Universidad Nacional de Ingeniería

Área de Conocimiento de Tecnología de la Información y Comunicación

Chatbot para Consultas del Digesto Jurídico Nicaragüense

Trabajo Monográfico para optar al título de Ingeniero en Computación

Elaborado por:

Br. Sjeff Michael Gómez Espinosa
Carnet: 2019-1063U

Br. Brandon Ramsés Espinoza Ruiz
Carnet: 2019-0023U

Tutor:

MSc. Danny Oswaldo Chávez Miranda

19 de agosto de 2025
Managua, Nicaragua

Dedicatoria

Sjeff Michael Gómez Espinosa:

A mis padres, **Migdalia del Socorro Espinosa** y **Oscar Javier Gómez**, por todo el amor y apoyo brindado, por siempre darme su voto de confianza para cumplir mis metas, este logro es fruto de su esfuerzo. Prometo seguirlos haciendo sentir orgullosos.

A familiares y amigos, en especial a mi novia **Malhi Berrios**, por ser un pilar importante en mi vida y por motivarme a cumplir mis metas y a mi colega **Melvin Mendoza**, a quien considero un gran amigo y mentor, gracias por todos los consejos y enseñanzas.

A todos los maestros que formaron parte de esta etapa de mi vida, especialmente a la Ing. **Glenda Barrios** por su ardua labor y compromiso como docente y por ser quien nos ayudó a iniciar este proyecto y a nuestro tutor, el Ing. **Danny Chávez** por guiarnos durante este proceso con dedicación y sabiduría.

Brandon Ramsés Espinoza Ruiz:

A Dios, por haberme acompañado en todo este proceso, dándome salud, paciencia y claridad para seguir adelante, incluso en los momentos más exigentes. Gracias por fortalecerme y mantenerme enfocado hasta alcanzar esta meta.

A mis padres, **Freddy Ernesto Espinoza** y **Arlen María Ruiz García**, por su apoyo constante, por creer en mí y por brindarme siempre su amor y confianza. Gracias por motivarme a seguir adelante, incluso cuando las circunstancias se tornaban difíciles. Este logro también es de ustedes.

A mi familia en general, por sus palabras de aliento, por estar presentes de una u otra manera, y por ser parte importante de mi vida.

Y a todos aquellos amigos y compañeros que me acompañaron a lo largo de este camino, con quienes compartí aprendizajes, desafíos y momentos inolvidables. Gracias por estar ahí.

Resumen del tema

Este trabajo monográfico se centró en el desarrollo de un chatbot que permite realizar consultas, sobre información contenida en el Digesto jurídico nicaragüense, a través lenguaje natural, que brinde acceso rápido e intuitivo a la información legal.

La implementación del proyecto se gestionó bajo la metodología ágil Scrum, dividiendo el trabajo en cuatro etapas enfocadas en el diseño, procesamiento de datos, desarrollo de servicios y evaluación. El núcleo del chatbot, desarrollado con Python, Django y Langchain se basó en la creación de un sistema de generación aumentada por recuperación (RAG por sus siglas en ingles) que emplea el método de búsqueda híbrida que combina *embeddings* densos con *embeddings* dispersos. Los resultados de esta búsqueda son refinados por un modelo de reranqueo antes de ser procesados por un modelo grande de lenguaje (LLM por sus siglas en ingles) para generar una respuesta coherente y contextualizada.

Inicialmente, se extrajo la información del portal web del Digesto mediante técnicas de web scraping. Posteriormente, los documentos legales obtenidos fueron segmentados, transformados en vectores numéricos mediante modelos de *embeddings*, y almacenados en una base de datos vectorial (Milvus) para su consulta.

La efectividad del sistema se evaluó cuantitativamente utilizando el framework RAGAS, que midió la calidad de las respuestas a través de métricas como fidelidad, precisión del contexto, relevancia de la respuesta y recuperación del contexto. Los resultados demostraron un alto rendimiento, destacando en la relevancia de las respuestas generadas y en la recuperación de contexto pertinente para fundamentarlas.

En conclusión, con este proyecto no solo se logró construir una herramienta funcional que democratiza el acceso a la información legal en el país, sino que también validó la eficacia del método RAG para el dominio legal nicaragüense.

Tabla de Contenido

Índice de Anexos

Índice de figuras

Índice de Tablas

1. Introducción

En un mundo cada vez más conectado, la inteligencia artificial se ha convertido en un pilar fundamental para resolver problemas complejos y avanzar en el desarrollo tecnológico. Bajo esta premisa, se desarrolló un chatbot basado en modelos y técnicas de inteligencia artificial, con el objetivo de responder a consultas sobre información contenida en el Digesto Jurídico Nicaragüense.

Este proyecto surge con la intención de facilitar el acceso a la información legal en el país, ayudando a superar las limitaciones de eficiencia y usabilidad que presenta la plataforma oficial del Digesto. De esta manera, se busca no solo agilizar la labor de los profesionales del derecho, sino también democratizar el acceso al conocimiento legal para el público general. Asimismo, este trabajo se establece como un caso de estudio que sirva de referencia para futuras iniciativas tecnológicas en un área que aún no ha sido explorada en el ámbito nacional.

Para la implementación del sistema se adoptó la metodología ágil Scrum, lo que permitió organizar el desarrollo del chatbot en fases iterativas que, fueron planteadas a través de la definición de requerimientos funcionales y no funcionales enfocados al cumplimiento de los objetivos.

En total se plantearon cuatro fases. En una primera fase, se diseñaron los componentes fundamentales del sistema, tanto del lado del servidor (backend) como de la interfaz de usuario (frontend), estableciendo así la arquitectura inicial del proyecto. La segunda fase se centró en la implementación de la base de datos relacional, así como en el procesamiento y almacenamiento de la información del Digesto. En la tercera fase, se desarrollaron los servicios del backend, incluyendo módulos para autenticación, gestión de usuarios, administración de la información legal, y el sistema de generación de respuestas. En la cuarta y última fase, se construyó la interfaz de usuario y se realizaron pruebas orientadas a evaluar el desempeño y la calidad de las respuestas generadas por el sistema.

2. Antecedentes

Actualmente, los chatbots han ganado mucha relevancia en nuestra sociedad, impulsados por los significativos avances en inteligencia artificial de los últimos años. En particular, el desarrollo de los grandes modelos de lenguaje (LLM, por sus siglas en inglés) ha potenciado enormemente las capacidades de los chatbots, transformando en gran medida la forma en que se consulta información.

Estos avances son especialmente relevantes en el sector legal, donde los chatbots pueden proporcionar respuestas rápidas y precisas a consultas legales, mejorando la eficiencia y accesibilidad de los servicios legales.

A nivel internacional se han planteado soluciones que demuestran el potencial de esta tecnología. Por ejemplo, en Palestina, se desarrolló un chatbot para proporcionar soporte legal a cooperativas, ayudando a resolver inquietudes legales de sus miembros (Qasem et al., 2023). Asimismo, la iniciativa OpenJustice trabaja en la creación de una IA legal basada en LLM avanzados, capaz de procesar consultas y documentos, utilizando amplias fuentes legales para brindar respuestas a estudiantes de derecho y abogados (Witten, 2023).

Hoy en día, se accede a información legal a través de formatos tradicionales como el Digesto Jurídico Nicaragüense, una plataforma web donde se recopila y organiza el marco legal del país. Para consultar información en ella el usuario debe ingresar en los distintos cuadros de búsqueda del formulario web palabras claves o datos como: fecha de publicación, fecha de aprobación, materia relacionada, categoría, número de la norma, entre otros, para finalmente encontrar lo que busca.

Si bien este sistema permite búsquedas detalladas mediante múltiples filtros, presenta ciertas limitaciones. En primer lugar, su efectividad depende del conocimiento previo del usuario sobre términos legales y criterios de búsqueda específicos, lo que dificulta su uso para quienes no poseen formación jurídica, lo que lleva a resultados de búsqueda poco precisos. Además, aunque la variedad de filtros posibilita consultas

específicas, el proceso de rellenar cada uno de estos parámetros puede volverse tedioso y consumir tiempo considerable. Cabe destacar que no se conoce a simple vista cuales son los campos requeridos que necesita ingresar el usuario para obtener un resultado de búsqueda satisfactorio.

Figura 1

Filtros utilizados para realizar una búsqueda sobre materia Administrativa en el Digesto Jurídico Nicaragüense

3. Justificación

La propuesta de un chatbot para proporcionar respuestas rápidas y precisas sobre información legal surge como una alternativa innovadora para superar las limitaciones del Digesto Jurídico Nicaragüense en términos de eficiencia y facilidad de uso. Este sistema aprovecha el potencial de los LLM, los cuales han demostrado ser herramientas poderosas en la optimización del acceso a la información. Sin embargo, dado que estos modelos por sí solos no son suficientes para garantizar resultados precisos en el ámbito legal, la solución propuesta integrará enfoques avanzados, como la utilización de la técnica de generación aumentada por recuperación (RAG, por sus siglas en inglés), para mejorar la precisión y confiabilidad de las respuestas.

El desarrollo de este chatbot permite agilizar y mejorar la precisión de los resultados en las búsquedas de información legal, reduciendo significativamente el tiempo que los profesionales del derecho invierten en esta tarea. Además, facilitaría el acceso a normativas y disposiciones legales a personas con poca o nula formación jurídica, eliminando barreras de conocimiento y promoviendo un acceso más equitativo a la información legal. De esta manera, la propuesta contribuiría a la democratización del conocimiento legal, haciéndolo más accesible y comprensible para la ciudadanía en general.

Desde una perspectiva técnica, la implementación de este chatbot también representaría un caso de estudio valioso sobre la integración de LLM avanzados en sistemas legales existentes. Su desarrollo no solo serviría como referencia para futuras iniciativas tecnológicas en Nicaragua, sino que también podría inspirar proyectos similares en otros países de la región, fomentando la modernización del sector legal mediante el uso de inteligencia artificial.

4. Objetivos

4.1. General

- Desarrollar un chatbot que permita acceder de manera fácil y rápida a información contenida en el Digesto Jurídico Nicaragüense mediante la realización de consultas en lenguaje natural.

4.2. Específicos

- Planificar las fases de desarrollo del chatbot utilizando el marco de trabajo Scrum por medio de un análisis de requisitos basado en historias de usuario.
- Analizar la información del Digesto Jurídico Nicaragüense relevante para el chatbot mediante el uso de técnicas de web scraping, modelos de embeddings y bases de datos vectoriales.
- Diseñar la interfaz de usuario y lógica interna del chatbot, a través de herramientas de diseño y diagramas para describir la arquitectura del sistema, flujos de datos, manejo de consultas y generación de respuestas.
- Implementar los diferentes componentes del chatbot, incluyendo la interfaz de usuario y su estructura lógica, utilizando el lenguaje de programación Python.
- Evaluar la efectividad del chatbot utilizando RAGAS, enfocándose en métricas como fidelidad, precisión del contexto, relevancia de las respuestas y recuperación del contexto, asegurando la coherencia, precisión y pertinencia de las respuestas generadas.

5. Marco Teórico

En este apéndice se describen los fundamentos teóricos que permitirán comprender la base ingenieril de esta investigación.

5.1. Digesto Jurídico Nicaragüense

El Digesto Jurídico es una herramienta clave utilizada por los parlamentos modernos para enfrentar la Inflación legislativa, que es la producción excesiva y desordenada de normas que puede generar contradicciones legales, y la contaminación legislativa, que se define como la ineficacia normativa causada por la aplicación de leyes derogadas, expiradas, obsoletas o mal redactadas (Asamblea Nacional, 2024).

El Digesto Jurídico Nicaragüense es elaborado por materia y garantiza el ordenamiento del marco jurídico vigente del país, a través de los procedimientos de recopilación, compilación, ordenamiento, análisis, consolidación, depuración, sistematización y actualización de las normas jurídicas vigentes, no vigentes, históricas e instrumentos internacionales.

5.2. Inteligencia Artificial

La inteligencia artificial es una rama de la ciencia informática que tiene como objetivo diseñar tecnología que emule la inteligencia humana. Esto significa que, mediante la creación de algoritmos y sistemas especializados, las máquinas pueden llevar a cabo procesos propios de la inteligencia humana, como aprender, razonar o autocorregirse (tableau, 2023).

Estos sistemas pueden aplicar el conocimiento adquirido para resolver problemas de manera similar a los procesos humanos. Desde responder a conversaciones humanas hasta generar contenido original y tomar decisiones basadas en datos en tiempo real.

5.3. Aprendizaje Automático (Machine Learning)

El aprendizaje automático (ML) es el proceso mediante el cual se usan modelos matemáticos de datos para ayudar a un equipo a aprender sin instrucciones directas (microsoft, 2023). Se divide en aprendizaje supervisado (datos etiquetados con respuestas conocidas) y no supervisado (datos sin etiquetar). El primero incluye técnicas como regresión y clasificación, mientras que el segundo abarca métodos como el agrupamiento y el análisis de componentes. La elección entre ambos depende del caso de uso y los datos disponibles.

5.4. Red Neuronal

Una red neuronal es una programa o modelo de ML que toma decisiones de manera similar al cerebro humano, mediante el uso de procesos que imitan la forma en que las neuronas biológicas trabajan juntas para identificar fenómenos, sopesar opciones y llegar a conclusiones (IBM, 2021).

Cada red neuronal consta de capas de nodos o neuronas artificiales, una capa de entrada, una o más capas ocultas y una capa de salida. Cada nodo se conecta con otros y tiene su propio peso y umbral asociados. Si la salida de cualquier nodo individual está por encima del valor umbral especificado, ese nodo se activa y envía datos a la siguiente capa de la red. De lo contrario, no se pasan datos a la siguiente capa de la red.

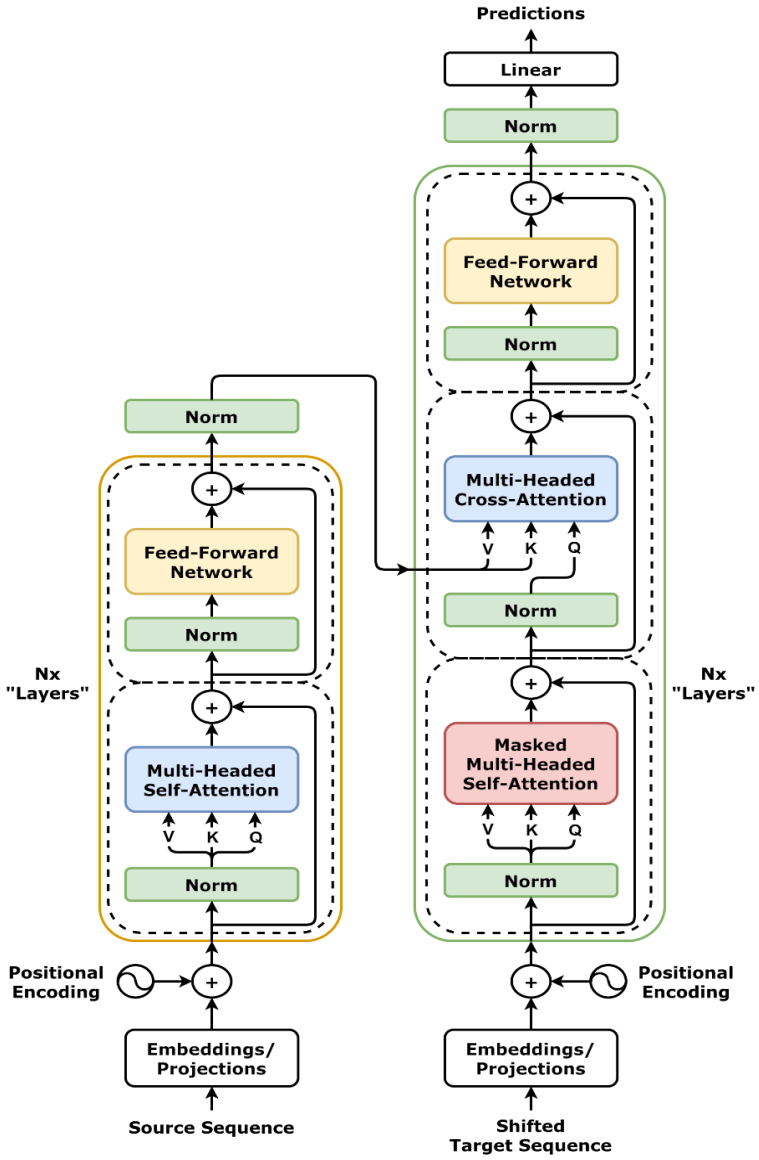
5.5. Modelo de Transformadores

El modelo del transformador es un tipo de red neuronal arquitectura que se destaca en el procesamiento de datos secuenciales, más prominentemente asociada con los LLM. Los modelos de transformadores también han logrado un rendimiento de élite en otros campos inteligencia artificial (IA), como visión por computadora, reconocimiento de voz y pronóstico de series temporales (Stryker & Bergmann, 2025).

La característica central de los modelos de transformadores es su mecanismo de autoatención, de donde los modelos de transformadores derivan su impresionante capacidad para detectar las relaciones (o dependencias) entre cada parte de una secuencia de entrada. A diferencia de las arquitecturas de redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN) que la precedieron, la arquitectura del transformador utiliza solo capas de atención y capas de retroalimentación estándar. En la siguiente figura se muestra una arquitectura estándar del modelo de transformadores en donde la parte izquierda muestra un codificador y la parte derecha muestra un decodificador.

Figura 2

Diagrama de bloques de la arquitectura completa del modelo de transformadores



5.6. Chatbots Inteligentes

Los chatbots son programas de software diseñados para interactuar de forma conversacional con los usuarios, basándose en el procesamiento del lenguaje natural (PLN), que analiza y comprende el lenguaje del usuario, la comprensión del lenguaje natural (CLN), que interpreta la expresión del usuario y la generación de lenguaje natural (GNL), que permite al chatbot responder de manera coherente (Apd, 2023).

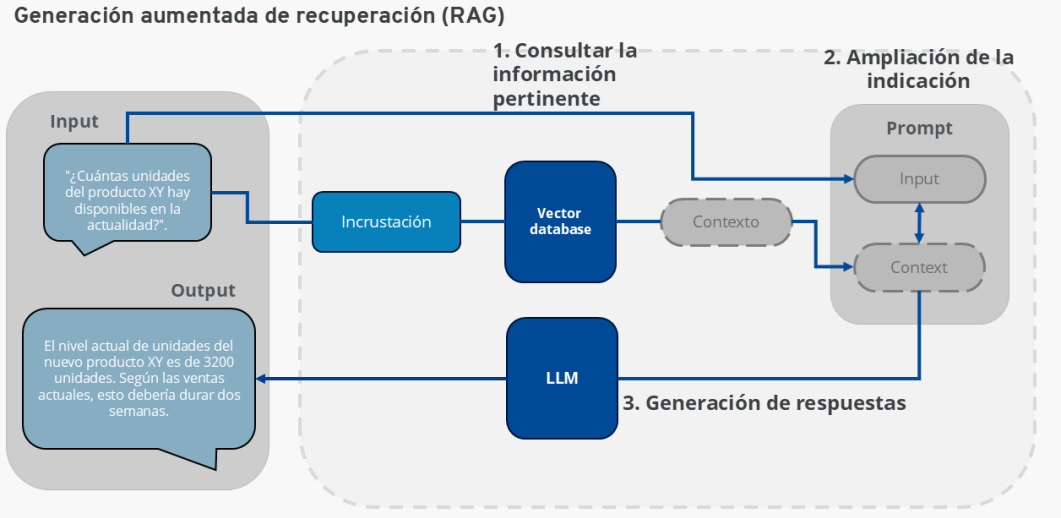
Dentro de los tipos principales de chatbots se encuentran los simples, que operan mediante palabras clave predefinidas, y los inteligentes, que emplean inteligencia artificial y aprendizaje automático para entender y responder a una amplia gama de preguntas y solicitudes en lenguaje natural.

5.7. RAG

RAG es un marco de inteligencia artificial para mejorar la calidad de las respuestas generadas por los LLM al basar el modelo en fuentes externas de conocimiento para complementar la representación interna de la información del LLM. La implementación de RAG en un sistema de respuesta a preguntas basado en LLM tiene dos beneficios principales: garantiza que el modelo tenga acceso a los hechos más actuales y confiables, y que los usuarios tengan acceso a las fuentes del modelo, lo que garantiza que se pueda verificar la precisión de sus afirmaciones (Computing & Computing, 2023).

Figura 3

Diagrama de flujo de la técnica RAG



5.8. LLM

Los LLM son modelos de propósito general de Inteligencia Artificial desarrollados dentro del campo del PLN que puede entender y generar texto al estilo humano (Na & Na, 2024).

Los modelos más famosos actuales como “GPT” tienen una arquitectura basada en modelos de transformadores y usan redes neuronales que son entrenadas con inmensas cantidades de texto obtenidos y “curados” de internet, incluyendo libros, periódicos, foros, recetas, paper científicos, patentes, enciclopedias, abarcando diversas temáticas, lo que hace que este tipo de modelos sean de propósito general.

5.9. Modelos de Embeddings Densos

Los **modelos de *embeddings* densos** son técnicas de representación vectorial en las que cada unidad (palabra, oración, documento, etc.) es codificada como un vector de dimensión fija donde la mayoría de los valores son distintos de cero. Estos vectores densos capturan relaciones semánticas de forma distribuida y continua, permitiendo medir similitudes entre elementos. Son entrenados generalmente mediante métodos de aprendizaje profundo como redes neuronales. A diferencia de los *embeddings* dispersos, estos modelos requieren mayor capacidad de cómputo, pero suelen ofrecer mejor rendimiento en tareas como recuperación de información, clasificación y traducción automática (Reimers & Gurevych, 2019).

5.10. Modelos de Embeddings Dispersos

Los modelos de embeddings dispersos (sparse embeddings) representan elementos (como palabras o documentos) como vectores de alta dimensión donde la mayoría de los valores son cero. A diferencia de los embeddings densos, estos modelos conservan interpretabilidad y eficiencia en recuperación a gran escala, ya que permiten aprovechar estructuras esparsas en la representación. Se utilizan ampliamente en motores de búsqueda y sistemas de recuperación de información donde la escalabilidad es clave (Formal et al., 2021).

