Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, 23 de julio de 2024

Autor: Gerson David Zambrano Barco

Título del Trabajo de Grado: "Plataforma tecnológica para habilitar la venta consultiva de un producto digital con inteligencia artificial"

Director: MSc. Alexander Chaparro Romero

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

Alexander Chaparro Romero

Firma del Director del Trabajo de Grado

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Ingeniería de Sistemas y Computación.
Proyecto de Grado.

Plataforma tecnológica para habilitar la venta consultiva de un producto digital con inteligencia artificial.

Gerson David Zambrano Barco

Director: MSc. Alexander Chaparro Romero

27 de mayo de 2024

Santiago de Cali, 27 de mayo de 2024.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Plataforma tecnológica para habilitar la venta consultiva de un producto digital con inteligencia artificial." realizado por el estudiante de Magister en Ingeniería de Software Gerson David Zambrano Barco (cod: 8980054), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

Alexander Chaparro Romero

MSc. Alexander Chaparro Romero

Santiago de Cali, 27 de mayo de 2024.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Plataforma tecnológica para habilitar la venta consultiva de un producto digital con inteligencia artificial." con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,

Gerson David Zambrano Barco
Código: 8980054

FICHA RESUMEN
TRABAJO DE GRADO MAESTRÍA

Título: Plataforma tecnológica para habilitar la venta consultiva de un producto digital con inteligencia artificial.

1. **Énfasis**: Inteligencia Artificial
2. **Tipo de proyecto**: Aplicado
3. **Área de trabajo**: Arquitectura de Software
4. **Estudiante**: Gerson David Zambrano Barco
5. **Correo electrónico**: dzambrano226@javerianacali.edu.co
6. **Dirección y teléfono**: Tv 97a # 2 - 70 T 21 Apto 1202 Bogotá - (316) 391 7256
7. **Director**: Alexander Chaparro Romero
8. **Vinculación del director**: NA
9. **Correo electrónico del director**: alexander.chaparro@javerianacali.edu.co
10. **Palabras clave**: Inteligencia Artificial, Arquitectura de Software, Procesamiento de Lenguaje Natural, Automatización, Trabajadores Digitales
11. **ODS que aplica al proyecto (Agenda 2030)**: Industria, Innovación e Infraestructura
12. **Fecha de inicio**: 16/02/2024
13. **Resumen**: Cada día la inteligencia artificial está tomando más fuerza en el mercado, los negocios se están apalancando de los beneficios que provee para brindar una mejor atención a sus clientes, automatizar diferentes procesos y generar más ingresos; Los trabajadores digitales inteligentes se convierten en un canal de atención muy importante para las empresas, debido a que a través de una conversación consultiva pueden generar más ingresos, sin embargo, adoptar un trabajador digital inteligente sin tener en cuenta una definición clara de una arquitectura tecnológica que habilite la interacción con los modelos de inteligencia artificial, puede generar problemas de integración, rendimiento y seguridad. Es por ello que este proyecto de grado plantea una solución a los desafíos descritos anteriormente mediante la implementación de una arquitectura tecnológica que habilite la adopción de un modelo de inteligencia artificial como un trabajador digital inteligente, el cual se encuentra en la capacidad de sostener una conversación consultiva con un cliente para la venta de un producto digital.

0.1. Resumen

Cada día la inteligencia artificial está tomando más fuerza en el mercado, los negocios se están apalancando de los beneficios que provee para brindar una mejor atención a sus clientes, automatizar diferentes procesos y generar más ingresos; Los trabajadores digitales inteligentes se convierten en un canal de atención muy importante para las empresas, debido a que a través de una conversación consultiva pueden generar más ingresos, sin embargo, adoptar un trabajador digital inteligente sin tener en cuenta una definición clara de una arquitectura tecnológica que habilite la interacción con los modelos de inteligencia artificial, puede generar problemas de integración, rendimiento y seguridad. Es por ello que este proyecto de grado plantea una solución a los desafíos descritos anteriormente mediante la implementación de una arquitectura tecnológica que habilite la adopción de un modelo de inteligencia artificial como un trabajador digital inteligente, el cual se encuentra en la capacidad de sostener una conversación consultiva con un cliente para la venta de un producto digital.

Palabras Clave: Inteligencia Artificial, Agentes Cognitivos, Procesamiento de Lenguaje Natural, Automatización, Trabajadores Digitales

0.2. Abstract

Digital workers today have taken great relevance in companies, achieving through artificial intelligence to automate some offer differential of products and services to their customers with customized attention with which they will find an advisor available whenever they need it. Having a software architecture that allows the integration of multiple natural language processing models will undoubtedly allow companies to take advantage of the capabilities of each one to automate their processes and offer the best user experience.

Keywords: Artificial Intelligence, Cognitive Agents, Digital Workers, Automation, Natural Language Processing (NLP), Software Architecture

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Planteamiento del Problema

Uno de los grandes cambios que dejó la pandemia del COVID-19 fue la digitalización de las empresas, lo que dio paso a que la mayoría de transacciones que se realizan hoy en día sean por medios electrónicos. Es por eso que canales tradicionales que a mediados del 2017 seguían siendo populares para los colombianos (Lopez, 2017), como los canales telefónicos, oficinas, correos electrónicos, entre otros, empleados por las entidades financieras para ofrecer los productos financieros como los créditos de libre destino, pasaron a ser poco relevantes para los consumidores financieros.

Por lo contrario, los canales digitales como aplicaciones web, móviles, chats y chatbots tomaron mucha más relevancia y se han convertido en los canales preferidos por los consumidores financieros (Fonseca, 2020).

El problema que encontramos al utilizar canales digitales, es la falta de capacidad para brindar una asesoría financiera personalizada al consumidor que le permita encontrar el producto financiero que mejor se ajuste a sus necesidades.

Uno de los canales de atención utilizados por las empresas es el canal de los chatbos; Actualmente, podemos encontrar dos tipos de chatbots, los chatbots basados en reglas y los chatbots basados en inteligencia artificial. Los chatbots basados en reglas funcionan a manera de preguntas y respuestas interactivas, estos chatbots están programados para reconocer ciertos términos y patrones a los cuales ellos pueden responder con respuestas pre configuradas. Los chatbots basados en inteligencia artificial, utilizan sofisticadas capacidades para procesar el lenguaje natural y cognitivo, estos chatbots son capaces de reconocer el contexto de una conversación, las emociones y continuamente aprender de las conversaciones con los usuarios (Jindal and Jha, 2020).

