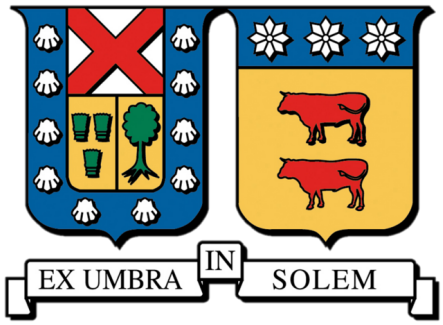
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



"PLATAFORMA DIGITAL PARA INTERACCIÓN CON MODELOS DE LENGUAJE DE GRAN TAMAÑO Y EL ANÁLISIS DE SU COMPORTAMIENTO"

VANIA SCHATLOFF VALENZUELA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL TELEMÁTICO

PROFESOR GUÍA:	**MAURICIO ARAYA**
PROFESOR CORREFERENTE:	**NICOLÁS JARA**

Abril 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Plataforma digital para interacción con modelos de lenguaje de gran tamaño y el análisis de su comportamiento

Nombre del candidato(a): Vania Schatloff Valenzuela

Carrera / Grado: Ingeniería Civil Telemática

Campus: Casa Central Valparaiso **;** **Departamento**: Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Mauricio Araya, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 08/10/2025 ; **Firma:** _____

Estudiante o Candidato(a):

Fecha: 08/10/2025 ; **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

Quiero agradecer a mi familia, por su amor, apoyo y confianza en mí durante este camino. A mi pareja, gracias por estar a mi lado y por su paciencia en los momentos más difíciles. Y a mis amigos, por siempre estar ahí cuando los necesité.

Resumen

Este informe aborda la implementación de una plataforma digital segura que democratiza el acceso a datos mediante modelos de lenguaje de gran tamaño (LLM). El objetivo principal es eliminar barreras en el acceso y análisis de datos en el ámbito de la salud, permitiendo consultas en lenguaje natural traducidas automáticamente a SQL para su ejecución en bases de datos. La plataforma incluye funcionalidades como manejo de sesiones, encuestas de satisfacción, registros de conversaciones, y control de privilegios.

El proyecto se desarrolló en etapas, destacándose la investigación inicial, diseño de la arquitectura, desarrollo de módulos (frontend, backend, API y base de datos), y creación de un prototipo funcional. Los resultados incluyen un sistema robusto con vistas para analizar métricas de satisfacción y registros de conversaciones, una API integrada con Amazon Bedrock, y funcionalidades avanzadas como generación de gráficos y evaluación del desempeño del modelo.

Se adoptaron prácticas de desarrollo seguras y escalables, garantizando una experiencia intuitiva y adaptable para diversos perfiles de usuario.

Glosario

LLM Modelos de lenguaje de gran tamaño entrenados con grandes cantidades de datos textuales. Son utilizados para generar y comprender texto en lenguaje natural.

RAG Técnica de *Retrieval-Augmented Generation* que combina un modelo generativo con un sistema de recuperación de información para generar respuestas más precisas utilizando datos contextuales.

SQL Lenguaje estándar para gestionar y manipular bases de datos relacionales, utilizado para generar consultas estructuradas.

Prompt Engineering Técnica para diseñar instrucciones (*prompts*) que guían el comportamiento de modelos de lenguaje como los LLM, optimizando la relevancia de sus respuestas.

Fine-tuning Proceso de ajuste fino de un modelo preentrenado utilizando datos específicos de una tarea para mejorar su desempeño.

Embedding Models Modelos que convierten palabras, frases o datos en representaciones vectoriales, facilitando tareas como búsqueda y análisis.

Supervised Fine-Tuning Método de ajuste fino basado en datos etiquetados para entrenar modelos de manera específica y mejorar su desempeño en tareas concretas.

Schema Linking Proceso de identificar tablas y columnas relevantes de una base de datos para convertir preguntas en lenguaje natural en consultas SQL precisas.

JSON *JavaScript Object Notation*, formato de intercambio de datos legible por humanos que organiza objetos estructurados.

Backend Parte de un sistema que maneja la lógica de negocio y el procesamiento de datos, interactuando con bases de datos y servicios externos.

Frontend Interfaz visible para el usuario que permite la interacción directa con el sistema.

API *Application Programming Interface*, conjunto de reglas y protocolos para la comunicación entre diferentes componentes de software.

ORM *Object Relational Mapping* es una herramienta que permite la interacción entre una aplicación y una base de datos relacional.

Índice de figuras

Índice de cuadros

Índice general

Capítulo 1

Introducción

En el ámbito de la salud pública, el acceso y análisis de datos son esenciales para respaldar decisiones informadas y promover avances en la investigación médica. Sin embargo, persisten barreras significativas en la democratización de los datos, especialmente cuando el acceso está limitado por la necesidad de intermediarios con conocimientos especializados en los modelos de datos de cada organización. Esta dependencia puede ralentizar los procesos y dificultar el uso eficiente de la información necesaria para la investigación y la toma de decisiones estratégicas en salud.

La falta de accesibilidad directa afecta negativamente a diversas áreas del sector, restringiendo su capacidad para utilizar los datos de manera autónoma y efectiva. Es necesario desarrollar soluciones que democraticen el acceso a los datos, permitiendo a los usuarios consumir y analizar información sin depender de intermediarios especializados. Estas soluciones deben simplificar los procesos y garantizar que los datos estén disponibles de manera comprensible y utilizable para todos los actores involucrados.

Según [1], la equidad en los datos abarca desde la recolección hasta el acceso, y es fundamental para identificar y abordar las desigualdades en salud. La autora destaca que, durante la pandemia de COVID-19, la falta de datos precisos y accesibles limitó la capacidad de la salud pública para implementar acciones efectivas a nivel local. Además, resalta la importancia de prácticas de recolección de datos que consideren

a comunidades minoritarias y marginadas, asegurando que la información obtenida se utilice para mejorar su salud y bienestar.

Por otro lado, el informe de Digital Health Insights [2] enfatiza que la democratización de datos en el sector salud implica no solo aumentar el acceso a la información, sino también garantizar que esta sea comprensible y utilizable por todos los actores, independientemente de su formación técnica. El informe señala que, aunque el aumento del acceso a los datos en el cuidado de la salud mejora los resultados de los pacientes, un sistema de salud solo habrá logrado la democratización de datos una vez que haya completado todos los pasos necesarios para garantizar que los datos sean accesibles y utilizables de manera efectiva.

Esta memoria tiene como objetivo desarrollar una plataforma web segura que facilite el acceso a información y funcionalidades de acuerdo con los privilegios de cada usuario. La implementación de esta solución permitirá a la Fundación Arturo López Pérez (FALP) optimizar el trabajo de sus analistas e investigadores, agilizando la consulta y análisis de datos de manera eficiente y estructurada.

La plataforma contará con un chatbot, un programa que simula conversaciones humanas, que permitirá interactuar humanamente con la información de la base de datos del cliente, además de ofrecer la obtención de métricas y conversaciones para el posterior análisis del comportamiento del chatbot y su recepción en el equipo.

Para los usuarios con privilegio de consultas al chatbot, la plataforma ofrecerá la capacidad de realizar consultas a través de un chat conectado a un chatbot. Este chatbot estará conectado a un sistema de modelos de lenguaje de gran tamaño (LLM) especializados para entender consultas en lenguaje natural y traducirlas a consultas SQL que se ejecutan en las bases de datos de la fundación, para así posteriormente con la información obtenida traducir a lenguaje natural y entregarle esta información al usuario, así como la posibilidad de descargar la data previo a su traducción. Una vez terminado este proceso, se llega a una evaluación de satisfacción de la interacción que deberá responder el usuario.

Por otro lado, los usuarios con más privilegios tienen acceso a métricas de satisfac-

ción de los usuarios y registros de conversaciones. Esto permite analizar el desempeño del modelo LLM y la calidad de las respuestas entregadas al cliente. También existirán usuarios con máximo privilegio, que serán considerados administradores y podrán administrar los accesos de cada usuario al sistema.

De esta manera, la solución propuesta consiste en una plataforma que alberga un chatbot capaz de consultar las bases de datos del cliente y brindar un análisis del uso del mismo.

1.1. Objetivos

1.1.1. Objetivo General

El objetivo de este trabajo consiste en facilitar el acceso a datos para aquellas personas que no tienen conocimientos técnicos.

1.1.2. Objetivos Específicos

- La LLM debe traducir las consultas en lenguaje natural a lenguaje SQL.
- Elaborar una API.
- El sistema debe administrar las métricas de satisfacción del usuario respecto a su interacción con la LLM.
- El sistema debe administrar el registro de conversaciones de los usuarios con la LLM.
- El sistema debe ser accesible mediante una plataforma.

1.2. Estructura

El capítulo 3 analiza las distintas soluciones existentes que abordan el desafío planteado en esta memoria. En el capítulo 4, se definen y detallan los requisitos del sistema. Luego, en el capítulo 5, se presenta el diseño del sistema, describiendo su arquitectura, módulos y la interacción entre ellos. El capítulo 6 expone los prototipos de las vistas, los eventos principales de la plataforma y su modelo de navegación. Posteriormente, en el capítulo 7, se establece un marco general del proyecto y se profundiza en su desarrollo. Finalmente, el capítulo 8 revisa los componentes del sistema y sus funcionalidades.

Capítulo 2

Marco teórico

Para el análisis de datos, la mayor parte de las instituciones y empresas utilizan herramientas como Excel y Power BI, y esta información suele ser entregada a los analistas por personas con acceso directo a las bases de datos. Los usuarios sin conocimientos técnicos actualmente no son capaces de obtener la información que necesitan mediante consultas en lenguaje natural. La creación de una plataforma intermediaria entre el usuario y la base de datos permitiría entregar información sin requerir la intervención de un funcionario técnico, y además sería más eficiente, ya que el usuario recibiría esta información de manera inmediata (o lo que demore en ejecutarse la consulta SQL en la base de datos).

A partir de esto, se proponen las siguientes hipótesis para validar la efectividad de la solución:

- **H1:** Los usuarios sin conocimientos técnicos pueden obtener respuestas útiles mediante consultas en lenguaje natural.
- **H2:** El sistema permite disminuir los tiempos de respuesta en comparación a los métodos tradicionales de solicitud de información.
- **H3:** Las respuestas generadas por el sistema son comprendidas por los usuarios sin necesidad de ayuda externa.

- **H4:** El sistema limita correctamente el acceso a funcionalidades según los privilegios de cada usuario.

Capítulo 3

Estado del arte

La generación de consultas SQL a partir de lenguaje natural ha evolucionado significativamente en los últimos años, dando lugar a diversas soluciones que combinan técnicas de procesamiento de lenguaje natural, aprendizaje automático y arquitecturas basadas en modelos de lenguaje. Este capítulo explora los enfoques más relevantes en el campo, destacando las contribuciones de distintos autores y analizando los avances logrados mediante diferentes estrategias y frameworks.

3.1. Plataforma y Framework para Generación de SQL

Para empezar, en ([3]), se crea un sistema con GUI y manejo de errores, donde el usuario puede ver los contenidos de una tabla, ingresar una query en sql o en lenguaje natural. El usuario debe iniciar sesión en el sistema, puede hacer preguntas que involucren la obtención, ingreso, actualización y eliminación de datos. La frase ingresada por el usuario pasará por una tokenización, lo que significa que se separará la frase en palabras y cada una representa un token, luego pasa al análisis léxico, donde la lista tokenizada se mapea con un diccionario, por lo que estas palabras son reemplazadas por las palabras del diccionario relacionadas a la base de datos, luego se analiza sintácticamente, donde cada palabra tokenizada se mapea con atributos de otro diccionario, y por último se pasa por un análisis semántico, donde el sistema encontrará palabras

que representen condiciones o símbolos y esa palabra se mapeará con el diccionario. El resultado es una consulta que luego puede ser ejecutada en la base de datos.

3.2. Ingeniería de prompt y Fine-Tuning

En el ámbito de las estrategias de ingeniería de prompt, en ([4]) se estudian empíricamente varias LLMs de código abierto para destacar el potencial de estas en in-context learning y supervised fine-tuning con varias representaciones. Se demuestra que, similar al aprendizaje in-context, la estrategia de representación es crítica para supervised fine-tuning. También, luego de fine-tuning, se observa una disminución en las capacidades de aprendizaje in-context. La solución integrada propuesta por el paper, DAIL-SQL, tiene un 86.6 % de exactitud en la ejecución y tiene el primer lugar en el spider leaderboard, una clasificación que evalúa modelos de inteligencia artificial en la tarea de Text-to-SQL.

Por otro lado, en ([5]) se plantea que la capacidad de comprensión de un LLM no es suficiente cuando se le proporciona únicamente el esquema de la base de datos y la consulta que se desea realizar, ya que el resultado tiende a ser impreciso. Por esta razón, se propone un framework llamado Knowledge-to-SQL, que utiliza un LLM experto en datos (DELLM). Esto se debe a que el desarrollo de una arquitectura no humana capaz de generar información experta a partir de un conjunto de datos aporta un gran valor.

Los desafíos en la generación de conocimiento experto incluyen la especialización de la consulta y la base de datos, la conciencia del contenido de la base de datos y la mejora del rendimiento esperado. Para abordar estos desafíos, se introduce PLDBF (Preference Learning via Database Feedback), un mecanismo que proporciona recompensas a DELLM en función del grado en que el conocimiento generado ayuda a recuperar datos precisos. Este proceso permite que los LLM generen consultas SQL más exactas y alineadas con la estructura y el contenido de la base de datos.

El método propuesto consta de tres módulos principales:

El módulo de ajuste fino supervisado de DELLM, que genera conocimiento basado

en la consulta del usuario y el esquema de la base de datos.

El módulo de retroalimentación, encargado de evaluar la calidad de las consultas generadas y proporcionar correcciones o mejoras.

El módulo de aprendizaje por preferencias con PLDBF, que optimiza el rendimiento del modelo mediante un sistema de recompensas basado en la precisión de las consultas SQL generadas.

En conjunto, estos componentes permiten que Knowledge-to-SQL mejore significativamente la precisión de las consultas SQL generadas por los LLM, haciendo que sean más adecuadas para entornos complejos donde la interpretación del esquema de la base de datos y la recuperación de información precisa son críticas.

En un enfoque diferente, ([6]) propone supervised fine tuning (SFT) como otra opción al fine tuning regular para entrenar LLM para una mejor tarea de generación de text-to-SQL. Se utiliza Llama-V2 por sus distintas ventajas arquitectónicas, el modelo alcanza resultados de primer nivel en el conjunto de datos Spider, con una notable precisión de ejecución del 89,6 %, junto con una precisión de coincidencia exacta del 86,8 %. El documento explica las técnicas de entrenamiento para LLMs, como el few show prompting, fine tuning o el supervised fine tuning, donde se explica por qué las dos primeras opciones se quedan atrás frente a la última en cuanto complejidad de consultas o cuando se trabaja con modelos pequeños respectivamente. Por otro lado, supervised fine tuning utiliza datos etiquetados. Se pasa todo el contexto a la vez, pero la pérdida final se figura solo sobre la etiqueta que el modelo debe generar. Esto permite que el modelo aprenda solo la generación sintáctica de la etiqueta en lugar de toda la declaración.

3.3. Modelos multiagentes, redes generativas adversariales (GANs) y optimización de consultas

Con otro enfoque, en ([7]) se introduce un framework multiagente basado en LLM, que explota esto con diferentes funcionalidades para que se haga un análisis efectivo de text-to-SQL. MAC-SQL comprende un agente que funciona como descomponedor principal para la generación de text-to-SQL, que, junto a dos agentes auxiliares, selector para el uso de herramientas y refinador para la refinación de SQL. El descomponedor toma una pregunta y la separa en sub-preguntas más simples y las resuelve progresivamente mediante razonamiento encadenado. El selector, cuando el caso lo amerita, reduce el tamaño de la base de datos, donde se crea una sub-base para que la información irrelevante no interfiera. El refinador por último utiliza herramientas externas para la ejecución de SQL, recibe retroalimentación y refina las consultas que considere erróneas. Con GPT-4 como LLM principal para todas las tareas de los agentes y el uso de BIRD y Spider como conjuntos de datos para evaluar el desempeño del framework. Los datos experimentales muestran una precisión de ejecución de 59,59 en el conjunto de pruebas de retención de BIRD al utilizar MAC-SQL-GPT-4, esta es la mejor marca en el momento que se escribe el artículo. El artículo también habla de como afinaron un modelo basado en instrucciones, SQL-Llama, todo con el uso de datos de instrucciones de agentes de MAC-SQL, lo que ayuda en la simplificación de bases de datos, descomposición de preguntas, generación de SQL y corrección de SQL.

Además, en ([8]) se propone un enfoque que aprovecha múltiples prompts para generar diversas respuestas candidatas y las agrega de manera efectiva. El proceso de generación de SQL comprende tres pasos: Schema linking, Multiple SQL Generation y Selection. Schema linking implica identificar las tablas y columnas relevantes de una base de datos para convertir una pregunta en lenguaje natural en una consulta SQL. La introducción de schema linking ha mejorado significativamente el rendimiento tanto de los enfoques basados en ajuste fino como de los basados en ICL. Este paso consta de dos sub-pasos, extracción de tablas relacionadas con la pregunta en lenguaje natural (Table

Linking), para después extraer las columnas necesarias dentro de esas tablas (Column Linking). Se emplean múltiples prompts en ambas fases con el objetivo de lograr un alto nivel de recuperación. Multiple SQL Generation es generar varias consultas SQL basadas en múltiples prompts distintos. Se generan múltiples prompts donde varía tanto el método de selección de los ejemplos de few shot como el orden en que se presentan, lo que asegura una exploración más amplia de posibles consultas SQL, ya que se ha descubierto que hacer esto puede diferir considerablemente en la salida de la LLM. Selection buscar la consulta más precisa entre las consultas candidatas. El conjunto de candidatos se filtra en función de los puntajes de confianza, y luego se asigna al LLM la tarea de seleccionar la consulta más precisa entre el conjunto refinado.

En cuanto a técnicas de entrenamiento, destacan las GAN, o generative adversarial networks, esta técnica se basa en dos modelos que trabajan a la par, los cuales compiten entre ellos. Estos se tratan de un modelo, el generador, el cual construye información sintética en base a retroalimentación enviada por el otro modelo, el discriminador, este último recibe información real y sintética y decide de cual tipo se trata.

La meta detrás de este método es lograr que el discriminador tenga certeza respecto a cuál tipo de dato con el que interactúa, mientras que el generador busca engañar al discriminador. Por esto, esta técnica se destaca en cuanto a entrenamiento no supervisado o semi-supervisado sin requerir de grandes cantidades de datos de entrenamiento.

En cuanto a alternativas para esta configuración, en ([9]) se muestran diversas opciones, las cuales posteriormente se comparan, entre estas se encuentran FCGANs, GANs convolucionales, LAPGANs, DCGANs, GANs condicionales, infoGANs, y GANs con modelos de interferencia. Estos últimos utilizan dos tipos de redes, enconders (de interferencia) y decoders, ambas cumplen el rol del generador.

Además, se muestra que el entrenamiento se basa en encontrar los parámetros correctos del discriminador para hacer la clasificación mas precisa, y encontrar parámetros del generador para confundir al discriminador.

Por otro lado, se entrega información útil para el entrenamiento, como el orden en que los parámetros de cada modelo se deben actualizar, o que para cada generador hay

un discriminador óptimo.

Adicionalmente, en ([10]) se muestra un caso de uso específico de entrenamiento mediante GANs el cual utiliza modelos de lenguaje. En este trabajo se utilizan las facultades de procesamiento de los modelos de lenguaje para la tarea de recomendar trabajos a postulantes según el currículum que estos ingresan a una plataforma.

Se hace hincapié a la necesidad de extraer más información que la disponible explícitamente en los currículum, mediante el uso generador para refinar documentos de baja calidad a contra partes similares más detallas. Esto se realiza debido a la necesidad de los modelos de lenguaje de grandes cantidades de información con el fin de evitar delirios o resultados fabricados, una de las grandes falencias de los LLMs. Este documento también presenta en detalle el proceso usado para establecer la red, se explicitan parámetros del generador, discriminador y clasificadores, además del proceso de aprendizaje y la metodología de evaluación.