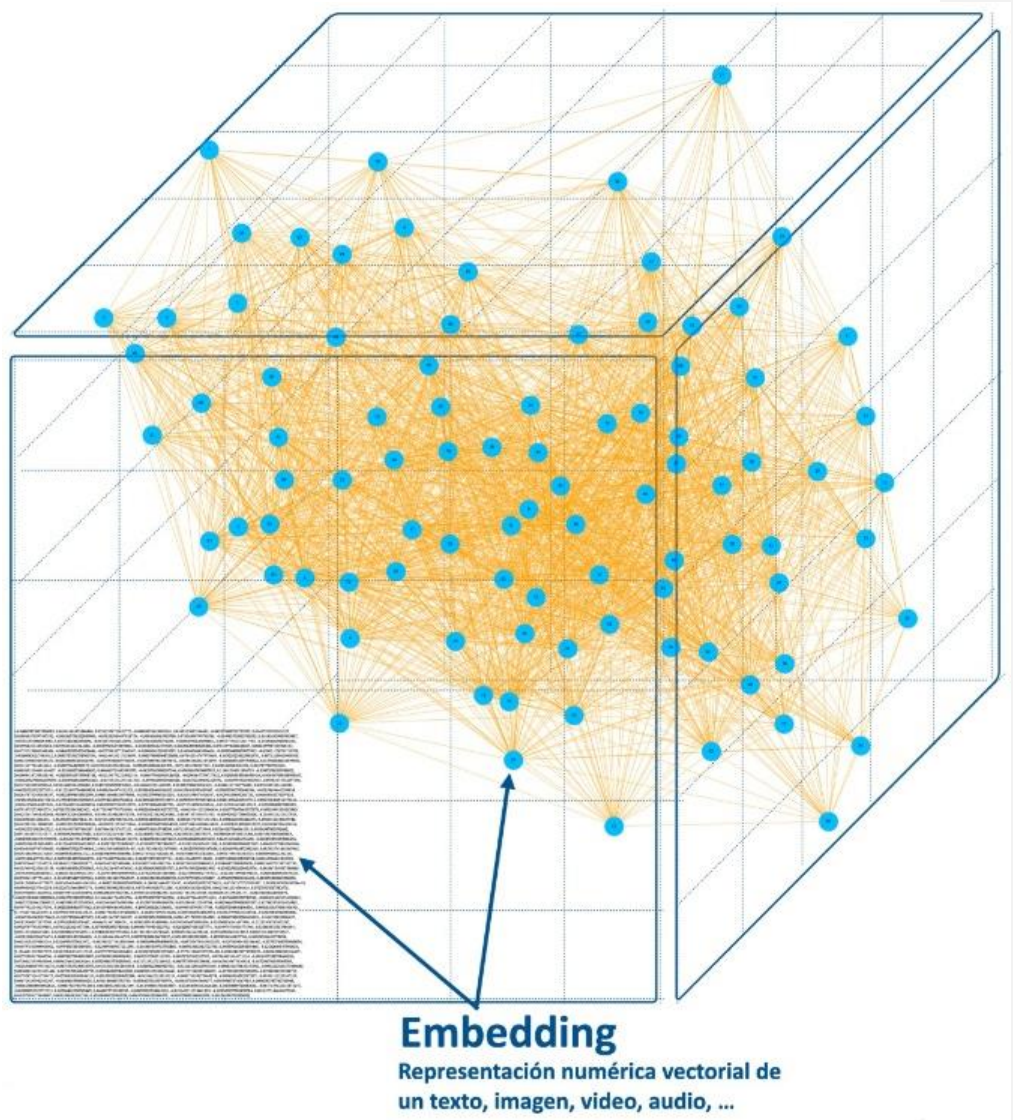
5.11. Base de Datos Vectoriales

“Las bases de datos vectoriales proporcionan la capacidad de almacenar y recuperar vectores como puntos de alta dimensión. Añaden capacidades adicionales para la búsqueda eficiente y rápida de los vecinos más cercanos en el espacio N-dimensional.” (Amazon Web Services, 2023).

Se emplean en búsquedas vectoriales, recomendaciones de productos y búsqueda conversacional, combinándose con IA generativa para acelerar el desarrollo de aplicaciones de IA. Ofrecen ventajas en seguridad, escalabilidad y gestión de recursos, aunque enfrentan desafíos como escalabilidad, precisión y rendimiento.

Figura 4

Representación visual de una base de datos vectorial



5.12.Metodología de Desarrollo

5.12.1. SCRUM

SCRUM es una metodología ágil que fomenta la colaboración y la adaptabilidad en equipos, especialmente en el desarrollo de software. Organiza equipos autoadministrados que entregan valor en períodos llamados sprints, permitiendo adaptarse a cambios y gestionarse eficientemente (Martins, 2024).

5.12.2. Fases de la Metodología Scrum

5.12.2.1. Sprint Planning

Durante esta fase los miembros del equipo se reúnen para planificar el sprint. Durante este evento se deciden las tareas que se le asignarán a cada miembro del equipo y el tiempo que necesita para completarla. De esta manera se define el tiempo de duración del Sprint.

5.12.2.2. Scrum Team Meeting

El equipo de trabajo debe realizar reuniones diarias que deben ser sesiones cortas para discutir el progreso, los obstáculos y las próximas tareas a abordar. Si alguno de los miembros tiene algún inconveniente que obligue a extender el encuentro, este debe tratarse más a fondo en una reunión enfocada en buscar la solución para ello.

5.12.2.3. Backlog Refinement

En esta fase, el Product Owner revisa cada elemento del Product Backlog con el fin de solventar dudas por parte del equipo de desarrolladores. También, sirve para evaluar el tiempo y esfuerzo empleados en cada tarea.

"El Product Owner es el encargado de optimizar y maximizar el valor del producto, siendo la persona responsable de gestionar el flujo de valor del producto a través del Product Backlog" (deloitte, 2024).

“El backlog de un producto es una lista de trabajo ordenado por prioridades para el equipo de desarrollo que se obtiene de la hoja de ruta y sus requisitos.” (atlassian, 2019).

5.12.2.4. Sprint Review

En la revisión del sprint, equipo y clientes se reúnen para exhibir el trabajo de desarrollo de software, demostrando los requisitos finalizados. La presentación la lideran el *Scrum Master* y el *Product Owner*. Estas sesiones, llamadas *Sprint Review*, incluyen al cliente, buscando mostrar resultados y obtener feedback real, fortaleciendo así una relación cercana y productiva.

5.13. Herramientas de Desarrollo

5.13.1. Python

Python es un lenguaje de programación interpretado, orientado a objetos, de alto nivel y con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con escritura y enlace dinámicos, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso como lenguaje de secuencias de comandos o pegamento para conectar componentes existentes. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y, por lo tanto, reduce el costo de mantenimiento del programa. Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código (*What Is Python? Executive Summary*, n.d.).

5.13.2. Django

Django es un marco de aplicación web que permite un desarrollo rápido y eficiente. Es popular por su código abierto, velocidad de desarrollo y respaldo de proyectos notables como Instagram. Utiliza la arquitectura Modelo-Vista-Template (MVT), donde los modelos interactúan con la base de datos, las vistas procesan solicitudes y las plantillas manejan la presentación en el navegador. Aunque puede parecer "obstinado"

en ciertas prácticas, ofrece flexibilidad a través de su arquitectura desacoplada y proporciona módulos como formularios y autenticación de usuarios para mejorar la funcionalidad de las aplicaciones web (*Introducción a Django - Aprende Desarrollo Web | MDN*, n.d.).

5.13.3. Langchain

"LangChain es un marco de trabajo de código abierto para crear aplicaciones basadas en LLM." (Amazon Web Services, 2023a).

LangChain facilita la personalización y precisión de la información generada, optimizando el desarrollo de aplicaciones al integrar estos modelos con datos externos. Con LangChain, los desarrolladores pueden reutilizar modelos en dominios específicos sin necesidad de reentrenamiento, simplificando el proceso de desarrollo de inteligencia artificial y contando con el respaldo de una comunidad activa.

5.13.4. RAGAS

Ragas es un framewok que ayuda a evaluar y cuantificar el rendimiento de los flujos de RAG. RAG denota una clase de aplicaciones LLM que utilizan datos externos para aumentar el contexto del LLM (*Introduction | Ragas*, n.d.).

6. Análisis y Presentación de Resultados

6.1. Planificación

El objetivo de esta fase es establecer las bases del proyecto, definir los requisitos funcionales y no funcionales, asignación de roles y organizar el trabajo de manera que cada sprint tenga objetivos claros y alcanzables. La planificación es fundamental para gestionar los tiempos de entrega y asegurar que el desarrollo del chatbot se realice de manera estructurada bajo la metodología Scrum.

6.1.1. Requerimientos Funcionales

Tabla 1

Requerimientos funcionales

Código	Descripción
RF01	El chatbot debe permitir a los usuarios crear nuevas conversaciones, visualizar el historial de conversaciones, acceder a conversaciones anteriores y eliminar conversaciones específicas de forma segura.
RF02	El sistema debe permitir el registro, autenticación y desactivación de usuarios.
RF03	El sistema debe permitir la asignación y gestión de roles de usuario.
RF04	El sistema debe extraer, procesar y almacenar información legal del Digesto Jurídico Nicaragüense mediante técnicas de web scraping, modelos de embeddings y bases de datos vectoriales para su posterior consulta.
RF05	Debe utilizar un sistema RAG que permita generar respuestas basadas en la información contenida en el contenida en el Digesto Jurídico Nicaragüense.
RF06	El sistema debe proporcionar referencias verificables

	sobre la fuente de la información utilizada en cada respuesta generada por el chatbot.
RF07	Debe contar con un proceso de evaluación de respuestas basado en métricas como fidelidad, precisión del contexto y relevancia.

6.1.2. Requerimientos no Funcionales

Tabla 2

Requerimientos no funcionales

Código	Descripción
RNF01	El chatbot debe ser escalable y capaz de manejar múltiples consultas simultáneas sin afectar el rendimiento.
RNF02	Debe implementar autenticación y autorización para garantizar la seguridad y privacidad de los datos de los usuarios y el sistema.
RNF03	La arquitectura debe ser modular y mantenible, permitiendo la actualización de componentes sin afectar el sistema global.
RNF04	Debe ser accesible desde navegadores modernos y dispositivos móviles sin pérdida de funcionalidad.
RNF05	La interfaz de usuario debe ser intuitiva y permitir la visualización clara de las respuestas generadas.

6.1.3. Historias de usuario

Basándose en los requerimientos funcionales y no funcionales del sistema, se han definido las historias de usuario, las cuales pueden descomponerse en subtareas para facilitar su desarrollo a lo largo de los distintos sprints. Dado que todas las historias de usuario son de alta prioridad, el *Scrum Master* será el encargado de establecer el orden de ejecución dentro de cada sprint.

Tabla 3

Historias de usuario

Número	Nombre	Descripción	Prioridad	Sprint
HDU1	Diseño de la interfaz de usuario	Como diseñador, quiero definir la interfaz de usuario del sistema, para asegurar una experiencia intuitiva y amigable para los usuarios finales.	Alta	1
HDU2	Diseño de la arquitectura del backend	Como arquitecto de software, quiero definir la estructura del backend del sistema, para garantizar su escalabilidad, mantenimiento y seguridad.	Alta	1
HDU3	Diseño de la base de datos relacional	Como desarrollador, quiero diseñar la base de datos relacional del sistema, para almacenar y gestionar la información de manera eficiente y estructurada.	Alta	1
HDU4	Creación de la base de datos relacional	Como desarrollador, quiero implementar la base de datos relacional diseñada, para almacenar de manera eficiente los datos del sistema.	Alta	2
HDU5	Recopilación, procesamiento y almacenamiento de información legal	Como administrador, quiero que el sistema recopile, procese y almacene información del Digesto Jurídico Nicaragüense, para que el chatbot pueda ofrecer	Alta	2

		respuestas basadas en información actualizada.		
HDU6	Desarrollo del servicio backend para gestión de la información legal	Como desarrollador, quiero implementar un servicio backend que permita gestionar la información legal, para facilitar su consulta.	Alta	3
HDU7	Desarrollo del servicio backend para gestión de usuarios	Como desarrollador, quiero crear un servicio backend que gestione los usuarios y sus roles, para controlar el acceso al sistema de manera segura.	Alta	3
HDU8	Desarrollo del servicio RAG para generación de respuestas	Como usuario, quiero que el chatbot genere respuestas basadas en información legal utilizando un sistema RAG, para obtener respuestas precisas y fundamentadas.	Alta	3
HDU9	Desarrollo de la interfaz de usuario	Como usuario, quiero una interfaz intuitiva y fácil de usar, para interactuar con el chatbot y acceder a la información legal de manera eficiente.	Alta	4
HDU10	Aplicación de métricas para evaluar desempeño y exactitud del chatbot	Como desarrollador, quiero aplicar métricas para evaluar la precisión, relevancia y exactitud de las respuestas del chatbot, para mejorar su rendimiento continuamente.	Alta	4

6.1.4. Tablero Kanban

El tablero Kanban permitirá visualizar el flujo de las tareas y posibles cuellos de botella.

La estructura del tablero constará de cuatro columnas:

1. Por hacer: En esta columna estarán todas las tareas que se realizarán en un sprint.
2. En curso: En esta columna estarán las tareas que se están trabajando actualmente.
3. Revisión de código: Una vez finalizada una tarea se hará una revisión de código por otro miembro del equipo.
4. Listo: Una vez finalizada la revisión de código la tarea pasará a la columna Listo.

Figura 5

Tablero Kanban en el software JIRA

6.1.5. Roles de SCRUM

La definición de roles permitirá delegar responsabilidades a los miembros del equipo.

Tabla 4

Asignación de roles de SCRUM

Rol	Funciones	Responsables
Product Owner	Se encarga de desarrollar la visión del proyecto asegurándose que se cumpla con los objetivos establecidos y actúa como un intermediario entre el equipo de desarrollo y los usuarios finales.	• Danny Chávez
SCRUM Master	Se encarga de la organización, planeamiento y la revisión del sprint. Este rol se irá alternando entre los dos miembros del equipo de desarrollo durante la ejecución del proyecto.	• Sjeff Gómez • Brandon Espinoza
Equipo de desarrollo	Se encargan del desarrollo de las historias de usuario.	• Sjeff Gómez • Brandon Espinoza

6.2. Sprint 1: Diseño y Arquitectura del Sistema

En este primer sprint, el enfoque principal es establecer las bases sobre las cuales se desarrollará el chatbot. Se define la estructura de la interfaz de usuario, la arquitectura del backend y la base de datos relacional que permitirá el almacenamiento y gestión eficiente de la información.

6.2.1. Diseño de la interfaz de usuario

Para el diseño de las diferentes vistas que conformarán el sistema se utilizó Figma un software que permite el diseño de interfaces gráficas. Estos diseños se ocupan como una referencia para el desarrollo de la interfaz gráfica así que el resultado final podría diferir un poco de lo mostrado en las siguientes ilustraciones.

Figura 6

Diseño de formularios de creación de cuenta y formulario de inicio de sesión

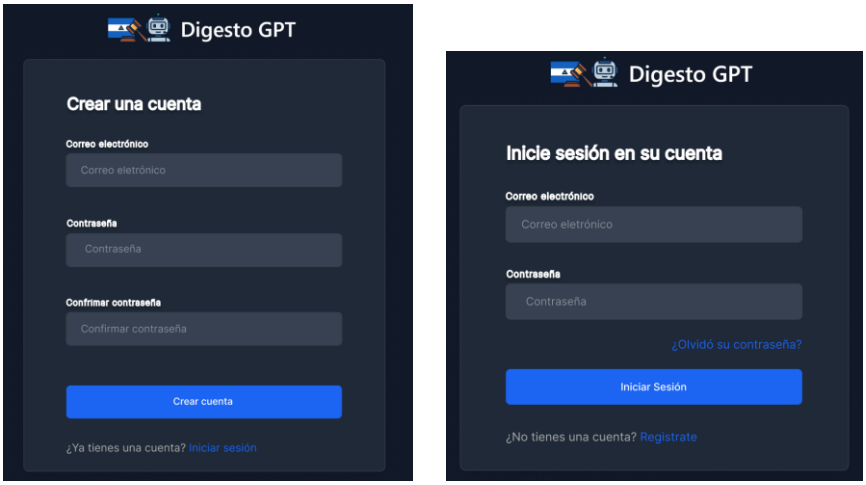


Figura 7

Vista principal del chat

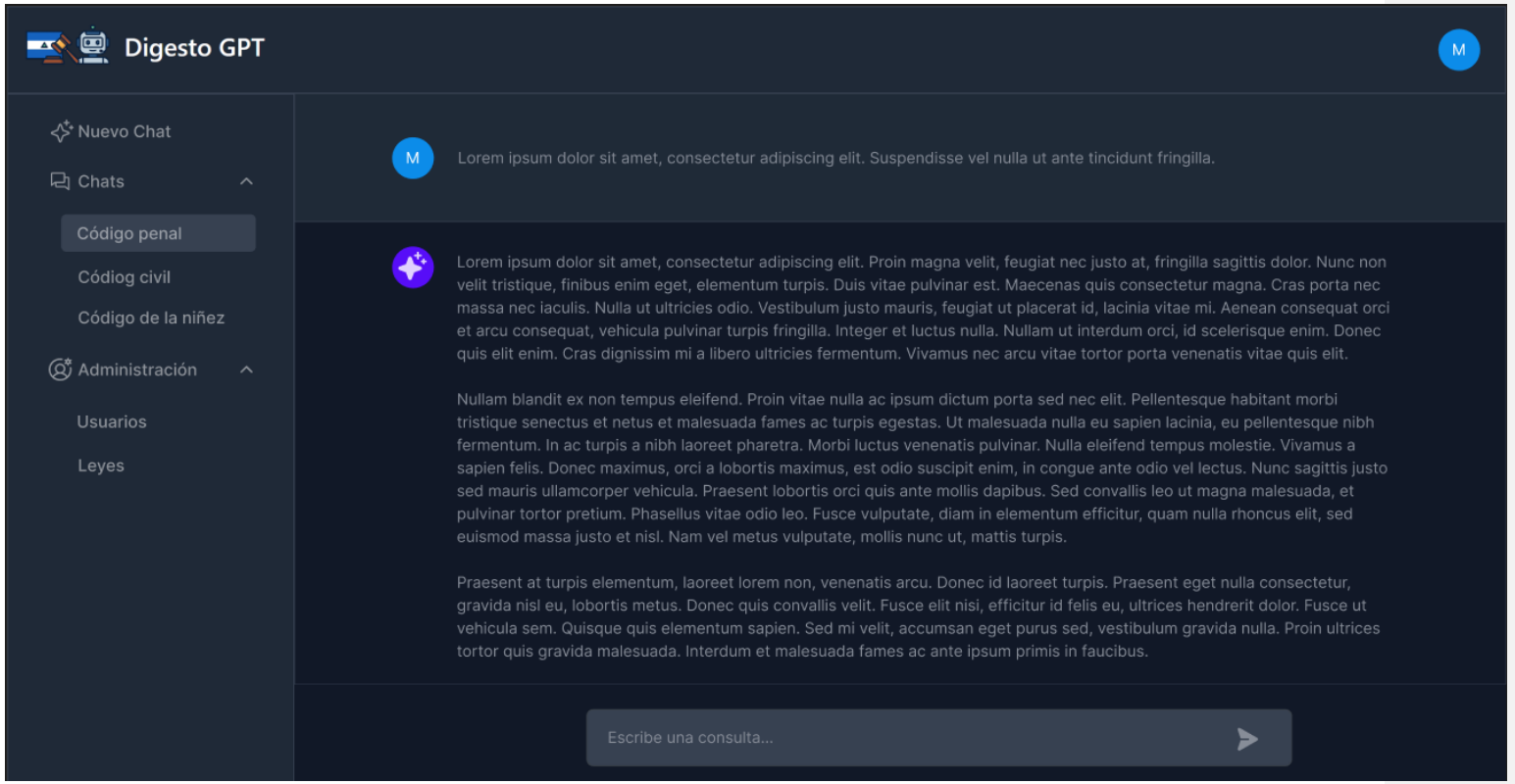
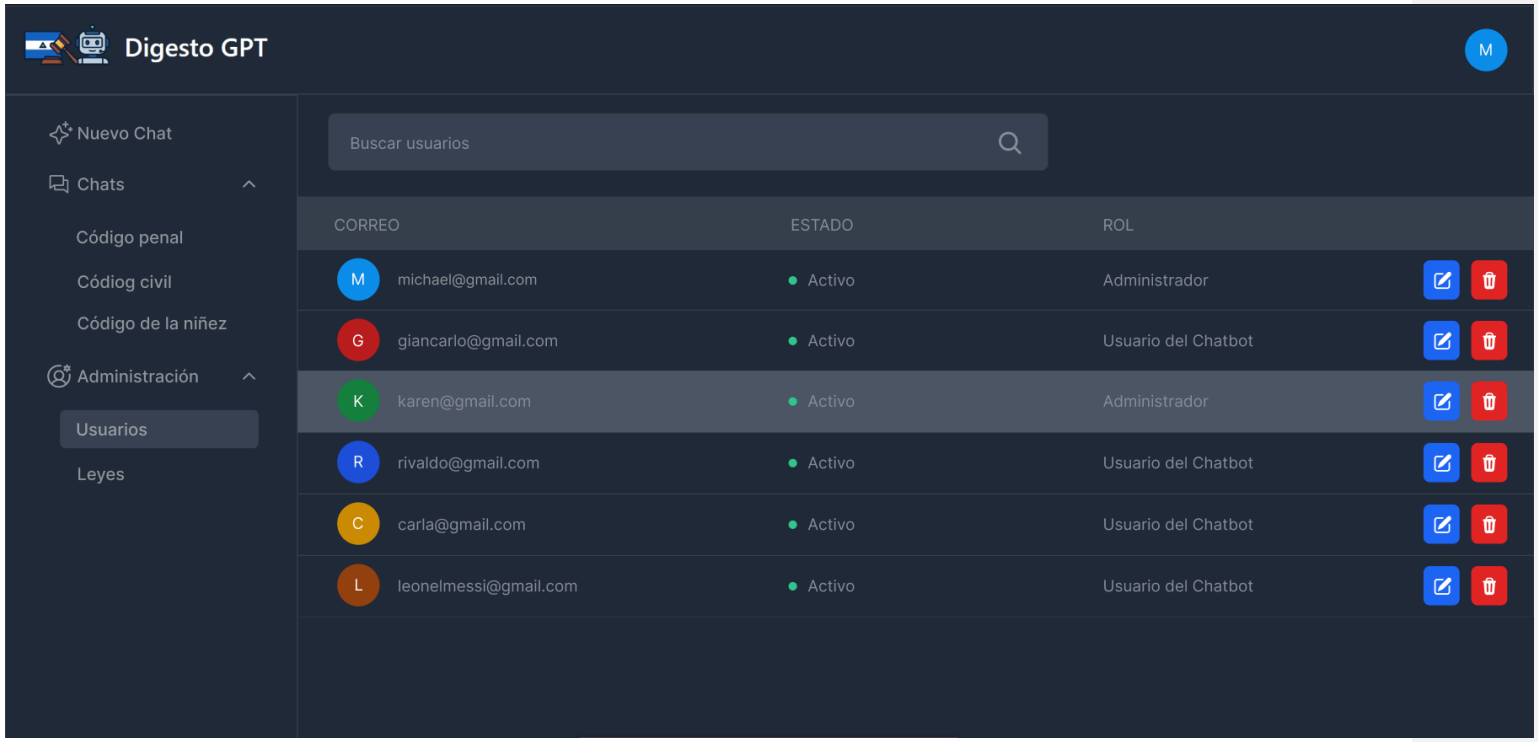


Figura 8

Vista de administración de usuarios



6.2.2. Diseño de la arquitectura del backend

El diseño de la arquitectura del backend es un componente fundamental en el desarrollo del chatbot, ya que define la estructura y funcionamiento del sistema en términos de procesamiento, almacenamiento y comunicación entre sus distintos módulos. En esta sección, se detallan los elementos clave que conforman el backend, garantizando un diseño escalable, eficiente y alineado con los requerimientos del proyecto.

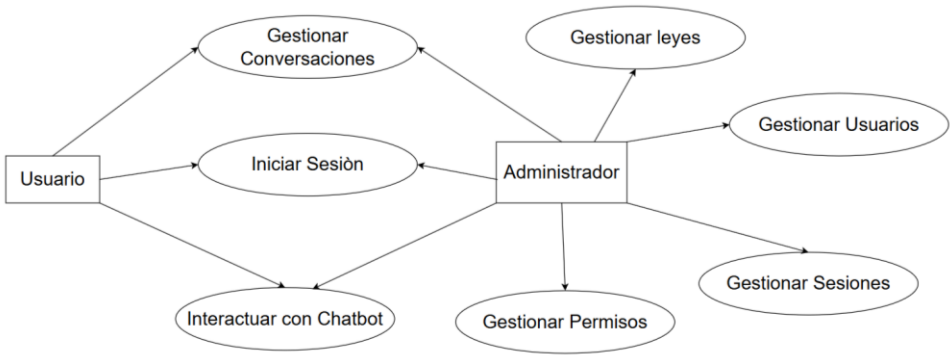
6.2.2.1. Diagramas de casos de uso

El diagrama de casos de uso tiene como objetivo representar las principales interacciones de los usuarios con el sistema, permitiendo visualizar de manera clara las funcionalidades disponibles según el rol asignado. En este proyecto, se identifican dos tipos principales de usuarios:

- Usuarios del chatbot: Tienen acceso exclusivo a la funcionalidad de consulta, permitiéndoles interactuar con el chatbot para obtener información legal.
- Usuarios admistradores: Poseen un rol con mayores privilegios, lo que les permite gestionar usuarios, administrar leyes y realizar consultas al chatbot.

