UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Ingeniería Informática

Innovación y Emprendimiento: Avances del Desafío 6 - Gestión de Reclamos y Resolución de Problemas

Integrantes:

- Jazmin Melo
- Gonzalo Moncada
- Sara Salas
- Nicolas Muñoz
- Nicolas Salinas
- Nicolas Saavedra
- Benjamin Valenzuela

Profesor: Francisco Parra Ortiz

ÍNDICE

Resumen Ejecutivo

En el presente informe se mostrarán los avances relacionados a uno de los desafíos propuestos por la empresa de ecosistema logístico Fletzy, la cual se encarga de planificar, gestionar y ejecutar el transporte de productos, conectando trabajadores independientes con grandes empresas.

Entre los desafíos que se entregaron se tomó el desafío número seis, el cual consta de un gestor de reclamos y resolución de problemas vía whatsapp, este surge por la necesidad de que los Gig Worker o clientes suelen presentar problemas de diferentes índoles, los cuales no cuentan con un canal eficiente de contacto.

Para atender este desafío, primero se realizará un análisis del problema para posteriormente presentar una propuesta de solución a nivel general, mostrando como sería el flujo de los datos que se trabajará en este proyecto.

Posteriormente, se mostrará el estado de desarrollo de todo el proyecto, a través del avance de los siguientes equipos:

- Comunicación y QA (CyQ): Se encarga de la comunicación directa con el cliente y documenta la información de importancia. Además de la creación de metadatos y testeos del programa.
- Chatbot: Se encarga de crear el código de la IA para permitir el flujo entre cliente y chatbot.
- Whatsapp API (WA): Se encarga de la gestión y el uso de la API Whatsapp.
- Back-End: Se encarga de crear toda la lógica de la base de datos y el comportamiento interno de los datos.

Finalmente, se detallan los desafíos que se tuvieron en cada equipo al desarrollar este proyecto, no solo de manera individual, sino también a nivel global. Acompañados de una reflexión sobre lo aprendido en cada caso y de algunas propuestas para realizar mejoras a futuro.

Análisis del Problema

Contexto y Fundamentación

La economía gig, caracterizada por trabajos temporales o por encargo gestionados a través de plataformas digitales, ha experimentado un crecimiento exponencial en los últimos años. Dentro de este contexto, los gig workers (repartidores, conductores, trabajadores de apps de delivery y servicios bajo demanda) enfrentan desafíos laborales importantes: falta de transparencia en pagos, conflictos respecto a horarios, condiciones de trabajo irregulares y una notoria ausencia de canales eficientes para reportar y resolver problemas críticos.

Plataformas tradicionales de gestión de reclamos y soporte, como correos electrónicos o formularios web, presentan desventajas, entre ellas:

- Procesos burocráticos lentos.
- Sensación de indiferencia o despersonalización.
- Dificultad de acceso para quienes cuentan solo con dispositivos móviles.

Esto ha generado un déficit de confianza en las plataformas, una menor satisfacción laboral y un aumento del estrés y la rotación entre los gig workers.

Naturaleza Específica del Problema

La problemática concreta abordada es la ineficiencia en la gestión y resolución de reclamos laborales para gig workers, afectando áreas como:

- Pagos no realizados o demorados.
- Cambios injustificados de turnos o rutas.
- Condiciones laborales inseguras o no acordes a lo prometido.
- Dificultad para acceder a soporte en tiempo real.

En la mayoría de los casos, cuando un trabajador enfrenta alguno de estos problemas, debe ingresar a una plataforma poco intuitiva, completar formularios extensos o esperar días por una respuesta. Por lo que, dado que en muchas partes estos trabajadores dependen casi exclusivamente de su teléfono móvil para comunicarse y gestionar sus actividades, por lo que el canal debe ser simple, accesible y móvil.

Relevancia del Canal de Comunicación

Dado el presente problema, es importante definir un canal de comunicación accesible y disponible para todos los trabajadores de este rubro, de tal forma que tengan un sentimiento de pertenencia con la plataforma y la empresa, además de cubrir una

amplia cantidad de personas, ya que dentro de la comunicación, se debe usar un canal el cual sea conocido por el mayor número de personas, usuarios o Individuos, para que el sistema sea inclusivo con cualquier inteligencia tecnológica.

Desafíos de la Automatización y Priorización de Reclamos

Un aspecto clave es que no todos los problemas reportados merecen la misma urgencia. Asegurar que los casos críticos (por ejemplo, riesgos de seguridad, conflictos graves de pago o denuncias de acoso) sean resueltos en primer lugar es fundamental para proteger los derechos de los trabajadores y la fidelidad de la plataforma.

Por lo tanto, los desafíos concretos a cubrir serían:

- Evaluar automáticamente la gravedad del reclamo.
- Derivar casos a humanos cuando sea necesario.
- Balancear trato empático y eficiencia.

