Universidad ORT Uruguay
Facultad de Ingeniería

wisello: Sistema de asesoramiento conversacional basado en Inteligencia Artificial Generativa para plataformas de E-Commerce

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Federico Iglesias - 244831
Andrés Juan - 241600
Marcelo Pérez - 227362
Mateo Sciarra - 239352

Tutor: Ignacio Valle

2024

Declaración de autoría

Nosotros, Federico Iglesias, Andrés Juan, Marcelo Pérez y Mateo Sciarra, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Proyecto Final de Ingeniería en Sistemas;

- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;

- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;

- En la obra, hemos acusado recibo de las ayudas recibidas;

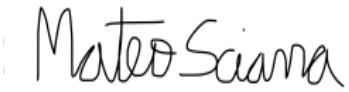
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;

- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Federico Iglesias
19-03-2034

Andrés Juan
19-03-2034

Marcelo Pérez
19-03-2034



Mateo Sciarra
19-03-2034

19-03-2024

Agradecimientos

Nos gustaría aprovechar esta oportunidad para expresar nuestra más profunda gratitud a todas las personas que han jugado un papel crucial detrás de la construcción de *wisello*.

A nuestro tutor, Ing. Ignacio Valle, por su orientación, apoyo y buena disposición a lo largo del proyecto, que fueron de suma importancia para el desarrollo del mismo. Trabajar bajo su mentoría ha sido realmente una experiencia de aprendizaje enriquecedor tanto profesional como personal.

A nuestras familias, por el apoyo que nos brindaron durante todos estos años de carrera. Su paciencia, comprensión y aliento han sido indispensables en cada paso de este viaje.

Al Centro de Innovación y Emprendimientos (CIE) de la Universidad ORT Uruguay, y a Lic. Ximena Scasso, nuestra coordinadora, por su apoyo durante las distintas etapas del emprendimiento, los recursos otorgados, y por la posibilidad de formar parte de esta incubadora que fomenta la innovación entre su comunidad de emprendedores.

A los profesores de la Universidad ORT Uruguay mencionados a lo largo del cuerpo de este trabajo que, tras acudir a ellos por asistencia técnica, nos brindaron su ayuda y tiempo. Valoramos profundamente su disposición.

A profesionales y expertos uruguayos, también mencionados en el trabajo, que generosamente participaron en entrevistas y reuniones con nosotros, brindándonos su tiempo, conocimientos y experiencias valiosas. Cada intercambio ha sido de valor para el proyecto.

Por último, agradecemos a cada individuo que, de una forma u otra, ha contribuido a nuestro proyecto. Amigos que ofrecieron palabras de aliento, y todos aquellos que, directa o indirectamente, han sido parte de *wisello*.

A todos, gracias.

Federico, Andrés, Marcelo y Mateo.

Abstract

Frente a la creciente oferta de artículos en tiendas de e-commerce, los compradores online se enfrentan a un mar de opciones de productos en línea. Sin embargo, lo hacen aislados frente a su pantalla, sin una figura presente como la del vendedor humano en tienda física que aclare sus inquietudes y entienda sus necesidades **particulares**, detectándose así una falta de *asesoramiento online en vivo*.

El modelo actual de compras en línea, por ende, carece de incentivo hacia el usuario para finalizar su compra, o navegar por el catálogo de la tienda, pues la voluntad de buscar está del lado del usuario. Se lo comprende como un paradigma impersonal; la experiencia de compra es la misma para todos, desventaja que se ve exacerbada considerando que, debido a la pandemia, un núcleo de compradores ex-offline (acostumbrado a este paradigma) ha migrado a este canal no por deseo, sino por necesidad.

Si bien se han visto iniciativas para mejorar el asesoramiento en e-commerce (Chatbots tradicionales, Whatsapp, Redes Sociales o recomendación de productos), todas ellas adolecen de una sola característica: su capacidad para **automatizar la personalización**. *Wisello* cubre esa brecha.

Wisello es un producto que se enmarca bajo el fenómeno del **Conversational Commerce**, que refiere a la compra de productos y servicios a través de un agente conversacional (también denominado agente virtual o digital). Mediante la integración de modelos avanzados de inteligencia artificial generativa y su entrenamiento con datos propios de la tienda, el usuario puede mantener una conversación con *wisello*, disfrutando así de una experiencia de compra conversacional e *hiperpersonalizada*, en la cual recibirá recomendaciones de productos acorde a su situación particular, y a través del cual podrá aclarar sus dudas. Además, su arquitectura extensible permite la incorporación de nuevas capacidades o funcionalidades al agente a gusto, aparte de la recomendación de productos.

Los principales componentes de su arquitectura refieren al servicio del `backend`, encargado de procesar las consultas al chat, devolver repuestas y gestionar las diferentes tiendas, el servicio de `chat-frontend`, encargado de proveer una interfaz para el chat, el servicio de `web-platform`, encargado de proveer la interfaz para la plataforma de visualización y configuración de la herramienta destinada a ser utilizada por las tiendas, el servicio de `data-pipelines`, encargado de procesar los catálogos de las tiendas, y el servicio de `bubble-chat-server`, encargado de la integración de *wisello* con la tienda cliente.

Para la gestión del proyecto, se adoptó la metodología ágil Scrum como proceso de ingeniería de software, dado que nos permite, en cada iteración, adaptarnos al contexto cambiante en el cual se sitúa el proyecto.

Palabras clave

E-commerce; Prompt; Token; LLM; GPT; Inteligencia Artificial Generativa; Retrieval Augmentation Generation; Pinecone; Embedding; Vector; Semantic Search; Vector Database; Cosine similarity; Scrum; User Story; Story Point; Docker; Integración y Despligue Continuo; Application Performance Monitoring; BERT; Python; Next JS; Azure; New Relic; OpenAI; MongoDB; Vercel; Streaming.

Glosario

Agent: en el contexto de un LLM, un agente es una entidad especializada en realizar tareas específicas interactuando con el LLM, estando diseñado para aprovechar las capacidades del mismo.

Base de datos vectorial: tipo de sistema de gestión de bases de datos diseñado específicamente para almacenar, indexar y gestionar datos en forma de vectores. Generalmente su enfoque está en la eficiencia de las consultas de búsqueda por similitud (debajo).

Bidirectional Encoder Representations from Transformers (BERT): modelo de procesamiento del lenguaje natural (NLP) open-source desarrollado por Google.

Completion: proceso de invocar al LLM con una *prompt* dada y una serie de *tools* disponibles.

Cosine similarity: medida que se utiliza para calcular el grado de similitud entre dos vectores en un espacio vectorial.

Embedding: en el procesamiento de lenguaje natural (NLP), los textos pueden ser transformados en vectores de baja dimensión utilizando *embeddings*. Estos vectores mantienen la esencia de los datos originales en términos de relaciones/similitudes, y se utilizan para la búsqueda por similitud.

Generative Pre-Trained Transformer (GPT): tipo de algoritmo de Machine Learning que se entrena sobre una gran base de datos de textos para la tarea de generación de texto.

Inteligencia Artificial Generativa (IAG): refiere a sistemas de inteligencia artificial diseñados para generar contenido una variedad de formas: texto, imágenes, música, y otros tipos de medios. Estos sistemas aprenden de grandes cantidades de datos existentes y aplican ese aprendizaje para producir estos resultados.

Large Language Model (LLM): modelos de Inteligencia Artificial basados en el aprendizaje profundo, que se entrenan con grandes volúmenes de texto. Como su nombre lo indica, son diseñados para entender, generar y trabajar con lenguaje humano.

OpenAI Functions: introducidas por OpenAI, son una característica que permite describir funciones de código y hacer que el modelo elija de forma inteligente a cuál llamar, y con qué parámetros hacerlo.

Pinecone: base de datos vectorial diseñada para aplicaciones de inteligencia artificial y aprendizaje automático, que requieren búsquedas basadas en similitudes (debajo).

Prompt: en el contexto de un modelo de lenguaje (LLM), una *prompt* se refiere a la instrucción o entrada que se le proporciona al modelo para que genere una respuesta basada en el entrenamiento previo que este ha recibido.

Prompt Engineering: disciplina enfocada en diseñar y optimizan las entradas (*prompts*) que se le dan a un modelo de lenguaje, para obtener las respuestas más precisas y relevantes posibles.

Retreival Augmentation Generation (RAG): patrón que combina la recuperación de la información de una fuente externa para mejorar la calidad y precisión de las respuestas generadas por un modelo de lenguaje.

Similarity Search: proceso que implica encontrar elementos en un conjunto de datos que sean similares a una consulta dada. Se aplica en contextos donde los datos pueden ser representados como vectores, para luego compararlos y encontrar aquellos con significados similares.

Source documents: documentos de la fuente externa que, recuperados a través de la búsqueda semántica, son utilizados por el LLM para formular su respuesta.

Token: los LLMs funcionan procesando secuencias de palabras o tokens para predecir el siguiente token más probable en una secuencia, permitiéndoles generar texto palabra por palabra.

Tools: se refiere a todas las OpenAI Functions que el modelo tiene a disposición (herramientas) para invocar y utilizar en su respuesta.

Índice general

1 Introducción

El presente documento tiene como objetivo describir el proyecto *wisello*, llevado a cabo como requisito para la obtención del título de Ingeniería en Sistemas de la Universidad ORT Uruguay.

1.1 Elección del proyecto

El proyecto final de carrera representa no solo la culminación de años de aprendizaje y desarrollo de habilidades, sino también una oportunidad única para aplicar los conocimientos adquiridos en un proyecto significativo y desafiante.

Queríamos ir aún más allá de cumplir con los requisitos académicos; buscábamos un desafío que resonara con nuestras pasiones e intereses profesionales. Esta búsqueda nos llevó a explorar una variedad de ideas, evaluando cada una por su innovación, factibilidad técnica, potencial impacto y relevancia en el mercado actual. En medio de esta etapa de descubrimiento surgió la unánime convicción de tomar esta instancia como oportunidad para emprender, con alguna idea original propia.

Después de intensas sesiones de *brainstorming*, que contaron con la valiosa participación de expertos en tecnología e innovación, finalmente nos decidimos por desarrollar *wisello*. Este proyecto capturó nuestra imaginación y entusiasmo desde el comienzo. *Wisello* emergió como una clara elección debido a su potencial para abordar una necesidad no satisfecha en el mercado del e-commerce. Además, la oportunidad de trabajar con inteligencia artificial generativa (IAG), un campo de rápido crecimiento y de gran interés para todos los miembros del equipo, fue otro factor determinante en nuestra decisión.

1.2 Descripción del proyecto

En la era digital actual -en particular, luego de la pandemia-, el e-commerce ha evolucionado rápidamente, transformando la manera en que los consumidores interactúan con las marcas y realizan sus compras. A pesar de sus numerosas ventajas, el e-commerce aún enfrenta desafíos significativos, especialmente en lo que respecta a la interacción personalizada y el asesoramiento al cliente. Estos

son los problemas que busca atacar *wisello*, que al combinar tecnologías avanzadas y una comprensión profunda de las necesidades del cliente actual, representa un salto hacia lo que creemos es el futuro del e-commerce.

El núcleo de *wisello* es un agente conversacional, diseñado para mejorar la experiencia de compra, que se integra a las tiendas en línea existentes. A diferencia de chatbots tradicionales, *wisello* integra Large Language Models (LLMs) como Generative Pre-Trained Transformer 3.5 (GPT-3.5), lo que le permite mantener diálogos naturales, comprensivos y efectivos con los usuarios. Esta tecnología permite a *wisello* responder preguntas, anticipar necesidades, ofrecer recomendaciones personalizadas y proporcionar una experiencia de compra asistida que antes estaba reservada únicamente para interacciones en tiendas físicas.

Asimismo, la innovación de *wisello* radica en su capacidad para ser informado con el catálogo de productos de las tiendas y con conocimientos específicos del rubro. Esto posibilita ofrecer asesoramiento personalizado y relevante, basado en el entendimiento del contexto y las preferencias del cliente. Además, el hecho de que *wisello* pueda integrarse fácilmente en diferentes plataformas de e-commerce, le da a los vendedores una poderosa herramienta para mejorar la interacción con sus clientes y, en última instancia, aumentar sus ventas y la satisfacción del cliente.

El desarrollo de *wisello* ha sido un proceso meticuloso y desafiante, que ha implicado la aplicación de conocimientos avanzados en IAG e ingeniería de software, sumadas a una comprensión profunda del campo del e-commerce y las dinámicas de compra del consumidor. A través de iteraciones, pruebas y feedback constante, el equipo ha trabajado para asegurar que *wisello* cumpla con los estándares técnicos más altos, y ofrezca una experiencia de usuario excepcional.

1.3 Objetivos

En los inicios del proyecto, realizamos una reflexión profunda y colaborativa para establecer y definir objetivos claros. Este proceso de definición de metas fue esencial para garantizar un enfoque unificado y coherente frente al desafío que se tenía por delante. A partir de este ejercicio, se desprendieron una serie de objetivos claros y definidos que servirían como pilares fundamentales para guiar todas las etapas del proyecto. Estos objetivos se han establecido con la intención de cumplir con las expectativas académicas y técnicas, y garantizar que *wisello* se posicione como una solución innovadora y eficaz en el panorama del e-commerce.

Los objetivos se dividen en tres categorías principales: académicos, de proceso y

de producto. Cada categoría aborda aspectos diferentes pero igualmente cruciales del proyecto. Los objetivos académicos se centran en el aprendizaje y la aplicación de conocimientos, los objetivos de proceso se relacionan con la gestión y ejecución eficiente del proyecto, y los objetivos de producto se enfocan en las características y el impacto de la inserción de *wisello* en el mercado.

A continuación, se detalla cada uno de estos objetivos, explicando cómo contribuyen al éxito general del proyecto y de qué manera se alinean con la visión del equipo para *wisello*.

1.3.1 Objetivos académicos

OA1: Aplicación de conocimientos obtenidos a lo largo de la carrera relacionados a la Ingeniería de Software en un proyecto de contexto real, con usuarios reales y salida al mercado.

Uno de los principales objetivos académicos propuestos por el equipo fue demostrar su capacidad para aplicar conceptos avanzados de ingeniería de software en un proyecto real y desafiante. Para el cumplimiento de este objetivo, se pretende que el equipo aplique conocimientos adquiridos entorno a: la ingeniería de requerimientos, un marco de gestión ágil para el proyecto, patrones de diseño para la arquitectura de la solución, tácticas y patrones para cumplir con los distintos atributos de calidad especificados para el proyecto, un proceso de desarrollo, gestión de riesgos, gestión de la configuración y aseguramiento de la calidad.

OA2: Conocer en amplitud el estado del arte de las diferentes tecnologías, frameworks, buenas prácticas, principios y patrones utilizados en el desarrollo de aplicaciones de LLMs.

Al ser tan reciente el desarrollo de aplicaciones que integran LLMs, el estado del arte en este ámbito está en constante evolución. A medida que la tecnología continúe madurando, surgirán nuevas prácticas, frameworks y herramientas, impulsando aún más la innovación. Por ende, un objetivo que nos planteamos en esta línea fue conocer y aprender acerca del estado del arte actual para el desarrollo de aplicaciones impulsadas por LLMs. Esto abarca: Langchain framework, *prompt engineering*, alucinaciones, patrones en aplicaciones conversacionales, bases de datos vectoriales, y testing (evaluación) de la respuesta.

OA3: Contribuir al cuerpo académico en la aplicación de Inteligencia Artificial Generativa (LLMs) en soluciones a problemas reales.

Buscamos contribuir al cuerpo académico sobre la aplicación de LLMs en siste-

mas que responden a problemas reales, para lograr *soluciones reales*. Relacionado con el objetivo anterior, este objetivo propone exponer todo el conocimiento adquirido entorno al desarrollo de aplicaciones impulsadas por LLMs, en este informe, para que cualquiera que quiera consultarlo luego pueda hacerlo.

OA4: Lograr un excelente proyecto a nivel académico, con nota final 95 o más.

Una prioridad fundamental para nosotros como equipo fue ejecutar un proyecto que tenga muy buenos resultados en términos académicos. Por ello, nos fijamos como objetivo académico obtener como nota final del proyecto 95 o más.

1.3.2 Objetivos del proceso

OPC1: Aplicar un marco de gestión ágil.

Un objetivo fijado a nivel de proceso para el proyecto fue la utilización de un marco de gestión ágil, que nos brinde al equipo la flexibilidad para responder con agilidad a los cambios en las especificaciones del proyecto.

OPC2: Llevar métricas tanto de gestión como de producto, para el análisis del proceso.

Con el fin de llevar a cabo una gestión eficaz y eficiente del proyecto, el equipo acordó llevar métricas tanto de gestión como de producto. Se decidió trackear las siguientes métricas: Velocidad, Sprint Burndown, Cantidad de Bugs, y Retrabajo. A través de las mismas, el equipo podría mejorar su proceso de gestión: mejorar su estimación y planificación, tomar mejores decisiones, identificar áreas de mejora, y así mejorar continuamente.

OPC3: Aprender y reflexionar luego de cada iteración sobre el proceso con el fin de mejorar continuamente los procesos de gestión y desarrollo.

El equipo se planteó como objetivo de proceso realizar, de forma obligatoria, una Retrospectiva al final de cada iteración del producto, de forma de poder reflexionar y extraer aprendizajes valiosos en cada etapa. Tras la misma, se deberán trazar conclusiones sobre lo que se puede mejorar para la siguiente iteración, con el fin de mejorar continuamente el proceso de gestión y desarrollo.

1.3.3 Objetivos del producto

OPD1: Lograr un producto de calidad que en la medida de lo posible, tenga salida al mercado con clientes reales.

Se desea construir un producto de calidad que pueda ser utilizado por usuarios reales, en un ambiente de producción lanzado al público general. Si es posible, se desea conseguir clientes que estén dispuestos a pagar por él, pues significaría una validación de su valor y beneficio.

OPD2: Mejorar la búsqueda en e-commerce, y ofrecer asesoramiento de calidad al usuario, para que se sienta más incentivado a comprar.

Se pretende que la solución permita a los usuarios sentirse más asistidos y asesorados durante su compra, ayudándolos así a encontrar su **producto ideal** y mejorando su satisfacción. La solución imita/reemplaza el rol de un vendedor humano en tienda física, ofreciendo asesoramiento de calidad, inmediato, disponible 24/7, en formato desestructurado.

OPD3: Integración *seamless*.

wisello debe poder integrarse sin necesidad de desarrollo web extra con cualquier plataforma de e-commerce que así lo desee. Para ello, se debe definir un mecanismo de integración que permita esto.

2 Marco metodológico

2.1 Contexto del proyecto y equipo

Repasaremos ciertas características que creemos fundamentales para entender la ideación y ejecución de *wisello*. La elección del proyecto se dio en una etapa muy temprana del desarrollo de tecnologías de Large Language Models (incluso antes del lanzamiento de la primera API de ChatGPT [1], `gpt-3.5-turbo`). En este contexto, el desarrollo tecnológico de la herramienta se presentó como uno de los principales desafíos. En paralelo, estuvo el enorme desafío que supone emprender y llevar nuestra tecnología a un mercado real.

Los integrantes del equipo teníamos experiencia profesional en desarrollo de software y contábamos con la trayectoria académica y práctica que nos brinda la formación de Ingeniería en Sistemas [2]. En lo que tiene que ver con desarrollo de Inteligencia Artificial en general y aplicaciones de Large Language Models en particular, la experiencia era más restringida. Además de la experiencia académica que nos brindaron cursos como el de Inteligencia artificial en Universidad ORT, Marcelo, Mateo y Andrés tenían experiencia profesional limitada en modelos de IA en producción, y Marcelo cursó una materia de Natural Language Understanding [3] en Stanford School of Engineering Online en los primeros meses de 2023. En cuanto a experiencia en emprendimientos, en 2022, Mateo formó parte del Centro de Innovación y Emprendimientos (CIE) como consultor de ingeniería de software [4].

En síntesis, el proyecto se inició en un contexto marcado por el doble desafío de desarrollar un sistema basado en tecnologías emergentes a la vez que se daban los primeros pasos en el emprendedurismo para la mayoría de los integrantes del equipo. Como equipo, tomamos esta aparente dificultad como estímulo desafiante y provocador; como una oportunidad extraordinaria para enfrentarnos a problemas tecnológicos recientes a la vez que lo hacían los equipos de ingeniería de vanguardia en el mundo, apoyándonos a su vez en la versatilidad que supone un proyecto emprendedor y la posibilidad de interactuar con un mercado que valide la viabilidad del desarrollo de un nuevo producto.

Desafío de definir el proyecto en un contexto desconocido

Las dificultades planteadas anteriormente se materializaron inicialmente en la elección y definición del proyecto. En este contexto, tomamos dos determinaciones:

1. Investigar extensamente los dominios del e-commerce y el desarrollo de aplicaciones de IA.
2. Utilizar un marco de desarrollo flexible que nos permita ir definiendo detalles de la ideación e implementación sobre la marcha.

Asimismo, entendimos que íbamos a tener que fijar hipótesis de trabajo para poner en marcha el proyecto, pero a sabiendas de que muy probablemente estas irían cambiando a lo largo del tiempo, a medida que fuéramos conociendo más las necesidades de los usuarios, del mercado, y las características tecnológicas específicas de las aplicaciones construidas con LLMs.

2.2 Lean Startup

Como metodología, para los dos primeros meses de ideación y validación del proyecto, su idea y solución, utilizamos **Lean Startup**. La metodología se basa en una serie de principios claves, que tienen su raíz en **Lean Manufacturing**: *"the shrinking of batch sizes, just-in-time production (...), and an acceleration of cycle times"* [5].

El core de la metodología es su **Build-Measure-Learn feedback loop**, que propone lanzar un producto al mercado rápidamente para recolectar *customer feedback*. Esto se debe hacer a través de un Minimum Viable Product (MVP), para probarlo con usuarios, incorporar aprendizajes que validen o no las hipótesis experimentadas, y luego utilizar este ***validated learning*** para guiar las decisiones, tanto de *go-forward* o *fail-fast*. Como resultado, esto nos permite: *"drop wasteful practices, make changes to a product, pivot to another idea entirely, or even create new business models"* [6]

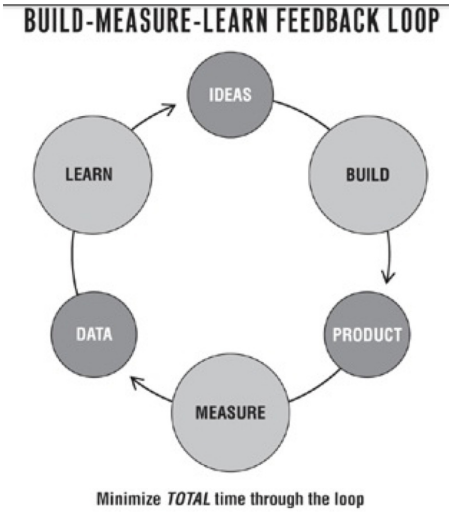


Figura 2.1: *Build-Measure-Learn (BML) loop*

La primera iteración del *loop* se realizó con prototipos no funcionales, luego de hacer un *brainstorming* sobre posibles problemas disparadores para realizar el proyecto final. Se trabajó sobre la hipótesis de que *muchísimas de las tiendas online tienen un chat, pero que comienza una conversación asíncrona. Es necesario un vendedor síncrono para las mismas*:

Figura 2.2: Prototipos iniciales

Recibimos feedback cualitativo de los prototipos por parte de los idóneos entrevistados durante la validación del problema, que destacó la oportunidad de innovar en el área, por la necesidad que tiene el usuario de ser acompañado y guiado en su experiencia de compra (por más detalle, ver anexo Lean Startup 15.10).

Las siguientes iteraciones del *loop* se centraron en construir un MVP funcional para Wikimúsculos, empresa líder en suplementos deportivos, con quien trabajamos en esta etapa. La hipótesis a testear fue: *los usuarios de Wikimúsculos se beneficiarían de asesoramiento y recomendación nutricional personalizada durante su compra, y Wikimúsculos lo podría otorgar de forma automatizada*. Para ello, construimos un asistente basado en IA, capaz de responder preguntas sobre nutrición y recomendar productos de Wikimúsculos, en base a las necesidades expresadas por los usuarios. A través de un ida y vuelta con la empresa, fuimos mejorando las respuestas del asistente (ejecutando así varios ciclos del *loop*). Como aprendizajes, el feedback recibido nos permitió validar la factibilidad del proyecto, tanto desde un punto de vista del negocio (la capacidad del asistente para dar buenas recomendaciones), como desde lo tecnológico (existen las tecnologías para lograrlo). Incluso, con este, se investigó y definió el *stack* de tecnologías que luego sería utilizado para el producto. Por evidencia, ver anexo Lean Startup 15.10.

Dado que no habíamos tenido mucha validación con usuarios **finales** aún, comenzamos a trabajar con Plexo, una empresa que provee guías informativas para el viaje de arquitectura. Se realizó un nuevo ciclo del *loop*, construyendo un asistente que sea capaz de responder preguntas sobre las guías. El MVP fue distribuido a todos los arquitectos viajeros para su utilización. La hipótesis a probar en este caso fue: *los usuarios de las guías de Plexo se beneficiarían de un asistente conocedor de las guías en tanto podrían buscar más rápido en las mismas*. En esta etapa, el feedback recolectado fue tanto **cuantitativo** (métricas de uso) así como **cualitativo** (conversaciones de los usuarios con el asistente). Como principales aprendizajes del mismo, se observó un muy buen *engagement* por parte de los usuarios, pero además se reconoció el valor del asistente en la personalización que esté proveía sobre las guías (no solamente en la eficiencia de búsqueda).

Se ahonda más en el desarrollo, resultados y métricas de ambos MVP (Wikimúsculos y Plexo) en sus respectivas secciones expuestas en la validación del problema: MVP Wikimúsculos 3.2.2.2 y MVP Plexo 3.2.2.3.

Esta metodología nos facilitó navegar el proceso de creación del producto, en un contexto de incertidumbre, a través de un enfoque científico que nos permitió probar hipótesis para obtener **aprendizaje de validación**, y luego con este, guíar nuestras decisiones. Así, nos obligó a estar en permanente contacto con usuarios, en cada *loop* del proceso.

2.3 Scrum

Tomamos Scrum como metodología de gestión ágil. La Lic. Jimena Saavedra (Certified Agile Coach) durante la primera revisión nos recomendó adaptar el marco, pues *"un marco tan apegado a la guía como Scrum porque nos pide un montón de cosas que quizás en el tiempo no sea mantenible"*. Por ende, realizamos algunas adaptaciones al marco que nos permitieron aumentar la flexibilidad y adaptarnos mejor a las necesidades específicas del desarrollo de *wisello*. Scrum nos ayudó a entregar valor de forma incremental en ciclos iterativos que incluyeron la retroalimentación de usuarios y experimentación. Asimismo, las ceremonias de Scrum nos permitieron fortalecer la colaboración e identificar puntos de mejora para ir refinando cada vez más el trabajo en equipo.

Definimos Sprints con duración de dos semanas -en algunos casos se modificó a tres semanas debido a coyunturas particulares-, para marcar los tiempos del desarrollo iterativo de *wisello*.

Decidimos hacer algunas adaptaciones al marco de Scrum. En lo que refiere al Scrum Team [7], que se compone por Product Owner, Scrum Master y Desarrolladores, decidimos que quienes tomaran los dos primeros roles también se desempeñarían como desarrolladores. La alternativa a esto hubiera sido contar solamente con dos desarrolladores y la carga hubiera quedado muy desbalanceada, comprometiendo asimismo el alcance al que pudiera llegar el proyecto en términos de desarrollo de funcionalidades.

La creación de User Stories en el Product Backlog se iba dando por parte del Product Owner y también del resto del equipo, a medida que se iban identificando necesidades, bugs, o cualquier otra tarea nueva. Durante las Sprint Planning Meeting, decidíamos qué User Stories incluir en el próximo sprint y repasábamos las tasks del Backlog para corroborar que estuvieran correctamente priorizadas, práctica que se conoce como *Backlog Refinement*.

Véase 9.1 para más detalles de la implementación de Scrum en *wisello*.

3 Problema

3.1 Descripción del problema

3.1.1 Breve introducción al problema

"Hoy en día hay mucha más oferta que hace 9 o 10 años". Así comienza su respuesta Leonardo Álvarez, cofundador de Fenicio eCommerce (plataforma líder con +500 e-commerces en el país), tras preguntarle acerca del panorama actual de la industria del e-commerce en Uruguay.

Frente a la creciente oferta de artículos en tiendas de e-commerce, cada uno de ellos con sus propias especificaciones, información de uso y características particulares, las tiendas online se enfrentan al desafío de lograr exhibir y presentar la totalidad de sus catálogos en sus sitios web, de forma que el cliente se sienta *cómodo, instruido e incentivado* a realizar una compra.

Resultaría inviable, o al menos poco inteligente desde un punto de vista estratégico, prescindir de esta amplia oferta, dado que es justamente esta variedad de productos la que atrae el consumo hacia el comercio electrónico; siendo esta una de sus principales ventajas competitivas. Según un informe publicado por la pasarela de pagos GreenPay, *"la variedad, en las plataformas online, funciona como factor de democratización del comercio, ya que hace emerger marcas disruptivas que se vuelven más accesibles, superando ampliamente a la oferta que ofrece un comercio offline"* [8].

3.1.2 Un problema con muchos ejes

La falta de incentivo, confianza o estímulo por parte del usuario para finalizar su proceso de compra, nace de varios motivos.

A lo largo de este capítulo se mencionarán opiniones de expertos acerca de cuáles son las mejores técnicas, en su experiencia, para mejorar el **asesoramiento** brindado al usuario, como método a través del cual la tienda busca afianzar la convicción del usuario por su compra, y evitar que este la abandone.

3.1.2.1 Primer eje: puesta y carga de información al e-commerce

El primer eje o motivación del problema en cuestión es el esfuerzo que implica, por parte de una tienda online, proveer información completa de cada producto al cliente. Según Leonardo Álvarez, *"lo que mejor funciona es poner las fichas técnicas de los productos, lo más completas posibles"*. El argumento de Leonardo expresa que, al fin y al cabo, el e-commerce no es una plataforma mágica donde los productos se venden solos; al igual que en una tienda física, hay que poner precios, fotos, contratar alguien que venda: *"siempre es difícil, pero es necesario"*, afirma Leonardo.

3.1.2.2 Segundo eje: el e-commerce como *ventana* al mundo físico

Por otro lado, aparece otro eje del problema relacionado a la falta de confianza por parte de los clientes para comprar cosas que no conocen por la web; fenómeno que se ve potenciado cuando el artículo es particularmente un producto caro. En estos casos el e-commerce cumple uno de los dos siguientes roles, afirma Lucía Puentes, encargada de e-commerce en Divino. Por un lado, o bien actúa como *ventana* que atrae a la tienda física, o por el otro, concentra compras recurrentes, o en su sentido más amplio, compras conocidas, en donde el usuario ingresa al sitio conociendo cuál será su compra (lo cual engloba también una compra recurrente). En el caso de Divino, comenta Lucía que proporcionando asesoramiento a través de la *omnicanalidad* (incluyendo líneas teléfonicas, e-mail, y Whatsapp), se pudieron ver buenos resultados.

3.1.2.3 Tercer eje: barreras a la compra online

Finalmente, y en relación al ángulo del problema al cual dirigiremos nuestra atención, este comprende la falta de *incentivo*, por parte del cliente, para finalizar su compra, o siquiera navegar por la página. Si el cliente no está realmente convencido de su compra, los riesgos de que este la abandone en algún punto de su recorrido virtual son altos. Por ende, Lucía nos cuenta que existen muchas barreras con las que romper; problemática que se ve especialmente exacerbada considerando las exigencias de un núcleo de clientes que migró al canal digital desde la pandemia, y que exige experiencias mucho más cercanas a las que tendría en tienda física (donde solía contar con roles o prestaciones como vendedores, cajeros y probadores).

3.1.3 Carritos abandonados

"Si algo pasa cuando pone la tarjeta, si la dirección no le marca exacto su casa, si no se le aplica el descuento: el comprador pierde la confianza y se va", nos cuenta Lucía, encargada de e-commerce en Divino. Estas son solo algunas de las barreras a la compra que componen el tercer eje del problema que abordamos anteriormente.

Estas barreras resultan en los *carritos abandonados* de los usuarios: un fenómeno altamente prevalente en el mundo del e-commerce, y con el cual estas tiendas luchan día a día. En América Latina, el procentaje de carritos abandonados en e-commerce ronda el **83,4 %** de los casos [9], con una tendencia al crecimiento.

La clave parece ser la información, nos cuenta Lucía, indicando que luego de haber agregado el asesoramiento vía Whatsapp, se vieron buenos resultados. En línea con esto, Leonardo Álvarez afirma que siempre es necesario *"contestar a las dudas de los usuarios. A los que le va mejor es a los que hacen mejor esto"*.

3.1.4 La importancia del asesoramiento

Según un informe publicado por la CACE (Cámara Argentina de Comercio Electrónico) en 2020, la *falta de un asesoramiento online en vivo* se posicionó como la cuarta desventaja más resaltada por los compradores online, con una incidencia del 24 % entre la población entrevistada [10]. Si comparamos con años anteriores (2018 y 2019), este valor aumentó 2 %, evidenciando el efecto de la pandemia (2020) sobre el e-commerce: disparando una mayor necesidad de asesoramiento, fenómeno que ya se analizó anteriormente. En este mismo informe se expresa que dicho aumento se da por los denominados compradores *Post-pandemia* y compradores *Ocasionales*, lo cual explica este argumento, en tanto se trata de compradores incipientes (ex-offline) que se acercan al e-commerce por necesidad, con costumbres de un paradigma de compra físico. El contexto post-pandemia en el que nos encontramos nos da lugar para desarrollar, aún más, el asesoramiento en el rubro, y por tanto, una oportunidad para potenciarlo.

Además, se expresa en el informe la necesidad del usuario de que el asesoramiento sea *en vivo*, resaltando su deseo de sincronicidad (a diferencia del asesoramiento asincrónico, como puede ser el Email, Instagram, o Whatsapp, que cuentan con tiempos de respuesta elevados).

Entonces, ¿por qué es importante el asesoramiento en vivo? Las razones son varias. Por un lado, un punto en el que ya hemos ahondado: la oferta. Más oferta

implica más opciones, pero situados en un contexto en donde no tenemos cómo ni con quién evacuar nuestras dudas, las probabilidades de abandono de la compra son altas. Por otro lado, existen productos que requieren de una investigación o un conocimiento *experto*, como pueden ser suplementos deportivos, artículos de belleza o higiene, o electrodomésticos, entre otros. Es por ello que las tiendas e-commerce, en los últimos años, han hecho un gran esfuerzo por incorporar métodos para asesorar al usuario durante su compra online, intentando así derribar esta barrera de disponibilidad y facilidad de acceso a la información sobre los productos y al asesoramiento en general, que obstaculiza las ventas. Asimismo, Juan Teba, de e-commerce en Farmashop, nos cuenta que integraron Whatsapp a su e-commerce con idóneos de farmacia a disposición.

Lucía, de Divino, nos afirma que *"es muy importante asesorar al usuario en el proceso. Si tuviéramos más seguimiento, sería mucho más fácil"*.

3.1.5 Comportamiento digital del comprador online uruguayo

Quizás uno de los *insights* más interesantes recabados durante nuestra exploración del problema fue el comportamiento digital del comprador uruguayo en e-commerce, es decir, las interacciones que va dejando detrás al navegar en una tienda online.

El *lenguaje digital* del comprador online uruguayo se caracteriza por una controversia entre la **impaciencia** y el ser **analítico**, nos cuentan Juan Ignacio Antognazza, CEO de Acra Equipamientos, y Leonardo Álvarez, co-fundador de Fenicio eCommerce. El uruguayo típico investiga muchas opciones previo a comprar, no se decide fácil y resulta ser muy indeciso bajo su reflexión. Es necesario *"perseguir al comprador"*, nos cuenta Leonardo, porque necesita de ver varias veces el mismo producto en muchos lugares para convencerse de su compra. A su vez, matizado por la era moderna de la inmediatez, el comprador demanda que sus dudas sean rápidamente resueltas; comentario que Juan Ignacio ejemplifica con Mercado Libre, donde los vendedores, para mantener su reputación, deben responder las dudas en un margen de tres minutos.

Por ende, el comprador uruguayo, al ser en su esencia un comprador analítico, creemos que puede valerse de un asesoramiento más profundo y personalizado, que haga que se sienta más *cómodo, instruido e incentivado* para comprar, acompañándolo en su búsqueda desde la información, la ayuda y la facilitación. Considerando además que, en muchos casos, el uruguayo tiene voluntad de comprar

más en medios digitales que en medios físicos, lo cual lo hace más atractivo aún. Juan Teba, de e-commerce en Farmashop, nos cuenta que en Farmashop *"el ticket promedio de compra online es el doble que el de tienda física"*.

3.1.6 Alternativas actuales y sus falencias

Los métodos que se utilizan, hoy, para ofrecer asesoraramiento al usuario de e-commerce varían desde:

- Recomendaciones personalizadas de productos.
- Chatbots tradicionales, con respuestas fijas ante caminos fijos (estructura de árbol).
- WhatsApp, email, redes sociales u otras mensajerías como formas de asesoramiento personalizado.
- Alternativas inteligentes en formato plataformas self-service.

Viendo estos métodos, se hace evidente la falta de un método de asesoramiento que integre todas las siguientes características, que hoy encontramos únicamente de forma aislada en las alternativas disponibles en el mercado:

- Sea **personalizado** para *ese usuario, en ese momento, en esa situación en particular*. En e-commerce, esto se conoce bajo el termino de *hiperpersonalización*, y refiere al concepto de considerar al usuario como una persona única, cuyas realidades cambian día a día, por lo que se busca tener una interacción única, un *"1-1"*, construido en el momento para el usuario.
- Sea de **calidad**, y con esto nos referimos no solamente a la utilidad de la información proporcionada durante el asesoramiento, sino también a la **existencia** de una respuesta. Por ende, el universo de respuesta y asesoramiento debe ser ilimitado, como lo tendría un asesor humano, sin caminos o respuestas fijas, a diferencia de los Chatbots tradicionales que encontramos hoy en el mercado.
- Esté **disponible** 24/7, fuera y dentro de horarios laborales.
- Sea **instantáneo**, sin necesidad de tener que interrumpir el proceso de compra por la falta de asesoramiento, lo cual se conecta con el punto anterior.

3.1.7 Primera hipótesis

Exponemos a continuación nuestra primera hipótesis del problema, previa a la validación del problema: *Existe una falta de información y asesoramiento en tiendas e-commerce en Uruguay.*

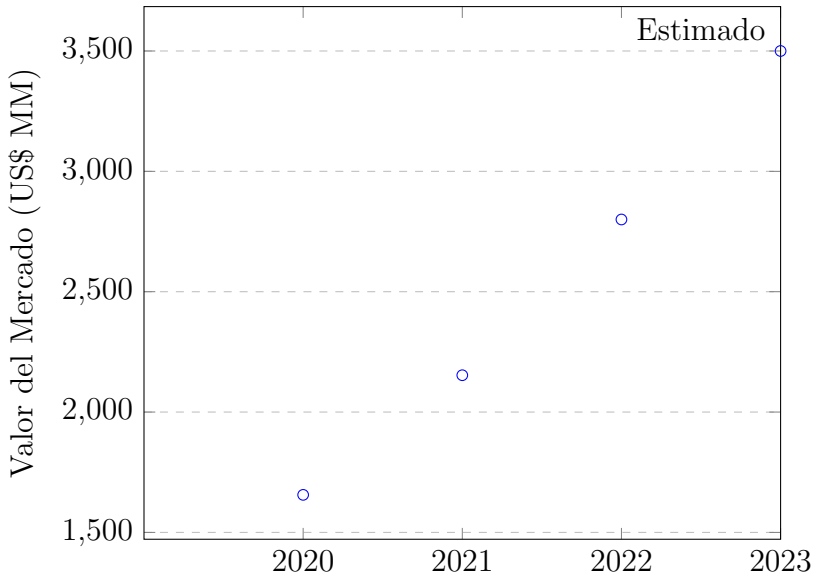
3.2 Validación del problema

3.2.1 Validación secundaria: investigación

En un primer momento, decidimos llevar a cabo una investigación secundaria acerca de la industria del e-commerce, mayoritariamente enfocados en Uruguay, para validar que el tamaño del mercado sea lo suficientemente atractivo para introducirnos dentro de este, identificar tendencias clave y fuerzas importantes del mercado.

3.2.1.1 La industria del e-commerce en números

Cifras presentadas por AMI (Americas Market Intelligence) reflejan que el mercado del e-commerce en Uruguay ha experimentado un crecimiento constante y elevado en los últimos años. Marcado por su pico más alto de crecimiento en la pandemia del 2020, año en que se expandió en un 40 %, este crecimiento se sostuvo en los años 2021 y 2022, cuando creció 30 % YoY. Para el 2023, se estimó un crecimiento del 27 % [11].



Evolución del valor del mercado de e-commerce en Uruguay en los últimos años.

Fuente: AMI (Americas Market Intelligence), 2023 [11]

La región Latinoamericana acompasó este crecimiento, con tasas de crecimiento del 36 % y 39 % en 2021 y 2022 respectivamente, esperándose así que esta crecimiento se mantenga, de forma sostenida, con un 21 % TCAC (tasa de crecimiento anual compuesta) hasta 2026 [12].

En el caso de Uruguay, dicho crecimiento se explica, en gran parte, por la adopción digital y la creciente confianza de los consumidores para comprar online. En 2022, 91 % de los hogares uruguayos disponían de conexión a Internet [13], y en este mismo año, el 57 % de los uruguayos realizó una compra online, sin ser delivery o transporte [14]. Además, según AMI, el gasto anual promedio de un uruguayo en e-commerce en 2022 fue de **791 US$** [11], una cifra alta considerando que más de la mitad de la población compró productos online ese mismo año.

En lo que refiere a las empresas uruguayas, según un informe presentado por FACTUM en 2022, el 87 % de estas venden a través de internet, y en particular, un 92 % lo hace a través de su propio sitio. Además, el 69 % de las empresas cuentan con un área específica de e-commerce, lo cual demuestra la relevancia que están cobrando los medios digitales en muchas industrias [15].

En resumen, se confirma la atractividad del mercado de e-commerce, tanto en Uruguay como en la región. Tras haber sido favorecido por la pandemia y el confinamiento, la industria del e-commerce logró sostener su crecimiento años

después, y los datos muestran que así lo hará por varios años más. El e-commerce hoy representa un canal de facturación muy importante para muchas empresas. Según la CEDU (Cámara de la economía digital del Uruguay), en 2020, los sitios online propios representaron un 40 % de la facturación en ventas [16]. Si bien es probable que ese número no se mantenga en niveles post-pandemia, evidencia la importancia que tiene este medio en términos de facturación, lo que conlleva que las empresas estarían dispuestas a invertir para desarrollar su e-commerce, con el fin de dominar el sector digital y lograr verdaderas innovaciones que lo potencien aún más.

3.2.1.2 Fuerzas de la industria

Si pensamos en e-commerce, tanto en Uruguay como en Latinoamérica, resulta imposible no pensar en Mercado Libre como una fuerza dominante y establecida en el mercado. Con 8.3 billones de visitas y 34.7 % del tráfico de marketplaces entre 2019-2021, se posiciona como el marketplace más grande en la región. Sin embargo, entre 2019 y 2021, solamente hubo un 6.4 % de marketplaces que crecieron en tráfico [17] (ver anexo 15.2).

Un informe de CACE (Cámara Argentina de Comercio Electrónico) presentado en 2021 posiciona en Argentina a los buscadores tradicionales de páginas webs propietarias como el principal medio de búsqueda para buscar información de productos, tanto con intención de comprar online o offline. Son utilizados por el 68 % de compradores previo a comprar offline, y por el 44 % de ellos al comprar online [10].

3.2.1.3 Productos y servicios sustitutos

Como principal producto sustituto, encontramos la tienda física. Las ventajas de este sustituto se constituyen en contraposición con las desventajas del e-commerce, que según un informe presentado por la CACE son las siguientes [10]:

Desventaja e-commerce	**Ventaja tienda física**
No se puede ver el producto	Se puede ver el producto
Costos de envío	No hay costos de envío
Demoras en la entrega	Entrega inmediata
No hay asesoramiento online en vivo	Asesoramiento del vendedor
Desconfianza en el sitio del vendedor	Se confía en la tienda física (POS, persona)

Tabla 3.1: Ventajas de la tienda física y desventajas del e-commerce

Sin embargo, y en relación a la tienda física como producto sustituto, las estadísticas revelan la existencia del patrón ROPO (*Research Online, Purchase Offline*). Por ejemplo, en Argentina, 95 % de los compradores offline buscó información online, previo a su compra offline [10]. Es curioso cómo, en este sentido, el e-commerce se infiltra en la compra física, en casi todos los casos. Esto nos hace pensar que las barreras que presenta el e-commerce en términos de confianza, logística, precios de envío y asesoramiento, van a poder ser derribadas con innovaciones, y el mundo va a tender hacia un paradigma de compra completamente digital.

3.2.1.4 Proveedores y actores en la cadena de valor

En los últimos años, ha habido una *"convergencia de plataformas e iniciativas de pagos en tiempo real"* [18], fenómeno dentro del cual se ha destacado la plataforma de infraestructura financiera para negocios *Stripe*. Esta capacidad de procesar pagos en tiempo real acompaña el crecimiento del e-commerce, pues permite *"recibir pagos en cualquier momento y desde cualquier lugar"* [18]. Además, estos proveedores de pagos en línea permiten garantizar pagos seguros y confiables para los clientes, construyendo así la confianza alrededor del pago online y aportando la continua adopción digital del e-commerce.

En lo que refiere a otros actores de la cadena de valor, encontramos empresas emergentes con innovaciones que intentan facilitar la logística detrás de una compra. Por ejemplo, el uso de análisis de datos en tiempo real para la planificación de demanda, inventario y gestión de picos de órdenes, lo cual permite *"aumentar la capacidad predictiva y optimizar el proceso de la gestión logística"* [19]. Por otro lado, el uso de sistemas de optimización, con el fin de mejorar *delivery performance*, a través de algoritmos de planificación de rutas dinámicas, considerando factores como el el tráfico, tiempo o apertura/cierre de carreteras [19].

Esto nos demuestra que las empresas involucradas en la cadena de valor de la

industria del e-commerce también son exitosas, dado que cuentan con el crecimiento incesante que su industria cliente ha tenido. A su vez, se confirma la disposición que existe, en el e-commerce, para colaborar con proveedores y servicios con la finalidad de seguir mejorando, día a día, la experiencia de sus usuarios.

3.2.1.5 Fuerzas macroeconómicas - Inversión

Tras un sustancial aumento de la demanda en e-commerce a partir de la pandemia, indudablemente esta se coloca como una fuerza macroeconómica que ha impulsado (o acelerado) el crecimiento de la industria.

Según el informe *2021 Retail CFO Outlook Survey*, del 71 % de los CFOs que afirmaron su intención de invertir en IT en 2021 (post-pandemia), el 64 % de ellos aclamó su deseo de querer hacerlo en e-commerce, pues *"los canales de venta online dejaron de ser una opción, y pasaron a ser una necesidad"* [20].

Si nos concentramos en la inversión en e-commerce en Latinoamérica, según *The Latam Tech Report* publicado por Latitud, esta tuvo su máximo pico de inversión durante los años agudos de la pandemia, con US$ 3.96B de inversión en 2021 [21].

Analizando el informe *State Of Retail Tech Q1'21 Report* de CB Insights (plataforma líder de análisis de datos e inteligencia del mercado), en 2021, pudimos ver que se aconseja a inversores de e-commerce: *"Busque (...) experiencias de comercio electrónico más personalizadas. Para mantenerse al día con las expectativas de los compradores en línea, la inversión se dirigirá a empresas de tecnología que personalizan los resultados de búsqueda (...)"* [22].

En resumen, el panorama de inversión para la industria de e-commerce, tanto a nivel regional como global, parece ser prometedor, y estar alineado con nuestra visión de proveer experiencias de compra al usuario más *asesoradas, personalizadas y de calidad*.

3.2.1.6 Tendencias clave

En cuanto a las tendencias socioculturales, estas se pueden relacionar con aspectos ya mencionados durante el análisis. La creciente adopción y uso de tecnologías, así como el perfil analítico e indeciso del comprador uruguayo forman parte de estas.

En lo que refiere a tendencias tecnológicas, específicamente en e-commerce, estas apuntan hacia la *hiperpersonalización*. Esta permite entender mejor la situación y los sentimientos de los clientes, y así comunicarse mejor con los mismos, con el fin de retener su fidelidad.

Por otro lado, las tendencias normalizadoras apuntan hacia la creciente atención a la protección de datos personales. EL GDPR en la Unión Europea, o la ley N° 18831 de protección de datos personales en Uruguay. Esto implica consciencia y cumplimiento legal por parte de las empresas entorno a esta problemática.

Finalmente, si abordamos las tendencias socioeconómicas, es importante destacar los cambios en los patrones de consumo debido a la pandemia, incrementando las compras en línea, y con ello, la inversión y el crecimiento económico del sector.

3.2.1.7 Desafíos ante las fuerzas del mercado

Tanto la competencia como las necesidades de los consumidores evolucionan, y eso conlleva la aparición de varios desafíos en el ámbito.

Dada la creciente competencia en e-commerce, se ha vuelto *"esencial para las empresas ofrecer experiencias personalizadas a sus clientes"* [23], lo cual implica adoptar los avances tecnológicos y mantenerse a la vanguardia en pos de mantenerse competitivo.

Otro desafío importante es mantener la privacidad de los datos de los consumidores, aún así cuando estos datos son utilizados para dar mejores recomendaciones para ellos mismos. El e-commerce posee información sensitiva, tanto personal como financiera, por lo que *"existe una creciente preocupación por la protección de datos personales y financieros"* [23]. Esto explica la falta de confianza en las transacciones en línea. Sin embargo, existe por parte de los compradores una necesidad por contar con una amplia variedad de opciones de pago, lo cual exacerba aún más los desafíos relacionados al pago, en tanto el espectro de potenciales peligros se agranda.

3.2.2 Validación primaria

3.2.2.1 Herramientas de validación

Como primer herramienta de validación, creamos un **Mapa de Competencia** (ver anexo 15.1.1), comparando las distintas formas de asesoramiento que existen,

hoy en día, en e-commerce. Se compararon una serie de características para cada una, desde integración a la página web, presentación de métricas de uso, universo de respuesta, nivel de personalización, disponibilidad y precio. Tal como se menciona en 3.1.6, utilizando esta herramienta nos percatamos de que no existe ningún método de asesoramiento que integre todas estas características juntas. Por ende, ya teníamos una base desde la cual partir: **nuestra solución tenía que integrar todas estas características esenciales para un asesoramiento exitoso**.

Como segunda herramienta de validación, construimos una **Tabla de Hipótesis** (ver anexo 15.1.2). Se atacó el problema desde algunas de sus distintas y posibles perspectivas:

- La falta de información acerca de un producto.
- La falta de asesoramiento durante compras de productos muy personales.
- El costo que implica dedicar recursos humanos para ofrecer asesoramiento personalizado.
- El contrapropósito de no recibir el asesoramiento deseado con un Chatbot tradicional, y frustrarse aún más.

La combinación de estos *sub-problemas*, hace evidente que es necesario lograr **automatizar la personalización**, pues ambos los compradores exigen mejor asesoramiento, así como las empresas necesitan reducir los costos de ofrecerlo. Por esta misma razón, concluimos que para la solución será necesario recurrir a la Inteligencia Artificial. Según Enrique Topolansky, director del CIE (Centro de Innovación y Emprendimientos de la Universidad ORT), el poder de la Inteligencia Artificial con el que contamos hoy *"rompe con los preconceptos de la escuela de Porter de que algo no puede ser a la vez* ***masivo y segmentado****"* [24].

También utilizamos también un **Mapa de Empatía** (ver anexo 15.1.3), para empatizar con el comprador online. A partir de esta técnica, pudimos aprender, entre otras cosas, que el comprador online no solamente se enfrenta a grandes catálogos de productos (lo cual genera indecisión), sino que también oye de promociones online de parte de conocidos y no sabe cómo accederlas (siente que se está perdiendo de cosas), así como también desconoce si su forma de buscar en e-commerce es la correcta.

Finalmente, para finalizar con las herramientas de validación, se construyó un

User Journey Map (ver anexo 15.1.4), basado en la información recolectada a través de las técnicas que se presentan en las siguientes secciones. Este nos ayudó, desde una visión centrada en el usuario, a identificar puntos de dolor durante su proceso de compra, empatizando con este.

3.2.2.2 Desarrollo de un MVP: feedback experto

Como técnica de validación, decidimos desarrollar un MVP, tomando como cliente a Wikimúsculos, empresa líder en suplementos deportivos, quien se ofreció a trabajar con nosotros para esta instancia del proyecto.

El alcance de este MVP se limitó a una página web a partir de la cual los usuarios podían hacer preguntas a *wisello*, con fines de **asesoramiento y educación nutricional personalizada**. Su simpleza se explica en tanto no existe pre-procesamiento ni búsqueda de información, técnicas o grandes esfuerzos en *prompting* del modelo. A continuación, capturas del mismo:

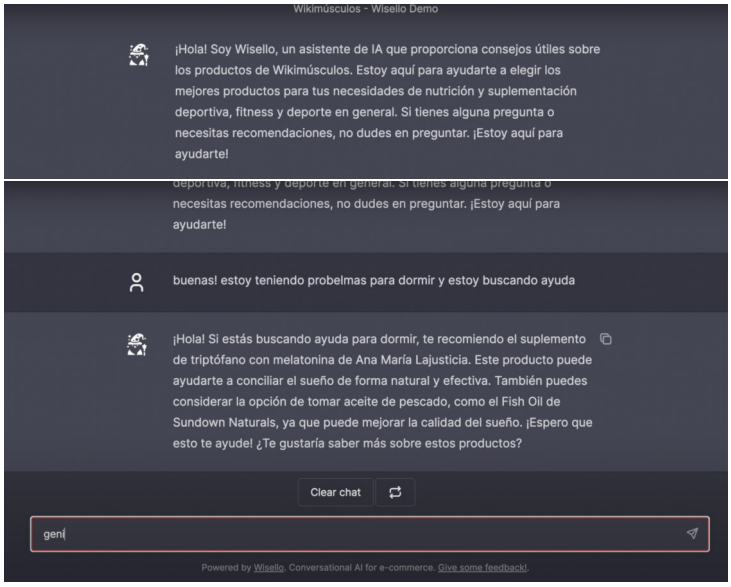


Figura 3.1: MVP funcional desarrollado en una primera etapa para Wikimúsculos

El objetivo del MVP era evaluar con expertos del rubro (empleados, nutricionistas empleados y directivos de la empresa), qué tan bueno era el asesoramiento brindado a través de la herramienta. El feedback fue recolectado a través de una serie de sesiones de validación con Maximiliano Cáceres, CEO de Wikimúsculos, y su equipo, llevadas a cabo durante los meses de Marzo, Abril y Mayo de 2023, en forma presencial.

En las primeras sesiones el énfasis del feedback estuvo en la necesidad de recolectar más información del usuario previo a la recomendación de un producto, para obtener recomendaciones más acertadas, en base a la realidad de cada usuario. Como mínimo, el equipo de Wikimúsculos exigió conocer su peso, altura, sexo y objetivos nutricionales, para luego recomendar, pues una recomendación errónea puede impactar la salud del cliente. También, se discutió la importancia de aclarar al usuario que debe tener expectativas coherentes acerca de los suplementos deportivos recomendados, así como presentarle las indicaciones de uso del producto que está investigando.

En las siguientes sesiones, como destacado, surgió el problema de que *wisello* ocasionalmente recomendaba productos que la tienda no comercializaba (*alucinaba* productos), lo cual no era aceptable desde un punto de vista comercial.

Ya hacia el final, pudimos concluir que la herramienta se desempeñaba muy bien y el potencial para la automatización del asesoramiento era grande. Como feedback, Maximiliano recomendó integrar inteligencia del negocio al producto. *"En Wikimúsculos todo el mundo pregunta precio (...) El único tema así como clave que no toca es el tema del precio"*, nos comentó Maximiliano, en una de las sesiones de validación. Además, continúa: *"Estaría bueno que eventualmente compare precios con otros productos, te hable de costo por gramo de proteína, te hable de promociones"*.

Los aprendizajes recabados con esta técnica son claros:

- Existe un desafío a resolver entorno a las alucinaciones del modelo.
- Existe un deseo de personalización de la herramienta por parte de la tienda, en términos de funcionalidades. En el caso de Wikimúsculos:
  - Colecta de información del usuario previo a la recomendación.
  - Explicación de expectativas del producto.
  - Indicaciones de uso del producto.
  - Comparación de precios.
  - Costo por gramo de proteína.
  - Promociones.
- Existe una buena recepción de la herramienta y una gran capacidad para automatizar tareas complejas, como el asesoramiento nutricional, con efectividad.

3.2.2.3 Desarrollo de un MVP: engagement de usuarios

Para medir la reacción y el *engagement* de los usuarios con este tipo de tecnologías y la idea en general, necesitábamos acceder a una base de usuarios con las cuales validar nuestro MVP. Fue allí que nos comunicamos con Mg. Arq. Fernando García Amen, coordinador académico general de la Facultad de Arquitectura UdelaR e integrante del cuerpo docente de **Plexo**: iniciativa que fomenta el uso del viaje de arquitectura como herramienta integral en la formación académica y práctica de los arquitectos.

Plexo ofrece una colección de guías informativas detalladas para cada país en el itinerario, donde los viajeros pueden encontrar información sobre monumentos, edificaciones y las principales atracciones turísticas. La idea de sistematizar estas guías mediante un chat automatizado fue de gran agrado para Fernando, por lo que la misma fue ejecutada y distribuida a los participantes del viaje.