Figura 9

Casos de uso



6.2.2.2. Diagramas de Interacción

Los diagramas de interacción permiten visualizar la comunicación entre los distintos componentes del sistema durante la ejecución de procesos clave, en este caso cuando un usuario realiza una consulta al chatbot y cuando se realizan operaciones que implican lecturas o escrituras en la base de datos relacional.

Figura 10

Operaciones de lectura y escritura a la base de datos relacional

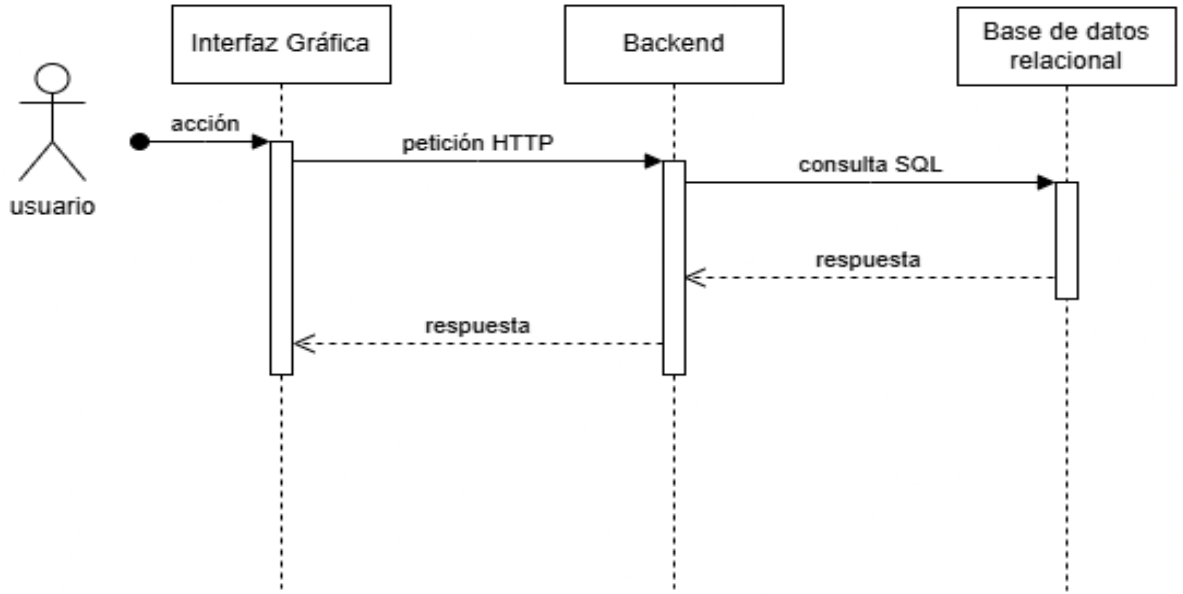
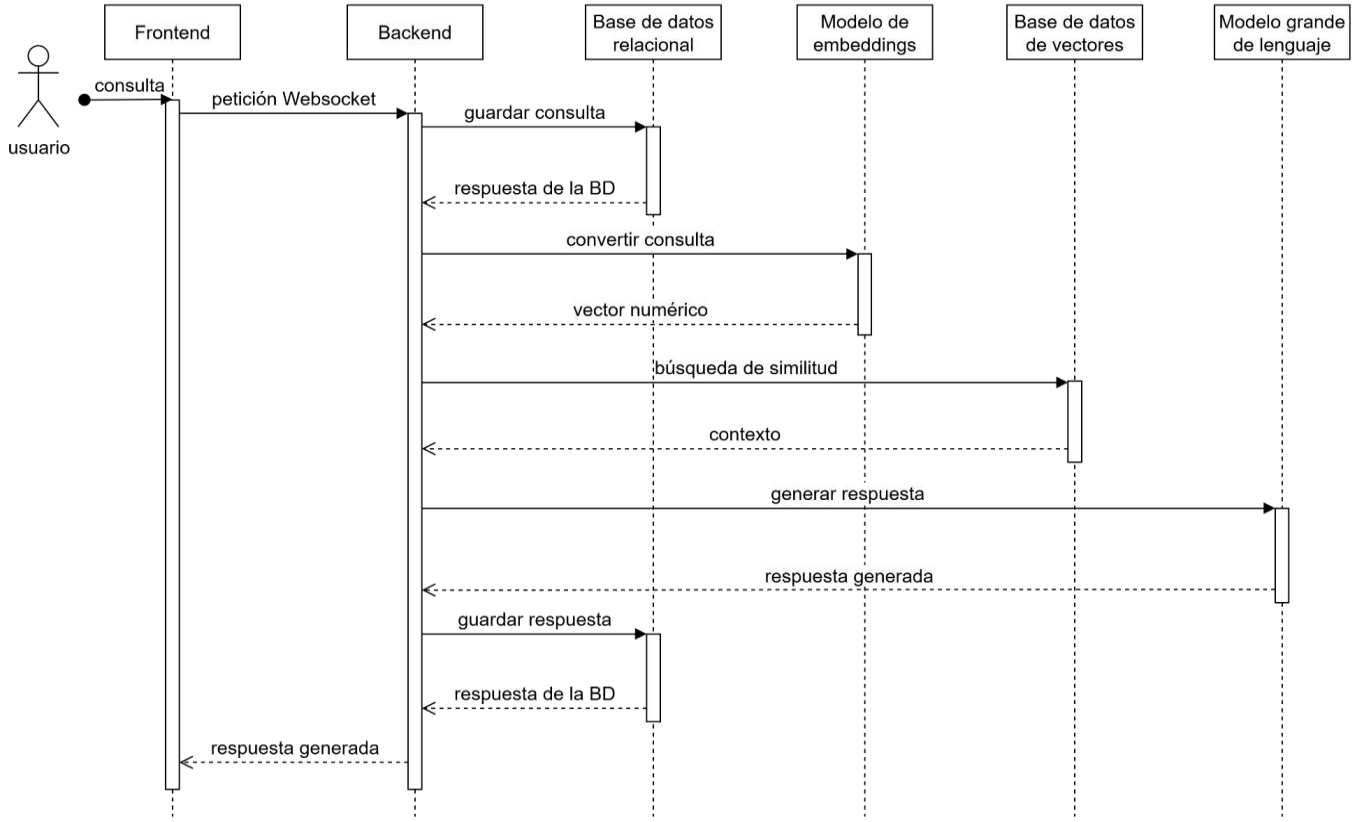


Figura 11

Realizar una consulta al chatbot



6.2.2.3. Diseño de la arquitectura general del sistema

Para el despliegue del sistema, se utilizará AWS, donde se iniciará una instancia de EC2 para alojar los diferentes componentes. Cada uno de estos estará encapsulado en un contenedor Docker, lo que facilitará su despliegue, mantenimiento y escalabilidad.

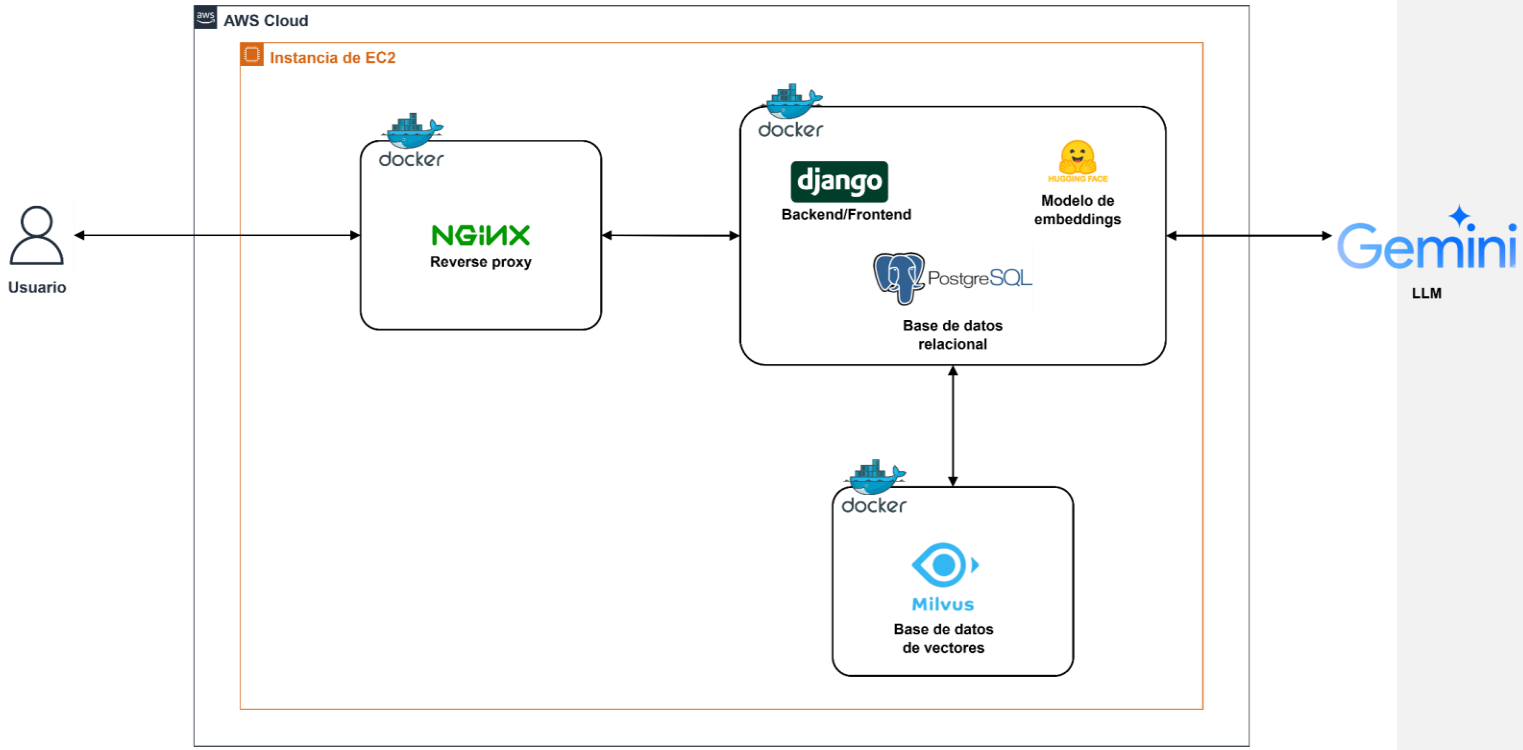
La arquitectura incluirá un servidor Nginx configurado como reverse proxy, permitiendo gestionar las solicitudes de los usuarios y servir archivos estáticos. El backend y frontend estará desarrollado en Django, el cual manejará la lógica del sistema y se comunicará con la base de datos relacional Postgresql para el almacenamiento estructurado de información.

Para la implementación del sistema RAG, se utilizará Milvus como base de datos de vectores para el almacenamiento y recuperación eficiente de representaciones semánticas. Se integrará un modelo de embeddings de Hugging Face para transformar consultas en representaciones vectoriales y se empleará Gemini como modelo de lenguaje, encargado de generar respuestas precisas y contextualizadas a partir de los datos recuperados.

Esta arquitectura modular y basada en contenedores garantiza un despliegue flexible, optimizando la búsqueda y generación de información legal de manera eficiente y escalable.

Figura 12

Diagrama de arquitectura general del sistema

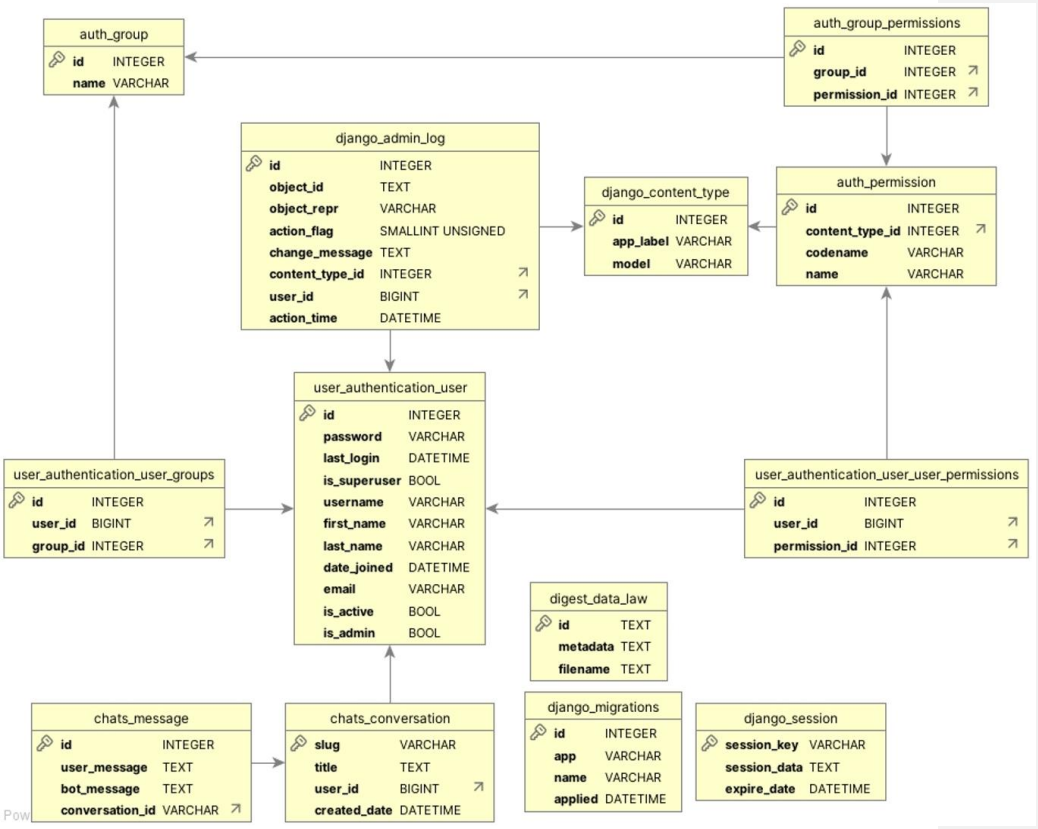


6.2.2.4.Diseño de la base de datos relacional

El diseño de la base de datos relacional es un aspecto fundamental para garantizar un almacenamiento estructurado y eficiente de la información dentro del sistema. En este caso, se ha optado por utilizar **SQLite** como motor de base de datos debido a su facilidad de integración con Django y su idoneidad para el desarrollo inicial del sistema. La base de datos está diseñada para gestionar tanto la información legal como los usuarios del sistema, permitiendo una administración estructurada de los datos.

Figura 13

Diagrama entidad relación



6.2.3. Revisión del Sprint

El propósito de este primer sprint fue establecer las bases del proyecto, incluyendo el diseño de la interfaz de usuario, la arquitectura del backend y la estructura de la base de datos.

Diseño de la interfaz de usuario

- Se crearon prototipos en Figma, proporcionando una primera versión visual del sistema.
- Se diseñaron las vistas clave:
    - Creación de cuenta
    - Inicio de sesión
    - Interfaz del chatbot
    - Interfaz del chatbot

Definición de la arquitectura del backend

- Se establecieron los componentes principales y el flujo de datos.
- Se definieron las tecnologías a utilizar, asegurando una infraestructura escalable.
- Casos de uso: Se identificaron los principales escenarios de interacción del sistema.
- Diagramas de interacción: Se modeló la comunicación entre componentes, cubriendo operaciones CRUD y consultas al chatbot.
- Arquitectura general del sistema: Se documentó la estructura global del sistema y la relación entre sus módulos.

Modelado de la base de datos relacional

- Se diseñó el diagrama entidad-relación (ER).
- Se establecieron relaciones entre entidades clave.

6.3. Sprint 2: Construcción de la base de datos y procesamiento de información legal

6.3.1. Creación de la base de datos relacional

Para la creación de la base de datos relacional se utilizó **Django ORM**, una herramienta que permite representar las entidades de la base de datos como clases de Python, conocidas como *modelos*. A continuación, se describen los modelos implementados para almacenar la información necesaria para el funcionamiento del sistema.

Modelo de Usuarios

Este modelo almacena la información de los usuarios del sistema. Se utiliza el **correo electrónico como clave primaria** y se implementan campos adicionales para la autenticación y autorización.

```
from django.contrib.auth.models import AbstractUser
from django.db import models


class User(AbstractUser):
    # Redefine el campo email como obligatorio, único y con longitud máxima
    email = models.EmailField(
        verbose_name="email address",
        max_length=255,
        unique=True,
    )
    # Campo que indica si el usuario está activo
    is_active = models.BooleanField(default=True)
    # Campo que indica si el usuario es administrador
    is_admin = models.BooleanField(default=False)
```

Modelo de Conversaciones y Mensajes

Estos modelos gestionan las conversaciones realizadas por los usuarios al chatbot, así como los mensajes intercambiados. Cada usuario puede tener múltiples conversaciones, y cada conversación puede contener varios mensajes.

```
from django.db import models
from django.utils import timezone

from apps.user_authentication.models import User


class Conversation(models.Model):
    # Clave primaria
    slug = models.SlugField(unique=True, primary_key=True)
    # Relación uno a muchos con el modelo ususario
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    # Título de la conversación
    title = models.TextField()
    # Fecha de creación
    created_date = models.DateTimeField(default=timezone.now)


class Message(models.Model):
    # Relación uno a muchos con el modelo conversación
    conversation = models.ForeignKey(Conversation, on_delete=models.CASCADE)
    # Mensaje del usuario
    user_message = models.TextField()
    # Mensaje del chatbot
    bot_message = models.TextField()
    # Fecha de creación
    created_date = models.DateTimeField(default=timezone.now)
```

Modelo de Leyes

Este modelo almacena información relacionada a la información del Digesto. Dicha información se guarda en la columna metadata en formato **JSON**, la cual contiene datos como el título de la ley, su número, fecha de aprobación, fecha publicación, entre otros. Cabe destacar que el contenido completo de las leyes no se almacena en la base de datos, sino que se encuentra en **ficheros almacenados en el sistema de archivos del servidor**.

```
from django.db import models
class Law(models.Model):
    # Clave primaria
    id = models.TextField(
        primary_key=True,
        unique=True,
    )
    # En este campo se almacena el detalle de la ley
    metadata = models.JSONField()
    # Nombre del archivo
    filename = models.TextField()
```

6.3.2. Desarrollo de los componentes del servicio RAG

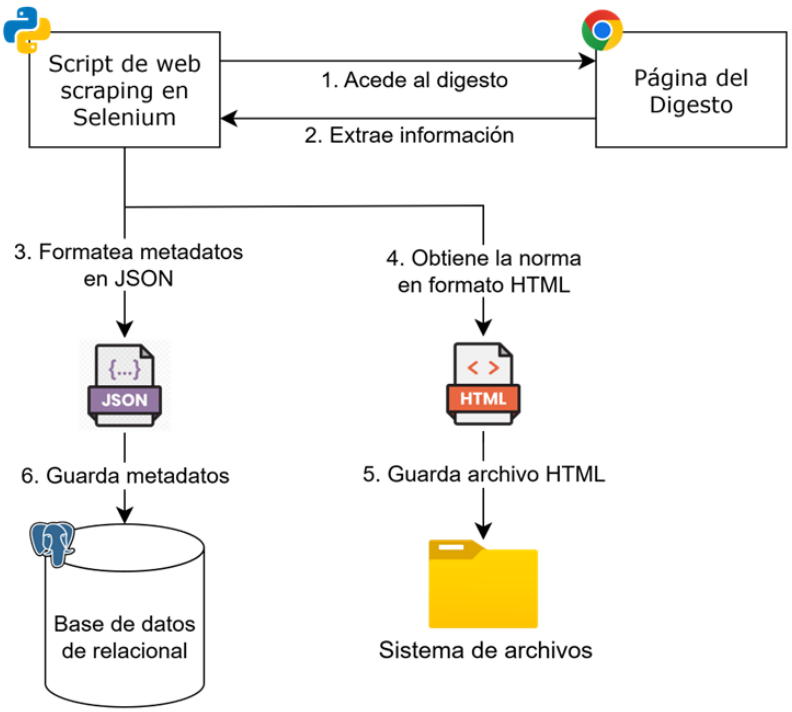
En esta sección, se detallan los componentes que integran el sistema RAG. El desarrollo se estructuró en tres fases fundamentales. La primera etapa consistió en la recopilación de datos mediante técnicas de web scraping. Posteriormente, en la segunda fase, se procedió a la indexación de la información; para ello, se emplearon modelos de embeddings densos y dispersos, almacenando los resultados en una base de datos vectorial. Finalmente, la tercera etapa corresponde a la fase de inferencia, en la cual el sistema genera respuestas a las consultas de los usuarios, integrando los datos indexados con un LLM.

Recopilación de datos

La información fue extraída desde el sitio web del Digesto Jurídico Nicaragüense utilizando la librería **Selenium**, la cual permite realizar **web scraping**, que se refiere al proceso de recolectar datos desde la web, ya sea de forma manual o automática. Este proceso fue automatizado mediante un script de Selenium escrito en Python que utiliza **ChromeDriver**, el cual es una API que permite la manipulación del navegador a través de código de programación, esto para simular la navegación del usuario a través de la página del digesto, accediendo a distintas secciones del sitio para extraer la información de interés, que luego será procesada.