Valor Estratégico y Social de la Solución

- Impacto positivo en la satisfacción y seguridad laboral: Atención oportuna y priorizada de problemas mejora la confianza de los trabajadores en la plataforma y reduce el ausentismo y la rotación.
- Optimización de recursos de soporte: Automatizar reclamos de baja complejidad libera recursos humanos para los casos más delicados y complejos.
- Rendición de cuentas y transparencia: Generar registros auditables de todos los reclamos y su atención permite responder ante quejas o fiscalizaciones futuras.
- Inclusividad digital: Garantiza acceso justo a quienes sólo usan teléfonos móviles y tienen baja alfabetización digital.

Desafíos Técnicos y Operativos

- Precisión en la interpretación automatizada: El sistema debe tratar base a ejemplos reales y ejemplos sintéticos para casos poco frecuentes, como reclamos críticos.
- Centralización y gestión eficiente de datos: Todas las interacciones, decisiones de priorización y resoluciones deben quedar documentadas y protegidas frente a accesos no autorizados.
- Actualización y adaptabilidad: El sistema debe permitir incorporar nuevos tipos de problemas, matices lingüísticos y contextos laborales cambiantes de forma sencilla y escalable.

Propuesta de Solución

El sistema propuesto consiste en una plataforma conversacional automatizada, sustentada en una arquitectura modular, cuyo hilo conductor es la atención a usuarios a través de WhatsApp. La elección de este canal responde a que es la vía de comunicación diaria de la mayoría de los gig workers y ofrece barreras mínimas de adaptación. La solución automatiza flujos conversacionales, permitiendo levantar reclamos, obtener respuestas en tiempo real y priorizar automáticamente los casos críticos gracias a técnicas de inteligencia artificial. Todo el proceso ocurre sin recurrir a formularios web ni correos electrónicos, en línea con las restricciones centrales del desafío.

Componentes Clave del Sistema

- Chatbot Inteligente: Gestiona toda la interacción conversacional, desde la recepción del reclamo hasta la generación de respuestas adaptadas y humanas.
- Priorización Automática: Emplea algoritmos de procesamiento de lenguaje natural para evaluar la gravedad de cada caso y asignar un nivel de prioridad, permitiendo canalizar recursos según la urgencia detectada.
- Comunicación 100% vía WhatsApp: Garantiza accesibilidad y evita canales alternativos que suelen ser engorrosos o poco amigables para el trabajador.
- Escalamiento a Personal Humano: Cuando la IA detecta casos fuera de patrón o de alta complejidad, deriva la solicitud a un operador humano para asegurar la resolución integral.

Funcionamiento del Sistema y Experiencia del Usuario

El trabajador inicia el proceso escribiendo su reclamo a un número de WhatsApp específico. El sistema responde automáticamente, solicitando información relevante según el contexto, como la descripción del problema, datos de identificación y eventuales detalles adicionales. El chatbot, equipado con IA, evalúa el mensaje recibido:

- Si la información no es suficiente, el sistema requiere al usuario que agregue datos específicos.
- El problema es analizado y comparado contra una matriz de situaciones registradas.
- Se asigna una prioridad automática (por ejemplo, en una escala del 1 al 100) y se entrega al usuario la respuesta correspondiente.

En cada paso, el usuario puede añadir información, pedir aclaraciones o solicitar que su caso se considere urgentemente. El proceso está diseñado para mantener siempre una comunicación cercana y empática, reduciendo la típica percepción de frialdad asociada a los bots.

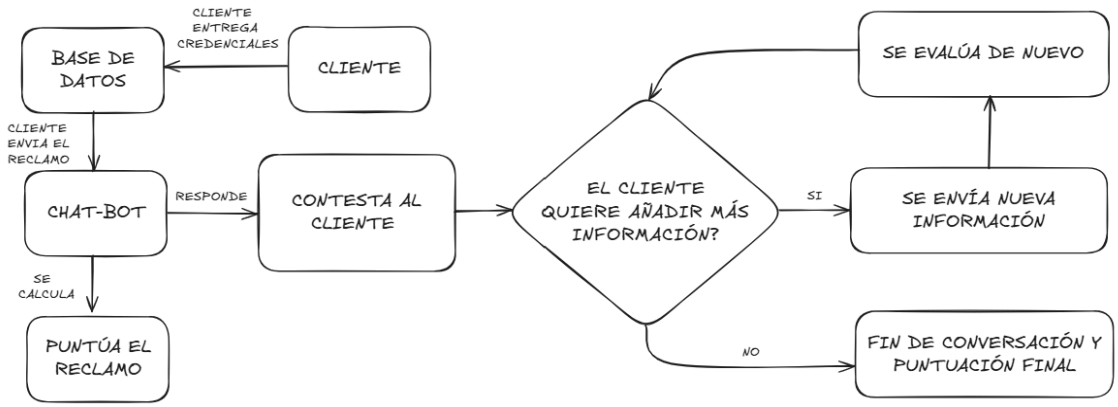

Figura 1: Diagrama de Flujo

Cuando corresponda, se notificará al usuario de la posibilidad de escalar el caso a un agente humano, especialmente si el sistema detecta situaciones de riesgo, insatisfacción reiterada, lenguaje crítico o ausencia de solución en la primera fase.