En Colombia se estima, según el DANE (Departamento Administrativo Nacional de Estadística) que está en auge la implementación de inteligencia artificial en un 25,8 % (La Nota Económica, 2023).

Los modelos de procesamiento de lenguaje natural, también conocidos como grandes modelos lingüísticos o LLM por sus siglas en inglés, dieron inicio a una generación de chatbots interactivos, su auge fue a finales del año 2022, introduciendo un nuevo paradigma que permite generar textos que se asemejan a los producidos por los humanos.

El procesamiento de lenguaje natural (NLP) es una tecnología de Machine Learning o aprendizaje de máquinas, que brinda a las computadoras la capacidad de interpretar, manipular y comprender el lenguaje humano. El procesamiento de lenguaje natural puede resolver las diferencias entre dialectos, jergas e irregularidades gramaticales típicas en las conversaciones cotidianas (AWS, 2023a).

En Colombia, la educación financiera de los colombianos es baja, por lo que la mayoría de consumidores requieren de una asesoría previa antes de adquirir un producto financiero (Bretón, 2022). Los modelos de procesamiento de lenguaje natural, abre una puerta grande para que cada vez más las empresas integren dentro de sus procesos la inteligencia artificial como uno de sus aliados a la hora de ofrecer valor al mercado, aprovechando todos sus beneficios para pasar de la asesoría a la venta.

Por lo anterior, con este proyecto se busca, diseñar una arquitectura de referencia en la nube de AWS, que permita integrar un modelo de procesamiento de lenguaje natural para construir un trabajador digital (Digital Worker); Este trabajador digital estaría en la capacidad de sostener una conversación con un posible cliente sobre el producto digital llamado crédito de libre destino.

1.2. Objetivos

1.2.1. Objetivo General

Diseñar una arquitectura de referencia en la nube de AWS que se integre con un modelo de procesamiento de lenguaje natural para construir un trabajador digital que responda efectivamente al menos el 99 % de las preguntas de una conversación consultiva con un posible cliente sobre el producto de libre inversión de una entidad financiera antes del 31 de julio de 2024.

1.2.2. Objetivos específicos

- Definir la arquitectura de referencia en la nube de AWS.
- Desarrollar la integración con el modelo de procesamiento de lenguaje natural.
- Desarrollar la funcionalidad del trabajador digital.
- Implementar mecanismos para garantizar la disponibilidad del trabajador digital.

1.3. Alcance

1.3.1. Definición del alcance

- Diseño de una arquitectura de referencia; Entendiéndose como arquitectura de referencia, como una arquitectura genérica para un tipo de sistema en específico, la cual es utilizada como base para el diseño de arquitecturas concretas (Angelov et al., 2012); La arquitectura de referencia, contemplará unos objetivos claros, el diseño de la misma y el contexto de su aplicación.
- Implementación de un prototipo funcional integrado con el servicio de mensajería de texto de WhatsApp a través de la capa gratuita denominada Sandbox de la herramienta Twillio y los servicios de la nube de AWS que permita evaluar la arquitectura diseñada.

1.4. Justificación del trabajo de grado

El sector financiero desde hace unos años ha venido experimentando un proceso de transformación, pasando de la banca tradicional a la banca digital por medio de la implementación de las tecnologías de la información y comunicación (TIC) que apalancan su estrategia de crecimiento y de disponibilizar cada vez más a sus clientes herramientas tecnológicas que les permitan autogestionar de sus productos financieros.

Con la llegada de la pandemia del COVID-19 en la cual todos nos encontrábamos aislados, dicha transformación digital de las entidades financieras tomo más fuerza, y se potenció el uso de diferentes servicios financieros por parte de los consumidores, cada vez era más frecuente ver como las personas realizaban transacciones entre diferentes cuentas, pagos electrónicos, apertura de algunos productos financieros o solicitudes de crédito de manera digital, todo con el fin de evitar el contacto, mantenerse seguros y continuar con una vida cotidiana desde una realidad diferente a la que comúnmente estábamos acostumbrados, aprendiendo a vivir una nueva realidad.

Sin embargo, a medida que todo fue retornando a la normalidad, nos dimos cuenta de que no podíamos acceder a todos los servicios de manera digital y que necesitábamos atención personalizada y asesoría antes de tomar una decisión a la hora de adquirir un producto financiero.

La pandemia llevó a que muchas empresas, de todo tipo, optaran por ofrecer a sus clientes nuevas formas de gestionar sus consultas y solicitudes de forma digital. Una de las herramientas más populares para ello es el chatbot de WhatsApp. Estos chatbots son como un asistente virtual que integra el mismo concepto de respuesta de voz interactiva (IVR), con la diferencia de que responde a las preguntas de los clientes a través de mensajes de texto.

La respuesta de voz interactiva (IVR), consiste en un sistema telefónico automatizado que permite a sus clientes elegir entre las opciones del menú de voz e interactuar mediante la voz y el teclado numérico (AWS, 2023b).

El anterior concepto se utiliza en los Chatbots de WhatsApp implementados actualmente por muchas compañías, en los cuales se presenta al usuario por medio de mensaje de texto a través de la aplicación de mensajería WhatsApp, un menú con diferentes opciones con las cuales el usuario puede ir interactuando y dentro de cada opción se puede presentar otro menú o puede que se soliciten algunos datos para realizar validaciones o verificación de identidad para acceder al servicio que está solicitando.

A partir del año 2022, con la puesta en marcha por parte de la empresa OpenAI de manera gratuita de la herramienta llamada ChatGPT, la cual permite interactuar a manera de preguntas y respuestas sobre la mayoría de temas por medio de un Chat en la Web con el modelo de inteligencia artificial denominado GPT 3.5, se popularizó el uso de esta y otras herramientas capaces de interactuar con este modelo de inteligencia artificial.