Figura 14

Flujo de datos en la recopilación de la información



La estrategia para recolectar la información fue la siguiente:

- En la página del digesto la información legal está organizada por materia. Así que el primer paso fue acceder a la sección del digesto donde están listadas las materias disponibles. Para este propósito se inicializó **ChromeDriver** y utilizando el método **get** se accedió al sitio, a como se describe en las siguientes líneas de código:

```
# Define el servicio del driver indicando la ruta del ejecutable de ChromeDriver
webdriver_service = Service(
      f"{WEB_DRIVER_DIR}/chromedriver-linux64/chromedriver"
)
# Crea una instancia del navegador Chrome con las opciones especificadas
browser = webdriver.Chrome(
      service=webdriver_service, options=chrome_options
)
# Abre la URL del Digesto Jurídico en el navegador
browser.get("http://digesto.asamblea.gob.ni/consultas/digestos/")
```

- Una vez en esta sección del sitio, se procede a leer la tabla que contiene el listado de materias, utilizando el método **presence_of_element_located** el cual permite obtener elementos de HTML por id, a como se describe en la siguiente línea de código:

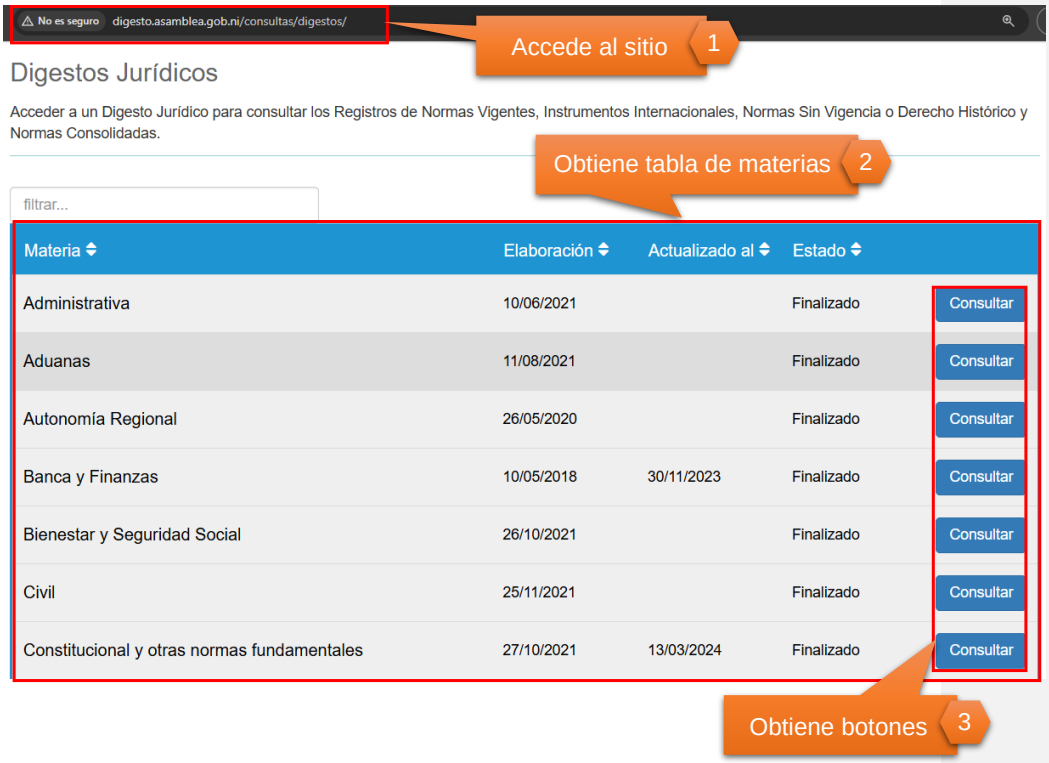
```
table = wait.until(EC.presence_of_element_located((
      By.ID, "tableDigestos"
)))
```

- Una vez obtenida esta tabla se extraen los botones correspondientes para acceder a cada una de las materias, esto se hace aplicando el método **find_elements** sobre esta tabla, el cual permite extraer elementos de HTML por nombre de etiqueta, esta función retorna una lista de botones que, luego se itera a través de un bucle **for**, donde en cada iteración se accede a cada una de las materias mediante el método **click**, a como se muestra en el siguiente código:

```
buttons = table.find_elements(By.TAG_NAME, "button")
for button in buttons:
    button.click()
```

Figura 15

Listado de materias en el digesto



- Una vez se accede a una materia, la información está organizada en una tabla, pero al ser muchos documentos, esta tabla está dividida por páginas de 100 elementos cada una, así que, para simular la navegación por las páginas de la tabla, lo primero que se hace es calcular el total de páginas, ya que no es un dato que se brinda de manera explícita. Para realizar este cálculo, se obtiene el total de documentos que contiene la materia y se divide entre 100, luego se define un bucle **for** que tiene un límite de iteraciones definido por el total de páginas, como se muestra en el siguiente código:

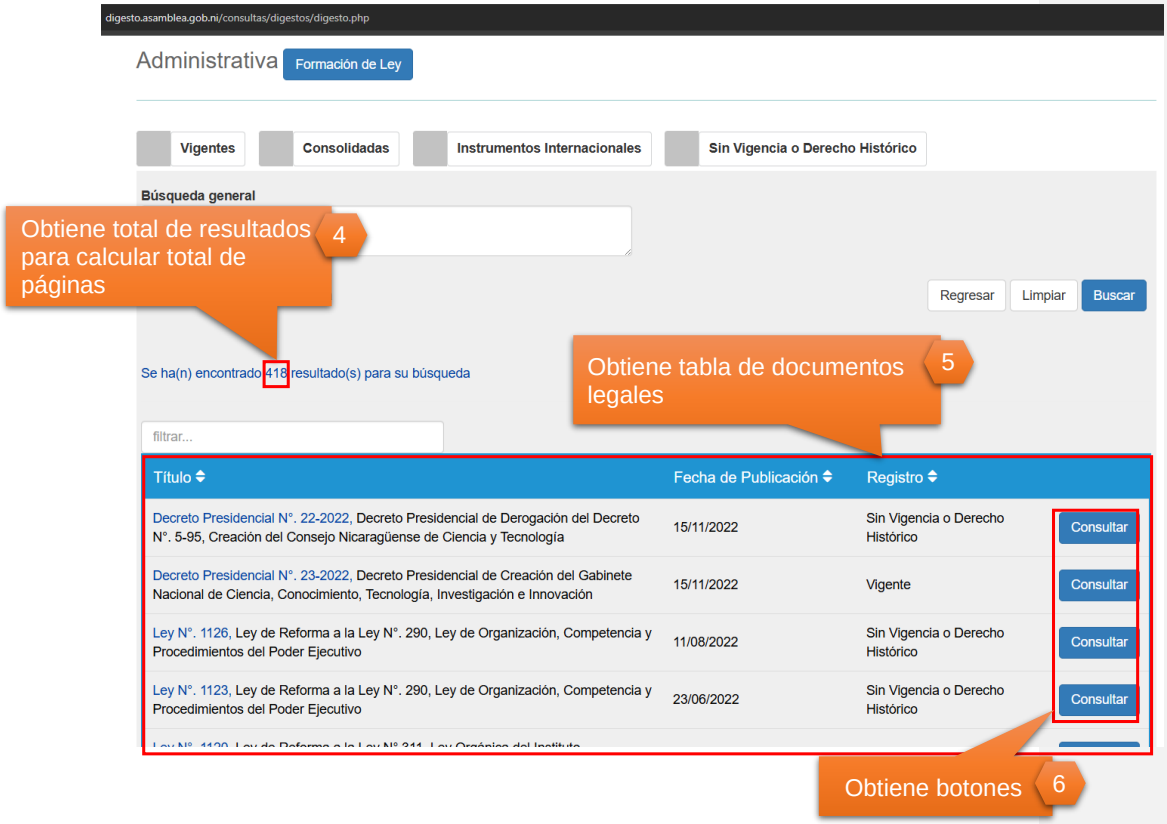
```
# Obtiene el texto del mensaje que indica la cantidad de resultados en
la tabla
filter_result_message = (
    browser.find_element(By.CLASS_NAME, "resultados")
    .find_element(By.TAG_NAME, "p")
    .text
)

# Extrae el primer número del mensaje usando una expresión regular
(total de resultados)
filter_result_total = re.search(r"\d+", filter_result_message).group()

# Calcula el total de páginas
pag_total = ceil(
    int(filter_result_total) / 100
)
# Itera sobre el rango de páginas para realizar la paginación
for pag_number in range(0, pag_total):
```

- Una vez establecida la lógica de paginación, se procede a obtener la tabla de la página actual y los botones correspondientes para acceder a cada uno de los documentos, similar a los primeros pasos, esto se hace utilizando nuevamente el método **presence_of_element_located**, al cual se le pasa como parámetro el id de la tabla, para luego extraer los botones de esta tabla utilizando el método **find_elements**, lo que devuelve una lista de botones, la cual luego se itera a través de un bucle **for** y por cada iteración se accede a cada una de las leyes, utilizando el método **click**, a como se muestra en el siguiente código:

```
# Espera hasta que la tabla con ID "tableIunps" esté presente en el DOM
y la asigna a 'law_table'
law_table = wait.until(EC.presence_of_element_located((
      By.ID, "tableIunps"
)))

# Obtiene todos los elementos <button> dentro de la tabla
law_buttons = law_table.find_elements(By.TAG_NAME, "button")

# Itera sobre cada botón encontrado
for law_button in law_buttons:
    # Elimina las solicitudes anteriores almacenadas por el navegador
    del browser.requests

    # Hace clic en el botón actual para acceder a la ley correspondiente
al botón
    law_button.click()
```

Figura 16

Listado de documentos legales en materia administrativa

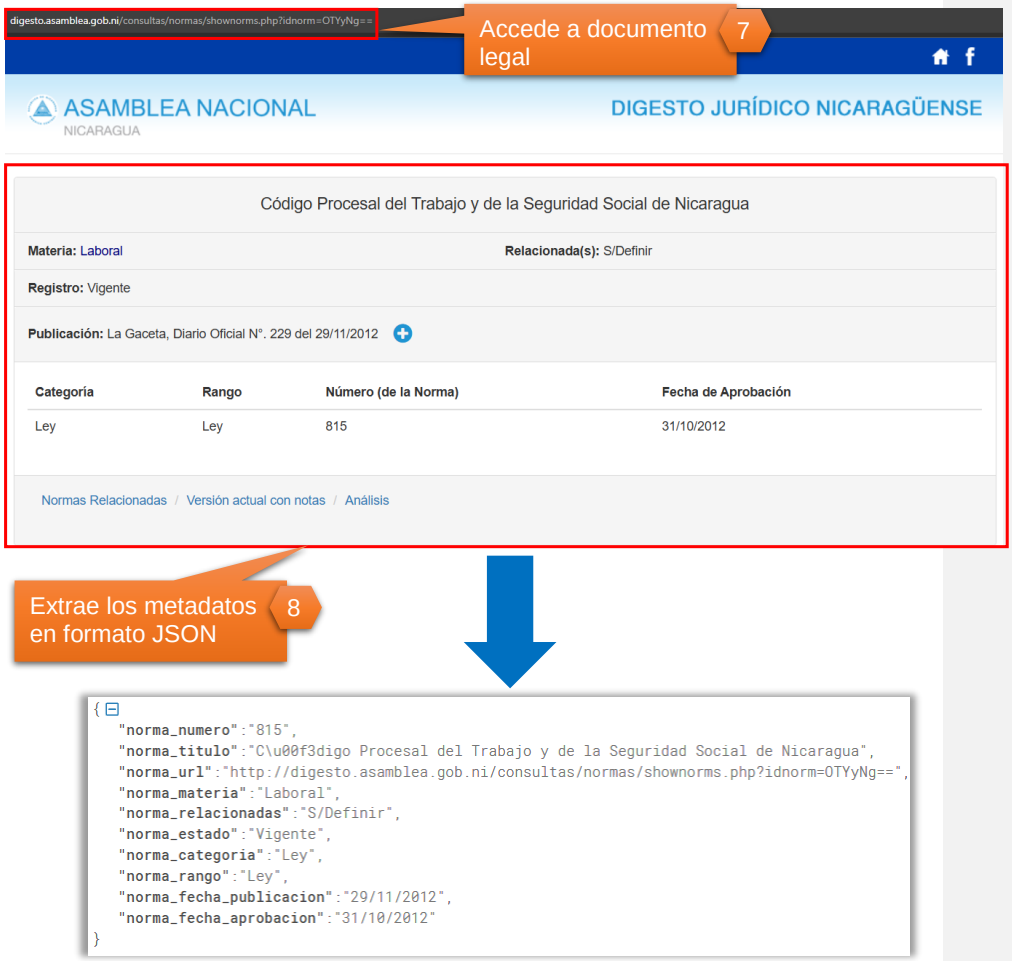


- Una vez se accede a un documento legal especifico, se procede a leer la url actual utilizando el método **current_url**, además de esto se leen datos relevantes como el título del documento legal, el numero de la norma, fecha de aprobación, etc. Esto utilizando el método **find_element**, para obtener cada uno de estos datos por id. Luego estos datos obtenidos se formatean en un diccionario. Esto se implementó utilizando las siguientes líneas de código:

```
# Obtiene las materias relacionadas y principales de la norma
subjects_related = browser.find_element(
    By.ID, "divMateriasPrin"
).find_elements(By.TAG_NAME, "td")

# Extrae y construye un diccionario con los metadatos de la norma
metadata = {
    "norma_numero": browser.find_element(By.ID, "tdNumero").text,
    "norma_titulo": browser.find_element(By.ID, "divTitle").text,
    "norma_url": browser.current_url,
    "norma_materia": subjects_related[0].text.split(":")[-1].strip(),
    "norma_relacionadas": subjects_related[1].text.split(":")[-
1].strip(),
    "norma_estado": browser.find_element(By.ID,
"divRegistro").text.split(":")[-1].strip(),
    "norma_categoria": browser.find_element(By.ID, "tdCategoria").text,
    "norma_rango": browser.find_element(By.ID, "tdRango").text,
    "norma_fecha_aprobacion": browser.find_element(By.ID,
"tdfaprobacion").text,
}
```

Figura 17

Documento legal sobre materia administrativa



- Luego se lee el texto que contiene la información legal, esto utilizando el método **presence_of_element_located**, al cual se le pasa como parámetro la clase de CSS **“document act”**, que especifica el segmento donde se encuentra la información legal. Una vez se obtiene esta información se guarda en un archivo HTML en el sistema de archivos del servidor, utilizando como nombre un uuid que servirá de identificador único para cada archivo, luego los metadatos obtenidos en el paso anterior se guardan en la base de datos relacional utilizando el uuid generado, como clave primaria, y de esta manera asegurar que cada archivo HTML quede asociado a su conjunto de metadatos correspondientes, para poder ser utilizados en etapas posteriores:

```
# Espera hasta que el contenido de la ley (con clase "document act")
esté presente en el DOM
law_content = wait.until(EC.presence_of_element_located(
        (By.CSS_SELECTOR, ".document.act")
))
# Genera un identificador único (UUID) para la ley
law_id = str(uuid.uuid4())
# Define el nombre del archivo HTML usando el UUID
filename = law_id + ".html"
# Abre (o crea) un archivo HTML en modo escritura dentro del directorio
de descarga
with open(
    LAWS_DOWNLOAD_DIR + "/" + filename, "w"
) as law_file:
    # Escribe el contenido HTML completo del elemento de la ley en el
archivo
    law_file.write(law_content)
# Crea una nueva instancia del modelo Law con su ID, metadatos y nombre
de archivo
new_law = Law(id=law_id, metadata=metadata, filename=filename)
# Guarda la nueva ley en la base de datos relacional
new_law.save()
```

Figura 18

Texto que contiene la información del documento legal

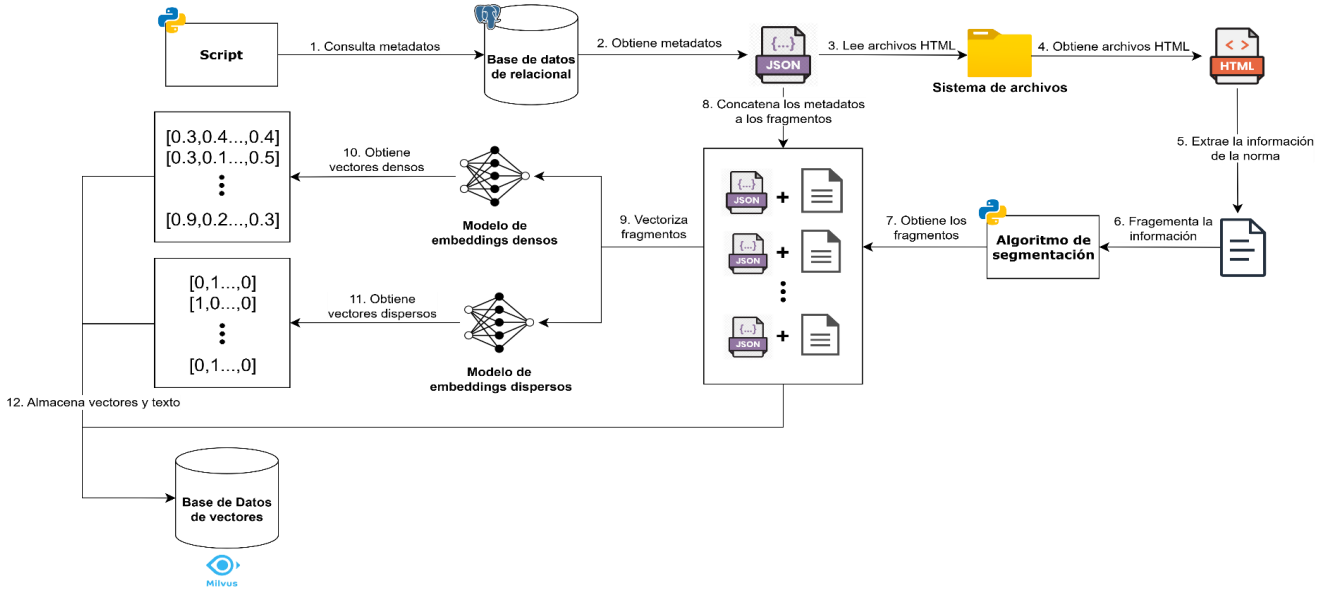
Este proceso se realizó por cada uno de los documentos legales contenidos en las materias disponibles. Al finalizar la ejecución del script de web scraping, se obtuvieron un total de 8,271 archivos HTML que contienen la información recopilada del Digesto Jurídico. Estos datos abarcan hasta el 10 de abril de 2025.

Indexación de datos

Para esta etapa se desarrolló un script de Python mediante el cual se automatiza la lectura, procesamiento y almacenamiento de los datos que luego serán utilizados como contexto en la generación de respuestas a consultas realizadas por los usuarios.

Figura 19

Flujo de datos del procesamiento y almacenamiento de la información legal



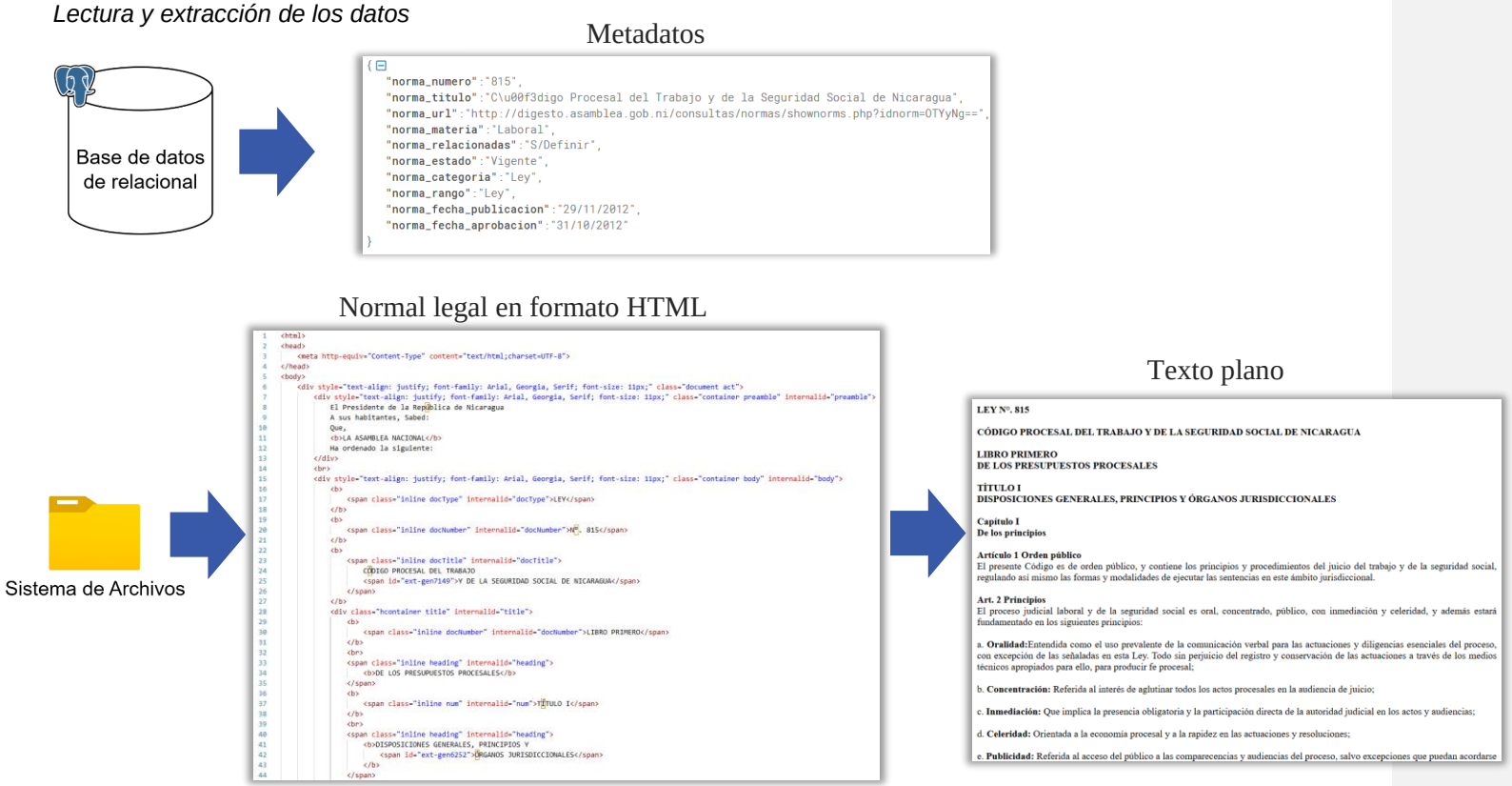
A continuación, se describen cada uno de los componentes de esta implementación:

Lectura de datos: Inicialmente, se consultaron los metadatos almacenados en la base de datos relacional. Posteriormente, se utilizó la clave primaria de cada registro para acceder a los archivos HTML que contienen el contenido de las normas, ya que dichos archivos fueron nombrados utilizando estas claves primarias. Esto permitió asociar de forma precisa cada archivo con sus metadatos correspondientes. Posterior a esto utilizando la librería **Beautifulsoup** se extrae el texto plano de los archivos HTML, descartando todas etiquetas de HTML, a como se muestra en el siguiente código.

```
from bs4 import BeautifulSoup
from apps.digest_data.models import Law

# Iteramos sobre cada ley no procesada
for law in not_proccessed_laws:
    # Abrimos el archivo HTML asociado a la ley usando su ID como nombre de
archivo
    with open(os.path.join(LAWS_DIR, law.id)) as law_file:
        # Parseamos el contenido HTML con BeautifulSoup
        soup = BeautifulSoup(law_file.read(), "html.parser")

        # Extraemos solo el texto plano del HTML
        law_text = soup.text
```

Figura 20

Lectura y extracción de los datos



Fragmentación de texto: Una vez obtenido el texto limpio a partir del contenido HTML, se procedió a realizar la **fragmentación semántica** del mismo. El objetivo principal de esta etapa es dividir el texto legal en segmentos más pequeños y coherentes que conserven una unidad temática, lo cual es fundamental para optimizar el uso de la ventana de contexto del LLM. Este procedimiento permite evitar el uso innecesario de información redundante o irrelevante, además de mejorar la eficiencia en la generación de respuestas.

Para llevar a cabo esta tarea, se empleó el algoritmo de **fragmentación segmentación** proporcionado por la clase **StatisticalChunker**, el cual implementa un enfoque basado en similitud semántica entre fragmentos consecutivos. El proceso se puede describir en los siguientes pasos:

1. **División inicial del texto**: El texto completo se divide en unidades más pequeñas (por ejemplo, oraciones o párrafos), utilizando una expresión regular para identificar los puntos donde se deben separar las oraciones dentro de un texto. Esta implementación toma en cuenta excepciones comunes del lenguaje, como abreviaturas, iniciales, números decimales y signos de puntuación combinados, para evitar cortes incorrectos.

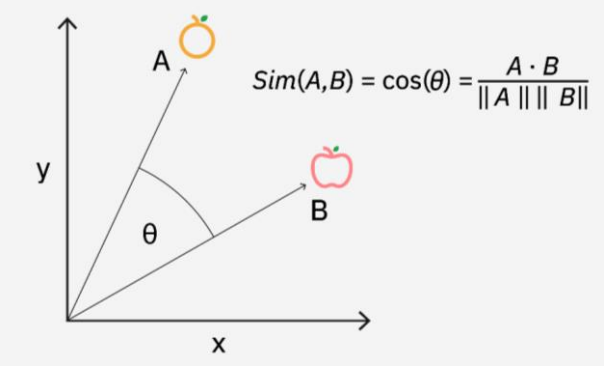
2. **Representación vectorial**: Cada fragmento inicial es convertido en un vector semántico utilizando un modelo de **embeddings densos**, en este caso, el modelo intfloat/multilingual-e5-large-instruct. Este modelo transforma el contenido textual en una representación numérica de alta dimensión, en la cual fragmentos con significados similares están cercanos entre sí en el espacio vectorial.