Priorización y Gestión Dinámica de Reclamos

Una de las innovaciones centrales radica en la capacidad de priorizar dinámicamente todos los reclamos recibidos. No todos los problemas ameritan la misma urgencia: mientras algunos afectan pagos inmediatos o la seguridad física del trabajador, otros pueden esperar dentro del flujo operativo.

El algoritmo de priorización utiliza criterios como:

- Palabras clave de urgencia detectadas en el texto (por ejemplo: “no me pagaron”, “acoso”, “accidente”).
- Historial de reclamos previos del usuario o similares.
- Impacto declarado (por ejemplo, si el problema impide seguir trabajando).

La asignación de niveles de prioridad es transparente; se informa al usuario en qué estado se encuentra su caso y los tiempos estimados de respuesta. Esta funcionalidad es clave para que los usuarios perciban el sistema como justo y eficiente.

En paralelo, el backend almacena todos los reclamos y su metadata relevante para auditorías, mejoras y análisis posteriores, asegurando también el cumplimiento con los requisitos de privacidad y trazabilidad.

Beneficios Estratégicos de la Solución

- Reducción de tiempos de espera y resolución: Los reclamos sencillos son resueltos al instante; los complejos se canalizan inmediatamente hacia agentes especializados, eliminando tiempos muertos y respuestas genéricas.
- Empatía y trato humano: El sistema está entrenado para responder en un lenguaje natural, detectando frustración y escalando cuando la situación lo requiere. Esto incrementa la satisfacción y la percepción de ayuda real.
- Automatización y optimización de recursos: La automatización reduce la carga operativa para los equipos humanos, orientando su energía a los reclamos que realmente requieren intervención directa.
- Inclusión digital: El uso exclusivo de WhatsApp promueve la equidad, permitiendo que personas con baja alfabetización digital o sin acceso a computadoras completen todo el proceso desde el móvil.
- Transparencia y auditoría: Cada interacción queda registrada, posibilitando revisiones posteriores y el mejoramiento de los protocolos a partir del análisis de casos concretos.

Factores Diferenciadores

- Adaptabilidad: El flujo conversacional puede ser ampliado rápidamente para incorporar nuevas categorías de problemas o modificar prioridades según el aprendizaje continuo de la IA.
- Escalabilidad y monitorización: El sistema soporta picos de consultas sin perder calidad y permite monitoreo en tiempo real para la detección de cuellos de botella o tendencias emergentes.
- Configuración personalizada: Es posible ajustar rutas, respuestas y niveles de automatización según las necesidades y perfil cultural del colectivo de trabajadores.

Consideraciones sobre Privacidad y Seguridad

El sistema está diseñado para cumplir con las normativas de confidencialidad y protección de datos, minimizando la recolección de información sensible y cifrando las comunicaciones de extremo a extremo. Los datos personales sólo se usan para identificar y asociar reclamos, y nunca se comparten sin autorización expresa. Además, toda la información almacenada en el backend se centraliza y custodia en servidores seguros, habilitando restauraciones y auditorías, y estableciendo capas de seguridad para el acceso a los registros.

Referencia a Casos Similares

Existen antecedentes exitosos en la implementación de chatbots para la gestión de reclamos en sectores como banca, comercio electrónico y servicios logísticos, donde la adopción de soluciones automatizadas ha demostrado mejorar la eficiencia y satisfacción del usuario, manteniendo la posibilidad de intervenir con asistencia humana en los casos más complejos o críticos.

Arquitectura Técnica

La arquitectura técnica del sistema está diseñada para garantizar una gestión eficiente, segura y escalable de la información, permitiendo una interacción fluida entre los distintos componentes que conforman la solución para la gestión de reclamos y resolución de problemas vía WhatsApp.

Visión General de la Arquitectura

La solución está compuesta por los siguientes módulos principales:

- Base de Datos MySQL: Aloja toda la información relevante de usuarios, reclamos, respuestas y contexto.
- Backend Python (Flask): Gestiona la lógica de negocio, operaciones CRUD, enlaces entre componentes e interfaz visual.
- API de WhatsApp (Twilio): Hace posible el intercambio de mensajes entre usuarios y el sistema.
- Chatbot IA Gemini: Procesa el lenguaje natural, evalúa la gravedad de los reclamos y brinda respuestas humanas.