Uno de los tantos usos que tienen los modelos de procesamiento de lenguaje natural, es el de crear lo que se conoce hoy en día como "Trabajadores Digitales"que, como lo explica la empresa IBM en su página oficial dentro del artículo "The Guide to Digital Worker Technology"; Los trabajadores Digitales, también conocidos como empleados digitales, son mano de obra basada en software que pueden ejecutar de forma independiente partes significativas de procesos complejos e integrales

utilizando una serie de habilidades (IBM, 2023).

Adicional a esto, menciona algunas de las cualidades de los trabajadores digitales como: La ejecución de pasos específicos dentro de procesos complejos, trabajo a través de sistemas que rompen los silos y dan un mejor soporte para la toma de decisiones y por último el entendimiento del lenguaje humano por medio de preguntas y respuestas en lenguaje natural (IBM, 2023).

El poder contar con un trabajador digital dentro de una entidad financiera genera un impacto social y económico; Desde el punto de vista social, genera un impacto en la educación financiera de los consumidores y en la concientización de los mismos a la hora de elegir un producto financiero que más se ajuste a sus necesidades por medio de las conversaciones consultivas generadas a manera de preguntas y respuestas con un objetivo en específico.

El impacto desde el punto de vista económico que tiene un trabajador digital, afecta tanto al consumidor financiero (Cliente) y a la entidad financiera; Por parte de la entidad financiera le brinda la capacidad de incrementar la conversión y venta de sus productos financieros debido a que disponibiliza un canal de atención capaz de manejar y sostener una conversación con un potencial cliente las veinticuatro (24) horas, los siete (7) días de la semana; Por el lado de los consumidores financieros, estos pueden acceder al producto financiero que más se ajuste a sus necesidades, siempre y cuando cumplan los requisitos mínimos para acceder a este.

Adicionalmente, contar con una arquitectura base o de referencia, desde el punto de vista tecnológico genera un impacto alto en lo relacionado con él reusó o reutilización de componentes de software, ya que permite, a partir de este, construir trabajadores digitales expertos en áreas de conocimiento especificas los cuales atenderán necesidades específicas.

1.5. Metodología

La metodología empleada para refinar la arquitectura propuesta fue la metodología de diseño de arquitectura basada en atributos de calidad (ADD), combinada con la metodología Kanban. Esta última permitió la organización del trabajo ejecutado y facilitó el seguimiento del estado de las tareas asignadas. El enfoque ADD se centró en identificar y abordar los atributos de calidad relevantes para el sistema en desarrollo, asegurando así que la arquitectura resultante cumpla con los requisitos no funcionales establecidos. La metodología Kanban, por otro lado, facilitó la visualización del flujo de trabajo, la asignación de tareas y la gestión eficiente de los recursos disponibles. La combinación de estas metodologías proporcionó un marco sólido para el proceso de refinamiento arquitectónico, promoviendo la claridad, la colaboración y la eficacia en el desarrollo del proyecto.

1.5.1. Metodología Kanban

El método Kanban recibe su nombre del uso de mecanismos de señalización visual Kanban para controlar el trabajo en curso para productos de trabajo intangibles (Agile Alliance, 2024).

Para la gestión de este proyecto, se definió utilizar metodología Kanban. Siguiendo las etapas de esta metodología, se realizaron las siguientes definiciones.

1.5.1.1. Descripción del proceso

Introducción

En este documento, se presenta la aplicación de la metodología Kanban para la documentación del diseño arquitectónico que habilita el despliegue de un trabajador digital con inteligencia artificial. La metodología Kanban se utiliza para gestionar y visualizar el flujo de trabajo, proporcionando una manera eficiente de planificar, priorizar y realizar un seguimiento de las tareas relacionadas con el diseño arquitectónico.

Tablero Kanban

El tablero Kanban se utiliza para representar visualmente el flujo de trabajo y el estado actual de las tareas de diseño arquitectónico (ver Figura 1.1). Se organiza en columnas que representan los diferentes estados por los que pasan las tareas, desde la planificación hasta la finalización.

Columnas del Tablero

- **BACKLOG:** Esta columna contiene todas las tareas pendientes de diseño arquitectónico que aún no se han comenzado. Las tareas en esta columna se priorizan según su importancia y urgencia.
- **TO DO:** En esta columna, se ubican las actividades que están listas para trabajar.
- **IN PROGRESS:** Las actividades que aparecen en esta columna, son actividades las cuales se encuentran en curso, es decir, están siendo desarrolladas o se viene adelantando un proceso.

- **COMPLETE:** Una vez se finalice una actividad, la cual previamente se encontraba en la columna "IN PROGRESS", se procede a cambiar el estado de la actividad a un estado "COMPLETE", es por ello, que en esta columna del tablero Kanban, registran las actividades que se trabajaron y se completaron en su totalidad.