Finalmente, en ([11]) se muestra un proceso de optimización de consultas SQL generadas a partir de preguntas en lenguaje natural mediante LLMs, se entrega la premisa que el modelo genera diversas consultas SQL a partir de una entrada en NL, y que gran parte del tiempo la consulta seleccionada no es la que mejor se adecua a la pregunta de entrada.

Por esto, se establecen diferentes métricas las cuales son revisadas para cada potencial consulta SQL generada, las cuales una vez son comparadas, resultan en una mejora en el desempeño del modelo utilizado.

La generación de consultas SQL a partir de lenguaje natural ha alcanzado avances significativos gracias a la combinación de técnicas de procesamiento de lenguaje natural, aprendizaje automático y el uso de modelos de lenguaje avanzados. Sin embargo, a pesar de los logros obtenidos, se siguen enfrentando desafíos, como la necesidad de manejar de manera efectiva la especialización de consultas y bases de datos, la mejora en la precisión del aprendizaje automático y la adaptación a bases de datos complejas. Los enfoques actuales continúan evolucionando, integrando nuevas tecnologías y opti-

mizando los procesos para brindar soluciones más robustas y precisas en la generación automática de SQL.

Capítulo 4

Requisitos del sistema

4.1. Análisis de requerimientos

En la siguiente sección se muestran los diferentes requisitos del sistema, estos se subdividen en requisitos funcionales, no funcionales, de interfaces y de ambiente.

Al profundizar en estos, los requisitos funcionales explican las funcionalidades específicas que debe cumplir el sistema para satisfacer las necesidades del usuario y alinearse con los objetivos de esta memoria, sirviendo además como base para las pruebas y validaciones. Los requisitos no funcionales muestran las distintas características del producto desarrollado relacionadas con aspectos de calidad como desempeño, escalabilidad y seguridad. Los requisitos de interfaces dan una idea general de las interacciones que el usuario realiza con el sistema, promoviendo la interoperabilidad y reduciendo errores durante el desarrollo. Finalmente, los requisitos de ambiente proporcionan información referente a las herramientas y condiciones necesarias para la elaboración de la propuesta, garantizando compatibilidad y estabilidad en su entorno de operación.

4.1.1. Requisitos Funcionales

Para identificar los siguientes requisitos funcionales, se realiza un análisis del propósito que se desea cumplir con el desarrollo de esta memoria.

- RF1: El sistema debe recibir consultas en lenguaje natural y generar una consulta SQL equivalente.

  Para cumplir con facilitar el acceso a la información de personas que no tengan conocimientos de SQL, es esencial el primer paso, que es recibir las consultas de estas personas y traducirlas a una consulta SQL, la que permite tener una base para las siguientes funcionalidades.

- RF2: El sistema debe generar un documento que contenga la información obtenida a partir de la consulta SQL creada.

  Al tener un sistema que genere una consulta SQL basada en una consulta en lenguaje natural, se necesita que esta información sea recibida por el usuario, por lo que esta consulta se ejecuta y su contenido es almacenado en un archivo que es entregado al usuario.

- RF3: La plataforma debe tener manejo de sesiones.

  Para resguardar la seguridad de la información, es necesario que cada usuario que ingrese a la plataforma esté autenticado . Es por esto que cada usuario tiene su propia cuenta y un acceso controlado a la plataforma.

- RF4: El sistema debe almacenar y desplegar las interacciones entre los funcionarios y el chatbot.

  Es importante almacenar las interacciones de los modelos con los usuarios, para que el modelo tenga memoria y el usuario tenga control sobre las conversaciones a su gusto. Esto eleva la experiencia del usuario, y además, entrega herramientas que permiten analizar las conversaciones del chatbot, lo que permite que se puedan identificar puntos de mejora para el chatbot de manera oportuna.

- RF5: La plataforma debe ofrecer encuestas de satisfacción relacionadas con las conversaciones del chatbot, las cuales el usuario puede optar por responder.

Para entender el recibimiento de esta nueva herramienta por parte de los usuarios se ofrece una encuesta de satisfacción que permite al usuario calificar una conversación con el chatbot y responder preguntas relacionadas a su desempeño, lo que, en conjunto al requisito funcional 4 (RF4) permite identificar puntos de mejora y analizar las conversaciones del chatbot.

- RF6: La plataforma debe almacenar y desplegar encuestas de satisfacción y métricas del chatbot.

  La información recopilada en el requisito funcional 5 (RF5) es almacenada y desplegada para su posterior análisis.

- RF7: La plataforma debe tener control de privilegios de usuario y permitir a usuarios con permisos la administración de roles.

  En conjunto al requisito funcional 3 (RF3), para resguardar la seguridad de la información es necesario que cada usuario tenga los permisos necesarios para acceder a la información de la base de datos. Es por esto que cada usuario, a excepción de un usuario administrador, siempre estarán con permisos limitados y se le otorgarán los permisos manualmente por un administrador.

4.1.2. Requisitos no funcionales

Para asegurar una buena experiencia de usuario, se definen los siguientes requisitos no funcionales.

- RNF1: La plataforma debe ser segura, con el fin de proteger la información del cliente, con el uso de privilegios de usuarios e inicio de sesión para tener control sobre los usuarios que pueden ingresan a la plataforma. La API contará con credenciales para su uso.

- RNF2: El código debe ser escalable, hacer uso de buenas prácticas de programación y un formateo claro, con el fin de un desarrollo modular que facilite la interoperabilidad.

- RNF3: La plataforma debe tener tiempos de respuesta acordes a la tarea solicitada.

- RNF4: La plataforma debe ser de uso intuitivo para cualquier usuario. Esto se logrará mediante el diseño de una interfaz de usuario clara y sencilla, con una navegación intuitiva y accesible.

- RNF5: La plataforma debe ser mantenible. Esto se logrará mediante la utilización de prácticas de desarrollo de software como el control de versiones y la documentación clara.

4.1.3. Requisitos de interfaces

Se definen eventos principales en la plataforma que permiten el manejo y uso de sus funcionalidades.

Evento	Descripción	Iniciador	Parámetros	Respuesta
Consulta NL	El usuario ingresa una consulta en lenguaje natural con el objetivo de obtener una respuesta de la base de datos ya sea en formato CSV/XLSX y/o lenguaje natural.	Presionar tecla Enter.	Consulta en lenguaje natural.	Resultado consulta.
Inicio de sesión	El usuario ingresa sus credenciales en las casillas correspondientes para acceder a la plataforma.	Clic en botón ”Sign in” una vez llenados los campos.	Credenciales de ingreso.	Resultado inicio sesión.
Ingreso vista de registro de conversaciones	El usuario desea ingresar al registro de conversaciones.	Clic en botón de navegación ”Conversations”.	Tipo de usuario, nombre de vista.	Resultado cambio de vista
Ingreso vista de chatbot	El usuario desea ingresar a la vista de chatbot.	Clic en botón de navegación ”Home”.	Tipo de usuario, nombre de vista.	Resultado cambio de vista
Ingreso vista de métricas de satisfacción del cliente	El usuario desea ingresar a la vista de métricas de satisfacción del cliente.	Clic en botón de navegación ”Metrics”.	Tipo de usuario, nombre de vista.	Resultado cambio de vista
Cierre de Sesión	El usuario desea cerrar sesión.	Clic en botón ”Log out”.	N/A	Resultado cierre de sesión

Cuadro 4.1: Eventos externos

Respuesta	Descripción	Parámetros
Resultado consulta.	Se entrega resultado de consulta ingresada por usuario en el formato que este solicite.	Consulta NL.
Resultado inicio sesión.	Verifica las credenciales del usuario y si son válidas permite el ingreso a la plataforma.	Credenciales de acceso.
Resultado cambio de vista.	Verifica permisos del usuario. En caso de tener permiso, muestra la vista solicitada.	Tipo de usuario, nombre de vista.
Resultado cierre de sesión.	Cierra sesión del usuario.	N/A

Cuadro 4.2: Respuestas del sistema

4.1.4. Requisitos de ambiente

En esta sección, se detallan los componentes clave del ambiente de desarrollo, incluyendo lenguajes de programación, frameworks, bases de datos y servicios en la nube, así como las especificaciones técnicas mínimas recomendadas para los equipos que ejecutarán la aplicación. Con esto se asegura un desempeño óptimo y se evitan problemas de compatibilidad o rendimiento.

Hardware de desarrollo

Los computadores utilizados en el desarrollo tienen, como base, las siguientes especificaciones:

- OS Microsoft Windows 11 Home.
- Procesador Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz.
- Memoria física instalada (RAM) 8,0 GB.

Los requisitos mínimos necesarios para el desarrollo son:

- Procesador (CPU) de 64 bits.
- 8 GB de memoria RAM.
- Al menos 50 GB de espacio libre en el disco, para la descarga de frameworks, plataformas y el código fuente.
- Linux (Ubuntu 20.04 LTS o superior), macOS o Windows 10/11.
- Buena conexión a internet para interactuar con servicios en la nube.

Software de Desarrollo

Dentro del software de desarrollo utilizado tenemos las siguientes herramientas:

- Python: El lenguaje de programación utilizado para el desarrollo de este proyecto. Se utiliza principalmente por preferencia del cliente, debido a que la fundación Arturo López Pérez utiliza este lenguaje de programación en sus desarrollos, pero además, cuenta con una amplia variedad de bibliotecas y frameworks, que permiten la interacción con servicios de la nube y el desarrollo de frontend y backend, entre más funcionalidades.
- JupyterHub: Es el servidor de entornos de programación utilizado por el cliente y por el equipo para el desarrollo del proyecto. La información sensible del cliente era entregada mediante el servidor de JupyterHub.
- Amazon Bedrock (modelos fundacionales): Modelos de lenguaje de gran tamaño provistos por Amazon Web Services.
- CSS: Lenguaje de programación que permite controlar el aspecto del desarrollo de frontend.
- HTML: Lenguaje de programación utilizado para crear y controlar páginas web.

- FastAPI: Framework de python para crear aplicaciones web (APIs).
- Flask: Framework de python que permite crear aplicaciones web (APIs) y administrar las vistas de la aplicación (Fullstack).
- PostgreSQL: Sistema de bases de datos relacional. Se utiliza debido a que es el sistema de bases de datos utilizado por el cliente.
- Docker: Plataforma que permite administrar contenedores.
- Auth0: Plataforma que permite verificar la identidad de los usuarios antes de otorgarles acceso a aplicaciones y sitios web.
- Google API: Permite crear e interactuar con aplicaciones que se comunican con los servicios de Google, como Gmail

4.1.5. Perfiles de usuarios

En la tabla 4.3 se muestran los distintos perfiles de usuarios que utilizarán la plataforma.

Perfil	**Socio económico y cultural**	**Ocupacional**	**Etario**	**Características físicas, fisiológicas, psicológicas**	**Otros**
Administrador	Empleado(a) de FALP	Del área de inteligencia artificial.	25 - 65 años	Persona que tiene interés en el desempeño de la solución y una responsabilidad en la administración de la solución.	Usuario con más privilegios.
Analista	Empleado(a) de FALP	Parte del área de datos o inteligencia artificial.	25 - 65 años	Persona que tiene interés en el desempeño de la solución.	Usuario con privilegios.
Usuario	Empleado(a) de FALP	Cualquier ocupación.	25 - 65 años	Persona que desea emplear la solución.	Usuario sin privilegios.

Cuadro 4.3: Perfiles de usuario

Capítulo 5

Diseño del sistema

En este capítulo, se detalla el proceso del diseño del sistema, describiendo cómo se estructuran sus componentes para garantizar un funcionamiento eficiente y modular.

Se presentan los diagramas que ilustran la arquitectura general y el flujo de información dentro del sistema, proporcionando una visión clara de su estructura y comportamiento.

Asimismo, se realiza la definición de módulos, identificando las distintas partes que componen el sistema, sus responsabilidades y la manera en que interactúan entre sí.

5.1. Diagrama de contexto

En la figura 5.1 se ve el diagrama de contexto donde se ven las 3 entidades que interactúan con el sistema. En este, el usuario requiere información y la recibe del sistema, el LLM es quien hace el proceso de traducción de NL a SQL, y la base de datos, la cual entrega la data que se solicite.

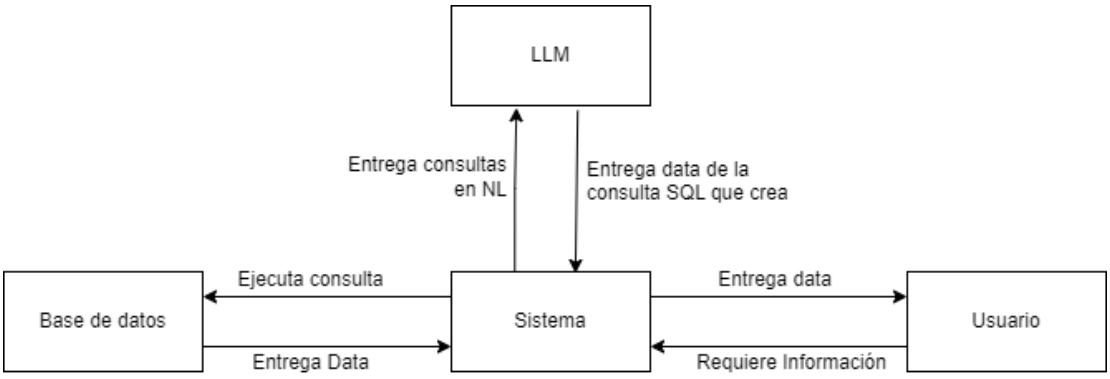


Figura 5.1: Diagrama de contexto

Para el diseño del diagrama de contexto, de forma más detallada, se toma el punto de vista de un usuario que quiere interactuar con el sistema. El usuario tiene el objetivo de obtener algo, en este caso, información que pueda resolver una pregunta, por lo que su interacción con el sistema es para el requerimiento de información, y el sistema responde con la información solicitada en el formato deseado por el usuario. Para poder obtener la información y entregarla al usuario, es necesario diseñar una consulta SQL, por lo que las consultas del usuario se entregan a los modelos de lenguaje (LLM) y la respuesta de los modelos es una consulta SQL. Para obtener la información de la base de datos, es necesario que el sistema esté constantemente interactuando con las bases de datos del cliente, por lo que las consultas SQL diseñadas por los LLM son ejecutadas en la base de datos y se obtiene la información requerida por el usuario, la cual es almacenada en el formato deseado por el usuario y entregada a él.

5.2. Diagrama de arquitectura

En la figura 5.2 se ve el diagrama de arquitectura con los distintos módulos internos del proyecto.

Para el diseño del diagrama de arquitectura, se comienza identificando los diferentes módulos que interactuan entre sí dentro del sistema. Comenzando por el módulo de frontend, que tiene dentro de su función principal ser el intermediario entre el usuario y

el sistema, mostrando las vistas al usuario y permitiendo su interacción con las distintas funcionalidades ofrecidas por la plataforma. El frontend se comunica directamente con el módulo backend, que es el intermediario entre la información de la base de datos, la api y el frontend. Se encarga de obtener y estructurar la información para ser consumida por el módulo del frontend. El módulo API es el encargado de toda la lógica del chatbot, la obtención de métricas y conversaciones. Interactua con los modelos de gran tamaño en amazon bedrock, y obtiene la información de la base de datos.

Todo esto se refleja en el diagrama, con las interacciones entre los módulos que permiten el funcionamiento del sistema.

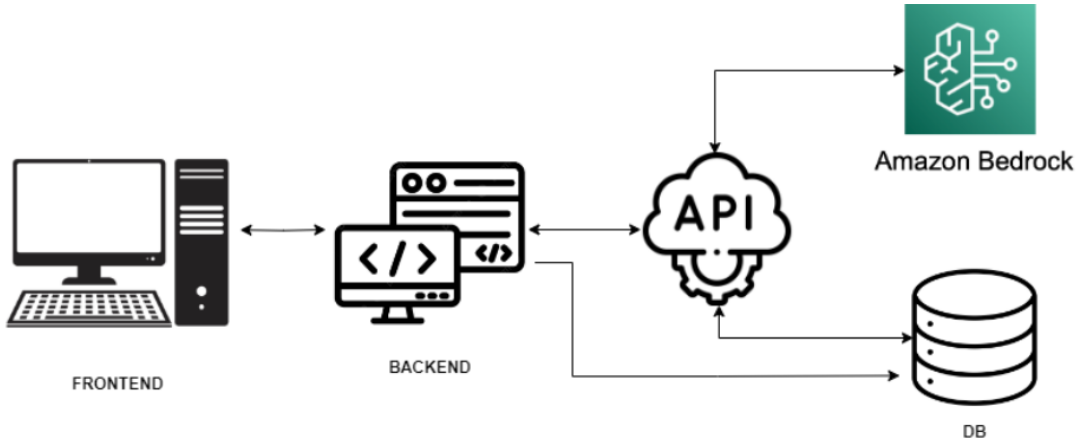


Figura 5.2: Diagrama de arquitectura

5.3. Enumeración de módulos

En la tabla 5.1 se definen los diferentes módulos descritos en la figura 5.2 con más detalle.

El sistema debe permitir a los usuarios interactuar con bases de datos de manera eficiente. Para esto, se divide en distintos módulos que cumplen funciones específicas.

El primer módulo es el de interacción, que corresponde al frontend de la plataforma. Este permite que los usuarios envíen consultas y reciban respuestas a través de una interfaz web. Sin embargo, el frontend no puede procesar estas solicitudes por sí solo,

por lo que necesita comunicarse con un backend, encargado de manejar la lógica de negocio y gestionar la interacción con el resto del sistema.

Para procesar y almacenar la información, el sistema incluye un módulo de base de datos, donde se guardan los datos y desde donde se extrae la información solicitada por los usuarios. Dado que el sistema debe interpretar consultas en lenguaje natural y traducirlas a SQL, se requiere un módulo de LLMs, que en este caso se implementa con Amazon Bedrock. Este módulo recibe las entradas del usuario, las convierte en consultas SQL y devuelve los resultados obtenidos de la base de datos.

Para estructurar mejor la comunicación entre los distintos componentes, toda la interacción con los LLMs y la base de datos se maneja a través de una API. Esto permite desacoplar la lógica del chatbot de la plataforma web, haciendo que el sistema sea más flexible y escalable. De esta forma, la API puede ser utilizada no solo por la plataforma, sino también por otras aplicaciones, como chatbots en Slack, WhatsApp y otros servicios.

Estos cinco módulos conforman el sistema y permiten su funcionamiento de manera modular y escalable.

Módulo	**Propósito**	**Sección**
Frontend	Módulo encargado de manejar las interfaces vistas por el usuario	Sección 5.4.1
Backend	Módulo encargado de manejar la lógica del sistema, recibe y devuelve data que llegue desde los módulos adyacentes	Sección 5.4.2
API	Módulo encargado de la interacción entre backend, LLMs y BBDD.	Sección 5.4.3
Modelos de lenguaje / Amazon Bedrock (LLMs)	Módulo encargado de la traducción de lenguaje natural a consulta SQL	Sección 5.4.4
Base de datos (BBDD)	Módulo que almacena los datos utilizados en las consultas, además de la información referente a la satisfacción con las mismas	Sección 5.4.5

Cuadro 5.1: Módulos y definiciones

5.4. Definición de módulos

5.4.1. Frontend

- **Propósito**: Módulo encargado de manejar las interfaces.

- **Alcance**: En este módulo el usuario ingresa la consulta, la cual es enviada al backend. Además, el frontend recibe información del backend para desplegar vistas dinámicas. Las cinco vistas principales de la plataforma son: inicio de sesión, vista de consulta, vista de satisfacción de usuario, vista de conversaciones y vista de administración. Se realizará en Python, y se hará uso de Flask, un framework flexible, el cual es adaptable a distintos tipos de aplicaciones web además de ser ligero y escalable.

- **Dependencias**: Únicamente dependiente del backend.

5.4.2. Backend

- **Propósito**: Módulo encargado de manejar la lógica del sistema, el cual recibe y retorna información recibida desde los módulos adyacentes.