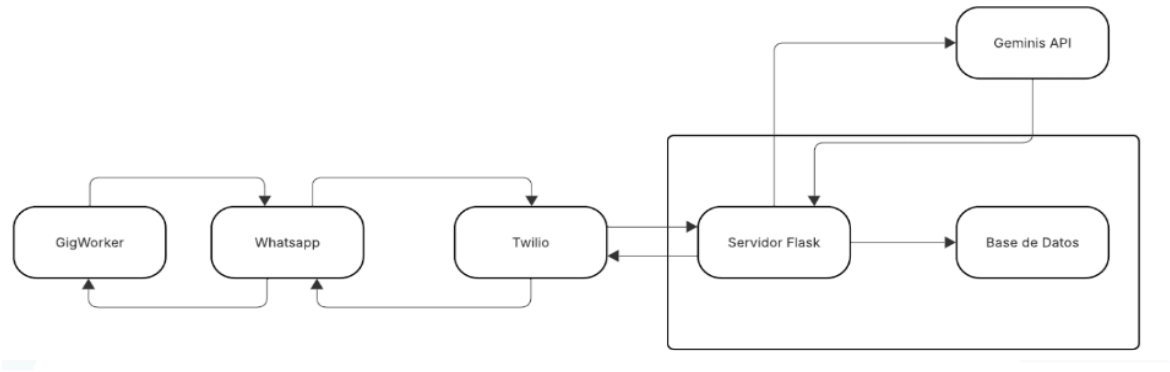

Figura 2: Arquitectura Técnica

Detalle de la Base de Datos

El sistema utiliza una base de datos relacional implementada en MySQL, con una estructura enfocada en mantener la integridad y trazabilidad de las interacciones. Se definen tablas específicas para:

- Usuarios: Almacena datos personales y credenciales.
- Preguntas: Registra las consultas realizadas al chatbot.
- Respuestas: Conserva las respuestas dadas por el sistema.
- Contextos: Relaciona preguntas y respuestas, incluyendo el puntaje asignado a la criticidad del reclamo y el nivel de avance respecto a la solución de la problemática del usuario.

La normalización de las tablas permite una consulta eficiente y la gestión adecuada de la información incluso ante un alto volumen de registros.

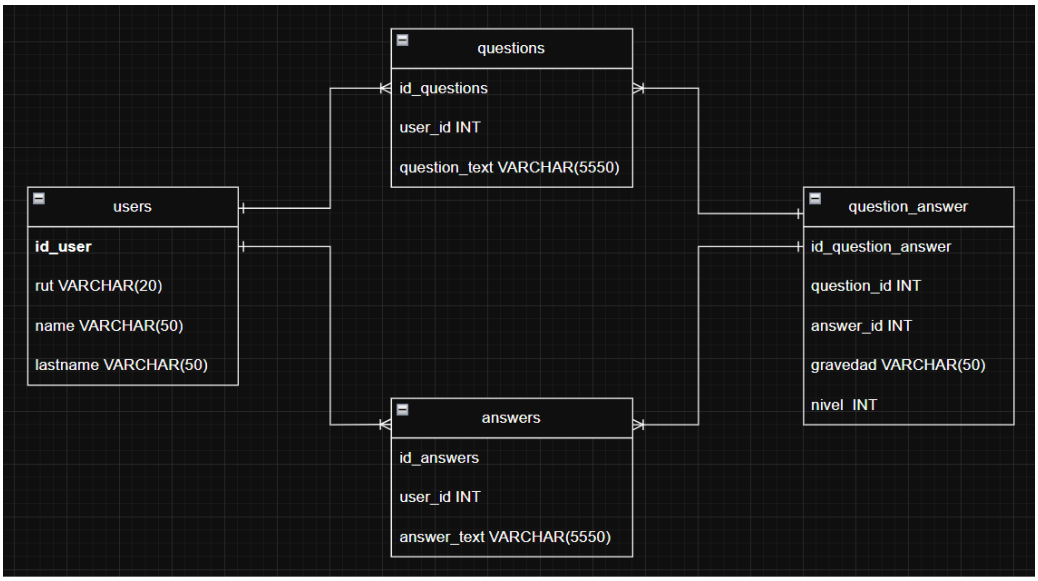

Figura 3: Modelo de base de datos

Modelo CRUD y Gestión Dinámica

Para la manipulación de los datos, se implementó un modelo CRUD utilizando el lenguaje de programación Python. Este modelo posibilita crear, leer, actualizar y eliminar información de manera dinámica, asegurando la coherencia de los datos y permitiendo auditorías y controles sencillos sobre las operaciones realizadas.

Todos los problemas reportados por los usuarios son almacenados directamente en el servidor Flask. Esta centralización no sólo facilita su gestión y trazabilidad, sino que también permite que dichos problemas sean insumos clave para los algoritmos de

priorización del sistema, ya que a partir de la información registrada, el backend evalúa y asigna automáticamente un nivel de prioridad a cada caso.

El backend expone endpoints específicos para cada función, garantizando la interacción segura y eficiente entre el usuario final y la base de datos a través del chatbot. Esta estructura asegura que cada reclamo pueda ser consultado, gestionado y utilizado como referencia para análisis futuros y mejora continua del servicio.