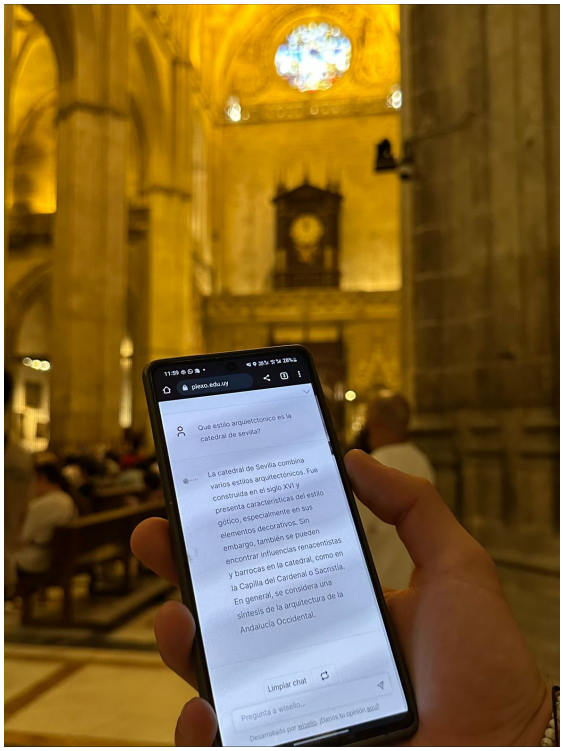


Figura 3.2: MVP desarrollado para Plexo

Las métricas de uso fueron realmente sorprendentes. En la primer semana de su lanzamiento en Mayo del 2023, obtuvimos **492** usuarios, que utilizaron el MVP **1036** veces esa semana. Esto se logró sin esfuerzos en marketing; el MVP fue colocado en la página web de Plexo, y a través del boca a boca, los arquitectos viajeros lo fueron adoptando.

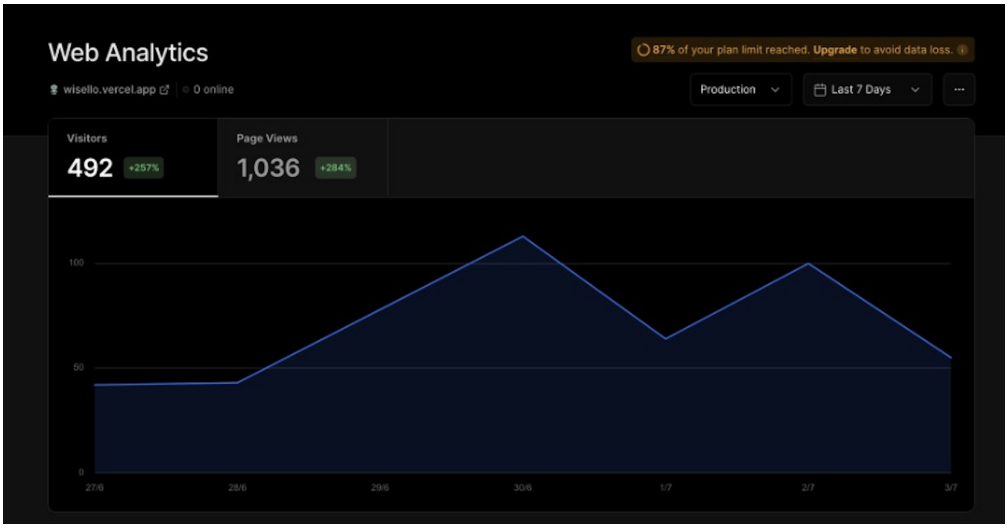


Figura 3.3: Métricas del MVP en su *primer semana* de lanzamiento

Las consultas de los viajeros variaban desde preguntas generales de turismo, arquitectura, historia o arte. A través del chat, los viajeros eran capaces de acceder a un servicio de personalización que las guías, por su naturaleza estática, no les ofrecían. Por ejemplo, habían muchas preguntas de personalización del estilo de `"Hola, recomendame cómo recorrer Lyon en 3 horas"`, o `"Dónde es un buen lugar para hacer una date con mi pareja en Basilea de noche"`. También, consultas de filtrado como `"Cuéntame sobre la materialización de la estructura de la unidad de Marsella de Le Corbusier"`, información que se encuentra en las guías pero que probablemente tome más tiempo de buscar.

En los siguientes meses del proyecto, el MVP continuó su actividad, registrando **677** y **634** usuarios mensuales. Por tanto, los aprendizajes que tomamos del desarrollo de esta actividad de validación fueron los siguientes. Por un lado, existe un deseo tanto de asesoramiento como de respuesta individual **personalizada**, así como de incrementar la productividad tras buscar información en grandes fuentes de información.

3.2.2.4 Entrevistas con expertos en e-commerce

Con el fin de validar, de primera mano, el problema y la oportunidad de potenciar el asesoramiento en e-commerce, decidimos llevar a cabo una serie de entrevistas con expertos en el rubro.

A través de **Leonardo Álvarez, CEO y cofundador de Fenicio eCommerce**, conocimos cuál es el comportamiento del comprador online uruguayo (ver 3.1.5); un comprador analítico e indeciso que concluimos podría valerse de un asesoramiento más profundo. A su vez, Leonardo nos compartió qué es lo que mejor funciona en e-commerce: tener información disponible para el usuario, lo que incluye contestar sus dudas e incluir puntos de contacto.

Otra entrevista muy enriquecedora fue con **Lucía Puentes, encargada de E-commerce en Divino**, quien nos recalcó la importancia del asesoramiento como forma de dar un seguimiento, convencer al usuario de su compra, y sobrellevar las barreras a la compra que existen hoy en e-commerce.

Por otro lado, al reunirnos con **Juan Teba, de e-commerce en Farmashop**, aprendimos que existe una voluntad por parte de los usuarios de gastar más en medios digitales. Además, Juan nos comunicó que hace falta asemejar y llevar la figura del vendedor, que asesora y facilita, al canal online, lo cual está fuertemente alineado a nuestra hipótesis. Según él, esto podría *"potenciar la venta en el canal online"*.

Otro experto con el que pudimos conversar fue **Ignacio Castañares, Head of Product de Fenicio eCommerce**, a quien tuvimos la oportunidad de presentarle el MVP realizado para Wikimúsculos. Dado su rol como encargado de producto en Fenicio, nos interesaba su visión y opinión acerca de cómo creía que podía encajar nuestro producto en el mercado. Su feedback fue realmente fascinante: no solo fue muy positivo sino que además llegamos a un **acuerdo para integrarnos con la plataforma Fenicio** (ver 4.2.4).

Finalmente, nos reunimos con otro experto, **Andrés Magrini, Country Manager en Janis**. Al mostrarle nuestro MVP, Andrés reconoció de inmediato su potencial para simplificar la catalogación de palabras clave en los productos de un e-commerce. Con la capacidad de buscar por semántica, este problema se resuelve, ahorrando el arduo trabajo de catalogación que muchos buscadores tienen.

3.2.2.5 Segunda hipótesis

Luego de realizar la validación pertinente, reformulamos nuestra hipótesis anterior:

Existe una falta de información y asesoramiento en tiendas e-commerce en Uruguay.

A la siguiente:

Falta de un mecanismo desestructurado y doble compuesto por la atención al cliente y búsqueda, para e-commerce en Latam, que permita potenciar la falta de asesoramiento en la tienda. Existe una necesidad simultánea: por parte de las tiendas de e-commerce de automatizar estos servicios y la personalización brindada, así como del usuario de tener experiencias personalizadas y humanas.

4 Solución

4.1 Descripción de la solución

A partir de las instancias iniciales de análisis y posterior validación del problema, identificamos las necesidades de los principales actores involucrados en el proceso de compra en línea (las tiendas e-commerce y los compradores).

4.1.1 Solución inicial y su evolución

La solución inicial propuesta para *wisello* fue la de un sistema de asesoramiento compuesto por un agente conversacional (chatbot), entrenado con el catálogo de productos de la tienda y conocimiento específico del negocio y su sector. Este agente es capaz de ofrecer una experiencia de compra asistida al usuario al recomendarle productos apropiados a sus necesidades e inquietudes, evacuar sus dudas, y proveer personalización durante la compra, utilizando tecnologías de LLMs (como GPT-3.5). A través del diálogo, *wisello* reflejaría el rol de un vendedor en tienda física pero en el canal digital.

Sin embargo, y manteniendo nuestra filosofía ágil, durante el transcurso del proyecto, el contacto con clientes y la puesta en producción nos obligaron a expandir y modificar la solución.

4.1.1.1 Acceso al catálogo de productos y datos de interés

Cuando comenzamos a validar con más tiendas, nos dimos cuenta que cada e-commerce, al ser desarrollado por distintas empresas, tendría una forma distinta de acceder a su catálogo de productos. Por ende, decidimos orientar la solución hacia plataformas, como Fenicio (se ahonda en el acuerdo con Fenicio en 4.2.4), donde la estructura de todos los e-commerces son iguales, y así poder lograr mayor escalabilidad del producto.

De lo contrario, ofreceríamos como opción al e-commerce exponer una API para nosotros poder consumir el catálogo de productos.

4.1.1.2 Requerimientos variados de funcionalidades

En el mes de junio, OpenAI lanzó una nueva funcionalidad: **OpenAI Functions**. Se trata de funciones (código, esencialmente), pero que el modelo puede, en forma inteligente, elegir a cuál llamar. Este **cambio tecnológico** implicó poder darle más extensibilidad al producto, y poder desarrollar funciones *custom* para cada cliente. Desde el punto de vista tecnológico, esto era una necesidad no resuelta que teníamos, pero que se había observado desde el inicio de la validación con el caso de Wikimúsculos (ver sección 3.2.2.2), donde se había expresado el deseo por contar con funcionalidades *custom* como: costo por gramo de proteína, indicaciones de uso del producto, explicación de expectativas del producto, entre otras.

4.1.1.3 Principales necesidades del cliente que surgieron en la marcha

Como pedidos del cliente **Casashub**, en las reviews que se tuvieron del producto, surgieron:

- La necesidad de que el cliente pueda, por algún medio, ver el funcionamiento general de la herramienta (cuánto se usa, qué están preguntando los usuarios), y configurar aspectos de la misma.
- Posibilidad de contactar a un agente humano si se desea.

4.1.2 Solución final

A continuación, se detallan cada uno de los componentes de la solución final de *wisello*, y profundizamos en sus características.

4.1.2.1 Sistema de asesoramiento

El sistema de asesoramiento, como componente central de la solución, se mantiene en su esencia como el descrito inicialmente. El cambio radica en que adquiere un poder de extensibilidad mucho mayor en cuanto a funcionalidad, por los avances tecnológicos. Se ahondará en esto más adelante al describir su arquitectura.

El agente, especializado en la venta de productos, se integra en una sola línea de código al e-commerce del cliente, por lo que no tiene costos extras de desarrollo web.

Al tratarse de una solución conversacional con tecnologías de LLMs, se ofrece un asesoramiento desestructurado, en donde los usuarios pueden mantener diálogos sin fronteras de pregunta/respuesta en los que se evalúan las necesidades del cliente, se buscan productos adecuados y se dan explicaciones que justifican las recomendaciones.

4.1.2.2 Plataforma de visualización

Como componente complementario al sistema de recomendación, ideamos una plataforma para las tiendas que adopten dicho software. Esta plataforma es un espacio donde dichas tiendas pueden acceder a diversas métricas relacionadas con el rendimiento de su asistente de ventas personalizado. Además, les brinda la posibilidad de personalizar varios aspectos de la interfaz de dicho asistente.

Es una herramienta de monitoreo y configuración, que permite realizar un seguimiento y tener visibilidad sobre la experiencia que tienen sus usuarios con el sistema de asesoramiento.

4.1.3 Diferenciación con el mercado

Tras conocer la solución, podemos confirmar que *wisello*, como forma de asesoramiento, integra todas las características necesarias y expuestas en 3.1.6:

- Es personalizado, en tanto considera todo lo que el usuario expresa, así como el historial de la conversación. La respuesta producida por *wisello*, es única para un mismo *input*.
- Es de calidad, pues se enriquece del propio catálogo de la tienda, instrucciones de la tienda, conocimiento del rubro y del negocio, así como de toda la información con la cual están entrenados los LLMs.
- Está disponible 24/7.
- Es instantáneo.

4.2 Validación de la propuesta de solución

A lo largo de esta sección, explicaremos los mecanismos e instancias que el equipo utilizó para validar la propuesta de solución presentada previamente.

4.2.1 Obtención del fondo VIN (Validación de Idea de Negocio) de ANII/ANDE

Decidimos presentar nuestra idea de negocio ante el fondo concursable VIN de ANII/ANDE, que tiene como objetivo apoyar emprendimientos con potencial de innovación en sus etapas más tempranas, con un monto correspondiente a US$ 5000.

La propuesta presentada logró captar el interés de ANII/ANDE, por lo que obtuvimos el fondo, lo cual supuso un enorme hito para el proyecto por la validación que supone, y un incentivo para el equipo detrás. A continuación, el comentario de ANII/ANDE:

Se plantea una solución con potencial vinculada a una tecnología cada vez más recurrente y más usada, que es la IA como asistente de compra. Se sugiere que luego de la validación técnica, se ponga foco en la validación comercial intentando encontrar el ROI (retorno de la inversión) de implementar esta solución en un eCommerce, frente a no tenerlo.

4.2.2 Entrevista con los fundadores de BrainLogic AI

Realizamos una reunión con **Martín Alcalá: Cofundador - BrainLogic AI, Tryolabs** y **Juan Pablo Pereira: Cofundador - BrainLogic AI, Tiendamia**, quienes estaban por lanzar **Zapia**, un asistente personal con Inteligencia Artificial Generativa para Latinoamérica, utilizando Whatsapp.

El objetivo de esta reunión era recibir consejo acerca de cómo navegar la ola de *GenAI* (Inteligencia Artificial Generativa). Martín nos recomendó especializarnos en una tecnología específica y ser muy buenos en ella. De ser open-source, también mencionó que puede ser una buena idea ser *contributors* del proyecto open-source. Concluimos que lo podíamos hacer sobre el framework **Langchain**. Langchain es

un framework diseñado para crear aplicaciones impulsadas por modelos grandes de lenguajes (LLMs), sobre el cual hemos adquirido mucha experiencia trabajando a lo largo del proyecto. Además, también nos aconsejó participar de eventos de tecnología y darnos a conocer como referentes de *GenAI* en el país.

4.2.3 Participación en eventos tech

Tal como nos aconsejó Martin Alcalá, participamos en eventos de tecnología, para dar a conocer nuestra marca, tanto a nivel comercial como personal.

4.2.3.1 Andrés como panelista en Fenicio Talks

Fuimos invitados al evento **Fenicio Talks**, organizado por Fenicio eCommerce, para participar como speakers en su panel de **Inteligencia Artificial en e-commerce**.

Junto a Enrique Topolansky del CIE y Alan Rytt de Fitit.ai, Andrés participó del panel, compartiendo la visión de *wisello* sobre el futuro de la IA en e-commerce: su poder para hiperpersonalizar experiencias de compra, potenciar la retención, y los desafíos que implica poner asistentes virtuales (especialmente aquellos impulsados por modelos grandes de lenguaje como GPT) en manos de nuestros clientes.

Figura 4.1: Andrés como panelista en Fenicio Talks

Se puede encontrar la entrevista completa aquí: `https://www.youtube.com/watch?v=F-cPJl5Ky5g&t=515s`

4.2.3.2 Stand en el evento de ITBuilders

Participamos con un stand en el evento **ITBuilders Live** organizado por ITBuilders en el Antel Arena. Este evento anual conecta todo el ecosistema tecnológico uruguayo, dándonos así la posibilidad de dar a conocer nuestro producto y hacer *networking*.

Figura 4.2: El equipo en ITBuilders Live

La participación en ambos eventos fue de gran valor para nuestro recorrido de emprender. Forjamos relaciones que luego nos ayudarían en el camino conectándonos con otras personas, o con sus propios recursos. Incluso, más que nada en el caso de Fenicio Talks, nos generó varios prospectos que demostraron su interés por integran la herramienta a su e-commerce, razón por la cual clasificamos estas actividades como de validación de la propuesta.

4.2.4 Acuerdo con Fenicio e-commerce

Tal como fue mencionado en la sección 3.2.2.4, logramos llegar a un acuerdo con Fenicio e-commerce, para poder integrar *wisello* a la plataforma, lo cual implicó tener acceso al backend de productos de Fenicio.

Para nosotros, esto supuso un gran paso, en tanto supone que una empresa establecida como Fenicio confíe en nuestro equipo y en nuestro producto, para que lo utilicen sus clientes. Ignacio Castañares, Head of Product, nos expresó que consideraba que *wisello* podía aportar valor a la experiencia final del usuario en un e-commerce de Fenicio, y es por ello que les parecía interesante contar con la herramienta.

4.2.5 Primer cliente Casashub

Luego de unos meses, lanzamos a producción con nuestro primer cliente pago: **Casashub**, un portal de real estate, cuyo diferencial es su especial énfasis en tecnología, ofreciendo recorridos virtuales por las propiedades y ahora, búsquedas personalizadas.

Si bien el cliente no cae bajo el rubro de e-commerce estrictamente, el paradigma de búsqueda en el portal funciona de igual forma, o en su defecto, es aún un mayor desafío, por el valor elevado que tienen las propiedades y la cantidad de factores a considerar al buscar una de ellas. Por ende, consideramos que, dado el desafío y las coincidencias, podíamos incursionar en el rubro de real estate.

Mariana y Andrés, fundadores del portal y en constante búsqueda de integrar innovaciones tecnólogicas al mismo, nos comentaron que a su buscador viejo de propiedades le faltaba especificidad y personalización; era imposible que el usuario busque por cualquier atributo que se le ocurra (como por ejemplo: cercanía a determinado colegio, vista a determinado lago, losa radiante, etc). Esto hacía que muchas de las búsquedas, aquellas que eran en su intento **personalizadas**, terminen siendo fallidas, dado que el usuario no podía buscar/filtrar por aquello que consideraba relevante.

Allí es donde *wisello* encajó como buscador **desestructurado y personalizado** (ver segunda hipótesis 3.2.2.5) de propiedades; momento a partir del cual pudimos confirmar que el dolor que estábamos atacando realmente existía.

4.2.6 Microsoft for Startups Founders Hub

Fuimos aceptados por el programa de Microsoft ***Microsoft for Startups Founders Hub***, a través del cual pudimos acceder a una variedad de créditos en distintas plataformas, entre otras: Azure (US$ 25.000), OpenAI (US$ 2.500) y Linkedin Ads (US$ 1.000), a través de los cuales sustentamos los gastos de infraestructura.

5 Listado de requerimientos

En esta sección se describen los principales requerimientos funcionales y no funcionales especificados para la construcción de la solución. Estos requerimientos fueron establecidos luego de realizar las correspondientes validaciones, expuestas en los capítulos anteriores.

5.1 Requerimientos funcionales

5.1.1 Sistema de asesoramiento

Obtener asesoramiento personalizado a través de un agente conversacional

Obtener recomendaciones personalizadas de productos

Acceder a los productos de la tienda de forma actualizada

El sistema debe proporcionar una interfaz de usuario intuitiva y accesible, permitiendo al usuario interactuar con un agente conversacional, para recibir asesoramiento personalizado a través de texto. Esta interacción debe ser lo más natural y fluida posible, imitando una conversación humana.

Este agente conversacional debe ser capaz de recomendar productos correspondientes al catálogo de productos de la tienda, en forma personalizada.

Al clickear sobre un producto recomendado por el asistente, el usuario debe ser redireccionado al e-commerce de la tienda, específicamente al producto seleccionado.

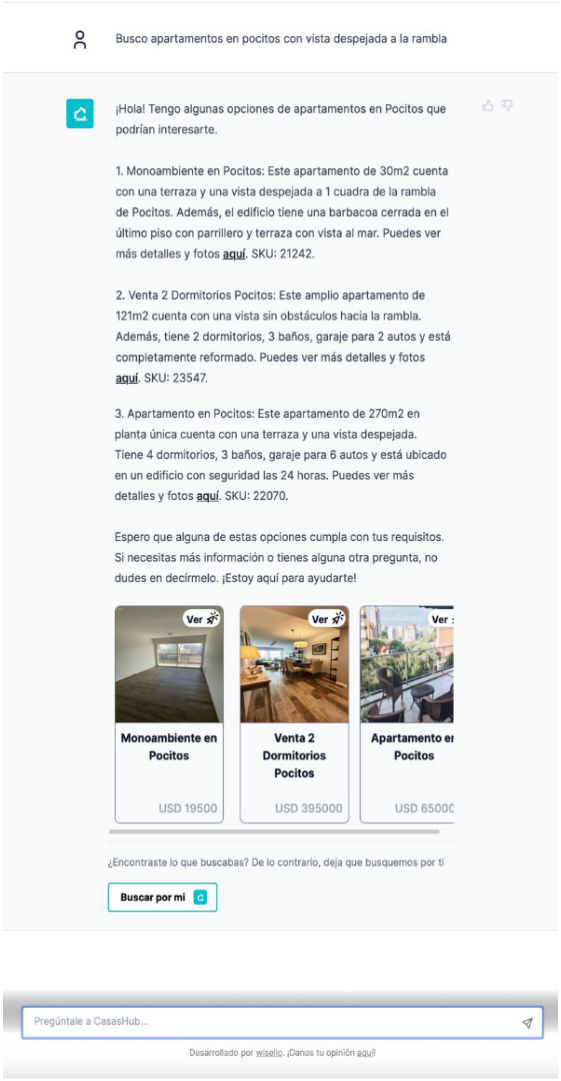


Figura 5.1: Recomendación de productos a través del agente conversacional

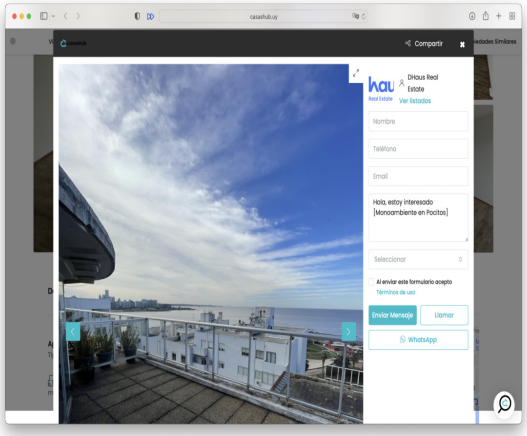


Figura 5.2: Click sobre el producto recomendado *Monoambiente en Pocitos*

Sugerencia de preguntas

El agente conversacional debe sugerir preguntas iniciales para que el usuario comience su interacción con el mismo. En su versión web se deben presentar tres preguntas, mientras que en *mobile* deben sugerirse dos de ellas a elección.

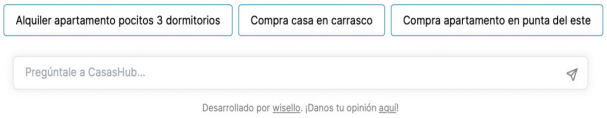


Figura 5.3: Sugerencia de preguntas iniciales, versión web

Figura 5.4: Sugerencia de preguntas iniciales, versión *mobile*

Feedback personalizado y genérico de búsqueda

Al iniciar una búsqueda de un producto específico con características particulares, el sistema debe proporcionar un feedback al usuario, primero general, y luego cuando se hace disponible, uno personalizado y detallado sobre el proceso de búsqueda en curso.

Este tiene como objetivo mantener al cliente informado, brindándole actualizaciones claras sobre el estado de su solicitud y asegurando una experiencia de usuario más interactiva y satisfactoria.

En el caso a continuación *"Buscando para ti... "* es el componente **general**, y todo lo que le sigue, que se hace disponible momentos luego, es el componente **personalizado**:

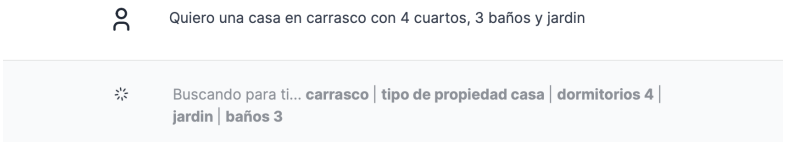


Figura 5.5: Feedback de búsqueda

Búsqueda inversa de productos / contacto con un agente humano

En el caso de que el usuario no encuentre el producto que esté buscando a través de *wisello*, este debe tener la posibilidad de poder completar un formulario en donde detalle las características de su búsqueda. Luego, este formulario será enviado por e-mail a los responsables de la tienda para poder atender al cliente, y responder a su búsqueda de forma adecuada y **personalizada**.

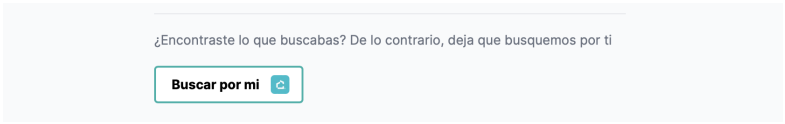


Figura 5.6: Búsqueda inversa de productos

Calificar la respuesta obtenida

El usuario debe poder calificar la respuesta obtenida a través del sistema de asesoramiento mediante un like/dislike.

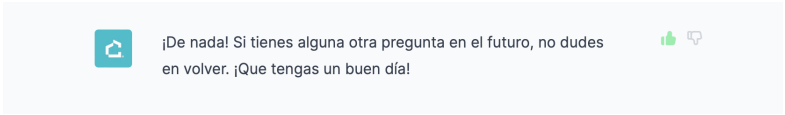


Figura 5.7: Calificación de respuesta

Vaciar chat

El usuario debe poder tener la posibilidad de vaciar la conversación que ha tenido previamente con el agente conversacional, y que por ende se encuentra almacenada en su chat.

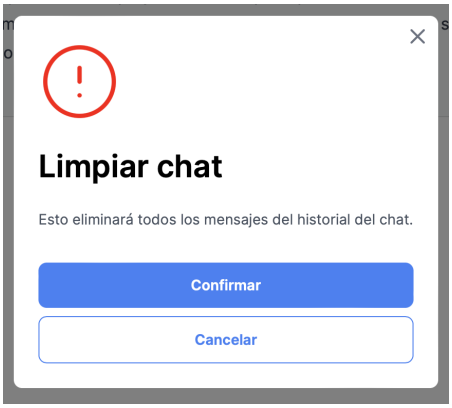


Figura 5.8: Vaciar chat

Integración en formato burbuja o embebido

Integración en una sola línea de código

Wisello se debe poder integrar al e-commerce del cliente en formato burbuja o embebido, y en una sola línea de código para ambas alternativas de integración.

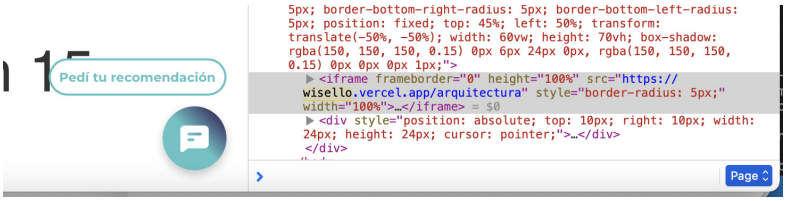


Figura 5.9: Integración burbuja de *wisello*

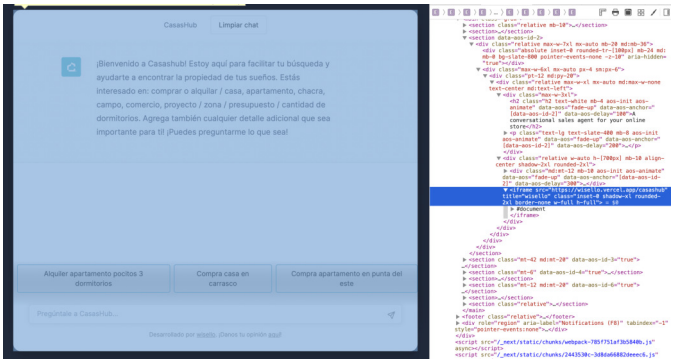


Figura 5.10: Integración embebida de *wisello*

Conversación de máximo 10 mensajes

El máximo número de mensajes para una conversación debe ser 10.

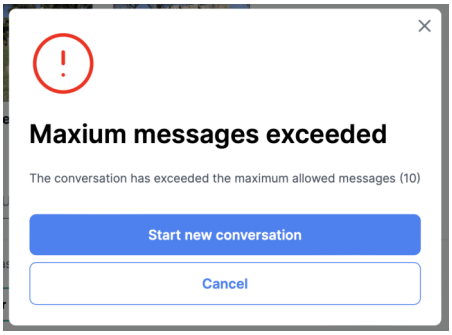


Figura 5.11: Máximo número de mensajes

5.1.2 Plataforma de visualización

Inicio de sesión

Cada tienda debe poder acceder a la plataforma de visualización a través de sus credenciales: nombre de usuario y contraseña.

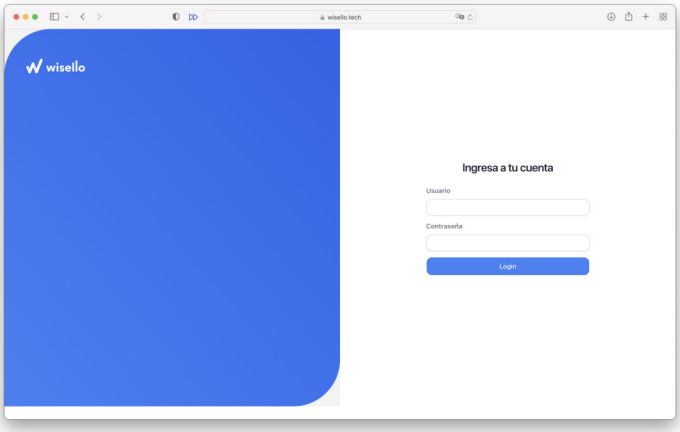


Figura 5.12: Inicio de sesión

Customización visual del chat

Cada tienda debe poder customizar su mensaje inicial de *wisello* en cualquier momento, y ver el cambio impactado de forma instantánea.

Figura 5.13: Panel de configuración del mensaje inicial

Descarga de reporte de conversaciones por fecha

Cada tienda debe poder descargar un reporte de las conversaciones que hayan tenido los usuarios con *wisello*, en una fecha determinada. La fecha debe ser seleccionada a partir de un calendario, y el *output* de la acción, es decir, el reporte, debe ser un archivo PDF, que se debe descargar automáticamente en la computadora del usuario solicitante.

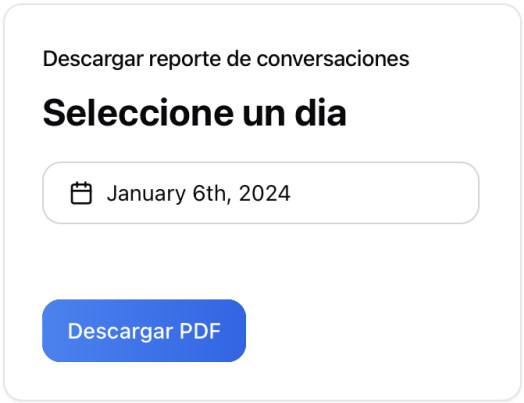


Figura 5.14: Descarga de reporte de conversaciones por fecha

Visualización de métricas

Cada tienda debe ser capaz de visualizar las siguientes métricas de uso:

- Cantidad de mensajes mensuales.
- Cantidad de mensajes históricos.
- Cuáles fueron los leads generados a través de la búsqueda inversa.
- Cuántos fueron los leads generados a través de la búsqueda inversa.

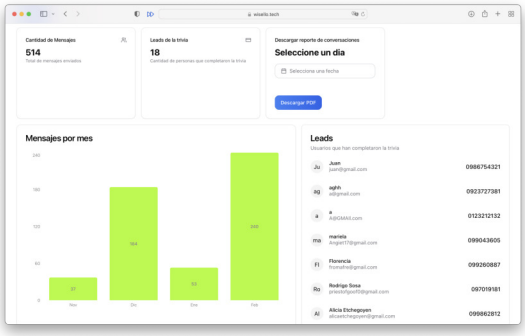


Figura 5.15: Dashboard de métricas de uso de *wisello*

Log out

Cada tienda debe poder finalizar su sesión, volviendo a la página de aterrizaje de *wisello*.

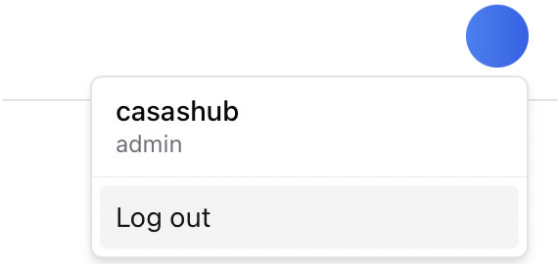


Figura 5.16: Log out

5.2 Requerimientos no funcionales

La determinación de los requerimientos no funcionales para el proyecto se fundamenta en base a dos criterios fundamentales, reflejando tanto las necesidades específicas de nuestros clientes como los atributos de calidad que, como equipo, identificamos como cruciales para el éxito del producto. Esta selección no solo apunta a satisfacer las expectativas y requisitos funcionales del cliente, sino también a garantizar la robustez, eficiencia y sostenibilidad de nuestro software.

5.2.1 Performance

RNF-PR-01: El tiempo de respuesta del chat en empezar a responder (*time to first token*) debe ser en promedio menor a 5 segundos, con una carga concurrente de 100 usuarios pidiendo recomendaciones al chat.

5.2.2 Desplegabilidad

RNF-DP-01: Se debe poder desplegar una nueva versión de cada subsistema sin que afecte a los otros y sin ocasionar downtime en ningún otro.

RNF-DP-02: Debe existir un proceso de despliegue automatizado, que permita desplegar nuevas versiones de la aplicación con una sola acción.

RNF-DP-03: El `backend` debe poder levantarse de forma local en un solo comando, utilizando docker-compose.

RNF-DP-04: Deben implementarse ambientes *staging* de despliegue iguales al de producción para cada subsistema, donde deben ser probados los cambios previo a su despliegue en producción. De esta forma, se asegura que la aplicación funcione correctamente en un entorno idéntico al de producción.

5.2.3 Usabilidad

RNF-US-01: Los usuarios deben ser capaces de aprender a usar el chat con todas sus funcionalidades (Obtener respuesta, Like/Dislike, Buscar por mi, Preguntas sugeridas, etc) en un tiempo máximo de 3 minutos. De manera similar, el tiempo máximo permitido para que los usuarios aprendan a interactuar con todas las funcionalidades de la plataforma de métricas y configuración es de 10 minutos.

RNF-US-02: El sistema hace visible al usuario su estado en todo momento, de forma que el usuario es partícipe del estado del sistema. Es decir, si el sistema está realizando una acción, está cargando contenido, o si la acción del usuario ha sido impactada correctamente, entre otros.

RNF-US-03: El usuario debe ser presentado con mensajes de error apropiados, entendibles e informativos para cada caso (y en todos ellos) de escenarios erróneos.

RNF-US-04: Tanto el chat como la plataforma deben ser responsive, pudiendo adaptarse a dispositivos con distintos tamaños de pantalla.

5.2.4 Disponibilidad

RNF-DP-01: El sistema debe estar disponible para su uso al menos el 99 % del tiempo, soportando picos de carga de fechas importantes (por ejemplo: ciberlunes, black friday).

RNF-DP-02: El sistema debe estar configurado con redundancia de servidores, para así poder garantizar la disponibilidad del sistema ante la posible falla de un servidor.

5.2.5 Seguridad

RNF-SG-01: El sistema debe contar con un mecanismo de autenticación para cada tienda y sus usuarios, de forma que no cualquiera pueda obtener datos o realizar acciones sobre el sistema.

RNF-SG-02: El sistema debe contar con un mecanismo de autorización o control de acceso que no permita a usuarios realizar acciones que estén por encima de sus niveles de permisos, o no correspondan a su tienda en particular. Bajo ningún concepto los usuarios de una tienda podrán acceder a datos de otra tienda.

5.2.6 Extensibilidad

RNF-EX-01: Sabiendo que habrá nuevas tiendas y surgirán nuevas funcionalidades para tiendas existentes, el sistema debe permitir agregar nuevas tiendas y funcionalidades fácilmente, sin modificar código existente, solamente extendiéndolo.

5.2.7 Observabilidad e Identificación de fallas

RNF-OB-01: Se debe poder monitorear las siguientes métricas del `backend` en un solo lugar:

- Peticiones por minuto
- Tiempos de respuesta de todos los endpoints
- Cantidad de fallos
- Consumo de CPU y memoria

RNF-OB-02: Se deben centralizar y retener los logs en un único lugar, separados por ambientes, para monitorear el funcionamiento del sistema e identificar fallas. Se deben poder filtrar según sus atributos.

RNF-OB-03: Se debe poder configurar alertas para notificar a todos los integrantes del equipo por mail ante incidentes que ocasionen una falla en el sistema.

5.2.8 Escalabilidad

RNF-ES-01: El sistema debe poder escalar horizontalmente de forma automática, agregando nuevas instancias ante picos de tráfico.

5.2.9 Portabilidad

RNF-PB-01: La aplicación debe poder ser accedida desde los navegadores Safari, Chrome y Microsoft Edge.

RNF-PB-02: El backend debe poder ejecutarse en cualquier sistema operativo o dispositivo.

RNF-PB-03: El sistema debe poder ser desplegado, según se requiera, en ambientes de producción, desarrollo y pruebas.

6 Diseño de aplicaciones basadas en Large Language Models

En este capítulo presentamos una serie de temáticas, patrones y estado del arte para el diseño y desarrollo de aplicaciones basadas en Large Language Models; y su implementación específica en el caso de *wisello*. Las buenas prácticas de desarrollo de software fueron y siguen siendo desafiadas por el nuevo paradigma que supone el no determinismo de los LLMs, y por eso creemos apropiado hacer un breve repaso de lo que algunos autores consideran como las prácticas más importantes para el diseño de aplicaciones con LLMs, junto con la experiencia y aprendizajes al implementarlos en *wisello*.

6.1 Prompt Engineering

Prompt Engineering es un campo emergente en el desarrollo de aplicaciones basadas en LLMs. Tiene por objetivo la creación de *prompts* (o *system prompts*) que optimicen las salidas de LLMs para distintas tareas. La eficacia y calidad de las respuestas depende, en gran medida, de cómo se formula la *prompt* y de ahí la gran importancia de esta disciplina.

A continuación presentamos algunas estrategias de *Prompt Engineering* utilizadas en *wisello*, junto con los principios y tácticas que fueron aplicados en su ejecución.

6.1.1 *Zero-shot learning*

Se trata de la estrategia de *prompting* más simple; utiliza las capacidades innatas del modelo para responder a tareas nuevas sin haber sido expuesto previamente a ejemplos específicos en su entrenamiento. Para lograr un buen resultado, es necesario que la *prompt* sea lo suficientemente clara y directa como para que el modelo entienda la tarea desde una sola interacción.

```
1          """
2          Tu nombre es wisello, trabajas para un portal inmobiliario llamado
       casashub y tu única tarea es recomendar propiedades. Un cliente esta
       buscando %s.
3
4          Tienes las siguientes propiedades disponibles: {agent_scratchpad}. Tu
       trabajo es siempre recomendar una o más de estas propiedas al cliente,
       limitandote únicamente a la información disponible de ellas. No recomiendes
        propiedades que no pertenecen a la empresa. No seas creativo. Cita el 'sku
       ' de las propiedades recomendadas siguiendo este exacto formato: "SKU: X",
       donde X es el 'sku' de la propiedad mencionada.
5
6          Recuerda, siempre recomienda! Intenta ser lo más breve posible, sin
       perder información importante.
7          %s
8          Conversación con el usuario:
9          """
10
```

La primer táctica utilizada es la de pedirle al modelo que adopte una *persona*: *"Tu nombre es wisello, trabajas para ..."*. Esta táctica se complementa con el principio de *instruir claramente al modelo con el resultado deseado*, lo vemos en: *"tu única tarea es recomendar propiedades"* y luego al final *"Recuerda, siempre recomienda!"*.

El principio de *solicitudes claras y específicas* se ve al realizar solicitudes sobre el resultado esperado (la recomendación), en específico: que debe limitarse *"a la información disponible ..."*. El uso del **contexto** de propiedades como parte de la *prompt* forma parte de la aplicación del patrón Retrieval-Augmented Generation (RAG), en el que profundizaremos más adelante.

Otro principio aplicado es *cuidar la longitud*: como se observa, las frases son muy breves (*"No seas creativo."*). Esto ocurre debido a que las instrucciones más extensas resultan más complicadas de manejar e interpretar para el modelo. También, se *eligen las palabras con cuidado*, evitando la ambigüedad con lenguaje claro como *"exacto"*, *"siempre"*: palabras que no dan lugar a la confusión.

6.1.2 *String interpolation*

Como se puede ver en el ejemplo de la *prompt* anterior, se utiliza *string interpolation* (string con parámetros) representado por el carácter `%s`. De esta forma, la *prompt* puede recibir parámetros fuera del contexto y no ser estática, variando así para cada request. Con esta táctica, logramos enviar al LLM *prompts* más adecuadas (pues son más personalizadas) para cada petición, y así recibir mejores respuestas.

6.1.3 *Dynamic prompting*

Otra táctica utilizada en *wisello* dentro del área de *Prompt Engineering* es lo que nosotros denominamos como *Dynamic prompting*. Consiste en cambiar la *prompt* según el curso del sistema. Por ejemplo, consideremos el siguiente caso en que el usuario ingresó ***"quiero una casa en el barrio dfjkdfnkjfsd"***. Como la aplicación detecta que el barrio es inexistente, la *prompt* cambia dinámicamente a la siguiente:

```
1       Tu nombre es wisello, trabajas para un portal inmobiliario llamado
    casashub. Un cliente esta buscando propiedades pero desafortunadamente no
    has encontrado el barrio que te ha indicado: dfjkdfnkjfsd. El cliente puede
     referirse al nombre de un edificio, proyecto, u otra cosa quizás? O puede
    ser que se haya confundido de nombre, haya tenido un error de tipeo o
    cualquier otra razón. Pídele que te aclare esta situación para que puedas
    buscarle propiedades por él y asistirlo. Conversación con el usuario: Human
    : quiero una casa en el barrio dfjkdfnkjfsd"
2
```

6.1.4 Complemento con otros modelos

Para complementar las *prompts* del LLM, decidimos implementar y alojar un modelo propietario `BERT` de pregunta-respuesta en *wisello*.

La finalidad de este modelo es obtener respuestas acerca de atributos no estructurados de los productos o propiedades (que se pueden encontrar mencionados, por ejemplo, en la descripción del artículo), y poder ingerir esta información a la *prompt* del LLM.

Por ejemplo, si un usuario hace la siguiente consulta: ***"quiero un apto en Pocitos 3 dormitorios con vista al mar y a la playa"***, se identifican los

atributos no estructurados y se los agrupa bajo el parámetro `query`. En la consulta ejemplificada, estos serían: `"query": ["vista al mar", "vista a la playa"]` (el barrio, la cantidad de dormitorios, el tipo de propiedad son atributos que se encuentran estructurados en la base de datos). El resto típicamente se encuentran mencionados en alguna parte de la descripción del producto, o no se encuentran porque no existen directamente.

Por ende, lo que logramos con este modelo `BERT` fue, para cada producto (o propiedad) del contexto, responder la consulta (*¿la propiedad cuenta con vista al mar?*, *¿la propiedad cuenta con vista a la playa?*) para poder enviarle al LLM la información descifrada, y que no lo tenga que hacer por su cuenta, evitando errores.

Por ende, la *prompt* enviada al LLM tiene el agregado de la respuesta de `BERT`. Para el ejemplo sería (se la abrevia para presentar el segmento de interés):

```
1       Tienes las siguientes propiedades disponibles, pero no necesariamente
    cumplen con ['vista al mar', 'vista a la playa']. Para saber si las
    propiedades cumplen con ['vista al mar', 'vista a la playa'] o alternativas
     parecidas, observa las respuestas a las siguientes preguntas ['la
    propiedad cuenta con vista a la playa o parecido? Describe.', 'la propiedad
     cuenta con vista al mar o parecido? Describe.'] que están en cada
    propiedad. Si no cumplen con alguno de ['vista al mar', 'vista a la playa'
    ], entonces encontrarás que dirá 'No posee nada parecido'. Puedes encontrar
     que tienen algo parecido, pero no exactamente lo que se pide, aclaralo.
2
3       Las propiedades son: [{\"property_type\": \"apartamento\", \"name\": \"
    Apartamento en Pocitos\", \"la propiedad cuenta con vista al mar o parecido
    ? Describe.\": \"vistas al mar\", \"la propiedad cuenta con vista a la
    playa o parecido? Describe.\": \"vistas al mar\"}
4
```

El modelo `BERT` hosteado recibe *inputs* en inglés, razón por la cual tuvimos que utilizar un servicio de traducción externo para traducir tanto las preguntas como las descripciones de los productos (fuente de respuesta), para invocarlo.

6.1.5 *Few-shot learning*

Para hacer frente a la falta de exposición a ejemplos específicos durante el entrenamiento, esta estrategia consiste en proporcionar algunos ejemplos (shots) en la prompt para ilustrar la tarea deseada. Así, el modelo *"entiende"* mejor el contexto y la expectativa de salida.

En nuestro caso, utilizamos esta estrategia para los parámetros de las **OpenAI Functions**. Tal como lo indica su nombre, son funciones (código) que el LLM puede elegir llamar, en forma inteligente, con parámetros para la misma. Para ello, se tiene que definir primero qué significa cada parámetro de la función, y de ello dependerá qué tan bien los identificará el LLM. A continuación el ejemplo del parámetro `neighbourhoods`, de la función `SearchProperties`:

```
1         neighbourhoods: Optional[list[str]] = Field(
2         description="""
3             Lista de ubicaciones deseadas de la propiedad, como si se la
      estuviese buscando en un mapa. Es decir, zonas, regiones o barrios del país
      , mencionados por el usuario
4             Considera la historia de la conversación con el usuario.
5             Ejemplos:
6                     - [Carrasco, Punta Gorda]
7                     - [Carrasco]
8                     - [Punta del Este]
9                     - [Playa Mansa Punta Del Este, Playa Brava Punta Del Este]
10                    - [Solís de Mataojo]
11                    """
12        )
13
```

Como se puede ver, se indican ejemplos (*few-shot prompting*) para mejorar la asertividad en la definición de los parámetros. Esto además se conoce como la táctica de *proporcionar ejemplos*.

6.1.6 Alucinaciones

Una de los mayores desafíos que nos enfrentamos en *wisello*, y que fue detectado inicialmente durante el desarrollo del MVP para Wikimúsculos (3.2.2.2), refiere a las alucinaciones del modelo. Las alucinaciones refieren a la generación de contenido falso, inexacto o no basado en datos verificables, por parte del LLM.

Concretamente, lo veíamos cuando el modelo alucinaba productos de la tienda que no existían, o inventaba características falsas para estos productos. Para enfrentar este problema, recurrimos a todas las técnicas de *Prompt Engineering* descritas anteriormente: principios y tácticas de *Zero-shot learning*, *String interpolation*, *Dynamic prompting*, complemento con otros modelos y *Few-shot learning*.

Se destaca el uso de *Dynamic prompting* en conjunto con las **OpenAI Functions**. De acuerdo con el curso que toma la función ejecutada y el contexto de la misma, se cambia la prompt dinámicamente para manejar esa situación particular

y redirigir la respuesta hacia una dirección mas certera (por ejemplo, el input de un barrio inexistente como el que se expuso antes). Esto permite que la instrucción provista al modelo sea más concisa, sin dejar a libre interpretación del modelo de cómo y qué responder. Algo así como una relación 1-1 entre prompt y caso de uso. Por ende, esto tiene como beneficios la reducción de alucinaciones, aumentando así la precisión y relevancia de las respuestas.

En este sentido, las OpenAI Functions ayudaron mucho en tanto nos permitieron tener un mayor control sobre la respuesta final del modelo, siendo capaces de poder cambiar el rumbo de la misma según nos parecía conveniente.

Por otro lado, el complemento con otros modelos también fue creado con el fin de reducir de alucinaciones. En nuestro caso, utilizamos un modelo `BERT` para alimentar al modelo con información explícita acerca de los atributos de un producto siendo consultados, y así evitar que el modelo tenga que descifrar esta información por su cuenta y potencialmente, en este proceso, **alucine** al contestar.

Finalmente, la aplicación del patrón RAG (profundizado en la siguiente sección) que permite incluir información de fuentes externas en el contexto de la prompt, nos dio la capacidad de poder insertar el catálogo de productos de la tienda en este contexto, para luego recomendar o responder preguntas sobre estos productos. En sí, este fue el punto de partida para proveer respuestas basadas en información real y factual. Luego, por encima de este patrón, se aplicaron todas las técnicas expuestas de *Prompt Engineering* para trabajar en la alucinación sobre este contexto.

6.2 Patrones

Basamos esta sección en el artículo *Patterns for Building LLM-based Systems & Products* [25] de Yan Ziyou, que define los patrones presentados en la Figura 6.1, organizados según dos dimensiones: 1) el objetivo con el que se aplican y 2) la naturaleza de su implementación.

Véase para más detalles [25].

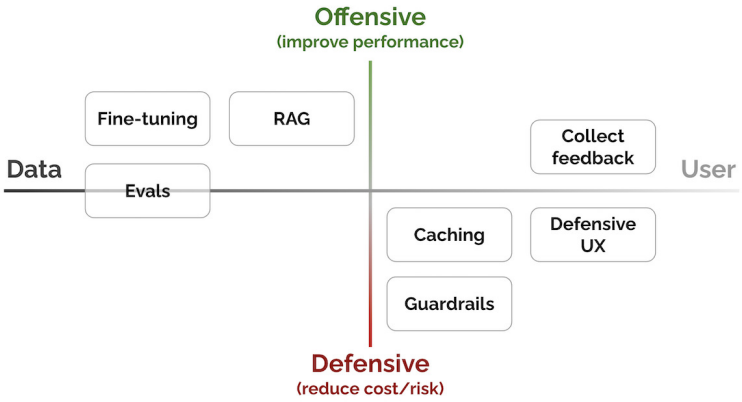


Figura 6.1: Patrones de Large Language Models.

6.2.1 Evaluación

Las aplicaciones basadas en LLMs se caracterizan por ser no deterministas y esto genera desafíos muy particulares a la hora de evaluar su rendimiento y detectar mejoras o regresiones. Asimismo, el estado del arte no se encuentra aún en una condición lo suficientemente madura como para prescribir pautas robustas sobre cómo evaluar las salidas de aplicaciones basadas en LLMs, y menos lo estaba cuando nosotros desarrollamos nuestro primer MVP.

En este contexto, aprovechamos esta oportunidad tanto para hacer una recopilación de las prácticas actuales que consideramos más fuertes en lo referente a evaluación de LLMs, como para reflexionar también sobre nuestra experiencia desarrollando sin la facilidad de tener guías reconocidas extensamente en la comunidad de desarrolladores.

Juicio humano

El juicio humano -definido como el enfoque de apoyarse en personas para hacer una evaluación de las salidas de una aplicación basada en LLMs-, se presenta comúnmente como el primer camino a tomar.

En nuestro caso, en etapas tempranas, los involucrados en la evaluación éramos nosotros mismos, los desarrolladores. Luego, esto se extendió a considerar la opinión de usuarios reales y clientes al utilizar el producto; información que era recabada durante las Reviews realizadas con clientes, que actuaban como “usuarios idóneos en el negocio”.

El principal impedimento para sistematizar este enfoque es el alto costo que tiene desempeñar la tarea a gran escala. Además, tiene varias limitaciones intrínsecas. Dada la inabarcabilidad de la gama de posibles entradas que puede tener una aplicación basada en LLMs, solo puede probarse un espectro acotado de las mismas; y estas, dado el no determinismo de los LLMs, pueden dar salidas diferentes cada vez. La métrica del juicio humano tiene, a su vez, otra característica problemática que es la presencia de *ruido*, es decir, de la discordancia entre los juicios de los anotadores humanos que surge de sus propios sesgos y hace que den evaluaciones diferentes frente a un mismo caso.

Dado que en el desarrollo de *wisello* no contamos con el presupuesto para hacerlo a gran escala, decidimos respaldarnos en las pruebas que nosotros mismos pudimos hacer, y en sacarle el mayor provecho posible a las pruebas de usuario. Además, buscamos alternativas que nos permitieran contar con métricas automatizadas a un bajo costo.

Métricas convencionales

Para evitar los costos altos del Juicio humano, se conocen distintas métricas de evaluación, que se basan en métricas de *precision*, *recall* y *cosine similarity* entre *embeddings*. Estas métricas, sin embargo, tienen tres desventajas principales:

1. Baja correlación con juicios humanos.
2. Baja adaptabilidad a diferentes tareas.
3. Poca reproducibilidad.

Nuevo paradigma: Evaluaciones automatizadas por LLMs

Frente a las desventajas anteriores, está emergiendo la tendencia a utilizar LLMs más fuertes para evaluar la generación de otros LLMs, que fue el camino que nosotros decidimos tomar. Esto significa, por ejemplo, utilizar GPT-4 para evaluar las salidas de GPT-3.5. Para ver en detalle la aplicación de esta técnica en *wisello*, referirse al capítulo de Aseguramiento de la Calidad, concretamente 12.2.

Estas nuevas metodologías han sido evaluadas tomando como referencia juicios humanos, y se demostró conclusivamente una correlación de Spearman más fuerte

en comparación con las métricas del título anterior (véase para más detalles [25]). Un punto interesante a notar es que estos métodos suelen tener un menor *ruido* que los juicios humanos, es decir que tienen menor *bias* entre sí, o que el *bias* está más sistematizado.

El paradigma de las evaluaciones automatizadas por LLMs también tiene sus limitaciones. Además, si bien estas pueden irse superando mediante estrategias *ad hoc*, regularmente se publican estudios que encuentran nuevos *biases*, y los mismos varían entre modelos. Algunas de las parcialidades detectadas son:

1. *Position bias*: la preferencia de las respuestas que se presentan en la primera posición -si se le pide al modelo indicar la mejor respuesta entre varias-.
2. *Verbosity bias*: la tendencia a preferir las respuestas más extensas.
3. *Self-enhancement bias*: la tendencia a preferir las respuestas creadas por el mismo modelo que hace la evaluación.

Evaluación en *wisello*

A lo largo de los primeros meses de la implementación de *wisello*, nos basamos exclusivamente en las pruebas manuales que hicimos nosotros mismos y los clientes con los que trabajábamos. En un título posterior profundizaremos sobre la recopilación de feedback de usuarios.

Aparte, desde las etapas iniciales nos vimos frente a una situación problemática: cada desarrollador que hacía cambios en las prompts del modelo buscaba optimizar las salidas para ciertas entradas que tenía en mente. Es decir que, imaginándose determinados escenarios posibles en los que podría encontrarse un usuario -entradas- y las salidas *esperables* para ellos, el desarrollador modificaba la prompt hasta conseguir satisfacer en buena medida ese escenario. El problema resultante era que cuando un desarrollador optimizaba la prompt para obtener buenos resultados en un conjunto de *escenarios A* y luego otro desarrollador optimizaba la prompt pensando en otro conjunto de *escenarios B*, frecuentemente nos encontrábamos frente a una regresión sobre los *escenarios A*.

Decidimos entonces utilizar una técnica de evaluación automatizada por LLMs. En ese momento no contábamos con frameworks sofisticados para esto, y nos basamos en un librería llamada auto-evaluator [26] adaptada para nuestro caso de uso. Véase Aseguramiento de la Calidad 12.2 para detalles de su implementación.

6.2.2 Retrieval-Augmented Generation (RAG)

El patrón de Retrieval-Augmented Generation consiste en la búsqueda de información relevante en fuentes externas, y la inclusión de la misma en el *context* de la *prompt*. RAG aborda las principales desventajas de los LLMs pre-entrenados: la falta de una memoria expansible y la incapacidad de mostrar las fuentes de información. En el caso de *wisello*, la información relevante a agregar en el *context* fueron los catálogos de productos de cada tienda.

6.2.2.1 Vector databases y Semantic Search

Una forma popular de buscar información relevante es conocida como Semantic Search, que se basa en la técnica de embeddings. Los embeddings son transformaciones que convierten textos en vectores o listas (en nuestro caso, de dimensión 1x1536). Tienen la característica de que la *distancia* entre vectores mide su relación semántica. Es decir que podemos buscar textos similares simplemente mediante la comparación de vectores (en nuestro caso, utilizamos *cosine similarity*).

Decidimos utilizar el modelo más popular de embeddings en la fecha de inicio del proyecto -que, sorprendentemente, siguió siéndolo a lo largo de toda la duración del mismo-: `text-embedding-ada-002` [27]. Como proveedor de vector database, utilizamos Pinecone [28].

Antes de aplicarse el proceso de embedding a textos extensos, suele utilizarse una estrategia de *chunking* que implica la descomposición del texto en segmentos pequeños, para poder dar una mejor representación semántica de los mismos [29]. En nuestro caso, la separación de la información en elementos independientes con una semántica coherente se dio naturalmente en la separación por producto. Es decir que por cada producto creamos un texto: un embedding, un vector. El texto de cada producto se generó con toda la información semántica relevante, desplazando algunas veces información a la *metadata* del vector, cuando esta no aportaba datos relevantes para la búsqueda semántica -por ejemplo, códigos identificadores-.

La *metadata* -definida como pares clave-valor adjuntos a un vector- también puede utilizarse para filtrar vectores, y limitar la búsqueda basado en estos campos. En el caso de *wisello* para CasasHub, utilizamos tags como ubicación (entre otros) para poder filtrar la búsqueda más eficientemente, y devolver respuestas aún más acertadas.

6.2.2.2 Integración de RAG

Para integrar RAG al modelo, implementamos dos arquitecturas diferentes. La implementación inicial fue mediante una *chain*[30] -es decir, una secuencia de LLM calls-. En la segunda y final, utilizamos Function calling [31]. La transición la hicimos apenas se anunció la opción de Function calling en los modelos de OpenAI, para aprovechar las ventajas que presentamos a continuación.

Para empezar, la arquitectura de chain consistía de dos LLM calls, la primera enviaba toda la conversación con el usuario y le solicitaba al usuario que sintetice la conversación en una *query* para aplicar en el retrieval o semantic search, y la segunda enviaba una system prompt al LLM con instrucciones y contexto conseguido en el retrieval, junto con la conversación con el usuario para generar una respuesta. Esta estrategia daba resultados satisfactorios pero tenía la desventaja de que siempre se hacía una query en la vector store, incluso si no era necesario. Esto generaba una latency que muchas veces era innecesaria -cuando no se estaba necesitando productos para responder-, y también generaba una insistencia excesiva en la recomendación de productos. Al tener siempre productos en el contexto, el modelo tendía a recomendarlos aunque se le instruyera que haga más preguntas para entender las necesidades del usuario antes de hacer una recomendación.