- **Alcance**: Este módulo se encarga de recibir las distintas peticiones por parte del usuario a través del frontend, logra esto al hacer un llamado a la API o base de datos para obtener la data requerida. Su responsabilidad radica en el inicio de sesión, administración de usuarios, manejo de privilegios y solicitud de datos a la API. Se realizará en Python y también utilizará el framework Flask, de manera que este será utilizado tanto para el backend como el frontend.

- **Dependencias**: Entradas provenientes del frontend, conexión con la API.

5.4.3. API

Definición del módulo

- **Propósito**: Módulo encargado de la interacción entre backend, LLMs y BBDD.
- **Alcance**: Este módulo se encarga de interactuar de manera segura con las BBDD, mantener una conexión con las LLMs y mantener conexión con el backend, de forma que permite la interacción entre los tres módulos. Se estiman aproximadamente 100 conexiones simultáneas como máximo. Tendrá autenticación y límite de conexiones en un intervalo de tiempo. Se desarrollará en python y se utilizará el framework FastAPI, debido a que el trabajo se realizará en un entorno de JupyterLab entregado por el cliente. Este framework facilita la generación de documentación, permite un alto rendimiento y trabajo asíncrono, lo que permite escalar. Utiliza modelos de datos que son validados o convertidos en caso de no ser el tipo de dato requerido, tiene una documentación extensa y es flexible.
- **Dependencias**: Este módulo depende del buen desempeño de módulos como la LLM, BBDD y el backend.

5.4.4. Modelos de lenguaje / Amazon Bedrock (LLMs)

Definición del módulo

- **Propósito**: Este modulo se encarga de la lógica con la que se manipulan los mensajes del usuario. Identificando mensajes, generando consultas, corrigiendo código SQL, traduciendo información de la base de datos a lenguaje natural, generando archivos o gráficos y orientando al usuario a realizar preguntas sobre la base de datos del cliente. Para esto se utilizan modelos de la familia Claude los cuales según el tipo de tarea, trabajan con un prompt predefinido para resolver lo que les llegue como entrada ya sea del usuario o desde otro modelo. La estructura de comunicación de los modelos depende de las salida de los mismos,

donde tenemos salidas que se proyectan de inmediato al usuario, comunicación secuencial entre modelos o bucles con intentos definidos de manera interna. El enfoque de trabajo en desarrollo de prompt permite obtener una comparación en el desempeño de modelos frente a una misma tarea lo cual mantiene

- **Dependencias**: Este módulo depende de su interacción con la base de datos y con la API.

5.4.5. Base de datos (BBDD)

Definición del módulo

- **Propósito**: Módulo que almacena los datos utilizados en las consultas, además de la información referente a la satisfacción con las misma.
- **Alcance**: Este módulo se utiliza para la interacción con los modelos, para que estos conozcan la estructura de las tablas. Además, se le realizan las consultas generadas por los modelos de lenguaje. Se almacenará también la información de sesiones para la plataforma, el almacenamiento de conversaciones y de métricas. Se utilizará el gestor PostgreSQL para el almacenamiento e interacción con los datos.
- **Dependencias**: Este módulo no presenta dependencias.

Capítulo 6

Diseño de interfaces

6.1. Modelo de navegación

Respecto al modelo de navegación, se presenta la figura 6.1 que representa de forma visual cómo se navegará la plataforma. Se incluyen los principales requerimientos, los cuales son: la consulta de datos con el chatbot (RF1 y RF2), la consulta de registros de conversaciones (RF4) y la consulta de métricas de satisfacción de usuarios (RF5 y RF6).

El modelo de navegación se desarrolla junto con el prototipo 1.1, descrito con más detalle en la sección 6.2.1. El flujo comienza en el óvalo ”Inicio”, donde el primer paso siempre es iniciar sesión. Para ello, el usuario ingresa su correo electrónico y contraseña. Si las credenciales son validadas correctamente, se inicia la sesión y se accede a la navegación entre las distintas funcionalidades del sistema.

El sistema cuenta con cuatro funciones principales, además de la opción de cerrar sesión.

La primera es el chat, al cual se accede desde la barra de navegación. Una vez dentro, el usuario puede cargar una conversación anterior o iniciar una nueva. Luego, al enviar un mensaje, el sistema responde con un archivo, una opción o un mensaje de conversación. Tras recibir la respuesta, el usuario puede continuar la conversación o

iniciar otra, creando un flujo cíclico entre conversaciones nuevas y existentes.

La segunda funcionalidad es la sección de conversaciones, donde se pueden cargar y filtrar conversaciones previas para su análisis. De manera similar, la tercera funcionalidad, métricas, permite visualizar información relevante sobre las interacciones en el sistema.

Por último, la funcionalidad de administración del sistema permite gestionar usuarios, roles y permisos. Dentro de esta sección, es posible asignar roles a los usuarios, crear o eliminar roles y definir permisos asociados a cada uno. También se incluye la administración de permisos, que permite visualizar y gestionar los permisos existentes en el sistema.

Este modelo de navegación organiza el acceso a las distintas funciones del sistema, asegurando una interacción estructurada y eficiente.

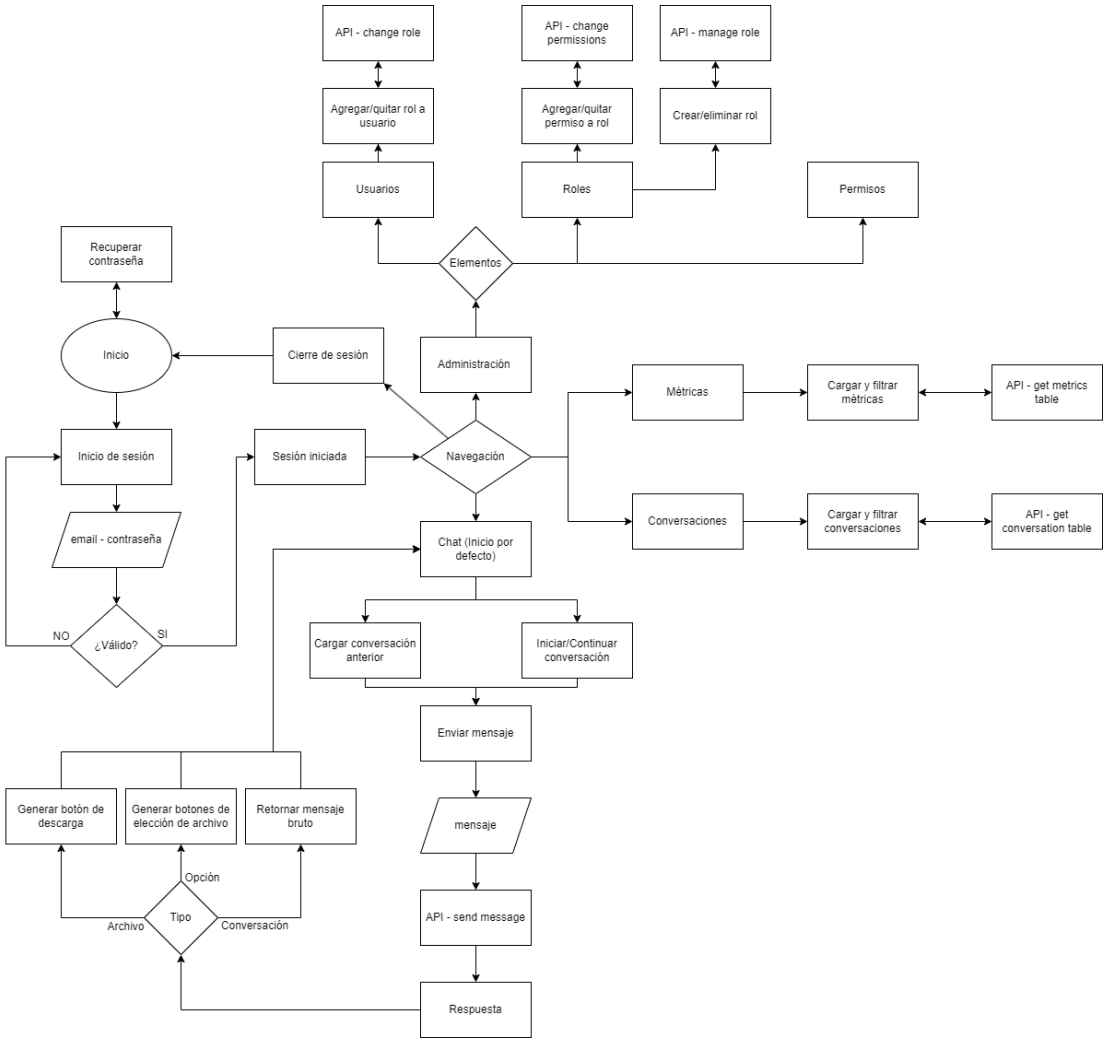


Figura 6.1: Modelo de navegación

6.2. Prototipo de interfaces usuarias

El Prototipo 1.1, descrito en la sección 6.2.1, representa la concepción inicial de la plataforma. Este modelo permitió materializar una primera versión funcional, sirviendo como base para el desarrollo posterior del Prototipo 1.2, detallado en la sección 6.2.2.

6.2.1. Prototipo 1.1

Figura 6.2: Inicio de sesión

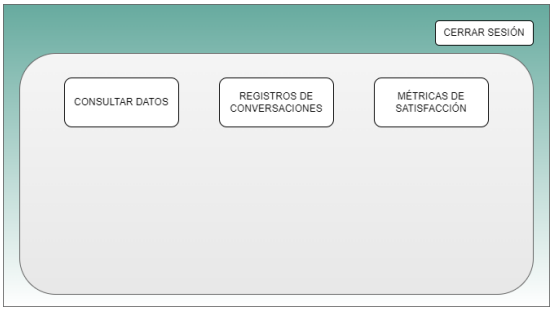


Figura 6.3: Menú principal

Figura 6.4: Registros de conversaciones

Figura 6.5: Métricas de satisfacción

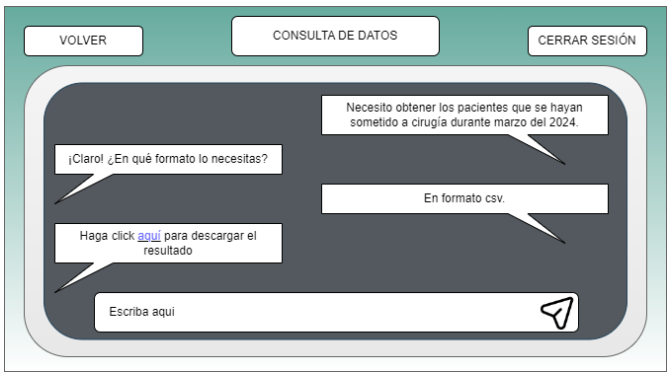


Figura 6.6: Consultar datos

En la tabla 8.3 se aprecian las principales interacciones con el prototipo de plataforma.

6.2.2. Prototipo 1.2

Para la siguiente fase del diseño, se tienen las siguientes vistas de la aplicación web. La primera es la vista de inicio de sesión, donde el usuario ingresa sus credenciales para acceder a la plataforma. Luego, está la vista del menú principal, que permite seleccionar las diferentes funciones disponibles, incluyendo la consulta de datos a través del chatbot.

También se incluye la vista de registros de conversaciones, que muestra una tabla con información detallada sobre las interacciones previas con el chatbot. Finalmente, se encuentra la vista de métricas de satisfacción del usuario, que presenta una tabla con datos sobre la evaluación de la experiencia en la plataforma.

En el siguiente enlace [1] se puede acceder al prototipo interactivo.

[1] https://tinyurl.com/yetfjw89

Evento	Interacción	Acción	Objeto afectado
Iniciar sesión	Apretar botón.	Se dirige a vista principal.	Botón ”Sign In”
Recuperar contraseña	Apretar botón.	Se dirige a vista de ayuda para recuperar contraseña.	Botón ”Forgot Password?”
Cerrar sesión	Apretar botón.	Se dirige a vista de inicio de sesión.	Botón ”Log out”
Consultar datos	Apretar botón.	Se dirige a vista de consulta de datos.	Botón ”Home’
Consultar registros de conversaciones	Apretar botón.	Se dirige a vista de consulta de registros de conversaciones.	Botón ”Conversations”
Consultar métricas de satisfacción de usuarios	Apretar botón.	Se dirige a vista de consulta de métricas de satisfacción de usuarios.	Botón ”Metrics”
Filtrar datos	Apretar botón.	Se filtran resultados de consulta de registros de conversaciones o de consulta de métricas de satisfacción de usuarios.	Botón ”Filtrar”

Cuadro 6.1: Eventos prototipo y definiciones

Capítulo 7

Desarrollo de la solución

7.1. Hitos principales del proyecto

El desarrollo del proyecto se estructuró en cuatro hitos clave que permitieron alcanzar los resultados actuales. Primero, en la definición del problema, se profundizó en la comprensión del problema central y se propuso una solución preliminar. Durante esta fase, se realizaron reuniones con los clientes para recopilar información y así sentar las bases de un enfoque viable.

En el segundo hito, diseño del prototipo, se desarrolló un esquema inicial de la solución, enfocado en crear una plataforma con un chatbot y tablas para el almacenamiento de información, incluyendo una tabla de conversaciones y otra de métricas. Esta etapa fue fundamental para concretar un diseño visual y funcional inicial de la plataforma.

Luego, el desarrollo de la solución permitió expandir las funcionalidades del prototipo inicial. Se incorporaron características adicionales que no estaban contempladas en el diseño original, como la vista y lógica de administración de la plataforma, la generación de gráficos y otros detalles funcionales menores. Este hito enriqueció significativamente la propuesta inicial.

Finalmente, en el cuarto hito se consolidó un prototipo funcional de la plataforma. En esta etapa, todas las funcionalidades planificadas están completas, todos los módulos

se encuentran operativos y las vistas se construyeron con el propósito de hacer una plataforma eficaz y con una alta usabilidad.

7.2. Revisión de requerimientos

En esta sección, se detallan los ajustes realizados a los requerimientos funcionales y no funcionales de la plataforma, desde su planteamiento inicial hasta el estado actual. Durante el desarrollo, algunos requerimientos fueron modificados para reflejar mejor los objetivos y asegurar que la plataforma funcione de manera eficiente y que sea de fácil adopción para los usuarios.

En cuanto a los requerimientos funcionales, el primer cambio se centra en la conversión de consultas en lenguaje natural a SQL. Inicialmente, la plataforma debía convertir las consultas en SQL, pero ahora se especifica que el sistema debe recibir estas consultas en lenguaje natural y generar su equivalente SQL. También se actualizó el requerimiento de generación de respuestas, el cual en lugar de solo ofrecer respuestas congruentes, ahora el también debe generar un documento con la información obtenida.

El siguiente requisito que evoluciona es el manejo de sesiones, este pasó de un simple inicio de sesión a una gestión detallada de sesiones que ofrece una mayor seguridad y personalización. Además, se redefinió el almacenamiento de interacciones para enfocarse en el chatbot en lugar de en el modelo de lenguaje general, alineando mejor el requerimiento con el uso previsto del chatbot en la plataforma. Por otro lado, las encuestas de satisfacción se actualizaron para que los usuarios puedan responderlas voluntariamente. Finalmente, se añadió un nuevo requerimiento de control de privilegios y administración de roles, permitiendo a usuarios con permisos específicos gestionar roles y aumentando la seguridad y flexibilidad del sistema.

Para los requerimientos no funcionales, se reforzó la seguridad del sistema, manteniendo la autenticación y el control de privilegios para proteger la información del cliente. La escalabilidad también sigue siendo esencial, con un código modular y estructurado. En cuanto a los tiempos de respuesta, estos pasan de no ser más largos que

un minuto a ser acordes a la complejidad de cada tarea.

Respecto a la interfaz de usuario, se simplificó el requerimiento para enfocarse en una navegación intuitiva, eliminando la mención explícita de pruebas de usabilidad. El requerimiento de alta disponibilidad fue eliminado, dándose por entendido en el diseño, mientras que se puso un mayor énfasis en la mantenibilidad del sistema mediante el control de versiones y una documentación clara, esenciales para una gestión eficiente.

7.3. Desarrollo de la solución

7.3.1. Desarrollo de la API

Endpoints

A continuación se presenta la tabla 7.1 que resume los endpoints desarrollados para la API.

En la funcionalidad de chat, el endpoint ”send message” permite interactuar con el chatbot enviando mensajes y recibiendo respuestas. Para obtener todas las conversaciones de un usuario, se dispone del endpoint ”get conversations”, mientras que ”get conversation messages” recupera los mensajes de una conversación específica. Además, ”change conversation name” permite modificar el nombre de una conversación y ”get conversations table” proporciona un resumen de las conversaciones con datos relevantes, soportando paginación y ordenamiento.

En cuanto a la gestión de archivos, el endpoint ”download” permite descargar archivos generados en una conversación, mientras que ”check” verifica su existencia en la base de datos.

Por otro lado, en la funcionalidad de métricas, el endpoint ”send” permite registrar las respuestas de una encuesta de satisfacción asociada a una conversación, y ”get table” proporciona un resumen de las métricas almacenadas, permitiendo paginación y ordenamiento de los datos.

Todos los endpoints requieren autenticación mediante OAuth2.0 para garantizar la

seguridad del acceso a la plataforma.

Blueprint	Endpoint	Descripción	Auth	Cuerpo	Respuesta	Método
	/ping/	Permite probar la conexión a la API.	SI		pong	GET
chat	/sendMessage/	Permite enviar mensajes al chatbot y recibir la respuesta.	SI	Se debe enviar el id de conversación (conversation_id), el id del usuario (user_id) y el mensaje (prompt).	Responde con un json que contiene response, response.text, response.file_id (si aplica), response.file_type (si aplica) y conversation_id.	POST
	/getConversations/	Permite obtener todas las conversaciones de un usuario específico.	SI	Se debe enviar el id del usuario (user_id).	Responde con una lista de diccionarios que contienen el id de la conversación (id), el nombre (name), cuando fue creada (created_at) y si ha sido calificada (qualified).	GET
	/getConversationMessages/	Permite obtener todos los mensajes de una conversación.	SI	Se debe enviar el id de la conversación (conversation_id)	Responde con una lista de diccionarios que contienen el rol de quien envió cada mensaje (role entre user y assistant) y el contenido del mensaje.	GET
	/changeConversationName/	Permite cambiar el nombre de una conversación.	SI	Se debe enviar el id de la conversación (conversation_id) y el nuevo nombre (name) de la conversación.	Responde con una variable llamada changed indicando si fue o no fue cambiado el nombre (True/False).	POST
	/getConversationTable/	Permite obtener la información para la tabla de conversaciones.	SI	Se debe enviar un valor para limitar la cantidad de datos (limit). Si no se envía, por defecto es 10. Se debe enviar un valor de paginación (offset) que por defecto es 0, la columna por la cual vamos a ordenar los datos (order_by), que por defecto es conversation_id, y el sentido del orden (order_way) que por defecto es descendente.	Retorna un diccionario con el total de datos (total), el valor del siguiente offset (next_offset) y data, que contiene una lista de diccionarios que tienen el id de la conversación, el id del usuario, el mensaje inicial y la consulta generada.	GET
files	/download/	Permite descargar un archivo.	SI	Se debe enviar el id del archivo (file_id).	Descarga el archivo en el dispositivo.	GET
	/check/	Permite verificar la existencia de un archivo.	SI	Se debe enviar el id del archivo (file_id).	Responde con un booleano (True/False) referente a la existencia del archivo.	GET
metrics	/send/	Permite enviar una encuesta de satisfacción respondida por un usuario.	SI	Se debe enviar el id de conversación (conversation_id), la calificación otorgada (qualification) y las respuestas (questions).	N/A	POST
	/getTable/	Permite obtener la tabla con información importante de cada calificación.	SI	Se debe enviar un valor para limitar la cantidad de datos (limit). Si no se envía, por defecto es 10. Se debe enviar un valor de paginación (offset) que por defecto es 0, la columna por la cual vamos a ordenar los datos (order_by), que por defecto es metric_id, y el sentido del orden (order_way) que por defecto es descendente.	Retorna un diccionario con el total de datos (total), el valor del siguiente offset (next_offset) y data, que contiene una lista de diccionarios que tienen el id de la métrica, el id de la conversación, el id del usuario, la puntuación dada (del uno al cinco), las preguntas respondidas por el usuario y las métricas obtenidas en el momento de la encuesta de satisfacción.	GET

Cuadro 7.1: Endpoints API

Seguridad de la API

La API implementa el protocolo OAuth 2.0 utilizando Auth0 como proveedor de autenticación, asegurando un control seguro y eficiente del acceso. Este esquema es particularmente adecuado para escenarios de comunicación entre máquinas (*Machine-to-Machine*, M2M). A continuación, se detallan los puntos clave:

1. **Elección de OAuth 2.0 y Auth0**:
    - OAuth 2.0 es un estándar ampliamente utilizado para la autorización segura.
    - Auth0 actúa como un servidor de autorización, simplificando la gestión de autenticación y autorización.
2. **Implementación en FastAPI**:
    - Se configura un cliente en Auth0 con un `client_id` y un `client_secret`.
    - Estas credenciales se almacenan como variables de entorno para mantener la seguridad.
    - El backend utiliza las credenciales para obtener un token de acceso que se utiliza en las solicitudes a la API.
3. **Flujo de Autenticación**:
    a) El cliente solicita un token de acceso al servidor de autorización de Auth0 utilizando sus credenciales.
    b) Auth0 valida las credenciales y emite un token de acceso.
    c) El cliente incluye este token en las solicitudes a la API.
    d) La API valida el token antes de procesar la solicitud.
4. **Validación de Tokens en FastAPI**:
    - Se utiliza la biblioteca `authlib` para decodificar y validar tokens JWT emitidos por Auth0.