Estructura y Flujos de Comunicación

El backend desarrollado con Flask orquesta la comunicación entre los distintos módulos mediante APIs RESTful. Cada componente cumple un rol específico:

- La API de WhatsApp recibe los mensajes del usuario y los reenvía al backend.
- El backend procesa la información, consulta o actualiza la base de datos según corresponda y coordina la interacción con el Chatbot Gemini.
- El chatbot con IA Gemini interpreta los mensajes, asigna prioridades y genera respuestas contextualizadas, que luego son almacenadas en la base de datos.
- Finalmente, las respuestas llegan al usuario desde el backend a través de la API de WhatsApp.

Esta estructura modular provee escalabilidad y permite la fácil incorporación de nuevas funcionalidades o mejoras tecnológicas en el futuro.

Seguridad y Escalabilidad

La protección de los datos sensibles es una prioridad: el sistema implementa controles de acceso, validaciones de entrada y mecanismos de autenticación para asegurar solo el acceso autorizado a la información. Además, el diseño garantiza la escalabilidad:

- Optimización de consultas y relaciones en la base de datos.
- Separación clara entre lógica de negocio, presentación y almacenamiento.
- Integración de servicios desacoplados que facilitan el mantenimiento y la actualización del sistema.

Implementación

API Whatsapp

A nivel de implementación, han ocurrido múltiples fallos para poder activar la cuenta de whatsapp Business, esto debido a las estrictas normas con las que maneja meta la creación de cuentas. De lo anterior se obtuvieron 2 errores:

- **Inhabilitación del portafolio:** Esto ha ocurrido en varias ocasiones, en algunas ha sido por que la página ha detectado automatización en alguna acción, y otras donde no se ha especificado la razón.
- **Suspensión de la cuenta:** Esto solo pasó en una ocasión, donde incluso se pidió una apelación para reabrir la cuenta, la cual fue rechazada. Se presume que se debió confundir con un bot al ser una cuenta nueva sin ninguna actividad.

Figura 4: Fallos de la creación de la cuenta de WhatsApp

Para resolver esta problemática se decidió usar la plataforma Twilio, la cual funciona como una interfaz más amigable para los desarrolladores. Entre sus funcionalidades incluyen: el envío y recepción de SMS y llamadas, el uso de la API de WhatsApp, y la gestión de correos electrónicos y videollamadas. Para poder acceder a todo lo anterior es necesario crear una cuenta en esta plataforma.

Además de lo anterior, para utilizar la herramienta de manera local se utilizó Ngrok. Esta es una herramienta que permite exponer un servidor local a Internet mediante la creación de un túnel seguro. Esto facilita que servicios, APIs o aplicaciones que solo funcionan en tu computadora puedan ser accesibles desde cualquier lugar.

Para poder integrar todo lo anterior se deben realizar los siguientes pasos:

1. Se ejecuta el comando ngrok http <puerto> (por ejemplo, ngrok http 8080), donde <puerto> es el puerto donde corre tu API local (véase figura 5). Esto genera una URL permitiendo el acceso a esta Api.

2. Luego, en la plataforma Twilio se debe acceder a la opción “Send a Whatsapp message”, que corresponde a un Sandbox que permite usar la Api de Whatsapp de manera limitada pero sin ningún tipo de cobro.
3. En el Sandbox, se escanea el código QR desde el teléfono con el cual se quiere interactuar, iniciando así la sesión.
4. Finalmente, en la configuración del Sandbox, se debe agregar la dirección donde se está ejecutando el programa, la cual es la entregada por ngrok, en la sección "When a message comes in" (vease en la figura 6) y guardar.

Figura 5: Ngrok

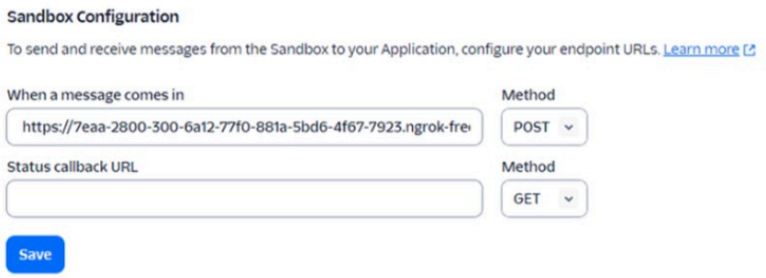

Figura 6: Configuración del Sandbox de Twilo

Chatbot

En lo referente al desarrollo del chatbot, en primera instancia se investigaron diferentes APIs, considerando también la posibilidad de generar un modelo propio entrenado con datos de Fletzy, o bien utilizar datos existentes para realizar dicho entrenamiento. Sin embargo, esta idea fue descartada rápidamente debido a la naturaleza y disponibilidad de los datos, y al hecho de que los modelos de inteligencia artificial existentes en el mercado actualmente son significativamente más eficientes que crear uno propio desde cero. Por lo tanto, se optó por utilizar una solución ya existente, eligiendo finalmente la API de Gemini.