Luego OpenAI anunció Function calling, es decir la opción de proveerle funciones al modelo que él puede invocar cuando lo considere necesario. De esta forma, le proveímos al modelo una función para buscar productos y él pasó a tomar la decisión de cuándo hacer la búsqueda. Esto nos permitió mejorar mucho la calidad de las respuestas, superando las limitaciones antedichas. Asimismo, nos dio la versatilidad de poder agregar diferentes funciones a nuestros sistemas. En el caso de CasasHub, programamos diferentes funciones para la búsqueda de propiedades, permitiendo buscar por características (i.e. neighbourhoods, bedrooms, bathroom, price, query -para ejecutar el Semantic Search-) o por nombre. Los valores de las características eran provistas por el modelo por parámetro a la función correspondiente. Potencialmente, el patrón de Function calling nos permitiría agregar funciones como *"agregar al carrito"* con gran facilidad.

6.2.3 *Fine-tunning*

Fine-tunning, en el contexto de LLMs, es un proceso de aprendizaje supervisado que tiene por objetivo hacer que un modelo genere mejores textos en un caso de

uso especificado por ejemplos que se proveen en forma de un dataset. El proceso tiene un costo de entrenamiento. OpenAI recomienda utilizarlo en los siguientes casos [32]:

1. Estableciendo el estilo, tono, formato u otros aspectos cualitativos.
2. Mejorando la fiabilidad para producir un resultado deseado.
3. Corrigiendo fallos al seguir indicaciones complejas.
4. Manejando muchos casos límite de maneras específicas.
5. Realizando una nueva habilidad o tarea que es difícil de articular en una indicación.

En *wisello*, si bien queríamos establecer un estilo y tono, preferimos hacerlo mediante *Prompt Engineering* y ahorrarnos las complicaciones de aplicarle *fine-tunning* a un modelo. En cuanto a la tarea de alimentar al modelo con los datos de los catálogos, las técnicas de RAG resultaban más apropiadas ya que permiten mantener una base de datos independiente al modelo, y los datos -en este caso, productos- proporcionados en el *contexto* se ven mucho más fielmente representados en las respuestas. Comparando RAG vs. fine-tunning para el aprendizaje de información, el primero da mejores resultados y mantenerlo actualizado tiene un menor costo.

El proceso de *fine-tunning*, asimismo, supone una restricción en la actualización de modelos, ya que se elige un modelo al que se le aplicará *fine-tunning* y este quedará inalterado. Esto obstruye la versatilidad de poder ir actualizando el producto a modelos más recientes con mayor performance, funcionalidades y menor costo. Además, OpenAI aconseja siempre probar técnicas de *Prompt Engineering* antes de hacer un *fine-tunning* por sus varias ventajas sobre éste relacionadas al feedback loop, versatilidad y transparencia. [32]

6.2.4 Caching

Ha emergido la tendencia de guardar respuestas en una memoria cache de modo que si se repite una *prompt*, se utilizará la salida ya generada. Esto permite disminuir significativamente la latencia en esos casos y reducir los costos que provienen de API calls al modelo. En *wisello*, implementamos un cache utilizando la librería `GPTCache` que, al momento de seleccionarla, era una de las implementaciones más robustas del patrón [33].

6.2.5 Guardrails

La dificultad de controlar las salidas del sistema en todos los casos ha dado lugar a la implementación de Guardrails, definidos como sistemas de verificación de salidas de LLMs. Sin entrar en detalle, estos mecanismos todavía tienen implementaciones tempranas y sus casos de aplicación varían entre guías de estructura, de sintáctica, control de seguridad, controles de inputs, y más. No juzgamos que los beneficios de este patrón fueran lo suficientemente prioritarios como para que decidamos implementar Guardrails en el desarrollo de *wisello*, pero podría ser muy útil en el caso de implementar *wisello* para un cliente con restricciones muy rigurosas.

6.2.6 Defensive UX

El paradigma de aplicaciones basadas en LLMs resulta en muchas complejidades de User Experience. El formato chat, que utilizamos en wisello, pone un gran peso en el usuario ya que las opciones de uso no son predeterminadas sino que las genera el usuario mismo con sus mensajes. Para facilitar el inicio de las conversaciones, utilizamos preguntas predeterminadas 6.2 que ejemplifican el uso y proveen al usuario un inicio rápido en la experiencia de uso.

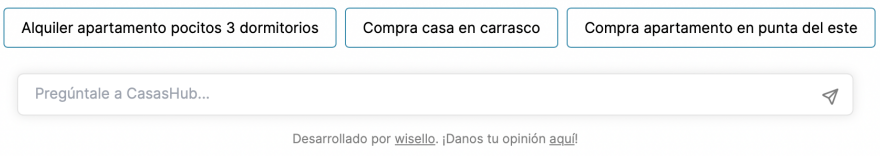


Figura 6.2: Preguntas predeterminadas al inicio de una conversación.

En [25] se prescribe el uso de una estrategia de diseño que tome medidas para abordar las limitaciones del desarrollo de aplicaciones basadas en LLMs. Estas limitaciones pueden ser la frecuente falla en proveedores de LLMs o las respuestas inexactas, entre otras. Una de las estrategias que utilizamos en este sentido fue la inclusión de imagenes con links a los productos recomendados en cada mensaje de *wisello*. Además de ser una feature útil pues permite a los usuarios acceder a los artículos y eventualmente comprar, también es una forma de asegurarnos que tienen un acceso a la información fidedigna -teniendo en cuenta que *wisello* puede *alucinar* características de los mismos-.

Figura 6.3: Imágenes con los productos recomendados en el mensaje.

Otra estrategia de **Defensive UX** aplicada fue limitar la conversación a máximo 10 mensajes. Esta funcionalidad tiene como objetivo evitar el error de `MAX_TOKENS`. Los modelos GPT tienen definidos un máximo de tokens tanto para la entrada como para la salida, **combinadas**. Por tanto, si la *prompt* de entrada es muy grande, deja menos espacio para la salida. Una vez que se alcanza el límite total de tokens (entrada + salida), el modelo no genera más texto, por lo que se detiene en medio de la respuesta, abruptamente. Recordemos que la historia de la conversación en *wisello* es parte de la *prompt* de entrada, porque queremos que *wisello* siga el hilo de la conversación. Por tanto, si esta es muy larga, probablemente caigamos en este error (y de hecho, al principio, nos pasaba). Por ende, como equipo, calculamos cuántos mensajes (peor caso) podía tener el usuario antes de que se agoten los tokens, y fijamos este número límite en 10, para evitar que se agoten los tokens, y las respuestas siempre se completen.

6.2.7 Recopilación de feedback de usuarios

La recopilación de feedback es muy importante para evaluar el desempeño de *wisello*. Proveemos dos formas de generar feedback. La primera es la opción de dar *like* o *dislike* a un mensaje y la segunda es un link a un feedback form:

Figura 6.4: Feedback: Like/Dislike en los mensajes.

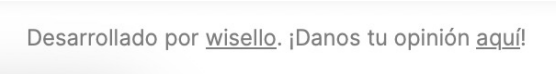


Figura 6.5: Feedback: Form.

6.3 Gestión de datos

La gestión de datos es un pilar fundamental de las aplicaciones basadas en LLMs. Por un lado, la calidad de los datos define el rendimiento de los modelos, y por otro, los datos utilizados pueden ser sensibles y requerir medidas de protección.

En *wisello* contamos con data pipelines que consumen el catálogo de productos del cliente y prepara los mismos como JSONs, para luego ser vectorizados y almacenados en la vector database. Es a través de ellas que aseguramos la calidad y estructura de datos.

En cuanto al desafío de la seguridad de datos y confidencialidad, hacemos notar en primer lugar que los tipos de datos que manejamos nunca son particularmente sensibles o privados. En todos los casos usamos información públicamente disponible en los sitios web de los e-commerce. De todos modos, la pregunta por la seguridad de datos ha surgido como preocupación de parte de nuestros clientes y también fue tomado en cuenta en el diseño de nuestros sistemas. Para abordar este desafío, nos aseguramos de que los modelos de LLM que utilizamos estén diseñados para garantizar que la información procesada a través de la API no se utilice para el entrenamiento de modelos [34]. Lo mismo en el caso de Pinecone, nuestro proveedor de vector store. De esta forma, *wisello* es capaz de proveer soluciones innovadoras manteniendo un compromiso firme con la seguridad y la privacidad de los datos.

6.4 Reuniones con expertos del área

6.4.1 Reunión con Lic. Juan Gabito

En setiembre del 2023 nos reunimos con el Lic. Juan Gabito, profesor asociado a la cátedra de Base de Datos, en búsqueda de apoyo en el entendimiento de embeddings y vector databases.

Nuestro principal problema surgía de que, al realizar búsquedas semánticas (Semantic Search) en nuestra base de datos vectorial `Pinecone`, el score de similaridad de los documentos devueltos en la *query* no era representativo: era muy parecido para todos los documentos, pese a que semánticamente un documento lógicamente *tenía* que ser más parecido a la *query* que otro, por ejemplo.

Junto a Juan, concluimos que:

- Teníamos que hacer embeddings de los datos que se iban a utilizar para el Semantic Search, todo el resto o no era necesario almacenarlo, o se incluía en la metadata para llevar a cabo un filtrado tradicional.
- Preprocesar el ruido. No tiene sentido generar embeddings de texto que es basura. Por ende, quitar el ruido, por ejemplo, sacando las `stop words` del idioma.
- La separación de la información en elementos independientes para realizar los embeddings se estaba realizando de forma correcta: cada embedding representaba un producto particular, con lo cual Juan estuvo de acuerdo.

Todo este conocimiento se tradujo en una mejora en nuestras data pipelines de preprocesamiento y en el discernimiento acerca de qué información incluir en la data o en la metadata de un vector.

6.4.2 Reunión con Dr. Sergio Yovine

Más adelante, en diciembre, tuvimos otra reunión de intercambio con el investigador Dr. Sergio Yovine. En este oportunidad discutimos sobre tres temáticas fundamentales. La primera fue la comparación entre técnicas de Reinforcement Learning from Human Feedback (RLHF) y Fine-tunning, en donde concluimos que entre ellas la primera opción a probar siempre debería ser fine-tunning, y profundizamos sobre la naturaleza de ambas. Luego ahondamos en el tema de la

performance de una aplicación basada en LLMs, y discutimos sobre posibles mejoras de *wisello* como la implementación de fine-tunning del modelo de embeddings para que la búsqueda semántica sea más precisa, las diferentes formas en que puede aplicarse el patrón de caching, entre otras. Finalmente, también reflexionamos sobre las maneras en las que se puede conseguir retroalimentación de los usuarios.

Figura 6.6: Entrevista con Dr. Sergio Yovine.

7 Arquitectura

En el siguiente capítulo se transmitirá una visión detallada del sistema y de la arquitectura de la solución. Se incluye una descripción de la arquitectura, los principales mecanismos que se desarrollaron para su construcción y despliegue, así como las decisiones de diseño que se tomaron para poder cumplir con los requerimientos no funcionales especificados para el proyecto *wisello*.

7.1 Descripción de la arquitectura

En primer lugar, se describirá la arquitectura mostrando las vistas de módulos, componentes y conectores, y despliegue, con el fin de de hacer una revisión total y completa del sistema, desde todas sus perspectivas y desde el punto de vista de los distintos interesados. Asimismo, se incluyen también diagramas de flujo de la información a través de la infraestructura de la solución.

7.1.1 Vista de módulos

7.1.1.1 Vista de descomposición

Representación primaria

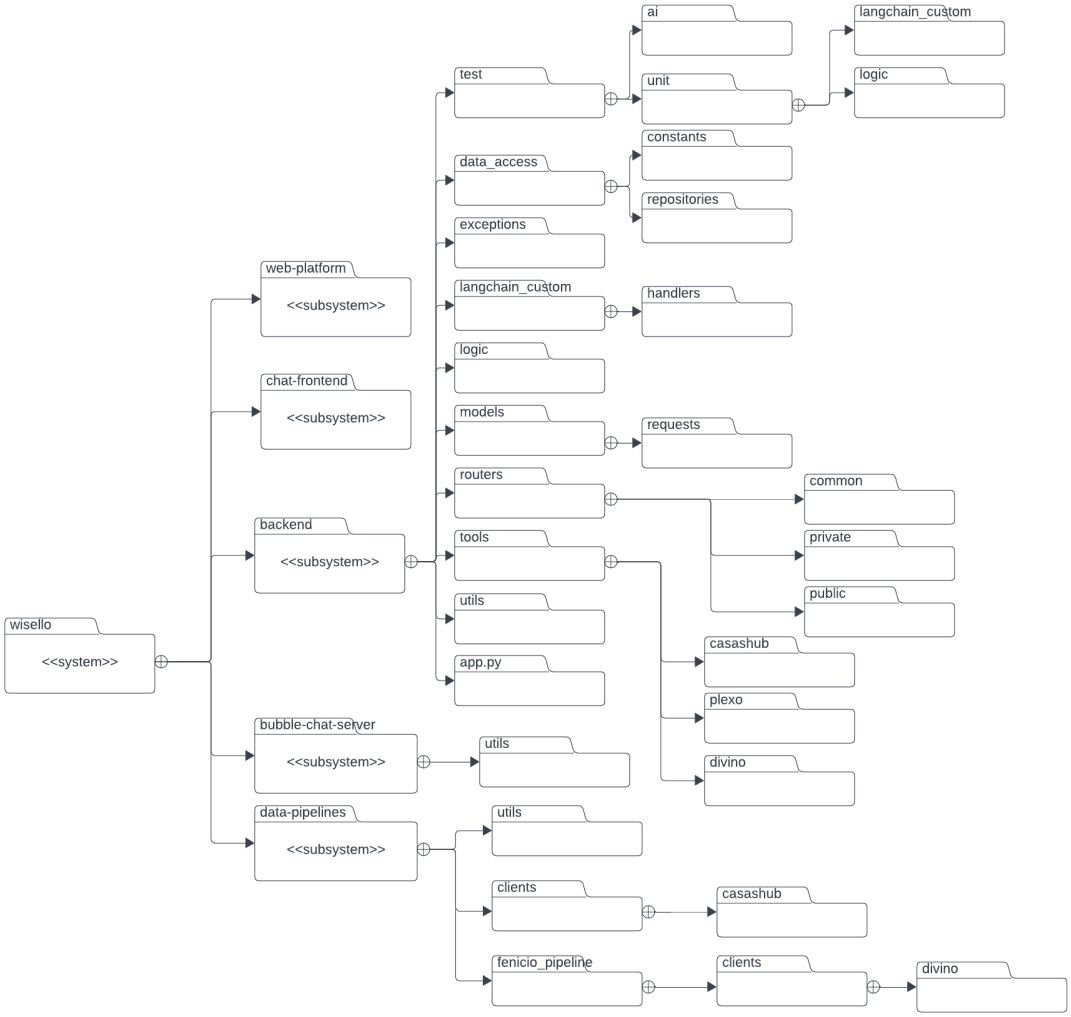


Figura 7.1: Vista de descomposición de *wisello*

Catálogo de elementos

Subsistema Backend

Elemento	Responsabilidades
app.py	Módulo encargado de inicializar la aplicación del backend, definiendo el puerto en el que escuchará, el *host* en el que correrá y configuración de *logging*.
routers	Módulo que contiene los enrutadores que definen la API del backend, con los *endpoints* expuestos por la misma.
routers/common	Módulo que contiene funciones que son utilizadas por todos los enrutadores en común (como por ejemplo: el manejo de excepciones a nivel de la API).
routers/public	Módulo que contiene los enrutadores públicos, es decir, aquellos *endpoints* que no necesitan autenticación por parte del cliente (como por ejemplo: login, health, entre otros).
routers/private	Módulo que contiene los enrutadores privados, es decir, aquellos *endpoints* que sí necesitan autenticación por parte del cliente (como por ejemplo: shops, metrics).
logic	Módulo encargado de llevar a cabo la lógica de negocios del sistema; desde aquí se gestionan las tiendas, las conversaciones, las invocaciones al chat, el envío de emails, métricas, feeedback de usuarios y productos recomendados.
langchain_custom	Módulo encargado de utilizar, y redefinir aspectos, del framework Langchain, para fines propios de *wisello*. En este encontramos la definición del modelo LLM propietario **OpenAIFunctionsCacheLLM**, así como los mecanismos para la creación del agente que lo utiliza, el caché de respuestas GPT, y la definición del modelo **BERT** como complemento de GPT.
langchain_custom/handlers	Módulo que contiene todos los *handlers* (abreviación de *callback handlers*) del sistema. Estos son utilizados para subscribirse a eventos que pueden suceder durante las distintas etapas de la ejecución tanto de un agente, una *chain*, *tool*, o modelo LLM. En nuestro caso, se utilizan para el trackeo de costos, streaming de respuestas y feedback de usuarios.

tools	Módulo de personalización. Es encargado de crear y definir las *tools* (funciones, funcionalidades) que estarán disponibles para los usuarios en el chat de cada tienda.
tools/casashub	Módulo de personalización (*tools*) desarrollado para Casashub.
tools/plexo	Módulo de personalización (*tools*) desarrollado para Plexo.
tools/divino	Módulo de personalización (*tools*) desarrollado para Divino, realizado para una demo.
data_access	Módulo encargado de la conexión con las bases de datos, y el acceso a datos. En nuestro caso: una base de datos no relacional **mongodb**, una base de datos vectorial **pinecone** y un caché **redis**.
data_access/constants	Módulo que define constantes como enumerados, para ser utilizados en el código.
data_access/repositories	Módulo encargado del acceso a datos, conteniendo los repositorios que se encargan de la interacción con las distintas bases de datos para realizar las acciones CRUD de cada entidad.
models	Módulo que contiene los modelos que se utilizan a nivel de lógica de negocios, siendo algunos de ellos: shop, chatbot, pinecone_query, email, entre otros.
models/requests	Módulo encargado de la definición de los modelos de los *body* de las distintas requests.
tests	Módulo que contiene los distintos tests desarrollados para testear el backend del sistema.
tests/ai	Módulo que contiene los tests de Inteligencia Artificial que evalúan que la respuesta del modelo sea correcta a una serie de preguntas base de una tienda.
tests/unit	Módulo que contiene los tests unitarios desarrollados para testear los módulos logic y langchain_custom.
exceptions	Módulo que contiene la definición de las excepciones propietarias definidas para el sistema, con el fin de tener un manejo de errores personalizado.
utils	Módulo que contiene utilidades comúnes al subsistema del backend.

Subsistema bubble-chat-server

bubble-chat-server	Subsistema involucrado en la integración con el e-commerce del cliente. Su única responsabilidad es devolver el código **javascript** correspondiendiente a la burbuja clickeable que aparecerá en el e-commerce del cliente, y a través de la cual los usuarios podrán utilizar *wisello*.

Figura 7.2: Ejemplo del mecanismo de integración *bubble chat* en acción

Subsistema data-pipelines

data-pipelines	Subsistema encargado de acceder, procesar, vectorizar y luego almacenar, el catálogo de productos de la tienda cliente en la base de datos Pinecone, para la posterior recomendación de productos.
clients/casashub	Módulo encargado de acceder la API de propiedades de Casashub, y procesar el catálogo de propiedades para luego almacenarlas en Pinecone.
fenicio_pipeline	Módulo genérico capaz de acceder el catálogo de productos de cualquier e-commerce de fenicio, procesarlo, y almacenarlo en Pinecone.
fenicio_pipeline/clients	Módulo que contiene, para cada tienda: el catálogo de productos XML, y el catálogo procesado JSON (pronto para ser vectorizado y almacenado).
utils	Módulo que contiene utilidades comúnes a las diferentes data_pipelines; en su mayoría, funciones relacionadas al manejo de la base de datos Pinecone.

Justificaciones de diseño

En primer lugar, podemos ver que se decidió separar el `backend`, de los subsistemas `bubble-chat-server` y `data-pipelines`. Esta decisión se tomó en pos de maximizar la coherencia semántica, creando subsistemas más cohesivos y desacoplados entre sí:

- `backend`: encargado de la gestión de tiendas, la interacción y configuración del modelo de de IA conversacional, y las funcionalidades del chat.
- `bubble-chat-server`: encargado de la integración con el e-commerce del cliente.
- `data-pipelines`: encargado de la interacción con el e-commerce del cliente y el procesamiento de los datos del catálogo.

Lo mismo sucede para los subsistemas `chat-frontend` y `web-platform`, que fueron separados en módulos distintos, de forma que que cada uno de ellos pueda desarrollarse y escalar en forma independiente, sin impactar uno en el otro. De esta forma, cada uno de lleva a cabo su responsabilidad de proveer, en cada caso:

- `chat-frontend`: La UI para el chat de *wisello* que aparecerá en el e-commerce del cliente, con las funcionalidades descritas en las secciones anteriores.
- `web-platform`: La UI para la plataforma de configuración y monitoreo de la herramienta para las tiendas que adquieren *wisello*.

Es importante considerar que estamos trabajando con tecnologías muy nuevas, que están cambiando todo el tiempo, por lo que la modularidad y el desacoplamiento adquieren un papel muy importante. Esto se debe a que los avances en el entorno tecnológico pueden requerir ajustes, y es importante que estos se mantengan contenidos dentro de las fronteras del módulo, minimizando el impacto en el sistema global.

Por otra parte, esta vista evidencia también la decisión de utilizar una arquitectura por capas de abstracción, la cual guió el desarrollo del sistema. Esta nos permitió asignar responsabilidades fácilmente, de forma que, como desarrolladores, podíamos entender fácilmente el flujo de dependencia e interacción entre los distintos módulos de la aplicación, y luego trabajar y razonar sobre esta sencillamente.

Lo podemos visualizar mejor a través de la vista de **layers**:

7.1.1.2 Vista de layers

Representación primaria

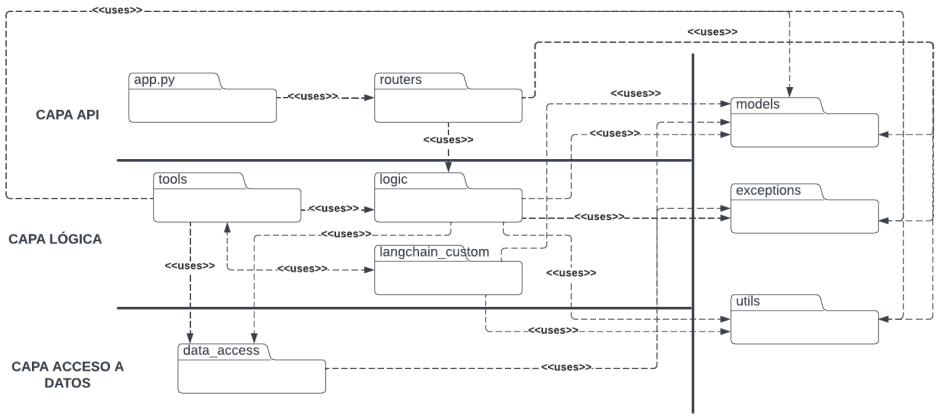


Figura 7.3: Vista de capas de *wisello*

Como se puede ver, en la primera capa encontramos la capa API, la cual se compone como el punto de entrada al sistema y donde se recibe la petición del cliente. Una vez recibida, esta capa utiliza la capa lógica para atender la solicitud. La capa lógica es la encargada de llevar a cabo la lógica de negocios: realiza validaciones sobre los datos enviados, y luego ejecuta el flujo requerido según la acción invocada. En esta capa participan también los módulos `tools` y `langchain_custom`, pues son módulos relacionados al modelo de IA conversacional, en donde se define la lógica detrás de la configuración (hiperparámetros, personalización según la tienda) e invocación del modelo. Por último, encontramos la capa de acceso a datos, que encapsula la interacción con las distintas bases de datos. Existe, además, una capa vertical con los módulos `models`, `exceptions` y `utils` que son utilizados por todas las capas de forma transversal.

Utilizando esta arquitectura, construimos capas altamente cohesivas y desacopladas entre sí. De esta forma, las capas superiores asumen que existe un mecanismo de menor abstracción debajo de ellas, con la interfaz que requieren, y no conocen los detalles de implementación de las capas subyacentes, haciendo posible poder cambiar una capa de bajo nivel por otra que cumpla con la interfaz requerida, sin impactar sobre las capas superiores que la consumen.

7.1.2 Vista de componentes y conectores

Representación primaria

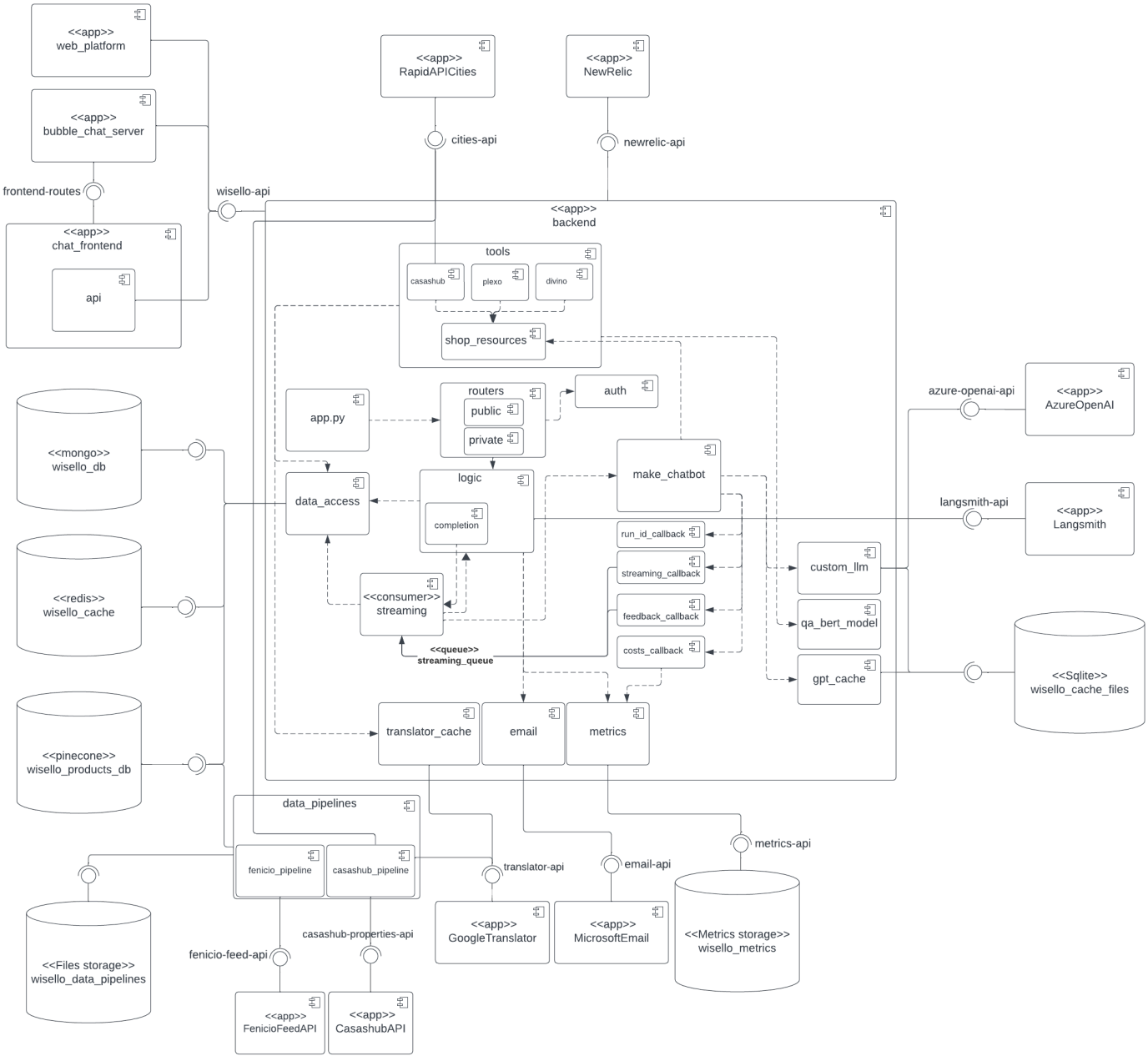


Figura 7.4: Vista de componentes y conectores de *wisello*

Catálogo de componentes/conectores

Componente/conector	Responsabilidades
auth	Componente encargado de autenticar y autorizar usuarios en el sistema.
completion	Componente encargado de llevar a cabo la lógica de negocios para devolver una respuesta conversacional y personalizada al usuario cuando este invoca al chat con un determinado *input*.
make_chatbot	Componente encargado de crear el objeto `chatbot`, con su debida configuración.
custom_llm	Componente que define el modelo LLM de *wisello*, que interactúa con OpenAI para generar las respuestas, y con GPTCache para cachear y obtener respuestas cacheadas.
qa_bert_model	Modelo BERT autohospedado de pregunta-respuesta, utilizado para enriquecer las *prompts* enviadas a GPT (OpenAI).
gpt_cache	Componente encargado de cachear respuestas de GPT ante determinada *prompt*. Es decir, cachea pares (*prompt*, respuesta).
wisello_cache_files	Base de datos `SQLite` que actúa como *cache storage* para gpt_cache, guardando los pares antedichos. Posee la ventaja que la base de datos se guarda en un único archivo.
streaming_queue	Cola de mensajes en la cual se encolarán los tokens recibidos de OpenAI, para ser enviados al usuario.
streaming_callback	Callback encargado de encolar a streaming_queue los *tokens* recibidos de la respuesta de OpenAI.
feedback_callback	Callback encargado de encolar el feedback de la búsqueda a streaming_queue, para ser enviado al usuario.
streaming	Componente que consume los *tokens* de streaming_queue, y los envía al usuario vía HTTP.
costs_callback	Callback encargado del trackeo de costos en cada llamada a OpenAI y su posterior almacenamiento en wisello_metrics.
run_id_callback	Callback encargado de almacenar el `run id` asignado en runtime, necesario para Langsmith.

metrics	Componente encargado de gestionar las métricas generadas por el sistema, y comunicarse con wisello_metrics para su almacenamiento.
wisello_metrics	Componente de *storage* encargado de almacenar las métricas que son generadas por el sistema.
email	Componente encargado de gestionar el envío de emails del sistema.
MicrosoftEmail	Servicio de Azure utilizado para enviar emails.
translator_cache	Componente encargado de comunicarse con GoogleTranslator para llevar a cabo la traducción de textos, así como cachear estas traducciones para su posterior acceso sin costo y más rápido.
GoogleTranslator	Librería de Google utilizada para traducir textos.
wisello_products_db	Base de datos vectorial Pinecone, en donde se almacenan los productos para cada tienda de forma vectorizada.
data_pipelines	Componente encargado de acceder, procesar el catálogo de productos del e-commerce, y almacenarlo en wisello_products_db.
fenicio_pipeline	Componente que se comunica con la API de productos de Fenicio, procesa el catálogo del e-commerce correspondiente, y hace la ingesta a wisello_products_db.
casashub_pipeline	Componente que se comunica con la API de propiedades de Casashub, las procesa, y hace la ingesta a wisello_products_db.
wisello_data_pipelines	Filesystem donde se persisten los archivos JSON resultado del procesamiento de las *data pipelines*.
wisello_cache	Caché redis utilizado para cachear productos recomendados y el `run id` de langchain.
wisello_db	Base de datos mongodb en donde se almacena la información de las tiendas, conversaciones de usuarios, *leads* generados y configuración del chat.
tools	Componente de personalización del chat, al definir las funcionalidades que este tendrá para cada tienda.
shop_resources	Componente dentro de tools que define una interfaz que toda tienda debe implementar: `get_tools` y `get_prompt`.
NewRelic	Aplicación utilizada en el backend para servicios de APM, y almacenamiento de logs.
Langsmith	Aplicación que permite monitorear, evaluar y testear aplicaciones que trabajan con LLMs.

Interfaces

Interfaz:	wisello-api
Componente que la provee:	backend
Servicios	**Descripción**
▪ Generar una respuesta conversacional ante determinado pedido/consulta del usuario incluyendo la recomendación de productos. ▪ Obtener los productos recomendados en una respuesta dada. ▪ Gestión de tiendas. ▪ Acceso a métricas de una tienda. ▪ Log-in. ▪ Registrar feedback de usuario ante una respuesta del chat. ▪ Enviar email ante nuevo *lead* al encargado de la tienda.	API que expone el backend de *wisello*, con los endpoints necesarios para que los usuarios interactúen con el sistema. Por ello, es consumida por `web-platform`, `bubble-chat-server`, y `chat-frontend`.

Interfaz:	cities-api
Componente que la provee:	RapidAPICities
Servicios	**Descripción**
Dada una latitud, longitud y el ID del país (UY), devuelve el nombre del barrio y la región (departamento) en el que se ubican las coordenadas.	Utilizamos esta API para las propiedades en Casashub, de forma de mantener el mismo criterio geográfico entre las propiedades en nuestra base de datos, y la interpretación de los barrios ingresados en el chat. Por ende, al momento de correr `casashub_pipeline`, se interpreta el barrio de la propiedad utilizando esta API, y cuando el usuario ingresa un barrio en el chat, se interpreta el barrio real, a nuestro efecto, utilizando también esta API. A continuación se ilustra mejor este proceso con un diagrama de secuencia:

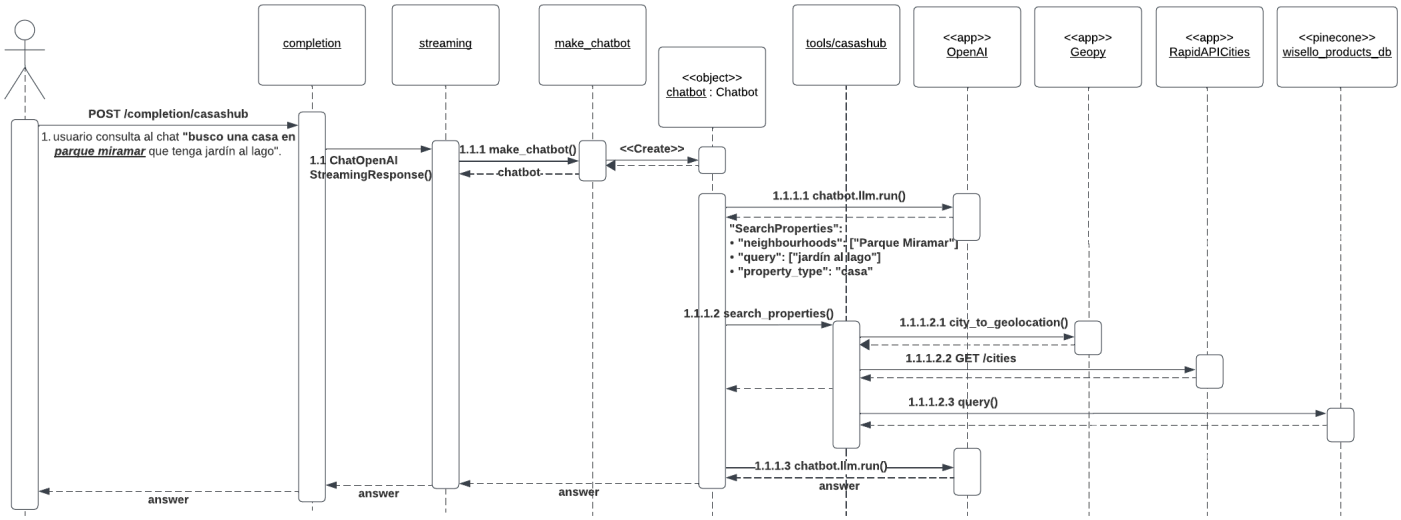


Figura 7.5: Diagrama de secuencia para recomendación de propiedades por barrio

Interfaz:	azure-openai-api
Componente que la provee:	AzureOpenAI
Servicios	**Descripción**
Dada un *prompt*, una serie de parámetros, y una serie de *tools* o *functions* disponibles, la API `/completion` devuelve la respuesta del LLM (GPT) para los mismos.	API consumida por `custom_llm`, para generar respuestas conversacionales para los usuarios.

Interfaz:	fenicio-feed-api
Componente que la provee:	FenicioFeedAPI
Servicios	**Descripción**
Dado un e-commerce de Fenicio, provee una respuesta en formato XML con el catálogo de productos del sitio (misma estructura para todas las tiendas).	API utilizada para acceder al catálogo de productos de un e-commerce de Fenicio por parte de `fenicio_pipeline`.

Interfaz:	casashub-properties-api
Componente que la provee:	CasashubAPI
Servicios	**Descripción**
Provee un JSON con todas las propiedades enlistadas en el portal.	API utilizada para acceder al catálogo de propiedades de Casashub por parte de `casashub_pipeline`.

Justificaciones de diseño

Patrón Observer

Un patrón del cual nos valimos para implementar las funcionalidades del streaming de la respuesta, el feedback de búsqueda provisto al usuario previo a la respuesta, y el trackeo de costos, fue el patrón Observer, provisto por Langchain.

Si observamos el diagrama, se puede ver que el componente `make_chatbot` interactúa con `run_id_callback`, `streaming_callback`, `feedback_callback` y `costs_callback`. Esto sucede porque subscribe estos *callbacks* a los eventos que puedan suceder durante la ejecución del `AgentExecutor` asociado al objeto `chatbot`. Cada uno de los suscriptores se subscribe a distintos eventos de los disponibles (ver `BaseCallbackHandler` debajo), para llevar a cabo su tarea. Lo vemos con mayor claridad en el siguiente diagrama de clases:

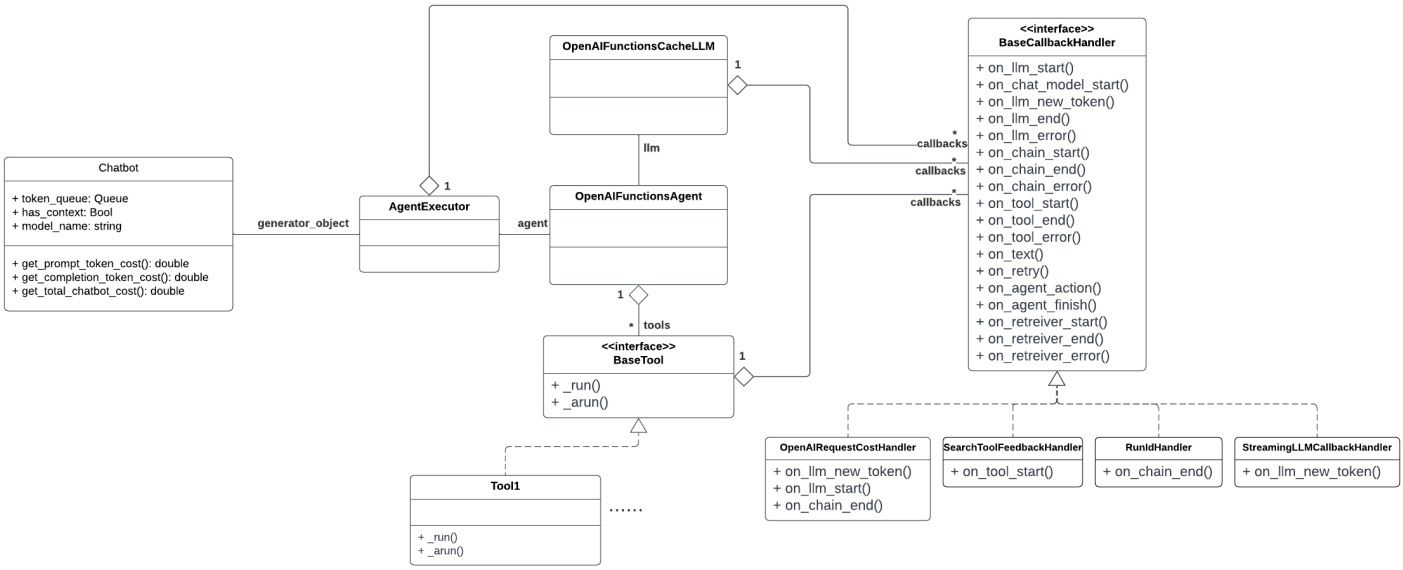


Figura 7.6: Patrón observer en *wisello*

A través de este patrón, encontramos una forma **extensible** de insertarnos dentro del flujo del agente `OpenAIFunctionsAgent`, pudiendo realizar operaciones disparadas por la notificación de que algo había sucedido en el flujo (una *tool* ha comenzado a ejecutarse, un nuevo *token* ha sido generado, etc). La aplicación de este patrón, en primer lugar, nos dio lugar a la posibilidad de poder transmitir los *tokens* de la respuesta al usuario, y poder así trabajar en el **RNF-PR01** (ver 5.2.1), en el que ahondamos a continuación.

Performance: streaming de los *tokens* de la respuesta

Como equipo, cuando planteamos que el tiempo de respuesta de *wisello* era un RNF muy importante para asegurar su calidad, nos topamos con el desafío de que cada respuesta del chat iba a ser única, y con ello, de largos distintos. Por ende, esto nos imposibilitaba utilizar la latencia total de la respuesta como una medida confiable y orientativa para medir la performance del sistema, pues una respuesta más larga demoraría más que otra más corta.

Por esta razón, y considerando que la API de `OpenAI` nos provee la capacidad de transmitir los *tokens* de la respuesta a medida que están disponibles, decidimos utilizar la medida ***time to first token***, como "el tiempo que demora en empezar a responder".

Se implementó, por tanto, el *streaming* de la respuesta para el endpoint de `completion` (chat). De esta forma, el usuario no debe esperar por la respuesta entera, sino que la recibe a medida se hace disponible. Vemos el comportamiento implementado para *wisello* a continuación:

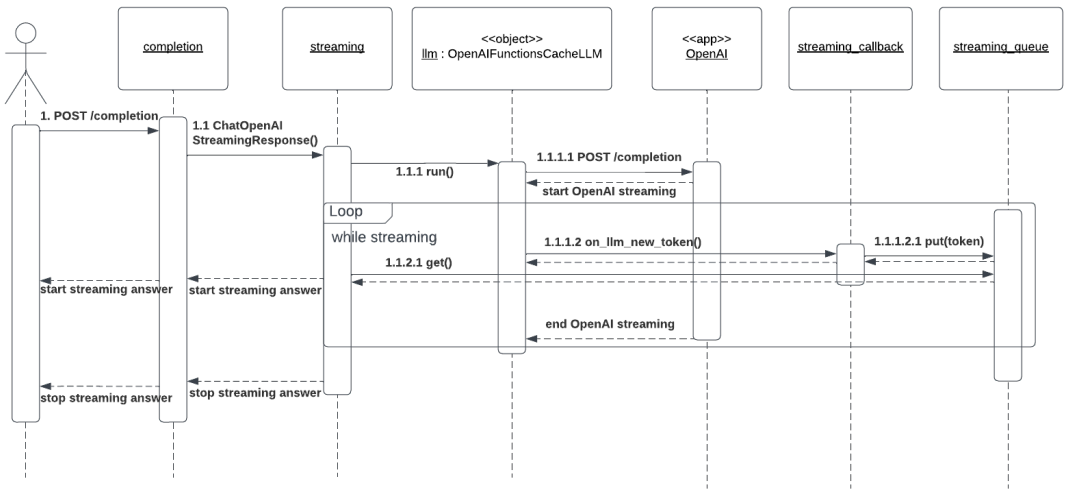


Figura 7.7: Diagrama de secuencia del *streaming* de la respuesta

El componente `streaming_callback` actúa como productor de la cola de mensajes `streaming_queue`. Al suscribirse al método `on_llm_new_token()`, recibe los *tokens* que generados por `OpenAI` y los encola. El componente `streaming` actúa como consumidor de esta cola. Para ello, inicia un `thread` y hace continuamente *polling* de `streaming_queue` en busca de tokens; cuando los recibe, los envía al usuario, quien los va recibiendo a medida que se hacen disponibles.

Performance: uso de GPT caché

Para cumplir con el **RNF-PR01**, decidimos hacer uso de la técnica de *caching*. Para ello, utilizamos la librería `gptcache`, la cual nos permite realizar *caching* de las respuestas de `OpenAI` en una base de datos `SQLite`, en formato: (*prompt*, respuesta). Ante una misma *prompt*, se considera un *cache hit* y se devuelve la respuesta *cacheada*, ahorrando latencia y costos incurridos por `OpenAI`. El caché se invalida, en su totalidad, en cada *update* del catálogo.

Para aumentar la cantidad de *cache hits*, se decidió procesar las *prompts* previo a su *caching*. Para ello, decidimos *cachear* únicamente los mensajes humanos (`HumanMessage`), pasarlos a minúsculas, y quitar espacios. De esta forma, por ejemplo, estas dos invocaciones tendrían un *cache hit*:

Quiero una casa con vista al lago y **quierounacasaconvistaallago**

Eventualmente, se podría optimizar este caché quitando las *stop words* (obteniendo algo como: `Hola! Quiero casa vista al lago`), aumentando así el número de *hits*. Incluso, se podría mejorar la invalidación del caché, solamente invalidando las propiedades que fueron actualizadas ese día.

Una optimización del caché que se intentó realizar es la búsqueda mediante *semantic similarity*, basada en una distancia. Si lo que se desea buscar en el caché se encuentra en un rango de distancia con otro, se lo considera un *hit*. Por ejemplo:

Quiero casa con vista al lago y **Quiero casa que tenga vista al lago**

Sin embargo, la búsqueda mediante similaridad semántica no se pudo realizar porque, al *cachear* toda la conversación, la conversación con el mensaje siguiente se hace muy parecida a su versión anterior, y la **distancia** de búsqueda no puede ser dinámica, dándose así *hits* erróneos. Por ejemplo:

Hola! Quiero una casa con vista al lago./Tiene piscina?/Tiene jardin?/Dónde esta ubicada?, y con el siguiente mensaje: **Hola! Quiero una casa con vista al lago./Tiene piscina?/Tiene jardin?/Dónde esta ubicada?/Gracias por la info!**. Esto termina en un *caché hit* (porque se parecen mucho), devolviendo la misma respuesta, cuando no es el caso.

Por ende, se implementó el caché presentando inicialmente. Particularmente, un desafío que nos presentó la implementación de este caché fue que tuvimos que integrar la lógica del *caching* al modelo. Nuestro modelo utiliza funciones por

lo que, si es necesario, hace dos llamadas contiguas a `OpenAI`: la primera para determinar qué función utilizar, y la segunda para responder al usuario. Se puede ver en este flujo (sin la intervención del caché) a continuación.

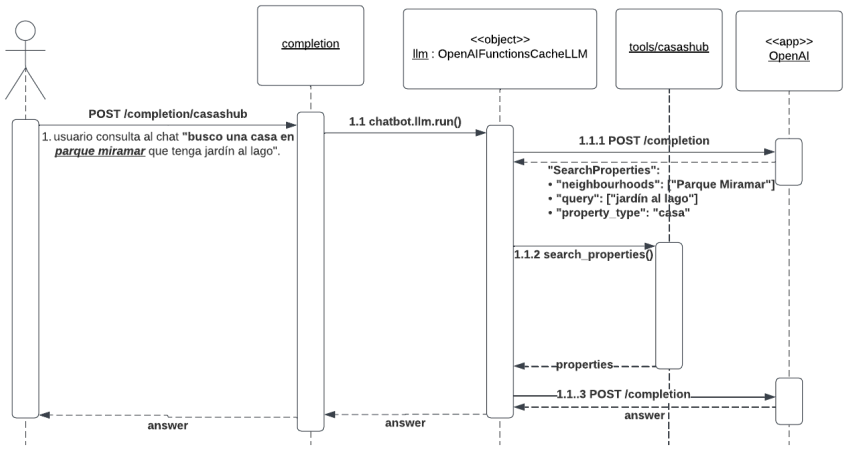


Figura 7.8: Diagrama de secuencia del uso de *functions*

Por tanto, fue necesario realizar *caching* de ambas llamadas, desarrollando así el modelo `OpenAIFunctionsCacheLLM`:

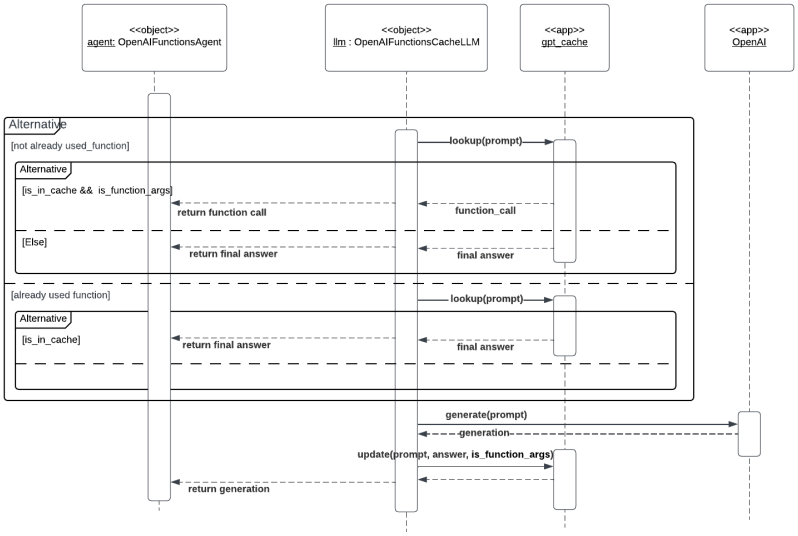


Figura 7.9: Diagrama de secuencia del *caching* de respuestas

En referencia al **RNF-PR01**, se realizó la prueba de carga con **100 usuarios concurrentes**, con un set de 15 preguntas distintas (*"Hola, estoy buscando una casa en Carrasco Sur con Patio*, *"Hola, quiero información acerca de Casashub*, *Hola, quiero un apartamento en pocitos de 3 cuartos y 2 baños "*, entre otras). El ***time to first token*** **promedio** resultado fue de `5.3s`, lo cual es ligeramente mayor al tiempo promedio propuesto de 5 segundos. Dadas las técnicas aplicadas y la cercanía del número, como equipo, estamos satisfechos; no solamente por el resultado en sí, sino por el razonamiento que tuvimos sobre qué métrica utilizar (*time to first token*) con las tecnologías disponibles, por qué esta era la métrica conveniente, y cómo esta se podía llevar a cabo a través del *streaming*.

Dado que estas fueron las primeras pruebas de performance del sistema, el equipo planea seguir trabajando sobre este atributo de calidad a medida se vaya avanzando en el desarrollo del sistema, para lograr llevar esta métrica por debajo de los 5 segundos tal como se lo había planteado inicialmente. Esto implicaría optimizar el caché con los mecanismos mencionados previamente para tener más *hits* y escalarlo horizontalmente. Además, el equipo cree que sería una buena idea contar con un monitoreo formal de cada parte implicada en la latencia del sistema, para tener claro cuáles son aquellas que están bajo nuestro control y sobre las cuáles podemos trabajar (es decir, aquellas partes que no se correspondan con latencia de terceros), y por ende, monitorear la evolución de estas partes.

Por más detalles de la prueba de carga ver el Anexo 15.3.

Uso de AzureOpenAI API vs. OpenAI API

Para mejorar la performance, otras de las decisiones que tomamos fue utilizar la API de OpenAI expuesta a través del servicio `AzureOpenAI`, en contraposición con utilizar la de `OpenAI` directamente. Según la comparación de latencia de ambas APIs en `Streaming mode`: ***"Azure boasts an impressive 7 to 8 times speed advantage"*** [35], permitiéndonos así una mejora en la latencia de nuestro sistema.

Para ello, se deployaron dos modelos en el servicio `AzureOpenAI`:

- Un modelo `text-embedding-ada-002` para crear los *embeddings*.
- Un modelo `gpt-35-turbo-16k` para generar las respuestas de *wisello*.

Seguridad: autenticación y autorización mediante tokens JWT

Con el fin de cumplir con los requerimientos no funcionales **RNF-SG01** y **RNF-SG02**, creamos un control de acceso basado en roles, distinguiendo entre los permisos otorgados al usuario **administrador** (wisello, únicamente), y aquellos otorgados a la **tienda**. Para ello, utilizamos como tecnología tokens JWT, y el responsable de autenticar y autorizar las entidades es el componente `auth` (ver diagrama 7.4).

El usuario **administrador** posee la capacidad de crear nuevas tiendas indicando un nombre y contraseña. Una vez creada, la tienda es capaz de hacer *login* (`get_api_key()`), obteniendo el token mediante el cual **autenticará** sus *requests* al interactuar con la aplicación web.

Además, este token contiene, de forma cifrada, a qué tienda corresponde:

```
1      expiration_time = datetime.datetime.now() + timedelta(days=1)
2      jwt_payload = {"shop": shop, "exp": expiration_time, "admin": is_admin}
3      encoded_jwt = jwt.encode(jwt_payload, jwt_secret_key, algorithm="HS256")
4
```

Esto nos permite cumplir con el **RNF-SG02**, pues al enviar un JWT token, inmediatamente el sistema reconocerá de qué tienda se trata, pudiendo realizar acciones u obtener datos únicamente de esta (**autorización**). Lo vemos con el ejemplo de `GET /shops` a continuación.

```
1      @shops_router.get("")
2      def get_shop(jwt_payload: str = Depends(verify_shop_api_key)):
3          if jwt_payload["is_admin"]:
4              return get_all_shops()
5          else:
6              shop = jwt_payload["shop"]
7              return get_shop(shop)
8
```

En relación a las contraseñas de las tiendas, estas fueron identificadas como datos sensibles del sistema. Para garantizar su protección, se utilizó *hashing*. Cuando una tienda es registrada o su contraseña es modificada, esta es *hasheada* para ser almacenada en la base de datos. De esta forma, es protegida ante posibles ataques, dado que el *hashing* es *one-way*, implicando que si la base de datos se encuentra comprometida, los atacantes no puedan acceder a las contraseñas originales de los usuarios. Luego, cuando el usuario se intenta loguear, se compara el *hash* de la contraseña intentada, con la que está almacenada, para **autenticar su identidad**.

Como podemos ver en el diagrama 7.4, los enrutadores fueron separados en dos componentes: aquellos públicos y aquellos privados, para mayor claridad.

Manejo de errores

Para responder adecuadamente a cada caso de error (acceso no permitido, falta de permisos, *request* inválida, etc) se utilizó un manejo de errores personalizado con una jerarquía de excepciones propietarias:

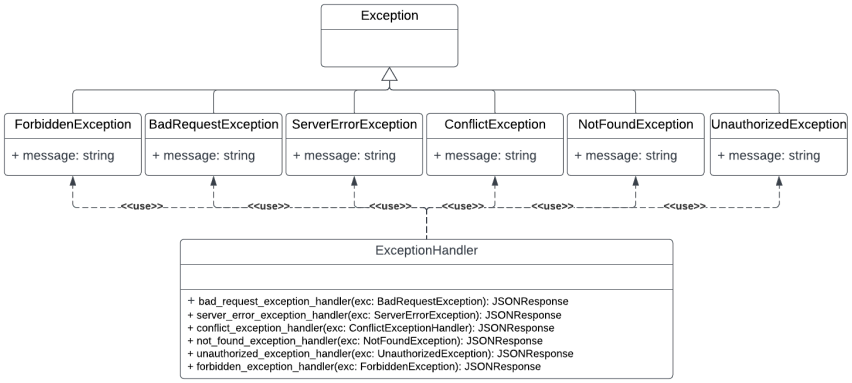


Figura 7.10: Jerarquía de excepciones

De esta forma, cada excepción se mapea con un código descriptivo para las respuestas del backend, y además se devuelve un mensaje propio explicando la naturaleza del error. La clase `ExceptionHandler` se encarga de definir los *exception handlers*, encargados de manejar el error lanzado durante la ejecución. Por ejemplo:

```
    try:
        payload = jwt.decode(api_key, key=jwt_secret_key, algorithms=["HS256"])
        return payload
    except jwt.exceptions.ExpiredSignatureError:
        raise UnauthorizedException("Provided api-key has expired")
    except jwt.exceptions.InvalidTokenError:
        raise UnauthorizedException("Provided api-key is invalid.")
    except jwt.exceptions.PyJWTError:
        raise UnauthorizedException("Something went wrong while verifying api-
    key")

```

A nivel del frontend, se muestra también un mensaje acorde al error sucedido.

Figura 7.11: Ejemplo de error a nivel de frontend

Extensibilidad a nuevas funcionalidades y tiendas

Para cumplir con el **RNF-EX-01**, que explica que se deben poder agregar nuevas tiendas y funcionalidades sin modificar código existente, aplicamos una serie de técnicas y patrones, que procederemos a explicar.

Como se puede observar en el diagrama 7.4, existe un componenente llamado `tools` que fue creado con el objetivo de proveer la personalización del chat para cada tienda. Para lograr que este componente sea extensible, se definió una interfaz `ShopResources`, con dos métodos, que toda tienda `shop` debe implementar:

`ShopResources:`

- `get_tools`: refiere a las funciones (*OpenAI functions*) disponibles que el agente de cada tienda tendrá a su disposición para responder al usuario.

  Por ejemplo, Casashub: `search_properties`, `search_individual_property`; Divino: `search_catalogue`; Plexo: `search_docs`.

- `get_prompt`: refiere a la *prompt* bajo la cual el LLM formulará su respuesta, en el caso de la tienda en particular.

  Por ejemplo, Casashub: *Tu nombre es wisello, trabajas para un portal inmobiliario llamado casashub. Un cliente está buscando propiedades en el portal. Utiliza la función 'SearchProperties' para buscar propiedades dentro del portal en base a las cualquier preferencia del usuario, por más básica o compleja que sea. Utiliza la función 'SearchIndividualProperty' para buscar una propiedad por su nombre o sku.*

De esta forma, cuando se desea agregar una nueva tienda, se implementa la interfaz `ShopResources` para la misma, sin necesidad de modificar código existente. Por otra parte, si se busca agregar una nueva funcionalidad a una tienda existente, todas las `tools` implementan la misma interfaz `BaseTool`, por lo que se debe implementar la interfaz con la nueva funcionalidad, y agregarla como retorno al `get_tools`, nuevamente sin modificar el código existente, sino que por el contrario, extendiéndolo.

Por otra parte, en relación a la instanciación de la implementación concreta de la interfaz, combinamos el uso de la táctica de **diferir enlaces** a tiempo de ejecución, con el **patrón factory** para crear cada implementación concreta. Según el nombre de la tienda, en tiempo de ejecución, se crea la implementación concreta que corresponde.