3. **Cálculo de similitud**: Se calcula la **similitud coseno** entre cada fragmento y un promedio de los fragmentos anteriores (una ventana de contexto). Esto permite evaluar el grado de continuidad semántica entre las partes del texto. La similitud del coseno es el cálculo del coseno del ángulo entre los dos vectores, para indicar qué tan cerca están alineados los vectores entre sí.

Matemáticamente, la similitud del coseno, cos(θ), entre dos vectores se calcula como el producto escalar de los dos vectores dividido por el producto de sus magnitudes.

Figura 21

Fórmula de similitud del coseno



4. **Identificación de puntos de corte**: Si la similitud entre fragmentos consecutivos cae por debajo de un umbral determinado, se considera que existe un cambio temático o de contexto, y, por lo tanto, se establece un punto de corte. Este umbral puede ser fijo o determinado dinámicamente en función de los datos. En este caso se utilizó el umbral fijo por defecto que es de 0.5.

5. **Generación de fragmentos finales**: A partir de los puntos de corte identificados, se forman los nuevos fragmentos o "chunks", los cuales representan unidades de texto coherentes, listas para ser utilizadas como contexto para que el LLM puede generar una respuesta.

Este enfoque es especialmente útil en el contexto legal, donde las normas pueden abarcar múltiples temas dentro de un mismo documento. La segmentación semántica garantiza que cada fragmento preserve una unidad temática, facilitando así su posterior procesamiento en la generación de respuestas.

La implementación se realizó utilizando las siguientes instrucciones en código:

```
from semantic_chunkers import StatisticalChunker

from semantic_router.encoders import HuggingFaceEncoder
# Se instancia el chunker utilizando un modelo de embeddings E5 multilingüe
chunker = StatisticalChunker(
    HuggingFaceEncoder(
        name="intfloat/multilingual-e5-large-instruct",  # Modelo de embeddings
densos
    ),
    max_split_tokens=500,   # Límite de tokens por fragmento
)
# Se aplica el chunker al texto legal, produciendo una lista de fragmentos
semánticos
chunks = chunker(docs=[law_text])
```

Modelo de embeddings densos: Para representar el contenido textual de manera semántica y contextualizada, se empleó el modelo **intfloat/multilingual-e5-large-instruct**, una variante multilingüe optimizada del modelo E5 (*Embedding Encoder Enhanced with Instructions*), desarrollado por Hugging Face y el equipo de intfloat (Wang et al., 2024). Este modelo pertenece a la familia de transformadores preentrenados y fue diseñado específicamente para tareas de búsqueda semántica y recuperación de información (*retrieval*) mediante la técnica de **sentence embeddings**, es decir, vectores densos que capturan el significado completo de oraciones, párrafos o consultas.

A diferencia de los modelos dispersos, los **embeddings densos** representan cada texto (ya sea una consulta o un documento) como un vector continuo de dimensiones fijas en un espacio de alta dimensión, en este caso de 1024 dimensiones. Cada dimensión de este vector codifica información semántica extraída del texto, permitiendo medir la similitud entre textos mediante métricas como la **similaridad del coseno**. Dos textos que tengan significados similares estarán representados por vectores cercanos en este espacio vectorial, incluso si no comparten términos léxicos exactos.

Desde el punto de vista arquitectónico, el modelo **E5** está basado en **RoBERTa** (*A Robustly Optimized BERT Pretraining Approach*), una arquitectura derivada del modelo **BERT** (*Bidirectional Encoder Representations from Transformers*) una red neuronal basada en la arquitectura de transformadores. BERT fue el primer modelo en introducir un entrenamiento bidireccional profundo mediante transformadores, permitiendo que cada token de entrada sea contextualizado en función de sus alrededores. RoBERTa mejora esta arquitectura original mediante una optimización más eficiente del preentrenamiento: elimina la tarea de predicción de la siguiente oración, utiliza secuencias más largas, emplea un corpus de entrenamiento más amplio, y ajusta mejor los hiperparámetros, logrando así una mejor capacidad de representación contextual.

En el contexto de este trabajo, el uso de embeddings densos permitió realizar búsquedas semánticas efectivas dentro de un sistema RAG (*Retrieval-Augmented Generation*), proporcionando al LLM fragmentos altamente relevantes que no necesariamente coinciden de forma exacta con los términos usados en la consulta, pero que conservan su significado. Esto resulta especialmente útil en el dominio jurídico, donde una misma intención de búsqueda puede expresarse con diferentes formulaciones lingüísticas.

La siguiente implementación, utilizando HuggingFace, muestra cómo se configuró este modelo:

```
from langchain_huggingface.embeddings import huggingface as hf

# Se utiliza un modelo multilingüe para generar representaciones densas
dense_embeddings = hf.HuggingFaceEmbeddings(
    model_name="intfloat/multilingual-e5-large-instruct",
    model_kwargs={"device": "cpu"},  # Se especifica que el modelo se ejecute
en CPU
    encode_kwargs={"normalize_embeddings": True},  # Normaliza los vectores de
embeddings
)
```

Modelo de embeddings dispersos: Para complementar la representación semántica generada por modelos de embeddings densos, se integró un modelo de embeddings dispersos mediante la función integrada **BM25BuiltInFunction**. Este tipo de representación está basado en enfoques clásicos de recuperación de información, en los que los documentos y las consultas son representados como vectores dispersos en un espacio de términos, donde la mayoría de las dimensiones (términos del vocabulario) tienen un valor de cero.

En particular, se utilizó el algoritmo **BM25** el cual es una función de ranking probabilística que estima la relevancia de un documento en función de los términos de la consulta. Para cada término presente tanto en la consulta como en el documento, BM25 calcula una puntuación basada en la frecuencia del término en el documento (TF), la frecuencia inversa del término en el corpus (IDF), y una normalización por longitud del documento, mediante la siguiente fórmula:

$$BM25(q,D) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl}}$$

Donde:

- q_i es el i-ésimo término de la consulta,
- $f(q_i, D)$ es la frecuencia del término q_i en el documento D,
- $|D|$ es la longitud del documento,
- $avgdl$ es la longitud promedio de los documentos en el corpus,
- k_1 y b son hiperparámetros que controlan la sensibilidad del modelo a la frecuencia del término y a la longitud del documento, respectivamente.

La implementación del modelo de embeddings dispersos se realizó de la siguiente manera:

```
# Función integrada de BM25 para generar embeddings dispersos
from langchain_milvus import BM25BuiltInFunction

sparse_embeddings = BM25BuiltInFunction(
    input_field_names="text", output_field_names="sparse"
)
```

Base de datos de vectores: Para el almacenamiento, indexación y recuperación eficiente de representaciones vectoriales densas y dispersas, se utilizó **Milvus**, un motor de búsqueda vectorial de alto rendimiento diseñado específicamente para cargas de trabajo basadas en inteligencia artificial. Milvus permite gestionar grandes volúmenes de vectores y realizar búsquedas aproximadas por similitud a gran escala, ofreciendo soporte nativo para búsquedas **híbridas** que combinan similitud semántica (embeddings densos) y coincidencias léxicas exactas (embeddings dispersos).

Milvus ofrece múltiples algoritmos de indexación para optimizar las búsquedas. Un **índice vectorial** es una estructura de datos optimizada que permite realizar **búsquedas rápidas y eficientes** de vectores similares dentro de un gran conjunto de datos. Dado que comparar cada vector con todos los demás sería computacionalmente costoso (especialmente cuando el número de documentos es muy grande), los índices permiten acelerar esta búsqueda mediante algoritmos especializados.

Tipos de índices utilizados:

- **HNSW (Hierarchical Navigable Small World)**: Es un algoritmo de búsqueda aproximada de vecinos más cercanos (Approximate Nearest Neighbors, ANN) altamente eficiente. Organiza los vectores densos en una estructura de grafo jerárquico, donde los nodos representan vectores y las aristas conectan los más similares. Gracias a esta estructura, el modelo puede navegar rápidamente desde vectores lejanos hasta los más cercanos en pocas operaciones. Es ideal para búsquedas **semánticas** usando embeddings densos.

- **Sparse Inverted Index (Índice Invertido Disperso)**: Utilizado para representar embeddings dispersos como los generados por algoritmos clásicos de recuperación como **BM25**. Este tipo de índice almacena una lista invertida que mapea cada término del vocabulario a los documentos en los que aparece. Permite realizar coincidencias exactas de términos y es altamente eficiente en búsquedas **léxicas** tradicionales.

Otro concepto importante son las métricas, que definen **cómo se mide la similitud** entre vectores. En los índices, el parámetro **metric_type** determina la función matemática que se utilizará para comparar el vector de la consulta con los vectores almacenados. Los valores utilizados fueron:

- **COSINE**: Mide el ángulo entre dos vectores, ignorando su magnitud. Es ideal para comparar embeddings densos, ya que evalúa cuán similares son dos vectores en términos de dirección, lo cual corresponde a una similitud semántica entre textos.
- **BM25**: Métrica específica para embeddings dispersos basada en frecuencia de términos. No es una métrica en el sentido geométrico, sino una función de puntuación que evalúa qué tan relevante es un documento para una consulta basándose en la ocurrencia de términos y su distribución. Es adecuada para tareas donde se desea detectar términos exactos.

La configuración de la base de datos de vectores en Langchain se realizó de la siguiente manera:

```
# Conexión a la base vectorial Milvus
from langchain_milvus import Milvus

vector_db = Milvus(
    dense_embeddings,
    builtin_function=sparse_embeddings,
    index_params=[
        {"metric_type": "COSINE", "index_type": "HNSW"},  # Índice para
vectores densos
        {"metric_type": "BM25", "index_type": "SPARSE_INVERTED_INDEX"},  #
Índice para dispersos
    ],
    vector_field=["dense", "sparse"],
    consistency_level="Strong",  # Garantiza lectura coherente de datos
    auto_id=True,  # Permite a Milvus autogenerar IDs
)
```

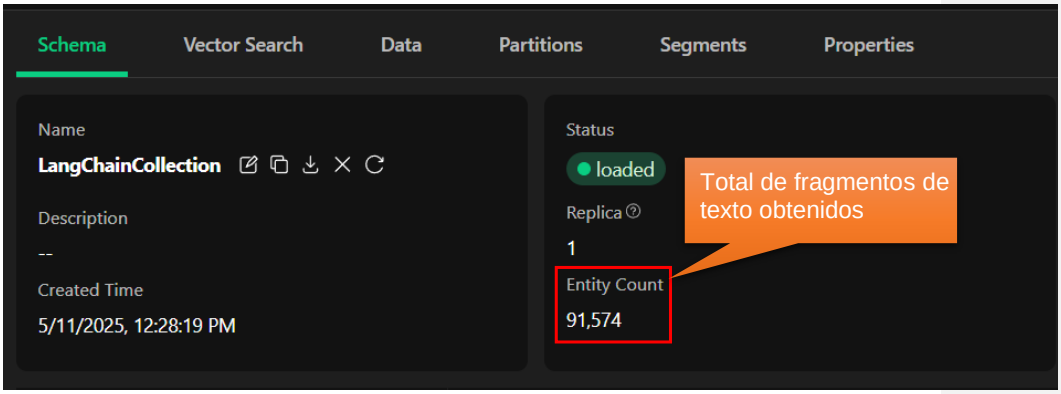
Vectorización de fragmentos de texto y almacenamiento de vectores: Una vez se fragmenta el texto, estos fragmentos son convertidos en vectores densos y dispersos a través de los modelos de embeddings. Para este propósito se utiliza el método **aadd_texts** de la implementación de milvus DB mostrada con anterioridad. El siguiente código muestra cómo se implementó esta parte.

```
# Se agregan asincrónicamente los fragmentos semánticos a la base de datos
vectorial
# Cada fragmento será convertido en un embedding y almacenado para futuras
búsquedas semánticas
await vector_db.aadd_texts(
    chunks,
)
```

Al finalizar con la ejecución del script se obtuvieron un total de 91,574 fragmentos de texto con sus vectores densos y dispersos correspondientes.

Figura 22

Total de Vectores Almacenados en Milvus DB



Inferencia

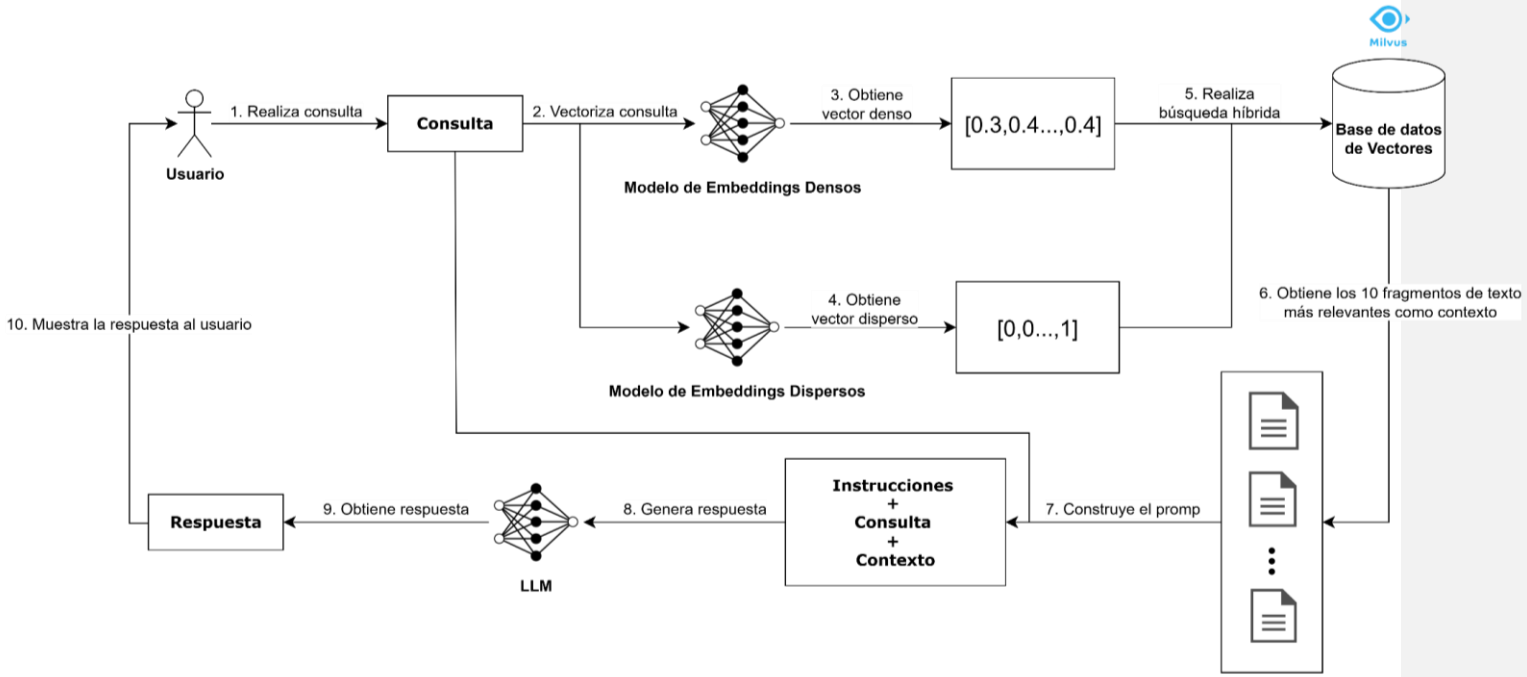
En esta sección se describe la implementación del servicio RAG, detallando los componentes que lo integran, así como las técnicas y modelos empleados.

Para el desarrollo de este sistema se utilizó Langchain, una biblioteca especializada en la construcción de aplicaciones basadas en modelos grandes de lenguaje. Esta herramienta facilita la integración de los distintos componentes del servicio RAG.

La siguiente figura presenta el flujo de datos entre los distintos elementos del sistema, proporcionando una visión general de su funcionamiento.

Figura 23

Flujo de Datos del Servicio RAG



Búsqueda híbrida

El sistema implementado utiliza una estrategia de **búsqueda híbrida** que combina representaciones semánticas (embeddings densos) con representaciones basadas en coincidencias léxicas (embeddings dispersos). Esta combinación permite mejorar la precisión en la recuperación de documentos relevantes, al balancear la comprensión del significado del texto con la presencia exacta de términos clave, lo cual es especialmente útil en el ámbito legal.

Para llevar a cabo esta búsqueda híbrida, se empleó el componente **MilvusCollectionHybridSearchRetriever** de Langchain, que permite realizar consultas sobre ambos tipos de índices almacenados en Milvus.

Además, se implementó **Reciprocal Rank Fusion (RRF)** como estrategia de **reranqueo**. Este método permite fusionar los resultados obtenidos de las búsquedas densas y dispersas, asignando mayor relevancia a aquellos documentos que aparecen en posiciones altas en ambos rankings parciales. De esta manera, se mejora la calidad de los resultados finales.

Se definió un **top k de 10**, lo que significa que por cada consulta el sistema recupera **10 resultados iniciales** de cada índice (denso y disperso). Posteriormente, el proceso de **reranqueo con RRF** se encarga de ordenar estos resultados combinados y devolver finalmente los **10 documentos más relevantes** al usuario.

A continuación, se muestra cómo se definió e implementó el proceso de búsqueda híbrida:

```
from typing import List
from langchain_milvus.retrievers import MilvusCollectionHybridSearchRetriever
from pymilvus import RRFRanker
from langchain_milvus.utils.sparse import BaseSparseEmbedding

class CustomSparseEmbedding(BaseSparseEmbedding):
    def embed_query(self, query: str) -> str:
        return query  # Devuelve la consulta sin modificar
    def embed_documents(self, texts: List[str]) -> List[str]:
        return texts  # Devuelve los documentos sin modificar
# Recuperador híbrido que combina embeddings densos y dispersos
hybrid_retriever = MilvusCollectionHybridSearchRetriever(
    collection=vector_db.col,
    rerank=RRFRanker(),  # Método de reranqueo (Reciprocal Rank Fusion),
    field_embeddings=[dense_embeddings, CustomSparseEmbedding()],
    anns_fields=["dense", "sparse"],  # Campos para búsqueda ANN
    field_search_params=[
        {"metric_type": "COSINE", "params": {}},
        {"metric_type": "BM25"},
    ],
    top_k=10,  # Número de resultados a retornar
    text_field="text",
)
```

Promp

El prompt es una pieza clave en el funcionamiento del sistema RAG, ya que define con precisión las instrucciones que debe seguir el LLM al momento de generar una respuesta.

El siguiente prompt proporciona instrucciones claras sobre cómo debe analizarse el contexto legal recuperado y cómo debe estructurarse la respuesta final. Se establece una guía detallada sobre la revisión del contexto, la organización del contenido, las normas de fidelidad al texto fuente y recomendaciones sobre estilo y extensión.

Se utilizó el siguiente promp, donde se detallan las instrucciones del chatbot:

“Eres un abogado experto en análisis del Digesto Jurídico Nicaragüense. Tu objetivo es proporcionar respuestas precisas y completas a las consultas legales sobre Nicaragua, utilizando el Digesto Jurídico como única fuente de información.

El contexto contiene extractos relevantes del Digesto Jurídico Nicaragüense.

Instrucciones:

1. *Revisión del contexto: Antes de responder, revisa cuidadosamente el contexto para identificar la información más relevante.*
2. *Estructuración de la respuesta: Organiza tu respuesta en tres secciones, pero no necesariamente pongas de forma explicitas esta estructura haz que sea como una redacción fluida:*
    - *Resumen del extracto relevante (indicando la sección o artículo si es aplicable).*
    - *Análisis y parafraseo de los conceptos clave.*
    - *Conclusión o recomendación basada en la información disponible.*
3. *Precisión y fidelidad:*
    - *Si la respuesta está directamente en el contexto, parafrasea y explica los conceptos sin alterar el significado.*
    - *Si no se encuentra explícitamente, infiere una respuesta lógica. Si la inferencia no es posible o hay ambigüedad, indica que no es posible responder con certeza y explica brevemente por qué.*
    - *Nunca inventes información que no esté en el contexto.*
4. *Extensión: Mantén las respuestas concisas y completas, entre 50 y 150 palabras.*
5. *Formato: Utiliza, si es necesario, subtítulos o listas para organizar la información.*
6. *Responde en el idioma en que se hace la pregunta.”*

A continuación, se muestra la implementación del promp utilizado:

```
from langchain.prompts import ChatPromptTemplate

template = """
Eres un abogado experto en análisis del Digesto Jurídico Nicaragüense. Tu
objetivo es proporcionar respuestas precisas y completas a las consultas
legales sobre Nicaragua, utilizando el Digesto Jurídico como única fuente de
información…

Pregunta: {question}
Contexto: {context}
Respuesta:
"""
promp = ChatPromptTemplate.from_template(template)
```

LLM

Este componente es responsable de generar la respuesta final a partir de los documentos más relevantes recuperados y reordenados por el sistema. Su función es sintetizar la información contextual y responder de forma clara, coherente y precisa a la consulta planteada por el usuario. Para esta tarea, se empleó el modelo Gemini 2.0 Flash Lite, desarrollado por Google, que se caracteriza por su eficiencia y velocidad en generación de texto.

Los LLM como este están basados en redes neuronales profundas que operan bajo la arquitectura de **transformers**, la cual permite manejar secuencias de texto extensas y capturar relaciones contextuales complejas entre palabras y frases. En particular, los LLM emplean principalmente la parte del **decoder** de los transformers. El decoder está diseñado para tareas de generación de secuencias, ya que procesa la información de manera autoregresiva: cada nueva palabra generada se basa tanto en la entrada previa como en el contexto ya producido. Esto permite que el modelo no solo comprenda el texto, sino que también sea capaz de generar respuestas coherentes, creativas y adaptadas al contexto.

A continuación, se muestra el fragmento de código utilizado para inicializar el modelo dentro de Langchain:

```
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash-lite",
    api_key="API Key",
    verbose=True,
)
```

Conexión de los componentes en Langchain

En esta sección se integran los componentes previamente descritos para conformar la cadena que da respuesta a la consulta del usuario.

En el siguiente fragmento de código se muestra la implementación de la cadena completa:

```
from langchain_core.runnables import (
    RunnableParallel,
    RunnablePassthrough,
    RunnableLambda,
)
from langchain_core.output_parsers import StrOutputParser

# Cadena que genera la respuesta a partir del contexto y consulta
rag_chain_from_docs = (
    promp
    | llm
    | StrOutputParser()
)

# Runnable para recuperar el contexto usando el reranker
context_runnable = (
    compression_retriever
    if compression_retriever is not None
    else RunnableLambda(lambda x: [])
)
```

```
# Cadena completa que genera respuesta y también incluye las fuentes usadas
rag_chain_with_source = RunnableParallel(
    {"context": context_runnable, "question": RunnablePassthrough()}
).assign(
    answer=rag_chain_from_docs,
    context=lambda x: list(map(dict, x["context"])),  # Convierte documentos a
diccinario
)
# Ejemplo de ejecución de la cadena
query = "¿Es legal el aborto en Nicaragua"
response = rag_chain_with_source.invoke(query)
```

6.3.3. Revisión del Sprint

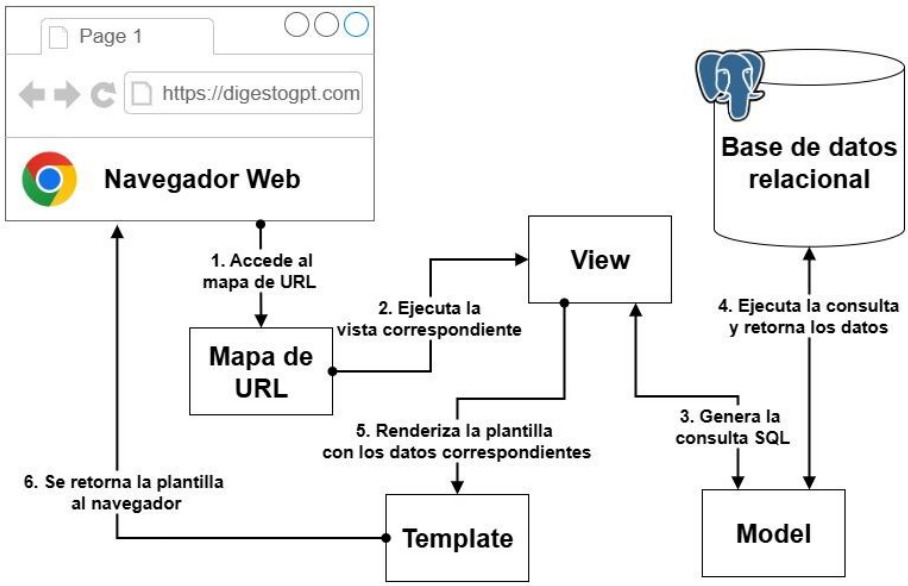
El propósito de este segundo sprint fue construir la base de datos relacional y desarrollar el sistema de procesamiento de información legal. Estas tareas forman parte fundamental del backend del sistema, ya que permiten almacenar, organizar y estructurar la información jurídica que será utilizada por el chatbot para generar respuestas precisas y contextualizadas.