Esta elección se basó en pruebas comparativas realizadas entre Gemini y ChatGPT, utilizando diversas consultas y contextos. A partir de estas pruebas, se concluyó que Gemini ofrecía un mejor desempeño frente a la problemática abordada.

Posteriormente, se desarrolló una API en Python que utiliza la IA de Gemini. Esta solución permite evaluar las consultas del usuario y mantener una conversación fluida, logrando una interacción más natural y humana.

Finalmente, se sostuvo una conversación con el equipo responsable del backend, y se decidió que, en lugar de crear una API independiente y consumirla mediante rutas, se implementarán funciones que serán integradas directamente dentro del backend, optimizando así la arquitectura general del sistema.

Posteriormente en la última iteración del proyecto, se integró el chatbot con el backend transformándolo en un único código, el cual va almacenando la información del usuario en una base de datos mysql, sumado a esto los controladores finales usables del chatbot son:

- /credenciales: este controlador solicita un nombre, apellido, rut a el usuario y almacena los datos del usuario en la base de datos, además de guardar los datos en memoria temporal, esto para poder asociar las preguntas y respuestas al usuario posteriormente.
- /ask: este controlador toma una queja del usuario y mediante una búsqueda vectorial en un archivo JSON el cual contiene una lista de quejas y de cómo manejarlas, se le asigna una gravedad, sumado a esto se le envía la consulta a gemini, para tener una respuestas más humana al usuario, además guarda tanto la pregunta como la respuesta y la gravedad en la base de datos asociados al usuario creado con el controlador anterior, sobra decir que si no se ejecuta el controlador anterior, antes este controlador no funcionará correctamente.
- /regravedad: este controlador recibe como entrada una consulta del usuario y tomando en cuenta la pregunta anterior, junto con la nueva vuelve a calcular la gravedad de la queja del usuario además de volver a usar a gemini para responderle, sumado a lo anterior le solicita añadir algún archivo para confirmar su reclamo.

Con todo lo anterior se logra manejar de buena manera las interacciones con el usuario, manejando correctamente sus quejas, asignándoles una gravedad correcta y almacenando tanto al usuario como a sus preguntas y respuestas en la base de datos, además se consigue dentro de lo posible que el usuario no presienta que está siendo atendido por un bot, si no por una persona, de esta manera el usuario se sentirá valorado y tratado con el respeto que merece.

Back-End

El lenguaje a utilizar para la base de datos es un lenguaje relacional (SQL), utilizando la interfaz MySQL. Este lenguaje permite trabajar con bases de datos relacionales, lo que autoriza almacenar la información de un usuario quien realiza las preguntas al chatbot, guardando sus preguntas y respuesta, además de una tabla llamada “contexto” que relaciona una pregunta con una respuesta y contiene el puntaje asignado y el nivel de avance respecto a la solución de la problemática a atender.

La realización del modelo CRUD para crear, leer, actualizar y eliminar información de esta base de datos se utiliza el lenguaje python, utilizando las extensiones Database, junto a la extensión Flask que posee la capacidad de crear una interfaz que nos permitiese modificar la información en tiempo real. La interfaz se puede observar en las figuras 7, 8, 9, 10 y 11. Donde la eliminación de un usuario, elimina toda pregunta, respuesta y contexto relacionados a este.

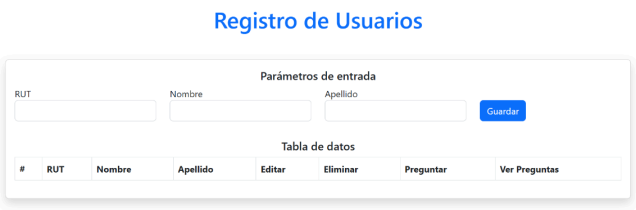

Figura 7: Interfaz

Figura 8: Creación de usuario

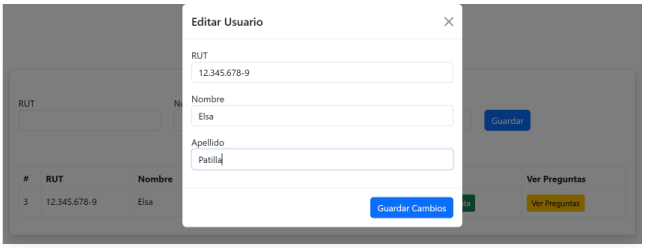

Figura 9: Edición de usuario

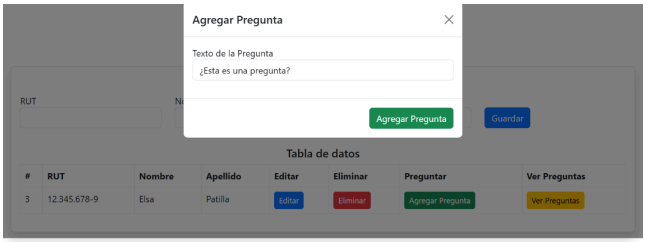

Figura 10: Agregado de pregunta

Figura 11: Eliminando usuario, pregunta y respuesta

QA

Reuniones y Coordinación con el Cliente

Durante el desarrollo del proyecto, el equipo del área de QA sostuvo reuniones semanales con el cliente con el objetivo de alinear la dirección del proyecto con sus expectativas y requerimientos. Estas instancias permitieron una mejor orientación técnica y funcional, en base a las necesidades expresadas por el usuario final.