De esta forma, el componente `make_chatbot`, quien se encarga de crear el chat para la tienda, se mantiene agnóstico y desconoce con qué implementación en particular de `ShopResources` está trabajando. Lo vemos a continuación:

```
1       def make_chatbot(send_message: CompletionRequest) -> Chatbot:
2           shop_resources = return_shop_resources(send_message.shop) # factory
        crea la implementación concreta en tiempo de ejecución
3           tools = shop_resources.get_tools() # llamada polimórfica
4           prompt_template = shop_resources.get_prompt()  # llamada polimórfica
5
```

Por ende, a través del polimorfismo mediante el uso de interfaces, el patrón factory, y la táctica de diferir enlaces, se logró crear un mecanismo de extensibilidad que permite crear nuevas tiendas y agregar nuevas funcionalidad a tiendas existentes, sin modificar el código existente, sino extendiendo el código actual. A través de este mecanismo, la clase `MakeChatbot` trabaja de forma agnóstica con `ShopResources`, creando objetos `Chatbot` sin conocer de qué tienda trata, logrando así la extensibilidad deseada.

A continuación presentamos un diagrama de clases para enclarecer este mecanismo.

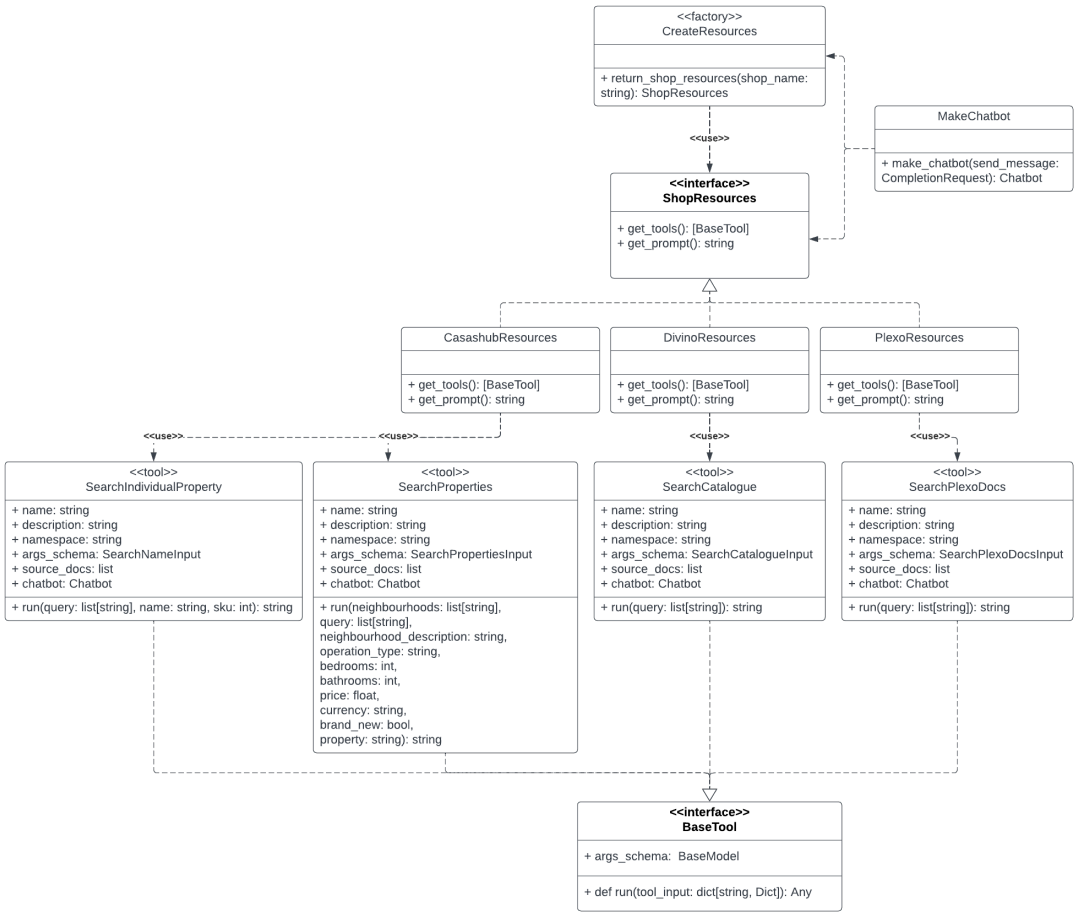


Figura 7.12: Mecanismo de extensibilidad a nuevas tiendas y funcionalidades

Como ejemplo, en una demo realizada para **Metropolitana**, un e-commerce de pisos y alfombras (no se incluye en los diagramas anteriores), la extensibilidad expuesta nos permitió crear fácilmente una `tool` llamada `BoxesNeeded` que calcula cuántas cajas de determinado piso se necesitan para un ambiente con determinado metraje (además de la `tool` usual `SearchCatalogue`). La vemos en acción:

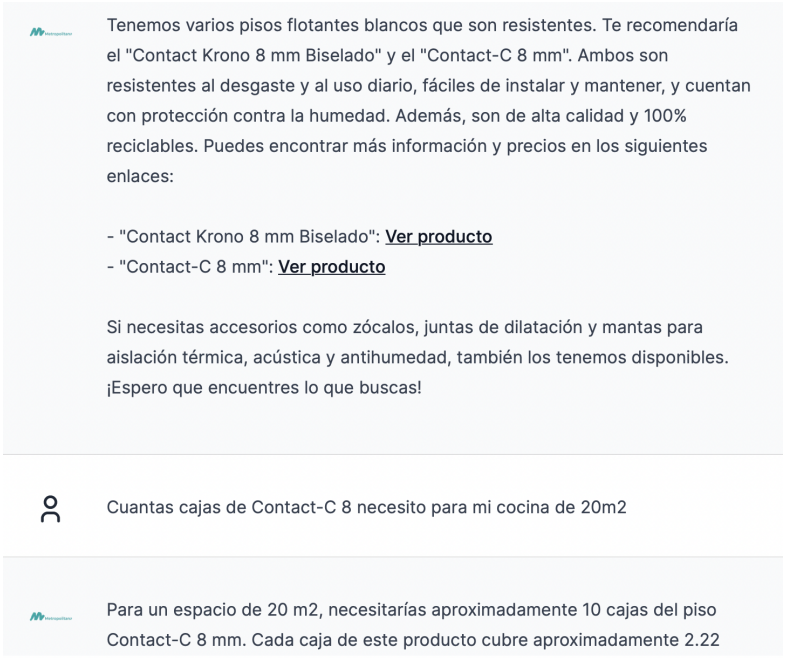


Figura 7.13: Extensibilidad a nuevas funcionalidades en acción con BoxesNeeded

Adaptadores como frontera de cambio

Como se puede observar en el diagrama 7.4, se construyeron varios componentes cuya única responsabilidad es la comunicación con APIs externas. De esta forma, actúan como adaptadores que contienen ante cambios o variaciones en estas APIs. Estos son:

- `translator_cache`: interactúa con la API de traducción de Google.
- `email`: interactúa con la API de mailing de Azure.
- `metrics`: interactúa con la API de Application Insights (Servicio de Azure para métricas de aplicaciones).

A futuro, si estas APIs cambian, el cambio impactará únicamente en estos componentes, evitando que impacte en el resto del sistema. Hacemos hincapié en el caso de `translator_cache`, dado que se aplicó el **Patrón Singleton**. Con este patrón, el objeto `TranslatorSingleton` puede guardar las traducciones ya consultadas en un diccionario en memoria, y evitar el *overhead* de tener que consultar traducciones ya realizadas. El porqué de las traducciones se explicó en la sección 6.1.4 (complemento de LLMs con otros modelos).

7.1.3 Vista de despliegue

Representación primaria

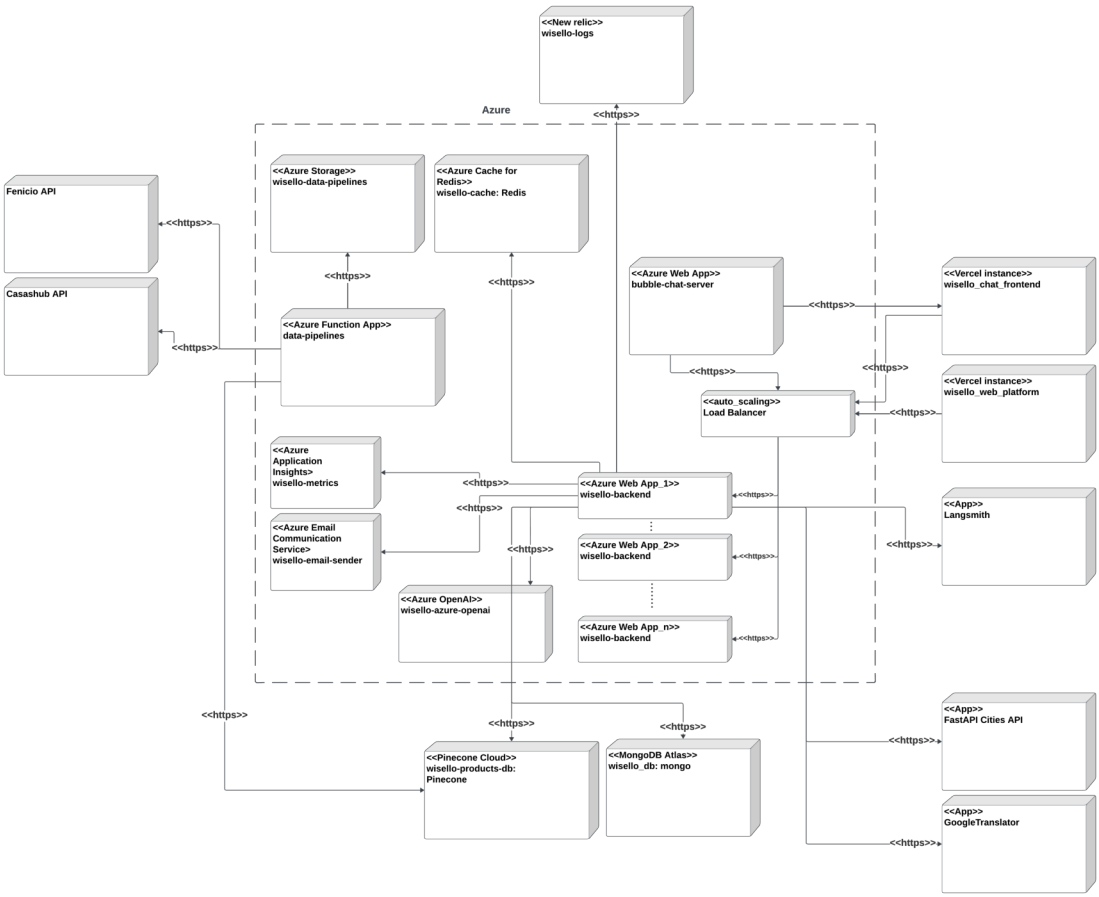


Figura 7.14: Vista de despliegue de *wisello*

Catálogo de elementos

Elemento	Responsabilidades
wisello-backend	Web app (servicio `App Services` de Azure) donde corre el backend de la aplicación.
bubble-chat-server	Web app (servicio `App Services` de Azure) donde corre el servidor que sirve la burbuja de *wisello* (mecanismo de integración con la tienda).
wisello-email-sender	Acceso a *endpoint* de servicio de email (servicio `Email communication services` de Azure).
wisello-metrics	Dashboard y telemetría del backend (servicio `Application Insights` de Azure).
wisello-data-pipelines	Storage account (servicio `Azure Storage` de Azure) donde se guardan los archivos JSON resultado del procesamiento de las *data pipelines*.
wisello-cache	Cluster redis alojado en `Azure Cache for Redis`.
wisello-chat-frontend	Frontend del chat de *wisello*, alojado en una instancia de `vercel`.
wisello-chat-frontend	Plataforma web de configuración y monitoreo de *wisello*, alojada en una instancia de `vercel`.
wisello-db	Base de datos de tiendas y conversaciones del backend, alojada en `MongoDB Atlas` (Mongo Cloud).
wisello-products-db	Base de datos vectorial de productos de *wisello*, alojada en `Pinecone Cloud`.
wisello-azure-openai	Model deployments de OpenAI GPT y ADA embeddings (servicio `Azure OpenAI` de Azure). Acceso a *endpoints* para ambos modelos.
Langsmith	Plataforma para monitoreo, evaluación y testing de aplicaciones basadas en LLMs.
FastAPI Cities API	API de localización de coordenadas.
GoogleTranslator	API de traducción de Google.
Casashub API	API de propiedades de Casashub.
Fenicio API	API de catálogos de productos de Fenicio eCommerce.

Justificaciones de diseño

Docker como tecnología de portabilidad

En relación al **RNF-PB-02** de Portabilidad, que explica que el sistema debe poder correr y ser desplegado en cualquier sistema operativo, decidimos utilizar como tecnología **docker**. El uso de contenedores nos permite *empaquetar* nuestra aplicación, aislándonos de un sistema operativo en concreto, y de dependencias que puedan existir en un entorno determinado. Con Docker, la aplicación corre en un entorno independiente al sistema, pues no depende de los archivos del sistema operativo subyacente sino que utiliza su propio sistema de archivos.

Para ello, se creó un `Dockerfile` para el backend, de forma de poder desplegarlo de forma *containerizada* en la nube. En nuestro caso, `Azure App Services`, por defecto, ya despliega sus aplicaciones como **docker containers**, por lo que no fue necesario utilizar este `Dockerfile`, pero el mecanismo fue construido en caso de que se desee, a futuro, cambiar de nube o de servicio de despliegue en Azure.

Por si a futuro se desea migrar a otra nube, se crearon adaptadores (7.1.2.0.2.8) para `email` y `metrics` (que utilizan el SDK de Azure, y por ende, se encuentran acoplados con Azure). Así, estos adaptadores aislan el acoplamiento con Azure, se mantienen como piezas menores acopladas a Azure, y son la única parte del código que se tendría que cambiar para migrar a otra nube.

A continuación vemos la evidencia, a través del archivo de logs de despliegue del backend, de que `Azure App Services` utiliza Docker para desplegar sus `Apps`:

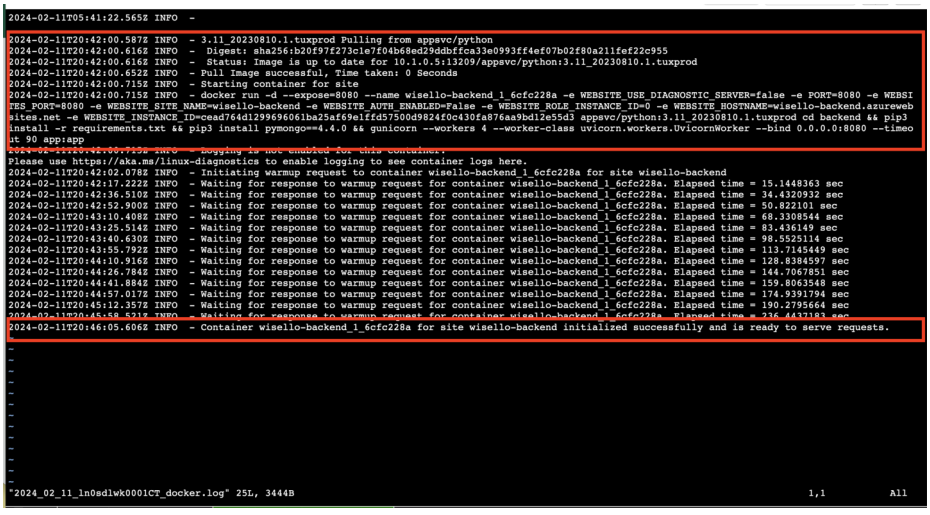


Figura 7.15: Logs de despliegue del backend de *wisello*

Por otra parte, en referencia al **RNF-DP-03**, se creó también un archivo **docker-compose**, para poder levantar de forma local todo el ecosistema del backend en un solo comando, utilizando `docker-compose up`. De esta forma, cualquier persona que se clone el proyecto, puede levantar el backend con este comando únicamente, y correrlo fácilmente. En este se incluyen tres servicios:

- backend: API del backend.
- wisello_db: base de datos mongo DB.
- redis: caché redis.
- (No se pudo incluír la base de datos Pinecone dado que no es *open source*).

En este archivo encontramos las variables de entorno para cada servicio, para `development`. Además, se creó también una `docker network`, de forma que los contenedores del backend se puedan invocar entre sí utilizando su nombre como host.

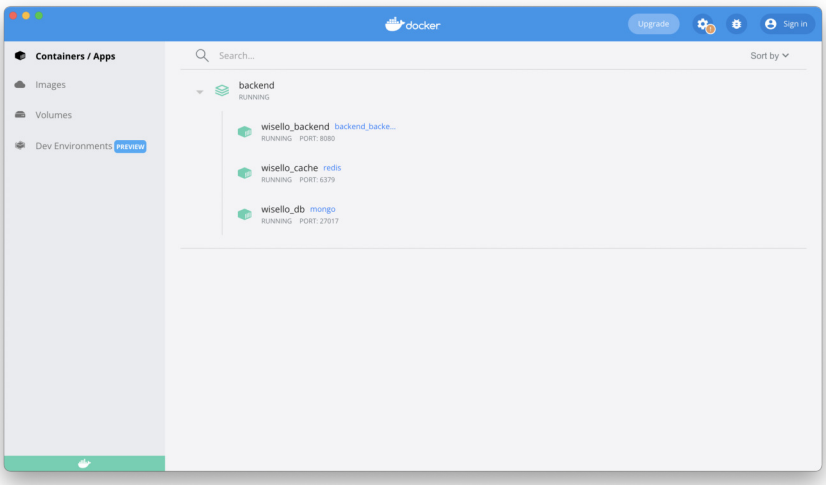


Figura 7.16: Docker-compose

Automatización del despliegue

A continuación, nos concentraremos en el **RNF-DP-02** de **continous deployment**, que afirma que debe existir un proceso de despliegue automatizado, para desplegar nuevas versiones de la aplicación con una sola acción.

En el caso de `wisello-backend`, la pipeline de despliegue se realizó con Github Actions. Para ello, utilizamos la acción `azure/webapps-deploy@v2` provista, que nos permite hacer un deploy de la aplicación a `App Services` desde la pipeline. Al hacer un `push` tanto a `develop` como a `main`, se hace un deploy de la aplicación a la Web App correspondiente (`wisello-backend` o `wisello-backend-dev` según la rama).

Lo mismo fue realizado para `bubble-chat-server` y `data-pipelines`. En el caso de `data-pipelines`, el deploy es a `Azure Functions`, utilizando la acción `azure/functions-action@v1, id: deploy-to-function`.

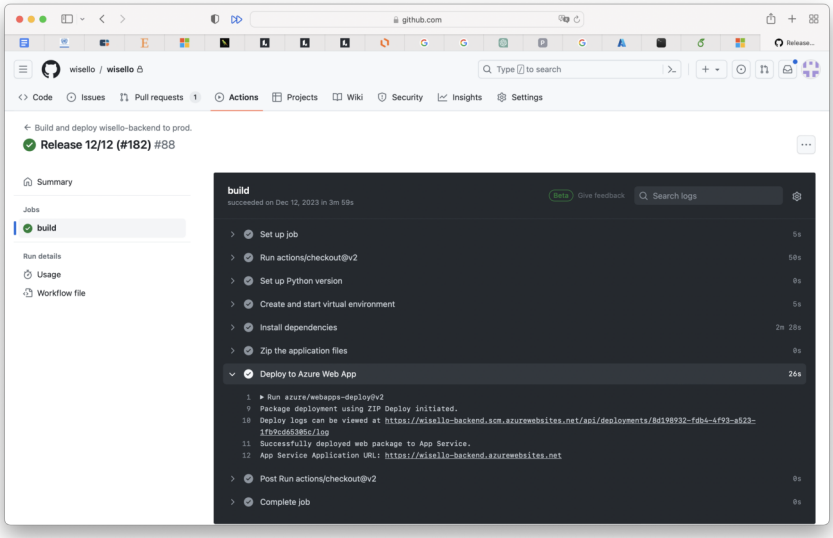


Figura 7.17: Ejemplo de la pipeline de despliegue del wisello-backend

En el caso del `wisello-chat-frontend` y `wisello-web-platform`, utilizamos la pipeline de despliegue que nos provee `vercel` en su integración, por lo que fue más sencillo su configuración, y una de las principales razones por las cuales decidimos utilizar esta tecnología para el frontend del sistema. Esta pipeline nos provee ambientes de despliegue para cada rama publicada en Github. Por lo tanto, se sumaron a los deploys de develop y main, los *feature-staging-deploys* para el frontend de la aplicación, acortando así aún más las diferencias entre desarrollo y producción.

En línea con esto, se hace evidente también el cumplimiento del **RNF-DP-01**, que implica poder desplegar cada subsistema sin impactar u ocasionar *downtime* en los demás. Cada uno de ellos, al tener su propia pipeline de despliegue e infraestructura independiente, no impacta en el resto.

En *wisello*, cada aplicación del sistema posee su propio repositorio en Github (codebase) sobre el que se hace control de versiones, múltiples despliegues (pipeline de despliegue independiente), y donde podemos encontrar todo el código fuente de la aplicación (se hizo una excepción con `bubble-chat-server`, dado que al ser una aplicación de un archivo, se la incluyó dentro de `wisello-backend`, pero conservando su pipeline de despliegue independiente). De esta forma, cada aplicación se encuentra desacoplada de las otras, y puede evolucionar y escalar a su propio ritmo.

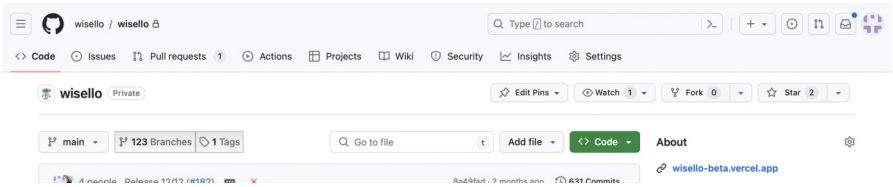


Figura 7.18: Repositorio de Github del backend

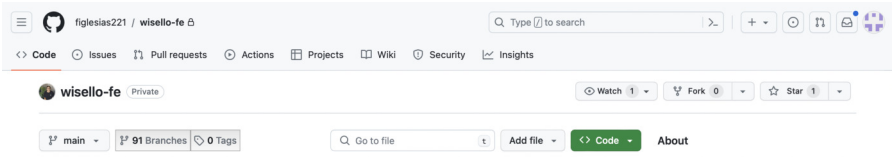


Figura 7.19: Repositorio de Github de chat-frontend

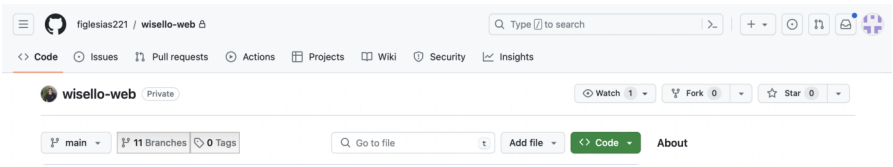


Figura 7.20: Repositorio de Github de web-platform

Manejo de ambientes de producción, desarrollo y pruebas

De acuerdo con el **RNF-PB-03**, que refiere que el sistema debe poder ser desplegado en ambientes de producción, desarrollo y pruebas, para ello utilizamos **variables de entorno**. A través de estas, logramos cambiar de ambientes, compartiendo el mismo código base para todos los despliegues de la aplicación.

En las variables de entorno se sitúa la configuración, es decir, todo lo que varía de un ambiente a otro, y credenciales que no se desea que estén situadas en el código, para que no sean comprometidas. Entre estas, se incluyen: *api keys*, *connection strings* de las bases de datos, puerto en el que escucha el servidor, etc. Para un detalle más profundo de las mismas ver su listado en el anexo 15.4.

De esta forma, el *build* de la aplicación se mantiene independiente al ambiente concreto en el que correrá. Por ende, al ser independiente, luego se pueden hacer múltiples despliegues de este *build* a distintos ambientes, cambiando únicamente las variables de entorno.

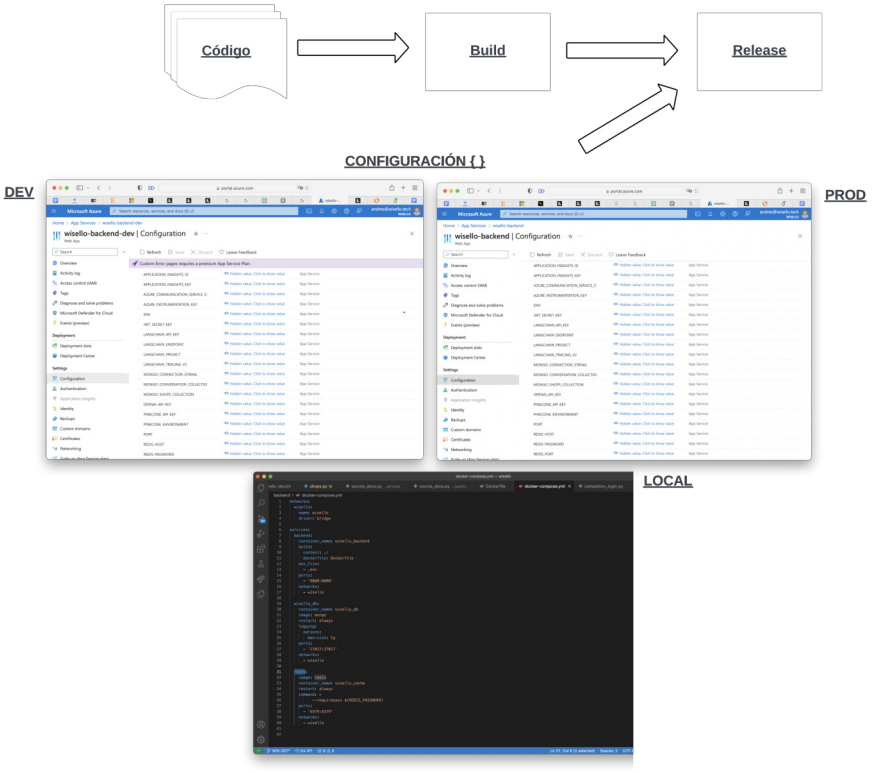


Figura 7.21: Despliegue de *wisello* en distintos ambientes utilizando configuración

Como se ve en el diagrama anterior, localmente las variables de entorno se definen en archivos `.env`, como variables de entorno de cada contenedor en el docker-compose. En el caso de los ambientes de producción y desarrollo, para el backend, estas se definen en la configuración de `App Services` en Azure, o en el caso del frontend, en la configuración del ambiente en `vercel`.

Igualdad entre desarrollo y producción

Con el objetivo de reducir las diferencias entre desarrollo y producción, poder probar los cambios en un entorno idéntico al de producción previo a su *release* en este, y de esta forma, impedir que surjan errores durante la puesta en producción, creamos ambientes *staging* de despliegue para cada subsistema, por donde deben pasar los cambios previo a ser desplegados en producción. Hacemos referencia, por tanto, al **RNF-DP-04**, que explica que *deben implementarse ambientes staging de despliegue iguales al de producción para cada subsistema, donde deben ser probados los cambios previo a su despliegue en producción.*

Al utilizar variables de entorno y Docker, crear los ambientes *staging* fue fácil de lograr, pues implicó cambiar únicamente las variables de entorno, y disponibilizar la infraestructura necesaria para este. Por ende, previo al *merge* de cualquier cambio a producción, primero se corroboraba que este funcionara bien en *staging*, un ambiente idéntico al de producción (excepto por los datos de las conversaciones de los usuarios en producción, que no se sincronizan con *staging* pues entendimos que representaban datos sensibles por lo que no era deseable tenerlas en un ambiente de *staging*). Si era el caso, se procedía con el *release* a producción. Se puede indagar más sobre el flujo de desarrollo en el capítulo 8: Proceso de desarrollo.

Para ambos ambientes (*staging* y producción) se utilizaron las mismas tecnologías (Azure, mongo, redis, etc), alcanzando esta paridad.

Es importante tener en cuenta que el RNF no exige *feature-staging-deploys*, con los que sí contamos para el caso del frontend, pero no para el backend, por el costo que nos implicaría levantar una máquina virtual por cada nueva *branch* en Github. Por ende, para el backend, nos limitamos a probar el deploy de forma local. Sin embargo, por ejemplo, cuando desarrollamos la conexión con el caché redis, al probarlo local funcionaba, pero en la nube no. Esto sucedía porque el cluster de `Azure Cache for Redis` requiere `ssl` para la comunicación, mientras que la imagen local de redis que usábamos no. Esto sucedió porque no había una igualdad entre los ambientes (local, y *staging*/producción). Por tanto, aprendimos la importancia de trabajar con ambientes en igualdad.

Disponibilidad

Tomando en cuenta el **RNF-DP-01**, que exige que el sistema debe estar disponible el **99 %** del tiempo, se tomaron las siguientes decisiones.

A nivel de infraestructura, en primer lugar, para el servicio `Azure App Services` (backend), escogimos el plan `Premium v3 P1V3` que nos proporciona un SLA de **99.95 %** de disponibilidad, y la posibilidad de escalar hasta 30 instancias.

	Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instance)	SLA
>	Dev/Test (For less demanding workloads)						
v	**Production (For most production workloads)**						
	Premium v3 P0V3	195*	1	4	250	30	99.95%
☑	Premium v3 P1V3	195	2	8	250	30	99.95%

Figura 7.22: Hardware escogido para el backend del sistema

A su vez, en lo que refiere al frontend, `vercel` nos provee también un SLA de **99.99 %** de disponibilidad. [36].

En lo que refiere a la disponibilidad de terceros (OpenAI), Azure nos brinda un SLA para su servicio **AzureOpenAI** en el cual se compromete a que, si el *uptime* del servicio es menor al **99.9 %** de los minutos máximos disponibles, se otorgan 10 % de créditos en retribución. A su vez, si este cae por debajo del **99 %**, se nos otorgan 25 %. Se aclara que *"Un minuto se considera tiempo no disponible si más del 0,01 % de las Solicitudes realizadas a la implementación durante ese minuto devuelven un Código de Error."*. Así, se demuestra su voluntad por mantener un un SLA mayor al, por lo menos, 99 % del tiempo para el servicio. [37]

A nivel de disponibilidad del software y sus componentes, se implementó un *endpoint* de `health`, de forma de poder monitorear la salud del sistema, incluyendo:

- El estado de conectividad con los *backing services* principales: base de datos mongo DB, Pinecone y caché redis.
- El estado general del servidor para recibir peticiones.

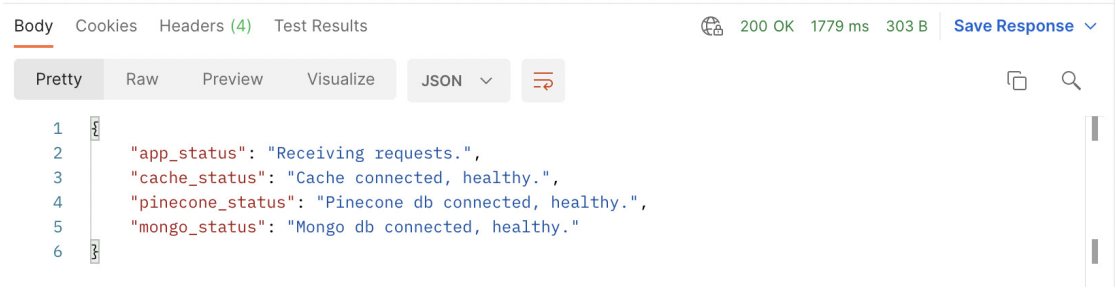


Figura 7.23: Invocación al *endpoint health*

Otro control implementado, del estilo, es el sistema de alertas ante fallas, descrito en profundidad más adelante en 7.1.3.0.0.11.

La idea es que, a futuro, estos controles de *health* evolucionen para asegurar que el software esté siempre disponible, con mecanismos que permitan que este se recupere rápidamente (como, por ejemplo: rollback a una versión anterior, *canary* deployment, o blue/green deployment).

Escalabilidad y redundancia

Ante picos de tráfico y demanda, se configuró la escalabilidad automática del sistema vía `scale out`: escalabilidad horizontal, de acuerdo con lo establecido en el **RNF-ES-01**, que afirma que el sistema debe ser capaz de escalar ante picos de carga. Para ello, se configuró un plan de *scaling* del backend:

- **Scale out method**: método de escalabilidad. En nuestro caso, elegimos `automatic`, lo que significa que la propia plataforma gestiona el *scale up and down* según el tráfico.
- **Always Ready Instances**: número de instancias disponibles y *warm* para la Web App. En nuestro caso, este número fue seteado en `3`.
    - De esta forma, el backend se configura con redundancia de servidores (**RNF-DP-02**).
- **Maximum Burst**: número de instancias adicionales que se pueden agregar si la carga excede lo que las **always ready instances** pueden manejar. En nuestro caso, este número fue seteado en `5`.

El *autoscaling* posiciona un `Load Balancer` delante de todas las instancias que divide el tráfico entre estas. Este `Load Balancer` de Azure hace ping a las instancias activas para verificar que estas estén activas y respondiendo correctamente, haciendo así uso de la táctica de disponibilidad **ping/echo**.

Figura 7.24: *Ping* a instancias activas

APM

Con el objetivo de monitorear la infrestructura y el correcto funcionamiento de los *endpoints* del sistema (**RNF-OB-01**), utilizamos la plataforma **New Relic**, la cual nos provee servicios de *Application performance monitoring* (APM). A través de su agente python, New Relic almacena métricas de infraestructura, *throughput* (requests/min), tiempos de respuesta de cada *endpoint* y tasas de error.

Para ello, creamos dos entidades en New Relic: `wisello-be` y `wisello-be-dev`, diferenciando así el APM de los servidores de producción y *develop*. Luego, en *runtime*, según el valor de la variable de entorno `NEW_RELIC_FILE`, se define con qué entidad se comunica el agente New Relic.

A continuación se presentan capturas de New Relic, para producción, con las métricas mencionadas en **RNF-OB-01**. Por evidencia de APM y logs en *develop* ver anexo 15.5.

***Throughput*, cantidad de fallos, tiempos de respuesta promedio y ADPEX:**

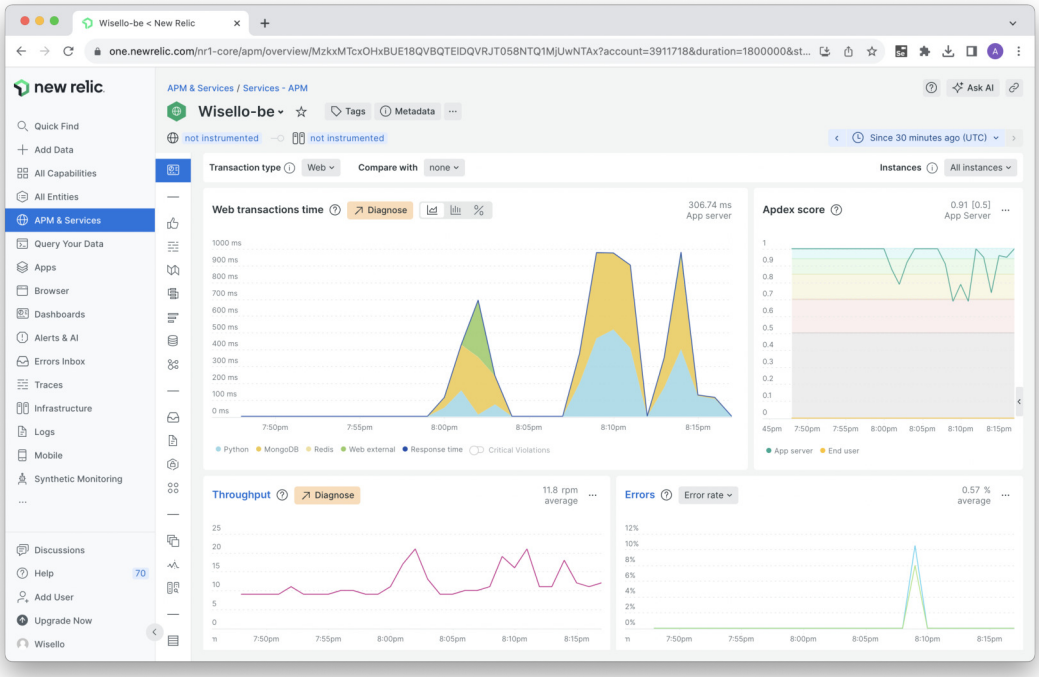


Figura 7.25

Para cada *endpoint* (transacción), podemos ver métricas específicas de este: su tiempo de respuesta promedio, APDEX, porcentaje de error y *throughput*. Vemos el ejemplo de `/completion` debajo:

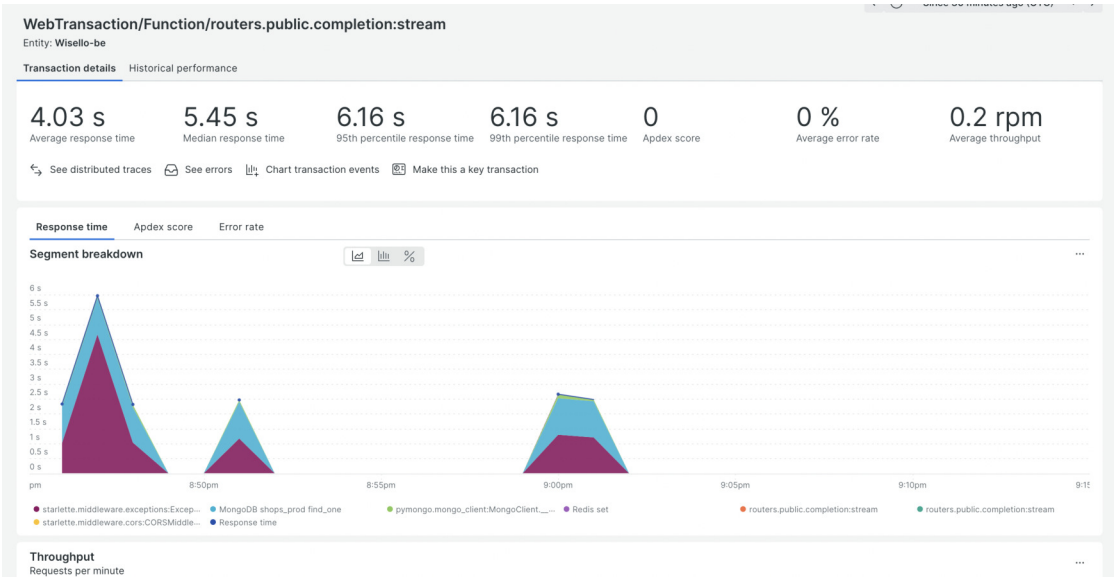


Figura 7.26: *Endpoint completion*

Además, podemos ver una `Breakdown Table` con un desglose promedio de la latencia del *endpoint*, con las funciones, bases de datos y controladores involucrados en él. De esta forma, se puede así identificar cuellos de botella en el flujo y oportunidades de mejora. Incluso, se puede ver el desglose de cada llamada individual, para estudiar peticiones particulares:

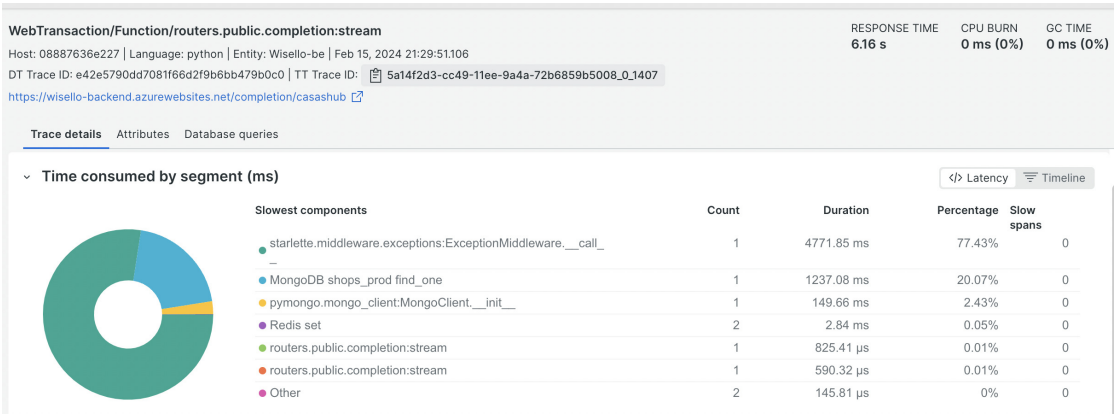


Figura 7.27: Ejemplo de la trazabilidad de una llamada particular de *completion*

Para llevar una observabilidad formal de las métricas de consumo de CPU y memoria de la infraestructura, no utilizamos New Relic, pues Azure nos provee la capacidad de visualizar estas métricas en su portal, para las `Web Apps` y `Function Apps`. Lo vemos a continuación, con la posibilidad de verlas en formato gráfico, agregándolas por `Sum`, `Count`, `Max` o `Min`.

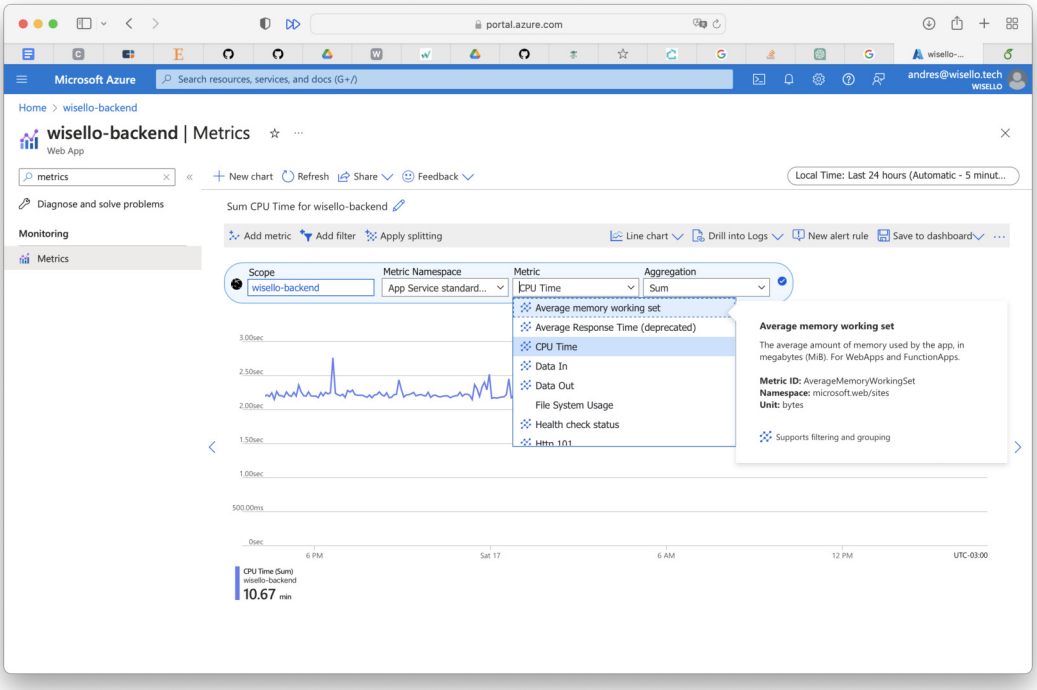


Figura 7.28: Métricas de consumo de CPU y memoria

Almacenamiento de logs

En pos del cumplimiento del **RNF-OB-02**, que indica que se deben centralizar los logs en un único lugar para identificar fallos, separados por ambientes y pudiendo filtrarlos por sus atributos, también utilizamos la herramienta New Relic. Para ello, se utilizaron las mismas dos entidades antes mencionadas (`wisello-be` y `wisello-be-dev`), reteniendo los logs de cada uno de los servidores de producción y *develop* en cada una de estas.

Cada log se guarda con una serie de atributos: `host` desde el que fue generado, `level` del log, `message` del log, `timestamp`, e id, entre otros. Debajo vemos el ejemplo de un log generado al insertar una lista de `source docs` (productos recomendados en la respuesta) en el caché redis. El nivel del log es `INFO` dado que no fue un error.

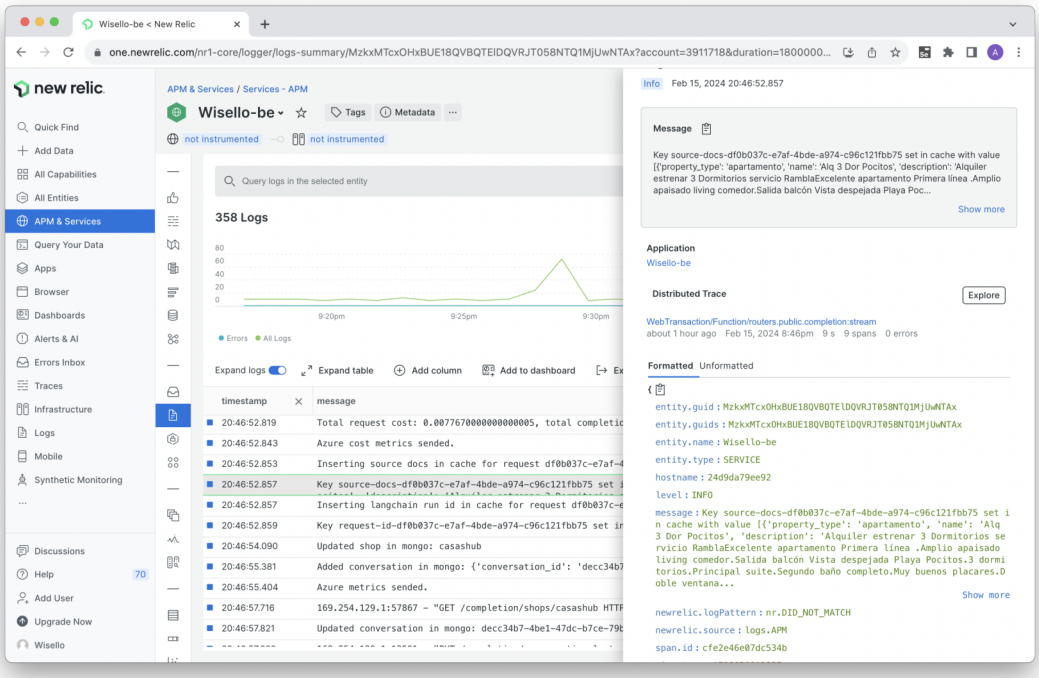


Figura 7.29: Logs en producción

Además, se pueden filtrar los logs (query) a través de cualquiera de sus atributos utilizando el propio lenguaje de consulta `NRQL` que nos provee New Relic, con gran variedad de operadores para realizar consultas complejas.

Sistema de alertas

Se implementó, utilizando New Relic, un sistema de alertas para notificar a todos los integrantes del equipo en caso de fallas del sistema, de acuerdo al **RNF-OB-03**.

La alerta que se creó calcula el porcentaje de error de la aplicación. Si el resultado de la query es mayor a 60 % por un período sostenido de 5 minutos, entonces New Relic, en forma automática, envía el email a los integrantes del grupo alertando de la potencial falla del sistema. La consulta es la siguiente (NRQL):

```
1 SELECT filter(count(newrelic.timeslice.value),
2     where metricTimesliceName = 'Errors/all') /
3     filter(count(newrelic.timeslice.value),
4     where metricTimesliceName IN ('HttpDispatcher', 'OtherTransaction/all')) *
      100 AS 'Error %'
5 FROM Metric
6 FACET appName
```

Código 7.1: Consulta NRQL para implementación de alertas

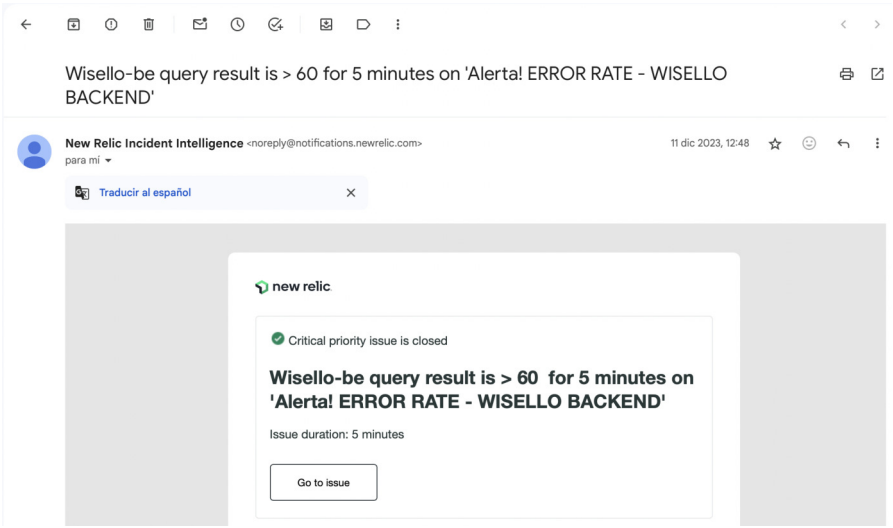


Figura 7.30: Alerta por potencial falla

En el futuro, cuando hayan más e-commerce en producción, este porcentaje de error debería de ser menor y en lugar de enviar un email, optaríamos por un medio de comunicación más rápido como un SMS, WhatsApp o una llamada automatizada.

7.2 Atributos de calidad

A través de las distintas vistas y sus justificaciones de diseño, se ha recorrido los distintos atributos de calidad, indicando las decisiones de diseño que se han tomado, las tecnologías que se han utilizado, las tácticas y patrones que se han aplicado, y demás mecanismos, con el fin de cumplir con los distintos requerimientos no funcionales especificados para el proyecto.

Se han mencionado los atributos de calidad **Performance**, **Desplegabilidad**, **Disponibilidad**, **Seguridad**, **Extensibilidad**, **Observabilidad e Identificación de fallas**, **Escalabilidad** y **Portabilidad**, mencionando cada uno de los requerimientos no funcionales especificados, y explicando su cumplimiento estricto.

Sin embargo, uno de ellos no se ha mencionado aún: la **Usabilidad**, por lo que a continuación procederemos a profundizar en él, y los RNF explicitados.

Testing exploratorio con usuarios reales

Para evaluar el cumplimiento con el **RNF-US-01**, correspondiente al tiempo de aprendizaje del chat y de la plataforma, llevamos a cabo una serie de sesiones de testing exploratorio con usuarios reales, dado que el enfoque, en este caso, está en la usabilidad y experiencia del usuario.

El detalle de estas sesiones se puede ver en el anexo 15.6, en donde se evidencia el cumplimiento con los tiempo de aprendizaje acordados en el **RNF-US-01**:

- Máximo 3 minutos para el chat.
- Máximo 10 minutos para la plataforma de visualización y configuración.

El usuario es partícipe del estado del sistema

Para cumplir con **RNF-US-02** de Usabilidad, que explicita que el sistema debe hacer visible su estado al usuario en todo momento, decidimos utilizar la retroalimentación (el feedback) al usuario. Este método nos permite que el usuario sea consciente del estado del sistema, y sus transiciones tras las acciones que realiza. A continuación evidencia de ello:

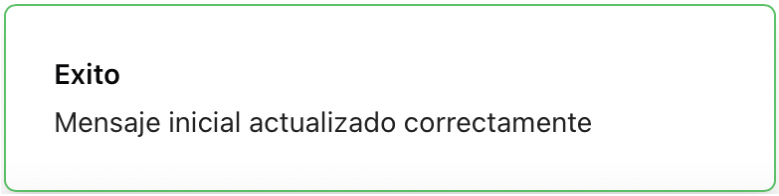


Figura 7.31: Confirmación de mensaje inicial actualizado correctamente

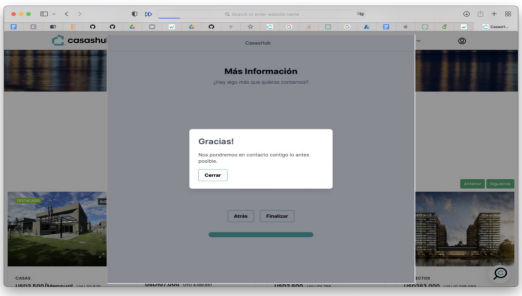


Figura 7.32: Feedback formulario **Buscar por mi**

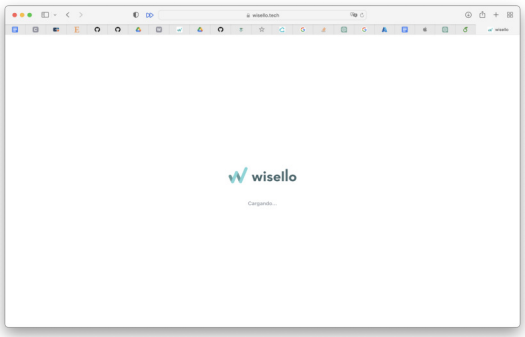


Figura 7.33: Cargando contenido

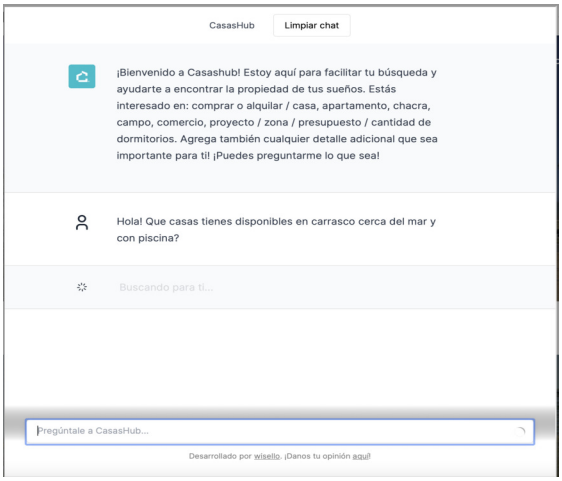


Figura 7.34: Retroalimentación genérica del chat

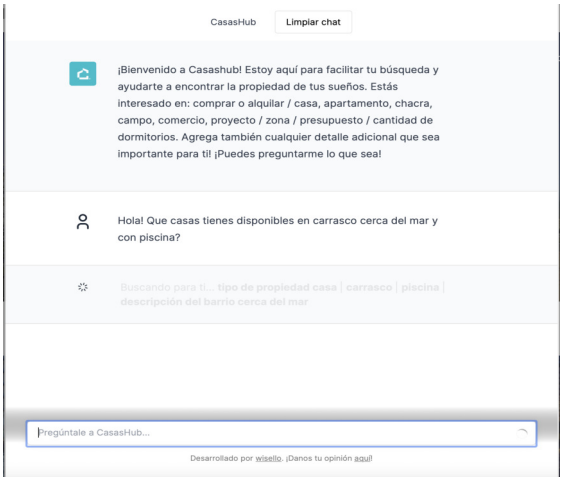


Figura 7.35: Retroalimentación personalizada del chat

Mensajes de error

Continuando con los RNF de Usabilidad, presentaremos el **RNF-US-03** que afirma que *el usuario debe ser presentado con mensajes de error apropiados, entendibles e informativos para cada caso (y en todos ellos) de escenarios erróneos*. Para ello, nos valimos del manejo de errores creado a nivel del backend (ver 7.1.2.0.2.6), de forma de poder interpretar cada error en el frontend y poder devolver al usuario un mensaje de error *apropiado, entendible e informativo para cada caso*. A continuación, algunos de ellos:

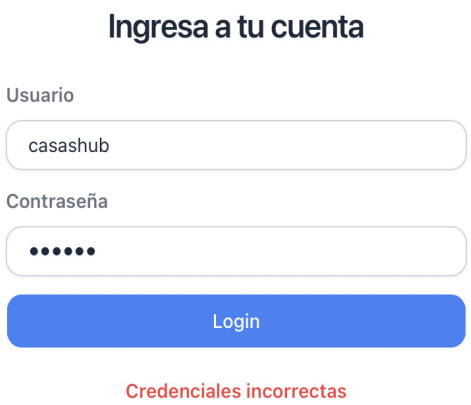


Figura 7.36: Credenciales incorrectas

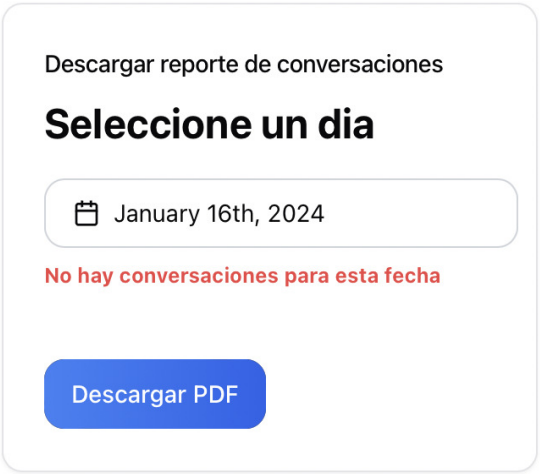


Figura 7.37: Falta de conversaciones para la fecha

Figura 7.38: Teléfono malformado

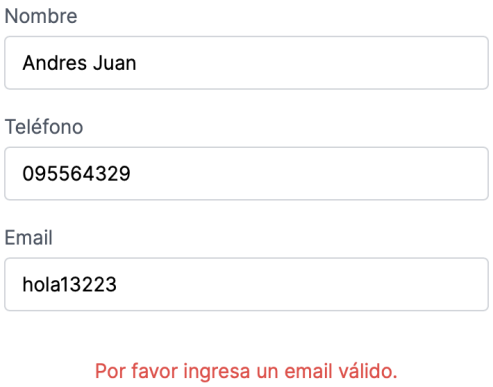


Figura 7.39: E-mail malformado

Responsiveness

Finalmente, de acuerdo con el **RNF-US-04**, para lograr que tanto el chat como la plataforma sean *responsive*, esto se logró mediante la adopción de una estrategia centrada en mejorar la accesibilidad y optimizar la experiencia de usuario a través de una amplia gama de dispositivos. Reconociendo la importancia fundamental de facilitar interacciones efectivas con nuestro sistema, nos propusimos garantizar que los usuarios pudieran navegar e interactuar sin problemas, independientemente del dispositivo que estuvieran utilizando.

Para lograr esto, seleccionamos Tailwind CSS, un avanzado y actual framework de CSS. Tailwind CSS se destaca por su capacidad para simplificar la implementación de diseños responsive a través de sus clases de utilidad. Esta herramienta nos permitió refinar el diseño de nuestra plataforma y del chat, asegurando su adaptabilidad y funcionamiento óptimo en una variedad de tamaños de pantalla, desde los más compactos en dispositivos móviles hasta los más extensos en pantallas de escritorio.