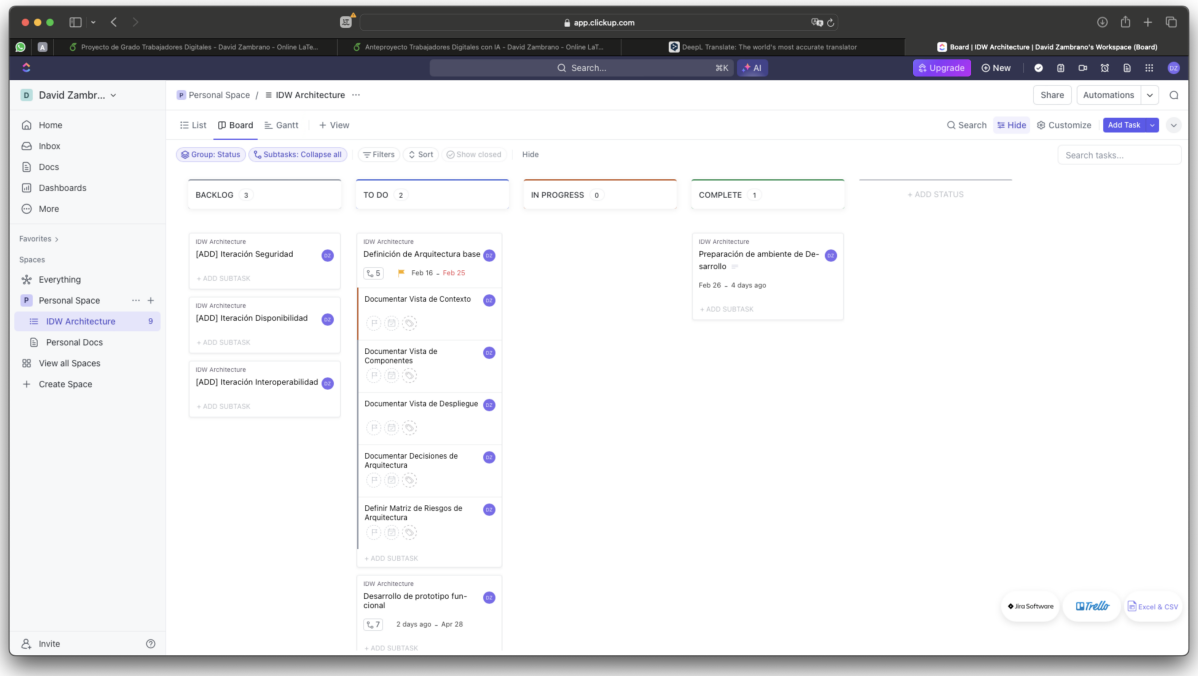


Figura 1.1: Tablero Kanban con actividades en curso.

Definición de las tareas

En cuanto a la definición y la documentación de las actividades identificadas y mapeadas dentro del tablero Kanban, se agregaron todos lo detalles relacionados con la actividad a trabajar, con el fin de tener mayor claridad y entendimiento de la misma a la hora de trabajar en ella.

Como se evidencia en la Figura 1.2, se ha establecido para cada actividad una serie de elementos definitorios. Estos comprenden un título descriptivo, indicando la naturaleza de la actividad, así como una fecha de inicio y una fecha de finalización estimada para delimitar su duración. Además, se incluye un campo para indicar el estado actual de la actividad, especificando si está en proceso, completada o pendiente, entre otros posibles estados. Asimismo, se asigna a una persona responsable que será encargada de llevar a cabo dicha actividad. Por último, se contempla una descripción detallada que explica lo que se espera lograr o realizar en el transcurso de la actividad, lo cual servirá como criterio para determinar su conclusión o completitud.

1.5.1.2. Límite de Trabajo en Progreso (WIP):

A continuación se describe el límite de trabajo en progreso establecido para este proyecto.

- **TO DO**: Máximo 5 tareas.
- **IN PROGRESS**: Máximo 2 tareas.

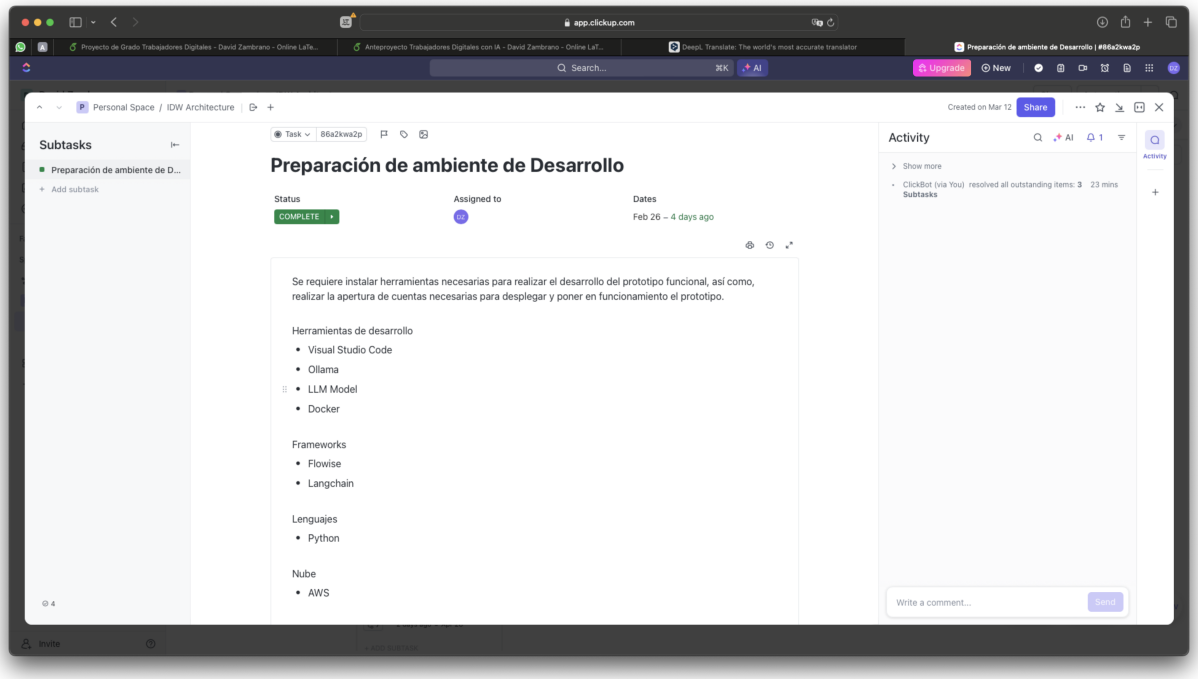


Figura 1.2: Descripción de una actividad mapeada en el tablero Kanban.

1.5.1.3. Procedimiento para la gestión de tareas

- **Creación de tareas**: Cualquier miembro del equipo puede crear una tarjeta en la columna "BACKLOG"; Cada tarea debe tener una descripción clara y unos criterios de aceptación.
- **Movimiento de tareas**: Los miembros del equipo pueden mover las tareas en las diferentes columnas del tablero Kanban respetando los límites de trabajo en progreso(WIP).
- **Actualización de las tareas**: El equipo debe actualizar el estado de las tareas a medida que estas avanzan, esto permite ver el avance del proyecto y el estado actual del mismo.

1.5.1.4. Finalización de las tareas

En la siguiente figura, se evidencia la finalización de todas las tareas definidas para el desarrollo de este proyecto de grado de forma satisfactoria, la metodología Kanban permitió guiar las etapas de diseño de la arquitectura y el desarrollo del prototipo funcional utilizado como mecanismo de evaluación para la arquitectura diseñada.