- La función `verify_token` asegura que el `aud` (audiencia) y el `iss` (emisor) coincidan con los valores esperados.
- El código de implementación se presenta en la figura 24.

7.3.2. Desarrollo del Backend y Frontend

Control de acceso

Se ha implementado un sistema de control de acceso basado en permisos, utilizando decoradores para gestionar las restricciones de acceso a las distintas funcionalidades y vistas de la aplicación. A continuación, se describen los principales avances realizados.

Control de Permisos Se añadió un decorador denominado `permissions_required`, que permite verificar los permisos asignados a un rol antes de acceder a un endpoint. Este decorador evalúa los permisos asociados al rol del usuario y controla la visibilidad de las funcionalidades principales. La implementación se presenta en la figura 25.

Uso del Decorador: El decorador `permissions_required` se utiliza para personalizar las restricciones de acceso dependiendo de los permisos asignados a un rol. Por ejemplo:

- *Para administración*:

```
@permissions_required(permissions_list=[2, 3, 6, 7], main_view=True)
    def admin_view():
        ...
```

Figura 7.1: Backend y Frontend - permisos administración

- *Para el chatbot*:

```
@permissions_required(permissions_list=[1])
    def admin_view():
        ...
```

Figura 7.2: Backend y Frontend - permisos chatbot

La diferencia principal radica en los permisos que se envían a la función, permitiendo controlar dinámicamente el acceso a distintas funcionalidades.

Control de Sesión Se añadió el decorador `login_required` para verificar que el usuario haya iniciado sesión antes de acceder a un endpoint. La implementación es la siguiente:

```
def login_required(f):
        @wraps(f)
        def login_review(*args, **kwargs):
            if 'user_name' not in session or 'user_role' not in session:
                return redirect(url_for('auth.index'))
            return f(*args, **kwargs)
        return login_review
```

Figura 7.3: Backend y Frontend - requerimiento de sesión

Uso del Decorador: Este decorador se aplica a los endpoints que requieren autenticación.

Comportamiento del Sistema

- Si un usuario intenta acceder a un endpoint sin tener los permisos requeridos, será redirigido automáticamente a la vista de inicio con un mensaje de error:

Figura 7.4: Backend y Frontend - Ejemplo de intento de ingreso sin permisos

- Si un usuario no ha iniciado sesión, el sistema lo redirigirá a la página de inicio de sesión.

```python
@app.route('/admin', methods=['GET'])
    @login_required
    @permissions_required(permissions_list=[2, 3, 6, 7], main_view=True)
    def admin_dashboard():
        return render_template('admin_dashboard.html')
```

Figura 7.5: Backend y Frontend - Ejemplo Integrado de Control de Acceso

Ejemplo Integrado En este caso, solo los usuarios con los permisos adecuados podrán acceder al panel de administración. Si no cumplen con los requisitos, serán redirigidos a la vista correspondiente.

Inicio de Sesión y Registro

Para gestionar el inicio de sesión y registro, se diseñaron dos templates separados:

- **Template de Inicio de Sesión**
- **Template de Registro**

Inicio de Sesión:

Cuando el usuario ingresa sus datos y hace clic en `Iniciar sesión`.

Funcionamiento en la Práctica:

- *Caso de ingreso incorrecto:*

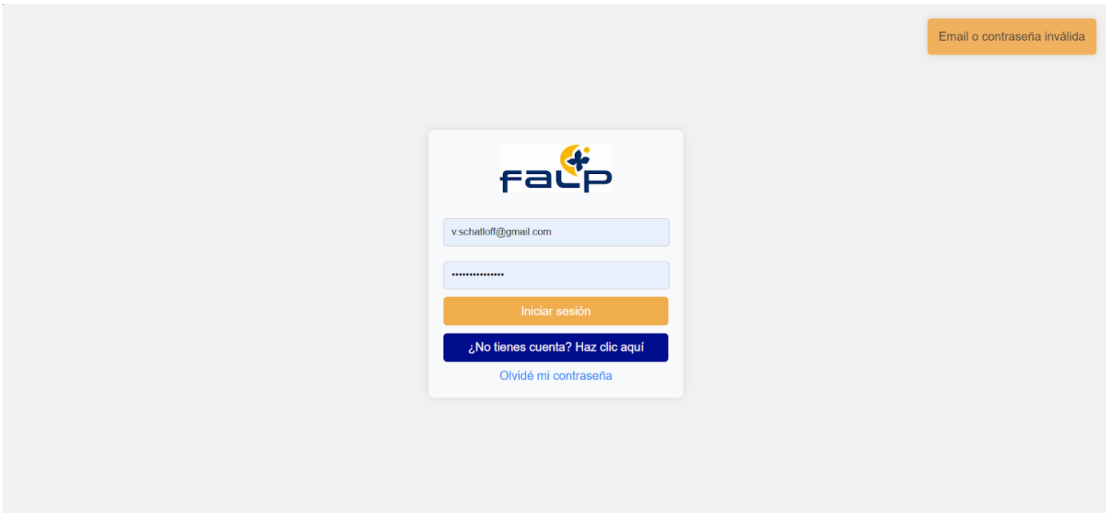


Figura 7.6: Backend y Frontend - Inicio de sesión fallido

- *Caso de ingreso correcto:* Ingresaría a la vista principal.

Registro:

Cuando el usuario selecciona `Me quiero registrar`.

Funcionamiento en la Práctica:

- *Caso de contraseñas que no coinciden:*

Figura 7.7: Backend y Frontend - error en registro

- *Caso de valores vacíos:* Aparece sobre el formulario de registro una notificación indicando que debe completar los valores vacíos.

- *Caso de registro correcto:*

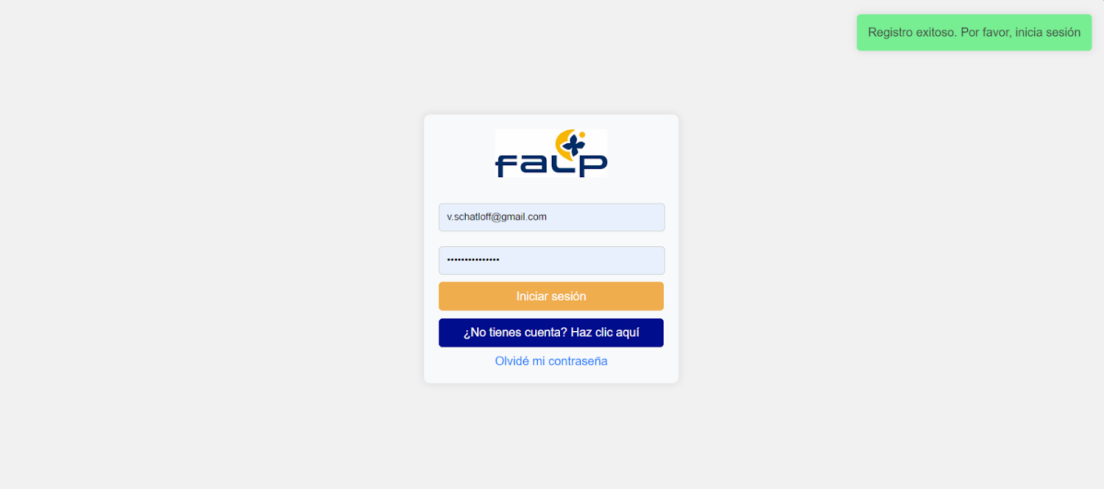


Figura 7.8: Backend y Frontend - registro exitoso

Datos Registrados en la Base de Datos:

Ejemplo de un usuario registrado:

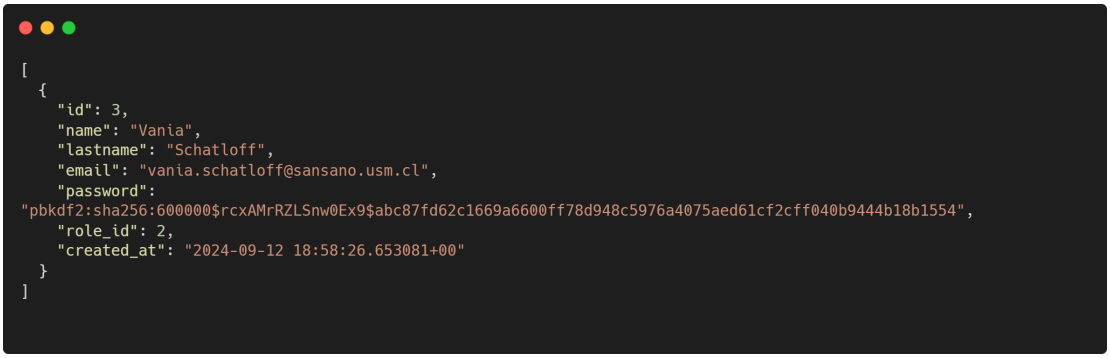


Figura 7.9: Backend y Frontend - Ejemplo de Datos en la Base de Datos

Roles:

El campo `role_id` se asocia con roles predefinidos, descritos en la siguiente tabla:

id	role_name
1	Administrador
2	No asignado

Cuadro 7.2: Roles en la Aplicación

Los roles por defecto son el 1 y el 2, dando libertad a los usuarios administradores de crear y administrar los roles, sus nombres y permisos.

Cierre de Sesión:

Al hacer clic en ”Cerrar sesión” en cualquiera de las vistas.

Página principal

La vista principal de la plataforma incluye diferentes experiencias según los permisos del usuario. Las vistas específicas son:

- **Usuario con todos los permisos:** En esta vista, el usuario tiene acceso completo a todas las funcionalidades de la plataforma. Se muestra un mensaje de bienvenida que introduce al usuario a la plataforma y explica brevemente su propósito, indicando que puede interactuar con bases de datos a través del chatbot.

Figura 7.10: Backend y Frontend - Vista principal de usuario con todos los permisos

- **Usuario con algunos permisos:** La vista es similar a la del usuario con todos los permisos, pero en este caso, se especifica el rol del usuario actual, lo que define

qué funciones puede utilizar dentro de la plataforma.

- **Usuario no asignado:** En la vista principal, se mantiene el mismo mensaje informativo de las vistas anteriores, pero se añade una sección adicional que indica al usuario que aún no tiene un rol asignado y que debe esperar a que un administrador le asigne uno para poder acceder a las funcionalidades.

Lo que se muestra bajo la barra de navegación varía dependiendo de la funcionalidad que utiliza el usuario. Al iniciar sesión, el usuario siempre aterriza en la vista de **Inicio**.

Código de la barra de navegación La barra de navegación permite la redirección a diferentes funcionalidades dependiendo de los permisos del usuario.

Cuando el usuario hace clic en el botón **Cerrar sesión** de la barra de navegación, se redirige al usuario a la vista de inicio de sesión con un mensaje de confirmación.

Redirección en la barra de navegación Dependiendo de los permisos otorgados, los enlaces de la barra de navegación pueden redirigir a distintas funcionalidades de la plataforma:

- **Inicio:** Página principal accesible para todos los usuarios.
- **Chat:** Funcionalidad de chatbot, disponible solo si el usuario tiene acceso habilitado (`chat`).
- **Conversaciones:** Sección donde se visualizan conversaciones previas, habilitada con el permiso `conversations`.
- **Métricas:** Vista de análisis y métricas, disponible si el permiso `metrics` está habilitado.
- **Administración:** Gestión de usuarios y configuraciones administrativas, habilitada solo para usuarios con permiso de administración (`admin`).

Estas redirecciones garantizan que los usuarios accedan solo a las funcionalidades correspondientes a su nivel de permisos.

Chat

Se diseñó una vista específica para el **chatbot**, que incluye las siguientes características:

- Un chatbot interactivo en el área principal.
- Un botón de menú en la esquina superior, el cual, al ser presionado, despliega un menú con las conversaciones anteriores.

A continuación, se describen las funciones principales que permiten el funcionamiento del chatbot y el manejo de conversaciones.

Funcionamiento del Chatbot Cada vez que el usuario hace clic en el botón **Enviar**, se ejecuta la función `sendMessage`. Esta realiza una llamada a la API enviando el mensaje ingresado. Cuando la API devuelve la respuesta, la función:

- Llama a `loadMessages`, que se encarga de obtener los mensajes correspondientes a la conversación actual.
- Llama a `updateChatMessages`, que carga los mensajes en la página para que el usuario los visualice.

Este flujo asegura que los mensajes enviados y recibidos se procesen y muestren dinámicamente en la interfaz.

Carga de conversaciones Para gestionar las conversaciones anteriores, el sistema utiliza el siguiente proceso:

- El botón de menú en la esquina superior despliega una lista de conversaciones almacenadas.

- El usuario puede seleccionar una conversación específica para cargarla en la vista principal del chatbot.

Opción de calificación de conversaciones Además, se implementó una funcionalidad para que los usuarios puedan calificar las conversaciones. Esto se realiza de la siguiente manera:

1. En el menú de conversaciones, aparece una estrella junto a cada conversación.
2. Al hacer clic en la estrella, se abre un **modal** con un conjunto de preguntas para calificar la conversación.
3. Si el usuario responde las preguntas, la estrella aparece marcada para indicar que la conversación ha sido calificada.

Este sistema de calificación permite recopilar retroalimentación sobre las conversaciones y mejorar la experiencia del usuario.

En la sección 8.3.7 se detallan todas las vistas relacionadas con lo explicado.

Conversaciones

La sección de **Conversaciones** permite gestionar y visualizar todas las interacciones almacenadas en el sistema.

Vista de conversaciones Se presenta una vista que incluye elementos de paginación y filtros para facilitar la navegación y exploración de los datos. En esta vista, los usuarios pueden:

- **Filtrar**: Aplicar filtros personalizados para visualizar subconjuntos específicos de las conversaciones.
- **Ordenar**: Cambiar el orden de las conversaciones en base a diferentes criterios (por ejemplo, por ID de conversación de manera descendente).

- **Navegar**: Utilizar la paginación para explorar distintas páginas de datos con límites configurables (por ejemplo, un límite de 5 datos por página).

Endpoint para la tabla de conversaciones Para generar la tabla de conversaciones, se define un formato preliminar que incluye los siguientes campos:

- **ID de conversación**: Identificador único de la conversación.
- **ID de usuario**: Identificador único del usuario asociado a la conversación.
- **Primer mensaje**: El primer mensaje que inició la conversación.
- **Consulta generada**: La consulta generada a partir de la interacción del usuario.

Métricas

Vista de ingreso de métricas

Al no poder identificar un momento final en una conversación, se decide dejar la calificación como opcional al lado de la vista de conversaciones del usuario, como se aprecia en la figura 8.3.

Las estrellas oscuras indican que la conversación ya fue calificada. Si el usuario intenta calificar una conversación ya calificada, aparece el siguiente mensaje:

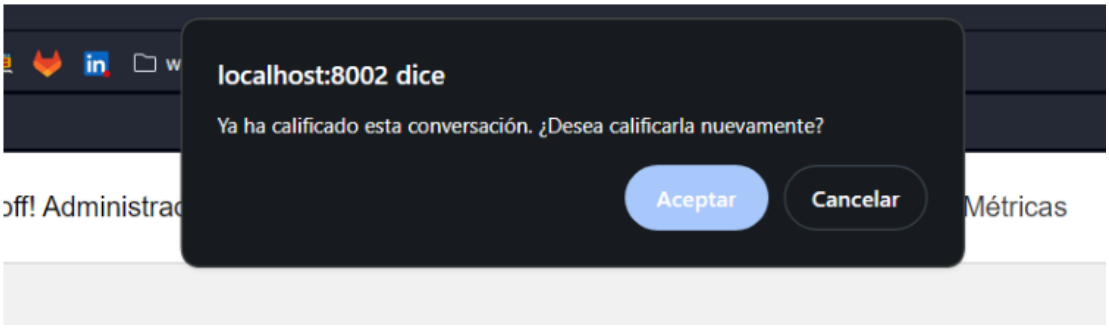


Figura 7.11: Mensaje al intentar calificar una conversación ya calificada

Si el usuario acepta, se carga el modal de calificación. En caso contrario, este paso se omite. Para conversaciones no calificadas, se permite directamente la interacción con el modal.

El usuario debe responder con una calificación y proporcionar al menos una respuesta a las siguientes 6 preguntas:

1. ¿El chatbot respondió adecuadamente a tus preguntas?
2. ¿Cómo calificarías la velocidad de las respuestas del chatbot?
3. ¿Consideras que el chatbot debería ser más conciso en sus respuestas?
4. ¿Hubo algún tema con el que el chatbot tuvo dificultades para proporcionar información? Si es así, ¿cuál fue?
5. ¿Qué funcionalidad adicional te gustaría agregar para mejorar las conversaciones con el chatbot?
6. Comentarios adicionales.

El usuario puede seleccionar las estrellas para calificar:

Figura 7.12: Interfaz para calificar conversaciones

Al hacer clic en enviar, los datos se envían a la API y se almacenan en la tabla `metrics`. Además, la conversación es marcada como calificada.

Vista de métricas

Se implementa la vista en la figura 8.5 para visualizar las métricas:

La vista cuenta con elementos de paginación y filtros. Los usuarios pueden filtrar, ordenar y navegar entre las páginas de datos. Por ejemplo, en la imagen superior se filtran las métricas con un límite de 5 datos, ordenadas por `id de métrica` en orden ascendente.

Administración

En la sección de **Administración**, se pueden gestionar usuarios, roles y permisos dentro de la plataforma. A continuación, se describen las funcionalidades principales:

Administración de usuarios La vista de administración de usuarios permite asignar roles a los usuarios. Al seleccionar un usuario, se despliega un modal que permite buscar y asignar un rol específico mediante un buscador. Una vez seleccionado el rol deseado, este se asigna correctamente al usuario.

Administración de roles Para acceder a la administración de roles, se navega desde un menú lateral (sidebar) disponible en la sección de Administración. Desde esta vista, se puede realizar lo siguiente:

- **Crear un rol:** Haciendo clic en el botón correspondiente, se despliega un formulario para definir los detalles del nuevo rol.
- **Actualizar un rol:** Se puede modificar la información de un rol existente.
- **Eliminar un rol:** Al intentar eliminar un rol, este no se elimina completamente de la base de datos, sino que se marca como eliminado mediante la columna `deleted`.

En la interfaz, los botones están codificados por colores: azul para editar un rol y rojo para eliminarlo.

Administración de permisos En esta sección, únicamente se permite visualizar los permisos disponibles en la plataforma, sin posibilidad de editarlos.