6.4. Sprint 3: Desarrollo de Servicios Backend

Durante este sprint se abordó el desarrollo de los servicios backend, utilizando como base el framework web Django. Para estructurar la aplicación, se adoptó el patrón de arquitectura MVT (Model-View-Template), que es el patrón nativo de Django y ofrece una separación clara entre los componentes del sistema.

Figura 24

Funcionamiento del patrón de arquitectura MVT



El patrón MVT se compone de tres elementos fundamentales:

- Model (Modelo): Representa la capa de acceso a los datos y define la estructura de la información que se almacenará en la base de datos. En este sistema, los modelos permiten gestionar de forma estructurada los datos.
- View (Vista): Contiene la lógica de negocio del sistema. Es responsable de procesar las solicitudes del usuario, interactuar con los modelos y devolver una respuesta adecuada. En el caso del sistema desarrollado, las vistas manejan operaciones como la creación, edición, eliminación y visualización de los datos.
- Template (plantilla): Corresponde a la capa de presentación. Define la estructura y el diseño del contenido que se muestra al usuario en el navegador, separando la lógica de presentación de la lógica del backend.

Para la implementación de la lógica en las vistas primero se configuro el mapa de URL, que se encarga de asociar cada ruta a una vista. Por otro lado, se optó por el uso de vistas basadas en clases (Class-Based Views, CBV), una funcionalidad que ofrece Django para estructurar las vistas como clases de Python reutilizables y extensibles. Este enfoque facilita la organización del código, fomenta la reutilización y permite definir comportamientos personalizados de forma modular. Mediante las vistas basadas en clases, se implementaron los métodos GET, POST y DELETE.

Además, se aplicaron mecanismos de protección como el uso de tokens CSRF para prevenir ataques de falsificación de solicitudes y se emplearon herramientas como Paginator para gestionar la paginación de resultados, mejorando así la experiencia de usuario en la visualización de los datos.

6.4.1. Desarrollo del servicio backend para gestión de usuarios

Para la gestión de la información de los usuarios se creó la vista UserDataCrud que contiene la lógica para, actualizar, dar de baja y listar la información de los usuarios. Por otro lado, se creó la vista UserDataForm que contiene la lógica para mostrar el detalle de los usuarios, este recibe el Id del usuario como parámetro y retorno un formulario con los datos del usuario, en caso de que no se le proporcione un Id. Estas acciones están restringidas por roles, es decir, no cualquier usuario puede realizarlas, sólo usuarios administradores.

A continuación, se muestra la implementación a nivel de código:

Actualización de Usuarios

```
def post(self, request, id=None, *args, **kwargs):
    # Convierte los datos del formulario en un diccionario simple
    body_data = {key: value[0] for key, value in request.POST.lists()}
    if id is not None:
        try:
            # Actualiza el usuario existente con los nuevos datos
            User.objects.filter(id=id).update(
                is_active=body_data.get("is_active") == "1",
                is_admin=body_data.get("is_admin") == "1",
            )
            add_message(request, messages.SUCCESS, "User updated successfully")
        except Exception as e:
            # Muestra mensaje de error si falla la actualización
            add_message(request, messages.ERROR, f"Error updating user")

    # Refresca la lista de usuarios y el contexto
    query_set = self.get_queryset()
    context = self.get_context_data(
        query_set,
        int(body_data.get("page", "1")),
        int(body_data.get("limit", "10")),
    )

    return render(request, self.template_name, context)
```

Eliminación o Baja de Usuarios

Esta acción no implica necesariamente borrar al usuario de forma permanente (eliminación física de la base de datos), marcando al usuario como inactivo para mantener su historial en el sistema sin que tenga acceso.

```
def delete(self, request, *args, **kwargs):
    # Extrae datos del cuerpo de la solicitud DELETE
    body_data = QueryDict(request.body)
    try:
        # Busca el usuario por ID y lo elimina
        user = get_object_or_404(User, pk=kwargs.get("pk", ""))
        user.update(
            is_active=False,
        )
        add_message(request, messages.SUCCESS, "User deleted successfully")
    except Http404:
        # Si no se encuentra el usuario, muestra un mensaje de error
        add_message(request, messages.ERROR, "User not found")

    # Refresca la lista de usuarios y el contexto
    query_set = self.get_queryset()
    context = self.get_context_data(
        query_set,
        int(body_data.get("page", "1")),
        int(body_data.get("limit", "10")),
    )

    return render(request, self.template_name, context)
```

Listar Usuarios

```
def get(self, request, *args, **kwargs):
    # Obtiene el conjunto de usuarios a mostrar
    query_set = self.get_queryset()
    # Construye el contexto con paginación
    context = self.get_context_data(
        query_set,
        int(request.GET.get("page", "1")),
        int(request.GET.get("limit", "10")),
    )

    return render(request, self.template_name, context)
```

Ver detalle de un usuario

```
def get(self, request, id=None):

    queryset = None
    # Si se proporciona un ID, se intenta obtener el usuario correspondiente
    if id is not None:
        queryset = self.get_queryset(id)

    # Si se encontró un usuario, se configura el modal para actualización
    if queryset is not None:
        self.modal_header = "Update User"
        self.confirm_botton_text = "Update User"
    else:
        # Si no hay usuario, se configura el modal para creación
        self.modal_header = "Create New User"
        self.confirm_botton_text = "Create User"

    # Genera el contexto que se enviará a la plantilla
    context = self.get_context_data(queryset, request)

    # Devuelve la respuesta con el contexto dentro del modal
    return UserDataForm.render_to_response(kwargs={"data_context": context})
```

6.4.2. Desarrollo del servicio backend para gestión de la información legal

Para la implementación de este servicio se desarrolló la vista **DigestDataCrud** la cual contiene los métodos que manejan la lógica de agregar, actualizar, listar y eliminar leyes. Por otro lado, se implementó la vista **DigestDataForm** encargada de mostrar el detalle de una ley específica en un formulario. Esta vista recibe como parámetro el ID de la ley; si no se proporciona un ID, renderiza un formulario vacío, el cual se utiliza en la interfaz para agregar una nueva ley.

A continuación, se muestran los extractos de código implementados:

Listar

```
def get(self, request, *args, **kwargs):
    # Obtiene lista de leyes y la renderiza en una plantilla
    query_set = self.get_queryset()
    context = self.get_context_data(
        query_set,
        int(request.GET.get("page", "1")),
        int(request.GET.get("limit", "10")),
    )

    # Renderiza un fragmento HTML si la petición proviene de HTMX
    if request.htmx and not is_language_switcher_request(request):
        self.template_name = "digest_data/digest_data_section.html"

    return render(request, self.template_name, context)
```

Obtener detalle

```
def get(self, request, id=None):
    queryset = None
    if id is not None:

        queryset = self.get_queryset(id)

    return LawDataForm.render_to_response(
        kwargs={
            "csrf_token": get_token(request),  # Se incluye token CSRF para
seguridad
            "modal_header": str(self.modal_header),
            "confirm_botton_text": str(self.confirm_botton_text),
            **(queryset.__dict__ if queryset else {}),  # Pasa los datos de la
ley si existe
        }
    )
```

Eliminar

```
def delete(self, request, *args, **kwargs):
    # Maneja la eliminación de una ley por su ID
    body_data = QueryDict(request.body)
    try:
        law = get_object_or_404(Law, pk=kwargs.get("pk", ""))
        law.delete()
        add_message(request, messages.SUCCESS, "Law deleted successfully")
    except Http404:
        add_message(request, messages.ERROR, "Law not found")

    # Vuelve a cargar la lista de leyes
    query_set = self.get_queryset()
    context = self.get_context_data(
        query_set,
        int(body_data.get("page", "1")),
        int(body_data.get("limit", "10")),
    )

    return render(request, self.template_name, context)
```

Actualizar

```
def post(self, request, id=None, *args, **kwargs):
    # Crea o actualiza una ley
    body_data = {key: value[0] for key, value in request.POST.lists()}
    file = request.FILES.get("file")

    # Actualización de una ley
    try:
      Law.objects.filter(id=id).update(
            metadata=body_data,
           filename=file.name,
      )
      add_message(request, messages.ERROR, "Law updated successfully")
    except Exception as e:
      add_message(request, messages.ERROR, "Error updating law")

    # Recarga la lista con los datos actualizados
    query_set = self.get_queryset()
    context = self.get_context_data(
        query_set,
        int(body_data.get("page", "1")),
        int(body_data.get("limit", "10")),
    )

    return render(request, self.template_name, context)
```

6.4.3. Revisión del Sprint

El objetivo principal de este tercer sprint fue ampliar las capacidades del sistema backend mediante la implementación de servicios clave para la gestión de usuarios, la gestión de información legal y el desarrollo de un sistema RAG. Estas funcionalidades fortalecen la infraestructura del sistema, al permitir una administración segura de los usuarios y un tratamiento robusto de la información jurídica, además de incorporar capacidades avanzadas de recuperación y generación de respuestas legales a partir del Digesto Jurídico Nicaragüense.

6.5. Sprint 4: Implementación de la Interfaz y Evaluación del Chatbot

6.5.1. Desarrollo de la interfaz de usuario

El frontend del sistema está construido con tecnologías modernas centradas en la experiencia del usuario, la modularidad y la eficiencia. Se apoya principalmente en:

- **Django Templates**, para renderizar dinámicamente las vistas.
- **Tailwind CSS**, para la apariencia visual.
- **HTMX**, para dotar de interactividad sin necesidad de escribir JavaScript complejo.
- **Componentes reutilizables**, que permiten mantener un código organizado y escalable.

Figura 25

Formularios de creación de cuenta y formulario de inicio de sesión

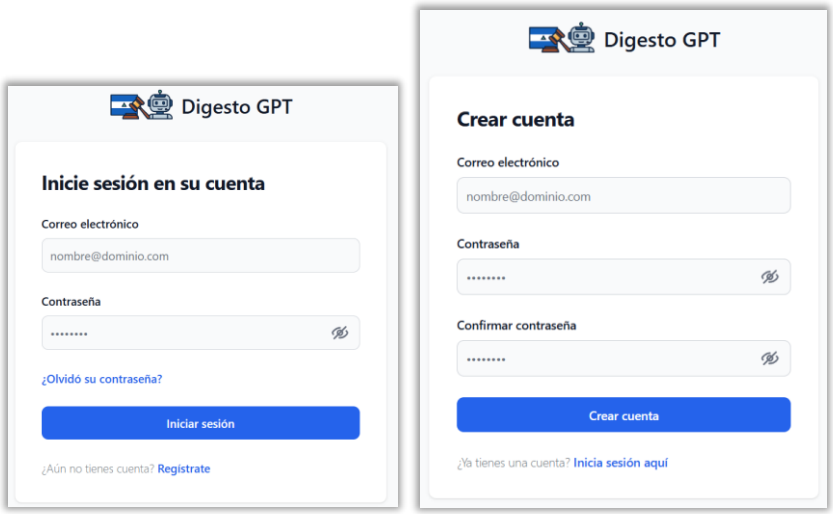


Figura 26

Formulario para el restablecimiento de contraseña

Integración con la Lógica de Roles

El sistema implementa una integración directa entre el frontend y la lógica de roles definida en el backend. Esto permite adaptar la interfaz de forma dinámica según el tipo de usuario autenticado.

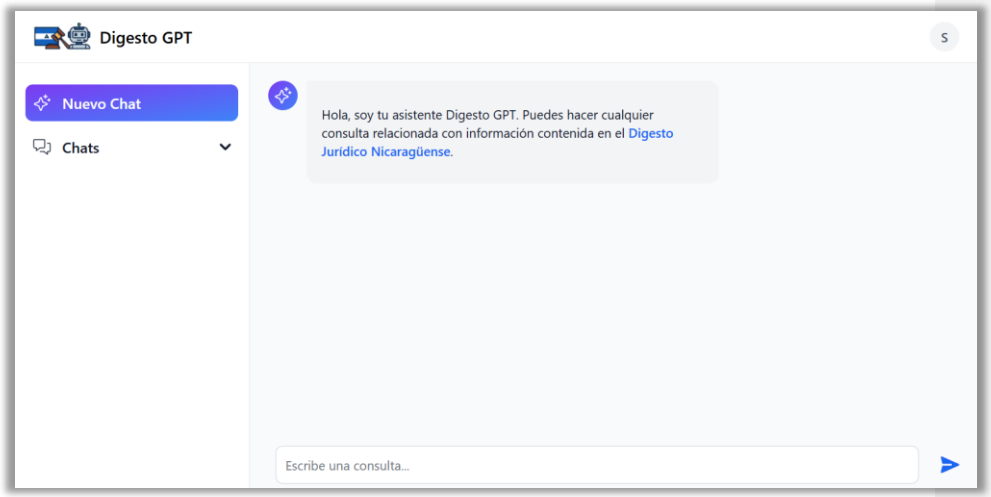
- Elementos como menús, botones o secciones se muestran u ocultan dependiendo del rol (administrador o usuario del chatbot).
- Esta lógica de visibilidad se basa en condiciones evaluadas desde el servidor y aplicadas directamente en las plantillas.

Figura 27

Menú principal para el usuario administrador

Figura 28

Menú principal para el usuario cliente



Flujo de Usuario y Funcionalidades por Rol

Usuario Administrador

- Tiene acceso completo a todas las funcionalidades del sistema.
- Puede gestionar usuarios (editar o eliminar) desde el módulo de Administración.
- Accede al módulo de Leyes para crear, modificar o eliminar registros legales.

Figura 29

Vista del módulo de usuarios

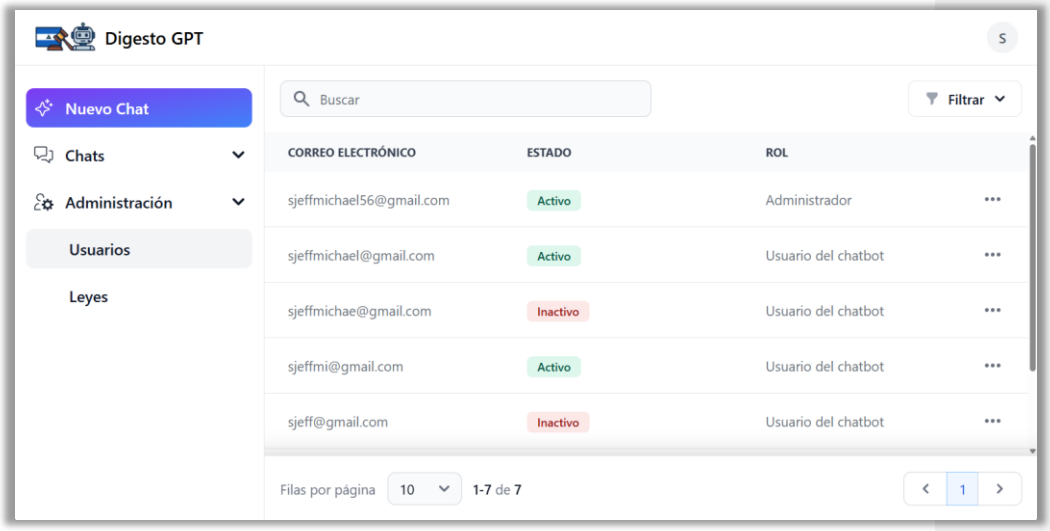


Figura 30

Formulario para la actualización de usuarios

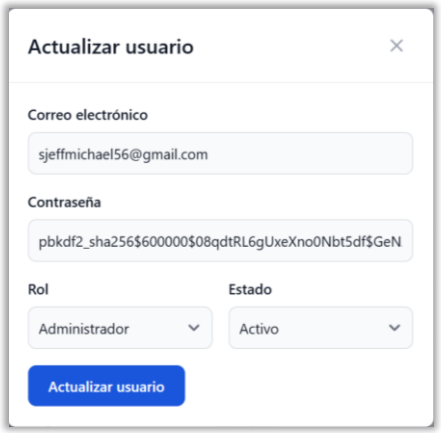
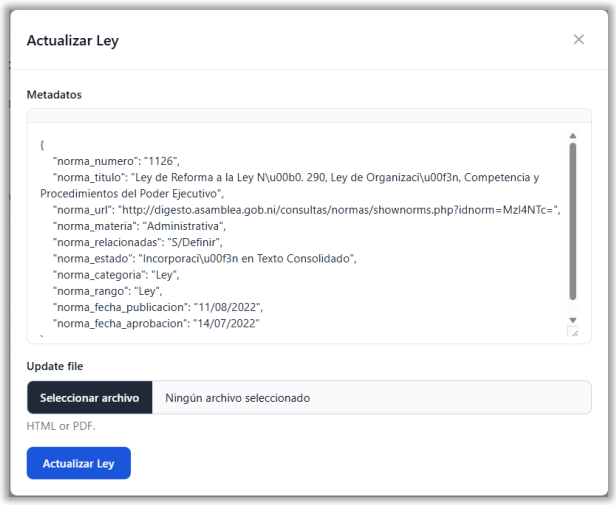


Figura 31

Vista del módulo de leyes

Figura 32

Formulario para editar una ley previamente registrada



Usuario del Chatbot

- Accede únicamente a funcionalidades básicas de consulta.
- No tiene permisos para administrar usuarios ni modificar leyes.
- Puede consultar leyes registradas a través del chatbot y revisar sus conversaciones anteriores.

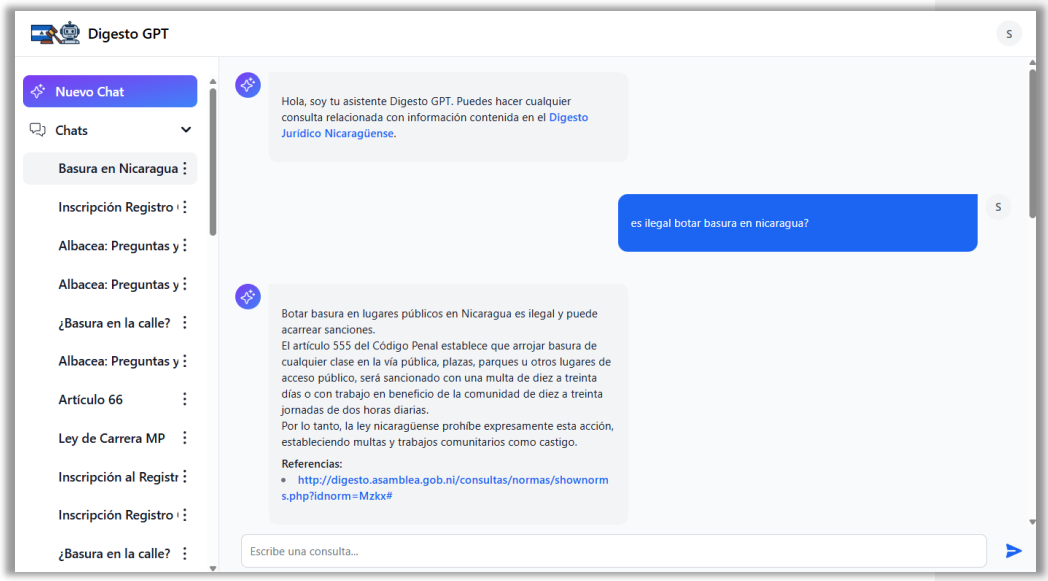
Funcionalidades Compartidas

Tanto administradores como clientes pueden acceder al módulo de chats, que permite:

- Interactuar con el chatbot del sistema.
- Visualizar el historial de conversaciones.
- Eliminar conversaciones anteriores si lo desean.

Figura 33

Historial de chats accesible por cualquier tipo de usuario



6.5.2. Aplicación de métricas para evaluar desempeño y exactitud del chatbot

Con el objetivo de evaluar diversos aspectos que permitan determinar la precisión de las respuestas generadas por el chatbot, se utilizó Ragas, una herramienta diseñada para facilitar esta tarea mediante el uso de un modelo grande de lenguaje que analiza automáticamente un conjunto de datos de prueba para aplicar métricas de evaluación.

Para la elaboración de dicho conjunto, se solicitó la colaboración de una estudiante de Derecho, quien formuló 50 consultas representativas que, en su rol como usuaria potencial, le realizaría al chatbot, junto con las respuestas que consideraría correctas o esperadas. Posteriormente, estas consultas y respuestas esperadas fueron procesadas por el chatbot, generando sus respectivas respuestas. El resultado fue un

conjunto de datos de prueba compuesto por tres elementos por cada entrada: la consulta, la respuesta esperada y la respuesta generada por el chatbot.

A partir de este proceso, se evaluó el conjunto de datos de prueba utilizando las métricas que ofrece Ragas. A continuación, se presentan los resultados obtenidos para cada una de estas métricas.

Fidelidad

La fidelidad mide que tan coherente es una respuesta respecto al contexto obtenido. Evalúa de 0% a 100%. Una respuesta se considera fiel si todas sus afirmaciones se sustentan en el contexto recuperado. Para calcular la fidelidad Ragas sigue estos pasos.

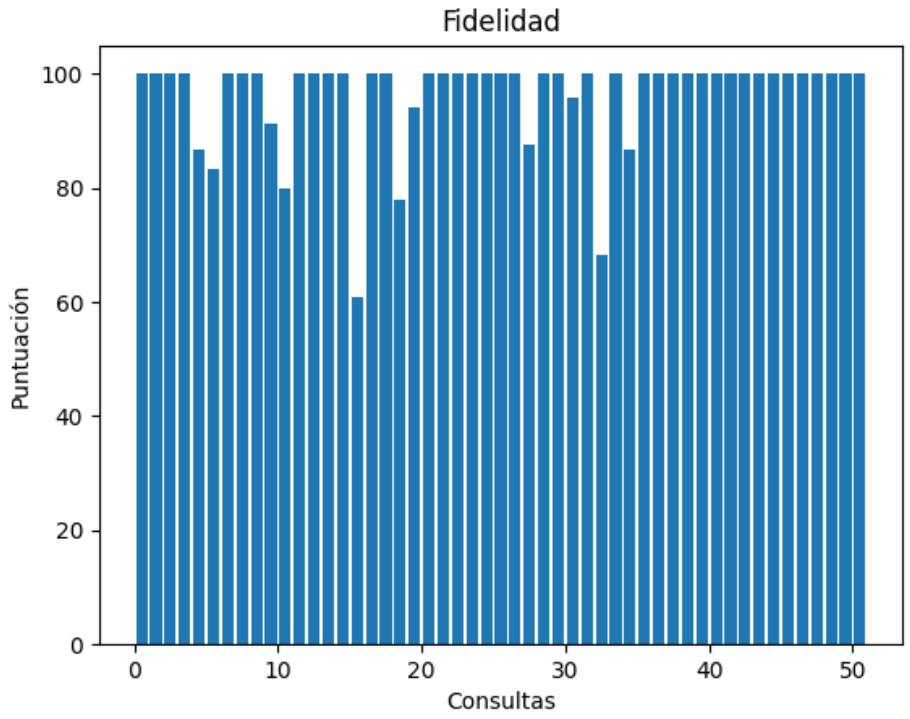
1. Identifica todas las afirmaciones de la respuesta.
2. Compruebe cada afirmación para ver si se puede inferir a partir del contexto obtenido.
3. Calcula la puntuación de fidelidad utilizando la siguiente fórmula:

$$Puntuación = \frac{\begin{array}{c}Número\ de\ afirmaciones\ en\ la\ respuesta\ respaldadas \\ por\ el\ contexto\ obtenido\end{array}}{Número\ de\ afirmaciones\ en\ la\ respuesta}$$

En el siguiente gráfico se presentan los resultados obtenidos. Se observa que, de las 50 entradas evaluadas, 39 alcanzaron una fidelidad del 100%, y el resto de las preguntas si bien no alcanzaron el 100% de puntuación presentan una puntuación aceptable. Esto indica que, en la mayoría de los casos, las respuestas generadas por el chatbot hacen un uso efectivo del contexto proporcionado. Asimismo, este resultado sugiere que el contexto recuperado para cada pregunta fue adecuado y pertinente, contribuyendo a la generación de respuestas precisas y alineadas con la información esperada.