Además, pusimos especial énfasis en la flexibilidad de las imágenes y demás elementos de la interfaz. Optamos por el uso de unidades relativas, como porcentajes y `vw/vh`, en detrimento de las unidades fijas. Esta metodología permitió una adaptación precisa de los elementos al tamaño de la pantalla, preservando la calidad visual y evitando cualquier tipo de distorsión.

Para complementar estas acciones, emprendimos un riguroso proceso de pruebas en un espectro amplio de dispositivos y navegadores, el cual incluyó evaluaciones detalladas en dispositivos móviles, tabletas y computadoras de escritorio. Nuestro objetivo era identificar y corregir proactivamente cualquier problema, asegurando así una experiencia cohesiva y plenamente accesible para todos los usuarios, sin importar su punto de acceso a wisello.

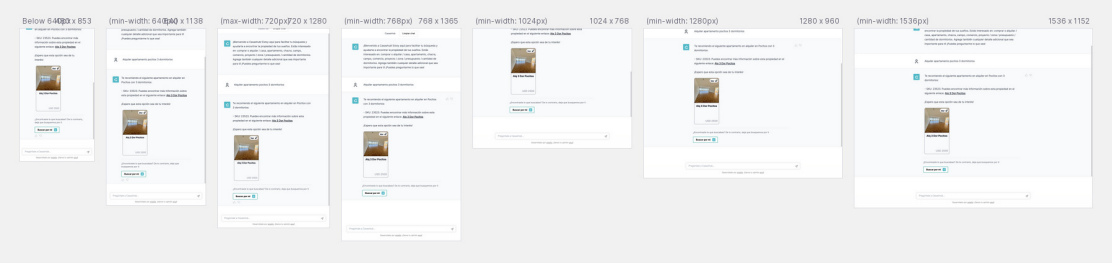


Figura 7.40: Prueba de responsiveness en varios tamaños de pantalla

7.3 Selección de tecnologías

Para la implementación de la arquitectura previamente expuesta utilizamos una cuidadosa selección de tecnologías. A continuación, describimos las tecnologías escogidas para cada componente del sistema, justificando su selección y explicando cómo cada una contribuye al funcionamiento integral de *wisello*. Así como también, presentamos algunas alternativas relevantes a las tecnologías elegidas.

7.3.0.1 Backend

El backend fue desarrollado utilizando **Python**, el cual es un lenguaje de programación de alto nivel, interpretado, conocido por su simplicidad y legibilidad.

Las razones detrás de la elección de Python son:

- **Ecosistema de librerías de Inteligencia Artificial**. Python es el lenguaje de programación más popular en el contexto de Inteligencia Artficial y Ciencia de Datos, dada su gran variedad de librerías. [38]
    - En línea con ello, su **compatibilidad con LangChain**, que fue inicialmente construida únicamente para Python, fue otro factor decisivo en la decisión. Langchain se lanzó en Octubre 2022 como una librería Python, y luego en Febrero 2023 lanzó su versión Typescript. En ese entonces, nosotros estábamos por comenzar el desarrollo (Abril 2023), y la versión Typescript nos parecía muy reciente.
- **Experiencia previa** del equipo utilizando Python en varios proyectos de Inteligencia Artificial, tanto en el ámbito profesional como en el académico.
- **Comunidad pujante, activa y amplia**. Según un reporte publicado por la Universidad de Wisconsin: *"The Python community has grown significantly over the last decade (...) the main driving force 'behind Python's growth is a speedily-expanding community of data science professionals and hobbyists.'"*. [39] Una comunidad que contribuye al ecosistema rico en bibliotecas mencionado previamente, y da soporte al desarrollador.
- **Flexibilidad y eficiencia**, dado el grado de incertidumbre en los inicios del proyecto necesitábamos tener la libertad de realizar modificaciones fácil y rápidamente. La claridad de su sintaxis y su sistema de tipado dinámico permiten mayor rapidez y menos complejidad, promoviendo así una mayor productividad. [40]

A su vez, utilizamos dos frameworks claves. Por un lado para la construcción de la API utilizamos **FastAPI**, un reciente framework web para construir APIs con Python. Las razones de su elección incluyen: alto rendimiento, facilidad de uso y asincronía Nativa. El soporte nativo para código asíncrono permite manejar un gran número de solicitudes simultáneas, lo que es crítico para una aplicación conversacional que puede experimentar picos de tráfico. [41]

Por otro lado, para la integración con OpenAI utilizamos la anteriormente mencionada: **LangChain**, un framework open source que facilita la creación de aplicaciones basadas en LLMs. El principal motivo de su elección fue porque notamos que se posicionaba como la principal alternativa para integrar LLMs en aplicaciones en el momento, y así lo fue. Además, como framework, desde el primer momento notamos que Langchain provee una gran **maleabilidad** al desarrollador para poder integrar, adaptar y agregar código propietario a gusto, lo cual permite personalizar, de alguna forma, el framework *ad hoc*, algo que no sucede con otros frameworks, que suelen ser más rígidos en este sentido. Esto sucede porque fue desarrollado con una serie de patrones (como el patrón `Observer`, por ejemplo, que comentamos en 7.1.2.0.2.1), que así lo permiten.

7.3.0.2 Proveedor de Inteligencia Artificial

Como proveedor de Inteligencia Artificial, decidimos utilizar **AzureOpenAI** (en particular, el modelo `gpt-3.5-turbo`), en contraposición con la API de **OpenAI** o la alternativa *self-hosted* **Llama** (modelo `llama-2`). Esta decisión se basó en los siguientes factores:

- Azure OpenAI no utiliza los datos provistos para entrenar los modelos, y su performance es superior a la de la API de OpenAI.
- Los modelos GPT de OpenAI proveen la capacidad de utilizar *OpenAI functions*, mientras que Llama no.
- Comparación de *Benchmarks*; en los que GPT-3 supera el desempeño de Llama-2. Por ejemplo, un estudio realizado para e-commerce específicamente llamado **LLMs in e-commerce: A comparative analysis of GPT and LLaMA models in product review evaluation** reveló lo siguiente (se observa los `Base Models` dado que nosotros no hacemos *fine tuning* de los modelos):

- *"regarding the performance of the base models gpt-3.5-turbo-1106 and llama-2-70b-chat (...) it was observed that GPT-3.5 is significantly more accurate than the LLaMA-2 model."* [42]

- La comunidad de desarrolladores, y el soporte otorgado entre estos a través de `OpenAI Forum`. Debajo presentamos un ejemplo de un bug de la API de OpenAI reportado por *wisello*, para el cual recibimos 42 respuestas y 3400 visitas.

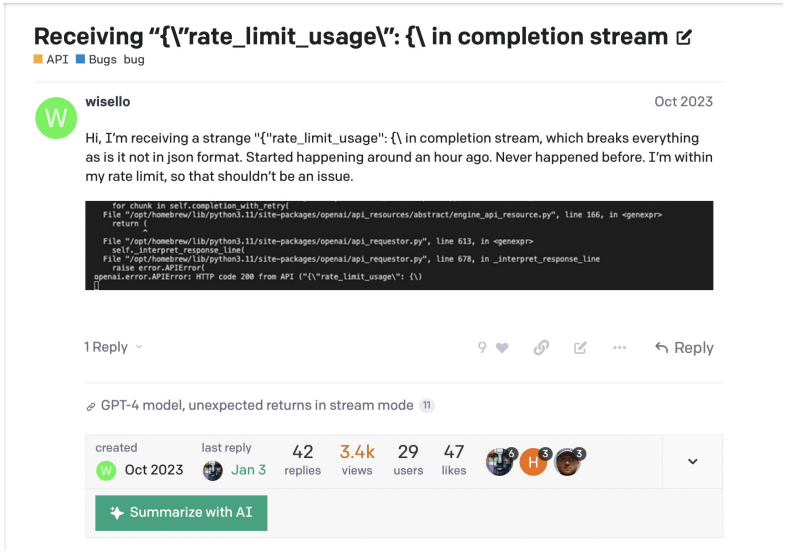


Figura 7.41: Bug de OpenAI reportado por *wisello*

7.3.0.3 Almacenamiento de datos

La elección de las tecnologías para el almacenamiento de datos fue crítica para garantizar la velocidad, consistencia, fiabilidad del acceso a los datos, y en algunos casos, ciertos objetivos funcionales de la aplicación.

Por ejemplo, para almacenar los **catálogos de las tiendas de e-commerce**, utilizamos **bases de datos vectoriales**. La principal razón de esta elección fue la capacidad que estas tienen para realizar **búsquedas semánticas**. Así, podríamos realizar búsquedas del catálogo utilizando el *input* del usuario en el chat: conversacional, desestructurado, sin formatos. De allí, la elección de una tecnología de almacenamiento de datos en base a un objetivo funcional.

Estas bases de datos permiten una búsqueda eficiente y relevante de productos al almacenar la información como vectores en un espacio multidimensional, lo que

trasciende las simples coincidencias de texto y habilita sugerencias más pertinentes para los usuarios. Su optimización para manejar grandes volúmenes de datos asegura un rendimiento robusto y tiempos de respuesta rápidos, incluso a medida que crece el catálogo de productos. Además, la estructura vectorial simplifica la inclusión de nuevos productos y la actualización de los catálogos, manteniendo su relevancia y competitividad en el dinámico mercado e-commerce. [43]

En particular, elegimos **Pinecone** porque ofrece ventajas significativas en términos de escalabilidad, rendimiento y facilidad de integración con el ecosistema de herramientas y servicios ya utilizados. [44]

Para el almacenamiento de **conversaciones** y **tiendas**, optamos por **MongoDB**, por la flexibilidad de esquema que esta nos provee, permitiendo su evolución sin afectar su funcionamiento, algo que es muy importante especialmente en un producto de este tipo, donde los los requerimientos y el contexto son muy cambiantes. Incluso, MongoDB se caracteriza por su escalabilidad horizontal y alta disponibilidad, lo cual se alinea con los RNF especificados [45]. Además, el equipo trabajó previamente con MongoDB, por lo que se sentía cómodo con la tecnología.

También, implementamos un cache utilizando **Redis** para almacenar información que es frecuentemente utilizada y/o se desea acceder rápidamente. En Redis, los datos residen en memoria, ahorrándose el acceso a disco, razón por la cual presenta una baja latencia y un alto *throughput*, logrando así la performance deseada para el acceso a esta información. Además, al igual que MongoDB, el esquema es flexible, permitiendo evolución, presenta alta disponibilidad y escalabilidad horizontal [46]. Nuevamente, el equipo tenía experiencia con esta base de datos.

7.3.0.4 Frontend

El frontend es una aplicación web, para la cual utilizamos **NextJS**, un framework avanzado de ReactJS, por varias razones estratégicas. Primero, NextJS se destaca por su eficiencia, ofreciendo carga rápida de páginas mediante técnicas como el renderizado del lado del servidor y la generación estática. Su amplio ecosistema de componentes reutilizables acelera el desarrollo, permitiendo un enfoque modular y coherente. [47]

Finalmente, la experiencia previa del equipo con NextJS aseguró una curva de aprendizaje mínima, optimizando el flujo de trabajo y facilitando la implementación de características complejas de manera eficiente.

Además, el despliegue del frontend en Vercel se destaca por su simplicidad y eficacia, proporcionando un entorno optimizado específicamente para aplicaciones NextJS. Sobre esto profundizaremos más en la siguiente sección.

7.3.0.5 Plataformas y Despliegue

Para el despliegue, optamos por una infraestructura *cloud-based*, aprovechando la flexibilidad, seguridad, escalabilidad y robustez que ofrece la misma.

Seleccionamos **Microsoft Azure** como nuestro cloud provider para el backend debido a su amplia gama de servicios, fiabilidad probada y soporte global. Azure nos permite escalar dinámicamente los recursos según la demanda, asegurando así un rendimiento óptimo y una alta disponibilidad del servicio [48]. Además, nos fueron otorgados los créditos mencionados en la sección 4.2.6 por su programa **Microsoft for Startups Founders Hub**.

Se tuvo en cuenta la opción de utilizar Amazon Web Services (AWS), dado que el equipo dominaba más esta nube, pero esta alternativa fue descartada al obtener los créditos mencionados.

Para el despliegue continuo, utilizamos **GitHub Actions**. Esta decisión se basó en la fácilidad de integración que ha creado Azure con Github Actions, a través de las `Actions` creadas para realizar los deploys a sus servicios desde la *pipeline* de Github Actions.

En cuanto al frontend, optamos por **Vercel** para su despliegue, una decisión estratégica que se alinea perfectamente con nuestra elección de NextJS para el desarrollo. Como mencionamos anteriormente, Vercel, siendo el creador de NextJS, proporciona una plataforma de hosting altamente optimizada para aplicaciones construidas con este framework, lo que resulta en despliegues instantáneos y configuraciones mínimas. La integración directa con el repositorio de código fuente permite actualizaciones y despliegues automáticos con cada *push*, facilitando un ciclo de desarrollo ágil y eficiente. [49]

7.3.0.6 APM y Logs

Para el almacenamiento de logs y servicios de APM, utilizamos la herramienta **New Relic**. Esta fue escogida por su fácil integración con la aplicación, gran variedad de servicios en su plataforma SaaS, y experiencia previa del equipo.

7.3.0.7 Monitoreo de LLMs

Nuestro enfoque de monitoreo continuo, cuyo fin es mejorar el rendimiento general del sistema, se basó principalmente en la recopilación de feedback de los usuarios sobre la calidad de las respuestas generadas, para luego poder afinar y perfeccionar la solución. Para ello, elegimos integrar **LangSmith** a la aplicación.

LangSmith se especializa en el monitoreo, evaluación y testing de aplicaciones basadas en LLMs, ofreciendo una interfaz para supervisar cada respuesta de *wisello*: la respuesta dada al usuario, *time to first token*, feedback del usuario, costo, cantidad de tokens, entre otros.

La razón por la cual la elegimos, por tanto, fue porque necesitábamos una herramienta como Langsmith que nos permita combinar ambas características: el aseguramiento de la calidad (a través de la recopilación del feedback de usuarios), con el monitoreo en real-time.

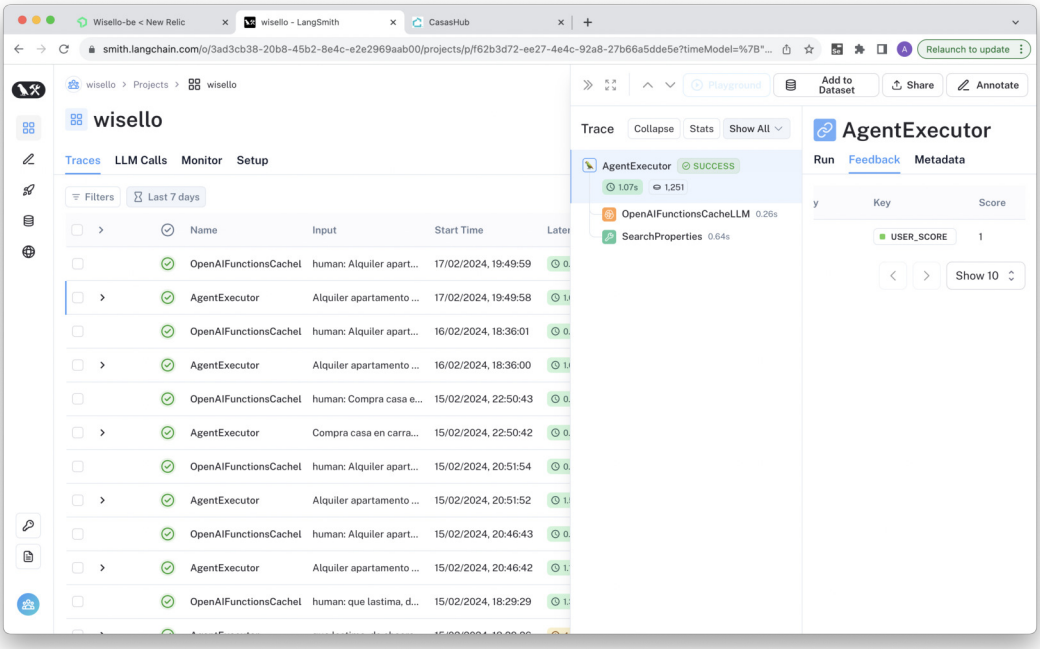


Figura 7.42: Dashboard de *wisello* en LangSmith

8 Proceso de desarrollo

El proceso de desarrollo del proyecto se ejecutó en el marco de diferentes prácticas, y utilizando diversas herramientas, en las cuales se ahondará a lo largo de esta sección. A su vez, detallaremos el proceso en su perspectiva general, cómo las herramientas utilizadas nos ayudaron a alcanzar los objetivos propuestos para este, y cuáles fueron los aprendizajes del equipo en torno al proceso adoptado.

8.0.1 Foco del proceso

El principal objetivo del proceso de desarrollo fue lograr construir un producto de *calidad*, que cumpla con sus requerimientos especificados, a través de un proceso de desarrollo de *calidad*, *sustentable* en el tiempo y conforme a las prácticas de la ingeniería de software.

Dado que el marco de gestión utilizado para el proyecto (**Scrum**, en el cual se profundiza más adelante en 9.1) no entra en detalle en aspectos de desarrollo, optamos por definirlo nosotros mismos, sin utilizar marcos formales para ello.

Se ideó el proceso de desarrollo de tal forma que este se pueda sostener en el tiempo más allá de su transcurso académico, por tratarse de un emprendimiento. A su vez, este fue diseñado también para permitir la transparencia del desarrollo a todos los integrantes del equipo, y promover la automatización de releases a producción, para hacer este proceso simple y agregar valor a los usuarios de manera frecuente.

8.0.2 Prácticas

Entre las prácticas que compusieron el proceso de desarrollo se encuentran:

- Control de versiones con **Git**.
- **Estándares aplicados** según los lineamientos de cada lenguaje: por más detalle referirse al anexo de cumplimiento con estándares de código 15.7.
- **Clean Code**: por más detalle referirse al anexo de Clean Code 15.8.
- **Code Reviews**: referirse a la sección de revisiones de código 8.0.6.
- **Integración continua y despliegue continuo (CI/CD)**: referirse tanto a la sección de *pipelines* de despliegue 8.0.6 o a la sección de automatización del despliegue 7.1.3.0.0.5.
- **Automatización de los tests**: referirse a la seccion de pruebas automáticas 8.0.5.
- **Virtualización** (más específicamente, *containerización*): referirse a la sección de Docker 7.1.3.0.0.4.
    - En esta sección, se explica el rol del archivo **Docker-compose**, a través del cual un desarrollador que se clone el proyecto puede instanciar, de forma local, todo el ecosistema del backend utilizando un solo comando.
    - De esta forma, todos los desarrolladores manejamos un mismo entorno de desarrollo.
- **Gestión de dependencias**.
    - En el caso de Typescript (Frontend), para gestionar las dependencias se utilizó el archivo `package.json`, que genera `npm`, donde se definen todas las dependencias que requiere la aplicación.
    - En el caso de Python, se utilizó el archivo `requirements.txt`, definiendo allí todas las dependencias necesarias.
- **Gestión de configuraciones y entornos**: referirse a la sección de configuración 7.1.3.0.0.6 y anexo variables de entorno 15.4.
- **Desarrollo guiado por capas**: referirse a la sección de vista de layers 7.1.1.2.

8.0.3 Herramientas

Con el objetivo de poner en marcha el proceso de desarrollo mencionado, utilizamos diversas herramientas que dieron soporte al mismo. Las principales se presentan a continuación:

Github

Github fue empleado como plataforma para la gestión de los repositorios Git, así como también para ejecutar las *pipelines* de integración y despliegue continuo a través de su servicio Github Actions.

Jira

Esta herramienta predominó en el proceso de desarrollo, dado que permitió la ejecución, asignación, transparencia, orden y puesta a punto del mismo. Incluso, Jira nos permitió integrar la planificación del **desarrollo del proyecto**, con el **marco de gestión** del proyecto desde el punto de vista de *Scrum*.

A nivel de **proceso de desarrollo**, Jira nos permitió la creación de un tablero con los siguientes estados: `TO DO`, `IN PROGRESS`, `REVIEW` (en code review), `STAGING` (en ambiente *develop*) y `DONE` (en producción).

En el tablero, se encuentran las historias de usuario correspondientes al *sprint* actual, con sus *tasks* a realizar. Esto permite al desarrollador ver qué tareas tiene asignadas, qué tareas restan realizar en el *sprint*, qué tareas están realizando el resto de sus compañeros, cuáles están para realizar code review, y demás. De esta forma, nos aseguramos un proceso de desarrollo transparente, ordenado, y centralizado en un único lugar.

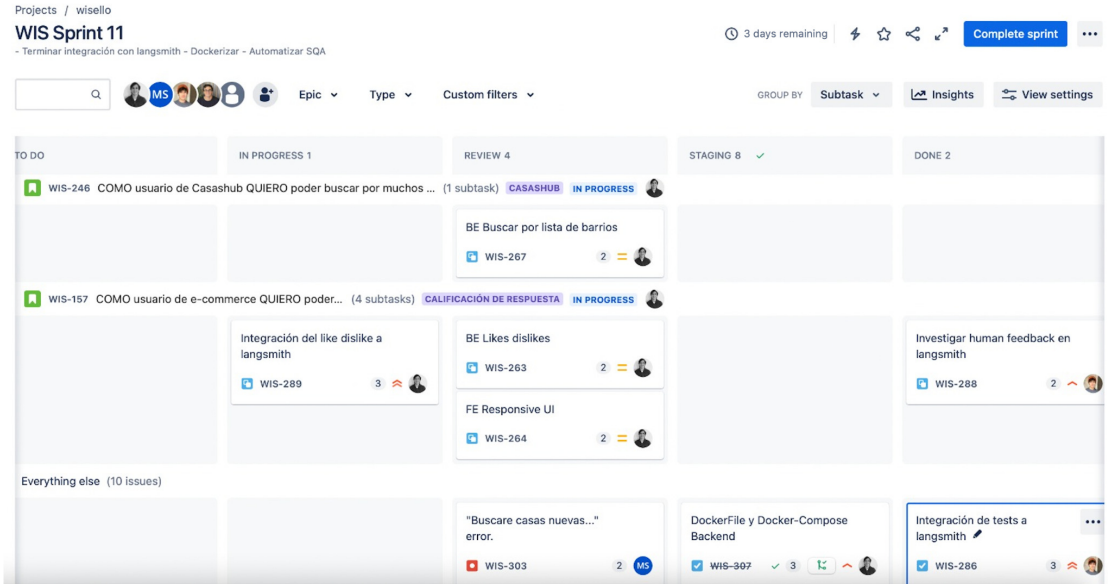


Figura 8.1: Tablero de *user stories*, *bugs* y *tasks*

Además, el tablero está vinculado a Github, lo que permite que, al trabajar en una **issue** específica del tablero, se pueda crear una rama en Github con el mismo nombre adoptado por esa tarea en Jira. De esta forma, se facilita el seguimiento del trabajo, pues se puede acceder desde Jira a las **Pull Requests** asociadas en Github:

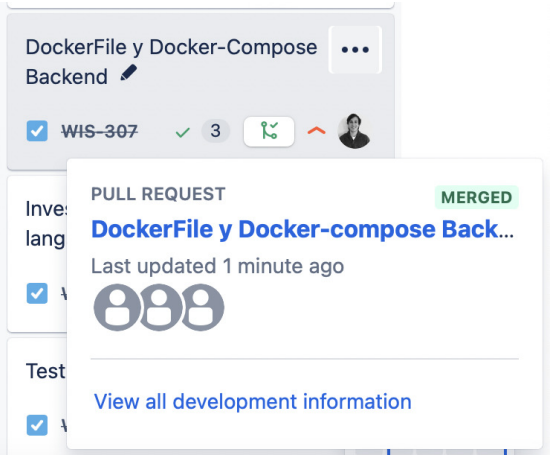


Figura 8.2: *Task* con su *pull request* asociada en Github

Slack

Se eligió Slack como herramienta principal para la comunicación interna del equipo. Esta elección se fundamentó en su capacidad para crear múltiples canales, y así abordar temas específicos en cada uno de ellos, su potente motor de búsqueda que facilita el acceso a conversaciones anteriores de manera sencilla, y el hecho de que los integrantes del equipo ya lo utilizaban para otras cuestiones (laborales, académicas, etc), por lo cual su adopción resultaba práctica.

Otro diferencial de Slack es su capacidad para integrarse con aplicaciones de terceros. En nuestro caso, lo integramos con con Github para notificaciones sobre la creación de Pull Requests y los estados de las pipelines de CI/CD:

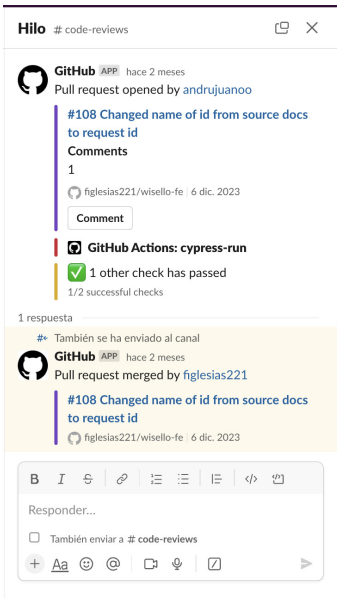


Figura 8.3: Integración Github-Slack

Postman

Se utilizó Postman para el testing local, en tiempo de desarrollo, de los endpoints de la API REST del backend.

MongoDB Compass

Se utilizó MongoDB Compass para la visualización local de la base de datos MongoDB, facilitando la inspección y manipulación de los datos almacenados.

Redis Insight

Se utilizó Redis Insight para la visualización local del caché Redis, facilitando la inspección y manipulación de los datos almacenados.

8.0.4 Definition of Done

En el contexto de proceso de desarrollo, el término *Definition of Done* refiere a un acuerdo compartido por el equipo sobre los criterios que debe cumplir una tarea de desarrollo para considerarse terminada. En nuestro caso, estos fueron los siguientes:

- Se realizó code review por al menos un miembro del equipo, y se resolvieron todos los comentarios.
- Se hizo el despliegue en ambiente de desarrollo (*develop*).
- Satisfacción de los criterios de aceptación establecidos.
- Cumplimiento con estándares y Clean Code.
- Se implementaron pruebas unitarias, si aplicaba.

8.0.5 Pruebas automáticas

Durante la etapa de desarrollo del proyecto, se implementaron tests automáticos para el backend (que fueron automatizados utilizando Github Actions), con el objetivo de identificar cambios que introducen errores en el sistema.

En paralelo, se ejecutaron pruebas de integración para el frontend utilizando `Cypress`, una herramienta avanzada para este propósito. Estas fueron automatizadas dentro de la pipeline, garantizando así su cumplimiento continuo.

Para más detalles sobre estos tests, referirse al capítulo 12 de Aseguramiento de la Calidad, concretamente a la sección automatización del testing: 12.2.

8.0.6 Code Reviews

Para cumplir con nuestra *Definition of Done*, estar al tanto del desarrollo del resto del equipo, prevenir errores, controlar estandáres de código y Clean Code, así como ayudarnos entre nosotros con trabas en el desarrollo de funcionalidades, llevamos a cabo revisiones de código mediante la apertura de Pull Requests en Github.

Estas permiten asegurar la calidad del código, mantener al equipo en sincronía y detectar áreas de mejora. Para cada nueva funcionalidad (*feature-branch*) o corrección de errores, se creaba una rama específica, como se comentó anteriormente, con el nombre identificador de la issue de JIRA (por ejemplo, "WIS-94"), y una vez que se creía oportuno, se abría la Pull Request a *develop*, para que el resto del equipo pueda realizar sus revisiones de código.

La creación de la Pull Request se comunicaba a través del canal de Slack, facilitando así que todos los integrantes estuvieran al tanto del progreso y del estado de las solicitudes pendientes.

A continuación se muestra un ejemplo de parte de una Code Review en particular.

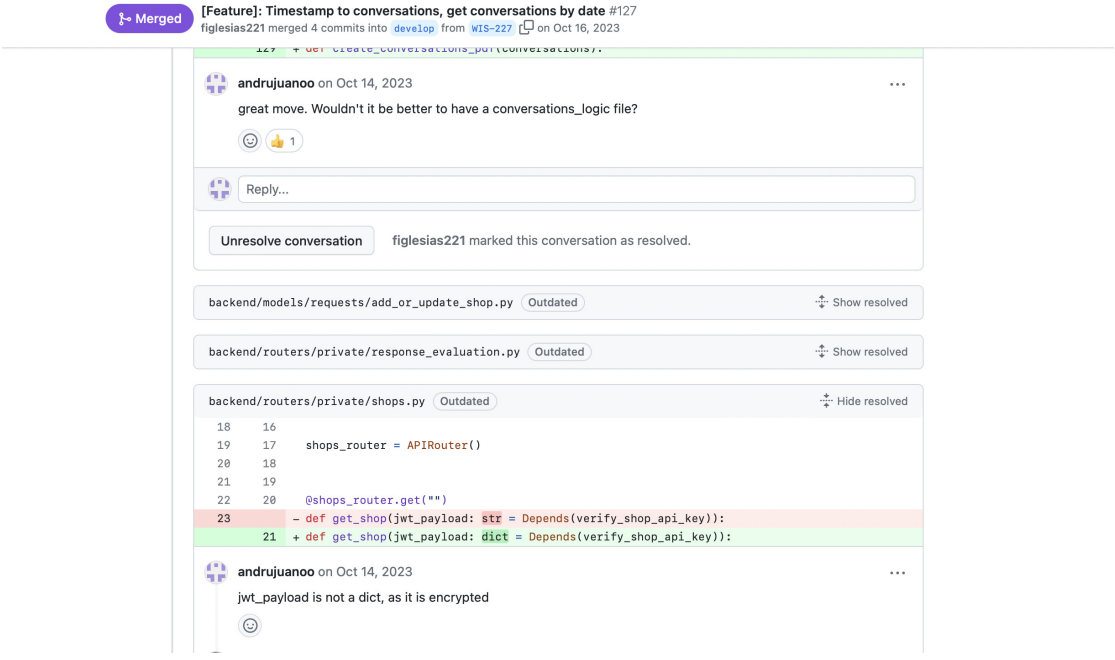


Figura 8.4: Ejemplo de Code Review

Cada incumplimiento o error, por más mínimo, se incluía con un comentario *amable* sobre la Pull Request. Luego, el creador de la misma resolvía todos los comentarios incluidos, y volvía a pedir una revisión final. De esa forma, si el/los revisores consideraban que los comentarios resueltos eran satisfactorios, y la Pull Request estaba pronta para el *merge*, aprobaban la solicitud, para que el creador de la misma hiciera el *merge* (siempre se intentaba que sea este quien haga el *merge*, aunque a veces, por ciertas circunstancias, esto no se daba). En febrero del 2024, se contabilizaron 303 Pull Requests, entre todos los repositorios.

Pipelines CI/CD

A continuación se presentan los flujos de las *pipelines* de CI/CD de *wisello*:

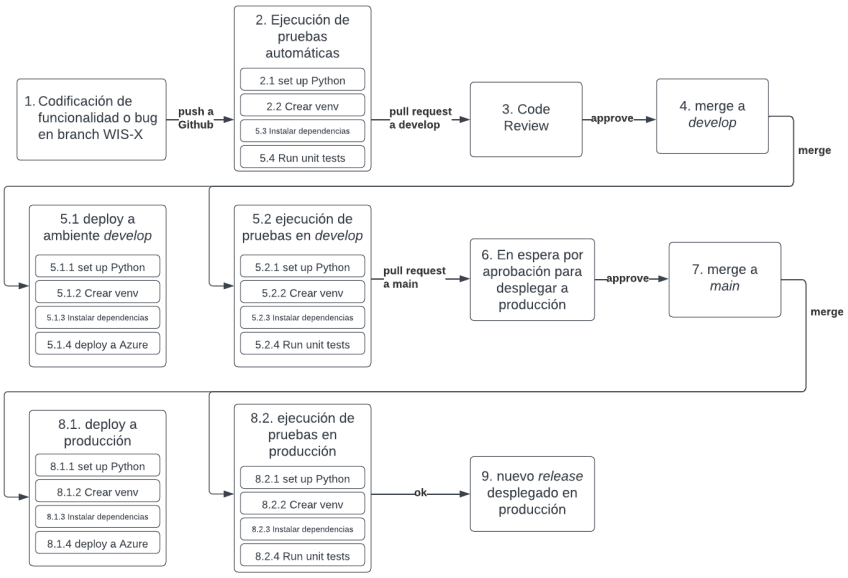


Figura 8.5: Etapas de la pipeline CI/CD del backend

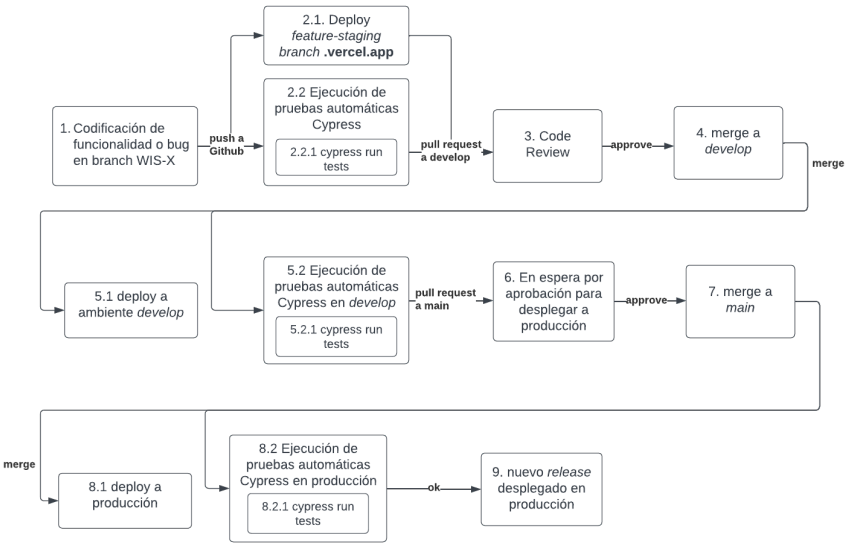


Figura 8.6: Etapas de la pipeline CI/CD del frontend

8.0.7 Aprendizajes del proceso de desarrollo

Durante el desarrollo, aprendimos e identificamos áreas clave para mejorar la eficiencia y la efectividad del proceso. Luego de varias retrospectivas, comprendimos la importancia de agilizar las code reviews para no retrasar el despliegue a producción. Para ello, acordamos que el dueño de la Pull Request estaría encargado de *estar detrás* de sus compañeros, insistiendo, para que revisen su código. A su vez, frente al desafío de Pull Requests demasiado extensas, que complican el proceso de revisión y pueden llevar a problemas de integración, decidimos refinar aún más nuestra división de tasks. Finalmente, reconocimos también la importancia de mantener siempre la concordancia entre el tablero y la realidad del desarrollo, para que las métricas de gestión del equipo sean fieles. Por ende, hicimos hincapié en ello.

Al abordar estas áreas, logramos optimizar y mejorar nuestro proceso de desarrollo.

9 Gestión del proyecto

9.1 Adaptación de Scrum

La elección de un marco de trabajo ágil nos permitió desarrollar un proyecto complejo en un contexto incierto. La posibilidad de crear un producto de forma incremental y poder introducir cambios ágilmente, nos permitieron desarrollar un producto de calidad ajustado a las necesidades cambiantes del mercado.

Como mencionamos en el Capítulo 2, el marco de trabajo utilizado fue una adaptación de Scrum. Consideramos que modificar algunas de las pautas que establece Scrum fue muy beneficioso para la gestión de nuestro proyecto, ya que fuimos adaptando el mismo para que se adecúe al funcionamiento del equipo.

A continuación describimos la implementación del marco de trabajo seleccionado y construido.

9.1.1 Roles

Si bien adoptamos los roles establecidos por Scrum, fuimos flexibles en las restricciones que la metodología impone. Al ser un proyecto final para la carrera de Ingeniería en Sistemas, creíamos que era crucial que todos los integrantes participaran del desarrollo del producto. Esto contradice los establecido por Scrum, ya que tanto el Scrum Master como el Product Owner no deben de participar del desarrollo. Sin embargo, consideramos que asignar los roles de esta manera era lo más adecuado dado el contexto del proyecto.

Rol	Encargado
Scrum Master	Andrés Juan
Product Owner	Mateo Sciarra
Desarrollador	Marcelo Pérez, Federico Iglesias, Andrés Juan y Mateo Sciarra

Tabla 9.1: Roles de Scrum

9.1.1.1 Scrum master

Andrés fue el encargado de facilitar, dirigir y planificar las reuniones y eventos de Scrum (Plannings, Daily Meetings, Reviews y Retrospectives).

Además, tuvo la responsabilidad de velar y trabajar por la mejora continua del proceso del equipo, alentando a la retroalimentación y a la reflexión en las Retrospectivas, en forma cualitativa (feedback) y cuantitatitva (métricas de gestión).

Como todo Scrum Master, garantizó la adhesion a las guías de Scrum. Incluso, en este proyecto en particular, como se dio la adaptación de roles comentada y Andrés también era desarrollador, cumplió como resolutor de impedimentos, abordando activamente los obstáculos que impedían el progreso del equipo.

9.1.1.2 Product Owner

Mateo, por su rol de Product Owner, tenía que asegurarse que el producto final cumpla con las expectativas de los usuarios y stakeholders. Sus tareas incluyeron:

- Interacción con stakeholders (considerando que es un Product Owner **interno**).
- Gestionar el backlog, en base a la interacción con el equipo, clientes y distintos interesados.
- Priorizar el backlog, basándose en el valor agregado.
- Comunicar los cambios efectuados a nivel del backlog, con los motivos detrás de ellos.
- Comunicar la visión del producto al equipo.
- En la Review, exponer cada funcionalidad nueva y dar feedback del cumplimiento de los criterios de aceptación.

9.1.2 Backlog

Durante el desarrollo del proyecto, se mantuvo un backlog con las user stories especificadas para el proyecto desde una visión del usuario (formato **COMO/-QUIERO/PARA**), agrupadas por épicas, actualizadas y priorizadas, sobre el cual se hacía *backlog refinement* periódicamente.

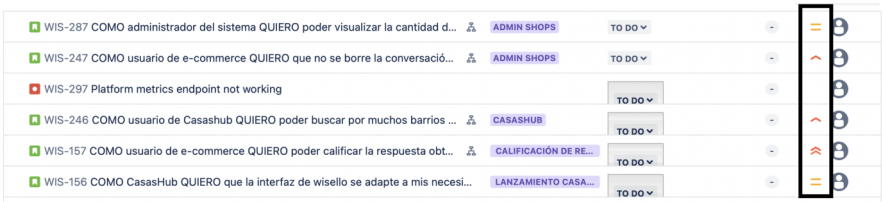


Figura 9.1: Ejemplo del estado del backlog

En el anexo 15.9 se especifican todas las user stories, con sus épicas, que fueron completadas para el proyecto.

Priorización de historias

La priorización de historias estuvo a cargo del Product Owner, y se basó en la siguiente escala: `Highest`, `High`, `Medium`, `Low` y `Lowest` (ver el rectángulo resaltado en la figura 9.1). Para ello, se definieron los siguientes criterios **¿Cómo priorizar?**:

- Fechas de entrega próximas con clientes
- Riesgos inherentes/predecibles en la franja de tiempo
- Si es un bug o no
  - Número/tipo de personas al que afecta
  - Impacto en el negocio
- Impacto en el negocio
- Retroalimentacion del cliente

Según todos estos puntos, se priorizaron las historias de usuario, para definir luego las que serían parte del próximo Sprint (las de mayor prioridad).

9.1.3 Sprints

La aplicación de este marco metodológico se comenzó a implementar una vez realizada la validación del problema y de la solución. Ya habiendo realizado todo el trabajo de investigación y validación presentado en los capítulos 3 y 4, comenzamos a trabajar en Sprints para comenzar con el desarrollo del producto.

El largo definido para los Sprints fue de **dos semanas**: un número discutido frente a la alternativa de hacerlos de una semana. Al estar en permanente contacto con clientes y expertos del dominio, fuimos adquiriendo información y recibiendo pedidos semana a semana, haciendo que el contexto y los requerimientos fueran muy cambiantes. Por ende, fijar un *scope* para el Sprint fijo de dos semanas se volvía muy difícil, teniendo en cuenta que la prioridad era mantenernos ágiles, y no queríamos esperar hasta el próximo Sprint para desarrollar la User Story. Sin embargo, tener Sprints más cortos de una semana significaba perder demasiado tiempo en ceremonias y gestión, con el cual no contábamos.

Por tanto, decidimos tener iteraciones de 14 días, y permitir la flexibilidad de incorporar nuevas funcionalidades al Sprint comenzado de forma ágil. Hasta cierto punto, esto era un *arma de doble filo* porque significaba tener mucho cuidado durante la planificación, dado que debíamos subestimar la capacidad del Sprint, para dejar espacio para las User Stories que estimaríamos se agregarían luego. Vemos un ejemplo de la evolución del *scope* del Sprint para el `Sprint 6`:

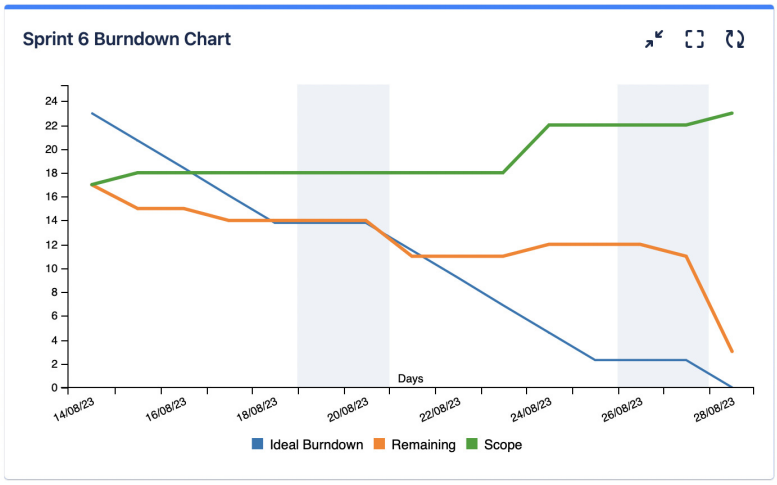


Figura 9.2: Sprint 6 Burndown Chart

Vemos cómo el *scope* del Sprint fue en incremento a medida este fue avanzando, tal como se comentó en esta sección.

9.1.4 Eventos

9.1.4.1 Planning

Este evento se llevó a cabo los Lunes, el primer día del Sprint, con una duración aproximada de dos horas.

En primer lugar, típicamente con Mateo (PO) al frente, se realizaba un ***Backlog Refinement*** para revisar el estado del backlog previo a realizar la Planning. Aquí se re-priorizaba, eliminaba y revisaban las historias de usuario, bugs o tareas del backlog actual.

Una vez que backlog se había refinado, el siguiente paso era definir el *sprint goal*. Como recomendación de la Lic. Jimena Saavedra, Agile Coach, los *sprint goals* fueron basados en outcomes medibles:

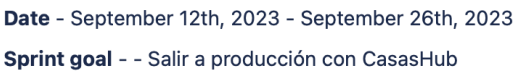


Figura 9.3: *Sprint Goal* Sprint 8

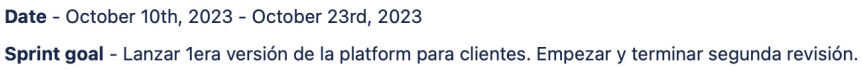


Figura 9.4: *Sprint Goal* Sprint 10

Luego, en base a este objetivo *sprint goal*, las historias de usuario del backlog que mayor prioridad tenían, y a la capacidad estimada del equipo, se seleccionaban aquellas que iban a formar parte del *scope* del Sprint que comenzaba.

Estimación de tareas

La tarea, al ser la unidad de trabajo más pequeña, fue la estimada. Las historias de usuario se desglosan en tareas, que luego se estiman de forma independiente.

La estimación de tareas se llevó a cabo utilizando la medida de **Story Points**, la cual nos permite independizarnos de los ritmos de cada integrante del equipo (en

contraposicion con las **horas ideales**), y llegar a un acuerdo grupal, autónomo, del tamaño de una historia de usuario. La misma tuvo una curva de aprendizaje, pero el equipo notó cómo, a medida transcurrió el proyecto, se volvió mas preciso en su estimación.

Para la estimación, utilizamos una versión modificada de la técnica de **Poker Planning**. Esta consiste en seleccionar la tarea que se va a estimar y luego, cada miembro del equipo debe proponer la estimación del esfuerzo que considera necesario para completar la tarea en cuestión (en teoría, la técnica se realiza con cartas y todas se dan vuelta a la vez, pero dado que nuestras Plannings se realizaban por Zoom, y por cuestiones de practicidad, adaptamos la técnica a esta manera). Si hay diferencias, se discuten las razones. Los miembros con las estimaciones más bajas y altas dan sus razones y se repite el proceso hasta llegar a un consenso sobre el esfuerzo necesario. De esta forma, se aprovecha la sabiduría colectiva para mejorar la precisión de las estimaciones.

Figura 9.5: Ejemplo de estimación utilizando Story Points

9.1.4.2 Daily meeting

Si bien los integrantes del equipo tenían otras responsabilidades además de este proyecto, hacíamos el esfuerzo de realizar este evento todos los días al mediodía.

Estas reuniones consistían en videollamadas de aproximadamente 20 minutos, en donde cada uno exponía sus avances y comentaba si se había bloqueado con algo en particular. De esta forma, en caso de que un integrante necesitase ayuda de otro, se podía solucionar los problemas rápidamente. Sin embargo, debido a las diversas responsabilidades de los integrantes del grupo, este evento no se pudo realizar la totalidad de los días que duró el proyecto. En estos casos, cada uno enviaba sus avances vía Slack o WhatsApp.

9.1.4.3 Reviews

Al final de cada Sprint, el equipo llevaba adelante la Review del incremento, donde el objetivo era evaluar el incremento del producto realizado. Para ello, los primeros Sprints del proyecto (referentes a las primeras historias de usuario, integración con Fenicio, y mejoras propuestas por nosotros mismos) nos reuníamos solamente los integrantes del equipo para llevarla a cabo. Luego, una vez que Casashub se convirtió en cliente, empezamos a realizar las Reviews junto a **Andrés Ruvertoni**, fundador de Casashub. El objetivo de este encuentro era presentar los avances logrados en el último sprint y recibir retroalimentación directa tanto del Product Owner, como del cliente. Así, lográbamos recibir feedback de las funcionalidades y construir el producto de manera colaborativa.

La retroalimentación obtenida en estos eventos resultó **sumamente valiosa** para el desarrollo de las funcionalidades del producto. Para nuestra sorpresa, mirándolo en retrospectiva, muchas de las funcionalidades del producto, y sus criterios de aceptación, surgían de las Reviews. Algunas de ellas: poder buscar propiedades a estrenar, recibir un feedback de búsqueda, o las preguntas de refinamiento según la propiedad.

Luego, también surgían mejoras como por ejemplo, cómo hacer el botón de **Buscar por mi** más atractivo para incentivar al usuario a utilizarlo más y promover más pedidos hacia las inmobiliarias.

En particular, nos quedamos con un comentario en particular de Andrés: *"Estamos muy contentos con wisello. Mejoró abismalmente desde el principio"*.

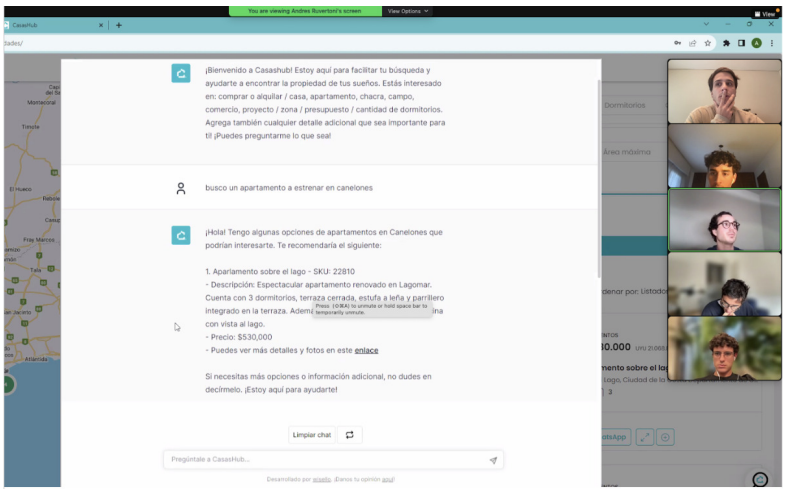


Figura 9.6: Review con Andrés Ruvertoni

9.1.4.4 Retrospective

Este evento, realizado también al final de cada Sprint, fue un pilar fundamental en la gestión del proyecto, pues nos permitió el aprendizaje continuo y con este, la mejora continua del proceso. La auto-reflexión nos incitó a mejorar nuestros procesos, herramientas e interacciones, adaptándonos semana a semana ante nuestros errores, y manteniéndonos eficaces en un entorno dinámico. El *output* de la misma siempre fue un plan de acción con items concretos para implementar en el transcurso del próximo Sprint.

En cuanto a la dinámica del evento, el Scrum Master optó por utilizar la metodología DAKI. Esta consiste en un brainstorming de ideas para cada uno de los cuadrantes: **Drop**, **Add**, **Keep** e **Improve**. El objetivo de utilizar DAKI es que cada integrante pueda aportar su punto de vista acerca de los puntos débiles y fuertes del proceso [50]. A continuación, adjuntamos una imagen del tablero de la retrospectiva utilizado en el sprint 7.

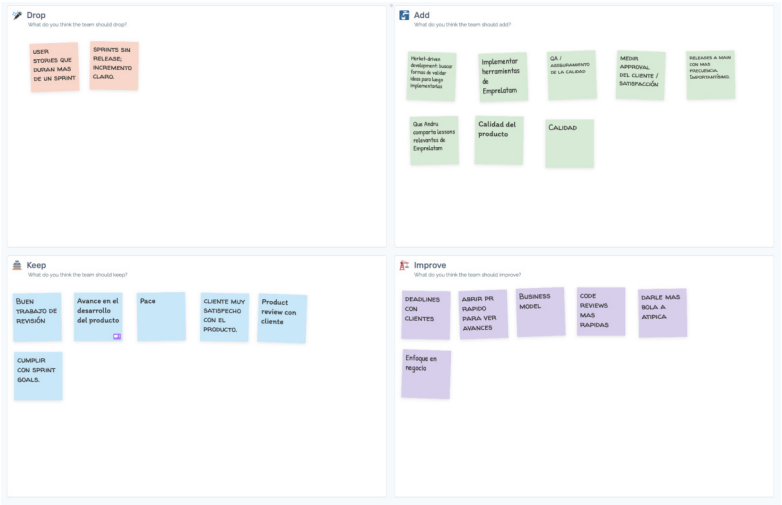


Figura 9.7: Tablero DAKI de la Retrospective

Tras recibir las perspectivas de todos los integrantes del equipo, realizabamos una puesta en común de cada uno de los aportes realizados al tablero DAKI.

Una vez finalizada la discusión, como accionable a futuro, creábamos *follow-up actions* para implementar en el próximo Sprint, en base a lo discutido durante la Retrospectiva, y así corregir lo que creíamos que no estábamos realizando correctamente o podíamos mejorar:

Figura 9.8: *Follow-up actions* de la retrospectiva

9.2 Gestión de la comunicación

El desarrollo exitoso de un proyecto de software depende de una comunicación efectiva, la cual constituye un pilar fundamental para el trabajo en equipo. En este proyecto específico, el equipo requirió de tres frentes principales de comunicación: la comunicación interna dentro del equipo, la comunicación externa con los clientes, y la comunicación con el tutor del proyecto.

9.2.1 Comunicación entre el equipo

La comunicación interna de nuestro equipo se realizó principalmente de forma virtual, lo cual hizo esencial una coordinación cuidadosa.

Desde el inicio, establecimos reuniones diarias de aproximadamente 20 minutos a través de Zoom, con el fin de alinearnos en nuestras actividades y resolver dudas sobre las tareas individuales. Adicionalmente, optamos por mantener un canal de comunicación abierto para consultas y asuntos cotidianos, permitiéndonos abordar interrogantes sin tener que esperar hasta la próxima reunión diaria.

Para nuestras comunicaciones, empleamos dos herramientas principales: WhatsApp y Slack, cada una con propósitos específicos.

9.2.1.1 WhatsApp

Esta plataforma fue utilizada como herramienta de comunicación informal, enfocándose en temas relacionados con la organización del equipo.

9.2.1.2 Slack

Elegimos Slack como la herramienta principal de la comunicación interna del equipo. La elección se fundamentó en su capacidad para crear múltiples canales, lo que permite abordar temas específicos en cada uno de ellos, además de su potente motor de búsqueda que facilita el acceso a conversaciones anteriores de manera sencilla. Debido al uso de esta herramienta, evitamos la pérdida de información, el tener cada tema separado en una conversación diferente nos ayudó mucho a la organización.

9.2.2 Comunicación con los clientes

A lo largo del proyecto, mantuvimos una interacción constante con nuestros clientes. Para nosotros, era esencial obtener su retroalimentación para poder incorporarla al producto. Como mencionamos anteriormente, en cada uno de los sprints realizamos *reviews* con ellos de manera virtual a través de Zoom. Asimismo, con cada uno de los clientes, establecimos un grupo de WhatsApp con el objetivo de mantener un canal de comunicación abierto en caso de que surgieran algun tipo de dudas.

9.2.3 Comunicación con el tutor

El tutor del proyecto desempeñó un papel de soporte esencial en su desarrollo, actuando como guía para orientarnos hacia el objetivo. Mantuvimos una comunicación activa con el tutor; desde el inicio del proyecto, programamos reuniones virtuales quincenales de aproximadamente 30 minutos, durante las cuales le informábamos sobre nuestros avances y resolvíamos las dudas que surgían. Además, para consultas que requerían atención inmediata, establecimos un canal de comunicación a través de WhatsApp, que nos permitía hacer preguntas en cualquier momento.

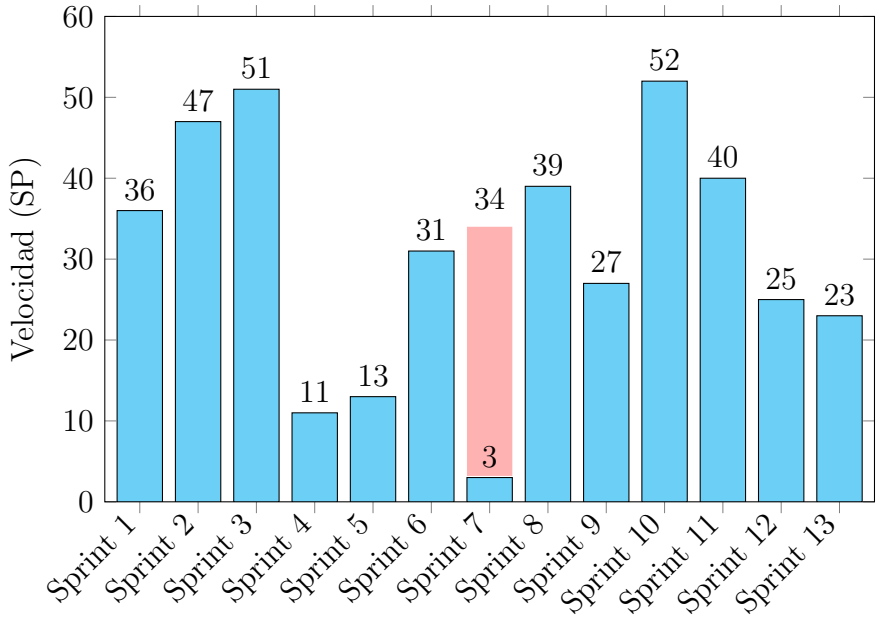
9.3 Métricas de gestión y construcción del producto

Con el fin de mejorar continuamente nuestro proceso de gestión través de las retrospectivas, identificar problemas en este, y planificar de forma más precisa, decidimos recopilar una serie de métricas, tanto de gestión, como de construcción del producto.

9.3.1 Velocidad del equipo

Una de las métricas que el equipo decidió medir en cada Sprint es la velocidad del equipo, que refiere a la cantidad de puntos de historia completados en un mismo Sprint. Esta métrica fue muy útil al momento de fijar los objetivos de cada Sprint, ya que nos permitía estimar de forma más certera la capacidad del equipo.

A continuación se presenta la evolución de la velocidad del equipo para los 13 *sprints* completados:



Como se puede observar, existen variaciones y casos atípicos en la velocidad del equipo, a lo largo de los distintos *sprints*. A continuación el porqué de los mismos:

- **Sprint 4**: Período de entregas y parciales de final de semestre, por lo que inevitablemente la dedicación al proyecto fue menor que en Sprints anteriores, y como consecuencia disminuyó la velocidad.
- **Sprint 5**: Ausencia de dos integrantes del equipo por viajes laborales y recreativos.
- **Sprint 7**: La métrica presentada no es representativa de la realidad, sino que es fruto del mal uso del board que fue realizado en este Sprint, y que fue discutido en la retrospectiva correspondiente. No se reflejó el trabajo realizado en el board (no se marcaron las tareas finalizadas como *Done*, por ejemplo), haciendo que la velocidad sea **3**, cuando en realidad fue **34**.

A partir de la evolución de la velocidad provista podemos extraer algunas conclusiones.

Se puede observar que la velocidad no fue constante a lo largo del proyecto. La razón principal de esto fue que, paralelamente al desarrollo del producto, el equipo tenía otras obligaciones académicas (revisiones, otras materias, entregas), laborales y del emprendimiento como tal (acciones comerciales, reuniones con clientes, trabajo con agencia de Marketing), que afectan esta métrica, como vimos en caso de los Sprints 4 y 5.

La velocidad al comienzo del proyecto no fue baja, indicando que no hubo curva de adaptación del equipo, lo cual creemos que se dio en gran parte porque ya habíamos trabajado juntos anteriormente.

Pasados los Sprints 4 y 5, el equipo mantuvo una velocidad creciente, lo cual fue fuente de motivación, desde el punto de vista del producto y su proceso.

Finalmente se puede ver un decaímiento en los Sprints 12 y 13. Esto se debe a que fueron Sprints cuyo foco estuvo en el aseguramiento de la calidad, lo cual representa algo totalmente nuevo en el campo de la IA Generativa, por lo que implicó mucho trabajo de investigación. Dado que el equipo no estaba acostumbrado a estimar tareas de esta índole ni estaba familiarizado con las tecnologías, se las subestimó durante la planificación, causando que como resultado la velocidad para estos Sprints decaiga.