7.4. Revisión del diseño

El diseño de la aplicación mantiene su núcleo y objetivo, aunque ha recibido cambios tanto en como se procesa un mensaje por medio de los modelos, así como también

se han añadido características al diseño de la pagina web.

Por el lado del modulo de LLM, se ha realizado un flujo distinto al planeado en un comienzo, principalmente para poder cubrir los casos donde se obtiene una respuesta distinta a la esperada. Se añaden identificadores de mensaje y correctores de lenguaje SQL para cumplir con la tarea.

Por el lado de las vistas, se ha añadido una barra lateral que permite ver los mensajes anteriores dentro de una misma sesión. También se añade una ventana de administración que aun esta en desarrollo, pero tiene como objetivo ver los permisos de los distintos usuarios para el acceso de métricas y de conversaciones. Estos cambios se evidencian en la imagen 7.13.

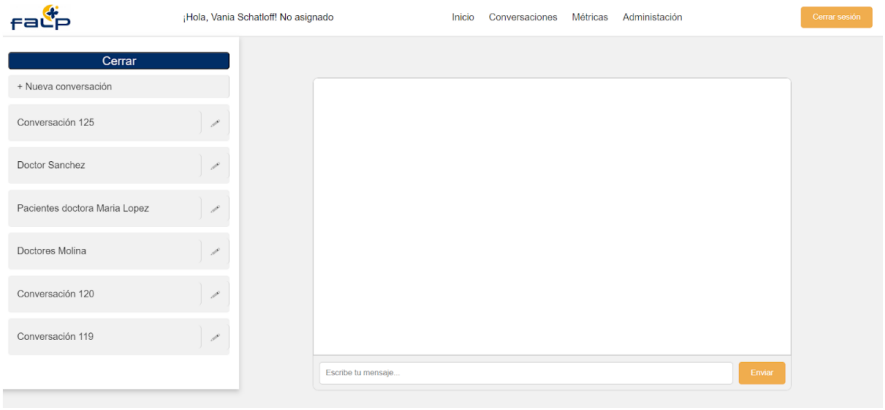


Figura 7.13: Vista principal con barra lateral y sección de administrador

Como otra modificación al diseño de la interfaz, se tienen cambios realizados a la navegación del sistema, los cuales se evidencian en la figura 6.1, principalmente, se evidencian las diferentes funcionalidades del chatbot y las interacciones con la API.

Capítulo 8

Revisión del prototipo

8.1. Contexto general del prototipo

La aplicación busca poder tener conversaciones humano-sistema donde estas se pueden almacenar. Su objetivo es tener un sistema de obtención de información eficiente que evite la necesidad de utilizar horas hombre en traducir consultas manualmente.

8.2. Descripción de componentes

1. Barra de navegación

   General: Permite el cambio entre vistas, así como cerrar sesión y dar la bienvenida al usuario.

   Interacción: El usuario puede hacer clic en distintos botones que lo dirigirán a las funcionalidades.

   Funciones: Permitir el cambio de vistas de manera sencilla.

2. Botones

   General: Permite la interacción del usuario con algunas funciones del sistema.

   Interacción: El usuario hace clic en el componente.

Funciones: Cada botón tiene una función en particular, como iniciar sesión, cerrar sesión, enviar mensaje, cambiar de vista, abrir nueva conversación, etc.

3. Barra lateral

   General: Permite ver y acceder a conversaciones anteriores de un mismo usuario con el modelo.

   Interacción: El usuario hace clic en un botón para que se muestre el componente, de esto se puede hacer clic en las conversaciones anteriores del usuario.

   Funciones: Tener acceso a las conversaciones antiguas del usuario.

4. Input para chat

   General: Es un formulario que actúa como input para chat. Permite al usuario escribir mensajes.

   Interacción: El usuario escribe un mensaje luego de hacer clic en el input.

   Funciones: Permite al usuario escribir un mensaje y enviarlo al chatbot.

5. Módulo para mensajes de chat

   General: Permite visualizar los mensajes anteriores en la conversación.

   Interacción: El usuario puede ver los mensajes que se han tenido durante la conversación con el sistema.

   Funciones: Mostrar los mensajes entre el chatbot y el usuario de una conversación específica.

6. Tabla de conversaciones

   General: Una tabla de conversaciones que muestra los atributos más importantes de cada conversación, como el primer mensaje, la consulta generada, entre otros datos.

Interacción: El usuario puede ver los datos y filtrarlos.

Funciones: Ver los datos más importantes de las conversaciones.

7. Paginación

   General: Permite navegar entre una gran cantidad de datos.

   Interacción: El usuario puede hacer clic en distintos botones.

   Funciones: Permite cambiar la página actual de los datos.

8. Filtros

   General: Permite el filtrado información.

   Interacción: El usuario indica un filtro y genera una llamada distinta desde el backend a la API.

   Funciones: Filtrar información según el usuario lo indique.

9. Tabla de Métricas

   General: Una tabla de métricas que muestra las métricas de cada conversación, como la satisfacción del usuario, comentarios, entre otros datos.

   Interacción: El usuario puede ver los datos y filtrarlos.

   Funciones: Ver los datos de las métricas.

10. Sección para gestión de permisos

    General: Vista que permite al usuario administrador gestionar permisos.

    Interacción: El usuario puede seleccionar usuarios o roles y administrarlos al hacer clic en distintas opciones y botones.

    Funciones: Permite cambiar los permisos de los usuarios y roles.

11. Cuadro de ingreso de credenciales

General: Es un formulario que recibe los datos para el inicio de sesión en la plataforma.

Interacción: El usuario rellena el formulario con sus credenciales para acceder a la plataforma.

Funciones: Permitir al usuario ingresar sus credenciales para acceder a la plataforma.

12. Formulario de creación

    General: Permite crear un nuevo usuario en el sistema.

    Interacción: El usuario rellena el formulario y aprieta el botón para crear el usuario.

    Funciones: Crear nuevos usuarios para acceder a la plataforma.

Las interfaces implementadas en el prototipo se encuentran en la sección 8.3.7.

8.3. Prototipo funcional del sistema

8.3.1. Entornos de desarrollo

Entorno de desarrollo local

Para el entorno de desarrollo local, es necesario contar con un IDE y tener instalada la versión 3.9 de Python para ejecutar los códigos. Cada repositorio se puede iniciar en su contenedor Docker respectivo o en un entorno virtual. En este entorno, se deben instalar los requerimientos de cada repositorio desde el archivo requirements.txt. Una vez completada la instalación, se debe ejecutar el archivo entrypoint.py para iniciar la API o la plataforma. La interfaz de la plataforma será accesible desde un navegador web, ingresando al puerto local donde se haya desplegado.

Entorno de producción

En el entorno de producción, es necesario configurar un servidor adecuado con Python 3.9 instalado. Cada repositorio debe desplegarse dentro de su contenedor Docker correspondiente, o bien en un entorno de producción virtualizado. Es crucial asegurarse de instalar todos los requerimientos especificados en el archivo requirements.txt de cada repositorio. Luego, se debe ejecutar el entrypoint.py para iniciar la API o la plataforma. La interfaz de la plataforma estará disponible a través del navegador web, accediendo a la dirección IP del servidor y al puerto configurado para el despliegue.

8.3.2. Base de datos

La base de datos utilizada en el sistema es PostgreSQL en su versión 16.3, gestor para base de datos relacionales en el que el equipo de trabajo almacena información de lo necesario para pruebas de la funcionalidad del sistema (Tablas con información accedida por los flujos), como información referente a la data que se maneja dentro de la aplicación (conversaciones, métricas etc.)

8.3.3. API

Endpoint	Tipo de solicitud	Respuestas	Cuerpo
Endpoint	**Tipo de solicitud**	**Respuestas**	**Cuerpo**
/ping/	GET	[200]	
/chat/sendMessage/	POST	[200-422]	prompt: string, conversation_id: int, user_id: int
/chat/getConversations/	GET	[200-422]	user_id: int
/chat/getConversationMessages/	GET	[200-422]	conversation_id: int
/chat/changeConversationName/	POST	[200-422]	conversation_id: int name: string

/chat/getConversationTable/	GET	[200-422]	limit: int, offset: int, order_by: string, order_way: string
/files/download/	GET	[200-422]	file_id: int
/files/check/	GET	[200-422]	file_id: int
/metrics/send/	POST	[200-422]	conversation_id: int, calification: int, questions: dict
/metrics/getTable/	GET	[200-422]	limit: int, offset: int, order_by: string, order_way: string

Cuadro 8.1: Endpoints API

8.3.4. Autenticación

Para la autenticación de la API, se utiliza el protocolo OAuth 2.0, que es un estándar ampliamente usado para la autorización segura. En este caso, dado que la API será accedida mediante un esquema Machine-to-Machine (M2M), se implementa Auth0 como proveedor de autenticación. Auth0 actúa como un servidor de autorización, encargado de recibir las solicitudes de tokens y emitir los tokens necesarios para acceder a la API.

Por otro lado, se configura un cliente dentro de Auth0, el cual estará asociado a un client_id y un client_secret. Estos datos son credenciales únicas que se pasarán al backend como variables de entorno para mantener la seguridad. El backend utilizará estas credenciales para solicitar un token de acceso a Auth0, el cual será utilizado para hacer llamadas a la API.

Cuando la API recibe una solicitud acompañada de un token, valida su autenticidad

y permisos antes de permitir el acceso a la información solicitada. De esta manera, se asegura que sólo las máquinas autorizadas puedan acceder a la API de forma segura y controlada.

8.3.5. Notificaciones

El sistema de notificaciones utiliza como base la API de Google, con la cual vía correo se notifican tanto los cambios de contraseñas como de permisos en la plataforma a los usuarios respectivos.

Casos de uso:

- Restablecer contraseña: Le llega a todos los usuarios, independiente de su rol, que necesitan cambiar su contraseña. Un ejemplo de este mensaje se muestra en la figura 8.1.

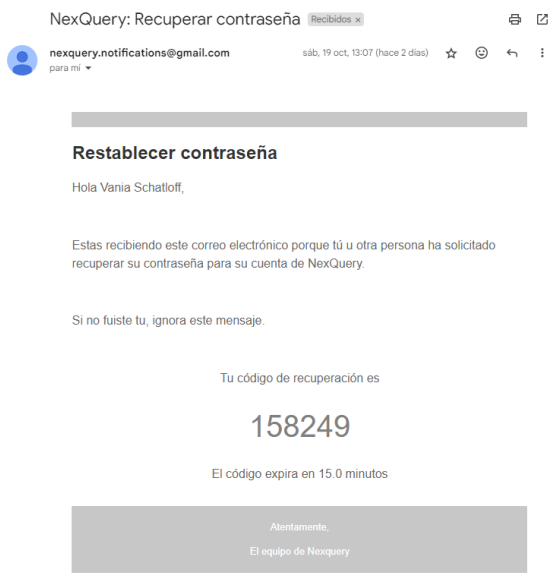


Figura 8.1: Correo para restablecer contraseña

- Nuevo usuario: Le llega a los usuarios administradores. Un ejemplo de este mensaje se muestra en la figura 13.

- Cambio de rol: Le llega a los usuarios cuyo rol fue actualizado. Un ejemplo de este mensaje se muestra en la figura 12.

8.3.6. Almacenamiento de archivos

Para el almacenamiento de archivos, los archivos se almacenan de forma local dentro del proyecto. Estos archivos se guardan en el sistema local del proyecto, permaneciendo ahí hasta que son procesados por un cron. Este cron, que se ejecuta de manera programada, se encarga de recorrer los archivos almacenados localmente y los elimina después de que haya transcurrido un determinado periodo de tiempo desde su creación. Es decir, una vez que los archivos alcanzan un tiempo específico desde su creación, el cron los detecta y procede a eliminarlos automáticamente.

8.3.7. Interfaz gráfica

La interfaz gráfica comprende una aplicación con múltiples funcionalidades a través de varias vistas en las que se puede navegar e interactuar de manera intuitiva y amigable.

Contando con:

- Un inicio de sesión en el cual se encuentra implementado todo el sistema de crear una cuenta como el de cambiar contraseña.
- Vistas para el cambio de contraseña, comenzando con la solicitud, la verificación de código y finalizando con el cambio de contraseña. En el anexo 9 se encuentran las vistas para esta función.
- Una vista de bienvenida dinámica en base a los permisos que el usuario posea. En el anexo 9 se encuentran las vistas para esta función.
- Una vista de chat donde se puede tener conversaciones con el chatbot especializado en obtener respuestas de la base de datos en NL. Esta vista cuenta ademas con una barra lateral desplegable donde se puede ver el historial de conversaciones de la cuenta que se este utilizando, estas conversaciones también cuentan con

un sistema de valoración para así obtener un feedback directo de los usuarios y saber en que tipo de preguntas puede estar fallando el sistema.

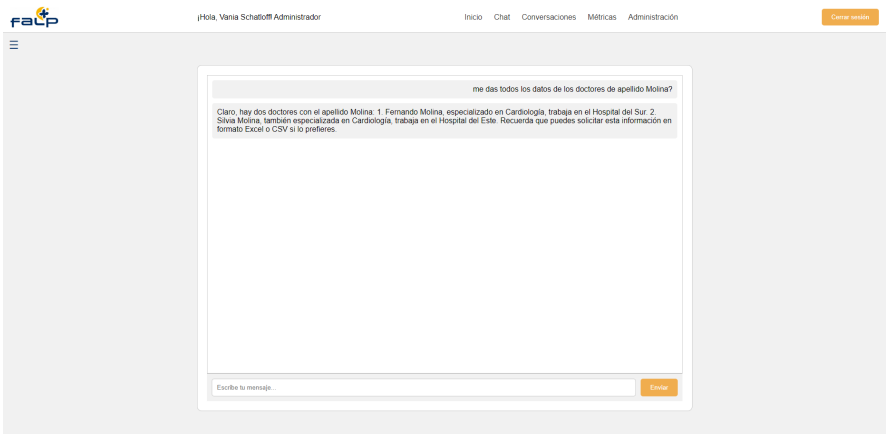


Figura 8.2: Vista principal de chatbot

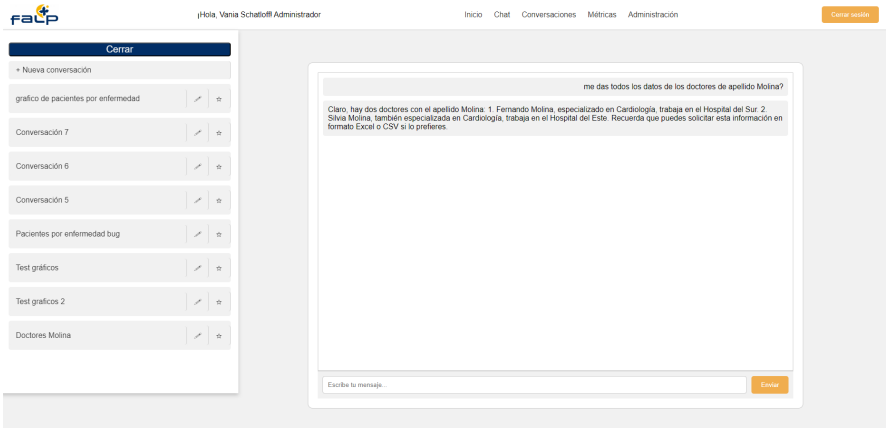


Figura 8.3: Vista de barra lateral con conversaciones

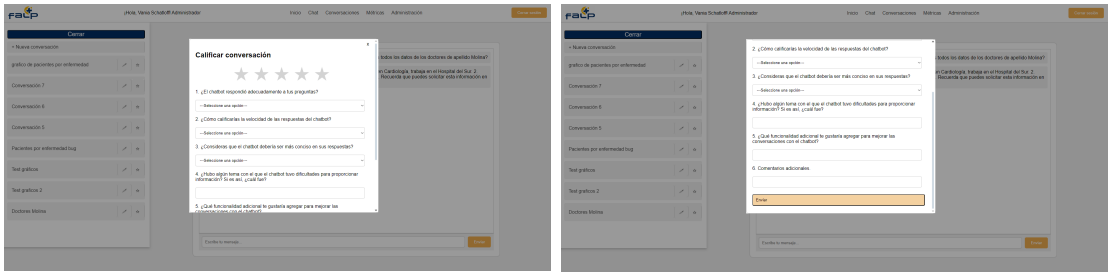


Figura 8.4: Vista para calificar conversación

- Una vista de conversaciones donde se pueden ver todas las conversaciones que han tenido los usuarios. Esta cuenta con un sistema de filtrado para facilitar su uso y generar un uso mas amigable para los usuarios con acceso a esta vista.

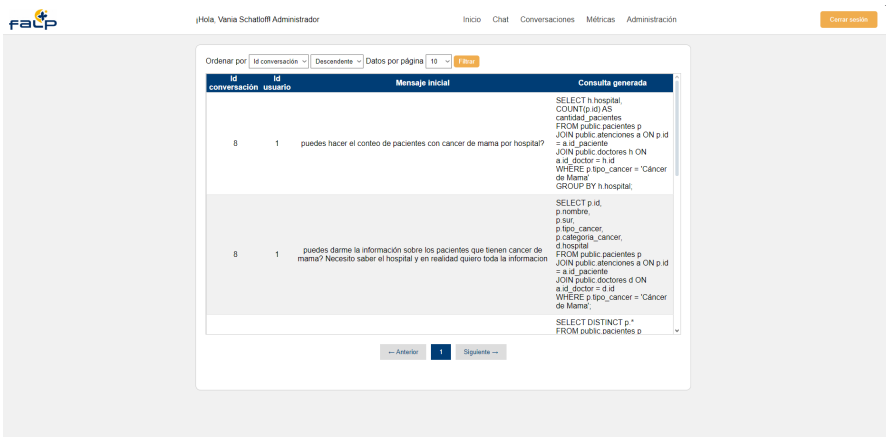


Figura 8.5: Vista de tabla de conversaciones

- Una vista de métricas donde se pueden observar las respuestas de los usuarios y la configuración básica del chatbot con la que este esta trabajando para responder esa conversación en particular, esta vista también cuenta con un sistema de filtrado dentro de la pestaña de ”ver métricas” y ”ver preguntas” para acceder de manera rápida a la información que al administrador le interese.

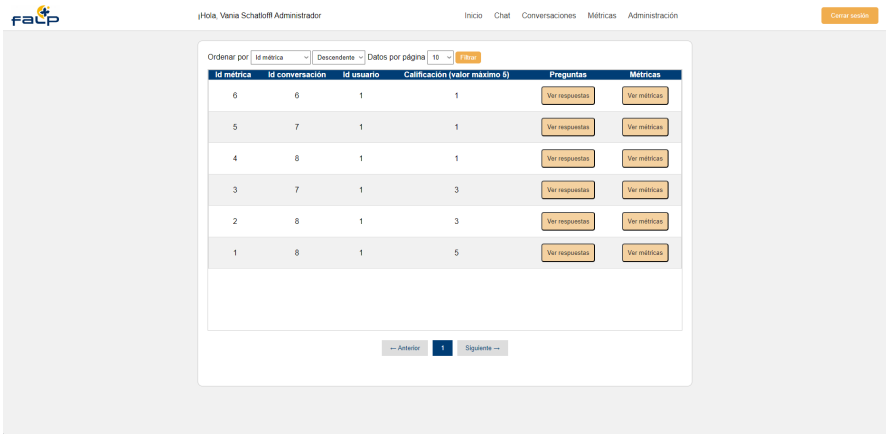


Figura 8.6: Vista de tabla de métricas

Figura 8.7: Vista de modal con preguntas respondidas del usuario

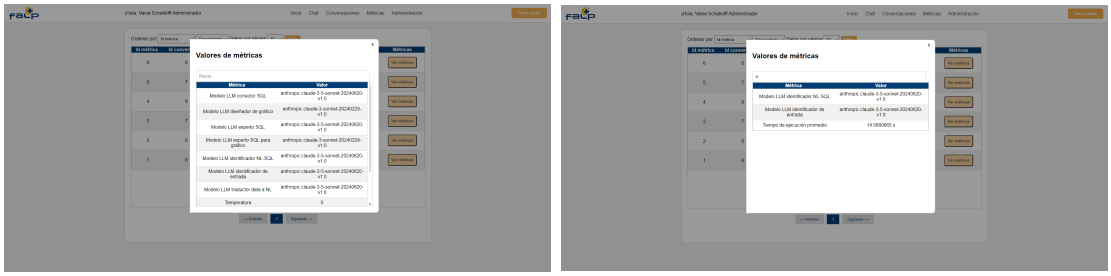


Figura 8.8: Vista de modal con métricas de la conversación y ejemplo de uso del buscador

- Una vista de administrador donde se pueden crear y manejar los permisos de los distintos usuarios de la plataforma para otorgarles los distintos tipos de acceso a la misma.