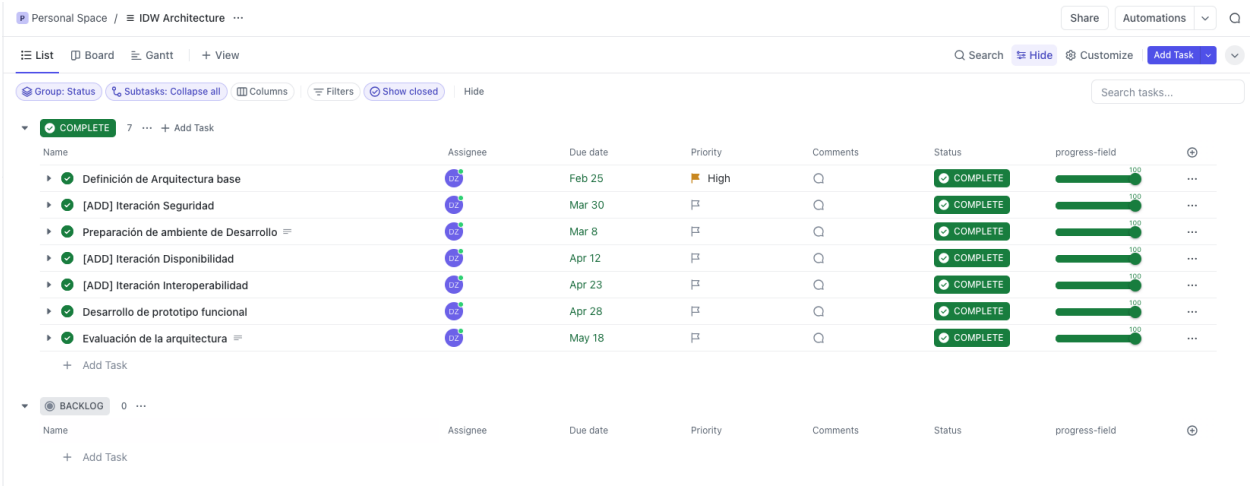


Figura 1.3: Actividades finalizadas en el tablero Kanban.

1.5.2. Diseño de Arquitectura Basado en Atributos (ADD) 3.0

Antes de empezar a diseñar con ADD (o con cualquier otro método de diseño), hay que pensar qué se está haciendo y por qué. Aunque esta afirmación pueda parecer obvia, la clave está, como siempre, en los detalles. Estas preguntas sobre él "qué" y él "por qué" se denominan impulsores arquitectónicos (Architectural Drivers). Como se muestra en la Figura 1.4, estos impulsores incluyen un propósito de diseño, atributos de calidad, funcionalidad primaria, preocupaciones arquitectónicas y restricciones. Estas consideraciones son fundamentales para el éxito del sistema y, como tales, dirigen y dan forma a la arquitectura (Cervantes and Kazman, 2016). El método de diseño de arquitecturas basadas en atributos es un método sistemático el cual se ejecuta paso a paso para diseñar una arquitectura de software que satisfaga los atributos de calidad seleccionados (Wojcik et al., 2006). Este método consta de siete pasos, los cuales se muestran dentro de la figura 1.4.

La aplicación de esta metodología en este proyecto de grado se encuentra en el Capítulo 4.