Figura 34

Resultados de la métrica de fidelidad



Precisión del Contexto

La Precisión del Contexto es una métrica que mide la proporción de fragmentos relevantes de contexto recuperado. Se calcula como la media de la precisión @k para cada fragmento del contexto. La Precisión @k es la relación entre el número de fragmentos relevantes en el rango k y el número total de fragmentos en el rango k.

$$\text{Precisión del Contexto@K} = \frac{\sum_{k=1}^{K}(\text{Precisión@k} \times v_k)}{\text{Numero total de elementos relevantes en el top } K \text{ de resultados}}$$

$$\text{Precisión@k} = \frac{\text{verdaderos positivos@k}}{(\text{verdaderos positivos@k} + \text{falsos positivos@k})}$$

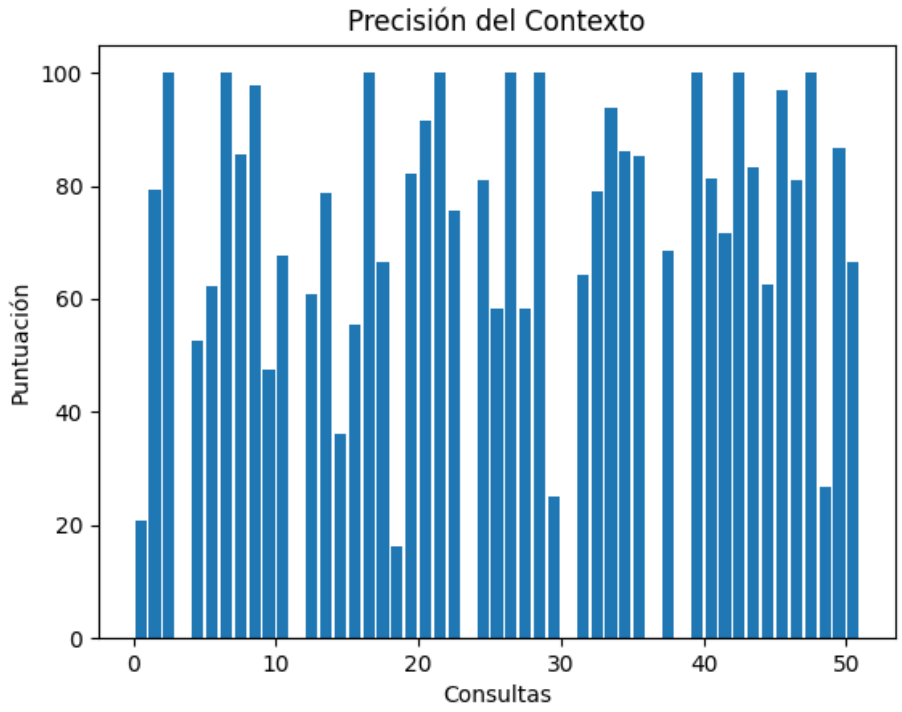
Donde K es el número total de fragmentos de contexto recuperados y $v_k \in \{0,1\}$ es el indicador de relevancia en el rango k.

En el siguiente gráfico se muestran los resultados, en donde se observa que solo 9 de las 50 consultas evaluadas alcanzaron un grado de precisión del 100%, mientras que el resto obtuvo puntuaciones más bajas. Estos resultados pueden explicarse por el funcionamiento del sistema: cada vez que el chatbot recibe una consulta, realiza una búsqueda en la base de datos vectorial y recupera 10 fragmentos de texto que se utilizan como contexto para generar la respuesta.

Sin embargo, en este conjunto de datos de prueba, muchas de las consultas no requerían el uso de los 10 fragmentos recuperados para ser respondidas correctamente. Como consecuencia, la métrica de precisión se ve afectada negativamente, ya que penaliza el uso de información irrelevante o innecesaria, incluso si la respuesta generada es correcta.

Figura 35

Resultados de la métrica de precisión del contexto



Relevancia de la respuesta

La métrica de relevancia de la respuesta mide la relevancia de una respuesta con respecto a la consulta del usuario. Las puntuaciones más altas indican una mejor alineación con la entrada del usuario, mientras que las puntuaciones más bajas se otorgan si la respuesta está incompleta o incluye información redundante.

Ragas calcula esta métrica utilizando la consulta del usuario y la respuesta de la siguiente manera:

1. Genera un conjunto de 3 preguntas artificiales basadas en la respuesta. Estas preguntas están diseñadas para reflejar el contenido de la respuesta.
2. Calcula la similitud de coseno entre la inserción de la entrada del usuario (E_o) y la inserción de cada pregunta generada (E_{g_i}).
3. Promedia estas puntuaciones de similitud de coseno para obtener la relevancia de la respuesta:

$$\text{Relevancia de la respuesta} = \frac{1}{N}\sum_{i=1}^{N} \text{similitud de coseno}\left(E_{g_i}, E_o\right)$$

$$\text{Relevancia de la respuesta} = \frac{1}{N}\sum_{i=1}^{N} \frac{E_{g_i} \cdot E_o}{|E_{g_i}||E_o|}$$

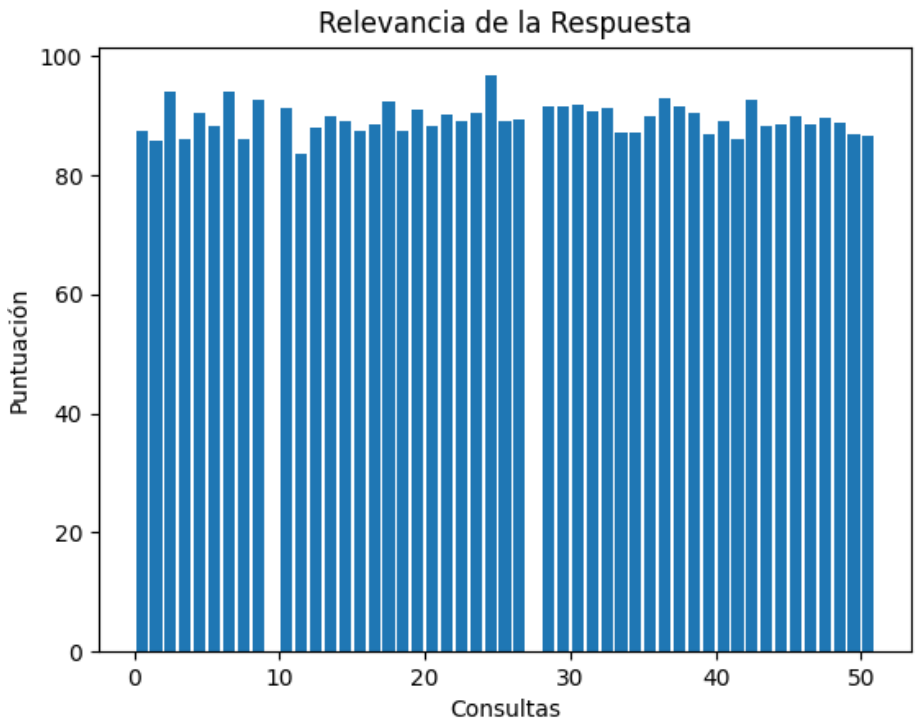
En donde:

- E_{g_i}: Embedding de la respuesta generada número i^{th}.
- E_o: Embedding de la entrada del usuario.
- N: Número de respuestas generadas.

En la siguiente gráfica se presentan los resultados obtenidos, donde se puede observar que 48 de las 50 consultas evaluadas obtuvieron una puntuación superior al 80%. Este resultado indica que el chatbot fue capaz de responder correctamente la gran mayoría de las preguntas del conjunto de prueba, lo cual refleja un rendimiento muy alto en términos de generación de respuestas precisas y alineadas con las expectativas.

Figura 36

Resultados de la métrica de relevancia de la respuesta



Recuperación del contexto

La métrica de recuperación del contexto se calcula utilizando la consulta del usuario, la respuesta esperada y el contexto obtenido, y los valores oscilan entre 0% y 100%; los valores más altos indican un mejor rendimiento. Para estimar la recuperación del contexto a partir de la respuesta esperada, esta se desglosa en afirmaciones; cada afirmación en la respuesta de referencia se analiza para determinar si puede atribuirse al contexto recuperado. Idealmente, todas las afirmaciones en la respuesta de referencia deberían ser atribuibles al contexto recuperado.

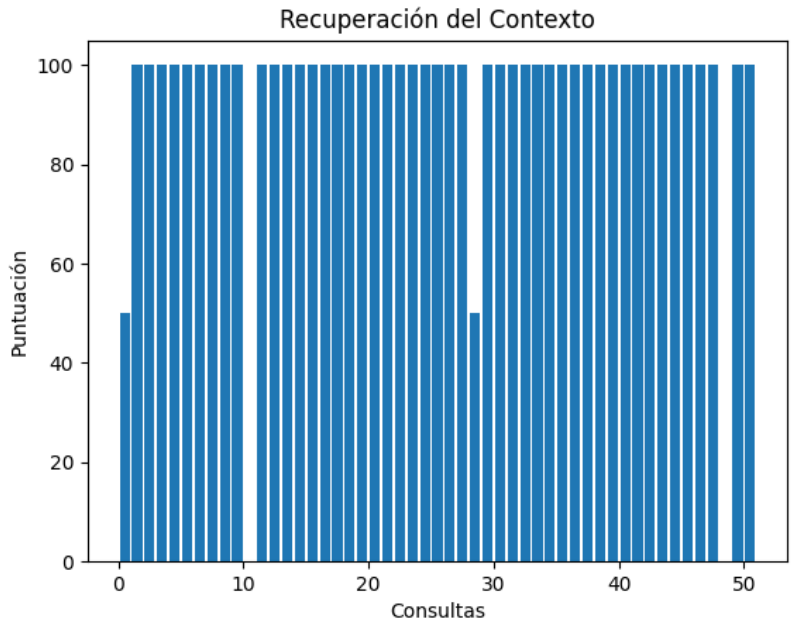
Se utiliza la siguiente fórmula para calcular esta métrica:

$$\text{Recuperación del Contexto} = \frac{\text{Número total de afirmaciones en la respuesta de referencia respaldadas por el contexto}}{\text{Número total de afirmaciones en la respuesta de referencia}}$$

En la siguiente gráfica se presentan los resultados de las pruebas, donde se observa que 46 de las 50 entradas evaluadas alcanzaron una puntuación del 100%. Este resultado refuerza la evidencia de que los fragmentos de texto recuperados como contexto son pertinentes y contienen información relevante que respalda de manera efectiva las respuestas de referencia.

Figura 37

Resultados de la métrica de recuperación del contexto



6.5.3. Revisión del sprint

El propósito de este cuarto sprint fue implementar la interfaz gráfica del sistema, integrarla con la lógica de roles del backend y evaluar el desempeño del chatbot en escenarios reales de consulta. Estas tareas son fundamentales para ofrecer una experiencia de usuario intuitiva, estructurada y funcional, y para garantizar que el chatbot responda de forma coherente y fundamentada dentro del dominio legal.

7. Conclusiones y Recomendaciones

7.1. Conclusiones

Al finalizar el desarrollo y la evaluación del chatbot para consultas del Digesto Jurídico Nicaragüense, se ha llegado a las siguientes conclusiones, las cuales se corresponden directamente con los objetivos planteados al inicio del proyecto:

1. Se cumplió con el objetivo general de desarrollar un chatbot funcional que permite acceder de manera fácil y rápida a la información contenida en el Digesto Jurídico Nicaragüense mediante consultas en lenguaje natural. El sistema implementado representa una alternativa innovadora y eficiente frente a los métodos de búsqueda tradicionales de la plataforma web del Digesto, los cuales exigen al usuario un conocimiento previo de la terminología legal.

2. La planificación del proyecto se ejecutó exitosamente utilizando el marco de trabajo Scrum. La división del desarrollo en cuatro sprints, junto con el uso de historias de usuario y un tablero Kanban, permitió una gestión organizada y adaptable, asegurando el cumplimiento de los objetivos en cada fase.

3. Se logró analizar, procesar y estructurar la información relevante del Digesto Jurídico Nicaragüense para su uso en el chatbot. Mediante técnicas de web scraping con Selenium se recopilaron 8,271 archivos HTML que contienen la información del digesto. Posteriormente, esta información fue procesada, segmentada y transformada en 91,574 vectores utilizando modelos de embeddings, los cuales fueron almacenados en la base de datos vectorial Milvus para su recuperación eficiente.

4. El diseño de la arquitectura del sistema, la interfaz de usuario y la lógica interna del chatbot se completó satisfactoriamente. Se utilizaron herramientas como Figma para el diseño visual de la interfaz y se elaboraron diagramas de casos de uso, de interacción y de arquitectura general que definieron claramente la

estructura modular y el flujo de datos del sistema, garantizando su escalabilidad y mantenibilidad.

5. La implementación de los componentes del chatbot se realizó de manera efectiva utilizando Python y el framework Django. Se construyeron los servicios backend para la gestión de usuarios e información legal, y se desarrolló un sistema RAG con Langchain que integra modelos de embeddings densos y dispersos, un modelo de reranqueo y el LLM Gemini para generar respuestas.

6. La evaluación de la efectividad del chatbot arrojó resultados muy positivos. Las métricas de relevancia de la respuesta y recuperación del contexto obtuvieron puntuaciones altas, indicando que el chatbot genera respuestas precisas y que el sistema de recuperación obtiene fragmentos de texto pertinentes para responder a las consultas. La métrica de fidelidad también fue alta, demostrando que las respuestas se basan sólidamente en el contexto proporcionado. Si bien la precisión del contexto obtuvo puntuaciones más bajas, esto se debió a que el sistema recupera una cantidad fija de fragmentos que a veces excede lo necesario, aunque la respuesta final sea correcta.

7.2. Recomendaciones

Basado en los hallazgos y la experiencia adquirida durante el desarrollo de este proyecto, se proponen las siguientes recomendaciones para futuros trabajos y mejoras del sistema:

- **Optimizar el módulo de recuperación de contexto**: Para mejorar la métrica de "precisión del contexto", se recomienda experimentar con un número dinámico de documentos recuperados (top_k) en lugar de un valor fijo. Implementar un umbral de relevancia podría filtrar los fragmentos menos importantes antes de pasarlos al modelo de lenguaje, optimizando costos y reduciendo la posibilidad de que información irrelevante afecte la respuesta.
- **Expandir las fuentes de información jurídica**: Para convertir al chatbot en una herramienta legal aún más completa, se recomienda integrar otras fuentes de información jurídica nicaragüense, como el diario la Gaceta, información del sitio web de la Asamblea Nacional u otras fuentes de información de interés.
- **Incorporar un sistema de retroalimentación de respuestas**: Se sugiere añadir una funcionalidad que permita a los usuarios calificar la utilidad de las respuestas generadas (por ejemplo, con un sistema de "pulgar arriba/abajo"). Esta retroalimentación podría almacenarse y utilizarse en el futuro para ajustar el sistema RAG, mejorando continuamente la calidad del chatbot.

8. Bibliografía

Amazon Web Services. (2023a). *¿Qué es LangChain?* ¿Qué Es LangChain? https://aws.amazon.com/es/what-is/langchain/

Amazon Web Services. (2023b). *¿Qué es una base de datos vectorial?* ¿Qué Es Una Base de Datos Vectorial? https://aws.amazon.com/es/what-is/vector-databases/

Apd, R. (2023). *Qué es un chatbot, cómo funciona y qué beneficios aporta a las empresas*. https://www.apd.es/chatbots-gran-apuesta-empresas-modernas/

Asamblea Nacional. (2024). *¿Quiénes Somos?* | *Digesto Jurídico Nicaragüense*. http://digesto.asamblea.gob.ni/?page_id=131

atlassian. (2019). *Backlog*. https://www.atlassian.com/es/agile/scrum/backlogs

Computing, R. C., & Computing, R. C. (2023). *¿Qué es la generación aumentada de recuperación?* https://www.revistacloudcomputing.com/2023/08/que-es-la-generacion-aumentada-de-recuperacion/

deloitte. (2024). *Scrum: roles y responsabilidades*. https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html

Formal, T., Lassance, C., Piwowarski, B., & Clinchant, S. (2021). *SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval*. https://arxiv.org/abs/2109.10086

IBM. (2021). *Neural Networks*. https://www.ibm.com/think/topics/neural-networks

Introducción a Django - Aprende desarrollo web | *MDN*. (n.d.). https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

Introduction | *Ragas*. (n.d.). https://docs.ragas.io/en/stable/

Martins, J. (2024). *Scrum: conceptos clave y cómo se aplica en la gestión de proyectos [2024] • Asana*. https://asana.com/es/resources/what-is-scrum

Na, & Na. (2024). *LLM: ¿Qué son los Grandes Modelos de Lenguaje?* | *Aprende Machine Learning*. https://www.aprendemachinelearning.com/llm-que-son-los-grandes-modelos-de-lenguaje/

Qasem, R., Tantour, B., & Maree, M. (2023). Towards the exploitation of llm-based chatbot for providing legal support to palestinian cooperatives. *ArXiv Preprint ArXiv:2306.05827*.

Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. https://arxiv.org/abs/1908.10084

Stryker, C., & Bergmann, D. (2025). *Transformer Model*. https://www.ibm.com/think/topics/transformer-model

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., & Wei, F. (2024). Multilingual E5 Text Embeddings: A Technical Report. *ArXiv Preprint ArXiv:2402.05672*.

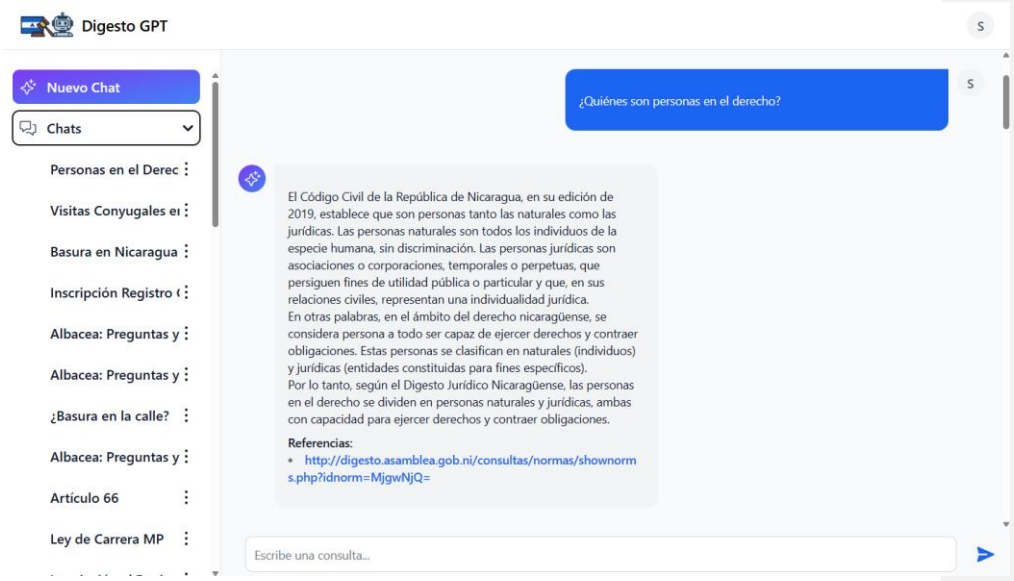
What is Python? Executive Summary. (n.d.). https://www.python.org/doc/essays/blurb/

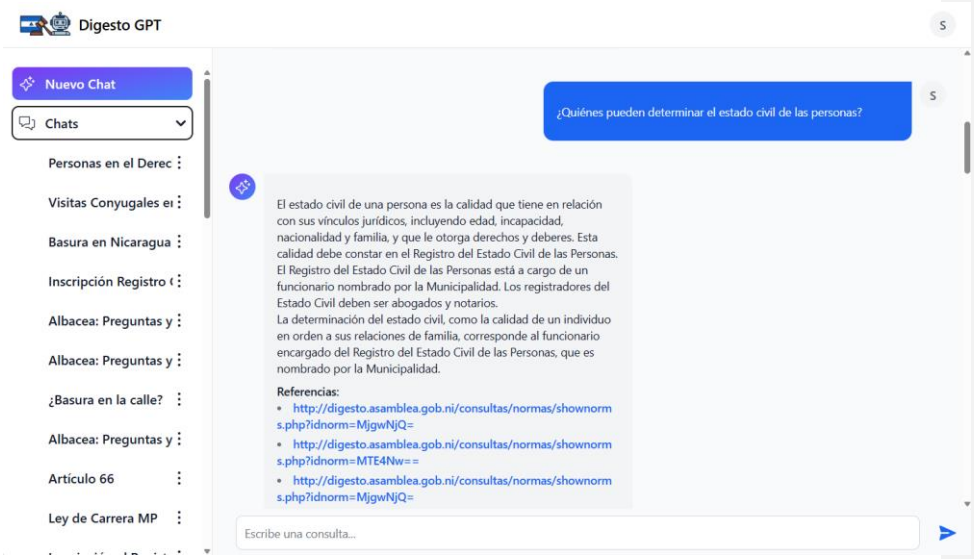
Witten, M. (2023). *Applying generative AI to law: Opportunities and risks* | *Queen's Law*. https://law.queensu.ca/news/Applying-generative-AI-to-law

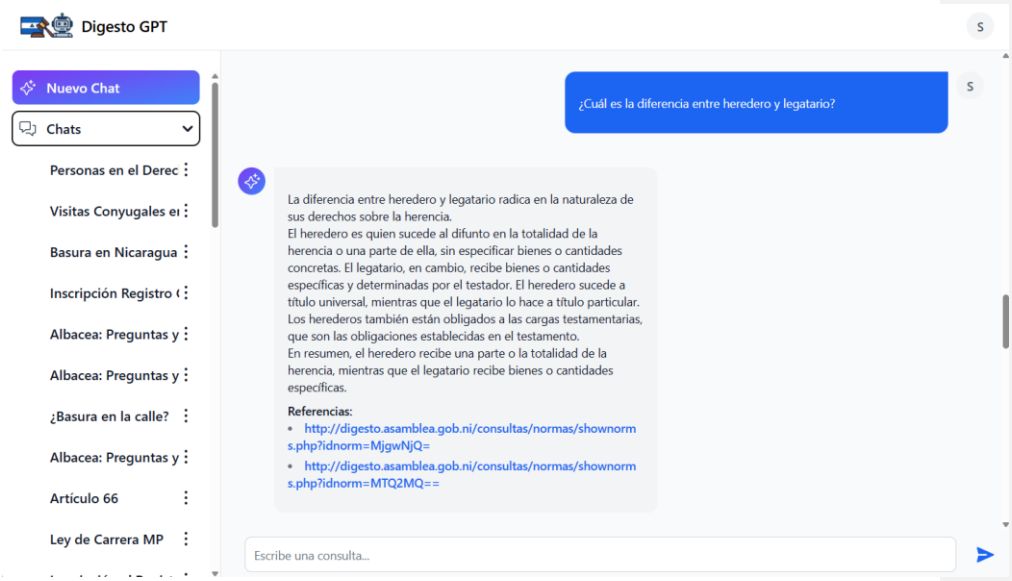
Anexos

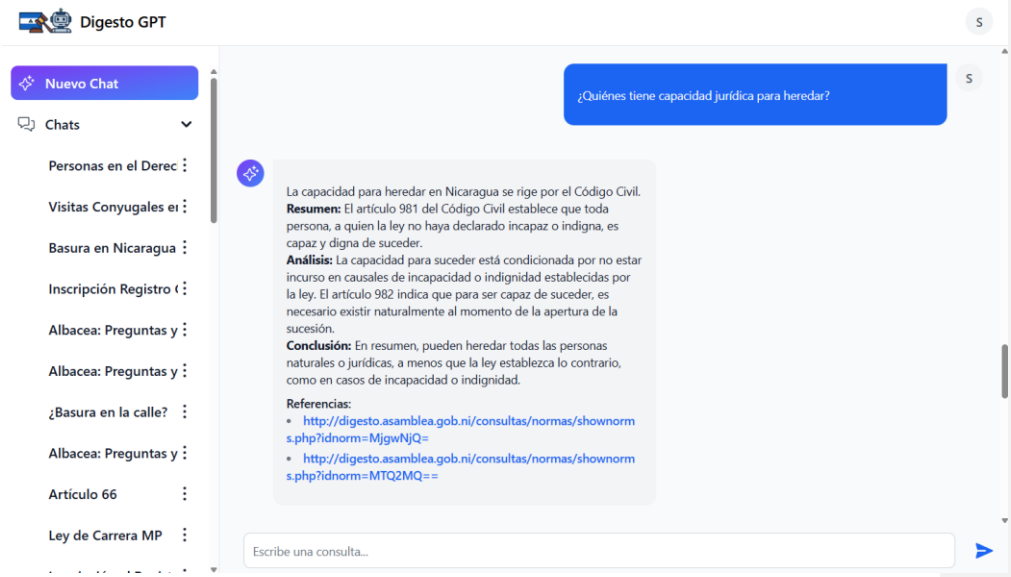
Comentado [dc1]: Se pueden poner las pruebas realizadas como evidencia que fue utilizado y probado por abogados. Es valido poner algunas capturas de pantalla, también expresar cuantas personas hicieron pruebas con el mismo y sus opiniones al respecto.