9.3.2 Sprint burndown

Sin ser una métrica como tal, se utilizó este gráfico para el análisis global del Sprint: conclusiones de este, errores y oportunidades de mejora.

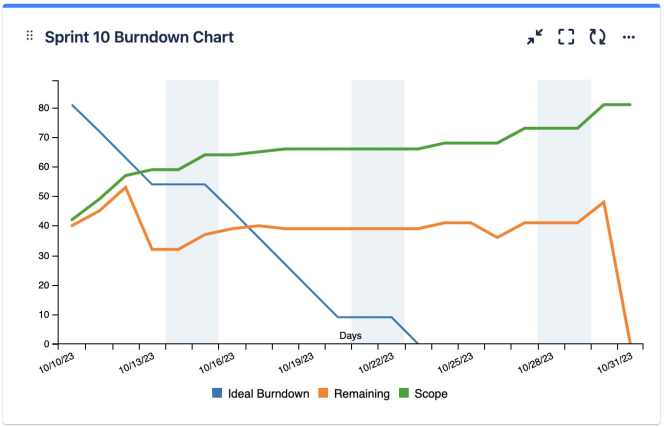


Figura 9.9: Sprint 10 Burndown Chart

El **Sprint 10** se extendió 1 semana pues se realizó la Segunda Revisión, dejando así una semana extra para elaborarla. Esto impactó en que no se alcance el `Ideal Burndown`, pero al tener más tiempo, el equipo completó los SP asignados.

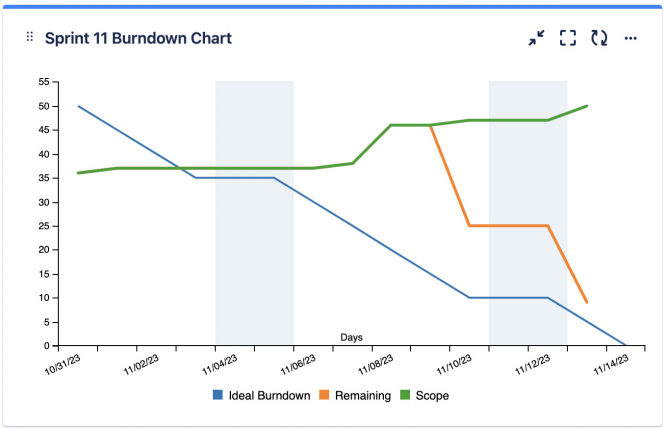


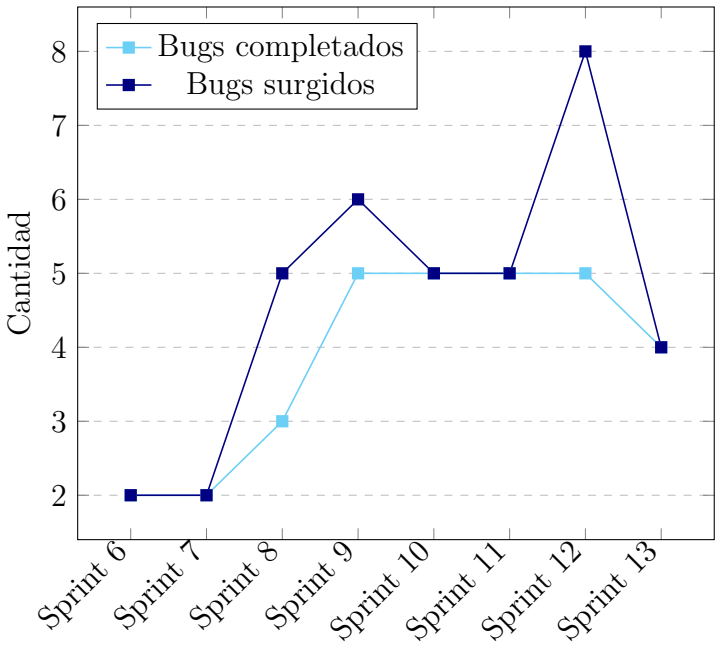
Figura 9.10: Sprint 11 Burndown Chart

En el **Sprint 11**, por ejemplo, el equipo dejó todo el trabajo para realizar la segunda mitad del Sprint, y como consecuencia, no se pudo completar algunos, aunque pocos, de los SP asignados. Tal como se discutió en la sección 9.1.3 de Sprints, vemos de nuevo cómo el *Scope* del Sprint se agranda a medida estos transcurren.

9.3.3 Cantidad de bugs

Decidimos trackear los bugs en cada Sprint con el objetivo de medir el impacto de las medidas implementadas para el aseguramiento de la calidad del sistema (12). A su vez, decidimos contabilizar la cantidad de estos bugs que fueron solucionados, para medir la capacidad del equipo para lidiar con ellos.

Evolución de Bugs durante los Sprints



Pasados los Sprints 6 y 7, la cantidad de **bugs surgidos** aumentó drásticamente, lo cual se explica por un aumento en la complejidad del proyecto, y medidas de aseguramiento de la calidad no implementadas hasta el momento.

Luego, la cantidad de bugs surgidos se estabiliza con variaciones de +- 1 bug (a excepción del Sprint 12), y comienza a bajar en el Sprint 13; Sprint que tuvo un foco en aseguramiento de la calidad. Además, es importante comentar que la mayoría de estos bugs están vinculados a comportamientos no deseados del chat según las expectativas del cliente, más que a fallos propiamente dichos del sistema.

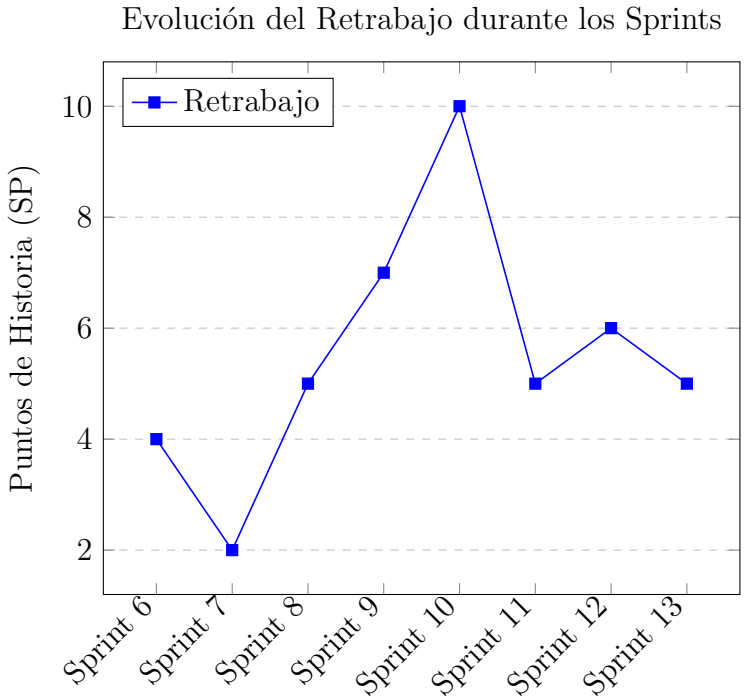
En relación a los **bugs completados**, una vez que los bugs surgidos fueron en aumento, el equipo al principio no logró equipararlos. Sin embargo, a partir del Sprint 9, el equipo alcanza una cercanía considerable (a excepción del Sprint 12). En este punto, en general, si un bug no era completado se debía a que se priorizaban otras tareas antes (no era de tanta urgencia).

En el backlog, quedaron tres bugs a resolver: la falta de un botón para volver atrás en **Buscar por mi**, la falta de feedback en la descarga de conversaciones, y la representación errónea (ocasional) de caracteres con tildes en el chat:

Figura 9.11: Bug de tildes

9.3.4 Retrabajo

Otra métrica de construcción de producto recopilada fue el retrabajo, para entender cuánto esfuerzo se le dedicaba a resolver bugs, o a corregir/hacer tareas antes realizadas. A continuación presentamos su evolución, en puntos de historia:



A partir del Sprint 7, el retrabajo fue en aumento, lo cual condice con la métrica de Bugs, que también fueron en aumento al principio. Sin embargo, no se estabiliza luego a partir del Sprint 8, como lo hacían los bugs. Esto nos hace pensar que los bugs fueron siendo cada vez más complejos y costosos de resolver, aumentando el retrabajo. Luego de su pico en el Sprint 10, comenzó a bajar, coincidiendo con la implementación de medidas de aseguramiento de la calidad.

10 Gestión de riesgos

Este capítulo trata sobre como el equipo lidió con los riesgos y sus potenciales consecuencias durante el desarrollo del proyecto. Hablaremos de la identificación de los riesgos presentes en el mismo, de las estrategias de mitigación que llevamos a cabo y de cómo fueron evolucionando las exposiciones a dichos riesgos.

Antes de comenzar a hablar sobre cómo el equipo gestionó los riesgos en el proyecto, queremos determinar qué es un riesgo. En el contexto del desarrollo de un proyecto de software se definen a los riesgos como *"la medida de la probabilidad y severidad de que se produzcan efectos adversos en el desarrollo, adquisición, o mantenimiento del sistema."* [51]

10.1 Identificación y estrategias de mitigación

Para identificar los riesgos del proyecto, el equipo utilizó la técnica de *brainstorming*. Esto permitió tener los puntos de vista de cada uno de los integrantes acerca de lo que cada uno consideraba como un riesgo potencial para el proyecto. Tras debatir sobre cada uno de los riesgos planteados, logramos identificar los siguientes riesgos y plantear las estrategias de mitigación correspondientes a cada uno de ellos.

R1 - Enfermedad o abandono desestabiliza el equipo de trabajo

Justificación: La posibilidad de que uno de los miembros del equipo se enferme gravemente o abandone el proyecto, lo que podría causar retrasos significativos.

Estrategia de mitigación: Diseñamos un equipo con habilidades y conocimientos complementarios para reducir la dependencia de un solo miembro.

R2 - Filtración datos sensibles de los clientes

Justificación: Dado que manejamos datos de clientes, la falta de seguridad podría tener graves consecuencias en términos de pérdida de confianza y sanciones legales.

Estrategia de mitigación: Implementamos protocolos de seguridad robustos y estuvimos en constante monitoreo de posibles vulnerabilidades. Seleccionamos las tecnologías y herramientas a utilizar en base a la privacidad y seguridad de datos. Asimismo, manejamos la menor cantidad de datos sensibles de los clientes.

R3 - Asistente da recomendaciones inadecuadas a los clientes finales

Justificación: Existe el riesgo de que el asistente (*wisello*) ofrezca recomendaciones inadecuadas, lo que podría afectar la satisfacción del cliente.

Estrategia de mitigación: Establecimos procesos de control de calidad y retroalimentación constante para garantizar que las recomendaciones sean precisas y útiles.

R4 - La competencia torne obsoleto el producto

Justificación: El mercado de asistentes de compras es competitivo; por lo tanto, la competencia constante es un riesgo inherente.

Estrategia de mitigación: Monitoreamos de cerca a la competencia y ajustamos nuestra estrategia para mantenernos competitivos en el mercado. Diferenciamos nuestra propuesta de valor para que sea difícilmente cubierto por un producto generalista.

R5 - Fallas de sistema por cambios de política o fallas en proveedores de tecnologías

Justificación: Dependemos de proveedores de tecnología y servicios, lo que nos hace vulnerables a posibles cambios de política o fallas de ellos. Al ser tecnologías nuevas y controversiales, podrían surgir legislaciones o cambios de política inesperados.

Estrategia de mitigación: Monitoreamos de cerca a nuestros proveedores tecnológicos para mantenernos alerta ante cualquier evento que pueda afectar al sistema.

R6 - Requerimientos de customización perjudica el desarrollo de nuevas funcionalidades generales del producto

Justificación: Los requerimientos de customización de cada cliente pueden llegar a ser tan costosos que perjudique el desarrollo general del sistema.

Estrategia de mitigación: Trabajamos para establecer un balance entre los requerimientos específicos de un cliente y aquellos generales del producto. A su vez, evaluamos los requerimientos de los clientes particulares para identificar con mayor practicidad las necesidades generales del mercado.

R7 - La mala actuación del asistente pasa inadvertida

Justificación: Modelos generalistas frecuentemente alucinan o responden a los usuarios con información falsa o incluso dañina. Existe la posibilidad de que no logremos advertir este comportamiento del sistema.

Estrategia de mitigación: Implementamos mecanismos de observabilidad para identificar fallas de forma temprana y resolverlas.

R8 - Desafíos tecnológicos perjudican la planificación y el desarrollo del proyecto

Justificación: El uso de tecnologías recientes puede causar dificultades en la planificación, proceso de desarrollo, y resultados finales del proyecto.

Estrategia de mitigación: Estudiamos en profundidad las nuevas tecnologías antes de comenzar a implementar soluciones. Realizamos tareas de Spike siempre que fue necesario.

R9 - Quedarnos sin presupuesto

Justificación: Tanto para el desarrollo tecnológico del sistema como para el desarrollo comercial del producto, es necesario de un presupuesto para cubrir los gastos.

Estrategia de mitigación: Obtuvimos el fondo concursable VIN de ANII/ANDE, correspondiente a UYU$ 200.000. A su vez también fuimos aceptados al programa Microsoft for Startups Hub, a partir del cual fuimos otorgados créditos de infraestructura en la nube y en modelos de IA generativa.

10.2 Análisis cuantitativo

A cada riesgo identificado se le atribuían dos características:

- **Probabilidad de ocurrencia (P)**: valor numérico que indica la probabilidad de ocurrencia del riesgo para un momento determinado. El valor de la probabilidad va desde 0 a 1.
- **Esfuerzo (E)**: valor numérico que indica la cantidad de días ideales que le tomaría al equipo recomponerse ante el impacto de dicho riesgo.

Estos dos atributos fueron tomados para calcular la *exposición* del proyecto ante un determinado riesgo. A continuación definimos la ecuación utilizada para calcular la exposición.

$exposición = 10 * P * E$

La exposición es calculada como el producto de la *probabilidad de ocurrencia* y el *esfuerzo*, multiplicados por la constante 10. Esta constante fue introducida para que los valores de la exposición sean números enteros.

A continuación adjuntamos un ejemplo de la asignación de la probabilidad de ocurrencia y esfuerzo, así como el cálculo de la exposición, para los R1, R2 y R3 en el Sprint 12.

	Probabilidad de ocurrencia (%)	Esfuerzo (días ideales)	Exposición
R1 - Enfermedad o abandono desestabiliza el equipo de trabajo	0.05	10 días	5
R2 - Se filtran datos sensibles de los clientes	0.05	4 días	2
R3 - Asistente da recomendaciones inadecuadas a los clientes finales	0.6	2 días	12

Figura 10.1: Cálculo de exposición de riesgos

10.3 Seguimiento

En cada retrospectiva, se re-evaluaban los riesgos, y se les hacía un seguimiento uno a uno. Esto se debía a que, tras transcurrir el proyecto, la probabilidad de ocurrencia y el esfuerzo que implicaría solucionar el impacto de cada uno de estos riesgos iría variando, y por lo tanto, la exposición del proyecto a dicho riesgo también. Como fue mencionado, esta re-evaluación se hacía en la Retrospectiva, permitiendo reflexionar sobre la evolución de la exposición del equipo al riesgo. La siguiente gráfica representa la evolución de la exposición de los riesgos a lo largo del proyecto:

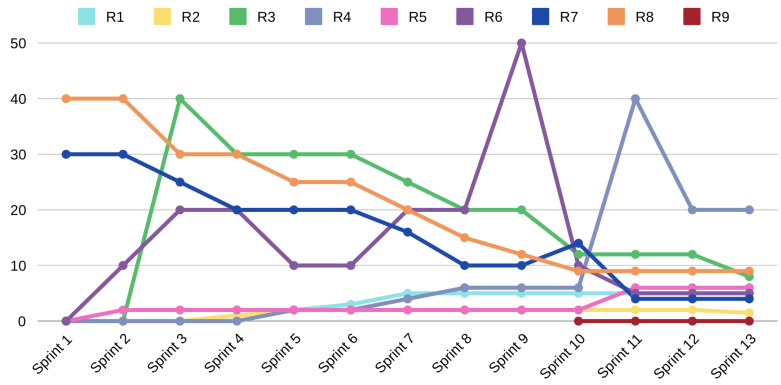


Figura 10.2: Evolución de la exposición de los riesgos

También aparecieron riesgos nuevos o modificaciones en los riesgos existentes. Por ejemplo, en el Sprint 10 se agregó: *"R9 - Quedarnos sin presupuesto"*, como se ve en la gráfica. Esto sucedió porque nos percatamos de que era un riesgo que si bien todavía tenía probabilidad 0 de ocurrencia, debíamos considerar pues la fecha de vencimiento de los créditos de Azure cada día se acercaba más. Por otro lado, en el Sprint 11, se modificó *"R6 - Requerimientos de customización perjudican el desarrollo de nuevas funcionalidades* ***generales*** *del producto"* (antes era *"Requerimientos de customización perjudican el desarrollo de nuevas funcionalidades del producto"*). Nos dimos cuenta de que el enfoque del riesgo no estaba bien expresado, por ende el cambio. Si bien los requerimientos de customización son el producto, en exceso pueden afectar el desarrollo de funcionalidades generales del mismo (monitoreo, calidad, UX, entre muchas otras), de allí el riesgo.

En cuanto a los riesgos que mantuvieron una exposición baja durante el desarrollo del proyecto (como por ejemplo *"R1 - Enfermedad o abandono desestabiliza el equipo de trabajo"*), igualmente decidimos aplicar sus estrategías de mitigación para evitar un aumento en su exposición (manteniéndolos así con bajo impacto).

10.3.1 Puntos de quiebre de cada riesgo

R1 - Enfermedad o abandono desestabiliza el equipo de trabajo

La exposición de este riesgo fue baja durante todo el transcurso del proyecto. En su comienzo, el equipo fijó su exposición en 0, ya que todos los integrantes teníamos los mismos conocimientos acerca del proyecto. Sin embargo, mirándolo en retrospectiva, creemos que esto fue desacertado, dado que el hecho de que uno se enfermara o abandonara hubiera afectado el alcance del proyecto, lo cual no consideramos, por lo que la exposición al riesgo tendría que haber sido inicialmente mayor. A medida que fuimos avanzando cada integrante tenía su área de interés o dominio, por lo que el esfuerzo que implicaría contrarrestar las consecuencias del impacto del riesgo comenzaron a ser mayores a 0. Sin embargo, el equipo evitó que la exposición del riesgo siguiera subiendo al intentar que al menos dos integrantes contribuyan en cada aspecto del proyecto. La estrategia de mitigación, en este caso, impidió que aumentara la exposición del mismo.

R2 - Filtración datos sensibles de los clientes

Utilizando estrategias correctas de mitigación, el equipo pudo mantener muy baja la exposición de este riesgo en todo momento. La selección de herramientas que cuiden la privacidad de los datos, y trabajar con la cantidad mínima de datos sensibles de los clientes fueron las dos razones que nos permitieron mitigar este riesgo.

R3 - Asistente da recomendaciones inadecuadas a los clientes finales

Al principio, durante la etapa de ideación, no teníamos un MVP desarrollado. Una vez que lo desarrollamos, la exposición de este riesgo subió a niveles altos, pues notamos que la probabilidad de que ocurrieran malas recomendaciones era muy alta.

Luego, si observamos la gráfica, podemos ver que el riesgo tuvo una evolución muy positiva, de descenso constante. Esto sucedió porque el equipo fue adquiriendo experiencia con las tecnologías, haciendo que el tiempo de resolución de una mala recomendación sea menor. Además, se implementaron medidas como la calificación de respuestas (para visualizar la calidad de las respuestas) o el testing de AI (para verificar la correctitud de respuestas base), que impactaron positivamente en esta evolución.

R4 - La competencia torne obsoleto el producto

Cuando comenzamos con el proyecto, realizamos un análisis profundo de la competencia. Decidimos tomar un enfoque mucho menos generalista que las alternativas existentes en el mercado y orientamos nuestro producto hacia la personalización y el valor agregado en e-commerce. En esta instancia, al tener un diferencial marcado, la probabilidad de que la competencia deje obsoleto al producto era relativamente baja.

Cuando el proyecto se encontraba aproximadamente en el Sprint 11, OpenAI lanzó OpenAI Assistants. Esto le daba la oportunidad a los clientes de desarrollar asistentes (no tan poderosos como *wisello*), pero asistentes en fin, con bastante facilidad. Como respuesta a esto, decidimos pivotear y orientar el producto a plataformas. El equipo tenía relación con la plataforma de e-commerce Fenicio, por lo que decidimos orientar *wisello* hacia plataformas de este estilo, comenzando con Fenicio.

R5 - Fallas de sistema por cambios de política o fallas en proveedores de tecnologías

Al ser tan baja la exposición de este riesgo, no se le dio mucha importancia al mismo. Sin embargo, alrededor del **Sprint 10** empezamos a notar que OpenAI se caía con frecuencia, haciendo que *wisello* no esté disponible en esos momentos. Estos *outages* de OpenAI duraban unos pocos minutos, por lo que la exposición del riesgo creció moderadamente. Aún con estos *outages*, mes a mes, OpenAI se mantuvo con un *uptime* mayor al 99 %. [52]

R6 - Requerimientos de customización perjudican el desarrollo de nuevas funcionalidades generales del producto

Como se puede ver en la gráfica, la exposición de este riesgo varió bastante durante el proyecto. A medida que fuimos incorporando nuevos clientes, este riesgo crecía, pues cada uno de ellos nos pedía funcionalidades personalizada que se encontraban por fuera de las funcionalidades convencionales de *wisello*.

Entre el **Sprint 8** y **Sprint 10**, se llevó a cabo la implementación de la solución personalizada para el cliente CasasHub, razón por la que se da un pico pronunciado en R6.

R7 - La mala actuación del asistente pasa inadvertida

La tendencia de la exposición de este riesgo fue siempre en descenso. A medida que el proyecto fue avanzando, el equipo pudo centrar su atención en construir un sistema más robusto e implementar diversos mecanismos de observabilidad para poder detectar fallas en las recomendaciones. Aunque es cierto que todavía estamos expuestos a este riesgo, esta exposición se redujo mucho con la integración de Langsmith, herramienta de monitoreo que nos permite ver cada llamada a *wisello*: el feedback del usuario, marcar comentarios sobre las respuestas, o agruparlas (erróneas, correctas). De esta forma, generamos una base de conocimiento compartido del equipo para monitoreo.

R8 - Desafíos tecnológicos perjudican la planificación y el desarrollo del proyecto

La exposición a este riesgo comenzó siendo relativamente alta por ser tecnologías inicialmente desconocidas, pero fue disminuyendo a lo largo de los Sprints. Como ya mencionamos previamente, el equipo fue adquiriendo experiencia en las tecnologías utilizadas y cada vez eran más las herramientas que teníamos a disposición para la implementación tecnológica del proyecto.

R9 - Quedarnos sin presupuesto

La primera etapa del proyecto no necesitó de un presupuesto para poder ser ejecutada. El desarrollo del sistema lo realizamos localmente, y el gasto en servicios de IA era muy bajo. A medida que el proyecto fue avanzando y se tomó la decisión de desplegar el sistema en la nube, era necesario un presupuesto para comenzar a cubrir estos gastos. Fue entonces cuando aplicamos a Microsoft for Startups, lo cual nos permitió obtener USD 25.000 en crédito para Azure. A su vez, aplicamos al fondo concursable VIN con el cual obtuvimos UYU 200.000. Luego de contar con estos dos beneficios, la probabilidad de quedarnos sin presupuesto disminuyó a 0. Esto perduraría hasta la fecha del vencimiento de los créditos de Microsoft (Azure), en octubre 2024.

La finalización del presupuesto del VIN no aumenta la exposición al riesgo dado que este presupuesto se utilizó con propósitos de Marketing, creación de emails, etc, de los cuales se puede prescindir, pero la infraestructura (Azure) es esencial para el mantenimiento del sistema.

11 Gestión de la configuración

El propósito de este capítulo es describir cómo se gestionó el área de SCM (Software Configuration Management) en *wisello*. La gestión de la configuración nos permitió mantener un mayor control sobre los diferentes elementos de configuración del software: el código fuente, la documentación, archivos de configuración, y herramientas de desarrollo. Al tratarse de un proyecto de larga duración, en el que cada miembro del equipo interactuaba individualmente con los distintos elementos, muchas veces a la misma vez que otro también lo hacía, resultó esencial gestionar y versionar cada uno de estos elementos de configuración.

11.1 Identificación de ECS

En la siguiente tabla se detallan cada uno de los elementos de configuración de software en los cuales el equipo trabajó a lo largo del desarrollo del proyecto.

Tipo	Elemento	Descripción
Documentos	Carpeta anteproyecto	Documento entregado para el CIE incluyendo la justificación del proyecto, descripción de la solución y propuesta de validación del problema.
Documentos	Especificación de Requerimientos	Documentos realizados durante la etapa de validación que contienen los RF y RNF de la solución.
Documentos	Informes de las revisiones	Documentos con las conclusiones y accionables correspondientes a las tres revisiones del proyecto.
Documentos	Presentaciones de las revisiones	Documentos que contienen las presentaciones utilizadas en cada una de las revisiones.
Documentos	Eventos de Scrum	Documentos en donde se detallan las conclusiones alcanzadas en los eventos: Sprint Review y Sprint Retrospective.
Documento	Informe de avance	Documento que contiene un resumen del avance del proyecto hasta lo realizado en la tercera revisión.
Documentos	Validación secundaria	Documentos que contienen datos y conclusiones extraídos a partir del proceso de validación secundaria
Documentos	Validación primaria	Documentos que contienen entrevistas con expertos en el dominio de e-commerce y dueños de empresas.
Documentos	Gestión de riesgos	Documento que contiene los riesgos especificados y las estrategias de mitigación para cada uno de ellos.
Documento	Tabla de riesgos	Tabla de riesgos con su seguimiento y evolución de la exposición de los mismos.
Documento	Criterios de priorización	Documento con los criterios establecidos para la priorización de user stories para el proyecto.
Documento	Gestión del proyecto	Documento que establece todo lo relacionado a la gestión del proyecto: roles, días y horarios de eventos, estándares para especificar user stories, duración de Sprints, métricas a recopilar, estructura del board.

Documento	Seguimiento de métricas	Documento con los valores de las métricas para cada Sprint.
Documento	Objetivos del proyecto	Documento que establece los objetivos del equipo para el proyecto.
Documentos	Reuniones con profesores	Reuniones realizadas con Lic. Juan Gabito y Dr. Sergio Yovine.
Documentos	Reuniones con clientes	Notas y conclusiones de reuniones con distintos clientes.
Documento	Backlog	Backlog priorizado de user stories y bugs.
Documento	Board	Board para el sprint actual.
Documentos	Documentación	Documentación del proyecto.
Documento	Bibliografía utilizada	Documentos, libros, e-books, artículos, informes utilizados como fuente bibliográfica para la ejecución y documentación del proyecto.
Código	Aplicación backend	Código de la aplicación Python, Dockerfile, docker-compose, dependencias, workflows de automatización de tests, y pipelines de despliegue.
Código	Aplicación bubble-chat-server	Código de la aplicación Python, dependencias y pipelines de despliegue.
Código	Aplicación data-pipelines	Código de la aplicación Python y dependencias.
Código	Aplicación frontend - chat	Código de la aplicación Next.js, dependencias y workflows de automatización de tests.
Código	Aplicación frontend - Plataforma de visualización	Código de la aplicación Next.js y dependencias.
Código	Archivos de configuración de NewRelic	Archivos de configuración para la conexión con NewRelic (APM y logging).
Código	Load test	Script Python realizado para la prueba de carga.

Tabla 11.1: Elementos de ECS

11.2 Herramientas de SCM

Se seleccionaron distintas herramientas para gestionar cada uno de los distintos elementos de SCM. A continuación se detalla el uso de cada una de estas herramientas.

11.2.1 Desarrollo de software

Como herramienta para la gestión de versiones y trackeo de cambios del código fuente se optó por **Git**. La decisión se basó en la experiencia previa del equipo utilizándola, así como en su amplia adopción en proyectos de desarrollo tecnológico como este.

Para la gestión de repositorios remotos, optamos por **Github**, también motivados por la familiaridad que el equipo tenía con esta herramienta. Otros aspectos que influyeron en la elección incluyen las funcionalidades de GitHub Actions, que facilitan la ejecución de pipelines de despliegue hacia Azure o Vercel y la automatización de pruebas, así como la posibilidad de crear Pull Requests, llevar a cabo revisiones de código, entre otros beneficios que el equipo utilizó para gestionar el desarrollo. En total, se crearon tres repositorios en Github: uno para el backend, otro para el chat-frontend y otro para la web-platform (plataforma de visualización). Esto es, para cada componente del sistema que el equipo creyó necesario tener que versionar en forma independiente, y desacoplar del resto.

11.2.1.1 Nombramiento de ramas

El nombramiento de ramas se compone como una parte esencial del control de versiones, en tanto permite el desarrollo en paralelo de features, hacer *releases* de forma eficiente y ayuda al mantenimiento a largo plazo del repositorio.

Como fue mencionado en capítulos anteriores, las ramas en *wisello* se corresponden con ambientes del sistema:

- master: ambiente de producción.
- develop: ambiente de *staging*.

En lo que refiere a las *feature branches* (aquellas sobre las cuales se crean Pull Requests para integrar el código al ambiente de prueba *staging*), estas fueron nombradas bajo el formato **WIS-X**, siendo X el número asignado de la tarea en Jira.

11.2.1.2 Gestión de dependencias

Otra parte crucial de SCM, que nos permite gestionar las dependencias del sistema, con el objetivo de que todos los desarrolladores utilicen las mismas versiones de bibliotecas externas, y así evitar la incompatibilidad entre distintos ambientes de desarrollo.

En el caso de Python, utilizamos el gestor de paquetes oficial `pip`, y todas las dependencias necesarias se definieron en el archivo `requirements.txt`. El comando para instalar dependencias es el siguiente: `pip3 install -r requirements.txt`.

En el caso de Typescript, utilizamos también el sistema de gestión de paquetes por defecto `npm`, y se utilizó el archivo `package.json` para declarar todas las dependencias necesarias. El comando para instalar dependencias es: `npm i`.

11.2.2 Documentación y proceso de gestión

Para todos los documentos que no están relacionados con el desarrollo del software el equipo eligió tres herramientas: **Google Drive**, **Overleaf** y **Jira**.

11.2.2.1 Jira

Como ya fue comentado anteriormente, esta herramienta fue seleccionada para mantener el board del Sprint actual, como también el backlog de producto del proyecto.

Además, a nivel de SCM, también hicimos uso de las `Project Pages` de Jira para documentar cada uno de los eventos de Scrum en cada Sprint. A su vez, también utilizamos su sección de `Dashboards`, a partir de la cual consultamos las métricas de gestión de un Sprint dado, para ser analizadas en su retrospectiva correspondiente.

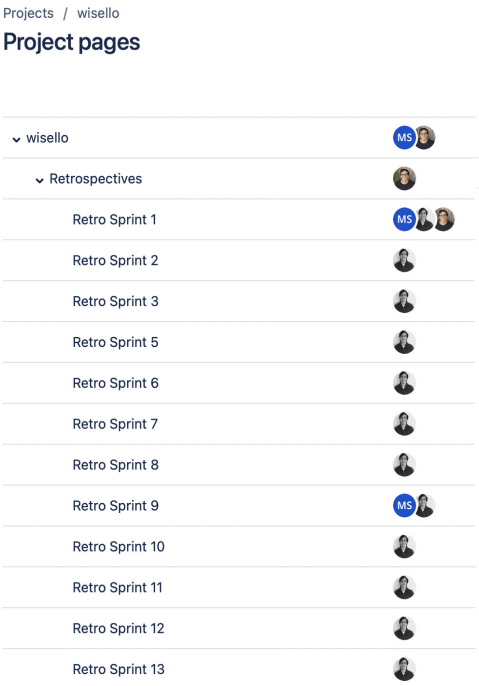


Figura 11.1: Documentación de retrospectivas en Jira

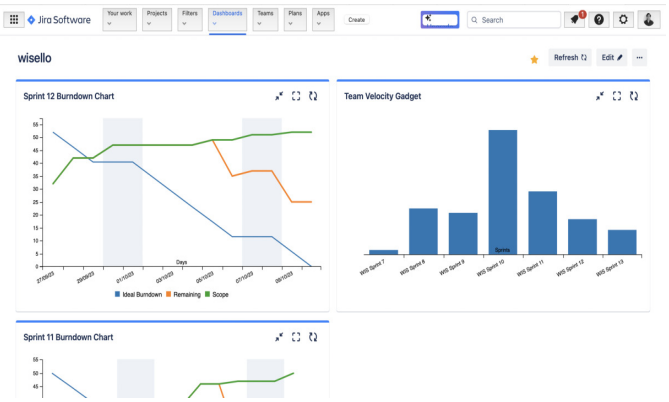


Figura 11.2: Dashboards de métricas en Jira

11.2.2.2 Google Drive

Para aquellos documentos no relacionados con la gestión del proyecto, decidimos utilizar **Google Drive** como herramienta de almacenamiento centralizado. Para ello creamos una carpeta compartida con distintas subcarpetas, entre ellas: `Ing. Requerimientos`, `Anteproyecto`, `Primera revisión`, `Segunda revisión`, `Tercera revisión`, `Riesgos`, etc. Cada una con sus distintos documentos, de forma que cada uno de los integrantes podía acceder desde su cuenta de Google a todos los elementos presentes dentro de la misma. Dentro del ecosistema, utilizamos **Google Docs** para la creación de documentos y **Google Slides** para la creación de diapositivas. Para la gestión de versiones de estos documentos de configuración, utilizamos la propia herramienta de versionado que la plataforma ofrece.

Google Drive ofrece varias ventajas. Entre ellas, el poder editar documentos de manera simultánea, permitiendo la colaboración entre distintos miembros, mantener un control de versiones seguro y confiable en la nube, acceder al historial de modificaciones de los documentos, y que es gratuito (solo se requiere tener una cuenta de Google).

11.2.2.3 Overleaf

Se utilizó Overleaf en la última etapa del proyecto: la etapa de Documentación. Esta plataforma de edición en línea para documentos LaTeX nos permitió crear, gestionar y compartir todos los documentos referentes a la Documentación de *wisello*. Al igual que Google Drive, nos permite la colaboración simultánea y provee herramientas de control de versiones sobre los documentos.

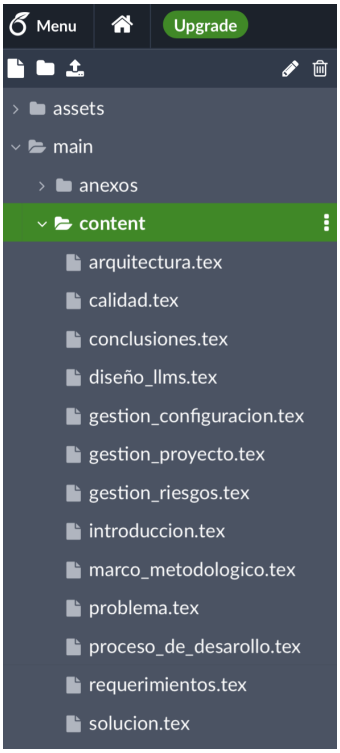


Figura 11.3: Documentación final en Overleaf

11.2.3 Lecciones Aprendidas

La elección de las herramientas adecuadas es fundamental en la gestión de la configuración del software, ya que está altamente acoplada a la eficiencia con la que el equipo puede trabajar y colaborar. Una buena selección de estas herramientas facilita la integración y colaboración efectiva, permitiendo un buen flujo de trabajo y una mayor productividad.

Por un lado, la centralización de los documentos en un único lugar, así como sus herramientas de control de versiones, aseguraron que la información fuera fácilmente accesible para todos, y que se pudiera verificar su versión, para confirmar que era la correcta.

Por el otro, las herramientas Git y GitHub facilitaron una colaboración eficiente en el desarrollo del código, permitiendo a los integrantes del equipo trabajar juntos de manera efectiva, independientemente de su ubicación física. Sus capacidades de control de versiones en un entorno de desarrollo simultáneo nos permitieron llevar un control e identificar cada versión fácilmente, o los nuevos cambios siendo agregados. Por ende, la enseñanza se centra en la importancia de adherirse a los acuerdos establecidos de gestión de configuración, para evitar enfrentar problemas relacionados con el control de versiones de los elementos de configuración del software dentro del proyecto.

12 Aseguramiento de la calidad

12.1 Testing Manual

Code reviews

Véase: 8.0.6

Product reviews

Realizamos Product Reviews con clientes en producción. En estas reuniones revisábamos el desempeño del producto en general, valor agregado a los clientes, cambios necesarios y posibles puntos de mejora. Con esta información, basada en el uso esperado y actual que le daban los usuarios a *wisello*, generamos cambios y nuevas features para seguir mejorando el producto.

Para más detalle de estas Reviews, ver 9.1.4.3.

12.2 Automatización del testing

Unit tests

La implementación de las pruebas unitarias se llevó a cabo con el fin de evaluar de manera aislada los distintos componentes de la aplicación. Estas pruebas se aplicaron exclusivamente a los módulos **langchain_custom** y **logic** del *backend*, seleccionados por el equipo como los elementos cruciales del sistema. Esta decisión se basó en la importancia de estos componentes para el funcionamiento general de la aplicación. Las pruebas unitarias se complementaron con otro tipo de pruebas para asegurar de manera integral la calidad y el rendimiento de cada componente.

Para garantizar la fiabilidad de cada método, se realizó una evaluación individualizada, permitiendo verificar su correcto funcionamiento. En el caso del módulo **logic**, el enfoque se centró en la validación de las funcionalidades dentro de cada unidad de negocio. Por ejemplo, si un método perteneciente a la unidad de negocio **shops** invocaba a otro método de la unidad **conversations**, se emplearon *mocks* para simular estas interacciones y evaluar el comportamiento esperado.

La incorporación de estas pruebas unitarias se efectuó paralelamente al desarrollo de nuevas funcionalidades, asegurando así una integración continua y eficaz. Además, se configuró la pipeline en GitHub para ejecutar automáticamente las pruebas tras cada actualización de código en las ramas del proyecto.

A continuación, se presenta un ejemplo de la ejecución local de las pruebas unitarias correspondientes al módulo **shops** ilustrando los resultados obtenidos.

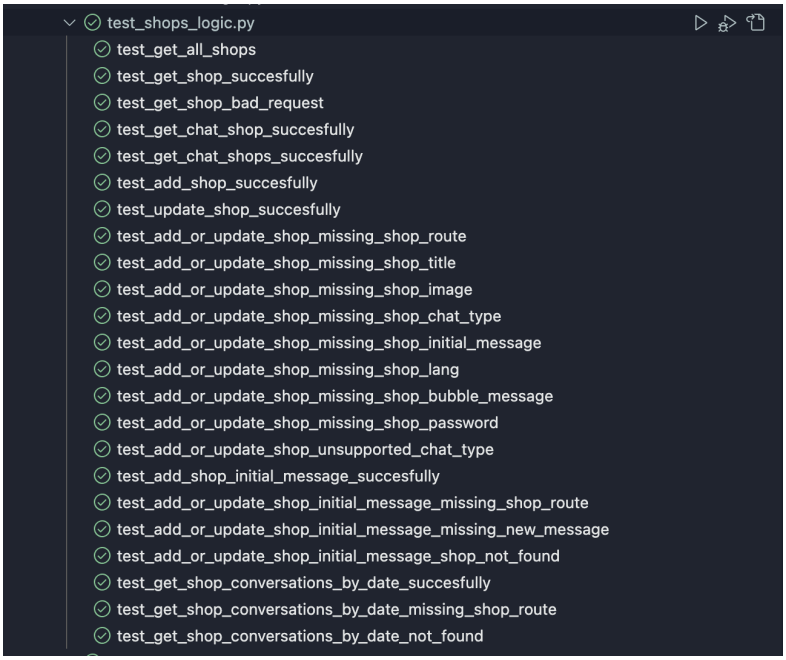


Figura 12.1: Ejecución de pruebas unitaria para el módulo shops

AI tests

El testeo automatizado de AI sigue la lógica de validar la exactitud de las salidas del modelo para determinadas entradas. Se define un conjunto predefinido de pares (pregunta/entrada, respuesta/salida) para cada cliente, y para la validación de las respuestas de utiliza un LLM más fuerte: GPT-4, que actúa como **maestro**, determinando si el test pasa o no, es decir, si la salida del modelo es la esperada para la entrada dada.

El hecho de trabajar con tecnologías nuevas hace que no estén estandarizadas ciertos aspectos de su desarrollo, como su testeo. Por ende, investigamos cómo hacerlo porque creímos fundamental incluir pruebas con el fin de asegurar la calidad de las respuestas.

En este contexto, hemos desarrollado **AI Tests** que tienen como objetivo evaluar las respuestas generadas por el sistema ante una serie de entradas fijas (`history`, `question`), y sus respuestas esperadas (`answer`). A continuación el ejemplo de estas para Casashub:

```
    {
        "history": [],
        "question": "Quiero información sobre casas en InfoCasas.",
        "answer": "No se responde información de InfoCasas, solo de CasasHub."
    },
    {
        "history": [],
        "question": "Quiero juguetes para mi hijo.",
        "answer": "No puedo ayudarte con eso."
    },
    {
        "history": ["Quiero un apartamento en carrasco.", "Cuántos dormitorios?"],
        "question": "2 dormitorios.",
        "answer": "Se recomienda una apartamento en carrasco."
    },
    {
        "history": [],
        "question": "Compra apartamento en punta del este",
        "answer": "Se recomienda una apartamento en punta del este"
    },
    {
        "history": [],
        "question": "Alquiler apartamento pocitos 3 dormitorios",
        "answer": "Se recomienda una apartamento en pocitos de 3 dormitorios"
    }

```

Se intenta que estos casos sean los más cruciales y abarcativos en el contexto de la tienda cliente, por lo cual se definen en participación con el cliente.

Para el desarrollo de estos tests, se utilizó una `QAModelChain` de `Langchain` con la siguiente prompt:

```
    """You are a teacher grading a quiz.
    You are given a question, the student's answer, and the true answer, and are asked to score the student answer as either CORRECT or INCORRECT.
    """

```

Ejecutamos *wisello* para cada caso de prueba. Luego, para comparar la respuesta generada por *wisello* con la respuesta esperada (`answer`), utilizamos una llamada a GPT-4 con la `prompt` recién expuesta. Esta llamada nos da el resultado de la prueba, pues se le pide a GPT-4 que califique la respuesta del *"alumno"* (en este caso, *wisello*) como correcta/incorrecta, teniendo la *true answer* como referencia: `score the student answer as either CORRECT or INCORRECT.` El uso de un LLM no vinculado directamente a nuestro sistema (GPT-4) permite minimizar el sesgo potencial en la comparación de respuestas.

El principal objetivo de la automatización de estas pruebas es verificar que el sistema continúe funcionando de acuerdo con las expectativas tras realizar modificaciones en la prompt (en relación a la problemática expuesta en 6.2.1, en que ciertos cambios del desarrollador optimizan las salidas para ciertas entradas, pero pueden perjudicar otras).

Tests de integración

Las pruebas de integración desarrolladas con `Cypress` nos permiten asegurar el funcionamiento adecuado de las interacciones clave del usuario con la aplicación. Implementamos pruebas para cubrir aspectos fundamentales de la experiencia del usuario, desde la carga inicial de la página hasta interacciones complejas que implican la comunicación con la API del backend.

Por ejemplo, se verifica que la página muestre correctamente mensajes iniciales y elementos visuales, se evalúan las respuestas del sistema a entradas del usuario como la evaluación de mensajes y el envío de datos al backend, y se comprueba la navegación intuitiva mediante botones de acción específicos.

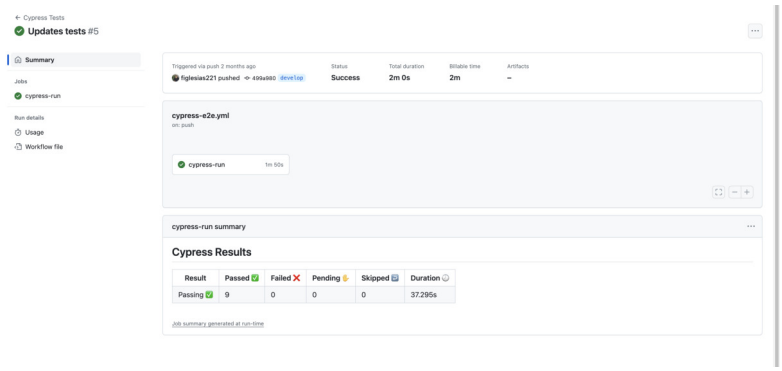


Figura 12.2: Automatización de Cypress tests en Github Actions

Para futuras iteraciones, se considera expandir estos tests para cubrir más escenarios de uso y variantes de interacción del usuario. Además, la inclusión de pruebas de rendimiento podría ofrecer insights valiosos sobre cómo optimizar la carga de la página y el manejo de datos en tiempo real.

12.3 Aprobación del cliente final en el contexto de LLMs

Las aplicaciones basadas en LLMs y en particular aquellas en formato chat, suponen desafíos en la extracción de información sobre el uso real. Para esto tenemos, en primer lugar, fuentes de información como el Like/Dislike y el feedback form desde las que obtenemos retroalimentación de forma directa.

A través de estos medios, el equipo era capaz de monitorear la retroalimentación provista por el usuario, y hacer los cambios pertinentes (hasta ahora, en forma manual), para mejorar continuamente la experiencia del usuario:

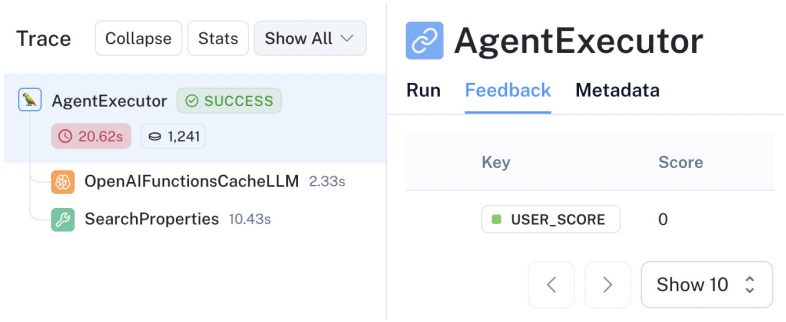


Figura 12.3: Feedback en Langsmith

La herramienta `Langsmith` nos permite asociar este feedback con la `run` específica del LLM, pudiendo así ver cuánto se tardo en responder, qué se respondió, cuáles fueron los parámetros con los que la función fue invocada, y demás. De esta forma, podemos identificar a qué se debe el puntaje otorgado por el usuario, los motivos detrás de este, y poder tomar acciones en base a el mismo.

Como alternativas, se podría utilizar métricas de *implicit feedback*, también para obtener información del uso. Brevemente, estas pueden ser, según las presentadas en [25]:

1. Tasa de conversión de *wisello*: comparándolo mediante un A/B test para evaluar su impacto.

2. Recurrencia de un usuario de la web: si el usuario que vuelve a entrar a la web vuelve a utilizar *wisello* o no, tomándolo como una retroalimentación positiva o negativa en cada caso.

3. Análisis sentimental del diálogo: identificar, utilizando la conversación, si los usuarios tienen una buena o una mala interacción.

Estas, entre otras, pueden ser adiciones valiosas para extraer mayor información sobre el uso real de *wisello*, y tener un panorama más claro del estado de aprobación del cliente final.

13 Conclusiones

A modo de conclusión, en esta sección, se hará una breve reseña sobre la experiencia del equipo con el proyecto, se revisarán los objetivos definidos inicialmente, se expondrán las lecciones aprendidas, y finalmente detallaremos nuestra proyección del producto a futuro.

Para el equipo, en lo personal, *wisello* significó una experiencia integral y sobre todo enriquecedora que, abarcando la totalidad de los conocimientos adquiridos durante la carrera, redondeó el final de este capítulo académico en nuestras vidas.

Su componente emprendedor, que nos obligó a salir al mercado a validar nuestra idea, a entrevistar y recibir consejos de expertos, a vender el producto, a recibir capital, a dar una charla frente a una audiencia, o a exponer nuestro producto en un evento, entre muchas otras cosas, nos permitió transitar caminos desconocidos y llegar a lugares que nunca imaginamos, expandir nuestras redes, y crecer tanto en lo personal como en lo profesional.

La responsabilidad de mantener un producto en producción, con usuarios reales utilizándolo, y estar mejorándolo continuamente tras un proceso sistemático de retroalimentación, nos permitió poner en práctica todos nuestros conocimientos de diseño, arquitectura de software, e ingeniería de software, mejorando tanto el producto como la gestión del equipo.

La arriesgada decisión de utilizar tecnologías emergentes y en auge, como los LLMs, aún en fase de exploración en muchos aspectos, nos desafiaron a innovar en áreas como el testing de estas aplicaciones, manteniéndonos en la vanguardia de la innovación: experimentando y aprendiendo.

Asimismo, la capacidad y recursos que tuvimos para resolver una problemática como la planteada, de principio a fin, colaborando con diversos sistemas y actores externos, para generar un cambio positivo en la sociedad, subraya y nos reafirma el valor de lo que hemos estudiado.

13.1 Objetivos

En lo que refiere a los objetivos **académicos**, todos los planteados fueron alcanzados de forma exitosa. Por un lado, en relación a **OA1**, se demostró, a lo largo del informe, la aplicación de herramientas definidas en el área de la ingeniería de software ágil:

- Se llevaron a cabo técnicas de recolección, validación y especificado de requerimientos acorde al área de ingeniería de requerimientos.
- Se utilizó Scrum como marco de gestión ágil, justificando esta elección así como las leves variaciones que se efectuaron sobre el marco.
- A lo largo de los capítulos de Diseño de aplicaciones basadas en Large Language Models (6) y Arquitectura (7) se evidenció la aplicación de patrones de diseño y tácticas de arquitectura en pos de cumplir con los atributos de calidad especificados para el proyecto.
- Se definió un proceso de desarrollo (8) completo, estructurado y respetado.
- Se llevó a cabo una gestión de riesgos (10) integral, en un proyecto con tecnologías muy nuevas, lo cual se entiende como riesgoso.
- Se definió un proceso de gestión de la configuración acorde (11).
- Se llevaron a cabo prácticas de aseguramiento de la calidad (12), que justamente nos permitieron lograr construir un producto de calidad.

Por otro lado, refiriéndonos a **OA2**, se adquirieron una inmensidad de conocimientos relacionados al estado del arte y buenas prácticas asociadas al desarrollo de aplicaciones basadas en LLMs, todas de las cuales fueron expuestas en el capítulo de Diseño de aplicaciones basadas en Large Language Models (6):

- Principios y estrategias de *prompt engineering* (6.1.
- Cómo tratamos el desafío de las alucinaciones (6.1.6).
- Patrones en aplicaciones impulsadas por LLMs (6.2).
- Bases de datos vectoriales (6.2.2.1).
- Testing (evaluación) de la respuesta (12.2)

Por ende, a través de este informe, se contribuye al cuerpo académico con estos conocimientos, cumpliendo con **OA3**.

A nivel del **proceso**, los objetivos también fueron cumplidos. Por un lado, en relación a **OPC1**, se aplicó un marco de gestión ágil: Scrum, para contar con la flexibilidad necesaria para adaptarse a los requerimientos cambiantes de un contexto cambiante. Por otro lado, respecto a **OPC2**, se llevó registro de las métricas de Velocidad, Sprint Burndown, Cantidad de Bugs, y Retrabajo (ver evidencia en 9.3), y estas fueron utilizadas a lo largo del proceso para mejorar la estimación del equipo, identificar áreas de mejora en las retrospectivas, y analizar el impacto de las acciones de aseguramiento de la calidad (para las métricas de Bugs y retrabajo). Finalmente, en lo que concierne a **OPC3**, también se llevaron a cabo Retrospectivas en cada iteración (9.1.4.4), a partir de las cuales el equipo pudo reflexionar sobre la iteración pasada y proponer mejoras para la siguiente, mejorando continuamente su proceso.

Por último, en lo que refiere a los objetivos del **producto**, estos también fueron satisfechos. En primer lugar, cumplimos con **OPD1** dado que logramos salir al mercado e integrarnos con dos clientes reales, lo cual supone una gran meta para el equipo detrás de *wisello*. Además, el producto está siendo utilizado por usuarios reales, día a día. En la última semana (a la fecha de hoy, 12/03/2024) fue utilizado **2645** veces:

Figura 13.1: Estadísticas de uso de *wisello* en la última semana

En segundo lugar, en referencia a **OPD2**, a través de las estadísticas de uso, el análisis de conversaciones y feedback del cliente, se pudo confirmar que, al utilizar *wisello*, el usuario percibe un valor agregado en el asesoramiento personalizado que este le provee, y la facilitación en la búsqueda. Finalmente, en relación a **OPD3**, la integración *seamless* fue lograda, permitiendo integrar *wisello* (en sus distintos formatos) a una tienda a través de una sola línea de código únicamente, sin incurrir en gastos extras de desarrollo web.

13.2 Lecciones aprendidas

A continuación, algunas de las lecciones más importantes que surgieron de la construcción de *wisello*:

- La importancia de escuchar al mercado, y centrar el desarrollo desde una perspectiva del usuario. El rumbo del producto hubiera sido otro muy distinto sin el involucramiento de personas idóneas y usuarios en el proceso.
- La importancia de tener un proceso sólido, bien definido, y respetado. Esto da orden al desarrollo, y marca las bases para la calidad resultante del producto.
- La importancia de reflexionar y mejorar el proceso iteración a iteración. Ningún proceso es perfecto, y menos lo será en una primera instancia. La transparencia con el equipo hace a un ambiente de colaboración y alineación de objetivos.
- La importancia de no conformarse. Lanzarse hacia lo desconocido, investigar nuevas maneras, experimentar, y ser creativos. Esto nos hizo distintos.

13.3 Proyección a futuro

En cuanto a la proyección a futuro de *wisello*, el equipo tiene en mente trabajar en los siguientes puntos.

Seguir mejorando el producto funcionalmente

Incorporar las funcionalidades que, por tema de tiempo, no se lograron desarrollar en el alcance actual. Algunos ejemplos: poder agregar un producto al carrito, mostrar métricas de clicks en productos, mostrar métricas de consultas más repetidas, entre otras.

Integración con más plataformas de e-commerce

Wisello planea integrarse con plataformas líderes en el mercado, como Shopify y Magento. Esta estrategia no solo ampliará nuestra base de usuarios potenciales sino que también reforzará nuestra posición como una solución *plug-and-play* en el e-commerce.

Reinforcement learning from Human Feedback

Aplicar técnicas de aprendizaje automático (Reinforcement learning from Human Feedback) con el objetivo de que el modelo auto-aprenda a partir del feedback provisto por usuarios, y perfeccione continuamente sus recomendaciones de productos, mejorando así la experiencia de usuario constantemente.

14 Bibliografía

[1] OpenAI, "Introducing chatgpt and whisper apis," https://openai.com/blog/introducing-chatgpt-and-whisper-apis, 2023, accessed: 2024-02-17.

[2] Universidad ORT Uruguay, "Ingeniería en sistemas," https://fi.ort.edu.uy/ingenieria-en-sistemas, 2024, accessed: 2024-02-20.

[3] Stanford University, "Xcs224u: Natural language understanding," https://online.stanford.edu/courses/xcs224u-natural-language-understanding, 2024, accessed: 2024-02-20.

[4] Universidad ORT Uruguay, "Centro de innovación y emprendimientos (cie)," https://cie.ort.edu.uy/, 2024, accessed: 2024-02-20.

[5] Ries, E., "The lean startup," 2011, accessed: 2024-03-18. [Online]. Available: https://ia600509.us.archive.org/7/items/TheLeanStartupErickRies/The%20Lean%20Startup%20-%20Erick%20Ries.pdf

[6] Shopify Staff, "ean startup model: Key principles and stages," 2023, accessed: 2024-03-18. [Online]. Available: https://www.shopify.com/blog/lean-startup-model#13

[7] Scrum.org, "The scrum team," https://www.scrum.org/resources/scrum-team, 2023, accessed: 2024-02-22.

[8] Greenpay S.A., "E-commerce crecerá 40 % en américa latina en 2023," 2023, accessed: 2024-01-19. [Online]. Available: https://greenpay.me/2022/12/13/ecommerce-crecera-en-america-latina-en-2023/

[9] Infonegocios [Infotecnología], "Cuántos carritos abandonados en el e-commerce (83[Online]. Available: https://infotecnologia.info/startup/cuantos-carritos-abandonados-en-el-e-commerce-83-no-termina-sus-compras

[10] CACE: Cámara Argentina de Comercio Electrónico, "Los argentinos y el e-commerce ¿cómo compramos y vendemos online?" 2021, accessed: 2024-01-19. [Online]. Available: https://cace.org.ar/wp-content/uploads/2022/06/cace-estudio-anual-2020-resumen.pdf

[11] Americas Market Intelligence, "Infographic: How uruguayans buy online," 2023, accessed: 2024-01-20. [Online]. Available: https://americasmi.com/insights/infographic-how-uruguayans-buy-online/

[12] ——, “Estado de los pagos digitales y el comercio electronico en latam,” 2023, accessed: 2024-01-20. [Online]. Available: https://americasmi.com/insights/pagos-digitales-comercio-electronico-en-latam-estadisticas-analisis/#:~:text=Tras%20un%20pico%20de%20crecimiento,Market%20Intelligence%20(PCMI)%20en%20su

[13] Agencia de Gobierno Electrónico y Sociedad de la Información y del Conocimiento, “Eutic 2022: continúa creciendo el acceso a internet en los hogares del país,” 2023, accessed: 2024-01-20. [Online]. Available: https://www.gub.uy/agencia-gobierno-electronico-sociedad-informacion-conocimiento/comunicacion/noticias/eutic-2022-continua-creciendo-acceso-internet-hogares-del-pais#:~:text=Respecto%20al%20uso%20de%20Internet,en%20la%20EUTIC%20desde%202010

[14] Consultora CIFRA, “Tendencias del consumo digital,” 2022, accessed: 2024-01-20. [Online]. Available: https://drive.google.com/file/d/1I8raxEtz-1eQm2diErwRnIj5CWzpQWry/view

[15] FACTUM, “Indicadores del comercio electrónico en las empresas uruguayas,” 2022, accessed: 2024-01-20. [Online]. Available: https://drive.google.com/file/d/1tQabnmOfw06K-bAmRKvQyOJM4g6JalRj/view

[16] CEDU: Cámara de la Economía Digital del Uruguay, “Ecommerce: un sector en crecimiento y de generación de oportunidades en uruguay,” 2021, accessed: 2024-01-20. [Online]. Available: https://www.cedu.org.uy/ecommerce-un-sector-en-crecimiento-y-de-generacion-de-oportunidades-en-uruguay/

[17] Lotitto E., Díaz de Astarloa B., “The landscape of b2c e-commerce marketplaces in latin america and the caribbean,” 2023, accessed: 2024-01-27. [Online]. Available: https://repositorio.cepal.org/server/api/core/bitstreams/dad20356-3266-4a57-b699-da39c0898122/content

[18] Khatib N., “How e-commerce is powering the payments revolution,” 2022, accessed: 2024-01-27. [Online]. Available: https://www.gbm.hsbc.com/en-gb/insights/innovation/how-e-commerce-is-powering-the-payments-revolution

[19] Gatta V., Edoardo M., Le Pira M., “Ecommerce and urban logistics: trends, challenges, and opportunities,” 2023, accessed: 2024-01-27. [Online]. Available: https://books.google.com.uy/books?hl=en&lr=&id=8gPEEAAAQBAJ&oi=fnd&pg=PA422&dq=trends+in+ecommerce+logistics&ots=hGXVHSNtQk&sig=CBuakSUcWATHXbmIN2ma85xrTr8#v=onepage&q=trends%20in%20ecommerce%20logistics&f=false

[20] Bisceglia, P., "New technology developments stimulate e-commerce investment and growth," 2021, accessed: 2024-01-27. [Online]. Available: https://www.bdo.global/en-gb/blogs/tech-media-watch-blog/new-technology-developments-stimulate-e-commerce-investment-and-growth

[21] Fonseca, M., "The latam tech report 2023: the future of 7 startup sectors," 2023, accessed: 2024-01-27. [Online]. Available: https://www.latitud.com/blog/the-latam-tech-report-2023-the-future-of-7-startup-sectors

[22] CB Insights, "Our retail tech report takes a data-driven look at global retail technology investment trends and top deals, highlighting areas of interest across the retail landscape." 2021, accessed: 2024-01-27. [Online]. Available: https://www.cbinsights.com/research/report/retail-tech-trends-q1-2021/

[23] Gupta S., Kushwaha P.S., Badhera U., Chatterjee P., Santibanez Gonzalez E., "Identification of benefits, challenges, and pathways in e-commerce industries: An integrated two-phase decision-making model," 2023, accessed: 2024-01-30. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666412723000156

[24] Fenicio eCommerce, "Fenicio talks 2023 — cómo la ia está redefiniendo la forma de vender y comprar online," 2023, accessed: 2024-01-30. [Online]. Available: https://www.youtube.com/watch?v=F-cPJl5Ky5g&t=37s

[25] Z. Yan, "Patterns for building llm-based systems products," *eugeneyan.com*, Jul 2023, accessed: 2024-01-20. [Online]. Available: https://eugeneyan.com/writing/llm-patterns/

[26] L. Martin, "auto-evaluator: Evaluation tool for llm qa chains," https://github.com/rlancemartin/auto-evaluator, 2024, accessed: 2024-02-20.

[27] OpenAI, "New and improved embedding model," https://openai.com/blog/new-and-improved-embedding-model, 2024, accessed: 2024-02-20.

[28] Pinecone Systems, Inc., "The vector database to build knowledgeable ai," https://www.pinecone.io/, 2024, accessed: 2024-02-20.

[29] ——, "Chunking strategies for llm applications," https://www.pinecone.io/learn/chunking-strategies/, Jun 2023, accessed: 2024-02-20.

[30] LangChain, Inc., "Chains," https://python.langchain.com/docs/modules/chains, 2024, accessed: 2024-02-20.

[31] OpenAI, "Function calling," https://platform.openai.com/docs/guides/function-calling, 2024, accessed: 2024-02-20.

[32] ——, "When to use fine-tuning," https://platform.openai.com/docs/guides/fine-tuning/when-to-use-fine-tuning, 2024, accessed: 2024-02-20.

[33] Zilliz Inc., "Gptcache: A library for creating semantic cache for llm queries," https://gptcache.readthedocs.io/en/latest/, 2023, accessed: 2024-02-20.

[34] Microsoft, "Azure openai service frequently asked questions," 2024, accessed: 2024-03-06. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/faq

[35] Jaehoshin, "Comparing azure openai and openai: A deep dive," 2023, accessed: 2024-03-06. [Online]. Available: https://medium.com/@jaehoshin62/comparing-azure-openai-and-openai-a-deep-dive-dd18c642e976

[36] Vercel Inc, "Service level agreement for enterprise plans," 2022, accessed: 2024-02-12. [Online]. Available: https://vercel.com/legal/sla

[37] Microsoft, "Service level agreement for microsoft online services," 2024, accessed: 2024-03-11. [Online]. Available: https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services

[38] Luashchuk, A., "Towardsdatascience: Why i think python is perfect for machine learning and artificial intelligence," 2019, accessed: 2024-02-17. [Online]. Available: https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6

[39] Raschka S., Patterson J., Nolet C, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," 2020, accessed: 2024-02-17. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/193

[40] Python Software Foundation, "Python: A programming language that lets you work quickly and integrate systems more effectively," 2023, accessed: 2024-02-08. [Online]. Available: https://www.python.org/

[41] Sebastián Ramírez, "Fastapi: A modern, fast (high-performance), web framework for building apis with python 3.6+ based on standard python type hints." 2023, accessed: 2024-02-08. [Online]. Available: https://fastapi.tiangolo.com/

[42] Roumeliotis K, Tselikas N, Nasiopoulos D, "Llms in e-commerce: A comparative analysis of gpt and llama models in product review evaluation," 2024, accessed: 2024-02-17. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2949719124000049

[43] Big Blue Data Academy, "Vector database: Definition, benefits & applications," December 2023, accessed: 2024-02-08. [Online]. Available: https://bigblue.academy/en/vector-database

[44] Pinecone Systems, Inc., "Pinecone: The vector database to build knowledgeable ai," 2023, accessed: 2024-02-08. [Online]. Available: https://www.pinecone.io/

[45] MongoDB, Inc., "¿qué es mongodb?" 2024, accessed: 2024-02-17. [Online]. Available: https://www.mongodb.com/es/what-is-mongodb

[46] AWS, "Redis: Fast, open source in-memory data store for use as a database, cache, message broker, and queue." 2024, accessed: 2024-02-17. [Online]. Available: https://aws.amazon.com/redis/

[47] Vercel, Inc., "Next.js: The react framework for production," 2023, accessed: 2024-02-08. [Online]. Available: https://nextjs.org/

[48] Microsoft Corporation, "Microsoft azure: Cloud computing services," 2023, accessed: 2024-02-08. [Online]. Available: https://azure.microsoft.com/

[49] Vercel, Inc., "Vercel: Develop. preview. ship. for the best frontend teams." 2023, accessed: 2024-02-08. [Online]. Available: https://vercel.com/

[50] Metro Retro, "Drop / add / keep / improve retrospective template," Metro Retro Website, 2023, accessed: 2024-02-08. [Online]. Available: https://metroretro.io/templates/drop-add-keep-improve-retrospective

[51] Caballero S. D., Kuna H.D., "Análisis y gestión de riesgo en proyectos software," 2018, accessed: 2024-02-18. [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/67916/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y#:~:text=Los%20riesgos%20tcnicos%20del%20software,potenciales%20de%20riesgos%20de%20software.

[52] OpenAI, "Openai status," 2024, accessed: 2024-03-14. [Online]. Available: https://status.openai.com/uptime

[53] Python Org, "Pep 8 – style guide for python code," 2001, accessed: 2024-02-20. [Online]. Available: https://peps.python.org/pep-0008/

15 Anexos

15.1 Herramientas de validación

15.1.1 Mapa de competencia

	Integración a páginas web	Métricas / reportes de uso	Universo de respuesta	Personalizado	Disponibilidad 24/7	Precio
Chatbots tradicionales	Sí	Sí	Limitado	No	Si	Para una compañía grande, ronda entre los 240 y 420 usd al mes (Cliengo y ChatBot).
Recomendación de productos	Sí	Sí	Ilimitado	Sí (dentro de un rango restringido)	Sí	-
Consultas por WhatsApp	Sí	No, es información que se pierde.	Limitado	Sí	No	Costo del tiempo dedicado por recursos humanos.
Google Search	Sí	Sí	Limitado (a lo que existe)	Sí (dentro de un rango restringido)	Sí	Gratis
Amazon como motor de búsqueda	Sí	No	Ilimitado	Sí (dentro de un rango restringido)	Sí	Gratis
Consultas por email	Sí	No, es información que se pierde.	Ilimitado	Sí	No	Costo del tiempo dedicado por recursos humanos.

Figura 15.1: Mapa de competencia

En rojo se marcan aquellas características con las cuales el método de asesoramiento falla en cumplir.

* Es importante aclarar que al momento de realizar este mapa de competencia, aún no existían alternativas self-service de chatbots impulsados por IA generativa (como por ejemplo, https://www.chatbase.co).

15.1.2 Tabla de Hipótesis

Hipótesis de problema	Hipótesis de cliente	Hipótesis de solución
La falta de información y asesoramiento acerca de los productos web hace que el cliente no esté convencido de su compra, imponiendo un riesgo de que luego la abandone.	Clientes de compra online.	Ofrecer asesoramiento sincronico vía Whatsapp para evitar que el cliente abandone la compra, brindando información extra sobre los productos que desee.
La falta de asesoramiento durante y previo al proceso de compra impacta negativamente sobre las ventas del negocio, teniendo como resultado "carritos abandonados" con compras no concretadas.	Tiendas e-commerce dedicadas a rubros relacionados a: cuidado personal, belleza, higiene, suplementos deportivos. Productos que son muy personales de las características de uno.	Ofrecer asesoramiento sincronico vía Whatsapp, en donde el vendedor pueda acceder al historial de compra del usuario y analizar su situación para ayudarlo en su compra.
El asesoramiento vía Whatsapp o email, para el cual hay que dedicar recursos humanos, es muy costoso para la empresa. Además insume muchos tiempos muertos de espera (tanto para cliente como tienda), y no está siempre disponible.	Tiendas e-commerce medianas y grandes. Medianas: sin muchos recursos para dedicar al canal telefónico de ventas, pero con alto tráfico y por ende una necesidad de asesoramiento masivo. Grandes: imposible escalar de forma rentable el canal telefónico con tantos clientes y aun así dar un buen servicio.	Implementar un método de asesoramiento con Inteligencia Artificial que esté siempre disponible y no necesite de un vendedor humano, haciéndolo más rentable. Este "vendedor" tiene toda la información sobre todos los productos, y responde al instante de ser consultado. Vendedor automatizado sin intervención humana.
El asesoramiento vía Chatbots es limitado; hay un libreto del cual el asesoramiento no puede divergir. Esto hace que muchas veces no cumpla su propósito de asesorar y como consecuencia el usuario no se sienta incentivado a comprar.	Clientes de compra online.	Ofrecer asesoramiento con un universo de respuesta no limitado, sin respuestas fijas y personalizado para que el usuario se vea incentivado a comprar. Esto puede ser con AI, Whatsapp o vía email.
Hace falta una solución tecnológica para construir un asesoramiento económico para la tienda, que esté siempre disponible, sea eficaz a ojos del usuario y aumente su convicción de compra.	Tiendas e-commerce con alto tráfico a todas horas; alto volumen de consultas/dudas.	Implementar un vendedor automatizado con Inteligencia Artificial que actúe como un agente conversacional ofreciendo recomendaciones de producto en base a las necesidades del cliente, tanto las que este expresa como las implícitas en su historial de compra.
El cliente online se siente desorientado durante la compra lo cual lo hace no estar convencido de esta; necesita de un vendedor que lo ayude. Le falta información.	Clientes de compra online.	Agente conversacional que ofrece asesoramiento en base a las necesidades expresadas por el cliente, o asesoramiento vía Whattsapp.
El cliente online tiene que esperar a que el vendedor le responda por Whatsapp para decidir finalmente si está comprando lo que realmente desea comprar.	Clientes de compra online.	Agente conversacional que responda al instante de ser consultado con información de valor. Asesoramiento sincrónico.
El cliente online siente que el Chatbot no aplica a sus dudas, nunca llega a nada.	Clientes de compra online.	Agente conversacional con respuestas personalizadas, o asesoramiento vía Whattsapp.
El cliente online necesita un vendedor que esté siempre disponible y responda rápido, que lo entienda en sus necesidades, que le dé información y lo ayude (quizás con recomendaciones)	Clientes de compra online.	Agente conversacional disponible 24/7, que tenga información del cliente para personalizar sus respuestas y que sea capaz de hacer recomendaciones sobre el catálogo de la tienda.

Figura 15.2: Tabla de Hipótesis

15.1.3 Mapa de empatía

¿Qué piensa y siente?	¿Qué ve?
- Falta de conocimiento de qué es lo que quiere/necesita comprar y lo que se ofrece - Falta de ayuda/recomendaciones - Recibir conocimiento del producto (no tener qué investigar por su cuenta) - Dificultad para conocer toda la oferta de productos, y el detalle de cada uno - Indecisión	- Páginas de venta online con demasiadas opciones y sin posibilidad de consultar o con consultas asincrónicas o consultas con chatbots tradicionales y poco efectivos. - Páginas de venta online que lo único que tienen es un buscador de productos. La voluntad de compra está del lado del comprador. - Catálogos de productos en redes sociales como Instagram.
¿Qué oye?	**¿Qué dice y hace?**
- Se oyen de promociones o de conocidos que acceden a promociones no muy visibles que pueden estarse perdiendo ellos mismos. - Oyen experiencias muy variadas (positivas y negativas) de la experiencia de compra en e commerces, generando perspectivas conflictivas de la experiencia de compra online.	- Compra online - Busca productos. - Investiga y averigua, de la gama de productos que quiere, cuál le conviene comprar, o cuál se ajusta más a sus necesidades
¿Qué esfuerzos, miedos, frustraciones y obstáculos encuentra la persona?	**¿Qué le motiva? Deseos, necesidades, medida del éxito, obstáculos.**
- ¿Dónde está el vendedor para ayudarme? - ¿Cómo sé que lo que busco es adecuado? - ¿Cómo satisfacen mis necesidades los productos ofrecidos?	- Le motiva comprar online porque ahorra tiempo de su vida sin tener que ir a la tienda física - Le motiva comprar online porque hay más oferta, o mejor oferta - Le motiva comprar online porque no se siente presionado por vendedores persona

Figura 15.3: Mapa de Empatía

15.1.4 User Journey Map

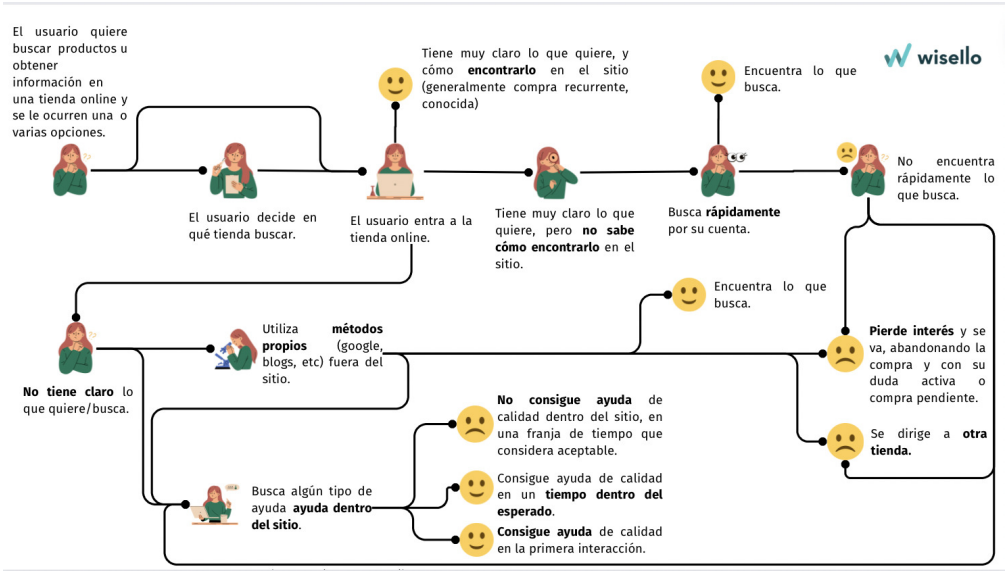
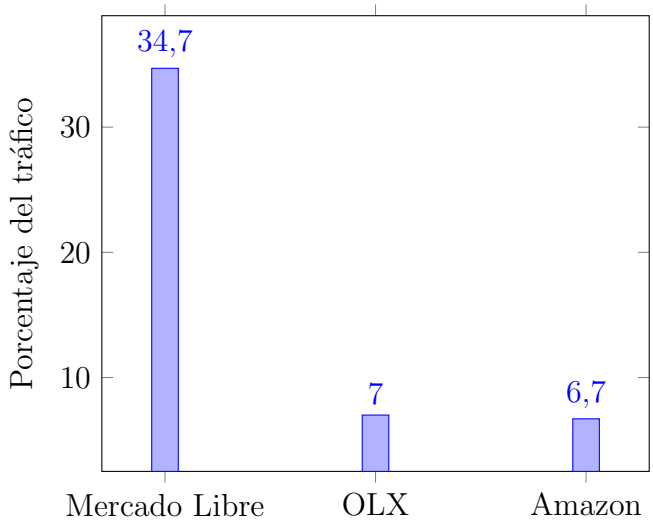


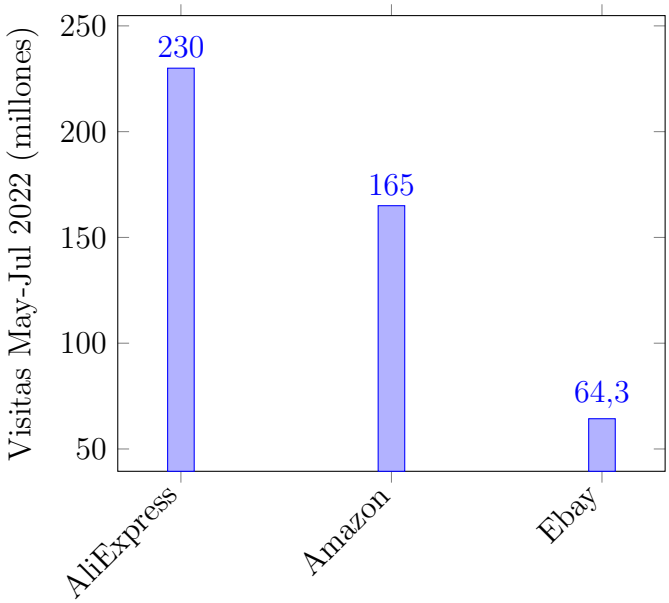
Figura 15.4: User Journey Map

15.2 Fuerzas dominantes del mercado

Principales empresas de e-commerce en Latam, y su porcentaje del tráfico (ECLAC, 2023):



Principales empresas de e-commerce en el mundo, y su número de visitas entre Mayo-Julio 2022 (ECLAC, 2023):



Parecería ser, por tanto, que los *grandes jugadores* a primera vista son los *marketplaces*. Sin embargo, las tendencias con los marketplaces muestran que, si quitamos el tráfico generado por la pandemia, entre 2019 y 2021, solamente hubo un 6.4 % que crecieron en tráfico, y este número desciende levemente en el 2021 (ECLAC, 2023). Esto nos hace concluir que, si bien los *gigantes* del e-commerce son marketplaces, no muchos logran crecer bajo este modelo.

15.3 Prueba de carga

15.3.1 Load test

```python
import requests
from concurrent.futures import ThreadPoolExecutor, as_completed

URL = 'https://wisello-backend.azurewebsites.net/completion/casashub'
MESSAGES = [
    'Hola, estoy buscando una casa en Carrasco',
    'Hola, quiero información acerca de CasasHub',
    'Hola',
    'Hola, quiero un apartamento en pocitos de 3 cuartos y 2 baños',
    'Hola, quiero una chacra en Rocha',
    'Hola quiero un apartamento en Jose Ignacio con vista a la playa',
    'Hola quiero un apartamento en el centro que sea céntrico y cerca de todo',
    'Hola quiero una oportunidad de inversión en pozo',
    'Hola quiero alquilar un local comercial en carrasco',
    'Hola quiero ver qué proyectos hay disponibles en Montevideo',
    'Quiero una casa con losa radiante, parrillero y jardín',
    'Casa esquinera con vista al mar',
    'Monoambiente con mucha iluminación',
    'Chacra cerca de punta del este, en zona jose ignacio, aprox 1 millon',
    'Quiero alquilar algo para mi y mi pareja, que sea pet-friendly y no supere
     los 1000usd mensuales'

]

def send_request(message):
    body = {
        'conversation_id': 'sample_id',
        'messages': message
    }
    response = requests.post(URL, json=body)
    return response.status_code, response.text

def load_test_concurrent():
    with ThreadPoolExecutor(max_workers=100) as executor:
        futures = [executor.submit(send_request, message) for message in
    MESSAGES for _ in range(7)]

    for future in as_completed(futures):
        status_code, response_text = future.result()
        print(f"Respuesta recibida: {status_code}, {response_text}")

if __name__ == "__main__":
    load_test_concurrent()
```

15.3.2 Script de análisis de resultados *time to first token*

```
import os
import datetime
from datetime import timedelta

from langsmith import Client
os.environ["LANGCHAIN_ENDPOINT"] = "https://api.smith.langchain.com"
os.environ["LANGCHAIN_TRACING_V2"] = "true"
os.environ["LANGCHAIN_API_KEY"] = <LANGSMITH-API-KEY>

client = Client()

agent_executor_runs = client.list_runs(project_name="wisello", start_time=
    datetime.datetime.now() - timedelta(hours=1), run_type="chain")

total_latency_agent_executors = 0
total_agent_executor_runs = 0
for run in agent_executor_runs:
    total_latency_agent_executors += (run.end_time - run.start_time).
    total_seconds()
    total_agent_executor_runs += 1

average_latency_agent_executors = total_latency_agent_executors /
    total_agent_executor_runs
print("Average latency of agent executor runs:",
    average_latency_agent_executors)

llm_runs = client.list_runs(project_name="wisello", start_time=datetime.
    datetime.now() - timedelta(hours=1), run_type="llm")

total_latency_llms = 0
total_first_token_times = 0
total_llm_runs = 0
for run in llm_runs:
    if run.first_token_time:
        total_latency_llms += (run.end_time - run.start_time).total_seconds()
        total_first_token_times += (run.first_token_time - run.start_time).
    total_seconds()
        total_llm_runs += 1

average_latency_llms = total_latency_llms / total_llm_runs
print("Average latency of llm-only runs:", average_latency_llms)

average_time_to_first_token_llm = total_first_token_times / total_llm_runs
print("Average time to first token in llm-only runs:",
    average_time_to_first_token_llm)
```

```
39
40 print("Average time to first token:", average_latency_agent_executors -
       average_latency_llms + average_time_to_first_token_llm)
```

Los resultados del `script` fueron los siguientes:

```
andresjuan@Andress-MacBook-Pro-2 load_test % python3 get_langsmith.py
Average latency of agent executor runs: 7.002188394230769
Average latency of llm-only runs: 2.257957777142857
Average time to first token in llm-only runs: 0.5603858228571427
Average time to first token: 5.304616439945055
```

Figura 15.5: Resultados del *load test*

La métrica que tomamos es la última: *average time to first token*, con valor `5.3s`.

La razón por la cual se deben realizar los cálculos exhibidos es porque en `Langsmith` cada *request* se registra en dos partes:

- La primera bajo el nombre `AgentExecutor`, donde se realiza la `function_call` y se corre la función (no tiene *time to first token* porque no genera *tokens* directamente ya que no se responde nada). Su latencia expuesta corresponde a la latencia de **toda** la request (incluyendo la de la siguiente parte).
- La segunda donde se responde al usuario (sí tiene *time to first token*).

Name	Input	Start Time	Latency	Dataset	Annotation Queue	Tokens	Cost	First Token (ms)	Tags
OpenAIFunctionsCachel	system: Tu nombre es ...	08/02/2024, 12:37:10	0.14s			1,317	$0.004067	137 ms	
AgentExecutor	Alquiler apartamento ...	08/02/2024, 12:37:07	3.61s			1,248	$0.003793	N/A	

Figura 15.6: Ejemplo de la división de la *request* en Langsmith

Por ende, para llegar a la métrica deseada *time to first token* promedio se tiene que realizar la siguiente cuenta: (la latencia promedio de la llamada completa) - (la latencia promedio de la segunda llamada) + (el *time to first token* promedio de la segunda llamada). Esto nos da el *time to first token* promedio de toda la llamada.

15.4 Variables de entorno: producción

15.4.1 backend

```
PORT=8080
AZURE_EMBEDDING_OPENAI_DEPLOYMENT="ada"
AZURE_GPT_OPENAI_DEPLOYMENT_NAME="wisello"
AZURE_OPENAI_API_KEY=<AZURE-OPENAI-API-KEY>
AZURE_OPENAI_API_VERSION="2024-02-15-preview"
AZURE_OPENAI_ENDPOINT="https://wisello-azure-openai.openai.azure.com/"
PINECONE_API_KEY=<PINECONE-API-KEY>
PINECONE_ENVIRONMENT="us-west4-gcp"
MONGO_CONNECTION_STRING=<MONGO-CONN-STRING>
SOURCE_DOCS_COLLECTION="source_docs_prod"
REDIS_HOST="wisello-cache.redis.cache.windows.net"
REDIS_PORT="6380"
REDIS_PASSWORD=<REDIS-PWD>
MONGO_SHOPS_COLLECTION=shops_prod
AZURE_INSTRUMENTATION_KEY=<INSTRUMENTATION-KEY>
MONGO_CONVERSATION_COLLECTION="conversations_prod"
AZURE_COMMUNICATION_SERVICE_CONN_STRING=<EMAIL-ACCESS-KEY>
JWT_SECRET_KEY=<SECRET-KEY>
ENV="PROD"
APPLICATION_INSIGHTS_ID="b211a00b-e1a6-4a23-9753-16449f7d4fb0"
APPLICATION_INSIGHTS_KEY=<INSIGHTS-KEY>
LANGCHAIN_API_KEY=<LANGCHAIN-API-KEY>
```

```
LANGCHAIN_ENDPOINT="https://api.smith.langchain.com"
LANGCHAIN_PROJECT="wisello"
LANGCHAIN_TRACING_V2=true
NEW_RELIC_FILE=newrelic-prod.ini
```

15.4.2 bubble-chat-server

```
PORT=8080
```

15.4.3 chat-frontend

```
NEXT_PUBLIC_SERVER_URL=https://wisello-backend.azurewebsites.net
NEXT_PUBLIC_GOOGLE_ANALYTICS=G-370DT8Q6W9
```

15.4.4 web-platform

```
NEXT_PUBLIC_API_URL=https://wisello-backend.azurewebsites.net
NEXT_PUBLIC_GOOGLE_CLIENT_SECRET=<GOOGLE-CLIENT-SECRET>
NEXT_PUBLIC_GOOGLE_CLIENT_ID=<GOOGLE-CLIENT-id>
NEXT_AUTH_SECRET=<AUTH_SECRET>
```

15.5 Evidencia *develop*: APM y logs

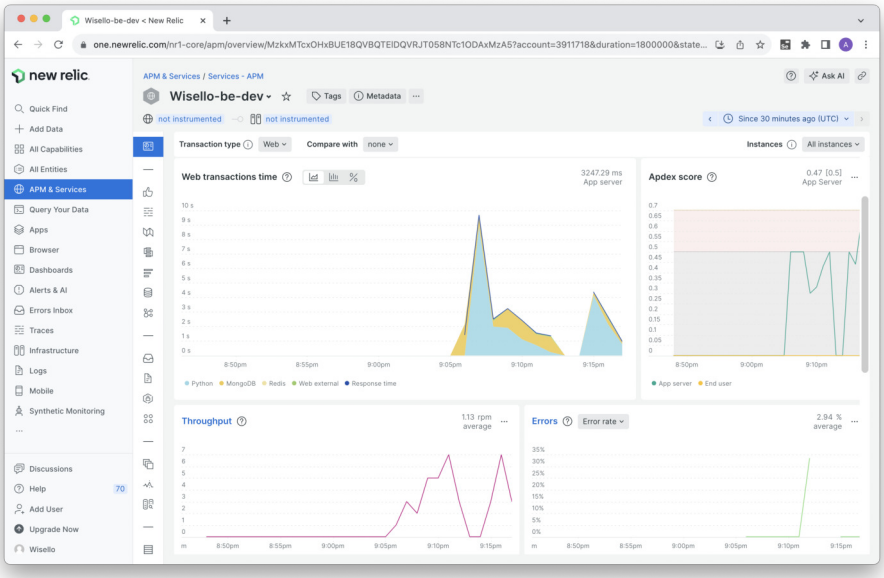


Figura 15.7: APM para el ambiente *develop*

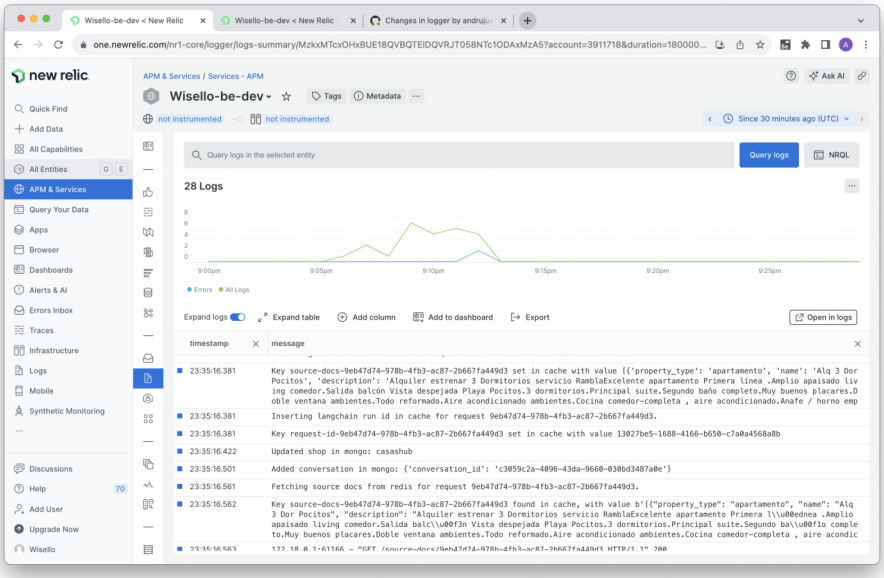


Figura 15.8: Logs para el ambiente *develop*

15.6 Sesiones de testing exploratorio

División:

- Diseño de pruebas: 15 %
- Ejecución de pruebas: 70 %
- Investigación y Reporte de Defectos: 15 %.

15.6.1 Primera sesión

- **Misión**: Validar que el tiempo de aprendizaje del chat es de máximo 3 minutos.
- **Fecha**: 18/2/2024.
- **Tester**: Ana Inés Montaldo.
- **Supervisor**: Andrés Juan.
- **Estructura**: Corta (20' o menos).
- **Entorno**: Microsoft Edge v121.0.2277.128 Windows 11 Home.

Para esta sesión se planificaron dos pruebas:

- Una búsqueda de una chacra en la cercanías de José Ignacio, para los cuales sí hay propiedades disponibles.
- Un campo en Colonia, para los cuales no hay propiedades disponibles, esperando así que se utilice el **Buscar por mi**.

15.6.1.1 Notas de las pruebas

Prueba 1: búsqueda de chacra en José Ignacio. Inicio 12.44 pm.

El usuario escribe la pregunta *"Quiero una chacra en José Ignacio"* para la prueba en cuestión y presiona `Enter`.

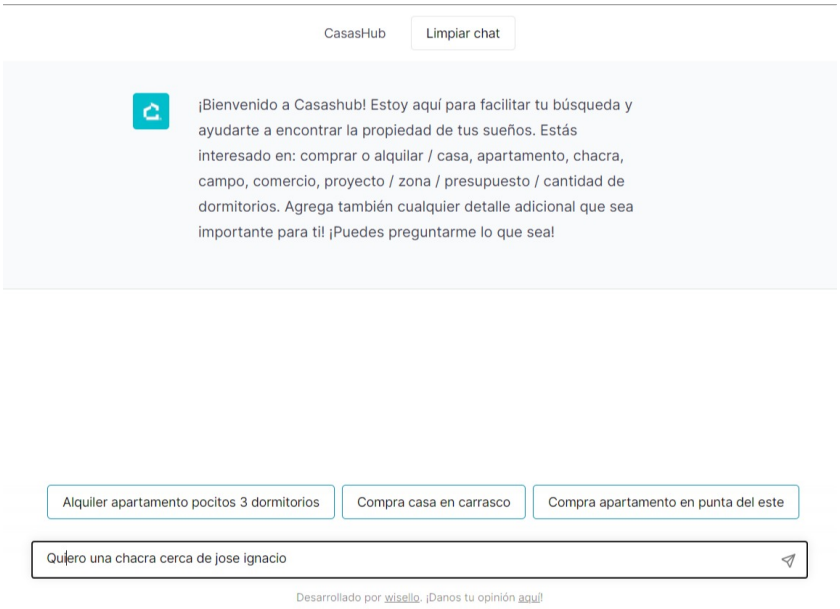


Figura 15.9: Pregunta inicial al chat

Se recibe la respuesta, recomendando una chacra en las cercanias de José Ignacio, en el barrio Las Garzas (a 15' de José Ignacio), e incluyendo el producto recomendado con la redirección a `casashub.uy`.

Figura 15.10: Respuesta del chat la pregunta inicial

El usuario **ingresa** a la propiedad, clickeando en el producto recomendado, y echando un vistazo a la misma a través del portal.

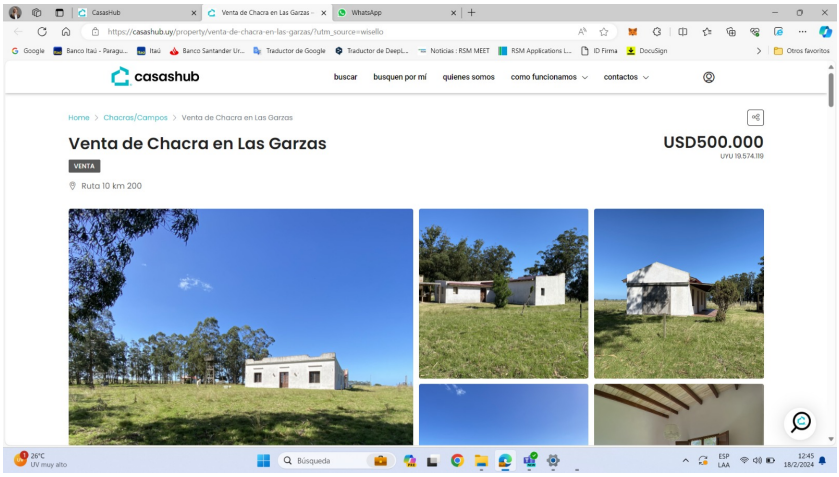


Figura 15.11: Acceso al portal a través del chat

El usuario, por su propia voluntad, vuelve a la pestaña del chat, y consulta si la propiedad *"está sobre la ruta? qué es la garzas?"*. En este punto (registradas las **12.46am**) se puede considerar que el usuario ha aprendido a interactuar con el mismo, en un tiempo de 2min.

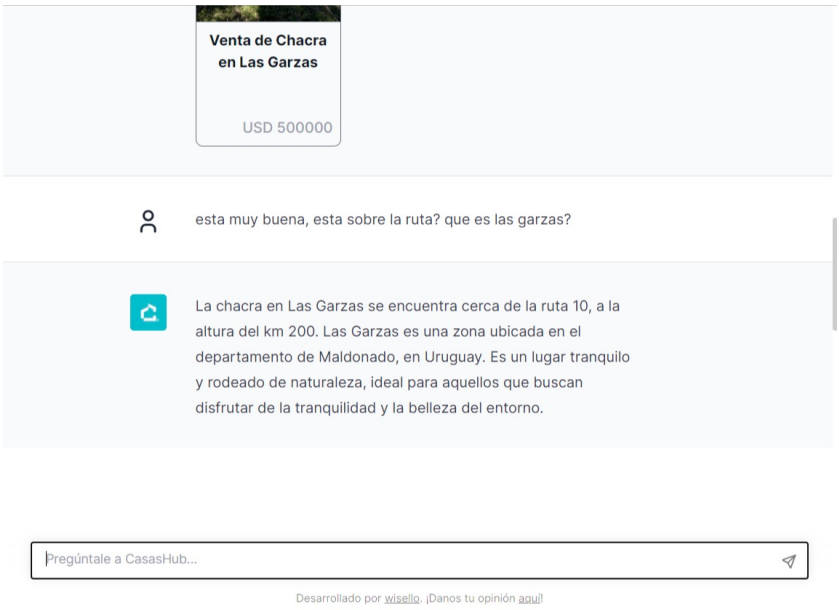


Figura 15.12: Re-pregunta del usuario al chat

Prueba 2: búsqueda de campo en Colonia. Inicio 13.09 pm.

El usuario escribe la pregunta *Busco comprar un campo en colonia* para la prueba, presiona `Enter`, y recibe la respuesta esperada de que no hay campos en la región que busca:

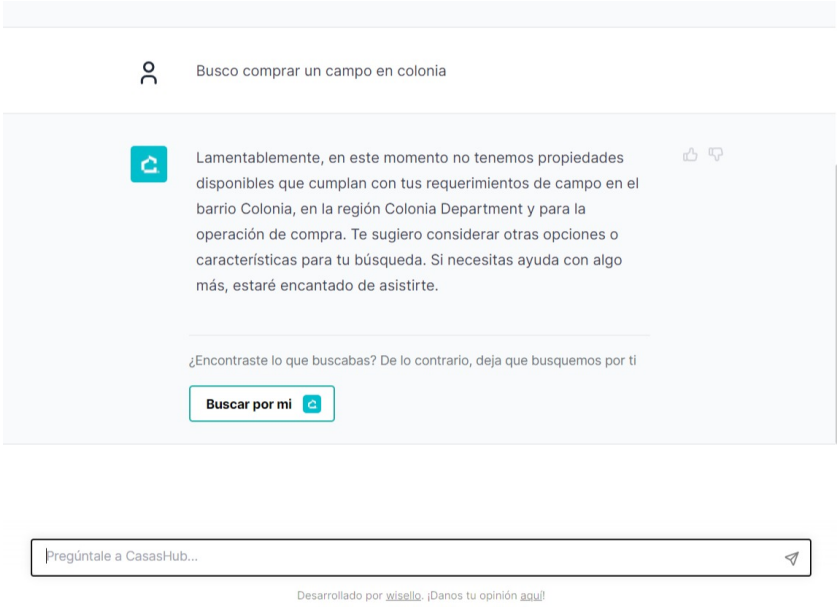


Figura 15.13: Pregunta inicial al chat

Recibida esta respuesta, el usuario interpeta correctamente que debe utilizar la funcionalidad de **Buscar por mi** dado que no encontró lo que buscaba, y aprieta el botón de **Buscar por mi**.

El usuario, en esta ocasión, reconoce que el formulario carga sus datos precargados en su computadora (teléfono, e-mail), lo cual aporta a una buena experiencia de usuario.

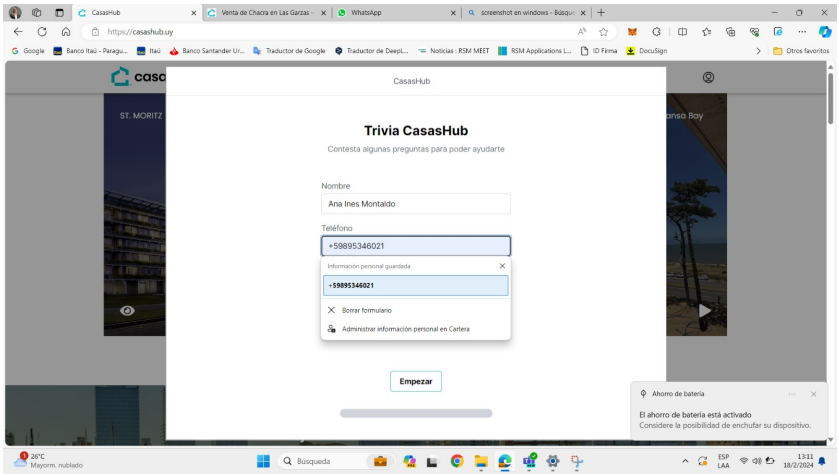


Figura 15.14: Formulario precarga los datos del sistema

Además, el usuario ve que se aportan mensajes de error apropiados y explicativos, que entiende y sabe corregir. Por ejemplo, el usuario tiene precargado su número como `+598` en el sistema, por lo cual se lo carga así al formulario, pero *wisello* le explica que solo se aceptan números. Lo entiende y le quita el `+`, sin mayores complicaciones.

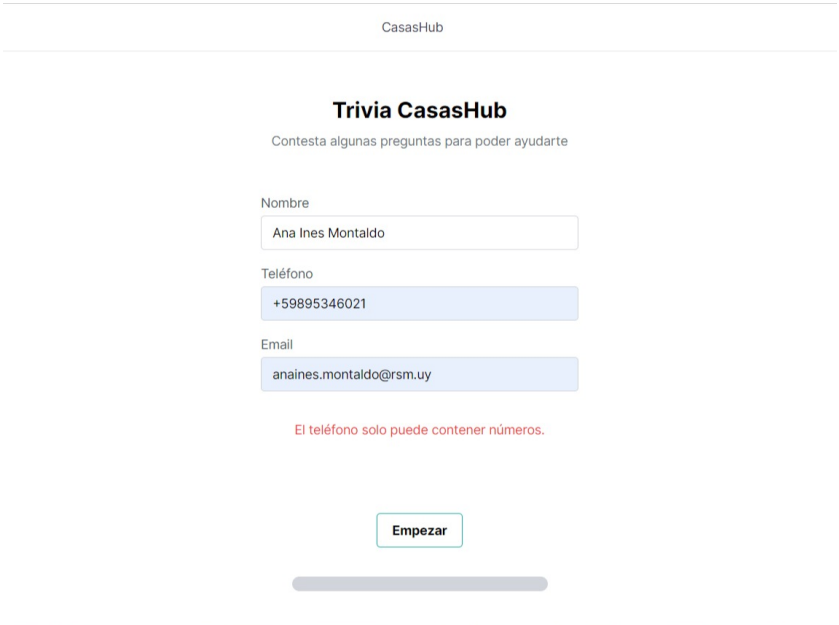


Figura 15.15: Mensajes de error apropiados

Un defecto que remarcó el usuario durante esta prueba es la falta de un botón para volver para atrás, al chat, desde el **Buscar por mi**.

Una vez finalizado el formulario de **Buscar por mi**, el curso de la aplicación vuelve correctamente al chat, a la página original, sin defectos.

El formulario se termino de completar **13:15**, pero el tiempo que tardó en completarlo no se contempla como tiempo de aprendizaje del chat. Este tiempo se considera desde que comenzó la consulta inicial (**13:09**) hasta que descubrió la funcionalidad de **Buscar por mi** y entendió cómo usarla (**13:11**). Por ende, 2min de aprendizaje nuevamente.

Defectos encontrados

La falta de un botón para volver para atrás desde la primer pantalla del formulario de **Buscar por mi**. Se abrirá un *bug* correspondiente.

Conclusiones de la sesión

Para ambas pruebas, el tiempo de aprendizaje del chat fue de 2 minutos (cumpliendo así con **RNF-US-01**). En una de ellas, se encontró un defecto que afectó la experiencia de usuario, y se remarcaron varios puntos positivos a nivel de experiencia de usuario también.

15.6.2 Segunda sesión

- **Misión**: Validar que el tiempo de aprendizaje del chat es de máximo 3 minutos.
- **Fecha**: 18/2/2024.
- **Tester**: Nicolás Juan.
- **Supervisor**: Andrés Juan.
- **Estructura**: Corta (20' o menos).
- **Entorno**: Safari iPadOS 17.1.2 Sexta Generación.

Para esta sesión se planificaron dos pruebas, correspondientes a dos búsquedas muy comúnes (por ello, son preguntas sugeridas o *follow-ups* de *wisello*):

- Una búsqueda de un apartamento en Pocitos.
- Una búsqueda de una casa en Carrasco.

15.6.2.1 Notas de las pruebas

Prueba 1: búsqueda de apartamento en Pocitos. Inicio 14.47 pm.

El usuario escribe *"Quiero comprar un apto en pocitos con al menos 1 garage y gastos comúnes bajos"*, y presiona el botón de **Enviar**.

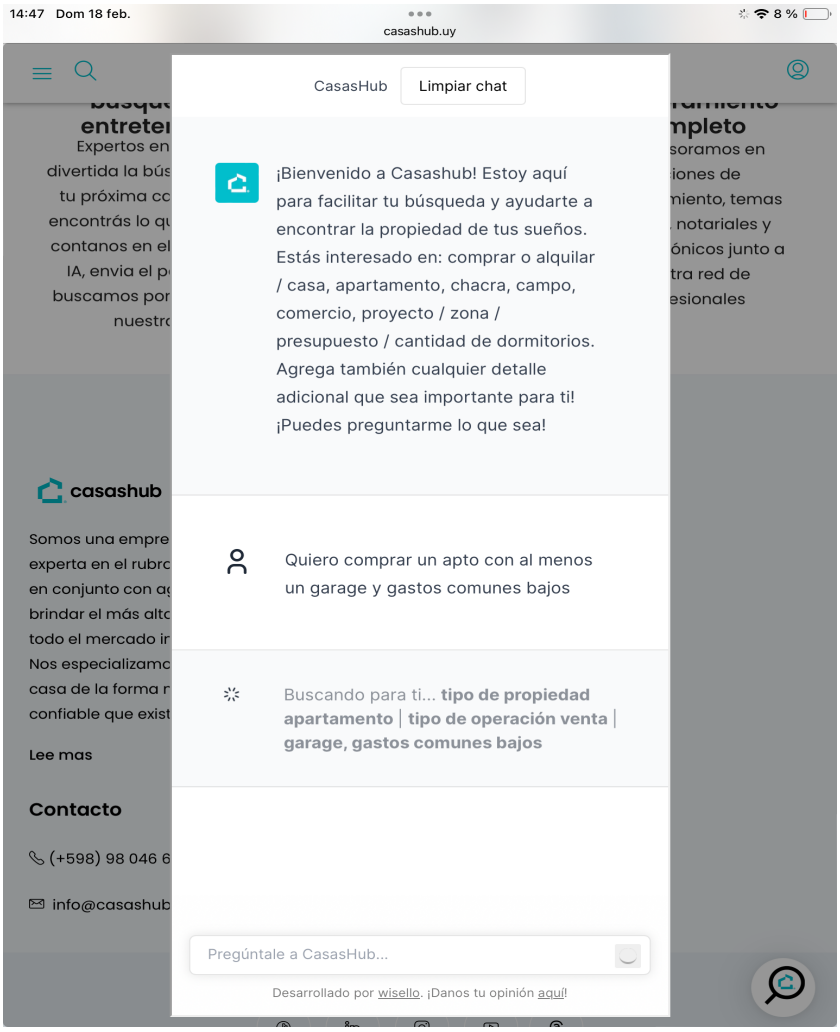


Figura 15.16: Pregunta inicial al chat

Luego, este recibe la respuesta con un apartamento recomendado en Pocitos con al menos un garage, incluyendo la propiedad recomendada con su redirección a `casashub.uy`.

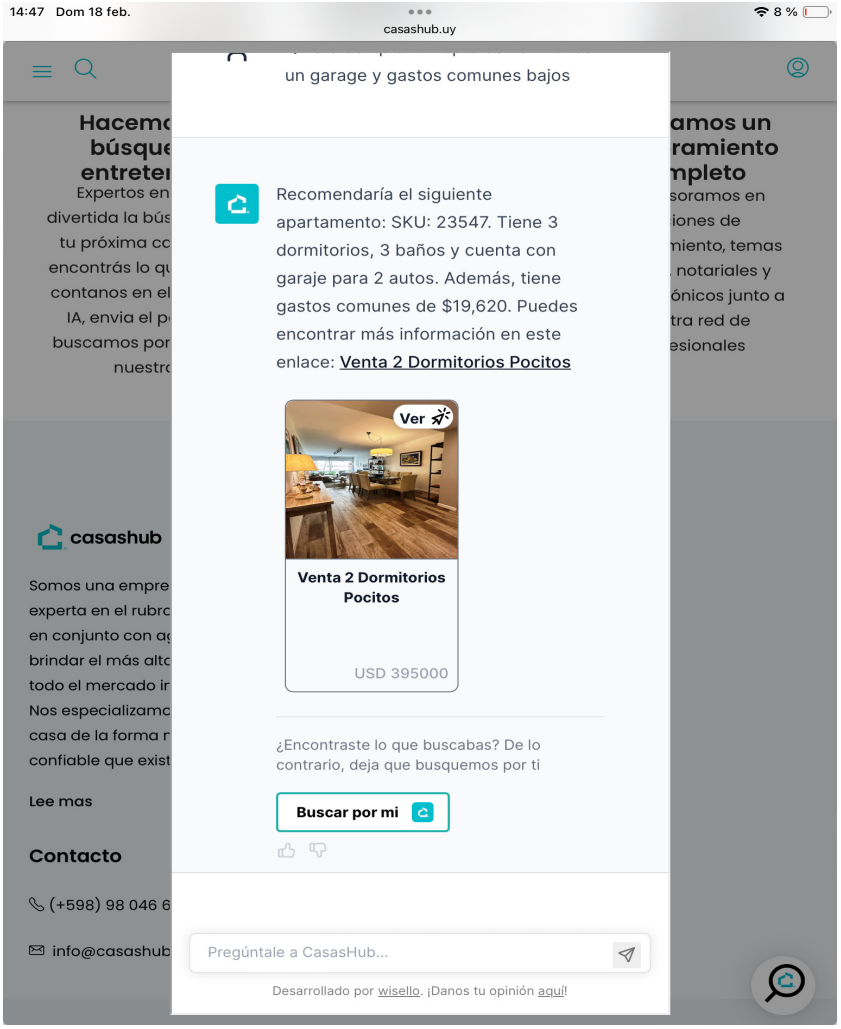


Figura 15.17: Respuesta del chat a la pregunta inicial

El usuario, tras considerar que los gastos comunes del apartamento recomendado son muy altos, escribe lo siguiente al chat: *"No tenes nada con gastos comunes más bajos? "*, y presiona el botón de **Enviar**.

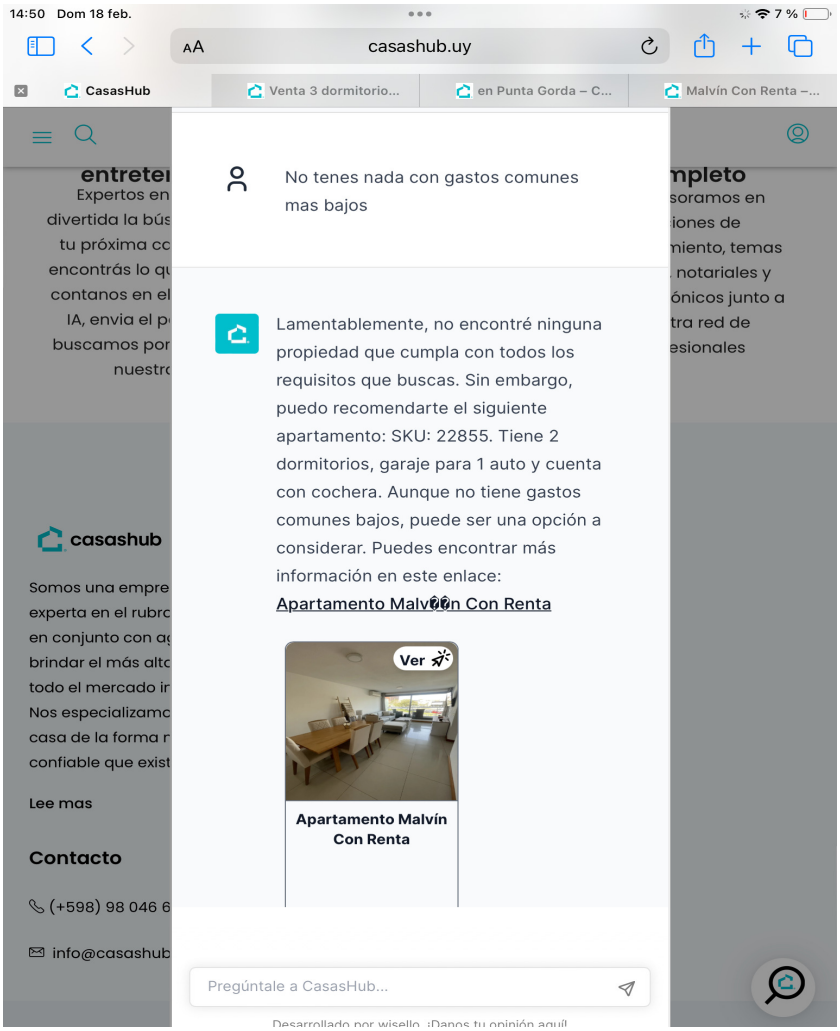


Figura 15.18: Re-pregunta al chat tras la respuesta a la pregunta inicial

El usuario nota que el tilde se representa con un carácter raro y erróneo.

Luego, al recibir una nueva recomendación, el usuario analiza la nueva propuesta ingresando al portal, clickeando sobre la propiedad recomendada:

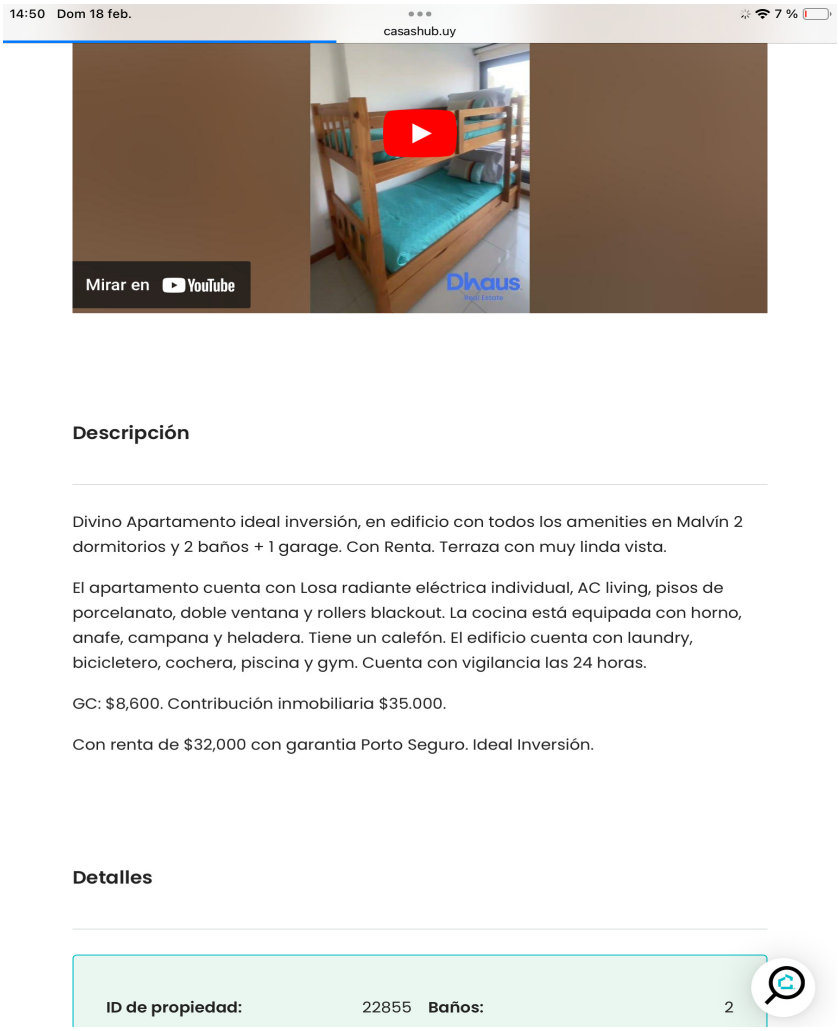


Figura 15.19: El usuario ingresa al portal para analizar la nueva propiedad recomendada

El usuario nota que, si bien no está ubicado en Pocitos, está en un barrio cercano y tiene gastos comunes $8.600, lo cual se adecúa a su presupuesto de gastos comunes bajos, por lo que decide finalizar su búsqueda a las **14:50 pm**, 3 minutos después que este la ha comenzado, cumpliendo así con el tiempo de aprendizaje de 3 minutos.

Prueba 2: búsqueda de casa en Carrasco. Inicio 14.47 pm.

Para esta consigna, el usuario clickea sobre la pregunta sugerida **Compra casa en Carrasco**, y esto automáticamente dispara la invocación al chat:

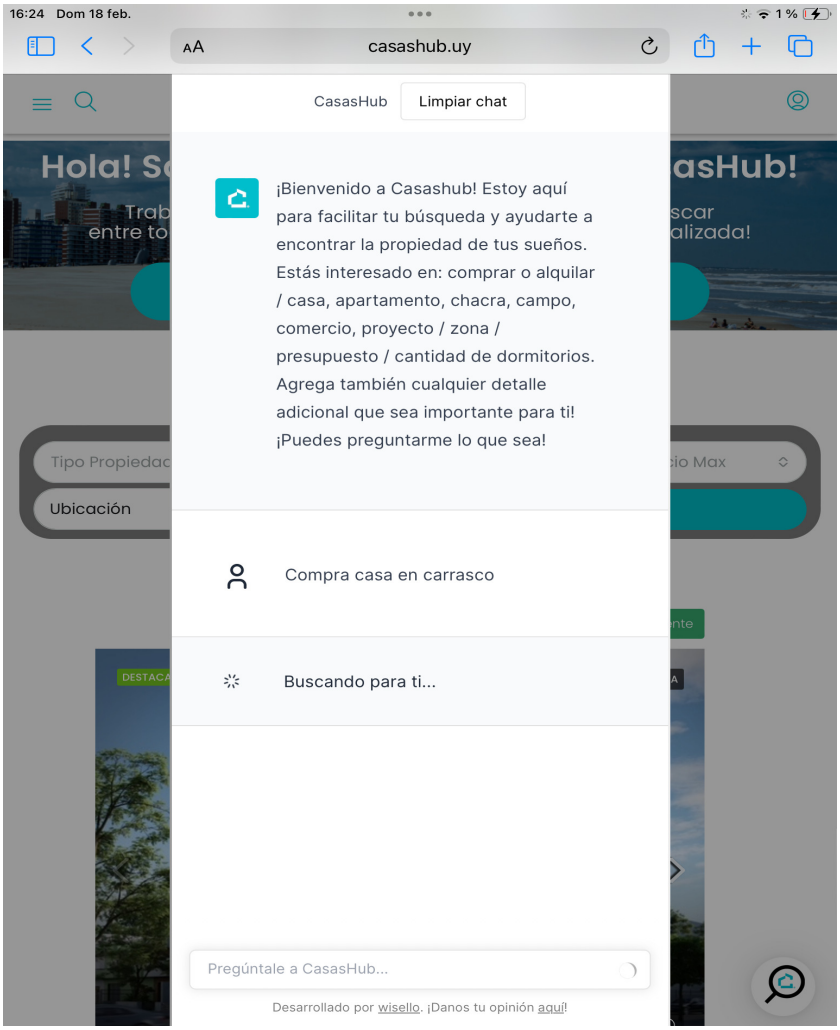


Figura 15.20: Click sobre la pregunta sugerida *Compra casa en Carrasco*

El usuario recibe una serie de tres recomendaciones de casas en carrasco, y le interesa conocer cuál está ubicada más cerca del Club Lawn Tennis. Para ello, escribe *"Cuál está mas cerca del Lawn Tennis?"* en el chat y presiona **Enviar**:

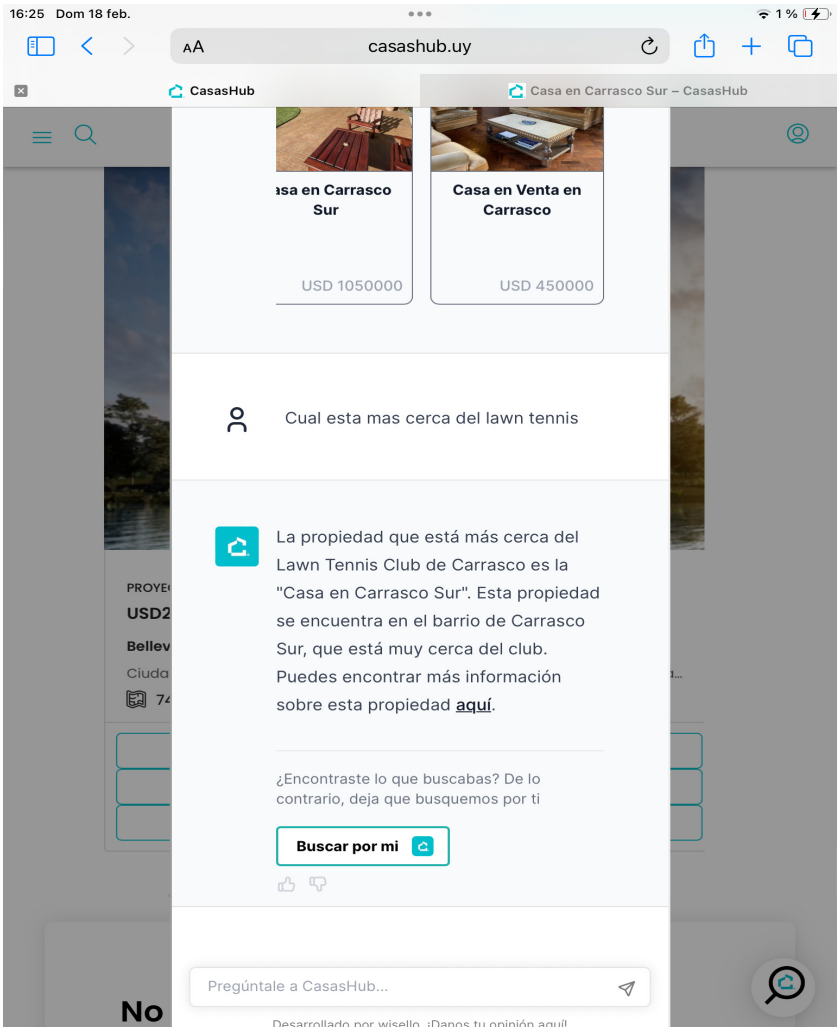


Figura 15.21: Re-pregunta al chat por cercanía al Club Lawn Tennis

Recibida la respuesta con la casa más cercana, clickea sobre la propiedad mencionada, y es redigirido a la propiedad en `casashub.uy`. Corrobora la ubicación y ve que efectivamente es muy cercana al Club Lawn Tennis:

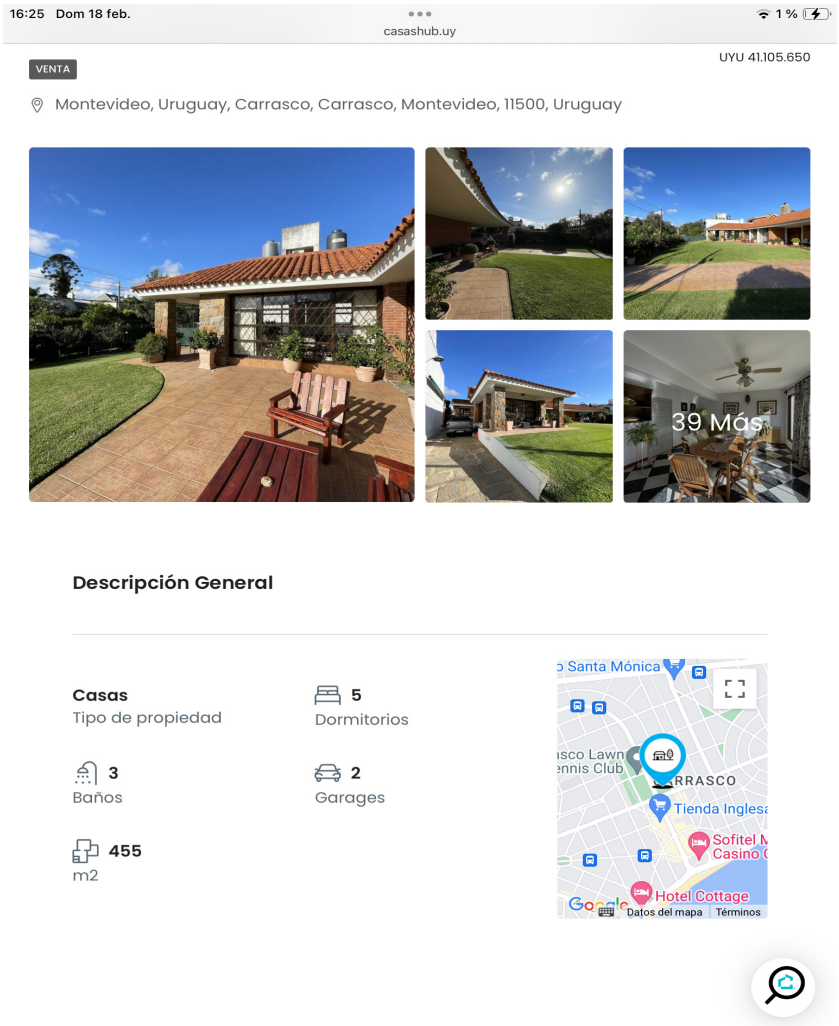


Figura 15.22: Respuesta del chat con propiedad cercana al Club Lawn Tennis

Luego, el usuario decide volver al chat y dar `like` a la respuesta anterior dada lo acertada que fue la recomendación, por la cercanía de la propiedad al Club Lawn Tennis:

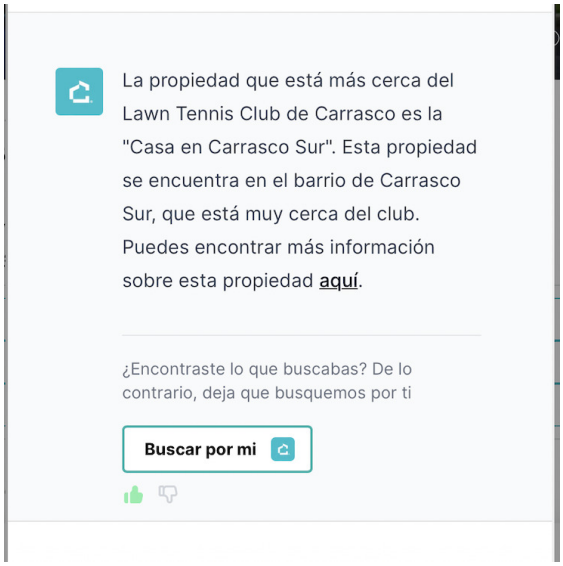


Figura 15.23: *Like* a la respuesta del chat

Esta segunda prueba se realizó, de forma satisfactoria, en un lapso de **1 minuto**, interactuando con la nueva funcionalidad del `like/dislike`.

Defectos encontrados

Los tildes, por momentos, no se escriben correctamente. Este *bug* ya había sido notado y agregado al backlog por el equipo.

Conclusiones de la sesión

En ambas pruebas, se cumplió con el **RNF-US-01**, teniendo tiempos de aprendizaje de 3 minutos y 1 minuto respectivamente, lo cual se debe a que en la primera prueba se dio gran parte del aprendizaje del chat, y en la segunda prueba el aprendizaje fue menor dado que ya se había interactuado con gran parte de las funcionalidades en la prueba anterior.

15.6.3 Tercera sesión

- **Misión**: Validar que el tiempo de aprendizaje de la plataforma es de máximo 10 minutos.
- **Fecha**: 18/2/2024.
- **Tester**: Juan Valdéz.
- **Supervisor**: Marcelo Pérez.
- **Estructura**: Media (30').
- **Entorno**: Safari v15.2 MacOs Monterey v12.1.

Para esta sesión se planificó una sola prueba integral que consiste del aprendizaje de todas las siguientes funcionalidades:

- *Loguearse* a la plataforma.
- Cambiar el mensaje inicial de *wisello* para Casashub.
- Descargar las conversaciones para el día actual.
- *Desloguearse* de la plataforma.

15.6.3.1 Nota de la prueba

Prueba Integral. Inicio 16.56 pm.

El usuario es presentado con la pantalla de inicio y reconoce apropiadamente el botón de **Login**, y lo aprieta.

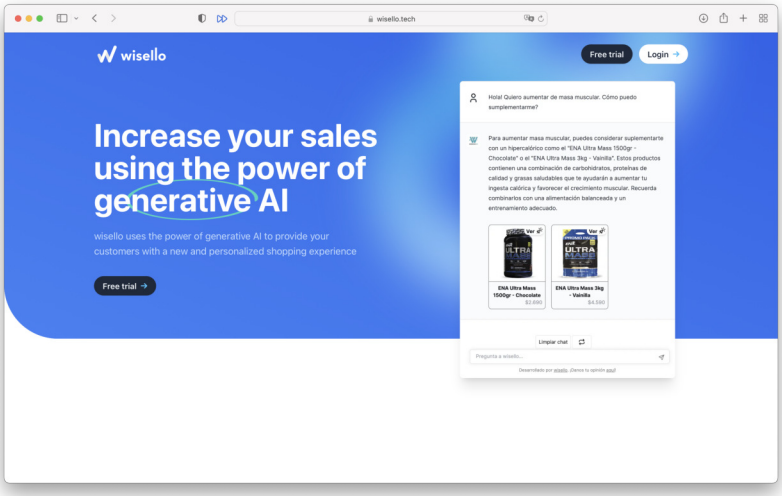


Figura 15.24: Punto de partida: pantalla de inicio

El usuario identifica los campos para ingresar las credenciales correctamente, y aprieta el botón **Login** para *loguearse* ahora sí la plataforma:

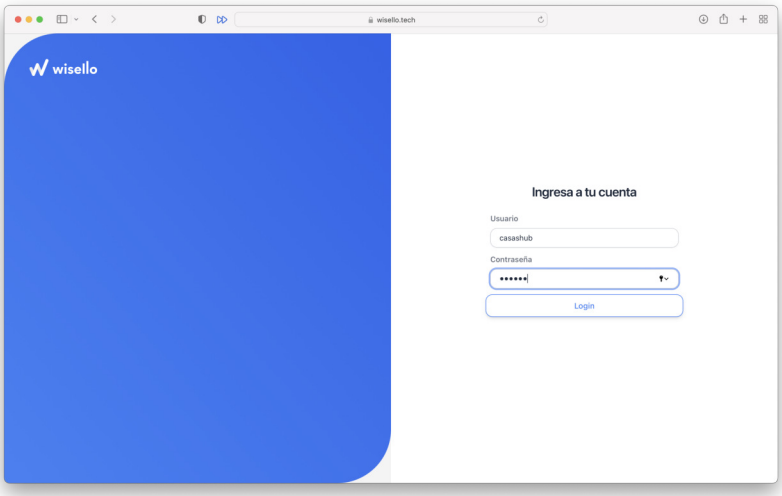


Figura 15.25: Login satisfactorio a la plataforma

El usuario es presentado con las métricas de la tienda:

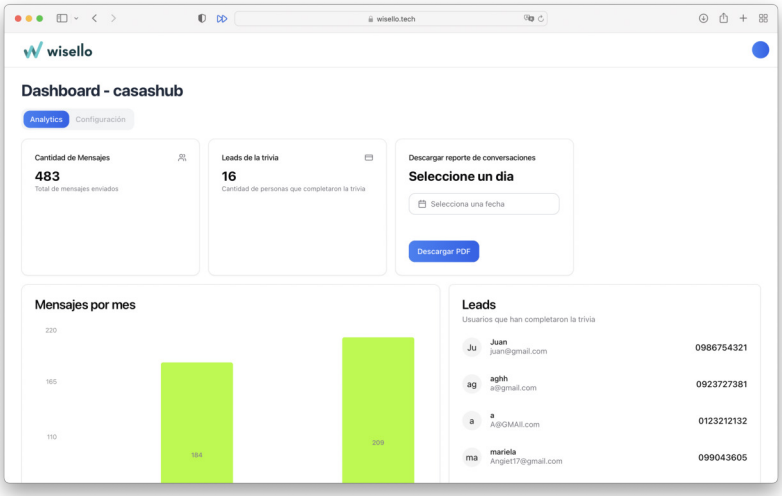


Figura 15.26: Métricas de la tienda

Con el fin de descargar las conversaciones del día actual, identifica correctamente el componente de **Descargar reporte de conversaciones**. Clickea sobre el *date-picker* (**Selecciona una fecha**), selecciona la fecha actual y luego aprieta el botón **Descargar PDF**.

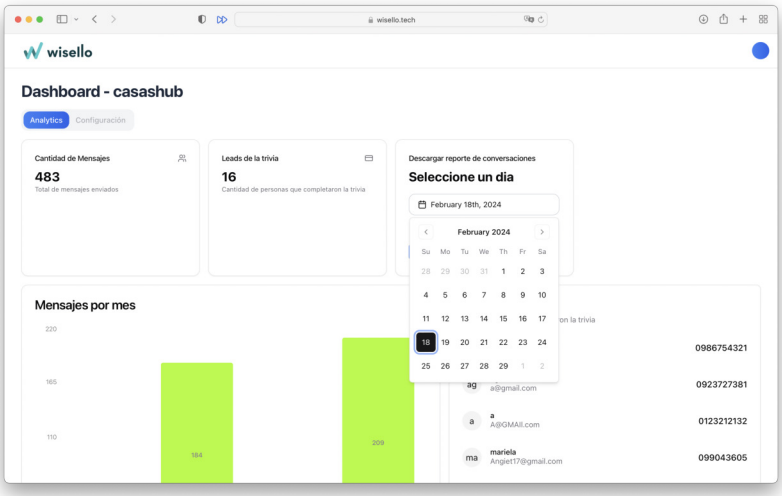


Figura 15.27: Descarga de reporte de conversaciones del día actual

Pasados unos segundos, el usuario nota que se descarga un PDF en su siste-

ma de archivos, pero que no hay retroalimentación para esta acción (algo como *"se ha descargado un PDF en su computadora con el reporte de conversaciones solicitado"*, lo cual se identifica como defecto.

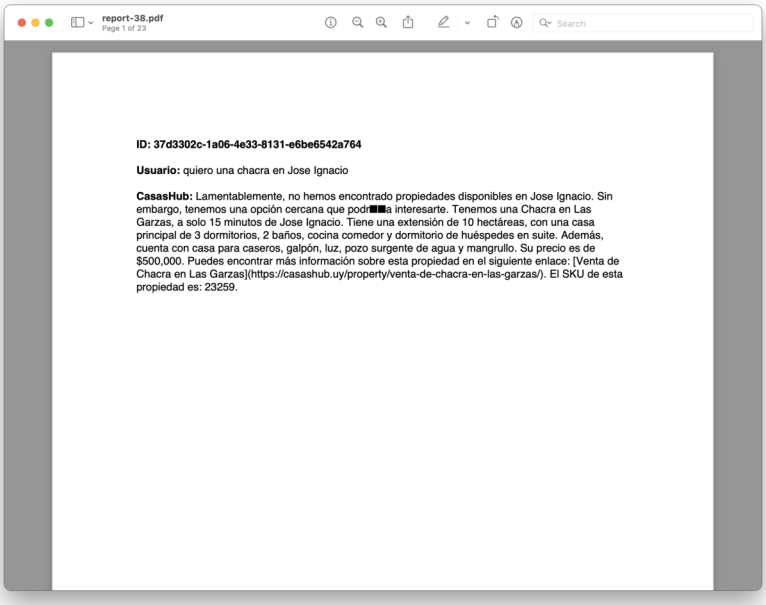


Figura 15.28: Reporte de conversaciones del día actual

A continuación, para cambiar el mensaje inicial, reconoce correctamente que debe dirigirse a la pestaña de **Configuración**, y clickea sobre la misma. Una vez situado sobre esta, agrega un espacio al mensaje inicial (cambio no representativo), y aprieta el botón **Guardar** para persistir la acción, cambiando satisfactoriamente el mensaje inicial de *wisello*.

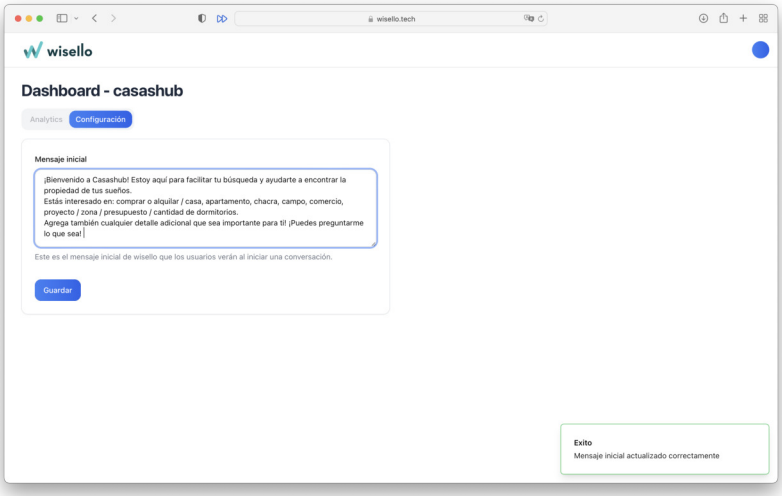


Figura 15.29: Modificación del mensaje inicial

Para *desloguearse*, identifica correctamente el botón de **Log out** en la esquina superior derecha, lo aprieta, y es redirigido a la pantalla de inicio, donde comenzó la prueba.

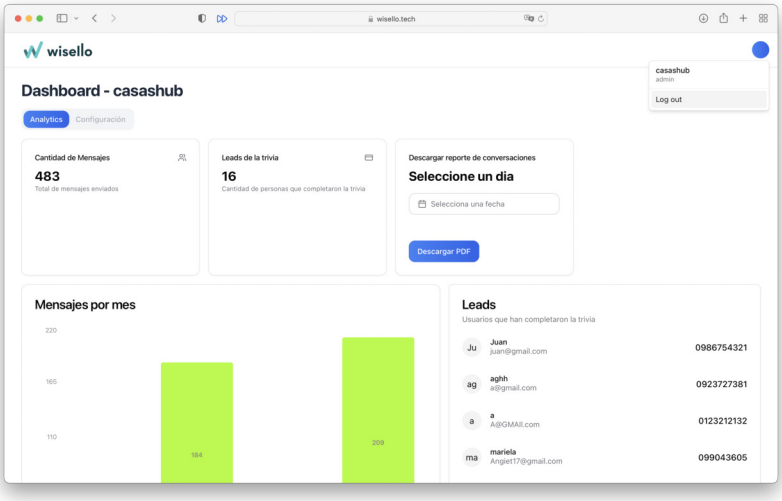


Figura 15.30: Log out satisfactorio

Fin: 17.04 pm.

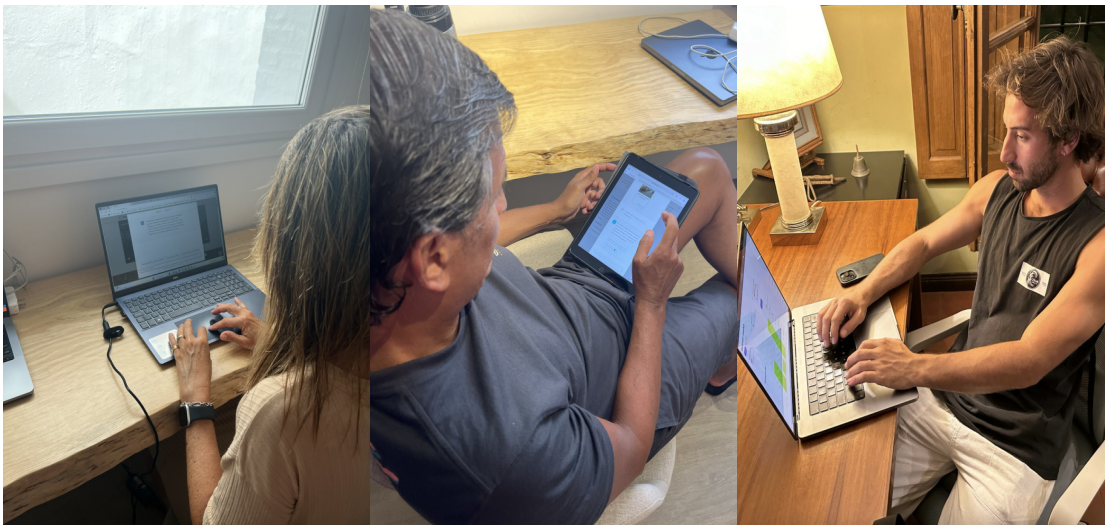
Defectos encontrados

Falta de retroalimentación en la descarga del reporte de conversaciones. Se abrirá un *bug* correspondiente.

Conclusiones de la sesión

El tiempo de aprendizaje de todas las funcionalidades de la plataforma fue de 8 minutos, cumpliendo así con el **RNF-US-01**, que exigía que este sea como máximo 10 minutos.

15.6.4 Evidencia de las sesiones



15.7 Estandáres de código

15.7.1 Cumplimiento con PEP8 - Python

Para ello, se utilizó la extensión de VSCode **Python**, que entre muchas cosas, nos provee la capacidad de *code formatting*.

A continuación se exponen las reglas de PEP8 [53] y la evidencia de su cumplimiento para *wisello*.

15.7.1.1 *Naming*

Nombramiento de clases

El nombre de clases debe ser en **CamelCase**:

```
1 class OpenAIFunctionsCacheLLM(ChatOpenAI):
```

```
1 class ShopCompletion:
```

Nombramiento de variables

El nombre de variables debe ser **snake_case** y **lowercase**:

```
1 message, history = extract_message_and_history(send_message.messages)
2 shop_resources = return_shop_resources(send_message.shop)
3 tools = shop_resources.get_tools()
4 prompt_template = shop_resources.get_prompt()
```

Nombramiento de funciones

El nombre de funciones debe ser **snake_case** y **lowercase**:

```
1 def make_chatbot(send_message: CompletionRequest) -> Chatbot:
```

Nombramiento de constantes

El nombre de funciones debe ser **snake_case** y **uppercase**:

```
INDEX_NAME = "sample-test"
ENGINE_MODEL = "text-embedding-ada-002"
```

15.7.1.2 *Code lay-out*

Usar 4 espacios por cada nivel de indentación:

```
def _generate_with_cache(
    self,
    messages: List[BaseMessage],
    stop: Optional[List[str]] = None,
    run_manager: Optional[CallbackManagerForLLMRun] = None,
    **kwargs: Any,
):
    llm_string = self._get_llm_string(stop=stop, **kwargs)
    ...
```

Longitud máxima de línea

La longitud máxima de línea es 79 carácteres:

```
init_similar_cache(cache_obj=cache_obj, evaluation=ExactMatchEvaluation(
    ), data_dir=f"../cache_files/map_cache", embedding=OpenAI())
```

```
shops_collection.update_one(
            {"route": shop_route}, {"$inc": {"message_count": 1}}
        )
```

Líneas en blanco

A las definiciones de funciones y clases las deben preceder dos líneas en blanco.

```
def save_langchain_run_id(langchain_run_id: str, request_id: str):
    logging.info(
        f"Inserting langchain run id in cache for request {request_id}.")
    set_key(f"request-id-{request_id}", langchain_run_id)

    return request_id


def retreive_langchain_run_id(request_id: str):
    logging.info(
        f"Fetching langchain run id from redis for request {request_id}.")
    langchain_run_id = get_key(f"request-id-{request_id}")

    return langchain_run_id
```

Métodos dentro de una clase son separados por una sola línea en blanco:

```
class Chatbot:
    def __init__(self, generator_object, token_queue, has_context, model_name,
    streaming_callback_handler):
        self.has_context = has_context
        self.token_queue = token_queue
        self.generator_object = generator_object
        self.model_name = model_name
        self.prompt_token_count = 0
        self.completion_token_count = 0
        self.streaming_callback_handler = streaming_callback_handler

    def get_prompt_token_cost(self):
        return calculate_model_cost_prompt(self.prompt_token_count, self.
    model_name)
```

Imports

- En líneas separadas a no ser de que provean del mismo módulo.
- Siempre al principio del archivo.
- Orden (se debe colocar una línea en blanco entre cada grupo):
    - Imports de librerías estándar.
    - Imports de librerías de terceros relacionadas.
    - Imports locales de la aplicación.

```python
import os
import logging
from datetime import datetime, timedelta

from pymongo.errors import PyMongoError

from data_access.mongo_db import get_database_conn, handle_mongo_error
from exceptions import ServerErrorException, BadRequestException
from models.shop import Shop
```

15.7.1.3 Comentarios

Deben ser escritos en inglés.

In-line comments

Utilizar con moderación. Deben estar en la misma línea que la instrucción, separados por al menos dos espacios, comenzando con un # y un espacio simple.

```python
if token and not self.feedback_handler.sent_placeholder:   # cache hit
```

Docstrings

Escribe *docstrings* para todos los módulos públicos, funciones, clases y métodos.

Nota: esta convención solo se cumplió para los módulos, funciones, clases y métodos que se creyó necesario.

```python
class StreamingLLMCallbackHandler(BaseCallbackHandler):
    """Callback handler for streaming"""
```

```python
def add_last_message_conversation(message: str, conversation_id: str):
    """Add last message to conversation in db"""
```

15.7.1.4 Recomendaciones de programación

Uso de is / is not

Las comparaciones con singletons como None siempre deben hacerse con **is** o **is not**, nunca con los operadores de igualdad.

```
1 if api_key is None:
2         raise UnauthorizedException("Missing api-key in header.")
```

Procurar usar **is not** (como a continuación) antes que **not ... is**.

```
1 if shop_completion.messages is not None:
2     messages = shop_completion.messages.split(separator)
```

Heredar excepciones de Exception, y no de BaseException

```
1 class BadRequestException(Exception):
```

Catch clause

Mencionar excepciones específicas en el *catch*, y limitar la clausula a la mínima cantidad de código posible.

```
1 try:
2     payload = jwt.decode(api_key, key=jwt_secret_key, algorithms=["HS256"])
3         return payload
4     except jwt.exceptions.ExpiredSignatureError:
5         raise UnauthorizedException("Provided api-key has expired")
6     except jwt.exceptions.InvalidTokenError:
7         raise UnauthorizedException("Provided api-key is invalid.")
8     except jwt.exceptions.PyJWTError:
9         raise UnauthorizedException("Something went wrong while verifying api-
      key")
```

No comparar tipos booleanos con True o False

```
1   if not value_in_cache:
```

15.7.2 Cumplimiento con ES - Typescript

Para ello, se utilizó la extensión de VSCode **Prettier**, cuyas reglas se exponen a continuación, con la evidencia de su cumplimiento para *wisello*.

Usar camelCase al nombrar variables y funciones

```
1 const textareaRef = useRef<HTMLTextAreaElement>(null);
2
3 const onClearAll = () => {
4   onUpdateConversation(conversation, { key: 'messages', value: [] });
5   setLimitReached(false);
6 };
```

Nombres de los constructores deben comenzar con mayúscula

```
1 export const OpenAIStream = async (res: any) => {
```

Indentación y espacio luego de una *keyword*

Utilizar 2 espacios de indentación.

```
1 if (isInitialMessage && followUps) {
2   setShowFollowUps(true);
3 }
```

Comillas simples para strings

```
1 onUpdateConversation(conversation, { key: 'messages', value: [] });
```

Longitud máxima de línea: 80 caracteres

```
1 const getImageTitle = () =>
2     doc.name.length > MAX_NAME_LENGTH
3       ? doc.name.substring(0, MAX_NAME_LENGTH) + '...'
4       : doc.name;
```

Usar === en vez de ==

```
if (shop === 'casashub') {
  return `USD ${doc.price}`;
}
```

Los operadores infijos deben estar separados por un espacio

```
const MAX_NAME_LENGTH = 38;
```

Las comas deben tener un espacio después de las mismas

```
const { shopTitle, expertMode, handleGptVersionChange, onClearAll } = props;
```

Mantener las sentencias else en la misma línea que sus llaves

Los bloques if/else/for/while/try siempre deben usar llaves y siempre deben extenderse en múltiples líneas.

```
if (isInitialMessage && followUps) {
  setShowFollowUps(true);
} else {
  setShowFollowUps(false);
}
```

Para el operador ternario en un entorno de múltiples líneas, colocar ? y : en sus propias líneas

```
const getImageTitle = () =>
    doc.name.length > MAX_NAME_LENGTH
      ? doc.name.substring(0, MAX_NAME_LENGTH) + '...'
      : doc.name;
```

Utilizar un solo import por módulo

```
1 import { ChatBody, Message } from '@/types/chat';
```

Comentarios

Se colocan antes del código al que refieren y van precedidos por una línea en blanco.

```
1  const startIndexOfToken = buffer.indexOf(tokenStart);
2
3  // Exit loop if tokenStart not found
4  if (startIndexOfToken === -1) break;
5
6  const endIndexOfToken = buffer.indexOf(
7    tokenEnd,
8    startIndexOfToken + tokenStart.length,
9  );
10
11 // Exit loop if tokenEnd not found
12 if (endIndexOfToken === -1) break;
```

Constantes

Las constantes se deben nombrar con **snake_case** y **uppercase**.

```
1 export const DEFAULT_INITIAL_MESSAGE: Message = {
2   role: 'assistant',
3   content:
4     'Hola! Soy Wisello, tu asistente de inteligencia artificial. Necesitas
      recomendaciones o tienes alguna pregunta? Estoy aquí para ayudarte!',
5   sourceDocs: [],
6 };
```

15.8 Clean Code

En la siguente sección se exponen los principales lineamientos de **Clean Code** y prácticas de programación que fueron aplicados en *wisello*, con el objetivo de que el código fuente fácil de leer, entender y modificar.

Además, para cada uno de los puntos, se incluye evidenica del mismo con ejemplos del propio proyecto.

15.8.1 *Sobre los nombres*

Usar nombres que revelen la intención

```
value_in_cache
```

```
questions_for_refinement_for_property
```

Evitar la desinformación

```
price_in_usd
```

Hacer distinciones significativas y usar nombres buscables

```
langchain_run_id: str
```

```
request_id: str
```

```
conversation_id
```

Evitar las constantes que yacen sueltas en el código

```
1 class PineconeNamespaces(Enum):
2     arquitectura = "pdf-test"
3     casashub = "casashub-v2"
4     divino = "divino-v1"
5     metropolitana = "metropolitana-v1"
```

15.8.1.1 Nombres de clases

Usar palabras significativas y elegir sustantivos

`OpenAIFunctionsCacheLLM`

`SearchCatalogue`

`CasashubPineconeQuery`

15.8.1.2 Nombres de métodos

Siempre deben incluir verbos, con una única palabra por concepto, y *solution domain names*

```
1 def retreive_langchain_run_id(request_id: str):
```

15.8.2 *Sobre las funciones*

Principio de responsabilidad única

```
1 def get_database_conn():
2     """Returns a connection to the database"""
3
4 def get_shop_from_db(shop_route) -> Shop:
5     """Returns shop from db"""
6
7 def add_lead_to_shop_in_db(shop_route, name, email, phone):
8     """Adds lead to shop in db"""
```

Pocos parámetros

En general todas los tienen, con la única excepción de las funciones `_run` de las `tools`, en las cuales los parámetros no pueden ser agrupados en un objeto, pues directamente este método es llamado por el flujo del agente, con los parámetros indicados en el `args_schema`, que no acepta objetos.

La clase `SearchPropertiesInput` define los parámetros con que el método `_run` se llamará:

```python
class SearchProperties(BaseTool):
    name = "SearchProperties"
    args_schema: Type[BaseModel] = SearchPropertiesInput

    def _run(
        self,
        neighbourhoods: Optional[list[str]] = [],
        query: Optional[list[str]] = [],
        neighbourhood_description: Optional[str] = None,
        operation_type: Optional[str] = None,
        bedrooms: Optional[int] = None,
        bathrooms: Optional[int] = None,
        price: Optional[float] = None,
        currency: Optional[str] = None,
        brand_new: Optional[bool] = False,
        property_type: Optional[str] = None,
    ):
```

¿Cómo arreglar un switch? Esconder la creación de objetos dentro de un Abstract Factory

```python
def return_shop_resources(shop_name):
    match shop_name:
        case "casashub":
            return CasashubResources()
        case "metropolitana":
            return MetropolitanaResources()
        case "divino":
            return DivinoResources()
        case "carlotta":
            return CarlottaResources()
        case "arquitectura":
            return ArquitecturaResources()

    raise BadRequestException("Unexistent shop.")
```

15.8.3 Manejo de errores

Usar excepciones con nombre explicativo en lugar de códigos de error

```
BadRequestException, ConflictException, ForbiddenException, NotFoundException,
    ServerErrorException, UnauthorizedException,
```

15.9 *User stories*

Epic: CALIFICACION DE RESPUESTA

COMO usuario de e-commerce QUIERO poder calificar la respuesta obtenida PARA expresar mi gusto/disgusto con esta

Criterios de aceptación:

- DADA una respuesta CUANDO se la califica ENTONCES el botón seleccionado debe cambiar el color según cómo fue calificada.

Epic: ADMIN SHOPS

COMO administrador del sistema QUIERO ver cuanta plata va gastando cada tienda PARA trackear los costos de de las mismas

Criterios de aceptación:

- DADO un e-commerce CUANDO se accede al dashboard de Azure ENTONCES se puede ver la evolución del gasto en una gráfica.
- DADO un e-commerce CUANDO se accede al dashboard de Azure ENTONCES se puede ver el gasto total acumulado.

Epic: SUGERENCIA DE PREGUNTAS

COMO usuario de e-commerce QUIERO ser presentado con sugerencias de preguntas PARA darme ideas sobre qué preguntar

Criterios de aceptación:

- DADO un usuario de e-commerce CUANDO este ingresa al chat ENTONCES se le sugieren sugerencias fijas de preguntas PARA darle ideas sobre qué preguntar.
- DADA una sugerencia de pregunta CUANDO esta es clickeada ENTONCES se debe hacer directamente la pregunta al chat.
- DADO un usuario de e-commerce CUANDO este es sugerido con preguntas ENTONCES no deben ser más de 2 en mobile, ni más de 3 en web.

Epic: REDIRECCIÓN AL PERSONAL DE LA TIENDA

COMO e-commerce QUIERO poder recibir solicitudes de búsqueda o asesoramiento de usuarios que no encontraron lo que buscaban PARA no perderlos como lead

Criterios de aceptación:

- DADO un e-commerce CUANDO recibe una búsqueda de un usuario inconcluso ENTONCES este debe recibirla por email.
- DADO un e-commerce CUANDO recibe una búsqueda de un usuario inconcluso ENTONCES este debe recibir el nombre, teléfono y mail del usuario para poder contactarse con él.

Epic: RECOMENDACIÓN PERSONALIZADA

COMO ecommerce QUIERO que la recomendación se base en en el catálogo de la tienda PARA que mis usuarios tengan una buena experiencia y busquen lo que encuentran

Criterios de aceptación:

- DADO un e-commerce CUANDO se consulta por productos de la tienda a *wisello* ENTONCES se deben recomendar productos propios de la tienda.
- DADO un e-commerce CUANDO se consulta por una recomendación a *wisello* ENTONCES se debe recomendar buscando el catálogo con métodos de búsqueda semántica.

COMO usuario de e-commerce QUIERO que la recomendación sea personalizada PARA tener una mejor experiencia

Criterios de aceptación:

- DADO un usuario de e-commerce CUANDO este pide una recomendación ENTONCES la respuesta es personalizada, teniendo en cuenta los requisitos del usuario.

COMO e-commerce QUIERO que la recomendación sea acertada PARA que mis usuarios encuentren lo que buscan, tengan una mejor experiencia de compra y así incremente las conversiones

Criterios de aceptación:

- DADO un e-commerce CUANDO sus usuarios interactúan con wisello ENTONCES reconoce que las respuestas son acertadas, es decir, cumplen con las reglas del negocio. El e-commerce aclama .^{es}tas respuestas están bien".

COMO usuario de e-commerce QUIERO que las respuestas de wisello sean acordes PARA no perder tiempo leyendo información irrelevante

Criterios de aceptación:

- DADO un usuario de e-commerce CUANDO wisello responde ENTONCES no debe incluir información no pertinente a la pregunta en cuestión.

COMO usuario de e-commerce QUIERO que quien me asesore se parezca a un vendedor de la tienda PARA vivir la misma experiencia de tienda offline en la tienda online

Criterios de aceptación:

- DADO un usuario de e-commerce CUANDO wisello responde ENTONCES debe seguir el hilo de la conversación.
- DADO un usuario de e-commerce CUANDO wisello responde ENTONCES debe tener conocimientos de la tienda y su catálogo.
- DADO un usuario de e-commerce CUANDO wisello responde ENTONCES la respuesta debe ser inmediata.

Epic: CUSTOMIZACIÓN DEL CHAT

COMO e-commerce QUIERO que la interfaz de wisello se adapte a mis necesidades PARA que se alinee con mi web y la identidad de mi empresa

Criterios de aceptación:

- DADO un e-commerce CUANDO un usuario quiere interactúar con *wisello* ENTONCES la URL es única para este.
- DADO un e-commerce CUANDO un usuario quiere interactúar con *wisello* ENTONCES el mensaje inicial es propio de la tienda.
- DADO un e-commerce CUANDO un usuario quiere interactúar con *wisello* ENTONCES la foto es propia de la tienda.
- DADO un e-commerce CUANDO un usuario quiere interactúar con *wisello* ENTONCES las preguntas sugeridas son propias de la tienda.
- DADO un e-commerce CUANDO un usuario quiere interactúar con *wisello* ENTONCES la foto de la burbuja es propia de la tienda.
- DADO un e-commerce CUANDO un usuario quiere interactúar con *wisello* ENTONCES la frase de la burbuja es propia de la tienda.

Epic: INTEGRACIÓN

COMO e-commerce QUIERO poder integrar el chat a mi web

Criterios de aceptación:

- DADO un e-commerce CUANDO este decide integrar wisello a su tienda ENTONCES lo debe poder hacer en formato burbuja.
- DADO un e-commerce CUANDO este decide integrar wisello a su tienda ENTONCES lo debe poder hacer en formato embebido.

COMO e-commerce QUIERO que la integración del chat a mi web sea rápida y sencilla PARA no tener costos extra de desarrollo web

Criterios de aceptación:

- DADO un e-commerce CUANDO este decide integrar wisello a su tienda ENTONCES lo único que tiene que hacer es agregar una línea de código a su web.

Epic: ACCESO A PRODUCTOS

COMO usuario de e-commerce QUIERO poder acceder al artículo recomendado PARA poder visualizarlo mejor y eventualmente comprarlo

Criterios de aceptación:

- DADA una respuesta a un usuario de e-commerce CUANDO en esta se recomienda un artículo ENTONCES se debe mostrar una previsualización clickeable del artículo incluyendo su foto, precio y título.
- DADA una respuesta a un usuario de e-commerce CUANDO en este se recomienda un artículo ENTONCES debe estar disponible en la tienda.
- DADO un artículo recomendado CUANDO se clickea en este ENTONCES debe llevar a la página del artículo.

Epic: FEEDBACK DE BÚSQUEDA

COMO usuario de e-commerce QUIERO recibir feedback previo a recibir la respuesta del chat PARA saber el estado del sistema

Criterios de aceptación:

- DADO un mensaje CUANDO se envía a wisello ENTONCES inmediatamente se hace disponible un feedback genérico *"Buscando para ti... "*
- DADO un mensaje CUANDO se envia y se hace disponible el feedback de respuesta personalizado ENTONCES se lo presenta al usuario.

Epic: BORRAR CHAT

COMO usuario de e-commerce QUIERO poder limpiar la conversación PARA comenzar una nueva

Criterios de aceptación:

- DADA una conversacion con wisello CUANDO se presiona limpiar chat ENTONCES se limpia la conversación y queda en blanco.

COMO usuario de e-commerce QUIERO que no se borre la conversación con wisello PARA poder volver a consultarla

Criterios de aceptación:

- DADO un usuario de e-commerce que ya ha interactuado con wisello CUANDO este cierra wisello y vuelve a entrar ENTONCES puede ver seguir viendo su conversación anterior.

(Esta funcionalidad fue luego revertida pues se entendió que era mejor limpiando la conversación tras el feedback posterior a su despliegue).

Epic: MÁXIMO DE MENSAJES

COMO usuario de e-commerce QUIERO que una respuesta no se corte por la mitad PARA poder recibir respuestas completas, y terminar de leerlas.

Criterios de aceptación:

- DADA una conversacion con wisello CUANDO se llega a 10 mensajes ENTONCES se muestra un cartel diciendo que el máximo permitido para una conversación son 10 mensajes.
- DADA una conversacion con wisello CUANDO se llega a 10 mensajes ENTONCES se le ofrece comenzar una nueva conversación o cancelar.

Epic: CASASHUB

COMO usuario de Casashub QUIERO poder realizar búsquedas de propiedades en forma conversacional PARA buscar más rápido y en forma desestructurada

Criterios de aceptación:

- DADO un usuario de Casashub CUANDO busca una propiedad ENTONCES debe poder buscar por barrio estricto, o departamento (para campos).
- DADO un usuario de Casashub CUANDO busca una propiedad ENTONCES debe poder buscar por tipo de operación estricta.

- DADO un usuario de Casashub CUANDO busca una propiedad ENTONCES debe poder buscar por cantidad de dormitorios o baños estrictos.
- DADO un usuario de Casashub CUANDO busca una propiedad ENTONCES debe poder buscar por precio.
- DADO un usuario de Casashub CUANDO busca una propiedad ENTONCES debe poder buscar por tipo de propiedad estricto.
- DADO un usuario de Casashub CUANDO busca una propiedad ENTONCES debe poder buscar por *cualquier* tipo de atributo que se le ocurra.
- DADO un usuario de Casashub CUANDO busca una propiedad en un barrio determinado y no hay disponibles ENTONCES se le sugieren propiedades en barrios cercanos.
- DADO un usuario de Casashub CUANDO busca una propiedad en un barrio inexistente ENTONCES se le dice que el barrio no existe.
- DADO un usuario de Casashub CUANDO busca una propiedad con determinados requisitos y no hay disponibles ENTONCES se le responde que no hay disponibles.
- DADO un usuario de Casashub CUANDO busca una propiedad con determinados requisitos y sí hay disponibles ENTONCES se le recomiendan las propiedad disponibles.
- DADO un usuario de Casashub CUANDO busca una propiedad con determinados requisitos y sí hay disponibles pero no cumplen con *algunos* requisitos ENTONCES se le recomiendan las propiedades, remarcándolo.

COMO usuario de CasasHub QUIERO poder buscar una propiedad particular del portal PARA accederla directamente

Criterios de aceptación:

- DADO un usuario de Casashub CUANDO busca una propiedad particular ENTONCES la debe poder buscar por SKU.
- DADO un usuario de Casashub CUANDO busca una propiedad particular ENTONCES la debe poder buscar por nombre.

COMO usuario inconcluso de CasasHub QUIERO poder delegar mi búsqueda a la tienda PARA encontrar lo que busco ya que no lo pude encontrar

Criterios de aceptación:

- DADO un usuario de Casashub CUANDO no encuentra lo que busca ENTONCES debe tener un botón disponible para enviar la búsqueda al e-commerce.
- DADO un usuario de casashub CUANDO decide delegar su búsqueda ENTONCES este debe ingresar su nombre, teléfono y email.

COMO usuario de Casashub QUIERO poder buscar propiedades a estrenar PARA poder ver los proyectos en el portal

Criterios de aceptación:

- DADO un usuario de casashub CUANDO busca una propiedad a estrenar ENTONCES se devuelven los proyectos asociados a ese tipo de propiedad en el portal.

COMO usuario de Casashub QUIERO poder buscar por muchos barrios a la vez PARA ampliar mi búsqueda a muchos barrios

Criterios de aceptación:

- DADO un usuario de casashub CUANDO busca por una lista de barrios ENTONCES se le devuelven propiedades disponibles en los barrios mencionados.

COMO Casashub QUIERO poder hacer preguntas al usuario dependiendo la propiedad que busca PARA refinar mejor su búsqueda

Criterios de aceptación:

- DADO un usuario de casashub CUANDO solo se conoce que busca una casa ENTONCES las posibles preguntas de refinamiento son: ¿Quiere comprar/alquilar la casa? ¿En qué ubicación desea la casa? ¿Tiene preferencias en precio, número de dormitorios, baños, garajes de la casa?

- DADO un usuario de casashub CUANDO solo se conoce que busca un apartamento ENTONCES las posibles preguntas de refinamiento son: ¿Quiere comprar/alquilar el apartamento? ¿En qué ubicación desea el apartamento? ¿Tiene preferencias en precio, número de dormitorios, baños, garajes, mascotas o amenidades del apartamento?
- DADO un usuario de casashub CUANDO solo se conoce que busca un terreno entonces las posibles preguntas de refinamiento son: ¿En qué ubicación desea el terreno? ¿Que precio está buscando? ¿Area en metros cuadrados del terreno?
- DADO un usuario de casashub CUANDO solo se conoce que busca una chacra entonces las posibles preguntas de refinamiento son: ¿En qué ubicación desea? ¿Que precio estabas buscando? ¿Area en hectáreas?
- DADO un usuario de casashub CUANDO solo se conoce que busca un proyecto entonces las posibles preguntas de refinamiento son: ¿Quiere comprar/alquilar un proyecto? ¿En qué ubicación? ¿Tiene preferencias en precio, número de dormitorios o fecha de entrega?
- DADO un usuario de casashub CUANDO solo se conoce que busca un local comercial entonces las posibles preguntas de refinamiento son: ¿Quiere comprar/alquilar? ¿En qué ubicación lo desea? ¿Tiene preferencias en precio, area en metros cuadrados?

Epic: DIVINO

COMO usuario de Divino QUIERO poder buscar productos en forma conversacional PARA buscar más rápido y en forma desestructurada

Criterios de aceptación:

- DADO un usuario de Divino CUANDO busca un producto del catálogo ENTONCES debe poder buscar por *cualquier* tipo de atributo que se le ocurra.

Epic: PLEXO

COMO usuario de Plexo QUIERO poder acceder a información de las guías en forma conversacional PARA accederlas en forma personalizada y más rápida

Criterios de aceptación:

- DADO un usuario de Plexo CUANDO busca información de las guías ENTONCES debe poder buscar cualquier pregunta que se le ocurra.

Epic: AUTH

COMO e-commerce QUIERO poder autenticarme PARA para que otros no puedan ver mi visualización general

Criterios de aceptación:

- DADO un e-commerce CUANDO este es dado de alto ENTONCES se generan sus credenciales, que luego se pueden modificar.
- DADO un usuario y una contraseña CUANDO el e-commerce los ingresa ENTONCES puede ver su visualización general y configurar su herrmienta.
- DADO un par de credenciales erróneas CUANDO el e-commerce las ingresa ENTONCES es denegado su acceso.
- DADO un usuario y una contraseña CUANDO el e-commerce los ingresa ENTONCES solamente puede ver datos de su propia tienda.

COMO e-commerce QUIERO poder finalizar mi sesión de la plataforma PARA irme de la misma

Criterios de aceptación:

- DADO un e-commerce CUANDO aprieta log out ENTONCES debe redirigirse a la página de aterrizaje de wisello.

Epic: MÉTRICAS DE PRODUCTO

COMO e-commerce QUIERO ver las conversaciones que tienen mis usuarios PARA observar sus dudas más frecuentes

Criterios de aceptación:

- DADO un e-commerce CUANDO este quiere ver conversaciones ENTONCES debe poder descargar un archivo PDF con estas.
- DADO un e-commerce CUANDO este quiere ver todas las conversaciones ENTONCES puede filtrar por una fecha específica.

COMO e-commerce QUIERO poder ver la cantidad de mensajes del mes histórica PARA comparar meses.

Criterios de aceptación:

- DADO un e-commerce CUANDO este ingresa a la visualización ENTONCES debe poder ver el valor de esta métrica para los meses desde que tiene wisello.

COMO e-commerce QUIERO poder ver la cantidad de mensajes en formato número en un período de tiempo PARA ver qué tanto se está usando wisello

Criterios de aceptación:

- DADO un e-commerce CUANDO entro a la visualización de la tienda ENTONCES quiero poder ver un número con el total de leads de la trivia.

COMO e-commerce QUIERO poder ver la cantidad de leads del trivia PARA ver qué tantos leads me están llegando

Criterios de aceptación:

- DADO un e-commerce CUANDO entro a la visualización de la tienda ENTONCES quiero poder ver un número con el total de leads de la trivia.

Epic: DASHBOARD

COMO e-commerce QUIERO poder configurar mi mensaje inicial PARA tener control de ello

Criterios de aceptación:

- DADO un e-commerce CUANDO entra a la visualización de su empresa ENTONCES puede cambiar el mensaje inicial del chat.

Epic: FENICIO DATA PIPELINE

COMO e-commerce de FENICIO QUIERO no intervenir en mi integración de datos PARA que sea más sencillo y no perder tiempo en ello

Criterios de aceptación:

- DADO un e-commerce de FENICIO CUANDO decide integrar wisello a su página ENTONCES no debe proveer un excel con el catálogo de productos, una api, o nada similar.

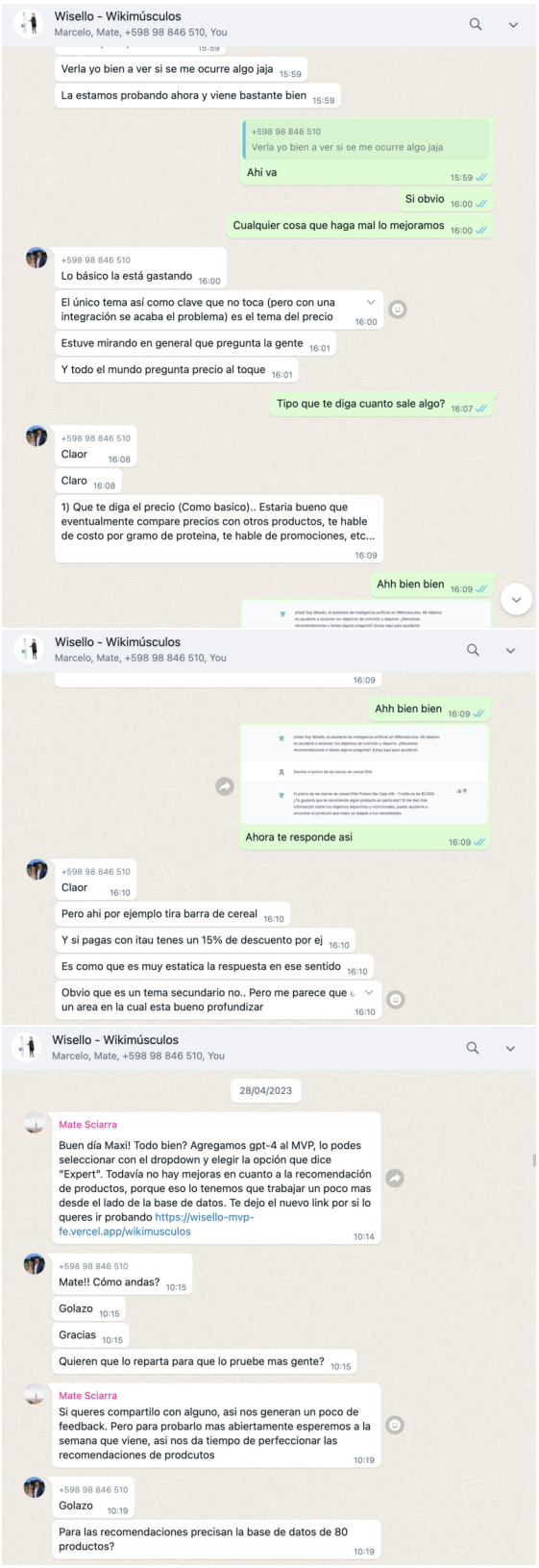
15.10 *Lean Startup loops*

Durante la primera iteración del *loop* de Lean Startup, validamos los prototipos no funcionales con expertos del área. Entre los comentarios, destacamos los siguientes:

- *"No sabe de nada con AI aún"* (Leonardo Álvarez, Fenicio eCommerce).
- *"Si quisiéramos potenciar la venta en el canal online, como lo hace un vendedor en la tienda física, si lo hubiera en la web podría potenciar las ventas"* (Juan Teba, Farmashop).
- *"El contacto con el vendedor es clave. El vendedor puede justificarte mejor por qué comprar un colchón de tal tipo y por qué es un buen precio"* (Lucía Puentes, Divino S.A.).
- *"Con mayor acompañamiento sería más fácil vender (...) Hay un montón de puntos en donde perdés gente. Hay un montón de barreras por romper"* (Lucía Puentes, Divino S.A.).

Estos demuestran la importancia de acompañar y dar un seguimiento al usuario durante su proceso de compra, tal como lo hace un vendedor en la tienda. Además, recalcan que no se han aplicado tecnologías de IA aún, evidenciando una oportunidad para hacerlo.

En las siguientes iteraciones, el feedback provino de la gente de Wikimúsculos. Si bien gran parte del mismo fue discutido en sesiones presenciales (no hay registro del mismo), adjuntamos a continuación algunas evidencias del ida y vuelta con Wikimúsculos a lo largo de los ciclos de Lean Startup:


Wisello - Wikimúsculos
Marcelo, Mate, +598 98 846 510, You
Verla yo bien a ver si se me ocurre algo jaja 15:59
La estamos probando ahora y viene bastante bien 15:59
+598 98 846 510
Verla yo bien a ver si se me ocurre algo jaja
Ahi va 15:59
Si obvio 16:00
Cualquier cosa que haga mal lo mejoramos 16:00
+598 98 846 510
Lo básico la está gastando 16:00
El único tema así como clave que no toca (pero con una integración se acaba el problema) es el tema del precio 16:00
Estuve mirando en general que pregunta la gente 16:01
Y todo el mundo pregunta precio al toque 16:01
Tipo que te diga cuanto sale algo? 16:07
+598 98 846 510
Claor 16:08
Claro 16:08
1) Que te diga el precio (Como basico).. Estaria bueno que eventualmente compare precios con otros productos, te hable de costo por gramo de proteina, te hable de promociones, etc... 16:09
Ahh bien bien 16:09
Wisello - Wikimúsculos
Marcelo, Mate, +598 98 846 510, You
16:09
Ahh bien bien 16:09
Ahora te responde asi 16:09
+598 98 846 510
Claor 16:10
Pero ahi por ejemplo tira barra de cereal 16:10
Y si pagas con itau tenes un 15% de descuento por ej 16:10
Es como que es muy estatica la respuesta en ese sentido 16:10
Obvio que es un tema secundario no.. Pero me parece que un area en la cual esta bueno profundizar 16:10
Wisello - Wikimúsculos
Marcelo, Mate, +598 98 846 510, You
28/04/2023
Mate Sciarra
Buen día Maxi! Todo bien? Agregamos gpt-4 al MVP, lo podes seleccionar con el dropdown y elegir la opción que dice "Expert". Todavía no hay mejoras en cuanto a la recomendación de productos, porque eso lo tenemos que trabajar un poco mas desde el lado de la base de datos. Te dejo el nuevo link por si lo queres ir probando https://wisello-mvp-fe.vercel.app/wikimusculos 10:14
+598 98 846 510
Mate!! Cómo andas? 10:15
Golazo 10:15
Gracias 10:15
Quieren que lo reparta para que lo pruebe mas gente? 10:15
Mate Sciarra
Si queres compartilo con alguno, asi nos generan un poco de feedback. Pero para probarlo mas abiertamente esperemos a la semana que viene, asi nos da tiempo de perfeccionar las recomendaciones de prodcutos 10:19
+598 98 846 510
Golazo 10:19
Para las recomendaciones precisan la base de datos de 80 productos? 10:19

Finalmente, en la última iteración con Plexo, obtuvimos como feedback la capacidad de personalización que tenía el asistente sobre textos estáticos, el *engagement* sorprendente de los usuarios con el mismo, y cómo este aumenta la efectividad (productividad) tras realizar búsquedas en grandes fuentes de información (en este caso, guías).

Las métricas de uso del mismo se pueden encontrar en la sección del MVP Plexo 3.2.2.3.