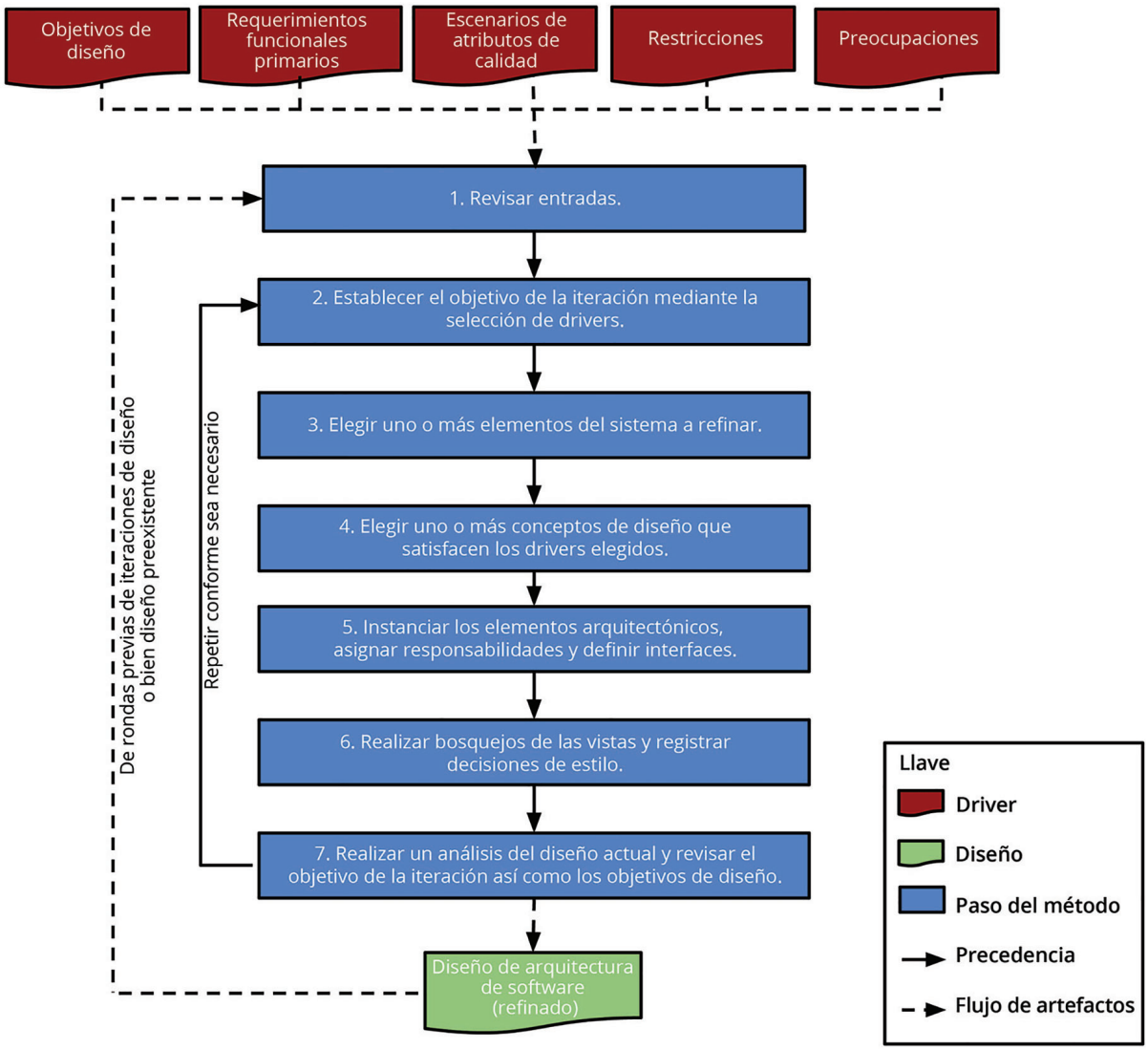


Figura 1.4: Pasos y Artefactos de la metodología de ADD (Cervantes, 2024).

1.6. Relevancia e impacto

Desde finales del año 2022, con el anuncio de la empresa OpenAI, se puso a disposición del público la herramienta denominada ChatGPT de forma gratuita (Abdullah et al., 2022). Esta herramienta es un chatbot basado en inteligencia artificial, tiene la capacidad de responder preguntas sobre diversos temas utilizando el lenguaje natural como formato de salida. Este acontecimiento marcó un hito importante en el campo de la inteligencia artificial, ya que amplió el acceso y la utilidad de esta tecnología para una audiencia más amplia. La disponibilidad de ChatGPT ha ge-

nerado un mayor interés y reconocimiento en torno a las capacidades y aplicaciones de la IA en la comunicación y el procesamiento del lenguaje natural.

El objetivo principal de este proyecto, es diseñar una arquitectura de referencia en la nube para desarrollar un chatbot integrado con un modelo inteligencia artificial, por lo cual, genera un impacto significativo en cuanto a la eficiencia, seguridad, interoperabilidad y capacidad de respuesta de las aplicaciones empresariales.

En resumen, la implementación de una arquitectura de referencia en la nube para un chatbot integrado con un modelo de inteligencia artificial, puede transformar la forma en que las organizaciones interactúan con sus clientes en sus operaciones diarias. Este proyecto contribuye en la automatización inteligente y la mejora continua de las empresas.

1.7. Estructura de la tesis

La presente tesis se encuentra estructurada en varias secciones que abarcan desde la introducción y la justificación del trabajo hasta las conclusiones finales, pasando por un desarrollo exhaustivo del proyecto y su evaluación. A continuación, se describe la estructura del documento.

1. **Introducción**

   1.1 **Planteamiento del Problema**: Se expone la problemática que motiva la investigación y los objetivos que se pretenden alcanzar.

   1.2 **Objetivos**: Se detallan los objetivos generales y específicos del estudio.

   1.3 **Alcance**: Se define el alcance del trabajo, delimitando el campo de acción del proyecto.

   1.4 **Justificación del Trabajo de Grado**: Se argumenta la relevancia y pertinencia del estudio en el contexto académico y profesional.

   1.5 **Metodología**: Se describen las metodologías utilizadas para llevar a cabo la investigación, incluyendo la metodología Kanban y el Diseño de Arquitectura Basado en Atributos (ADD) 3.0.

   1.6 **Relevancia e Impacto**: Se discute la importancia del trabajo y su potencial impacto en el campo de estudio.

   1.7 **Estructura de la Tesis**: Se presenta una visión general de la organización del documento.

2. **Marco de Referencia**

   2.1 **Bases Teóricas**: Se desarrollan los conceptos teóricos fundamentales, como la arquitectura de software, inteligencia artificial, procesamiento de lenguaje natural, entre otros.

   2.2 **Estado del Arte**: Se revisan las soluciones actuales y el progreso en el área de estudio.

3. **Low-code/No-Code (LC/NC)**

3.1 **Beneficios y Limitaciones**: Se examinan las ventajas y desventajas de utilizar plataformas LC/NC en el desarrollo de soluciones tecnológicas.

4. **Diseño Basado en Atributos de Calidad (ADD)**

   4.1 **Contexto y Requerimientos de Negocio**: Se analizan los requisitos y el entorno de negocio.

   4.2 **Atributos de Calidad**: Se identifican y describen los atributos de calidad relevantes.

   4.3 **Escenarios de Atributos de Calidad**: Se plantean escenarios específicos para evaluar la calidad en términos de seguridad, fiabilidad y compatibilidad.

   4.4 **Iteraciones de Diseño Basado en Atributos de Calidad**: Se documentan las iteraciones de diseño, enfocándose en los atributos de calidad priorizados que son seguridad, fiabilidad y compatibilidad.

5. **Desarrollo del Proyecto**

   5.1 **Contexto de la Solución**: Se proporciona un contexto detallado de la solución desarrollada.

   5.2 **Requerimientos No Funcionales**: Se especifican los requisitos que no están directamente relacionados con las funcionalidades, pero son cruciales para el desempeño del sistema.

   5.3 **Prototipo Funcional**: Se describe el desarrollo y despliegue del prototipo, incluyendo la infraestructura en la nube, el uso de herramientas LC/NC, integración de bases de datos y modelos de inteligencia artificial.

6. **Evaluación de la Arquitectura**

   6.1 **Metodología de Evaluación**: Se presenta la metodología utilizada para evaluar la arquitectura del sistema.

   6.2 **Aplicación de la Metodología**: Se aplican los métodos de evaluación a la arquitectura desarrollada.

7. **Conclusiones**: Se resumen los hallazgos del estudio, se discuten las implicaciones de los resultados y se sugieren posibles líneas de investigación futura.

8. **Bibliografía**: Se incluyen todas las referencias bibliográficas utilizadas a lo largo del estudio.

9. **Apéndices**: Se anexan documentos adicionales relevantes, como escenarios de atributos de calidad, decisiones de arquitectura y pruebas realizadas sobre la arquitectura refinada.

Capítulo 2

Marco de referencia

2.1. Bases Teóricas

2.1.1. Arquitectura de Software

2.1.1.1. Definición de arquitectura de software

Según el estándar 1471 del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE), se define a la arquitectura de software como "La organización fundamental de un sistema, representada por sus componentes, sus relaciones entre sí y con el entorno, y los principios que guían su diseño y evolución" Maier et al. (2001).

La arquitectura de software también se puede definir según el Instituto de Ingeniería de Software (Software Engineering Institute - SEI) en el libro Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al, Addison- Wesley, 2010, como "Conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos y propiedades de ambos" Garlan et al. (2010). Así como también el libro Software Architecture in Practice (2nd edition), Bass, Clements, Kazman; Addison- Wesley 2003 define la arquitectura de software como "La arquitectura de software de un programa o un sistema de cómputo es la estructura o estructuras del sistema, comprende elementos del sistema, las propiedades externas que son visibles de estos elementos y las relaciones entre ellos" Engineering Institute (2017).

Rick Kanzman, Len Bass y Mark Klein, establecen tres principios para una arquitectura de software correcta (Kazman et al., 2006):

1. **Primer Principio:** Una arquitectura de software debe definirse en términos de elementos que sean lo suficientemente gruesos para el control intelectual humano y lo suficientemente específicos para un razonamiento significativo.

2. **Segundo Principio:** Los objetivos empresariales (y/o de misión) determinan los requisitos de los atributos de calidad.

3. **Tercer Principio:** Los requisitos de los atributos de calidad guían el diseño y el análisis de las arquitecturas de software.

2.1.1.2. Diseño de una arquitectura de software

El diseño de una arquitectura de software, involucra la toma de decisiones, el trabajo con habilidades y materiales existentes para satisfacer los requerimientos y restricciones. Cuando diseñamos

una arquitectura, tomamos decisiones para transformar los propósitos de diseño, requerimientos, restricciones y restricciones de arquitectura, también llamados Drivers de Arquitectura dentro de las estructuras, como se muestra en la Figura 2.1. Dichas estructuras, son utilizadas como una guía del proyecto (Cervantes and Kazman, 2016).

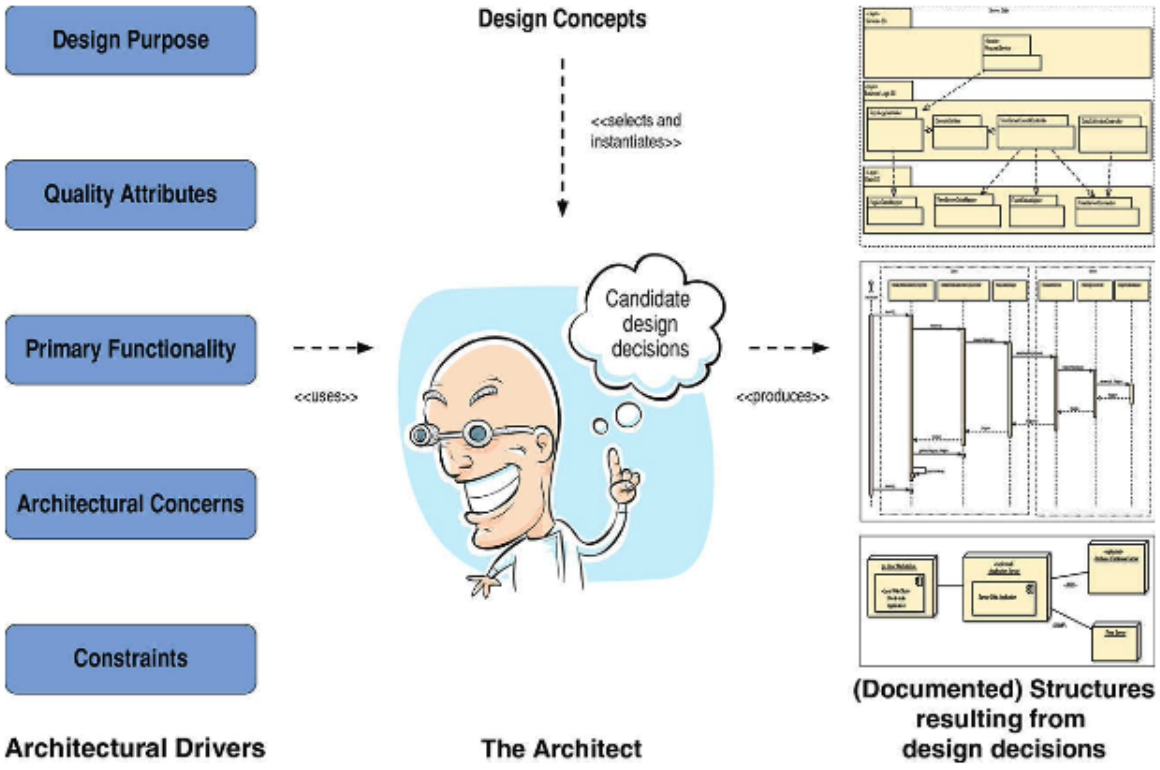


Figura 2.1: Resumen etapa de diseño de una arquitectura de software (Cervantes and Kazman, 2016).

De acuerdo a la Figura 2.1, podemos resumir el proceso de diseño de una arquitectura de software, como el uso de los Drivers de Arquitectura los cuales comprenden los propósitos de diseño, atributos de calidad, funcionalidades primarias, restricciones de arquitectura y restricciones por parte de un arquitecto de software quien apoyado de unos conceptos de diseño, produce unas estructuras como resultado de la toma de algunas decisiones de diseño.

A continuación, se da más detalle sobre los Drivers de Arquitectura basado en el libro Designing Software Architectures, A practical Approach (Cervantes and Kazman, 2016).