  - Usuarios: Permite administrar los roles de cada usuario.

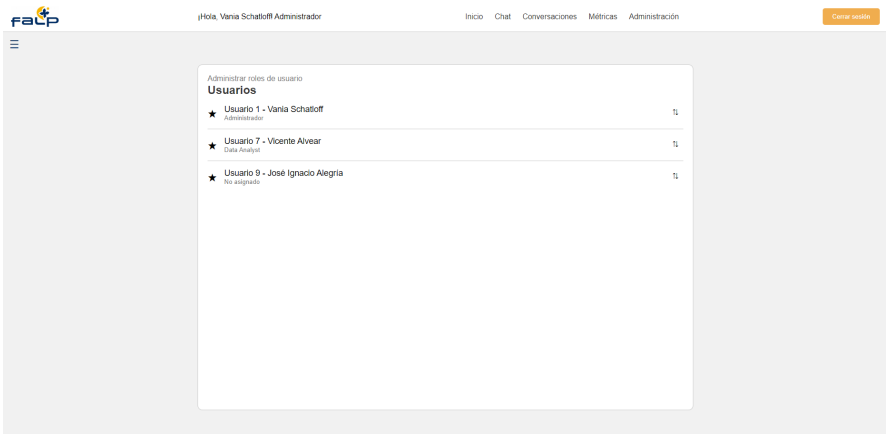


Figura 8.9: Vista de administración de usuarios

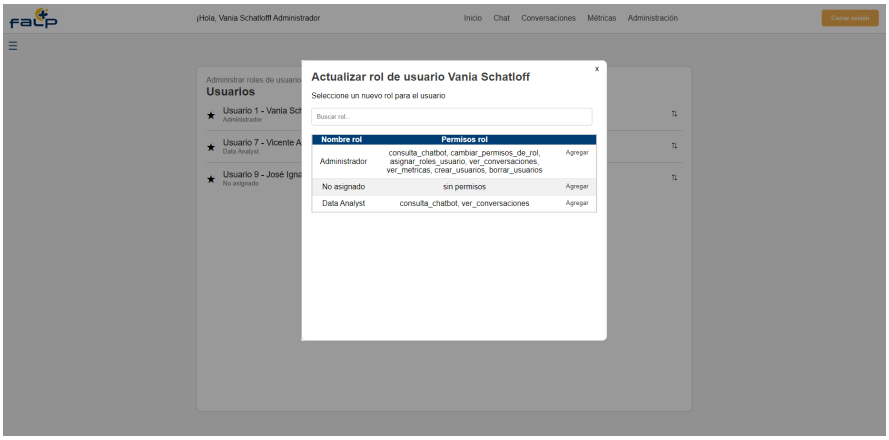


Figura 8.10: Vista de modal de actualización de rol de usuario

- Roles: Permite la creación y actualización de roles.

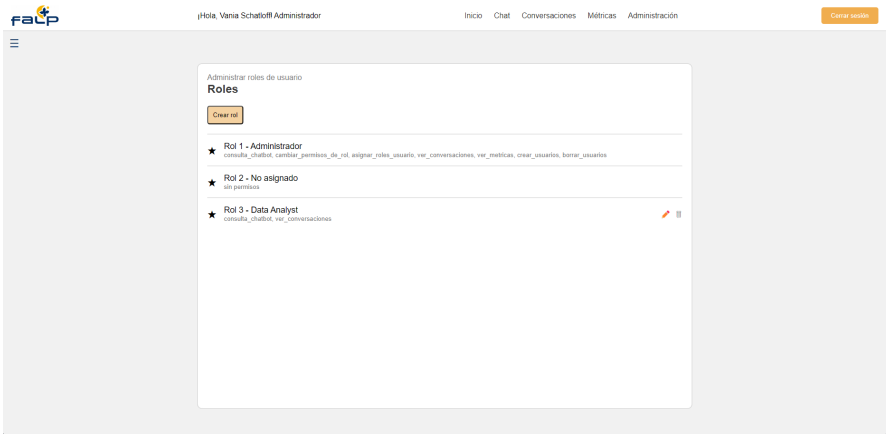


Figura 8.11: Vista de administración de roles

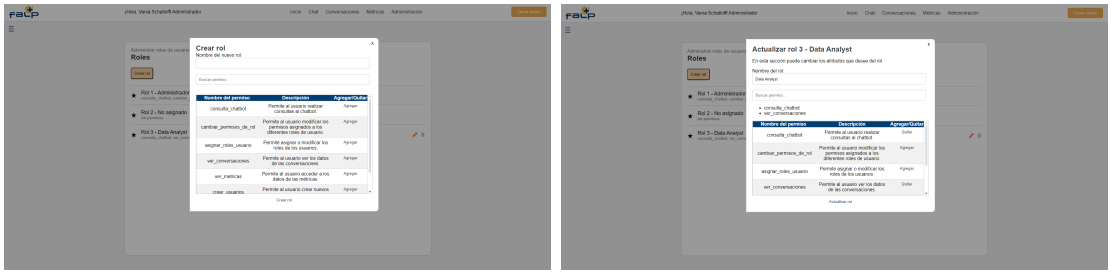


Figura 8.12: Vista de modal de creación y actualización de roles

- Permisos: Permite ver los permisos existentes para la plataforma.

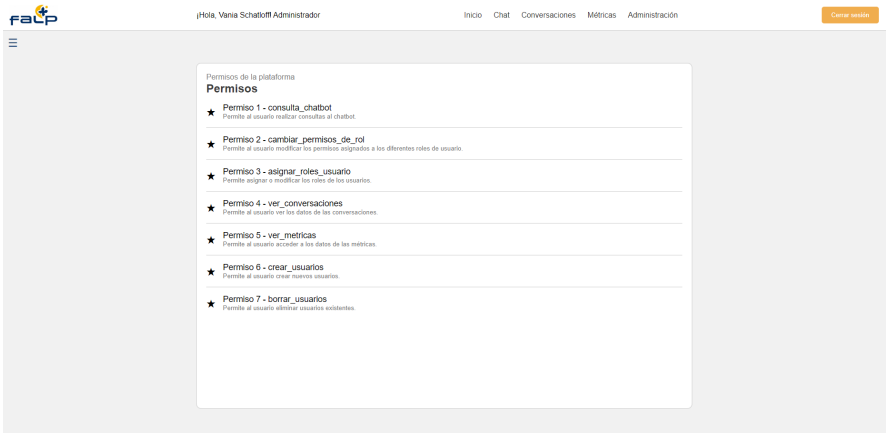


Figura 8.13: Vista de permisos

- Todas las vistas cuentan con una barra de navegación global en la sección superior de la vista para acceder a las distintas vistas previamente mencionadas,

así como también un botón para cerrar sesión. Esto permite una navegación más rápida y sencilla para el usuario.

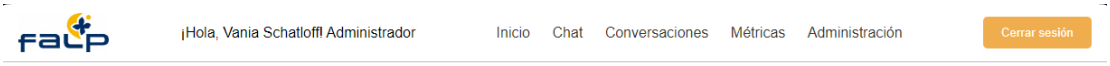


Figura 8.14: Barra de navegación

Capítulo 9

Conclusiones

Esta memoria cumple con el objetivo de democratizar el acceso a la información, proporcionando un prototipo funcional de una aplicación que permite interactuar con una base de datos mediante lenguaje natural. Esto se realizó mediante el uso de un chatbot que, con contexto sobre la base de datos del cliente, generaba consultas que eran luego ejecutadas y entregadas al usuario en formato excel, csv y en lenguaje natural. Para la interacción con el chatbot, la plataforma fue clave, debido a que permite a cada usuario identificarse, almacenar sus chats y consultarlos cada vez que sea necesario. Esto elimina la necesidad de que un profesional técnico sea el único responsable de comprender el modelo de datos, permitiendo que cada miembro de la organización acceda y utilice la información de manera independiente, sin depender de un grupo reducido de personas.

Como trabajo futuro, se planea incorporar una funcionalidad que permita a los usuarios ingresar directamente la información de su propia base de datos, eliminando la necesidad de un intermediario para la carga de documentos. Además, habilitar la API para integrarse fácilmente con aplicaciones como Slack o WhatsApp ampliaría las posibilidades de uso y facilitaría su adopción en distintos entornos.

Este trabajo ha permitido comprender la importancia de democratizar el acceso a la información. En este sentido, la solución propuesta representa un excelente punto

de partida para permitir que todas las personas dentro de una organización accedan a la información de manera rápida, libre e independiente. Los modelos de lenguaje de gran tamaño, con su capacidad de crecimiento, ofrecen un potencial significativo para automatizar dinámicamente todo el flujo de consultas y obtención de datos, facilitando así el acceso a la información y mejorando la eficiencia operativa. A medida que estos modelos evolucionan, también lo hará la capacidad de la solución para adaptarse y escalar, lo que abre nuevas posibilidades para optimizar la gestión de la información y transformarla en una herramienta más accesible y poderosa para todos.

Anexo - Vistas - prototipo 1.2

Descritas en la sección6.2.2.

Figura 1: Inicio de sesión

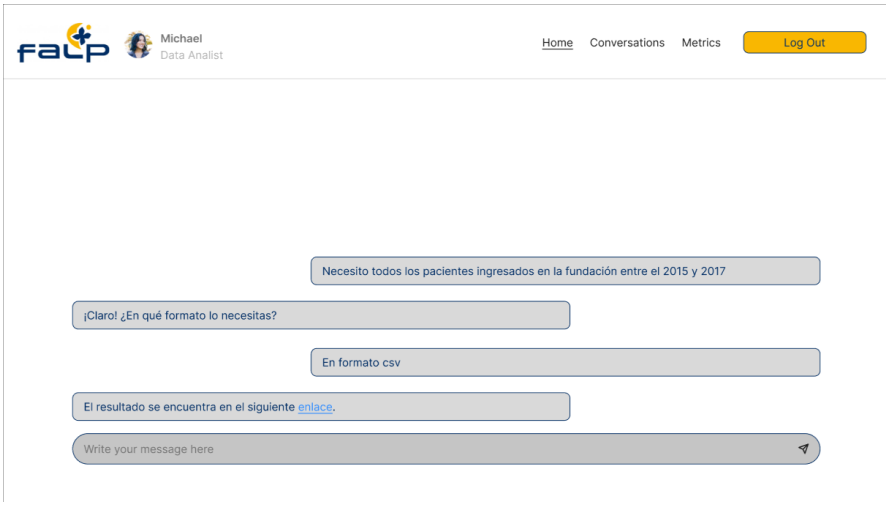


Figura 2: Menú principal - chatbot

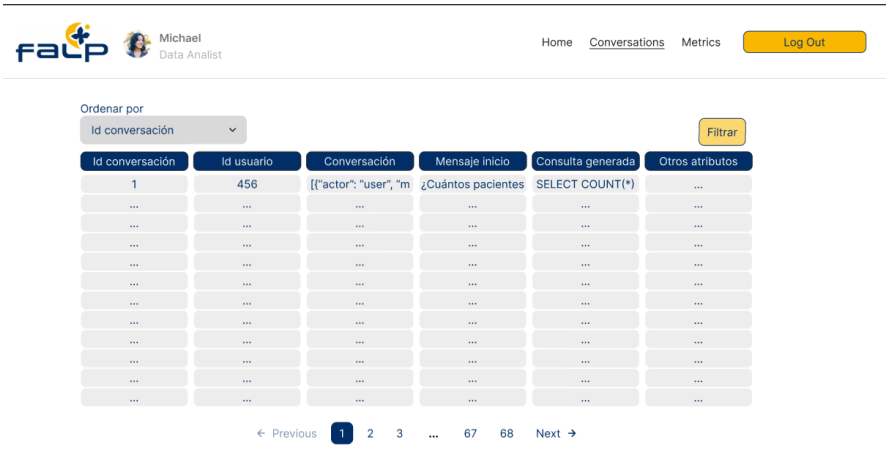


Figura 3: Registros de conversaciones

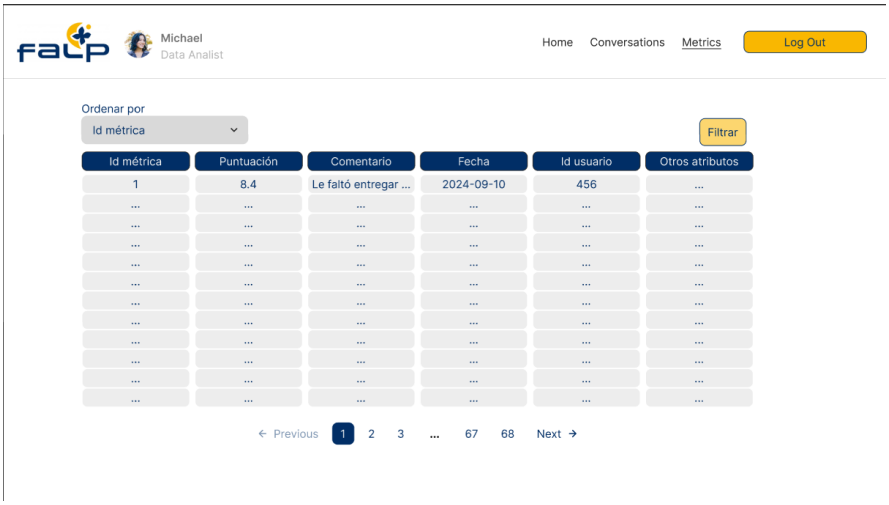


Figura 4: Métricas de satisfacción

En la tabla 8.3 se aprecian las principales interacciones con el prototipo de plataforma.

Anexo - Vistas - página principal

Descritas en la sección 7.3.2.

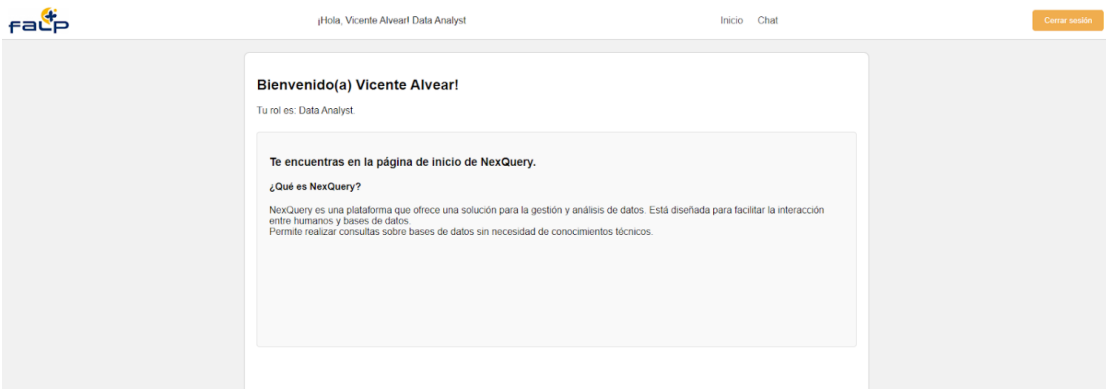


Figura 5: Backend y Frontend - Vista principal de usuario con algunos permisos

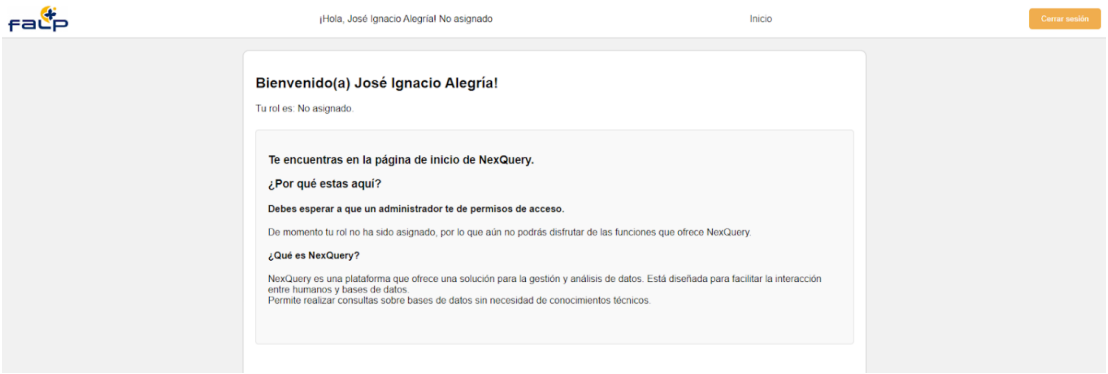


Figura 6: Backend y Frontend - Vista principal de usuario sin permisos

Anexo - Vistas - cambio de contraseña

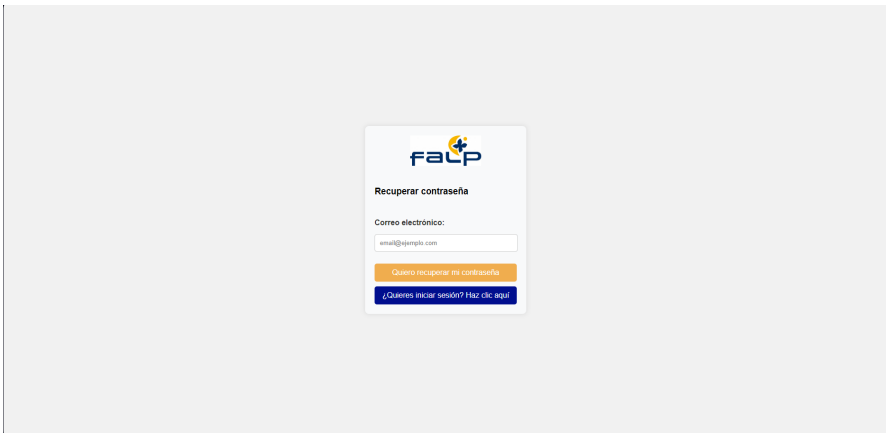


Figura 7: Vista de solicitud de nueva contraseña

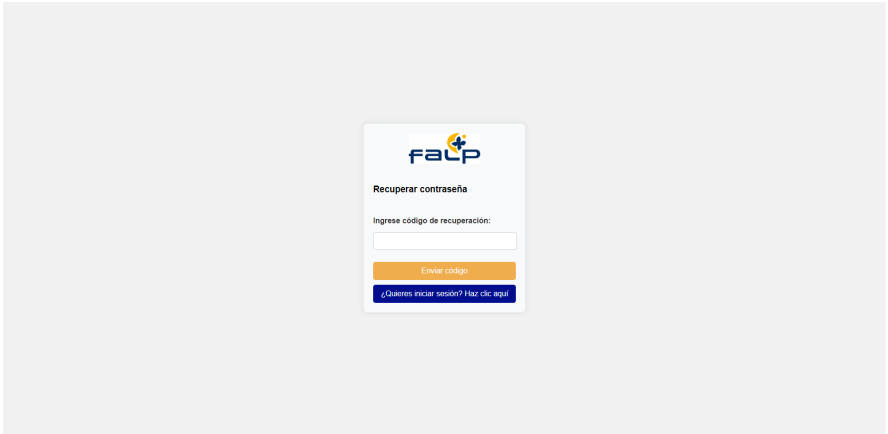


Figura 8: Vista de ingreso de código

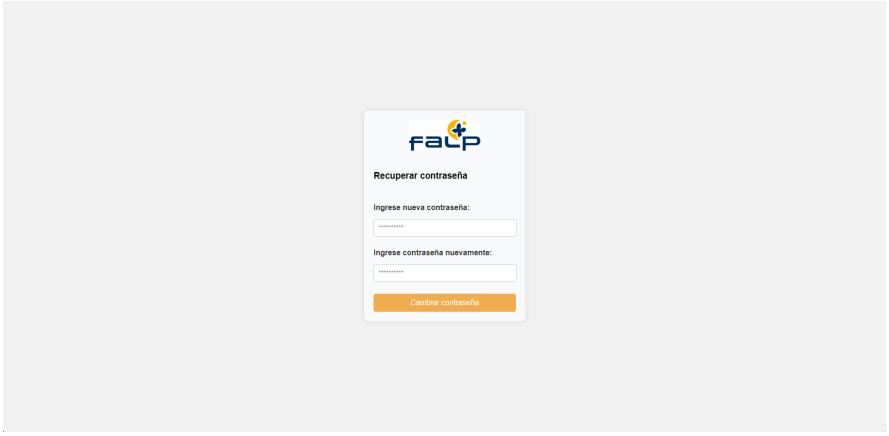


Figura 9: Vista de cambio de contraseña

Anexo - Vistas - Ingreso

Cuando se inicia sesión correctamente, aparece una notificación al costado que indica que se ha iniciado sesión correctamente.