A. Pruebas realizadas al chatbot por usuarios






Digesto GPT
S
Nuevo Chat
Chats
Personas en el Derec
Visitas Conyugales e
Basura en Nicaragua
Inscripción Registro
Albacea: Preguntas y
Albacea: Preguntas y
¿Basura en la calle?
Albacea: Preguntas y
Artículo 66
Ley de Carrera MP
¿Cuál es la diferencia entre heredero y legatario?
La diferencia entre heredero y legatario radica en la naturaleza de sus derechos sobre la herencia.
El heredero es quien sucede al difunto en la totalidad de la herencia o una parte de ella, sin especificar bienes o cantidades concretas. El legatario, en cambio, recibe bienes o cantidades específicas y determinadas por el testador. El heredero sucede a título universal, mientras que el legatario lo hace a título particular. Los herederos también están obligados a las cargas testamentarias, que son las obligaciones establecidas en el testamento.
En resumen, el heredero recibe una parte o la totalidad de la herencia, mientras que el legatario recibe bienes o cantidades específicas.
Referencias:
http://digesto.asamblea.gob.ni/consultas/normas/shownorms.php?idnorm=MjgwNjQ=
http://digesto.asamblea.gob.ni/consultas/normas/shownorms.php?idnorm=MTQ2MQ==
Escribe una consulta...


Digesto GPT
S
Nuevo Chat
Chats
Personas en el Derec
Visitas Conyugales e
Basura en Nicaragua
Inscripción Registro
Albacea: Preguntas y
Albacea: Preguntas y
¿Basura en la calle?
Albacea: Preguntas y
Artículo 66
Ley de Carrera MP
¿Quiénes tiene capacidad jurídica para heredar?
La capacidad para heredar en Nicaragua se rige por el Código Civil.
Resumen: El artículo 981 del Código Civil establece que toda persona, a quien la ley no haya declarado incapaz o indigna, es capaz y digna de suceder.
Análisis: La capacidad para suceder está condicionada por no estar incurso en causales de incapacidad o indignidad establecidas por la ley. El artículo 982 indica que para ser capaz de suceder, es necesario existir naturalmente al momento de la apertura de la sucesión.
Conclusión: En resumen, pueden heredar todas las personas naturales o jurídicas, a menos que la ley establezca lo contrario, como en casos de incapacidad o indignidad.
Referencias:
http://digesto.asamblea.gob.ni/consultas/normas/shownorms.php?idnorm=MjgwNjQ=
http://digesto.asamblea.gob.ni/consultas/normas/shownorms.php?idnorm=MTQ2MQ==
Escribe una consulta...

B. Set de datos de prueba y resultados obtenidos utilizando RAGAS

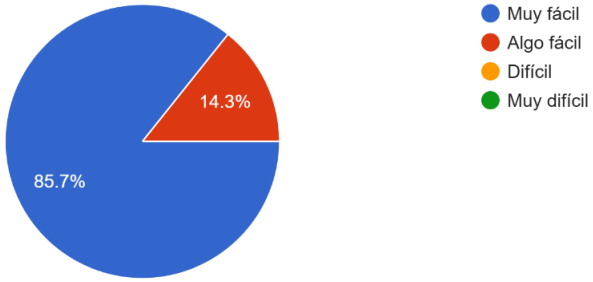
user_input	retrieved_contexts	response	reference	context_recall	context_precision	faithfulness	answer_relevancy
¿Quiénes son personas en el derecho?	['Numero: 870\n\nTitulo: Código de Fan	Según el Código Civil de la Repúblic	Artículo 1 Es persona todo ser capaz	0.5	0.208333333	1	0.875916138
¿Qué es el domicilio?	['Numero: 870\n\nTitulo: Código de Fan	El domicilio se define como el lugar d	El domicilio de una persona es el luga	1	0.792857143	1	0.85893695
¿La muerte civil es valida en la legislación nicaragüense?	['Numero: s/n\n\nTitulo: Código Civil de	El Código Civil de la República de Nic	Artículo 46 Termina la existencia de la	1	1	1	0.940166873
¿Quiénes pueden determinar el estado civil de las personas?	['Numero: 870\n\nTitulo: Código de Fan	El Código Civil de la República de Nic	Los Registradores del Estado Civil de	1		1	0.860839597
¿Cuánto tiempo tiene un padre para inscribir a su hijo en el registro?	['Numero: s/n\n\nTitulo: Código Civil de	El padre o la madre tienen hasta veint	Artículo 510 Todo padre o madre pers	1	0.527083333	0.866666667	0.90426907
¿En que materia se castigan los delitos de carácter civil?	['Numero: 641\n\nTitulo: Código Penal\n	De acuerdo con el Digesto Jurídico N	Artículo 586 Los delitos cometidos co	1	0.62202381	0.833333333	0.883604294
¿Cuáles son los tipos de bienes que se regulan en Nicaragua?	['Numero: s/n\n\nTitulo: Código Civil de	Los tipos de bienes regulados en Nic	Artículo 597 Los bienes consisten en	1	1	1	0.940792883
¿Qué pasa si encuentro un tesoro en mi propiedad? ¿Soy libre de quedármelo?	['Numero: s/n\n\nTitulo: Código Civil de	El Código Civil de Nicaragua, en su a	Artículo 619 El tesoro oculto pertenec	1	0.855257936	1	0.861333951
¿Es permitido tomar posesión de algo que no tiene dueño?	['Numero: s/n\n\nTitulo: Código Civil de	La respuesta a tu pregunta se encuen	Artículo 655 Es lícito a cualquiera apr	1	0.97654321	1	0.926906348
¿Cuáles son los requisitos para la caza de animales en Nicaragua?	['Numero: s/n\n\nTitulo: Código Civil de	No es posible responder a la pregunta	Artículo 656 Es lícito a todos, sin disti	1	0.475	0.913043478	0
¿Que pasa si encuentro un objeto aleatorio y no hay dueño aparente?	['Numero: s/n\n\nTitulo: Código Civil de	En Nicaragua, si encuentras un objeto	Artículo 690 Los bienes muebles aba	0	0.675680272	0.8	0.913781225
¿Qué es prescripción negativa o positiva?	['Numero: s/n\n\nTitulo: Código Civil de	La prescripción, según el Código Civi	Artículo 869 La adquisición de bienes	1		1	0.836631146
¿Cuáles son los requisitos de la prescripción positiva?	['Numero: s/n\n\nTitulo: Código Civil de	La prescripción positiva, según el Cód	Artículo 888 La posesión necesaria p	1	0.608333333	1	0.88106041
¿Cuál es el principal factor para que se dé una sucesión?	['Numero: s/n\n\nTitulo: Código Civil de	El principal factor para que se dé una	Artículo 934 La sucesión de una pers	1	0.786190476	1	0.899489448
¿Cuál es la diferencia entre heredero y legatario?	['Numero: s/n\n\nTitulo: Código Civil de	La diferencia entre heredero y legatar	Artículo 935 Llámase heredero aquel	1	0.361904762	1	0.891909107
¿Quiénes tiene capacidad jurídica para heredar?	['Numero: s/n\n\nTitulo: Código Civil de	* **Resumen del extracto relevante:**	Artículo 944 Toda persona natural o ju	1	0.553703704	0.607142857	0.874818585
¿Qué es un testamento?	['Numero: s/n\n\nTitulo: Código Civil de	Un testamento, según el artículo 945 c	Artículo 945 Testamento es un acto m	1	1	1	0.884424969
¿Cuándo deja de disponer una persona de una herencia?	['Numero: s/n\n\nTitulo: Código Civil de	La información proporcionada en el c	Artículo 977 Las disposiciones testan	1	0.666666667	1	0.923125489
¿Quiénes NO pueden heredar?	['Numero: s/n\n\nTitulo: Código Civil de	El Código Civil de la República de Nic	Artículo 979 No son hábiles para testa	1	0.1625	0.777777778	0.875638128
¿Qué es un albacea?	['Numero: s/n\n\nTitulo: Código Civil de	El albacea es la persona designada p	Artículo 1303 Albacea o ejecutor testa	1	0.821088435	0.941176471	0.910562704
¿En qué consiste el usufructo?	['Numero: s/n\n\nTitulo: Código Civil de	El usufructo, según el Código Civil de	Artículo 1473 El usufructo es el derecl	1	0.916666667	1	0.883129156
¿en que consiste la ley de justicia constitucional?	['Numero: 983\n\nTitulo: Ley de Justicia	La Ley de Justicia Constitucional, seg	La presente ley constitucional, tiene c	1	1	1	0.90201869
¿Cuáles son los principios constitucionales que existen?	['Numero: 983\n\nTitulo: Ley de Justicia	El Digesto Jurídico Nicaragüense est	1. Principio de supremacía constituci	1	0.755555556	1	0.892330808
¿Quiénes pueden ejercer la justicia constitucional?	['Numero: 870\n\nTitulo: Código de Fan	El Digesto Jurídico nicaragüense, en l	Son órganos competentes de la justic	1		1	0.904065359
¿Cuál es el órgano encargado de supervisar las elecciones electorales?	['Numero: 331\n\nTitulo: Ley Electoral\n\n	El órgano encargado de supervisar la	El Poder Electoral se encargará de or	1	0.810615079	1	0.967627
¿Cuáles son los requisitos para ser magistrado?	['Numero: 331\n\nTitulo: Ley Electoral\n\n	Para ser magistrado del Consejo Sup	1) Ser nacional de Nicaragua. En el c	1	0.583333333	1	0.890421181
¿Cuánto tiempo puede ejercer un magistrado?	['Numero: 331\n\nTitulo: Ley Electoral\n\n	Según el artículo 9 de la Ley Electoral	Los Magistrados y Magistradas del C	1	1	1	0.892744108
¿Cuáles son los requisitos que deben llenarse para ejercer la presidencia?	['Numero: 331\n\nTitulo: Ley Electoral\n\n	No es posible responder a esta pregu	El Presidente o Presidenta, las y los r	1	0.583333333	0.875	0
¿Qué edad se requiere para poder participar como ciudadano de las elecciones electo	['Numero: 331\n\nTitulo: Ley Electoral\n\n	La Ley Electoral establece la edad pa	El sufragio universal, igual, directo, lib	0.5	1	1	0.917210148
¿Cuáles son los requisitos para participar de las elecciones?	['Numero: 331\n\nTitulo: Ley Electoral\n\n	Para participar en las elecciones en N	Para ejercer el derecho al sufragio las	1	0.25	1	0.916849554
¿Qué cosas están prohibidas en días de votación?	['Numero: 331\n\nTitulo: Ley Electoral\n\n	En el día de las votaciones, la Ley Ele	El día de las votaciones se prohíbe: 1	1		0.958333333	0.918789576
¿Me puedo casar en el extranjero y mi matrimonio será válido en Nicaragua?	['Numero: 870\n\nTitulo: Código de Fan	El matrimonio celebrado en el extranje	El matrimonio celebrado en otro país	1	0.642757936	1	0.908372916
¿Me puedo divorciar en el extranjero y mi matrimonio será válido en Nicaragua?	['Numero: 870\n\nTitulo: Código de Fan	El Código de Familia de Nicaragua a	La disolución del vínculo matrimonial e	1	0.789285714	0.681818182	0.914373462
Concepto de familia en la legislación	['Numero: 870\n\nTitulo: Código de Fan	La legislación nicaragüense define la	La familia es el núcleo fundamental de	1	0.936734694	1	0.871769214
¿Qué el parentesco?	['Numero: 870\n\nTitulo: Código de Fan	El parentesco, según el Código de Fa	El parentesco es el vínculo que une a	1	0.86	0.866666667	0.870769585
¿Qué tipos de violencia contempla el código de familia?	['Numero: 870\n\nTitulo: Código de Fan	El Código de Familia de Nicaragua, e	a) Violencia física: Son las acciones,	1	0.852857143	1	0.900645781
¿A que edad se puede contraer matrimonio en Nicaragua?	['Numero: 870\n\nTitulo: Código de Fan	El Código de Familia de Nicaragua e	Son aptos legalmente para contraer n	1		1	0.929803092
¿Quiénes tienen autoridad para casar y declarar uniones de hecho estable?	['Numero: 870\n\nTitulo: Código de Fan	Claro, aquí está la información sobre l	Las personas autorizadas para celeb	1	0.68452381	1	0.916842998
¿Requisitos para contraer matrimonio?	['Numero: 870\n\nTitulo: Código de Fan	Para contraer matrimonio en Nicarag	Quienes intenten contraer matrimonio	1		1	0.905520545
¿Qué es la unión de hecho estable?	['Numero: 870\n\nTitulo: Código de Fan	La unión de hecho estable, según el C	La unión de hecho estable descansa	1	1	1	0.86801343
¿Quiénes pueden adoptar?	['Numero: 870\n\nTitulo: Código de Fan	El Código de Familia de Nicaragua e	Pueden adoptar las personas legitima	1	0.812131519	1	0.890632678
¿Qué es el ciberdelito?	['Numero: 1042\n\nTitulo: Ley Especial	El ciberdelito se define en la Ley Esp	4. Ciberdelitos: Acciones u omisiones	1	0.716666667	1	0.861438227
¿Qué formas de violencia contra la mujer se regulan en Nicaragua?	['Numero: 779\n\nTitulo: Ley Integral Co	La Ley Integral Contra la Violencia ha	La violencia hacia la mujer en cualqui	1	1	1	0.927913456
¿Qué medidas de atención se le dan a las víctimas de violencia?	['Numero: 779\n\nTitulo: Ley Integral Co	Se establecen diversas medidas para	Las medidas para la atención a las ví	1	0.832993197	1	0.884252903
¿Cuáles son los atenuantes de responsabilidad penal?	['Numero: 641\n\nTitulo: Código Penal\n	El Digesto Jurídico Nicaragüense, es	Son circunstancias atenuantes: 1. Ex	1	0.625	1	0.884398154
¿Cuáles son las circunstancias agravantes que se regulan en derecho penal?	['Numero: 641\n\nTitulo: Código Penal\n	Comencemos con un resumen de las	Son circunstancias agravantes: 1. Ale	1	0.968253968	1	0.898319533
¿Qué es alevosía?	['Numero: 641\n\nTitulo: Código Penal\n	La alevosía, según el Código Penal d	Hay alevosía cuando el culpable come	1	0.809523809	1	0.88483949
¿Cómo se clasifican las penas?	['Numero: 641\n\nTitulo: Código Penal\n	Las penas se clasifican en Nicaragua	Las penas se clasifican en principales	1	1	1	0.897039876
¿Cuál es el máximo estimado de una pena de cárcel?	['Numero: 641\n\nTitulo: Código Penal\n	El Código Penal de Nicaragua establ	La pena de prisión tendrá una duració	0	0.266666667	1	0.88722563
¿Que es la responsabilidad civil?	['Numero: 641\n\nTitulo: Código Penal\n	La responsabilidad civil, según el Cód	Toda persona penalmente responsab	1	0.866666667	1	0.869853262
¿Qué es el estupro?	['Numero: 641\n\nTitulo: Código Penal\n	El estupro, según el Código Penal de	Quien estando casado o en unión de l	1	0.666666667	1	0.867385251

C. Retroalimentación obtenida por parte de usuarios

A los usuarios que utilizaron el sistema, se les pidió que llenaran un formulario con el objetivo de recopilar retroalimentación cualitativa y cuantitativa sobre su experiencia de uso. El formulario incluyó preguntas relacionadas con la facilidad de uso de la interfaz, la claridad de las respuestas generadas por el sistema, la relevancia de los resultados obtenidos a partir de sus consultas, así como sugerencias para posibles mejoras. Esta información fue fundamental para evaluar el grado de satisfacción de los usuarios, identificar oportunidades de mejora en la interacción hombre-máquina y validar la efectividad del sistema en escenarios reales de uso, especialmente dentro del contexto jurídico. En total fueron 7 usuarios que llenaron el formulario.

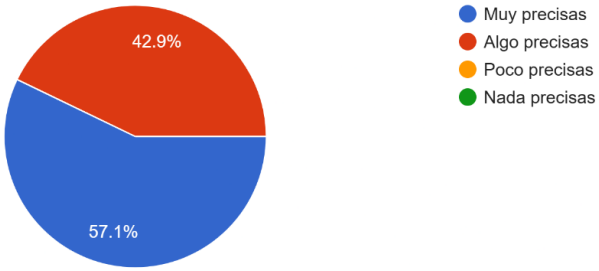
¿Qué tan fácil fue interactuar con el chatbot?

7 respuestas



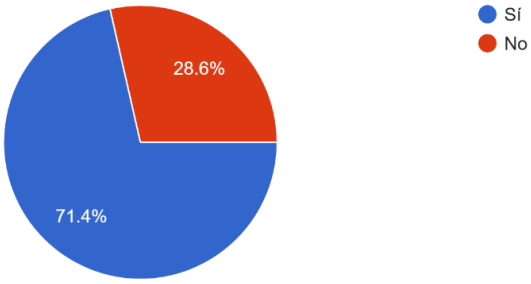
¿Cómo calificarías la precisión de las respuestas del sistema?

7 respuestas



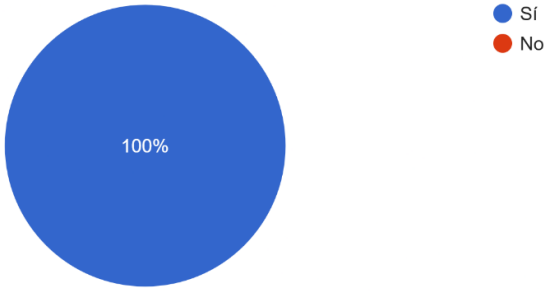
¿La información que recibiste fue suficiente para resolver tus consultas jurídicas?

7 respuestas



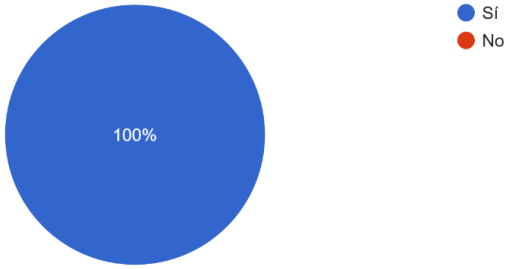
¿Consideras que la interfaz del sistema es clara e intuitiva?

7 respuestas



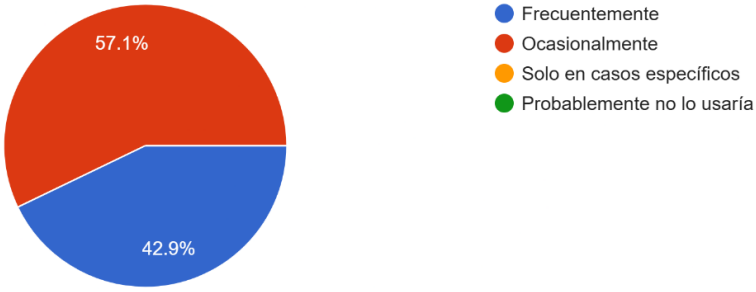
¿Recomendarías Digesto GPT a otras personas que necesiten información jurídica?

7 respuestas



¿Con qué frecuencia utilizarías Digesto GPT?

7 respuestas



¿Qué funcionalidades o informaición adicional te gustaría que tuviera Digesto GPT?

7 respuestas

- Jurisprudencia y doctrina
- Tal vez mas detalles sin que sea necesario revisar las referencias
- Entre más precisa sea la respuesta mejor, sin embargo, habrá ocasiones en que una explicación más detallada podría ser necesaria. También podría implementar sobre temas específicos información sobre cómo se aplican estas leyes durante algún proceso legal, esto en teoria deberia estar en códigos procesales pero no siempre detallan u ordenan bien los pasos de cada proceso.
- Que guarde contexto de la conversación
- Ninguna
- Que siga el hilo de la conversacion
- Un poco mas de detalle en las respuestas

¿Qué aspectos del sistema te gustaría que mejoráramos?

7 respuestas

- Ampliar la información
- colores cuando el sistema esta configurado con tema claro
- Únicamente la implementación de respuestas más elaboradas, cuando el usuario solicite la explicación de algo.
- Que admita documentos e imágenes
- Ninguna
- Ninguno
- Nada

¿Deseas compartir algún otro comentario o sugerencia?

3 respuestas

Es altamente aplicable o configurable en cualquier tema, podria ser interesante tener una herramienta de entrenamiento y otros temas y levantar diferentes instancias, podria aplicarlos a diferentes rubros.

Como estudiante de derecho doy por hecho que esta sería una herramienta útil para juristas en desarrollo, es común tener dudas sobre aspectos específicos como conceptos o pasos en algún proceso, las cuales son necesarias para entender temas más complejos, por eso, una herramienta que sea capaz de responder cosas basadas en la legislación nicaragüense sería de mucha utilidad.

Deshabilitar btn send cuando el input está vacío

D. Despliegue del sistema

Para el despliegue del sistema se utilizó una instancia EC2 de tipo t3a.medium en AWS, la cual es un tipo de instancia de propósito general, que cuenta con 2 vCPU, 4 GiB de memoria RAM. Esta instancia se aprovisionó con 28 GiB de almacenamiento y una IP estática para la utilización de un nombre de dominio. Para el nombre de dominio se utilizó name.com, un proveedor de nombres de dominio y el nombre de dominio que se definió fue https://digestogpt.works.

Es importante mencionar que, esta instancia se mantendrá activa hasta la fecha de la defensa final de la tesis, luego de esto se eliminará y el sistema dejará de estar disponible.

A continuación, se muestran capturas de las configuraciones e información general de la instancia en la consola de AWS: