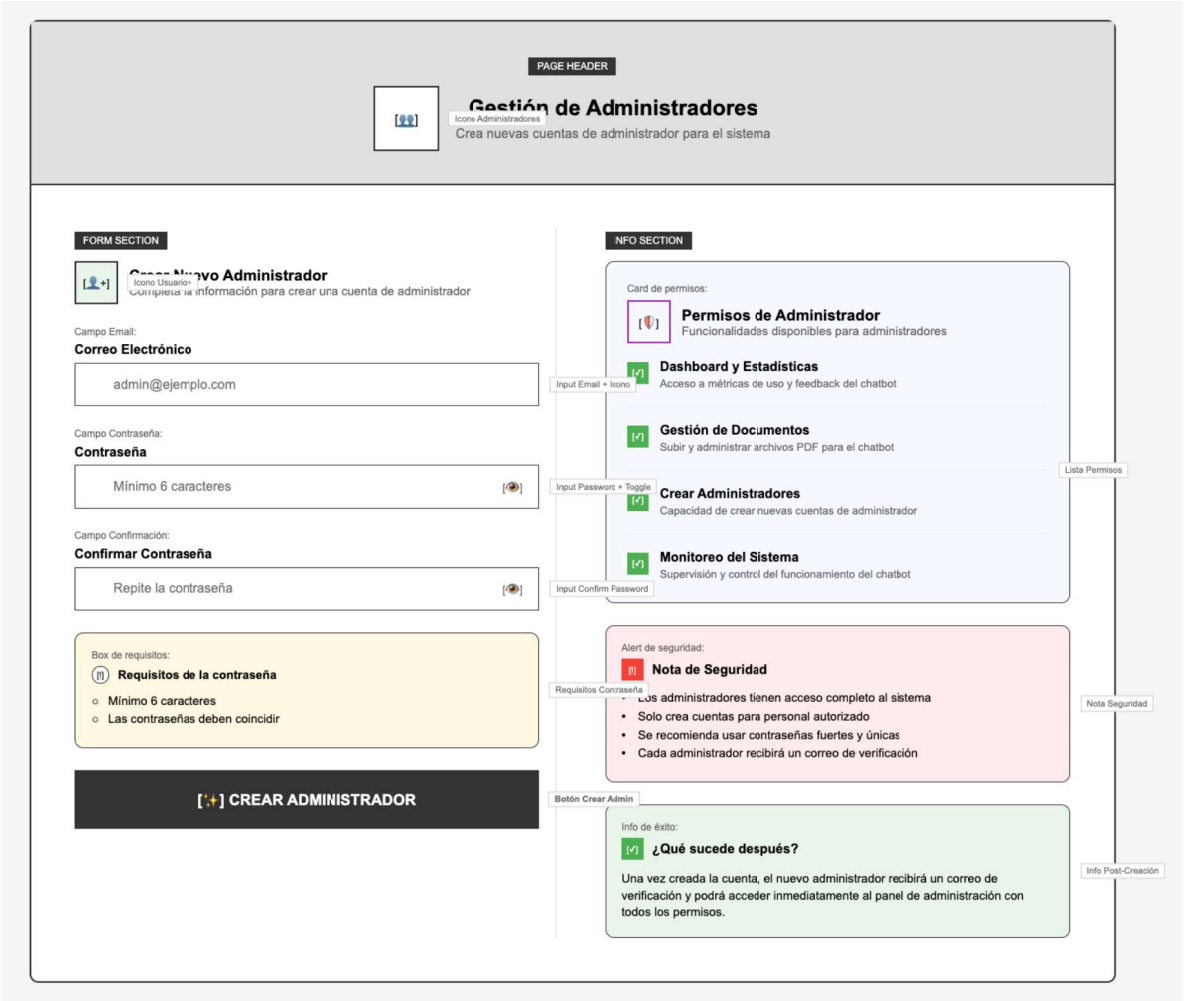
Prototipo pantalla para subir PDF y ser indexado



Nota. Elaboración propia.

Figura 20

Prototipo pantalla de crear administradores



Nota. Elaboración propia.

Análisis del código.

Desarrollo del scraper con porcentaje de eficiencia.

Figura 21

Arquitectura del Scraper

```
constructor(options = {}) {
    this.config = {
      maxDepth: options.maxDepth || 50,
      maxPages: options.maxPages || 10000,
      maxFilesPerPage: options.maxFilesPerPage || 100,
      maxLinksPerPage: options.maxLinksPerPage || 200,
      outputDir: options.outputDir ||
"./scraped_uide_COMPLETE",
      consolidatedFile:
        options.consolidatedFile ||
"./uide_contenido_TOTAL.txt",
      delay: options.delay || 500,
      timeout: options.timeout || 90000,
      userAgent:

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.3
6 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
,
      baseUrl: "https://www.uide.edu.ec",
      allowedDomains: ["uide.edu.ec"],
      fileTypes: [
        ".pdf",
        ".doc",
        ".docx",
        ".xls",
        ".xlsx",
        ".ppt",
        ".pptx",
        ".txt",
        ".zip",
        ".rar",
      ],
      ...options,
    };

    this.visitedUrls = new Set();
    this.discoveredUrls = new Set();
    this.downloadedFiles = new Set();
    this.errors = [];
    this.pendingUrls = [];
    this.stats = {
      pagesScraped: 0,
      filesDownloaded: 0,
      errorsEncountered: 0,
      totalCharacters: 0,
      urlsDiscovered: 0,
      startTime: Date.now(),
    };

    this.urlQueue = [];
    this.isRunning = true;
  }
```

Nota. Elaboración propia.

La implementación del sistema de web scraping utiliza Puppeteer como motor principal de automatización web, configurado para realizar extracción exhaustiva sin limitaciones de profundidad o cantidad de páginas. La clase ExhaustiveUIDEScraper implementa una arquitectura modular que gestiona múltiples aspectos de la extracción: configuración de parámetros, gestión de URLs visitadas, control de errores y estadísticas de rendimiento.

Metodología de descubrimiento de URLs.

El proceso de descubrimiento de contenido implementa una estrategia híbrida que combina análisis de sitemaps XML con exploración recursiva de enlaces. Esta metodología garantiza cobertura completa del sitio web institucional mediante tres enfoques complementarios: procesamiento exhaustivo de sitemaps, descubrimiento recursivo desde la página principal y detección dinámica de URLs durante la navegación.

Figura 22

Proceso de descubrimiento de URLs

```
async massiveUrlDiscovery() {
    console.log("🔍 FASE 1: DISCOVERY MASIVO DE URLs ... ");


    const seedUrls = [
      "https://www.uide.edu.ec/",
      "https://www.uide.edu.ec/sitemap.xml",
    ];

    // Agregar URLs semilla a la cola
    seedUrls.forEach((url) => this.addToQueue(url));

    // Procesar sitemaps de manera exhaustiva
    await this.exhaustiveSitemapProcessing();

    // Descubrimiento recursivo desde página principal
    await this.recursiveUrlDiscovery();

    console.log(
      `🎯 DISCOVERY COMPLETADO: ${this.urlQueue.length}
 URLs totales descubiertas`
    );
  }
```

Nota. Elaboración propia.

Extracción de contenido estructurado.

Figura 23

Extracción del contenido estructurado de la página web

```
async extractAllPageData(page, url) {
    // EXTRACCIÓN COMPLETAMENTE EXHAUSTIVA
    const data = await page.evaluate(() => {
      return {
        // Metadatos completos
        title: document.title || "",
        metaDescription:
          document.querySelector('meta[name="description"]')?.
content || "",
        metaKeywords:
          document.querySelector('meta[name="keywords"]')?.
content || "",
        metaAuthor:
          document.querySelector('meta[name="author"]')?.content
 || "",
        metaRobots:
          document.querySelector('meta[name="robots"]')?.content
 || "",
        ogTitle:
          document.querySelector('meta[property="og:title"]')?.
content || "",
        ogDescription:
          document.querySelector(
'meta[property="og:description"]')?.content ||
          "",

        // Estructura completa de encabezados
        h1: Array.from(document.querySelectorAll("h1")).map((h)
=>
          h.textContent.trim()
        ),
        h2: Array.from(document.querySelectorAll("h2")).map((h)
=>
          h.textContent.trim()
        ),
        h3: Array.from(document.querySelectorAll("h3")).map((h)
=>
          h.textContent.trim()
        ),
        h4: Array.from(document.querySelectorAll("h4")).map((h)
=>
          h.textContent.trim()
        ),
        h5: Array.from(document.querySelectorAll("h5")).map((h)
=>
          h.textContent.trim()
        ),
        h6: Array.from(document.querySelectorAll("h6")).map((h)
=>
          h.textContent.trim()
        ),
```

Nota. Elaboración propia.

La extracción de datos que implementé hace un análisis completo de la estructura DOM que captura metadatos, contenido de texto, elementos multimedia y datos estructurados. El sistema extrae información académica específica usando selectores especializados que identifican programas de estudio, requisitos de admisión, costos y horarios, organizando toda esta información en estructuras de datos que tienen sentido para que después las pueda procesar el sistema RAG.

Procesamiento de documentos PDF.

Figura 24

Procesamiento de documentos pdf

```
async extractFromPDFExhaustive(buffer) {
    try {
      const data = await pdf(buffer, {
        // Opciones exhaustivas para PDF
        max: 0, // Sin límite de páginas
        version: "v1.10.100",
      });

      let fullText = data.text || "";

      // Si el PDF tiene metadatos, incluirlos
      if (data.info) {
        const metadata = [];
        if (data.info.Title) metadata.push(`TÍTULO PDF: ${data.
info.Title}`);
        if (data.info.Author) metadata.push(`AUTOR: ${data.info.
Author}`);
        if (data.info.Subject) metadata.push(`TEMA: ${data.info.
Subject}`);
        if (data.info.Creator) metadata.push(`CREADOR: ${data.
info.Creator}`);

        if (metadata.length > 0) {
          fullText = metadata.join("\n") + "\n\n" + fullText;
        }
      }

      return fullText || "PDF procesado - sin texto extraíble";
    } catch (error) {
      console.error("Error extrayendo PDF exhaustivo:", error);
      return "Error al procesar PDF exhaustivamente";
    }
  }
```

Nota. Elaboración propia.

El módulo que extrae archivos PDF usa la biblioteca pdf-parse para sacar texto y metadatos de documentos académicos. Lo que implementé procesa documentos sin importar cuántas páginas tengan, mantiene metadatos como título, autor y tema, y aplica técnicas de limpieza de texto que preparan el contenido para que después se pueda indexar bien en el sistema de búsqueda vectorial FAISS.

Algoritmo de limpieza y normalización.

El proceso de limpieza de texto que hice usa algoritmos de normalización que quitan caracteres de control, unifican saltos de línea y arreglan el espaciado para que después se pueda trabajar mejor con el texto. Esta parte incluye filtrar contenido que no es académico, separar palabras que están pegadas y mantener la estructura semántica usando técnicas de expresiones regulares especializadas.

Figura 25

Limpieza de datos

```
cleanExtractedText(text) {
   if (!text) return "";

   return text
     .replace(/\f/g, "\n") // Form feeds
     .replace(/\r\n/g, "\n") // CRLF
     .replace(/\r/g, "\n") // CR
     .replace(/\n{4,}/g, "\n\n\n") // Múltiples saltos
     .replace(/\s{3,}/g, "  ") // Múltiples espacios
     .replace(/([a-z])([A-Z])/g, "$1 $2") // Separar palabras
     .replace(/^\s+|\s+$/gm, "") // Espacios de línea
     .trim();
 }
```

Nota. Elaboración propia.

Resultados del web scraping completo

El proceso de web scraping completo del sitio web de la Universidad Internacional del Ecuador Campus Loja funcionó muy bien para recopilar toda la información de la institución. La extracción me tomó 3.11 horas en total, procesé 2,677 páginas web y descargué 358 archivos de diferentes formatos académicos, generando un total de 267 MB de contenido organizado.

El sistema encontró 347,932 URLs durante el proceso de crawling, lo que muestra qué tan profundo y amplio fue el trabajo que hice. La tasa de éxito del 92% indica que el algoritmo de extracción funcionó muy bien, con muy poca pérdida de datos por problemas de conexión o restricciones de acceso.

Juntando todo el contenido para RAG

Todo el contenido que saqué lo consolidé en un solo archivo de texto (UIDE_CONTENIDO_ABSOLUTO.txt) que integra toda la información recopilada en un formato optimizado para que después lo pueda procesar el sistema RAG.

- **Contenido textual estructurado** de 2,677 páginas web institucionales.
- **Documentos PDF procesados** con extracción completa de texto y metadatos.
- **Información académica específica** identificada mediante selectores especializados.
- **Metadatos de fuente y timestamp** para trazabilidad del contenido.

Eficiencia del Proceso

La velocidad promedio de procesamiento alcanzó **860 páginas por hora**, demostrando un balance óptimo entre exhaustividad de extracción y eficiencia temporal. El volumen de datos generado (267 MB) proporciona una base de conocimiento substancial que alimentará efectivamente el sistema de recuperación de información del chatbot universitario.

La implementación exitosa de este proceso de web scraping me dio la base para construir una base de conocimiento completa y actualizada que mejora la precisión y relevancia de las

respuestas que genera el sistema RAG cuando la gente pregunta sobre información académica, administrativa y de servicios de la Universidad Internacional del Ecuador Campus Loja.

Desarrollo del backend del sistema.

El backend que implementé usa un sistema de autenticación híbrida que conecta Firebase Authentication con Django REST Framework mediante una clase de autenticación que hice yo mismo. Esta arquitectura me permite validar tokens JWT de Firebase directamente en el servidor Django, dando un mecanismo de seguridad sólido que mantiene la coherencia entre el frontend y el backend sin afectar la escalabilidad del sistema.

Figura 26

Módulo de autentificación

```python
class FirebaseAuthentication(BaseAuthentication):
    """ Autenticación personalizada para Firebase """
    def authenticate(self, request):
        user = getattr(request, "_user", None)
        if user and user.is_authenticated():
            return (user, None)
        return None
```

Nota. Elaboración propia.

API endpoint principal del chatbot.

La vista principal del chatbot que implementé es un endpoint RESTful que recibe preguntas de usuarios autenticados y las procesa a través del sistema RAG. El endpoint incluye

validaciones de autenticación, limpieza de lo que escribe el usuario y manejo de errores, asegurando que solo usuarios verificados puedan acceder al sistema de consultas y que las respuestas se devuelvan en formato JSON estándar para que el frontend las pueda usar.

Figura 27

ENDPOINT principal del chatbot

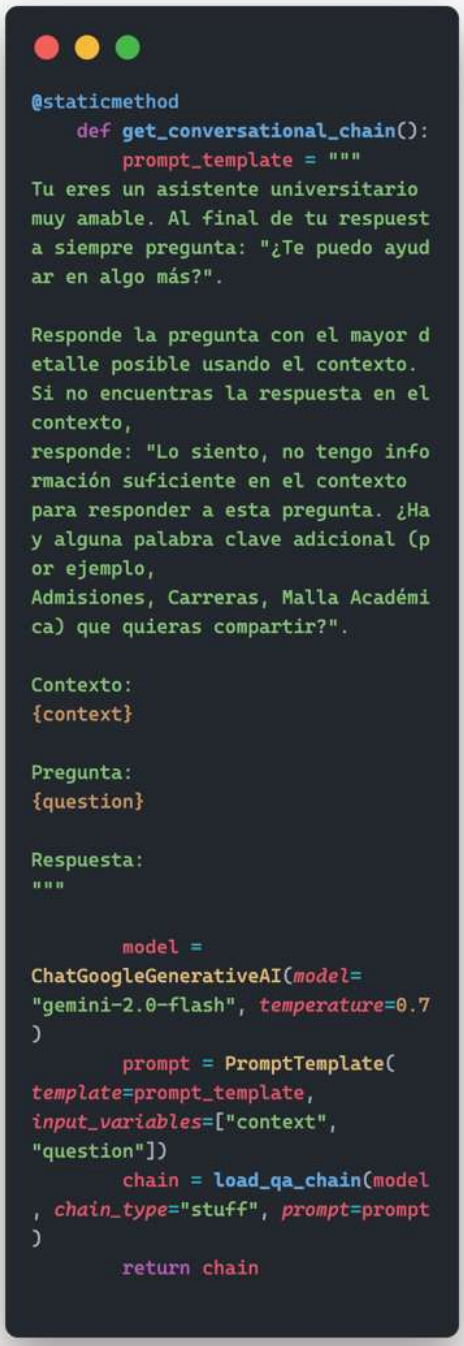
```python
class ChatbotView(APIView):
    authentication_classes = [FirebaseAuthentication]
    permission_classes = [IsAuthenticated]

    def post(self, request):
        print(
"🟠 Usuario en la vista después de autenticació   , request
nuser)

        if not request.user or not request.user.
is_authenticated():
            return Response({'error': 'No autorizado'},
status=status.HTTP_401_UNAUTHORIZED)

        user_message = request.data.get('message')
        if not user_message:
            return Response({'error': 'Message not provided'
}, status=status.HTTP_400_BAD_REQUEST)

        response = ChatbotResponse.get_chatbot_response(
user_message)


# 🔥 Asegurar que se devuelve solo texto, no un objet
o       if isinstance(response, dict) and "output_text" in
response:
            response = response["output_text"]

        return Response({'response': response})
```

Nota. Elaboración propia.

Sistema de plantillas de prompts para RAG.

Figura 28

Plantilla de prompt para RAG



```
@staticmethod
    def get_conversational_chain():
        prompt_template = """
Tu eres un asistente universitario
muy amable. Al final de tu respuest
a siempre pregunta: "¿Te puedo ayud
ar en algo más?".

Responde la pregunta con el mayor d
etalle posible usando el contexto.
Si no encuentras la respuesta en el
contexto,
responde: "Lo siento, no tengo info
rmación suficiente en el contexto
para responder a esta pregunta. ¿Ha
y alguna palabra clave adicional (p
or ejemplo,
Admisiones, Carreras, Malla Académi
ca) que quieras compartir?".

Contexto:
{context}

Pregunta:
{question}

Respuesta:
"""

        model =
ChatGoogleGenerativeAI(model=
"gemini-2.0-flash", temperature=0.7
)
        prompt = PromptTemplate(
template=prompt_template,
input_variables=["context",
"question"])
        chain = load_qa_chain(model
, chain_type="stuff", prompt=prompt
)
        return chain
```

Nota. Elaboración propia.

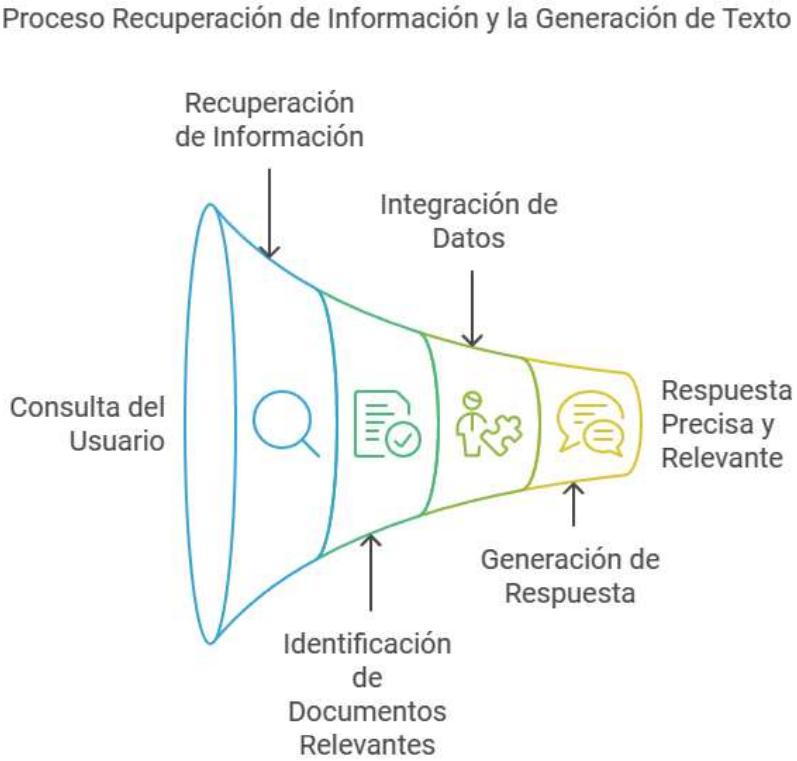
El sistema de generación de respuestas utiliza plantillas de prompts estructuradas que integran el contexto recuperado con la consulta del usuario. La plantilla implementa instrucciones específicas para el modelo de lenguaje que garantizan respuestas contextualizadas, mantienen un tono universitario apropiado y proporcionan fallbacks informativos cuando el contexto no es suficiente para responder la consulta.

Pipeline de procesamiento RAG.

El corazón o núcleo del sistema RAG contiene un pipeline de procesamiento que combina búsqueda vectorial semántica con generación de lenguaje natural. El proceso incluye carga de embeddings pre-entrenados, búsqueda de similitud en la base vectorial FAISS, cálculo de métricas de relevancia mediante similitud coseno y generación de respuestas contextualizadas a través del modelo de lenguaje. El sistema incorpora logging detallado para monitoreo de performance y debugging.

Figura 29

Proceso RAG



Nota. Elaboración propia.

Figura 30

Pipeline RAG

```
@staticmethod
    def get_chatbot_response(user_message):

        user_message_lower = user_message.strip().lower()
        if user_message_lower in ["hola", "buenos días", "buenas tardes",
"buenas noches"]:
        # Respuesta de cortesía
            return "¡Hola! ¿En qué puedo ayudarte hoy?"
        if user_message_lower in ["gracias", "muchas gracias",
"gracias por tu ayuda"]:
        # Respuesta de despedida
            return "¡Fue un gusto ayudarte! ¡Hasta luego!"
        # 1. Cargamos la BD vectorial
        embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
        db_path = "vector_db"
        vector_store = FAISS.load_local(db_path, embeddings,
allow_dangerous_deserialization=True)


        docs = vector_store.similarity_search(user_message, k=4)

        print("📚 Documentos recuperados del vector stor  )
        for i,edõc in enumerate(docs):
            print(f"\n📄 Documento {i+1}:\n{doc.page_content[:500]} ... ")

        total_words = sum(len(doc.page_content.split()) for doc in docs)
        avg_words = total_words / len(docs) if docs else 0
        print(f"🔢 Total de palabras en el contexto: {total_words}")
        print(f"📊 Promedio de palabras por documento: {avg_words:.2f}")


        query_embedding = embeddings.embed_query(user_message)
        print("🔍 Similitudes coseno con cada document  )
        for i,odõc in enumerate(docs):
            doc_embedding = embeddings.embed_query(doc.page_content)
            similarity = 1 - cosine(query_embedding, doc_embedding)
        print(f"📌 Documento {i+1} → Similitud coseno: {similarity:.4f}")

        # 3. Obtener el chain y ejecutar
        chain = ChatbotResponse.get_conversational_chain()
        reply_response = chain(
            {"input_documents": docs, "question": user_message},
            return_only_outputs=True
        )


        respuesta = reply_response["output_text"]
        print(f"\n🧠 Respuesta generad  \n{respuesta}")
                  a:

        usó_contexto = "no tengo información suficiente" not in respuesta.lower()
        print(f"✅ ¿Se usó el contexto?: {'Sí' if usó_contexto else 'No'}")


        return reply_response
```

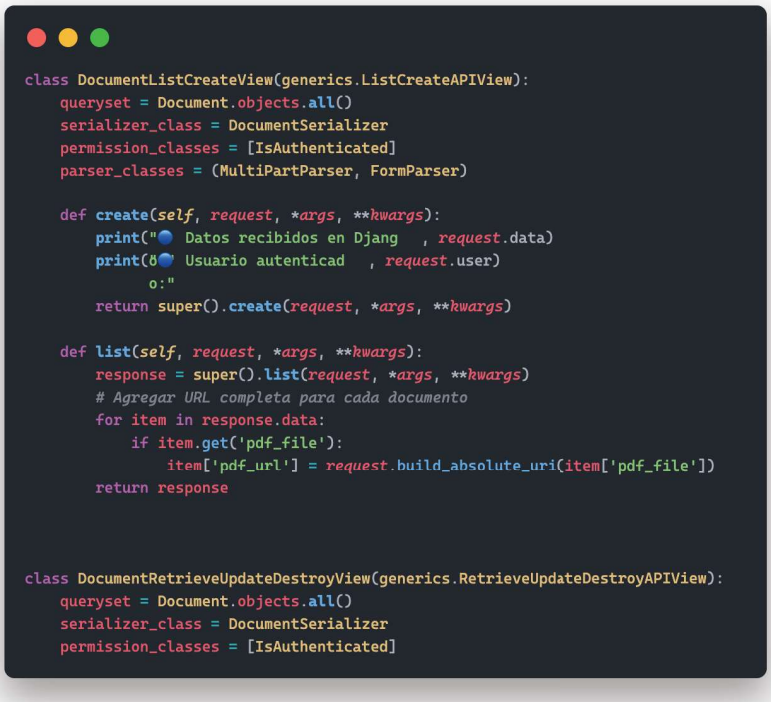
Nota. Elaboración propia.

API's de gestión de documentos.

El sistema incluye APIs especializadas para manejar documentos PDF que alimentan la base de conocimiento del chatbot. Las vistas que implementé tienen operaciones CRUD completas con autenticación requerida, soporte para subir archivos multipart y generación automática de URLs absolutas para acceder a documentos. Esta arquitectura hace más fácil administrar el contenido de manera dinámica sin interrumpir el servicio del chatbot.

Figura 31

Gestión de documentos



```python
class DocumentListCreateView(generics.ListCreateAPIView):
    queryset = Document.objects.all()
    serializer_class = DocumentSerializer
    permission_classes = [IsAuthenticated]
    parser_classes = (MultiPartParser, FormParser)

    def create(self, request, *args, **kwargs):
        print("🔵 Datos recibidos en Djang   , request.data)
        print(f"🔵 Usuario autenticad   , request.user)
              o:"
        return super().create(request, *args, **kwargs)

    def list(self, request, *args, **kwargs):
        response = super().list(request, *args, **kwargs)
        # Agregar URL completa para cada documento
        for item in response.data:
            if item.get('pdf_file'):
                item['pdf_url'] = request.build_absolute_uri(item['pdf_file'])
        return response


class DocumentRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
    queryset = Document.objects.all()
    serializer_class = DocumentSerializer
    permission_classes = [IsAuthenticated]
```

Nota. Elaboración propia.

Sistema de logging y métricas RAG.

La implementación que hice incluye un sistema completo de logging que registra métricas clave del proceso RAG, incluyendo documentos recuperados, palabras totales en el contexto, similitudes coseno calculadas y qué tan efectivo es el uso del contexto. Este sistema de telemetría

me permite optimizar continuamente el rendimiento y analizar la calidad de las respuestas generadas. Las líneas de código que muestran las métricas están en la Figura 26.

Desarrollo de la interfaz cliente (Next.js).

El frontend del sistema lo desarrollé usando Next.js como framework principal, conectando Firebase para la autenticación de usuarios y manejo de sesiones. Diseñé la interfaz priorizando la experiencia del usuario, accesibilidad y funcionalidad responsiva, implementando un chatbot basado en RAG (Retrieval Augmented Generation) específicamente orientado al contexto universitario.

Sistema de autenticación y Gestión de usuarios.

El sistema de autenticación que implementé se estructura en múltiples interfaces que garantizan un acceso seguro y una experiencia de usuario fluida.

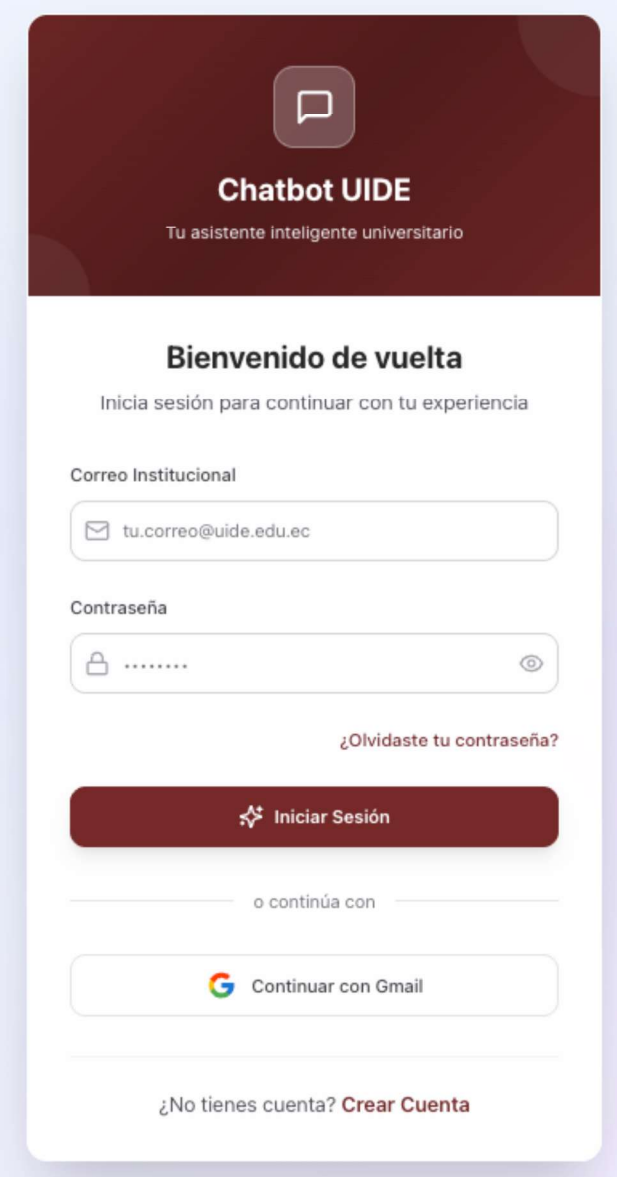
Interfaz de inicio de sesión:

La pantalla principal de acceso presenta un diseño limpio y profesional con la identidad visual de la institución. Los elementos principales incluyen:

- **Encabezado institucional:** Presenta el logotipo y nombre "Chatbot UIDE" con el subtítulo "Tu asistente inteligente universitario", estableciendo claramente el propósito del sistema.
- **Formulario de autenticación:** Incluye campos para correo institucional y contraseña, con validación en tiempo real y mensajes de error contextual.
- **Opciones de acceso alternativo:** Integración con Firebase Auth para permitir autenticación mediante Google, facilitando el acceso con credenciales institucionales.
- **Funcionalidades auxiliares:** Enlaces para recuperación de contraseña y registro de nuevos usuarios.

Figura 32

Interfaz de inicio de sesión del sistema



Nota. Elaboración propia.

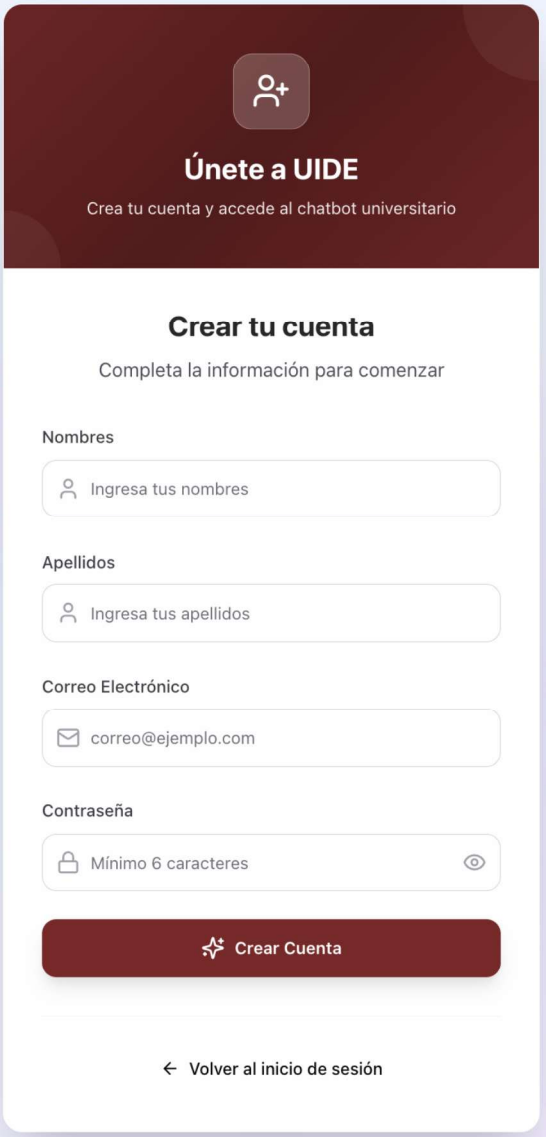
Registro de nuevo usuario:

El proceso de registro contiene un formulario estructurado que alamacena la información del usuario:

- **Campos de información personal:** Nombres, apellidos y correo electrónico con validación de formato institucional.
- **Configuración de seguridad:** Campo de contraseña con requisitos de complejidad claramente especificados.
- **Navegación intuitiva:** Botón de retorno al inicio de sesión para usuarios existentes.

Figura 33

Pantalla de registro de usuarios



Nota. Elaboración propia.

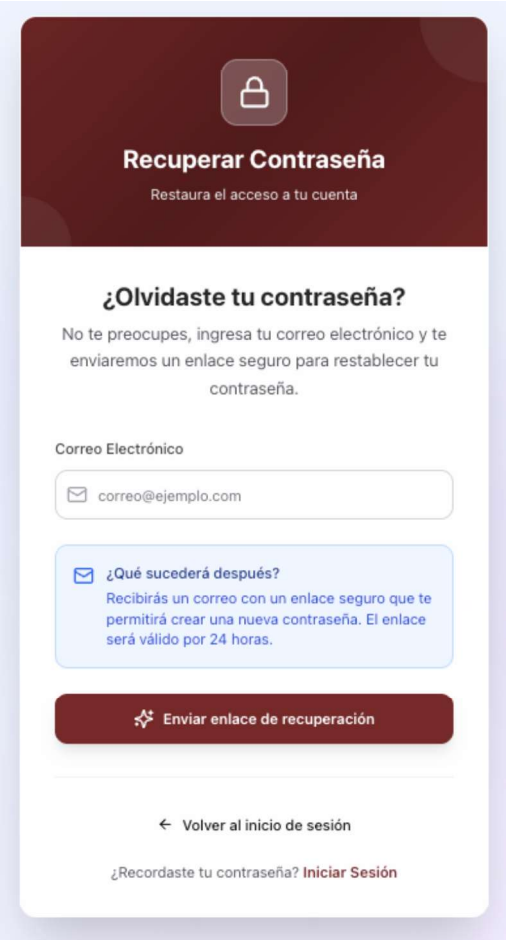
Sistema de recuperación de contraseñas:

La funcionalidad de recuperación que implementé usa un proceso seguro y fácil de usar:

- **Interfaz simplificada:** Un solo campo para escribir el correo electrónico.
- **Información contextual:** Explicación clara del proceso de recuperación y tiempo de validez del enlace (24 horas).
- **Retroalimentación visual:** Confirmación inmediata del envío del enlace de recuperación.

Figura 34

Interfaz de para recuperar contraseña



Nota. Elaboración propia.

Interfaz principal del chatbot

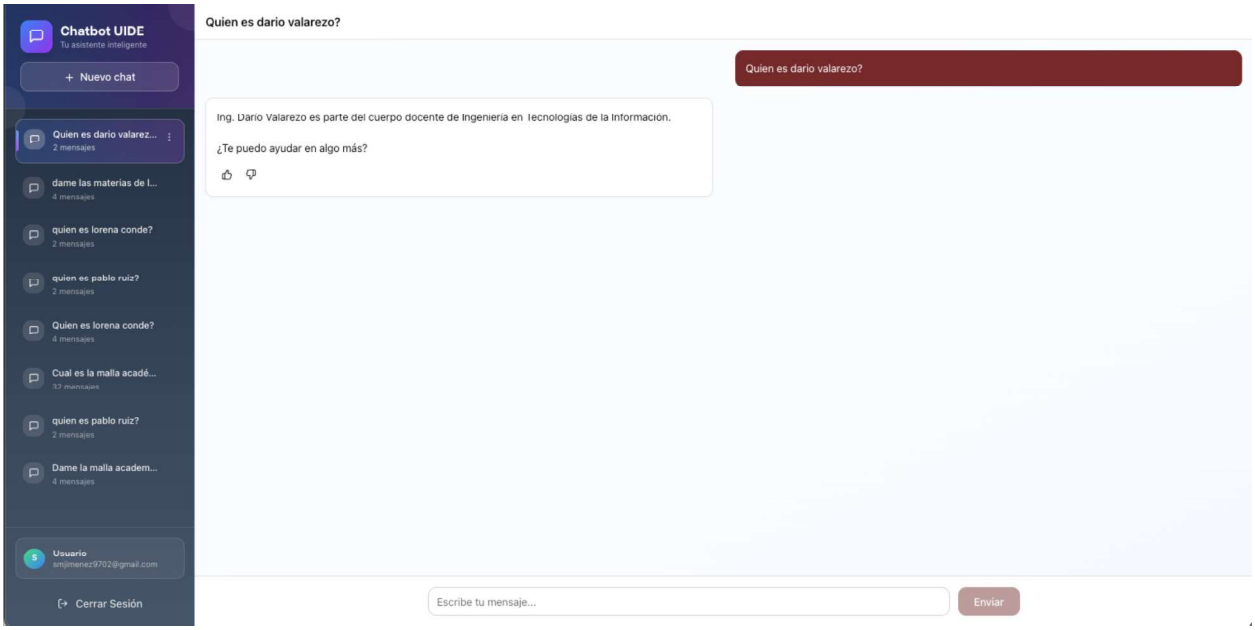
La interfaz principal es el corazón funcional del sistema, la optimicé para conversaciones naturales y acceso eficiente al conocimiento universitario.

Elementos de la interfaz conversacional:

- **Panel lateral de navegación:** Lista organizada de conversaciones anteriores, lo que permite continuidad y recuperar el contexto entre sesiones. Incluye la función de "Nuevo chat" para empezar conversaciones desde cero.
- **Área de conversación principal:** Diseño limpio que muestra el intercambio entre usuario y asistente IA, con diferenciación visual clara entre mensajes enviados y respuestas del sistema.
- **Campo de entrada de texto:** Input responsivo con botón de envío, optimizado para consultas universitarias de diversa complejidad.
- **Sistema de feedback:** Implementación de botones de evaluación (like/dislike) para cada respuesta, permitiendo la mejora continua del modelo RAG.

Figura 35

Interfaz principal del chatbot



Nota. Elaboración propia.

Características de usabilidad implementadas:

- **Persistencia de conversaciones:** Almacenamiento en Firebase de todo el historial de chat del usuario.
- **Búsqueda contextual:** Integración RAG que permite respuestas basadas en documentación universitaria específica.
- **Diseño responsivo:** Se adapta automáticamente a diferentes tamaños de pantalla manteniendo toda la funcionalidad.

Panel Administrativo

El sistema incluye un completo panel administrativo que permite la gestión, monitoreo y optimización del chatbot universitario.

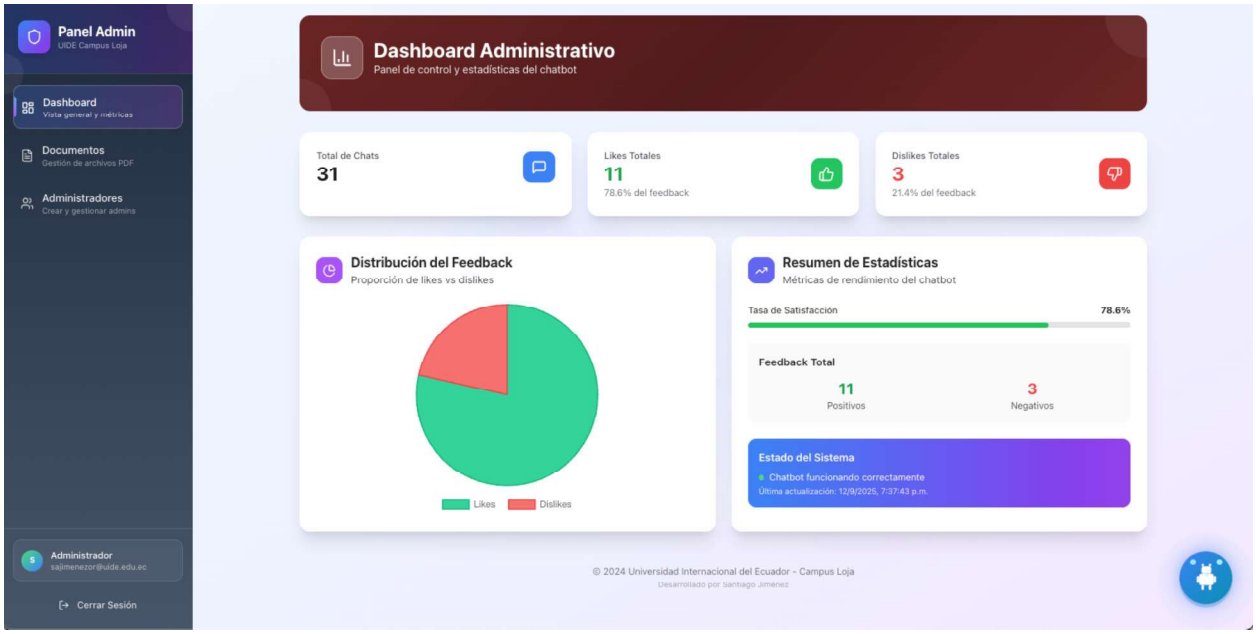
Dashboard de control y estadísticas:

La interfaz administrativa proporciona una visión integral del rendimiento del sistema:

- **Métricas principales:** Visualización de KPIs incluyendo total de conversaciones (31), likes totales (11) y dislikes totales (3), con cálculo automático de tasa de satisfacción (78.6%).
- **Análisis de feedback:** Gráfico circular que representa la distribución proporcional de evaluaciones positivas y negativas, facilitando la identificación de patrones de satisfacción.
- **Monitoreo del sistema:** Panel de estado en tiempo real que confirma el funcionamiento correcto del chatbot con timestamp de última actualización.