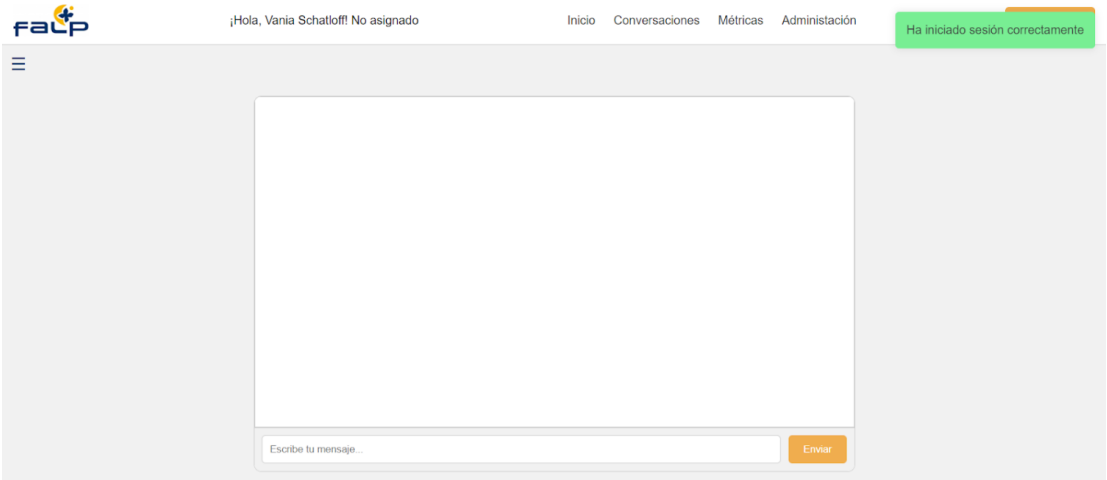


Figura 10: Backend y Frontend - Inicio de sesión correcto

En caso del registro, si hay un valor vacío, se notifica de la siguiente manera:

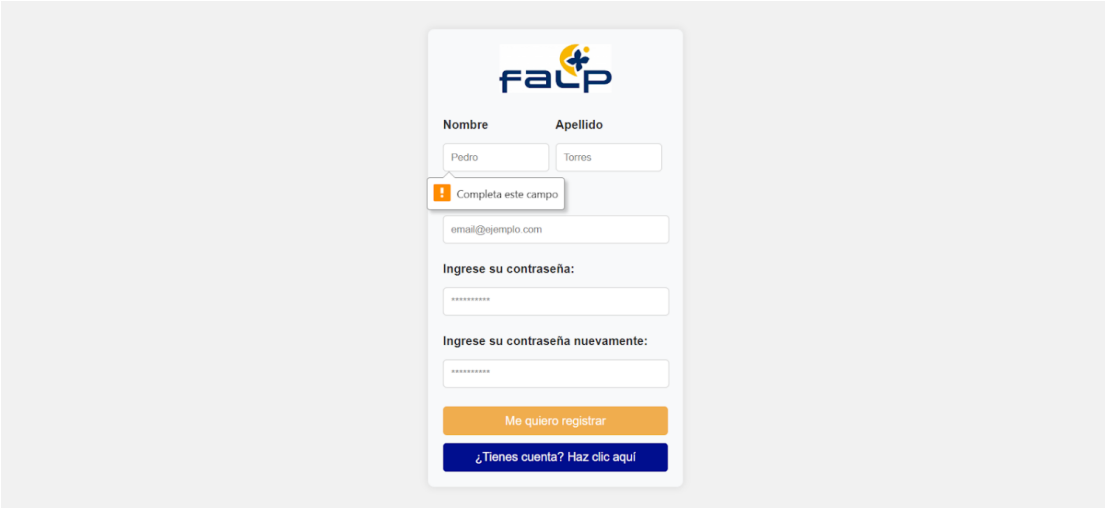


Figura 11: Backend y Frontend - falta de datos en registro

Anexo - Correos electrónicos

Se presentan a continuación ejemplos de correos electrónicos enviados al usuario desde la plataforma.

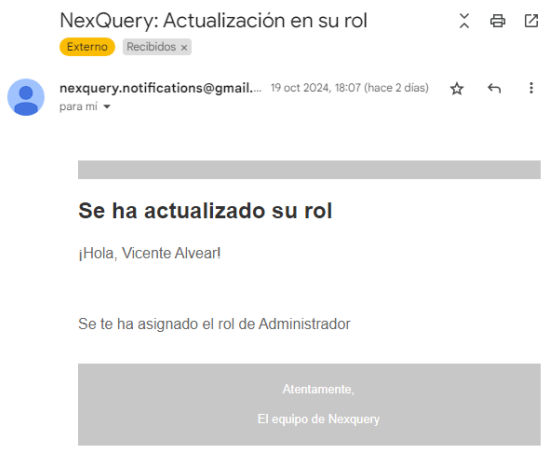


Figura 12: Correo para aviso de actualización de rol

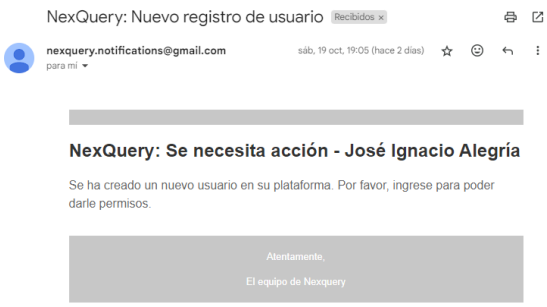


Figura 13: Correo para aviso de creación de usuario

Anexo - API - Estructura de directorios

Para el desarrollo de la API se tiene en consideración una estructura modular y escalable de código, la que se explicará a continuación.

```
app/
|____ crons/
|____ crud/
|____ external_services/
|____ models/
|____ routers/
|____ schemas/
|____ utils/
|____ __init__.py
|____ dependencies.py
|____ logger.py
temp_files/
tests/
.env
.gitignore
docker-compose.yml
Dockerfile
```

`README.md`

`requirements.txt`

En la carpeta **`crons`** se almacenan todos los archivos que contienen procesos recurrentes, organizados por categoría para facilitar su gestión.

En **`crud`** se encuentran los procesos encargados de administrar y manipular los datos de la base de datos, es decir, aquellos relacionados con la creación, lectura, actualización y eliminación de información (CRUD). Además, aquí se almacena el código del ORM utilizado para ejecutar las consultas a la base de datos.

La carpeta **`external_services`** contiene el código de los procesos que interactúan con servicios externos, como Amazon Bedrock de AWS, que permite la interacción con modelos avanzados.

En **`models`** se almacenan los modelos de la base de datos, definidos como clases adaptadas al ORM.

La carpeta **`router`** contiene todos los endpoints de la API, organizados mediante *blueprints*. Esto significa que los endpoints se agrupan por prefijos comunes; por ejemplo, todos los relacionados con la administración de archivos están en un archivo bajo el prefijo `/files`.

En **`schemas`** se definen las estructuras del cuerpo (*body*) requeridas para las llamadas a los endpoints, especificando los formatos necesarios para recibir dichas solicitudes correctamente.

La carpeta **`utils`** agrupa archivos utilizados de manera transversal por diferentes módulos, como ayudantes, controladores y otros elementos de soporte.

Por último, algunos archivos clave incluyen:

- **`__init__.py`**: contiene la definición principal de la API.
- **`dependencies.py`**: define las variables de entorno y los formatos requeridos para las solicitudes.
- **`logger.py`**: implementa el sistema de registro de eventos (*logging*) para la API.

Anexo - Funcionamiento API - Capturas de Postman

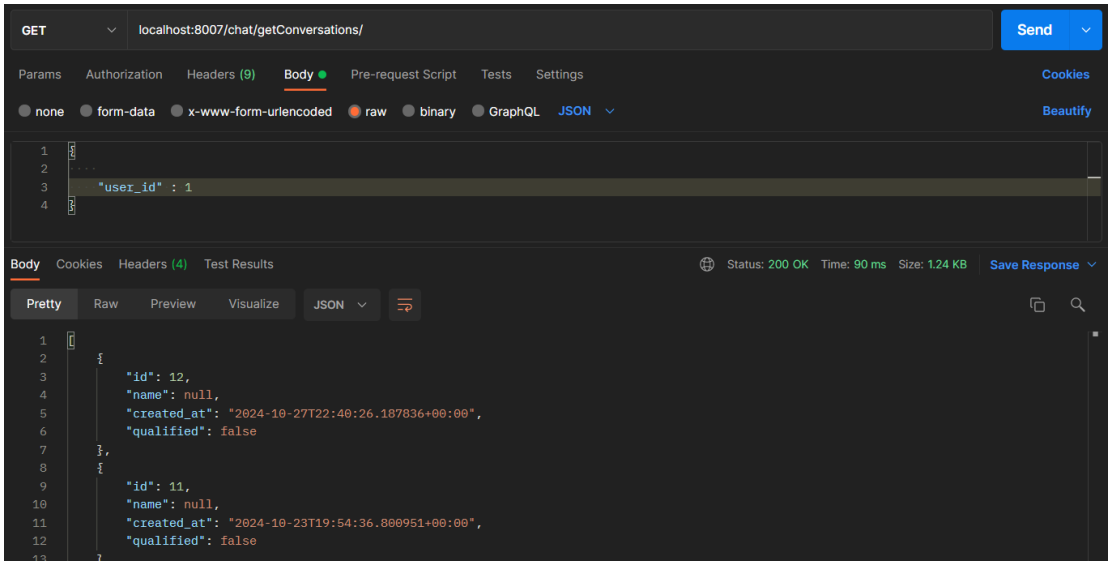


Figura 14: API - get conversations

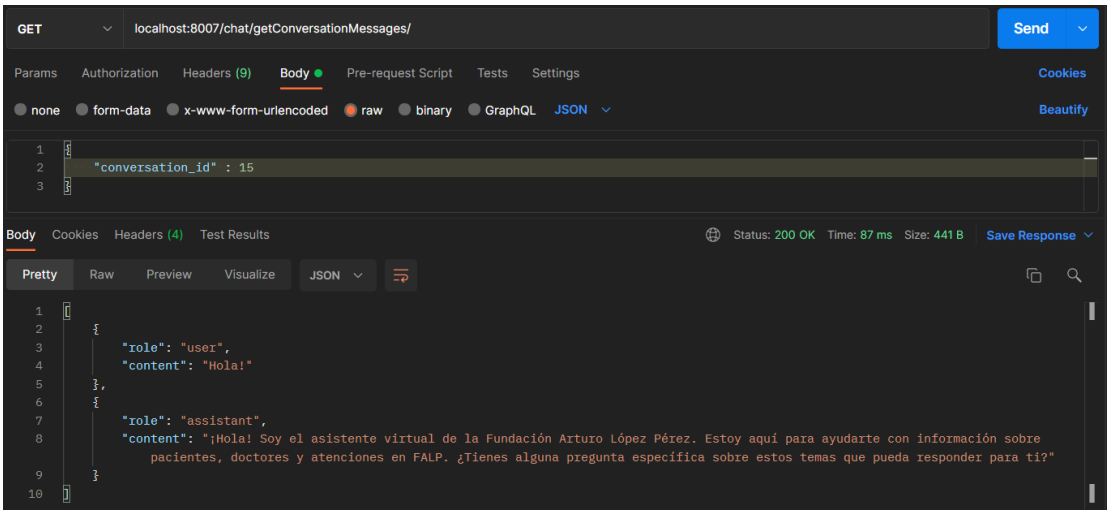


Figura 15: API - get conversation messages

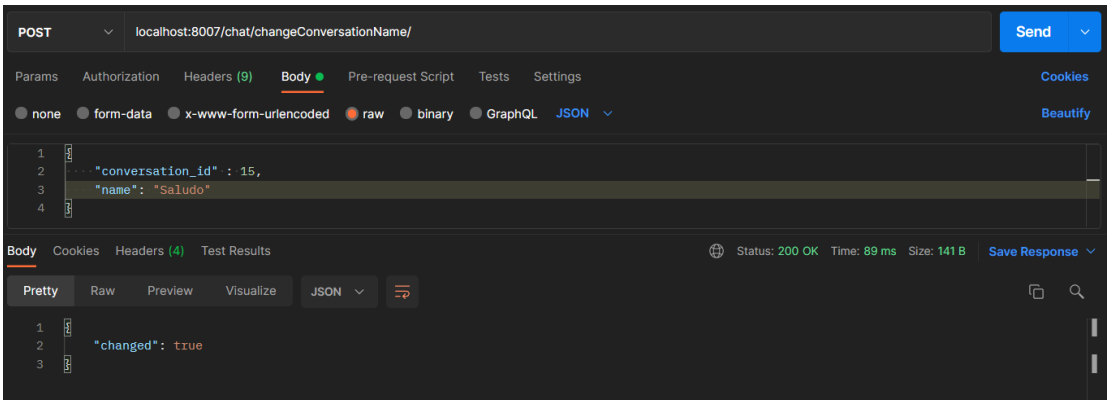


Figura 16: API - change conversation name

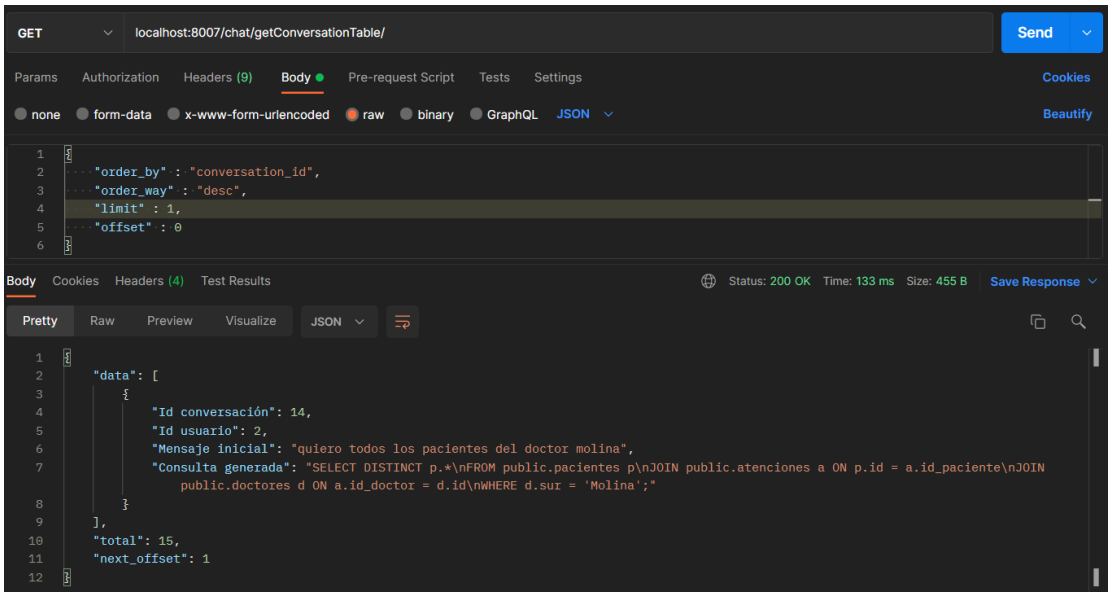


Figura 17: API - get conversations table

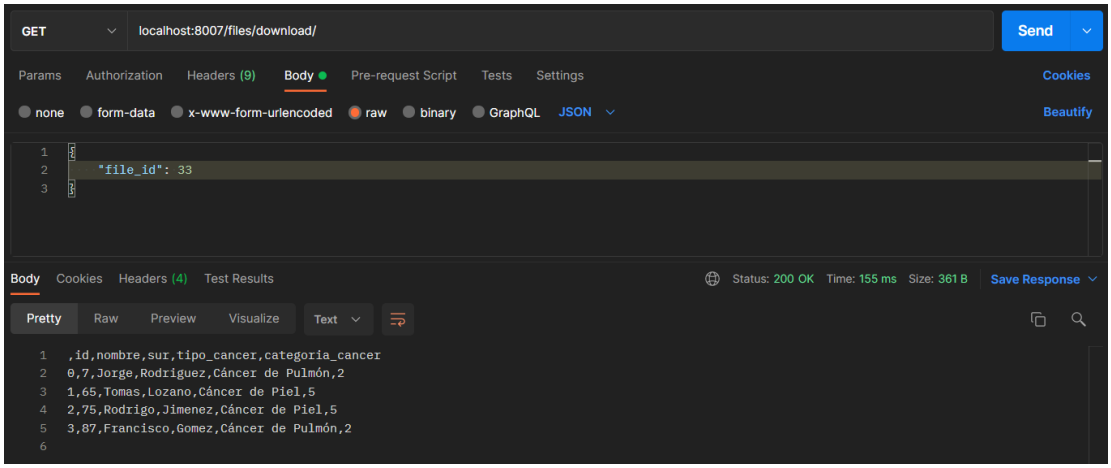


Figura 18: API - download

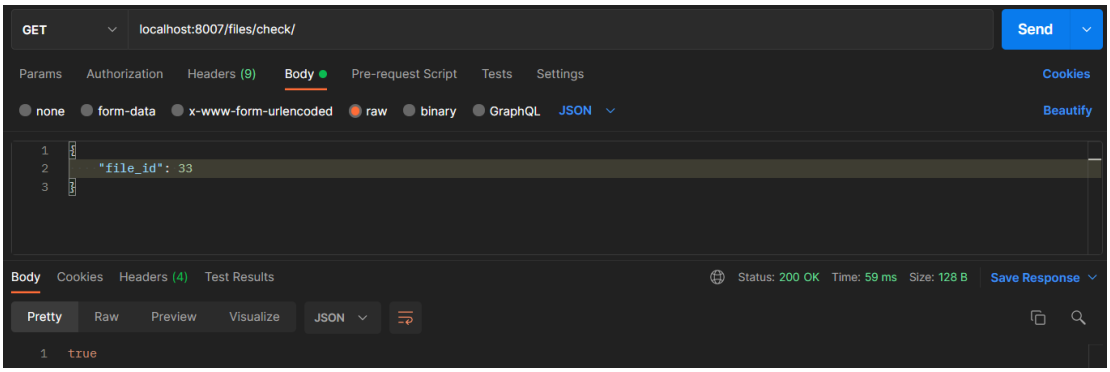


Figura 19: API - check

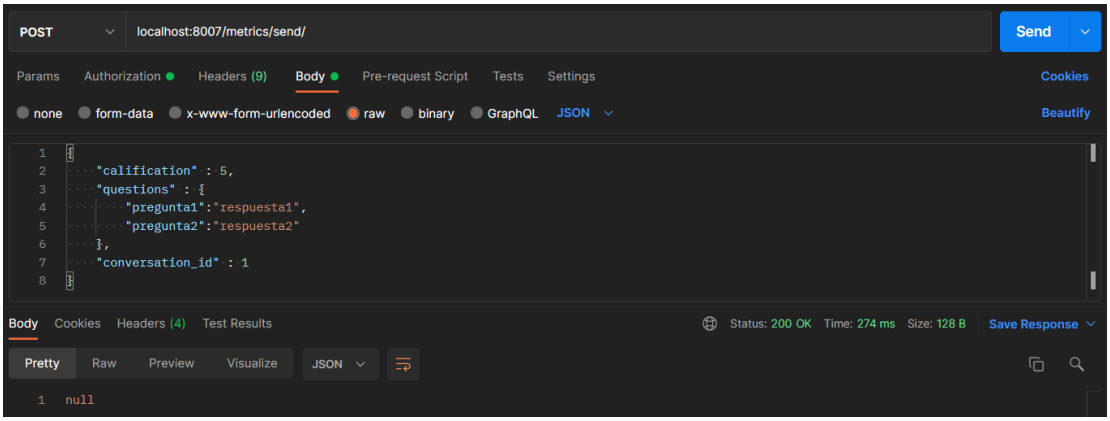


Figura 20: API - send

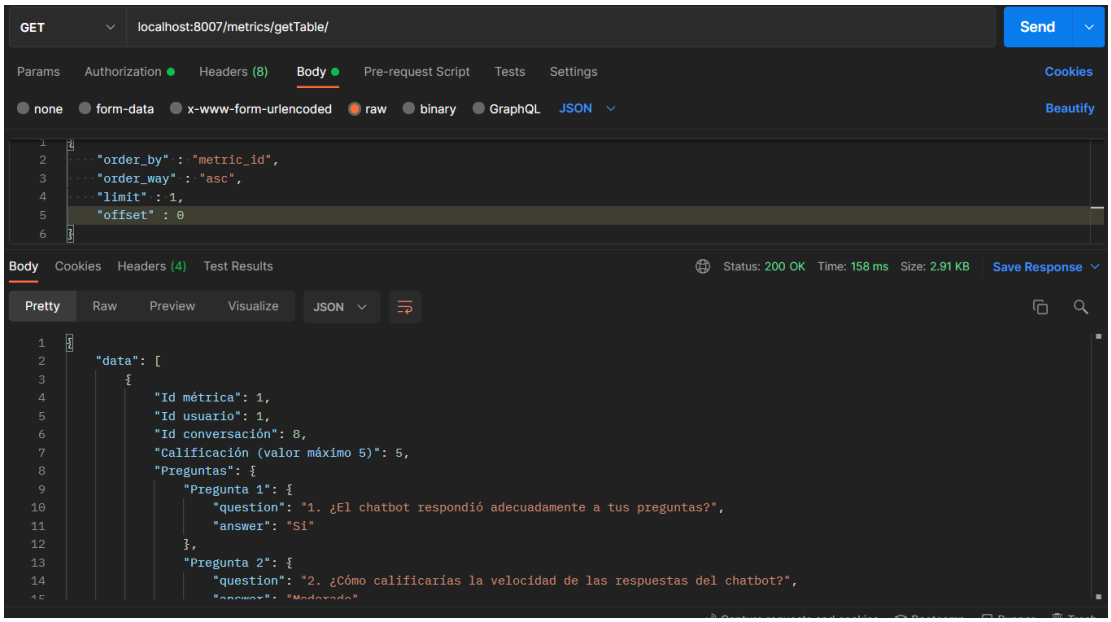


Figura 21: API - get table

```
{
    "data": [
        {
            "Id métrica": 1,
            "Id usuario": 1,
            "Id conversación": 8,
            "Calificación (valor máximo 5)": 5,
            "Preguntas": {
                "Pregunta 1": {
                    "question": "1. ¿El chatbot respondió adecuadamente a tus preguntas?",
                    "answer": "Si"
                },
                "Pregunta 2": {
                    "question": "2. ¿Cómo calificarías la velocidad de las respuestas del chatbot?",
                    "answer": "Moderado"
                },
                "Pregunta 3": {
                    "question": "3. ¿Consideras que el chatbot debería ser más conciso en sus respuestas?",
                    "answer": "No"
                },
                "Pregunta 4": {
                    "question": "4. ¿Hubo algún tema con el que el chatbot tuvo dificultades para proporcionar
información? Si es así, ¿cuál fue?",
                    "answer": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum"
                },
                "Pregunta 5": {
                    "question": "5. ¿Qué funcionalidad adicional te gustaría agregar para mejorar las conversaciones
con el chatbot?",
                    "answer": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum"
                },
                "Pregunta 6": {
                    "question": "6. Comentarios adicionales.",
                    "answer": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum"
                }
            },
            "Métricas": {
                "Temperatura": 0.0,
                "Top P": 0.1,
                "Modelo LLM identificador de entrada": "anthropic.claude-3-5-sonnet-20240620-v1:0",
                "Modelo LLM experto SQL": "anthropic.claude-3-5-sonnet-20240620-v1:0",
                "Modelo LLM identificador NL-SQL": "anthropic.claude-3-5-sonnet-20240620-v1:0",
                "Modelo LLM corrector SQL": "anthropic.claude-3-5-sonnet-20240620-v1:0",
                "Modelo LLM traductor data a NL": "anthropic.claude-3-5-sonnet-20240620-v1:0",
                "Modelo LLM diseñador de gráfico": "anthropic.claude-3-sonnet-20240229-v1:0",
                "Modelo LLM experto SQL para gráfico": "anthropic.claude-3-sonnet-20240229-v1:0",
                "Tokens de entrada": 1150,
                "Tokens de salida": 319,
                "Tiempo de ejecución promedio": "15.8340505 s"
            }
        }
    ],
    "total": 16,
    "next_offset": 1
}
```


Figura 22: API - respuesta tabla de métricas

Anexo - API - Endpoints

A continuación se describe de manera breve cada endpoint desarrollado.

- **chat - send message**

  - **Descripción -** Permite las interacciones con el chatbot y la obtención de respuestas.
  - **Método -** POST
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.

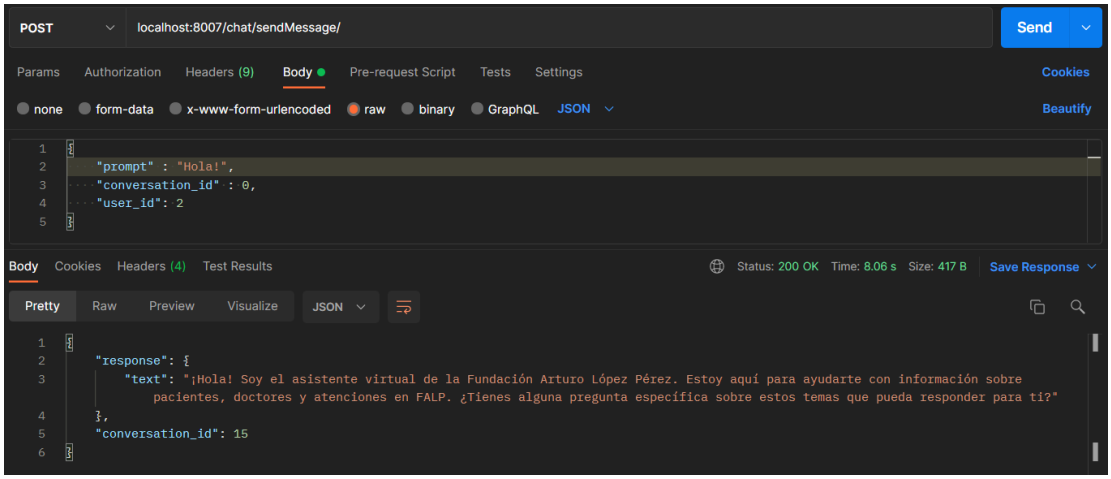


Figura 23: API - send message

  - **Modo de uso -** Para iniciar una nueva conversación, se debe enviar como id de conversación el valor 0. Esto hará que se cree una nueva conversación y el id es luego obtendo en la respuesta. Por ejemplo, en la figura 23, podemos notar que en la respuesta nos fue entregado un id de conversación.

Éste es el valor que se debe utilizar para seguir con la misma conversación luego, es decir, en la siguiente llamada al endpoint es necesario cambiar el conversation_id por 15.

- **chat - get conversations**

  - **Descripción -** Permite obtener todas las conversaciones de un sólo usuario.
  - **Método -** GET
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Se debe colocar el id de algún usuario en el cuerpo de la llamada al endpoint.

- **chat - get conversation messages**

  - **Descripción -** Permite obtener todos los mensajes de una conversación en particular.
  - **Método -** GET
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Se debe colocar el id de alguna conversación en el cuerpo de la llamada al endpoint.

- **chat - change conversation name**

  - **Descripción -** Permite cambiar el nombre de una conversación en particular.
  - **Método -** POST
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Se debe colocar el id de alguna conversación en el cuerpo de la llamada al endpoint.

- **chat - get conversations table**

  - **Descripción -** Permite obtener la información más relevante de cada conversación.
  - **Método -** GET
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Es necesario establecer un valor para el límite de datos que se obtendrán en la llamada, así como un valor para el desplazamiento (offset), que se utilizará para la paginación de los resultados. Además, se debe especificar una columna para ordenar los datos y el sentido de dicha ordenación (ascendente o descendente).

- **files - download**

  - **Descripción -** Permite descargar el archivo generado en la conversación.
  - **Método -** GET
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Es necesario establecer el id del archivo que se desea descargar. Esto, como se ve en la captura de postman de la figura 18, muestra los datos del archivo, pero cuando se llama desde el navegador, el archivo se descarga en el dispositivo del usuario.

- **files - check**

  - **Descripción -** Permite verificar la existencia del archivo generado en la conversación.
  - **Método -** GET
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.

- **Modo de uso -** Es necesario establecer el id del archivo que se desea verificar. Esto, como se ve en la captura de postman de la figura 20, esto responde con un booleano indicando si el archivo existe o no existe en la base de datos.

- **metrics - send**

  - **Descripción -** Permite enviar las respuestas de una encuesta de satisfacción referente a una conversación en particular.
  - **Método -** POST
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Es necesario establecer el id de la conversación, un valor para la calificación y un diccionario con las preguntas hechas al usuario y sus respectivas respuestas.

- **metrics - get table**

  - **Descripción -** Permite obtener la información más relevante de cada métrica.
  - **Método -** GET
  - **Autenticación -** Necesaria. Requiere del uso de OAuth2.0.
  - **Modo de uso -** Es necesario establecer un valor para el límite de datos que se obtendrán en la llamada, así como un valor para el desplazamiento (offset), que se utilizará para la paginación de los resultados. Además, se debe especificar una columna para ordenar los datos y el sentido de dicha ordenación (ascendente o descendente). A continuación se puede observar la respuesta completa de la figura 21

Anexo - Código

En las siguientes figuras, se encuentra el código desarrollado para el control de seguridad de la plataforma, específicamente, respecto a la identificación con la API y con el ingreso por usuario.

```python
from fastapi import Depends, HTTPException
from fastapi.security import OAuth2AuthorizationCodeBearer
from authlib.jose import jwt, JsonWebKey
from pydantic import BaseModel
import requests

from app.dependencies import get_settings

settings = get_settings()

AUTH0_DOMAIN = settings.auth0_domain
API_IDENTIFIER = settings.api_identifier
AUTH0_CLIENT_ID = settings.auth0_client_id
AUTH0_CLIENT_SECRET = settings.auth0_client_secret
ALGORITHM = settings.algorithm

oauth2_scheme = OAuth2AuthorizationCodeBearer(
    authorizationUrl=f"https://{AUTH0_DOMAIN}/authorize",
    tokenUrl=f"https://{AUTH0_DOMAIN}/oauth/token"
)

class TokenData(BaseModel):
    sub: str = None

async def verify_token(token: str = Depends(oauth2_scheme)):
    try:
        if settings.environment == "dev":
            return {"sub": "development_user"}

        response = requests.get(f'https://{AUTH0_DOMAIN}/.well-
known/jwks.json')
        jwks = JsonWebKey.import_key_set(response.json())
        unverified_header = jwt.decode_header(token)
        key = jwks.find_by_kid(unverified_header['kid'])
        if key:
            payload = jwt.decode(token, key, claims_params={
                'aud': API_IDENTIFIER,
                'iss': f"https://{AUTH0_DOMAIN}/"
            })
            return payload

        raise HTTPException(status_code=401, detail="Invalid token")
    except Exception:
        raise HTTPException(status_code=403, detail="Token verification failed")
```

Figura 24: API - seguridad

```python
def permissions_required(permissions_list=[], main_view : Optional[bool] = False):
    from ..common.DB import DB_ORM_Handler
    from ..mod_users.models.role_permission_assoc import RolePermissionAssocObject
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            role_id = session.get('role_id', None)
            if role_id == 2: # No asignado
                kwargs['admin'] = False
                kwargs['chat'] = False
                kwargs['conversations'] = False
                kwargs['metrics'] = False
                if len(permissions_list) == 0:
                    return func(*args, **kwargs)
                else:
                    flash('No tienes permisos para acceder a esta página.', 'error')
                    return redirect(url_for('auth.home'))
            if role_id:
                with DB_ORM_Handler() as db:
                    permissions = db.getObjects(
                        RolePermissionAssocObject,
                        RolePermissionAssocObject.role_id == role_id,
                        columns=[RolePermissionAssocObject.permission_id]
                    )
                    permissions = [perm.get("permission_id") for perm in permissions]
                if main_view:
                    admin = 2 in permissions or 3 in permissions or 6 in permissions or 7 in permissions
                    chat = 1 in permissions
                    conversations = 4 in permissions
                    metrics = 5 in permissions
                    kwargs['admin'] = admin
                    kwargs['chat'] = chat
                    kwargs['conversations'] = conversations
                    kwargs['metrics'] = metrics
                if contains_sublist(permissions, permissions_list):
                    return func(*args, **kwargs)
            flash('No tienes permisos para acceder a esta página.', 'error')
            return redirect(url_for('auth.home'))
        return wrapper
    return decorator
```

Figura 25: Backend y Frontend - requerimiento de permisos

La siguiente función es ejecutada cuando se hace clic en ”Iniciar sesión” luego de ingresar los datos.

```python
def login():
    email = request.form.get('email')
    password = request.form.get('password')
    user = get_user(email)
    if user != {} and check_password_hash(user.get("password"),
password):
        role = get_role(user.get("role_id"))
        session['user_id'] = user.get("id")
        session['user_name'] = user.get("name")
        session['user_lastname'] = user.get("lastname")
        if role:
            session['user_role'] = role
        session['conversation_id'] = 0
        flash('Ha iniciado sesión correctamente', 'success')
        return redirect(url_for('chatbot.main_chat'))
    else:
        flash('Email o contraseña inválida', 'danger')
        return redirect(url_for('auth.index'))
```

Figura 26: Backend y Frontend - Función de Inicio de Sesión

En el caso de querer registrarse, se ejecuta la siguiente función.

```python
def register():
    email = request.form.get('email')
    name = request.form.get('name')
    lastname = request.form.get('lastname')
    password = request.form.get('password')
    password_review = request.form.get('password_review')
    # Validación de entradas
    if not email or not name or not lastname or not password or not password_review:
        flash('Por favor, completa todos los campos', 'danger')
        return redirect(url_for('auth.register'))

    if password != password_review:
        flash('Las contraseñas no coinciden', 'danger')
        return redirect(url_for('auth.register'))

    hashed_password = generate_password_hash(password, method='pbkdf2:sha256')
    try:
        existing_user = get_user(email)
        if existing_user != {}:
            flash('El correo electrónico ya está registrado', 'danger')
            return redirect(url_for('auth.register'))
        insert_new_user(email, name, lastname, hashed_password)
        flash('Registro exitoso. Por favor, inicia sesión', 'success')
        return redirect(url_for('auth.index'))
    except Exception as e:
        flash('Ocurrió un error durante el registro. Por favor, intenta de nuevo', 'danger')
        return redirect(url_for('auth.register'))
```

Figura 27: Backend y Frontend - Función de Registro

Para cerrar sesión:

```python
def logout():
    session.clear()
    flash('Sesión cerrada.', 'success')
    return redirect(url_for('auth.index'))
```

Figura 28: Backend y Frontend - Función se cierre de sesión

La barra de navegación se carga dependiendo de los permisos del usuario, para que no aparezcan aquellos botones que no puede acceder.

```
<body>
    <header>
        <div class="navbar">
            <img src="{{ url_for('static', filename='images/logo.png') }}" alt="Logo"
class="logo">
            <div class="user-info">
                <span>¡Hola, {{name}} {{lastname}}!   {{user_type}}</span>
            </div>
            <nav>
                <a href="{{ url_for('auth.home') }}">Inicio</a>
                {% if chat %}
                    <a href="{{ url_for('chatbot.main_chat') }}">Chat</a>
                {% endif %}
                {% if conversations %}
                    <a href="{{ url_for('conv.main_conversations') }}">Conversaciones</a>
                {% endif %}
                {% if metrics %}
                    <a href="{{ url_for('metric.main_metrics') }}">Métricas</a>
                {% endif %}
                {% if admin %}
                    <a href="{{ url_for('admin.main_administration') }}">Administración</a>
                {% endif %}
            </nav>
            <form action="{{ url_for('auth.logout') }}" method="get" style="display: inline;">
                <button type="submit" class="logout">Cerrar sesión</button>
            </form>
        </div>
    </header>
    {% include 'flash_messages.html' %}
    {% block content %}{% endblock %}
</body>
</html>
```

Figura 29: Backend y Frontend - Código de la barra de navegación

Las conversaciones se almacenan en la base de datos utilizando los modelos correspondientes. Para extraer la información necesaria, se diseña la siguiente consulta SQL:

```
WITH query_messages AS (
    SELECT
        id,
        conversation_id,
        message
    FROM messages
    WHERE type = 'query'
),
question_messages AS (
```

```
    SELECT
        id,
        conversation_id,
        message
    FROM messages
    WHERE type = 'conversation' AND
    CAST(message as varchar) LIKE '%''user''%'
),
join_query_question AS (
    SELECT
        query.message as mq,
        question.message as mqq,
        question.conversation_id,
        c.user_id,
        ROW_NUMBER() OVER (
            PARTITION BY query.id ORDER BY question.id DESC
        ) AS row_num
    FROM query_messages query
    JOIN question_messages question
        ON question.conversation_id = query.conversation_id
        AND question.id < query.id
    JOIN conversations c ON c.id = query.conversation_id
)
SELECT
    conversation_id as ''Id conversación'',
    user_id as ''Id usuario'',
    mqq->>'content' as ''Mensaje inicial'',
    mq::json->'content'->>'query' as ''Consulta generada''
FROM join_query_question
```

```
WHERE row_num = 1 and conversation_id = 6
ORDER BY {ORDER BY} {ORDER WAY}
LIMIT {LIMIT}
OFFSET {OFFSET}
```

Bibliografía

[1] Farzana Kapadia. Data democratization for health equity: A public health of consequence, august 2023. *American Journal of Public Health*, 2023. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC10323841/pdf/AJPH.2023.307350.pdf.

[2] Digital Health Insights. Data democratization in healthcare: Unlocking insights for improved outcomes, September 2023. Available at: https://dhix.dhinsights.org/wp-content/uploads/2023/09/SE-HC-EB008-Data-Democratization-2.pdf.

[3] Abhilasha Kate, Satish Kamble, Aishwarya Bodkhe, and Mrunal Joshi. Conversion of natural language query to sql query. In *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, March 2018.

[4] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv:2308.153*, November 2023.

[5] Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. Knowledge-to-sql: Enhancing sql generation with data expert llm. *arXiv:2402.11517v2*, March 2024.

[6] Puneet Kumar Ojha, Abhishek Gautam, Ankit Agrahari, and Parikshit Singh. Sft for improved text-to-sql translation. February 2024.

[7] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv:2405.07467v1*, February 2024.

[8] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv:2312.11242v3*, May 2024.

[9] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, January 2018.

[10] Yingpeng Du, Di Luo, Rui Yan, Xiaopei Wang, Hongzhi Liu, Hengshu Zhu, Yang Song, and Zhang. Enhancing job recommendation through llm-based generative adversarial networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8363–8371, March 2024.

[11] Zhenwen Li and Tao Xie. Using llm to select the right sql query from candidates. *arXiv:2401.02115v1*, January 2024.