Requerimientos Funcionales del Chatbot

El cliente definió una serie de funcionalidades clave que debía cumplir el chatbot con Inteligencia Artificial (IA):

- Capacidad de **interpretar lenguaje natural**, entendiendo los mensajes del usuario de forma fluida y contextual.
- Detección **automática del tipo de reclamo** sin intervención humana.
- Evaluación **autónoma del nivel de prioridad** de cada caso, en función del contenido del mensaje.
- Capacidad para **resolver los problemas del cliente de forma automática**, sin intervención de un ejecutivo humano, salvo en casos críticos donde la IA no pueda gestionar adecuadamente la situación.

Requerimientos para la Gestión del Backend

En cuanto al desarrollo del backend, el cliente solicitó las siguientes funcionalidades específicas:

- El sistema debe **almacenar en la base de datos** el historial de casos **críticos y de mediana importancia**, incluyendo las credenciales del usuario, que constan del nombre y apellido de la persona junto con un número de contacto, una vez finalizada la conversación con el chatbot.
- El backend debe mantener un registro estructurado de **todos los tipos de casos gestionados**, incluyendo el **nombre del caso**, su **prioridad** y el **contexto asociado**. El cliente definió inicialmente seis casos críticos (como *Robo*, *Problemas de pago*, entre otros) y una categoría adicional denominada *Otros*.
- La base de datos debe presentar las preguntas y respuestas como tablas separadas, debido a la incapacidad de la interfaz de MySQL de trabajar con listas directamente, por lo cual la relación de estas dos tablas se tiene que aplicar en una tabla diferente que posea los identificadores de ambas tablas.

Desarrollo del Conjunto de Casos y Metadatos

Con base en los requerimientos, se desarrolló un archivo JSON con metadatos para **50 tipos de casos** distintos, con el fin de expandir la cobertura del chatbot y permitir su correcta identificación y tratamiento autónomo. A continuación, se detallan algunos de los tipos de casos incluidos:

Problemas de pago a Gig Worker	Gig Worker no apareció	Falta de seguimiento del paquete	Error en dirección de entrega	Política de devolución confusa
Pérdida de paquete	Gig Worker actuó de forma inapropiada	Horario no respetado	Rechazo de devolución injustificado	Errores frecuente en la app
Carga equivocada	Retraso en pagos de Gig Worker	Error en facturación	Falta de contacto previo a la entrega	Paquete de entrega fuera del rango de horario
Robo de producto	Paquete entregado a persona incorrecta	Aplicación no funcional	Demora en atención telefónica	Retiro programado no realizado
Reembolso no procesado	Gig Worker se negó a entregar	Producto no coincide con descripción	Pérdida de incentivos	Falta de instrucciones para entrega
Atención al cliente deficiente	Datos personales expuestos	Suscripción no autorizada	Descuento no aplicado	Cambio de ruta sin notificación
Servicio técnico ineficaz	Demora en entrega	Cancelación sin aviso	Problemas de garantía	Retraso por condiciones climática no informadas
Facturación duplicada	Paquete dañado	Cambios de condiciones sin aviso	Pedido incompleto	Rechazo de paquete sin causa válida
Trato de personal	Reembolso no procesado	Cobro adicional no informado	Maltrato por parte del cliente	Problemas con sistema de GPS
Carga entrega equivocada	Facturación duplicada	Promesa de entrega incumplida	Sistema de contratación injusto	Comunicación deficiente con Gig Workers

Estructura del Metadato

Cada tipo de caso está definido mediante tres variables clave:

- **Tipo**: Cadena de texto que indica el nombre del tipo de problema.
- **Prioridad**: Valor numérico entre 0 y 100 en un arreglo donde el primer número representa el mínimo y el derecho el número máximo que indica el nivel de criticidad del caso. Un valor de 100 representa un caso extremadamente crítico, que el chatbot deberá resolver con urgencia. Si no puede hacerlo, se derivará automáticamente a un ejecutivo humano.
- **Contexto**: Conjunto de frases clave asociadas al tipo de problema, utilizadas por la IA para identificar automáticamente la categoría del reclamo y gestionar una respuesta adecuada.
- **Respuesta:** cadena de texto que indica cómo debe comportarse la IA con el usuario y el flujo de solución que debe seguir para cada caso.

Este enfoque permite una gestión más eficiente y autónoma de los reclamos, mejorando tanto la experiencia del usuario como la capacidad de respuesta de la plataforma.

Pruebas y Validaciones

El programa mencionado anteriormente no fue probado en un entorno real debido a la interrupción del acompañamiento por parte de la empresa colaboradora, lo cual escapó a nuestro control. En consecuencia, su funcionamiento se limitó a entornos locales, sin posibilidad de validar su desempeño frente a los problemas que surgieron durante el desarrollo.

Se eligió Gemini por sobre otras inteligencias artificiales debido a la existencia de los modelos “Flash”, ya que estos tienen un alto rendimiento para tareas sencillas pero de alto tráfico, como es el caso del chatbot creado. Otro punto a favor son las capacidades multimodales que posee, ya que, si se quisiera expandir el alcance del chatbot al análisis de documentos o imágenes, e incluso a la transcripción de mensajes de voz, Gemini podría cumplir con lo requerido sin ningún problema. Además, hay que tener en cuenta que estamos trabajando con datos sensibles, por lo que la seguridad es prioritaria. Considerando que esta IA es desarrollada por Google, se puede asegurar que es confiable en cuanto a políticas de privacidad.

Conclusiones

A lo largo del desarrollo del proyecto, los equipos enfrentaron desafíos que, en muchos casos, pudieron ser resueltos exitosamente, permitiendo avances significativos tanto en lo técnico como en la coordinación general. Al mismo tiempo, estos avances dieron paso a nuevos aprendizajes y retos, que reflejan la evolución del trabajo conjunto y la mayor complejidad de las soluciones propuestas.

En el caso del equipo de QA (Aseguramiento de la Calidad), se fortaleció significativamente la orientación técnico-funcional, consolidando su rol en la definición y validación de requerimientos del sistema. Este aprendizaje se reflejó tanto en la redacción clara de funcionalidades como en la capacidad de coordinar entre los distintos equipos y el cliente para asegurar la coherencia del desarrollo. Uno de los logros más importantes fue la ampliación y reformulación del archivo JSON de metadatos, que pasó de contener 30 casos a abarcar una cantidad considerablemente mayor, incorporando ahora no solo tipos de reclamos, sino también contextos asociados. Esta evolución permite al sistema realizar una evaluación más precisa y contextualizada de los problemas reportados. El nuevo desafío del equipo consiste en mantener la trazabilidad de estos metadatos en un entorno cada vez más complejo, asegurando su correcta integración con el chatbot y el backend, además de establecer mecanismos de actualización continua para reflejar nuevas situaciones emergentes.

El equipo de Back-End superó las dificultades iniciales de integración y comunicación con otros equipos. A partir de la colaboración directa con el equipo del Chatbot, se logró unificar el código en una sola API, centralizando la lógica de negocio y facilitando el almacenamiento estructurado de usuarios, consultas, respuestas y niveles de gravedad en MySQL. Uno de los mayores aprendizajes fue la implementación eficiente de estructuras CRUD con Flask, manejando relaciones entre tablas como preguntas, respuestas y contextos. El nuevo desafío identificado es la necesidad de escalar esta arquitectura para soportar un volumen mayor de interacciones, optimizando el rendimiento y contemplando la posibilidad de implementar autenticación de usuarios y sesiones.

Chatbot evolucionó desde una implementación inicial basada en pruebas, hasta integrar una solución robusta utilizando la API de Gemini. Se definieron controladores específicos (/credenciales, /ask y /regravedad), permitiendo una experiencia conversacional más coherente y humanizada, con almacenamiento automático en la base de datos. El aprendizaje clave fue entender cómo diseñar interacciones dinámicas y adaptar la lógica del chatbot para gestionar múltiples entradas del usuario, calculando de nuevo el contexto de forma inteligente. Actualmente, el principal desafío es enriquecer el análisis contextual para mejorar la clasificación automática de reclamos y

la coherencia entre respuestas sucesivas, especialmente al recibir información fragmentada o no estructurada del usuario.

El equipo de WhatsApp enfrentó inicialmente grandes dificultades al intentar activar una cuenta oficial de WhatsApp Business, debido a las políticas estrictas de Meta. Tras múltiples bloqueos y apelaciones fallidas, se optó por migrar a Twilio, lo que permitió integrar exitosamente la API de WhatsApp mediante un entorno de pruebas (sandbox) y el uso de túneles seguros con Ngrok. El aprendizaje más valioso fue entender en profundidad el ciclo de vida de mensajes entre WhatsApp, Twilio y el servidor Flask, además de configurar correctamente los webhooks. El nuevo desafío se centra en migrar desde el sandbox de Twilio hacia una cuenta verificada y completamente operativa, además de robustecer el manejo de errores y eventos inesperados durante la interacción con usuarios reales.