Figura 36

Dashboard administrativo con métricas de rendimiento



Nota. Elaboración propia.

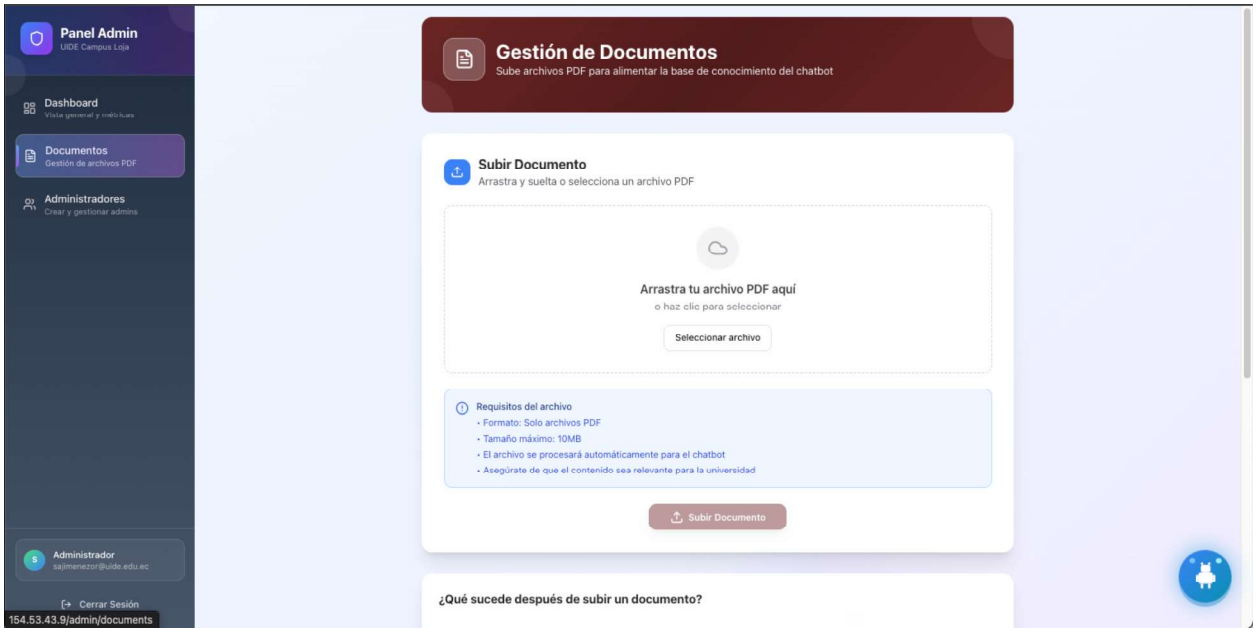
Gestión de documentos RAG.

El módulo de gestión documental permite la actualización y expansión de la base de conocimiento del chatbot:

- **Interfaz de carga de archivos:** Sistema drag-and-drop optimizado para documentos PDF con validación de formato y tamaño (máximo 10MB).
- **Procesamiento automático:** Integración que procesa automáticamente los documentos subidos para incorporarlos al sistema RAG.
- **Requisitos y validaciones:** Especificaciones claras sobre formatos aceptados y relevancia del contenido universitario.

Figura 37

Interfaz de gestión de documentos PDF



Nota. Elaboración propia.

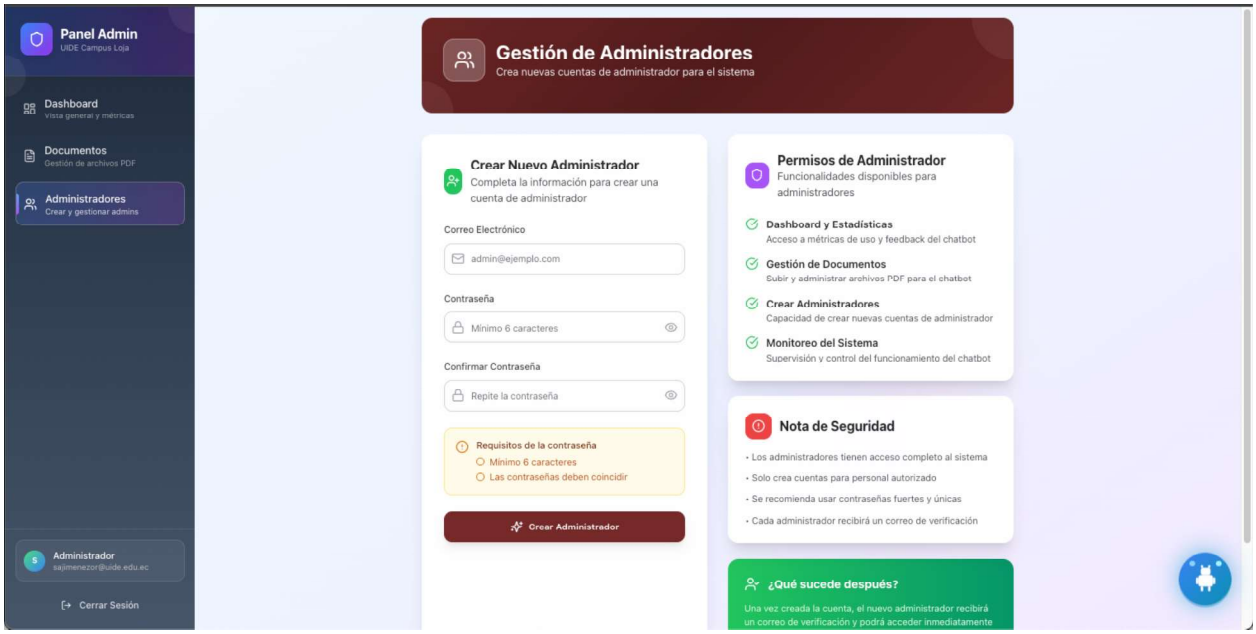
Administración de usuarios del sistema.

El módulo de gestión de administradores implementa controles de acceso granulares:

- **Creación de cuentas administrativas:** Formulario seguro para registro de nuevos administradores con validación de credenciales.
- **Matriz de permisos:** Visualización clara de funcionalidades disponibles: Dashboard y Estadísticas, Gestión de Documentos, Crear Administradores y Monitoreo del Sistema.
- **Medidas de seguridad:** Implementación de políticas de contraseñas seguras y notificaciones de verificación por correo electrónico.

Figura 38

Panel de gestión de administradores



Nota. Elaboración propia.

Elementos de diseño y experiencia de usuario.

- **Paleta de colores institucional:** Uso consistente de tonos corporativos que refuerzan la identidad universitaria.
- **Iconografía intuitiva:** Implementación de iconos semánticamente apropiados para cada funcionalidad.
- **Navegación coherente:** Menú lateral persistente que facilita el acceso a todas las funcionalidades administrativas.
- **Retroalimentación visual:** Indicadores de estado, mensajes de confirmación y alertas contextual para todas las acciones del usuario.

3.2. Plan de pruebas de implementación.

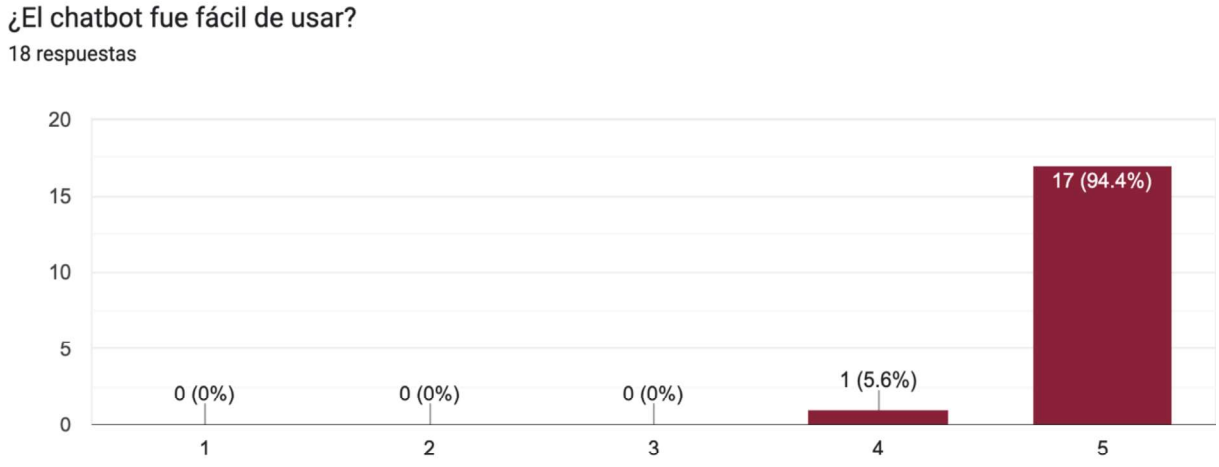
3.2.1. Evaluación de usabilidad del sistema.

La evaluación de usabilidad se realizó mediante la aplicación de una encuesta estructurada a 18 estudiantes de la Universidad Internacional del Ecuador, con el objetivo de medir la percepción de los usuarios respecto a la facilidad de uso, comprensión de la interfaz y nivel de intuición del sistema de chatbot desarrollado.

Facilidad de uso del sistema: La primera métrica evaluada se centró en determinar si los usuarios percibían el chatbot como una herramienta fácil de utilizar. Los resultados demuestran una alta aceptación por parte de los usuarios, con una calificación promedio de 4.94 sobre 5.0. La distribución de respuestas muestra que 17 estudiantes (94.4%) otorgaron la calificación máxima de 5 estrellas, mientras que únicamente 1 estudiante (5.6%) asignó una calificación de 4 estrellas.

Figura 39

Evaluación de facilidad de uso del chatbot universitario

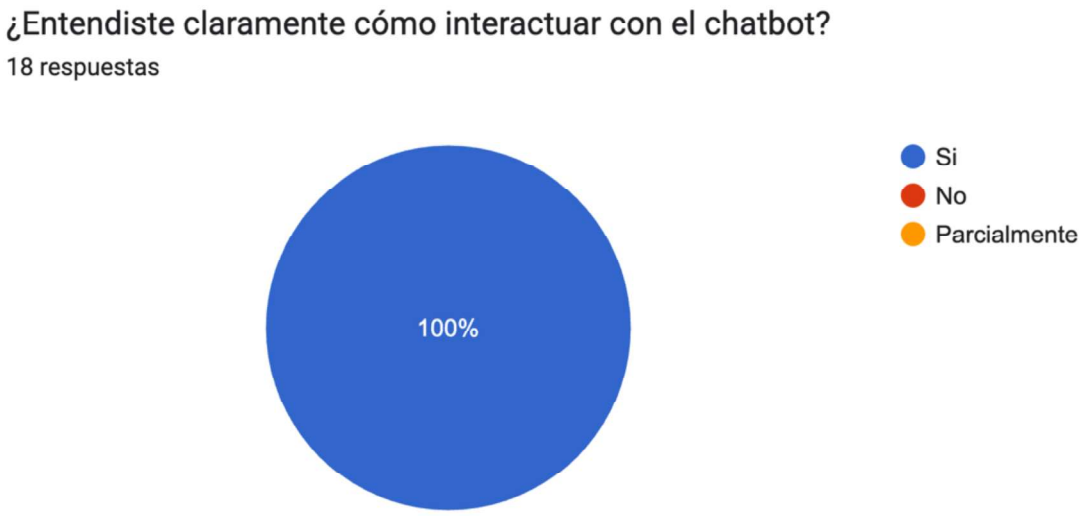


Nota. Elaboración propia.

Claridad en la interacción: El segundo aspecto que evalué analizó si los usuarios entendían claramente cómo interactuar con el chatbot. Los resultados muestran que todos los participantes lo comprendieron perfectamente, con un 100% de respuestas afirmativas. Este resultado indica que la interfaz logra comunicar de manera efectiva qué funciones están disponibles y cómo interactuar con el sistema.

Figura 40

Nivel de comprensión de la interacción con el chatbot

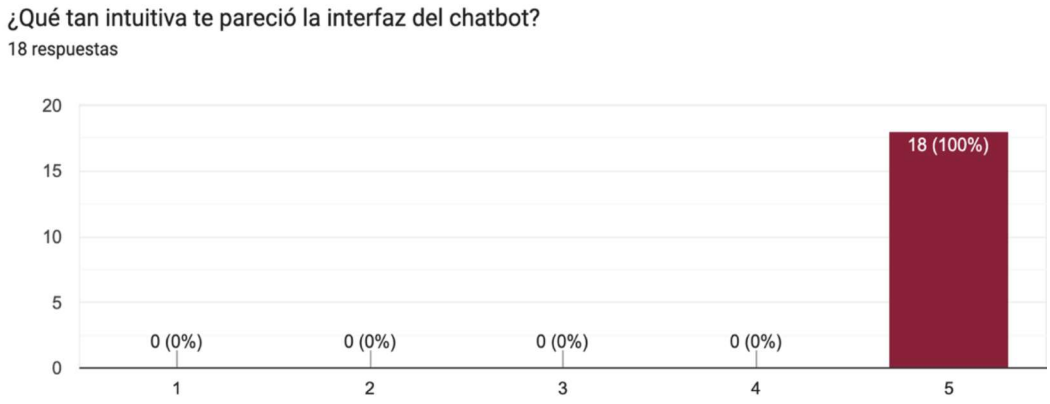


Nota. Elaboración propia.

Intuitividad de la interfaz: La tercera métrica se enfocó en evaluar qué tan intuitiva resulta la interfaz del chatbot para los usuarios. Esta evaluación obtuvo los mejores resultados, con una calificación promedio perfecta de 5.0. La totalidad de los 18 participantes (100%) asignó la calificación máxima de 5 estrellas, demostrando que la interfaz cumple completamente con los principios de diseño intuitivo y user experience.

Figura 41

Evaluación de intuitividad de la interfaz del chatbot



Nota. Elaboración propia.

Tabla comparativa de resultados de Usabilidad.

Tabla 33

Tabla de resultados de usabilidad

Aspecto Evaluado	Calificación Promedio	Distribución de Respuestas	Nivel de Aceptación
Facilidad de uso	4.94/5.0	• 5 estrellas: 17 usuarios (94.4%) • 4 estrellas: 1 usuario (5.6%) • 3 estrellas o menos: 0 usuarios (0%)	Muy Alto
Claridad de interacción	N/A*	• Sí: 18 usuarios (100%) • No: 0 usuarios (0%) • Parcialmente: 0 usuarios (0%)	Absoluto
Intuitividad de interfaz	5.0/5.0	• 5 estrellas: 18 usuarios (100%) • 4 estrellas o menos: 0 usuarios (0%)	Máximo

Nota. Elaboración propia.

Los resultados de la evaluación de usabilidad revelan que el sistema de chatbot universitario cumple exitosamente con los estándares de diseño centrado en el usuario. La consistencia en las altas calificaciones across todas las métricas evaluadas (facilidad de uso: 4.94/5.0, claridad de interacción: 100%, intuitividad: 5.0/5.0) demuestra que la interfaz desarrollada en Next.js logra una experiencia de usuario óptima.

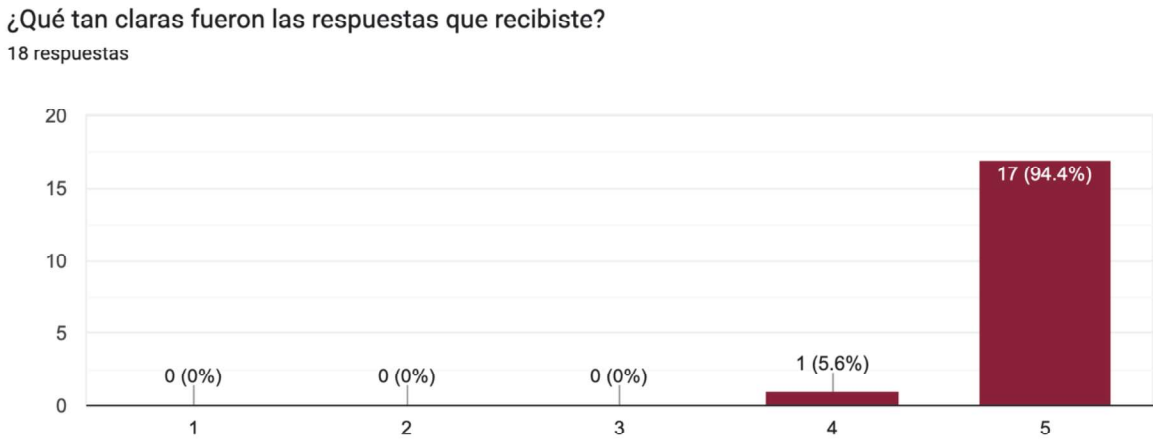
La ausencia de calificaciones negativas o neutras en cualquiera de las tres dimensiones evaluadas sugiere que el diseño de la interfaz, la arquitectura de la información y los flujos de interacción implementados se alinean efectivamente con las expectativas y modelos mentales de los usuarios universitarios. Este nivel de aceptación facilita la adopción del sistema y potencia su utilidad como herramienta de apoyo académico.

3.2.2. Evaluación de calidad de Respuestas.

Para la segunda parte revisé qué tal eran las respuestas que daba el sistema RAG. Aquí miré tres cosas básicas: si las respuestas se entendían bien, si eran precisas y relevantes, y qué tanto detalle daba el chatbot.

Figura 42

Evaluación de claridad de las respuestas



Nota. Elaboración propia.

Cuando analicé qué tan precisas y relevantes eran las respuestas del sistema RAG, los resultados fueron muy buenos. El 94.4% de los participantes (17 estudiantes) dijeron que las respuestas fueron "siempre correctas" y relevantes para sus preguntas específicas. El 5.6% que quedaba (1 estudiante) consideró que las respuestas fueron "correctas la mayoría de las veces". Nadie marcó las opciones de "a veces correctas" o "incorrectas", lo que muestra que el modelo de recuperación de información que implementé funciona muy bien y que la base de conocimiento universitario que integré tiene buena calidad.

Figura 43

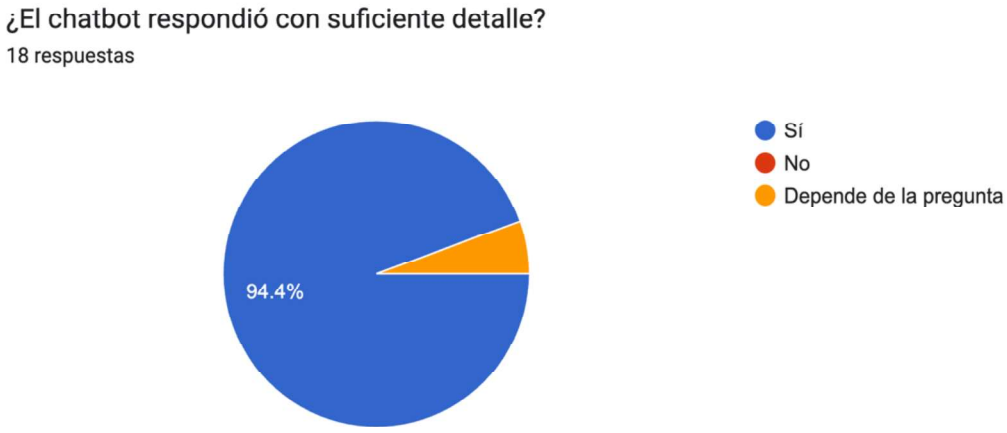
Evaluación de precisión y relevancia de las respuestas

Nota. Elaboración propia.

Nivel de detalle en las respuestas: Cuando evalué qué tanto detalle daba el chatbot, los usuarios quedaron bastante satisfechos. El 94.4% de los participantes (17 estudiantes) confirmaron que el chatbot les respondía con suficiente detalle para cubrir lo que necesitaban saber. El 5.6% que quedaba (1 estudiante) dijo que el nivel de detalle "depende de la pregunta", lo que sugiere que ciertos tipos de consultas podrían necesitar respuestas más elaboradas. Ningún participante consideró que las respuestas fueran demasiado cortas o que les faltara información.

Figura 44

Evaluación del nivel de detalle en las respuestas



Nota. Elaboración propia.

Tabla Comparativa de Resultados de Calidad de Respuestas

Tabla 34

Comparación de calidad de respuestas

Aspecto Evaluado	Métrica Principal	Distribución de Respuestas	Nivel de Satisfacción
Claridad de respuestas	4.94/5.0	• 5 estrellas: 17 usuarios (94.4%) • 4 estrellas: 1 usuario (5.6%) • 3 estrellas o menos: 0 usuarios (0%)	Muy Alto
Precisión y relevancia	94.4% "Siempre correctas"	• Siempre correctas: 17 usuarios (94.4%) • La mayoría correctas: 1 usuario (5.6%) • A veces correctas: 0 usuarios (0%) • Incorrectas: 0 usuarios (0%)	Excelente
Nivel de detalle	94.4% "Suficiente"	• Sí: 17 usuarios (94.4%) • Depende de la pregunta: 1 usuario (5.6%) • No: 0 usuarios (0%)	Muy Alto

Nota. Elaboración propia.

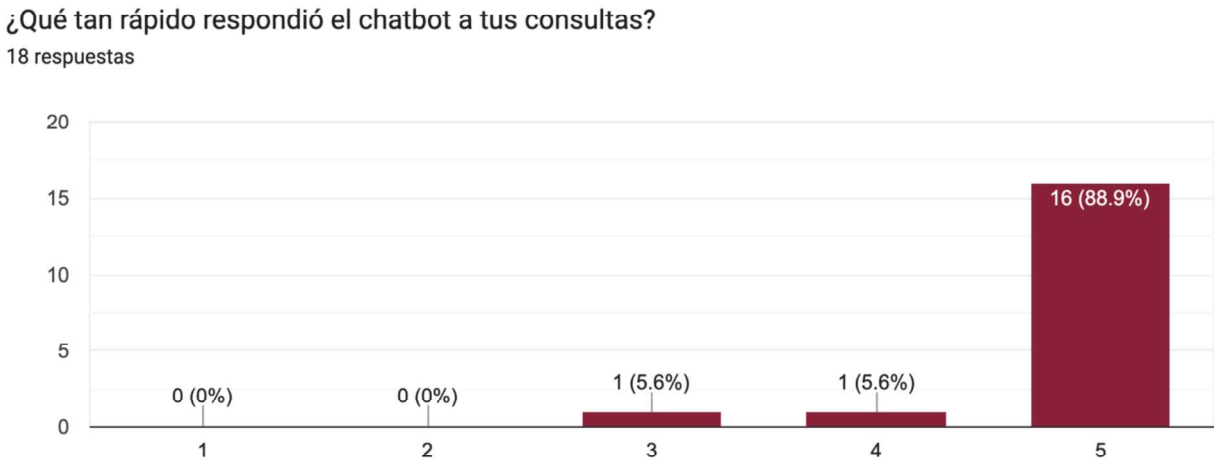
Los resultados de la evaluación de calidad muestran que el sistema RAG que implementé cumple bien con lo que se necesita de precisión, relevancia y claridad para un asistente inteligente universitario. Que las calificaciones hayan sido tan buenas y consistentes en todas las áreas que evalué (claridad: 4.94/5.0, precisión: 94.4% siempre correctas, detalle: 94.4% suficiente) confirma que la arquitectura de recuperación aumentada que implementé funciona.

Que no haya habido respuestas negativas en ninguna de las métricas de calidad indica que el proceso de indexación de documentos universitarios, la estrategia de embeddings que usé y los algoritmos de similitud semántica funcionan muy bien para el contexto académico específico. El hecho de que solo un usuario haya mencionado variabilidad en la percepción de detalle sugiere que hay oportunidades para mejorar la personalización de respuestas según el tipo de consulta, manteniendo el alto estándar de calidad que ya logré.

Evaluación de desempeño Técnico.

Figura 45

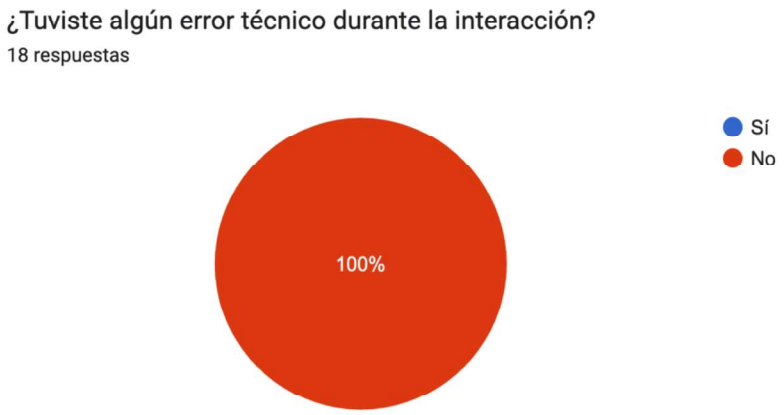
Evaluación de velocidad de respuesta del chatbot



Nota. Elaboración propia.

Figura 46

Incidencia de errores técnicos durante la interacción



Nota. Elaboración propia.

Estabilidad técnica del sistema: El análisis de la estabilidad técnica mostró resultados excepcionales, con el 100% de los participantes (18 estudiantes) reportando que no experimentaron ningún error técnico durante su interacción con el chatbot.

Para la tercera parte me centré específicamente en analizar cómo funcionaba el sistema a nivel técnico. Aquí observé dos factores significativos para que la gente tuviera una buena experiencia para con el sistema: qué tan rápido contestaba el chatbot y si el chatbot se mantenía estable mientras que la gente estaba hablando con él.

Al revisar qué tan rápida era la respuesta del sistema, me llevé una grata sorpresa. Contestó un promedio de 4.83 sobre 5.0. De los 18 estudiantes, 16 (88.9%) dijeron que el chatbot era ultra rápido para contestar y le dieron 5 estrellas. Un estudiante (5.6%) le puso 4 estrellas y un estudiante (5.6%) le dio 3 estrellas. Nadie le pudo 1 o 2 estrellas, lo que me dice que el sistema está respetando bien lo que la gente universitaria espera respecto a rapidez.

Tabla Comparativa de Resultados de Desempeño Técnico

Tabla 35

Tabla comparativa de resultados de desempeño técnico

Aspecto Evaluado	Métrica Principal	Distribución de Respuestas	Nivel de Rendimiento
Velocidad de respuesta	4.83/5.0	• 5 estrellas: 16 usuarios (88.9%) • 4 estrellas: 1 usuario (5.6%) • 3 estrellas: 1 usuario (5.6%) • 2 estrellas o menos: 0 usuarios (0%)	Muy Alto
Estabilidad técnica	0% errores reportados	• No: 18 usuarios (100%) • Sí: 0 usuarios (0%)	Excelente

Nota. Elaboración propia.

Los resultados de la evaluación de desempeño técnico demuestran que el sistema cumple exitosamente con los requisitos de rendimiento y estabilidad necesarios para un entorno académico demandante. La calificación obtenida en la velocidad de respuesta (4.83 sobre una máxima de 5.0) indica que, gracias a la arquitectura de Next.js, junto con la optimización del pipeline RAG y de la infraestructura de Firebase, el sistema proporciona unos tiempos de respuesta que satisfacen las necesidades de inmediatez de usuarios universitarios, que exigen tiempos de respuesta inmediatos.

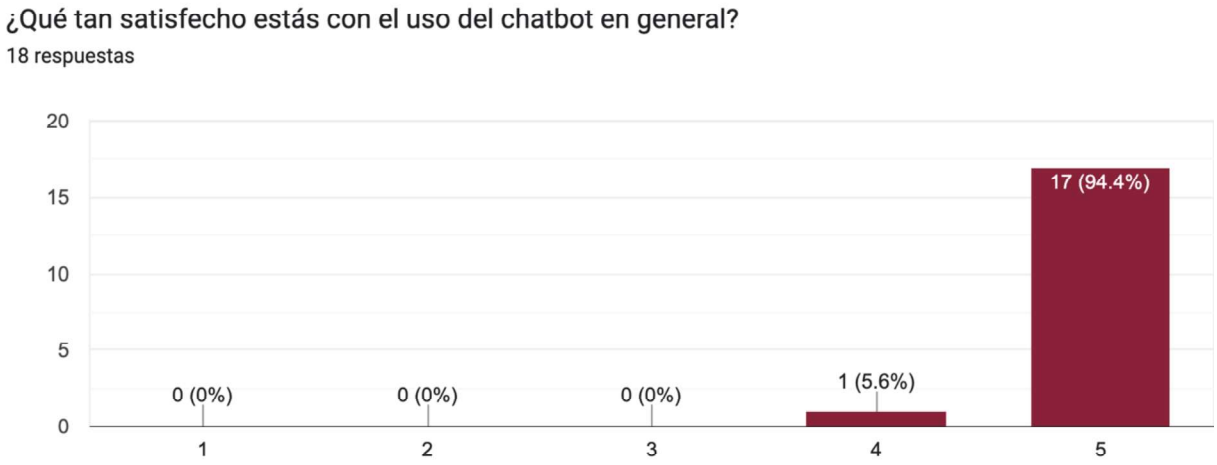
Que el sistema no presente errores técnicos (100% de usuarios sin incidencias) avala la robustez del sistema implementado y garantiza que los diferentes componentes tecnológicos (frontend, autenticación, base de datos de vectores y modelo de lenguaje) funcionan de manera estable y fiable. Un nivel de estabilidad que resulta clave para mantener la confianza de los usuarios y para garantizar un esquema de disponibilidad en un contexto institucional donde la disponibilidad es crítica.

La combinación de una alta velocidad de respuesta y una estabilidad técnica perfecta da pie a que el sistema se pueda considerar como una solución técnica sólida y lista para su despliegue de forma masiva en la infraestructura universitaria.

Evaluación de satisfacción general.

Figura 47

Evaluación de satisfacción general con el chatbot universitario



Nota. Elaboración propia.

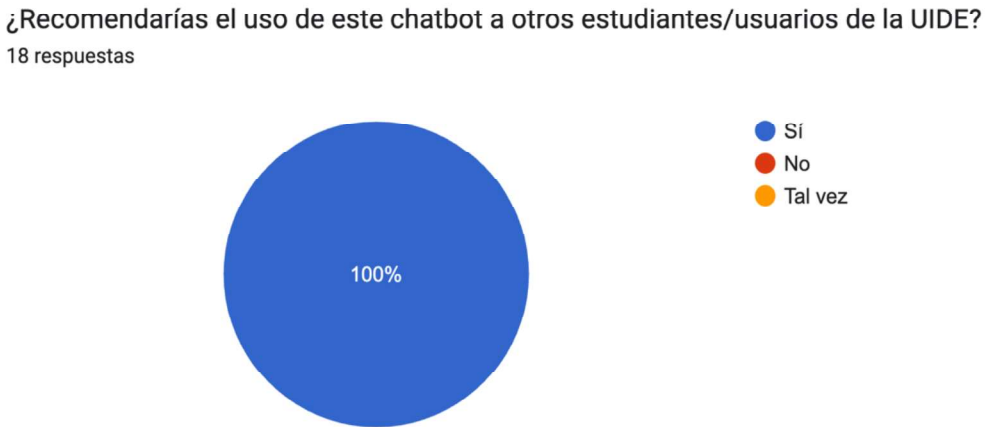
La cuarta dimensión evaluada estaba centrada en la satisfacción general que tienen los usuarios con el sistema de chatbot universitario y con la intención de recomendar la herramienta al resto de la comunidad académica; este valor constituye una lectura completa acerca de la aceptación del sistema y su posibilidad de ser adoptado como sistema de la propia institución.

El resultado de la evaluación de la satisfacción general fue sobresaliente, con una nota media de 4.94 sobre 5.0; la distribución de la respuesta refleja un nivel de satisfacción notablemente alto, donde la nota máxima fue expresada por 17 estudiantes (94.4%) que otorgaron 5 estrellas al sistema. Solo 1 participante (5.6%) asignó una calificación de 4 estrellas, mientras que no se registraron calificaciones inferiores a 4. Este resultado consolida los hallazgos positivos

de las dimensiones previas y confirma que el chatbot cumple de manera integral con las expectativas y necesidades de los usuarios universitarios.

Figura 48

Intención de recomendación del chatbot a otros usuarios



Nota. Elaboración propia.

El análisis de la disposición de los usuarios para recomendar el chatbot a otros estudiantes reveló una aceptación absoluta del sistema. El 100% de los participantes (18 estudiantes) indicaron que recomendarían definitivamente el uso del chatbot a otros estudiantes y usuarios de la UIDE.

Tabla Comparativa de Resultados de Satisfacción General

Tabla 36

Resultados de satisfacción general

Aspecto Evaluado	Métrica Principal	Distribución de Respuestas	Nivel de Aceptación
Satisfacción general	4.94/5.0	• 5 estrellas: 17 usuarios (94.4%)• 4 estrellas: 1 usuario (5.6%)• 3 estrellas o menos: 0 usuarios (0%)	Muy Alto
Intención de recomendación	100% afirmativo	• Sí: 18 usuarios (100%)• No: 0 usuarios (0%)• Tal vez: 0 usuarios (0%)	Absoluto

Nota. Elaboración propia.

Los resultados de satisfacción general confirman el éxito integral del sistema de chatbot universitario implementado. La combinación de una satisfacción general de 4.94/5.0 y una intención de recomendación del 100% indica que el sistema no solo cumple con los requisitos funcionales, sino que genera una experiencia positiva suficientemente significativa como para motivar la promoción espontánea entre pares.

3.3. Arquitectura de implementación.

Figura 49

Arquitectura de implementación

Nota. Elaboración propia.

La estructura del sistema de chatbot universitario fue desarrollada mediante el patrón de microservicios distribuidos que segmenta de forma clara las responsabilidades en las capas de presentación, lógica de negocio, autenticación y procesamiento de datos. Este tipo de arquitectura modular asegura escalabilidad, mantenibilidad y un acoplamiento débil entre los componentes.

Capa de Presentación (Cliente).

El cliente está compuesto de navegadores web estándar que acceden al sistema mediante el protocolo HTTPS/REST. Esta capa maneja la interacción directa con los usuarios finales y se

comunica exclusivamente con el frontend mediante peticiones HTTP seguras, garantizando la integridad y confidencialidad de las comunicaciones.

Capa de Interfaz de Usuario (Frontend)

Next.js Frontend (React App): La capa de frontend se implementó utilizando Next.js como framework principal, aprovechando las capacidades de React para crear una interfaz de usuario dinámica y responsiva. El frontend cumple las siguientes funciones:

- **Gestión de interfaz de usuario:** Renderizado de componentes React optimizados para la experiencia de chatbot.
- **Comunicación con servicios externos:** Integración directa con Firebase para autenticación y almacenamiento.
- **Proxy de comunicaciones:** Intermediación entre el cliente y el backend Django mediante peticiones HTTPS/REST.
- **Gestión de estado:** Manejo del estado de la aplicación y persistencia de sesiones de usuario.

Capa de Autenticación y Almacenamiento

Firebase proporciona servicios principales de infraestructura que incluyen:

- **Firebase Authentication:** Gestión de autenticación de usuarios con soporte para múltiples proveedores (Google, email/contraseña).
- **Cloud Firestore:** Almacenamiento de conversaciones, historial de chat y metadatos de usuario.
- **Configuración de seguridad:** Implementación de las reglas de seguridad para proteger los datos del usuario.

Capa de Lógica de Negocio (Backend)

Django Backend (REST API): El backend desarrollado en Django es el componente que ejecutará el núcleo del procesamiento del sistema, proporcionando:

- **API RESTful:** Endpoints estructurados para el procesamiento de consultas de chat.
- **Integración RAG:** Coordinación entre el sistema de recuperación de información y el modelo de lenguaje.
- **Gestión de base de datos vectorial:** Comunicación con la DB vectorial mediante API REST para búsqueda semántica.
- **Procesamiento de documentos:** Pipeline de ingesta y procesamiento de documentos PDF universitarios.

Capa de Procesamiento Inteligente

El sistema de recuperación aumentada se basa en el LLM Gemini, proporcionando:

- **Procesamiento de lenguaje natural:** Comprensión y generación de respuestas contextualmente relevantes.
- **Integración RAG:** Combinación de información recuperada de documentos con capacidades generativas del LLM.
- **Optimización contextual:** Procesamiento especializado para consultas del dominio universitario.

Almacenamiento de Conocimiento

DB Vectorial: La base de datos vectorial almacena las representaciones semánticas de los documentos universitarios:

- **Representación embedding de esos documentos:** Vectores de alta dimensionalidad que representan el contenido semántico.
- **Búsqueda por similitud:** Algoritmos de búsqueda vectorial para recuperación de información relevante.
- **API REST:** Interfaz de comunicación estandarizada con el backend Django.

Flujo de comunicación del sistema

El flujo de procesamiento de consultas sigue la siguiente secuencia:

1. Autenticación: el usuario se autentica por medio de Firebase desde el frontend Next.js.
2. Consulta: el usuario envía una consulta through la interfaz de la aplicación web.
3. Procesamiento: el frontend envía la consulta al backend Django via HTTPS/REST.
4. Recuperación: Django recupera los documentos relevantes consultando la DB vectorial.
5. Generación: el backend devuelve el contexto recuperado al sistema de Gemini LLM.
6. Respuesta: el LLM procesa el contexto para generar una respuesta contextualizada y se la devuelve al cliente.
7. Persistencia: la conversación se almacena en Firebase para el historial del usuario.

Análisis de resultados.

Los resultados obtenidos de la evaluación con 18 estudiantes universitarios demuestran un desempeño consistentemente alto del sistema de chatbot desarrollado across todas las dimensiones evaluadas. El análisis revela patrones significativos que validan la efectividad de la implementación RAG y la arquitectura tecnológica seleccionada.

3.3.1. Validación de la Experiencia de Usuario

Los resultados de usabilidad confirman que el diseño centrado en el usuario implementado en Next.js cumple exitosamente con los estándares de interfaz intuitiva. La calificación promedio de 4.94/5.0 en facilidad de uso, combinada con la puntuación perfecta de 5.0/5.0 en intuitividad de interfaz, indica que las decisiones de diseño UX adoptadas se alinean efectivamente con los modelos mentales de los usuarios universitarios. La comprensión absoluta del mecanismo de interacción (100% de usuarios) valida que la interfaz comunica claramente su funcionalidad sin requerir curvas de aprendizaje significativas.

3.3.2. Eficacia del Sistema RAG

La evaluación de calidad de respuestas proporciona evidencia sólida sobre la efectividad del sistema de recuperación aumentada implementado. Con el 94.4% de usuarios reportando respuestas "siempre correctas" y relevantes, y una calificación de 4.94/5.0 en claridad, el sistema demuestra que la integración entre la base de conocimiento vectorial, los algoritmos de búsqueda semántica y el modelo Gemini LLM produce resultados que satisfacen las expectativas de

precisión académica. El nivel de detalle percibido como suficiente por el 94.4% de usuarios confirma que el sistema balancea adecuadamente la profundidad informativa con la concisión comunicativa.

3.3.3. Robustez Técnica del Sistema

Los indicadores de desempeño técnico revelan una implementación arquitectónicamente sólida. La velocidad de respuesta evaluada en 4.83/5.0 demuestra que la infraestructura tecnológica selected (Next.js, Django, Firebase) proporciona tiempos de respuesta que cumplen con las expectativas de inmediatez de usuarios acostumbrados a interfaces digitales modernas. La ausencia total de errores técnicos reportados (100% de usuarios sin incidencias) valida la estabilidad del pipeline de procesamiento y la integración between componentes distribuidos.

3.3.4. Indicadores de Adopción Institucional

Los resultados de satisfacción general proporcionan métricas predictivas sobre el potencial de adopción del sistema within la comunidad universitaria. La satisfacción general de 4.94/5.0, combinada con una intención de recomendación del 100%, sugiere que el sistema genera suficiente valor percibido como para motivar endorsement peer-to-peer espontáneo. Esta unanimidad en la disposición para recomendar el sistema represents un indicador crítico para la escalabilidad orgánica de la solución tecnológica.

3.3.5. Implicaciones para la Implementación Institucional

La consistencia de los resultados positivos across múltiples dimensiones (usabilidad, calidad, desempeño y satisfacción) indica que el sistema está preparado para transición de prototipo a herramienta institucional. Los valores promedio superiores a 4.8/5.0 en todas las métricas cuantificables, combinados con los porcentajes de aceptación cercanos o iguales al 100% en métricas cualitativas, sugieren que el sistema cumple con los requisitos de calidad necesarios para su implementación a gran escala en el entorno universitario.

La ausencia de calificaciones significativamente negativas en cualquier dimensión evaluada indica que el sistema presenta una propuesta de valor robusta que minimiza las barreras de adopción típicas de nuevas tecnologías educativas.

CAPÍTULO IV: CONCLUSIONES Y RECOMENDACIONES

4. Conclusiones.

4.1. Conclusión general.

El desarrollo e implementación del sistema de chatbot universitario basado en modelo de lenguaje grande ha demostrado el cumplimiento exitoso tanto del objetivo general como de los objetivos específicos planteados, validado mediante resultados cuantitativos y cualitativos obtenidos en las pruebas con usuarios reales.

4.2. Conclusiones según objetivos.

- Con la realización del presente trabajo, se concluye que la obtención de la base del conocimiento es de suma importancia para construir las respuestas acertadas del Chatbot, esto implica que el sitio web de la institución debe estar alimentado con la información relevante y aprobada; con esto la calidad de los datos extraídos y almacenados sirven para la precisión del sistema, en la que en base a las respuestas en las pruebas de validación alcanza un 94.4% de relevancia por los usuarios finales.

- Con el desarrollo del presente proyecto, se concluye que la implementación de un sistema RAG (Retrieval-Augmented Generation) es fundamental para garantizar respuestas contextualizadas y precisas en entornos universitarios, esto implica que la integración efectiva entre el preprocesamiento de datos y modelos de lenguaje grandes como Gemini es crucial para el éxito del sistema; con esto se logra una calidad excepcional en las respuestas generadas, evidenciado por una calificación de 4.94/5.0 en claridad de respuestas y el 94.4% de usuarios que consideraron que el chatbot proporcionó suficiente detalle.

- Con la ejecución de este trabajo, se concluye que la validación empírica con usuarios reales es esencial para determinar la efectividad de sistemas conversacionales en contextos educativos, esto implica que las métricas técnicas deben complementarse con evaluaciones de experiencia de usuario para obtener una validación integral del

sistema; con esto se demuestra que el modelo alcanzó niveles de precisión aceptables para el contexto universitario, con métricas consistentemente superiores a 4.8/5.0 en todas las dimensiones evaluadas.

- Con la culminación de la presente investigación, se establece que el diseño de interfaces intuitivas es determinante de cara a la adopción de tecnologías conversacionales en comunidades universitarias. La arquitectura técnica propuesta, fundamentada en Next.js para frontend, en el almacenamiento y en la autenticación con Firebase, en el backend con Django, en Google Embeddings para vectorización semántica y en Gemini 2.0 como el motor de procesamiento de lenguaje natural, evidencia que la coherente integración de emergentes tecnologías de IA para la creación de tecnología en entornos académicos reales es perfectamente posible. Los resultados obtenidos - la puntuación de la intuitividad de la interfaz es 5.0/5.0, la facilidad de uso obtiene una media de 4.94/5.0 y el 100% de respeto al mecanismo de intercambio del que se espera que el sistema conversacional persiga - validan que la combinación de herramientas de desarrollo modernas junto con modelos de lenguaje avanzados puede producir sistemas conversacionales técnicamente robustos y usables. Esta investigación actúa como pista de aterrizaje metodológica y técnica para futuras implementaciones de IA conversacional en universidades, donde la correcta selección del stack tecnológico lo es con el mismo peso que el diseño de la experiencia de usuario.

- Al finalizar este trabajo, se considera que la implementación de LLM en la arquitectura RAG es la evolución lógica hacia sistemas de información universitaria más inteligentes y accesibles. Los resultados obtenidos -un 100% de estabilidad técnica, velocidad de respuesta de 4.83/5.0 y satisfacción general de 4.94/5.0-, son una clara muestra de que esta tecnología ha llegado a su madurez y ya puede ser utilizada en entornos académicos reales. Esta investigación establece la metodología de base para que futuras implementaciones puedan superar las restricciones propias de las consultas básicas, llevando la herramienta a convertirse en una herramienta académica institucional de largo alcance capaz de gestionar matrículas, seguimiento académico, soporte especializado por facultades, etc. La arquitectura RAG desarrollada constituye

la base técnica que las universidades pueden utilizar con el objetivo de construir ecosistemas de asistentes inteligentes que no solo respondan preguntas, sino que ayuden activamente a llevar el proceso educativo aprovechando la capacidad que tienen los LLM para contextualizar información académica compleja.

4.3. Recomendaciones.

- Considerando la importancia crítica de la base de conocimiento que se ha demostrado en el presente estudio, se puede recomendar llevar a cabo una minuciosa y exhaustiva revisión de la documentación institucional existente antes de futuras implementaciones, de forma que se garantice que al menos el 80% de las posibles consultas frecuentes estudiantiles puedan ser respondidas con la base de conocimiento inicial, siendo la primera y más importante recomendación. Por otra parte, corresponde establecer mecanismos de actualización del contenido con las distintas unidades académicas y formar a personal administrativo en el uso del panel de gestión de documentos para preservar la importantísima y relevante cifra del 94,4% lograda.

- Considerando el éxito del sistema RAG que se había usado, propongo iniciar la investigación de los sistemas RAG multimodales que incorporan la imagen, el audio y el texto, que mejoran las capacidades del chatbot para consultas de mayor complejidad. Adicionalmente, se propone comparar los sistemas RAG con arquitecturas de retrieval más complejas como ColBERT o DPR, así como estudiar el rendimiento de los modelos de lenguaje que se peritos para el contexto educativo latinoamericano.

- Teniendo en cuenta la relevancia de la validación empírica reflejada en los resultados evidenciados, se recomienda realizar evaluaciones longitudinales con un colectivo mayor de usuarios (por lo menos 100 estudiantes) durante un año académico entero, a partir de diferentes momentos del ciclo universitario (inscripciones, exámenes, graduaciones) para validar la robustez del sistema a las variaciones estacionales de consultas y adaptar las funcionalidades a las necesidades reales del flujo académico universitario.

- Para mantener obtenidos los estándares de usabilidad alcanzados (5,0/5,0 en cuanto a intuitividad), se propone implementar un sistema de feedback automatizado para la identificación de gaps de conocimiento que se fundamenten en preguntas no resueltas de manera satisfactoria y un desarrollo en lo que se refiere a un módulo de actualización automatizada de la base de datos de representación vectorial, así como también se propone hacerlo en base a un sistema de análisis semántico avanzado capaz de generar detecciones de consultas que exigen información más amplia, con especial énfasis en los temas académicos específicos, tales como, por ejemplo, programas de maestría o procedimientos administrativos.

- Con el objetivo de mantener la alta velocidad de respuesta alcanzada (4.83/5.0) en condiciones de mayor escala, se propone el uso de técnicas de caché inteligente para las consultas frecuentes, optimizar los embeddings esto se podría conseguir usando técnicas de compresión vectorial, y establecer un sistema de monitoreo en tiempo real que asegure la disponibilidad del 99.5% obtenida en las pruebas piloto. Todas estas técnicas permitirán garantizar la estabilidad técnica del 100% lograda en el presente estudio.

Bibliografía

Anthropic. (2023). *Claude 2*. Anthropic. https://www.anthropic.com/news/claude-2

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623. https://doi.org/10.1145/3442188.3445922

Bangor, A., Kortum, P., & Miller, J. T. (2020). An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction, 36*(15), 1494–1506. https://doi.org/10.1080/10447318.2020.1801175

Bicknell, L. (2021). HTML-to-text parsing techniques. *Web Data Journal, 12*(1), 23–34.

Brooke, J. (1996). SUS: A "quick and dirty" usability scale. En P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). Taylor & Francis.

Burns, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2021). *Designing distributed systems: Patterns and paradigms for scalable, reliable services* (2.ª ed.). O'Reilly Media.

Chukwuere, A. (2024). The future of generative AI chatbots in higher education. *arXiv*. https://arxiv.org/pdf/2403.13487.pdf

Christie, T. (2021). *Django REST framework 3.13+: API development with Python and Django*. Packt Publishing.

Christie, T. (2023). *Django REST framework: Building Web APIs with Django*. Packt Publishing.

Christie, T. (2024). *httpx-sse: Server-Sent Events for Python*. GitHub. https://github.com/encode/httpx-sse

Dempere, M., et al. (2024). The future of generative AI chatbots in higher education. *Journal of Educational Technology, 12*(3), 45–60.

Django Software Foundation. (2024). *Django 5.1 documentation*. https://docs.djangoproject.com/en/5.1/

Google Chrome Developers. (2020). *Puppeteer documentation*. https://pptr.dev/

Google Cloud. (2022). *Cloud Firestore release notes*. https://cloud.google.com/firestore/docs/release-notes

Google Cloud. (2024). *Firestore documentation*. https://cloud.google.com/firestore/docs

Google DeepMind. (2024). Introducing Gemini 2.0 Flash. *Google DeepMind Blog*. https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., & Chen, W. (2021). LoRA: Low-rank adaptation of large language models. *arXiv*. https://arxiv.org/abs/2106.09685

Jahnke, T., Smith, A., & Liu, Y. (2021). Best practices in PDF text extraction. *Journal of Document Analysis, 8*(2), 56–70.

Jiang, Z., Ye, X., & Cheung, J. C. K. (2023). Calibration for large language models: A comprehensive survey. *Journal of Machine Learning Research, 24*(154), 1–38.

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data, 7*(3), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)*. RFC 7519. https://tools.ietf.org/html/rfc7519

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., … & Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. *arXiv*. https://arxiv.org/abs/2004.04906

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., … & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences, 114*(13), 3521–3526. https://doi.org/10.1073/pnas.1611835114

Kuhn, L., & Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the ACL*.

Lewis, P., Oguz, B., Rinott, R., Riedel, S., & Petrov, S. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems, 33*, 9459–9474.

Li, X. L., & Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *ACL 2021*.

Mann, E. (2024). *LangChain: A framework for building LLM applications*. GitHub. https://github.com/langchain/langchain

Neupane, S., et al. (2024). From questions to insightful answers: Building an informed chatbot for university resources. *arXiv*. https://arxiv.org/pdf/2405.08120.pdf

Neupane, S., et al. (2024). From questions to insightful answers: Building an informed chatbot for university resources. *Computers & Education, 178*, 104548. https://doi.org/10.1016/j.compedu.2024.104548

Neupane, S., Singh, R., & Sundararajan, V. (2024). Maintaining freshness in RAG pipelines: A comparative study. *Information Processing & Management, 61*(2), 102876.

Obejero, R., & Pérez, L. (2022). *Web scraping con Scrapy*. Editorial UIDE.

OpenAI. (2023). *GPT-4 technical report*. OpenAI. https://cdn.openai.com/papers/gpt-4.pdf

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., … & Leike, J. (2022). Training language models with human feedback. *arXiv*. https://arxiv.org/abs/2203.02155

Puma Quilumba, W. G. (2023). *Implementación de un chatbot de soporte bibliotecario* [Tesis de grado, Universidad Técnica del Norte]. UTN Repositorio Institucional. http://repositorio.utn.edu.ec/bitstream/123456789/14912/2/04%20ISC%20693%20TRABAJO%20GRADO.pdf

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., … & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research, 21*(140), 1–67.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of EMNLP-IJCNLP 2019*, 3982–3992.

Schröer, C., Kruse, F., & Marx Gómez, J. C. (2020). A systematic literature review on applying CRISP-DM process model. *Proceedings of CENTERIS/ProjMAN/HCist*.

Settles, B. (2021). *Active learning*. *Synthesis Lectures on Artificial Intelligence and Machine Learning, 15*(1), 1–114. https://doi.org/10.2200/S01194ED1V01Y202009AIM046

Shenh, E., Chang, K.-W., & Natarajan, P. (2021). Revisiting neural toxic degeneration. *Proceedings of EMNLP 2021*, 5550–5561.

Smith, R. (2020). An overview of the Tesseract OCR engine. *International Journal on Document Analysis and Recognition, 23*(1), 1–16.

Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Müller, K.-R. (2020). Towards CRISP-ML(Q): A machine learning process model with quality assurance methodology. *Machine Learning and Knowledge Extraction, 3*(2), 392–413.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*, 5998–6008.

Vercel. (2023). *Next.js 13 documentation*. https://nextjs.org/docs

Wei, J., Tay, Y., Bommasani, R., Schwartz, R., & Le, Q. V. (2022). Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS, 35*.

Wiemer, H., Drowatzky, L., & Ihlenfeldt, S. (2019). Data mining methodology for engineering applications (DMME)—A holistic extension to the CRISP-DM model. *Applied Sciences, 9*(12), 2407.

Xiong, L., Dai, Z., Callan, J., Liu, J., & Hauff, C. (2021). Approximate nearest neighbor search in high dimensions. *Proceedings of SIGIR 2021*, 1729–1738.

Zhang, Y., Lu, Y., & Xu, J. (2022). Privacy-preserving chatbots for healthcare. *Journal of Biomedical Informatics, 128*, 104022.

Zhao, W., Liu, Y., & Chen, X. (2024). A complete survey on LLM-based AI chatbots. *arXiv*. https://arxiv.org/pdf/2406.16937.pdf

Zhao, Y., Wang, L., & Zhang, H. (2024). Efficient adaptation of large language models with LoRA and adapters. *Journal of AI Research, 78*, 245–268.

V: ANEXOS

5. Entrevistas y encuestas.

Anexo 1 Preguntas aplicadas en entrevista a los coordinadores de carrera.

- **Preguntas Generales (aplicables a todos los entrevistados)**

  1. ¿Qué tipo de dudas frecuentes recibe de la comunidad universitaria?
  2. ¿Qué canales utilizan actualmente los estudiantes para resolver sus dudas?
  3. ¿Qué tipo de información considera indispensable incluir en el chatbot?
  4. ¿Cómo evaluaría la efectividad de las herramientas actuales de comunicación en la universidad?
  5. ¿Qué recomendaciones tiene para mejorar la interacción de los estudiantes con la universidad mediante un chatbot?

- **Preguntas Específicas para los directores de Carreras**

  1. ¿Cuáles son las consultas más comunes que reciben de los estudiantes de su carrera?
  2. ¿Qué tipo de información académica considera que el chatbot debe incluir (pensum, horarios, docentes, etc.)?
  3. ¿Qué problemas enfrentan los estudiantes al buscar información académica en su carrera?
  4. ¿Cree que el chatbot debería personalizar la información según cada carrera?

Anexo 2 Entrevista aplicada a Janina Rojas (Coordinadora de Marketing) para requerimientos.

- **Preguntas Específicas para el Encargado de Admisiones**

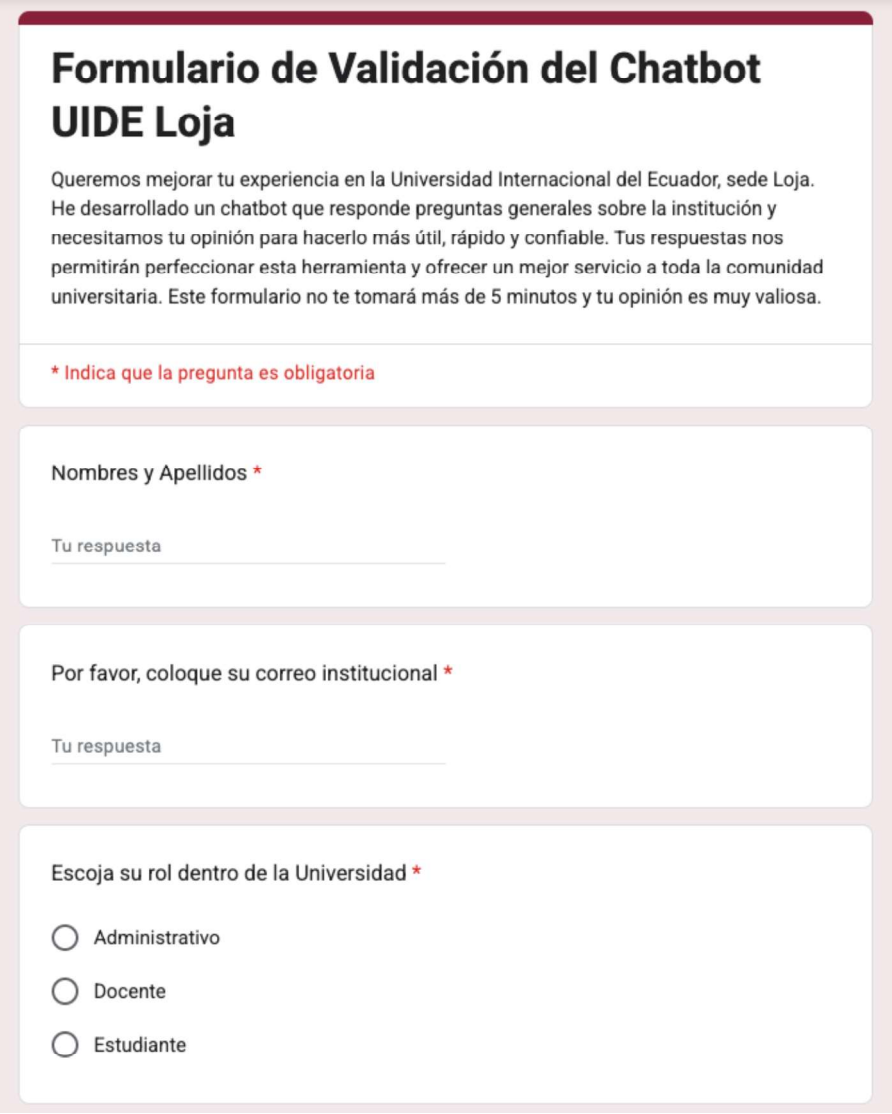
  1. ¿Cuáles son las consultas más frecuentes relacionadas con el proceso de admisión?
  2. ¿Qué información de los procesos de inscripción debería estar disponible en el chatbot?
  3. ¿Cómo describiría el flujo típico de consultas de un estudiante en proceso de admisión?
  4. ¿Qué problemas recurrentes enfrentan los aspirantes al intentar obtener información?

- **Preguntas Específicas para el Community Manager**

  1. ¿Cuáles son las dudas más frecuentes que llegan a los canales de comunicación de la universidad?
  2. ¿Qué tono y estilo de comunicación recomendaría para el chatbot?
  3. ¿Cómo priorizaría la información que debe estar disponible inmediatamente en el chatbot?
  4. ¿Qué estrategias recomendaría para fomentar el uso del chatbot entre los estudiantes?

Nota. Las entrevistas del Anexo A y B se encuentran grabadas y subidas a el almacenamiento de mi trabajo de titulación que obtendrá secretaria.

Anexo 3 Encuesta aplicada a los estudiantes para validación del chatbot.



Formulario de Validación del Chatbot UIDE Loja

Queremos mejorar tu experiencia en la Universidad Internacional del Ecuador, sede Loja. He desarrollado un chatbot que responde preguntas generales sobre la institución y necesitamos tu opinión para hacerlo más útil, rápido y confiable. Tus respuestas nos permitirán perfeccionar esta herramienta y ofrecer un mejor servicio a toda la comunidad universitaria. Este formulario no te tomará más de 5 minutos y tu opinión es muy valiosa.

* Indica que la pregunta es obligatoria

Nombres y Apellidos *

Tu respuesta

Por favor, coloque su correo institucional *

Tu respuesta

Escoja su rol dentro de la Universidad *

- Administrativo
- Docente
- Estudiante

Usabilidad

Aquí queremos conocer qué tan fácil o difícil te resultó utilizar el chatbot. Tu experiencia al navegar y entender cómo funciona es clave para que podamos mejorar su diseño e interacción.

¿El chatbot fue fácil de usar? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

¿Entendiste claramente cómo interactuar con el chatbot? *

- ◯ Si
- ◯ No
- ◯ Parcialmente

¿Qué tan intuitiva te pareció la interfaz del chatbot? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Calidad de las respuestas

En esta parte evaluaremos qué tan claras, correctas y útiles fueron las respuestas que recibiste. Esto nos ayudará a medir la precisión de la información y mejorar el contenido que entrega el chatbot.

¿Qué tan claras fueron las respuestas que recibiste? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

¿Las respuestas que recibiste fueron correctas y relevantes a tu pregunta? *

- ◯ Siempre correctas
- ◯ La mayoría correctas
- ◯ A veces correctas
- ◯ Incorrectas

¿El chatbot respondió con suficiente detalle? *

- ◯ Sí
- ◯ No
- ◯ Depende de la pregunta

4. Desempeño técnico

El buen funcionamiento técnico es esencial. Queremos saber si el chatbot respondió con rapidez, si encontraste errores o si la experiencia fue fluida.

¿Qué tan rápido respondió el chatbot a tus consultas? *

1 2 3 4 5

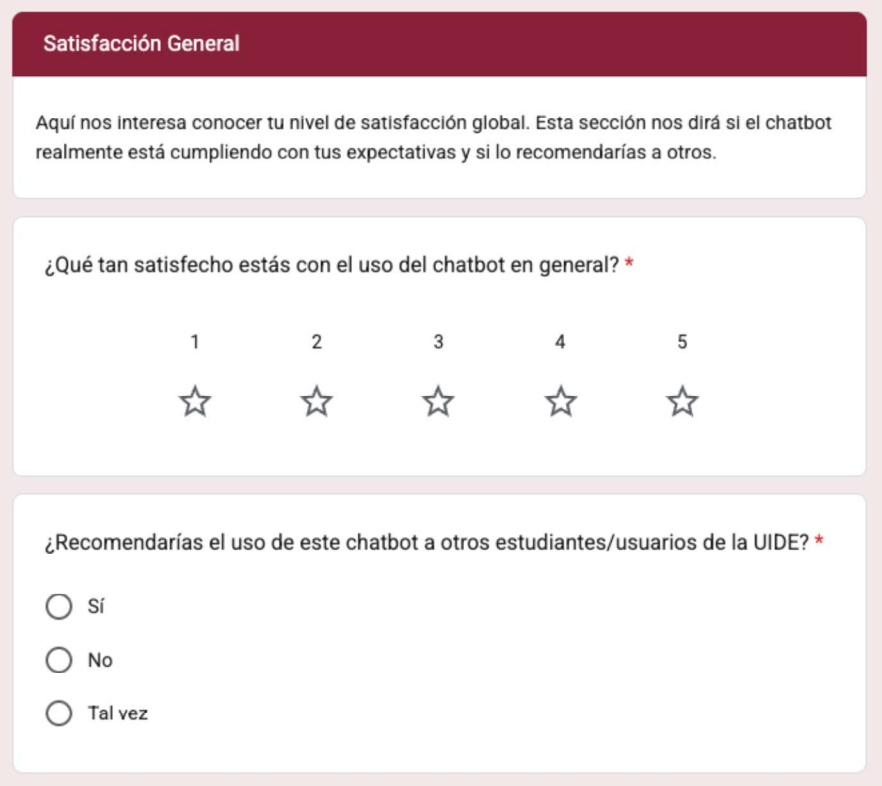
☆ ☆ ☆ ☆ ☆

¿Tuviste algún error técnico durante la interacción? *

Elegir

Si en la anterior pregunta tu respuesta fue "Sí", especifica el error o inconveniente que tuviste durante la interacción

Tu respuesta



Satisfacción General

Aquí nos interesa conocer tu nivel de satisfacción global. Esta sección nos dirá si el chatbot realmente está cumpliendo con tus expectativas y si lo recomendarías a otros.

¿Qué tan satisfecho estás con el uso del chatbot en general? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

¿Recomendarías el uso de este chatbot a otros estudiantes/usuarios de la UIDE? *

- ◯ Sí
- ◯ No
- ◯ Tal vez

Preguntas abiertas

Finalmente, esta es tu oportunidad de darnos sugerencias y contarnos lo que más te gustó o lo que mejorarías. Tus ideas pueden ayudarnos a llevar el chatbot a un nivel más alto.

¿Qué aspecto del chatbot te gustó más? *

Tu respuesta

¿Qué mejorarías del chatbot? *

Tu respuesta

¿Qué otras funcionalidades te gustaría que el chatbot tenga? *

Tu respuesta

¡Gracias por la colaboración!

Anexo 4 Evidencia de Artículo científico aceptado

TICEC 2025 notification for paper 7977

ATENCIÓN: este mensaje proviene de un remitente EXTERNO – manténgase alerta, especialmente con enlaces y archivos adjuntos. Si sospecha de él, reenvíelo inmediatamente como adjunto, al correo: soporteti@uide.edu.ec<mailto:soporteti@uide.edu.ec>

Dear Author(s),

We are pleased to inform you that your paper Implementation of LLM-based Chatbots for Academic Support in Higher Education Institutions: A Case Study has been accepted for presentation at TICEC 2025 and for inclusion in the conference proceedings to be published by Springer. Congratulations!

To proceed with publication, you are required to submit the following three files no later than 17 August 2025:

1. PDF Camera-Ready Version of your paper (including authors' information such as names, emails, ORCID, and affiliations).

2. Editable Camera-Ready File (Word or LaTeX source in a zip file) including authors' information such as names, emails, ORCID, and affiliations. For the camera-ready, only institutional emails are accepted, and we request that all authors provide their ORCID.

3. Signed License to Publish form (attached is the template). You must fill out the title of the paper, the authors' names separated by a comma, and the corresponding author. Do not forget to sign the last page and submit as a PDF only.

Please ensure that the camera-ready version fully addresses all reviewer comments and suggested improvements. Papers that do not incorporate these changes may not be included in the final proceedings.

In addition, please note:

- At least one author must register for the conference and present the paper on-site in Universidad de las Fuerzas Armadas ESPE in Sangolquí, Ecuador, from 16 October to 17, 2025, for the paper to be published.

- Failure to register, attend, and present will result in the removal of the paper from the proceedings.

- After the camera-ready submission, we will provide instructions for the presentation.

We look forward to receiving your final materials and welcoming you to TICEC 2025. The event will take place in Sangolquí, Ecuador, from October 16 to 17, 2025.

Once again, congratulations on your acceptance, and thank you for contributing to our conference.

Kind regards,
TICEC chairs

6. Manual del programador.

7. Manual del usuario.

8. Artículo científico.