- **Propósito de Diseño:** Lo primero es tener claro el propósito de diseño, se debe considerar: ¿Cuándo y por qué va a realizar este diseño de arquitectura? ¿Qué objetivos empresariales preocupan más a la organización en este momento?, un propósito de diseño puede ser el

realizar un diseño arquitectónico como parte de una propuesta de un proyecto, esto sucede en la fase de viabilidad del mismo en el cual se puede presentar el diseño arquitectónico, el cual no debe ser muy detallado. Otro propósito de diseño puede ser la creación y exploración de un prototipo, en este caso, el propósito del diseño arquitectónico no está basado en crear un sistema reutilizable o que se pueda distribuir, sino más bien explorar nuevas tecnologías, poner un sistema delante de un cliente de lo más pronto posible para obtener una retroalimentación o explorar algún atributo de calidad. Por último, un propósito de diseño también se puede considerar cuando se está diseñando la arquitectura durante la etapa de desarrollo, esto puede ser cuando se está diseñando un sistema nuevo o una nueva funcionalidad para un sistema nuevo o en su defecto cuando se está refactorizando un sistema ya existente, en este caso, el propósito es el de satisfacer los requerimientos del sistema, desarrollar una guía para la construcción del sistema, y prepararse para un eventual despliegue del sistema.

- **Atributos de Calidad:** Los atributos de calidad, están definidos como las propiedades medibles o que se pueden probar en un sistema, las cuales satisfacen las necesidades de los interesados en el sistema (stakeholders). La norma internacional ISO/IEC 25010 agrupa en nueve categorías los atributos de calidad que se deben considerar al momento de hablar de calidad de software (iso2500.com, 2023), los diferentes grupos y atributos los podemos ver en la Figura 2.2. La mejor forma de discutir, documentar y priorizar los atributos de calidad es contar con conjunto de escenarios. Un escenario, en su forma más simple, describe la respuesta de un sistema ante un estímulo. Un escenario está compuesto por seis partes, el fuete del estímulo, el estímulo, el artefacto, el ambiente, la respuesta y la medida de respuesta, tal como se muestra en la Figura 2.3.

- **Funcionalidades Primarias:** Una funcionalidad es la habilidad de un sistema de hacer el trabajo para el cual fue construido. Una funcionalidad primaria se puede definir como una funcionalidad fundamental para alcanzar los objetivos empresariales que motivan el desarrollo del sistema; Otra forma de encontrar una funcionalidad primaria, es evaluando si una funcionalidad implica un alto nivel de dificultad técnica o que requiera de interacción con muchos elementos de la arquitectura.

- **Restricciones de Arquitectura:** Se consideran restricciones de arquitectura todos aquellos aspectos que se deben considerar como parte del diseño de arquitectura y que, por lo general, no son expresados como requerimientos tradicionales. Existen diferentes tipos de restricciones de arquitectura como:

  - Restricciones generales: pueden comprender la organización del código base, la puesta en marcha, el cierre, el apoyo en la entrega, así como el despliegue y en la instalación de actualizaciones.
  - Restricciones específicas: son restricciones más específicas del sistema, como el manejo de errores, las configuraciones, autenticación, autorización, administración de dependencias, estrategias de caché, entre muchas otras.

- Requerimientos internos: estos requerimientos, no están especificados de forma explícita dentro de los documentos de requerimientos. Los requerimientos internos pueden venir de la necesidad de facilitar el desarrollo, el despliegue, la operación, o el mantenimiento del sistema. También son conocidos como "requerimientos derivados".
- Problemas: son el resultado de actividades de análisis, como la revisión del diseño de arquitectura, de esta forma, esta restricción no está presente en una etapa inicial.

- **Restricciones:** Es necesario tener un catálogo de restricciones como parte del proceso de diseño de arquitectura. Una restricción puede ser alguna tecnología que sea obligatoria de usar, otros sistemas con los que el sistema en que se está diseñando necesite inter operar o integrarse, la legislación actual, algunos estándares que se deban cumplir, las habilidades y la disponibilidad del equipo de desarrollo, las fechas límite de entrega que no se pueden negociar, la compatibilidad con versiones anteriores u obsoletas de un sistema, entre otras. Una restricción es una decisión sobre la que tienes poco o ningún control un arquitecto de software, por lo cual se debe considerar el diseño de una arquitectura teniendo en cuenta las restricciones.

Figura 2.2: Atributos de calidad ISO/IEC 25010 (iso2500.com, 2023).

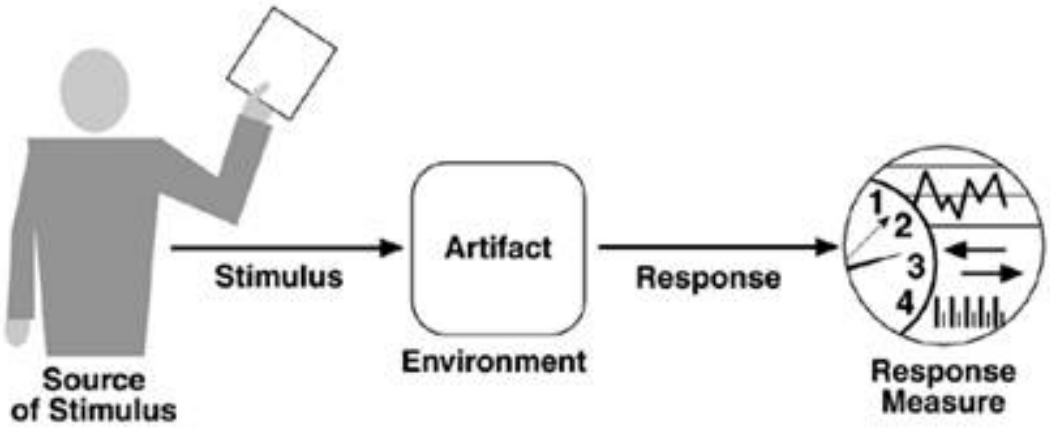


Figura 2.3: Pasos de un escenario de atributos de calidad (Cervantes and Kazman, 2016).

2.1.2. Inteligencia Artificial

La noción nació en los años 50 gracias al matemático Alan Turing. En su libro "Computing Machinery and Intelligence", plantea la cuestión de dotar a las máquinas de una forma de inteligencia. A continuación describe una prueba conocida hoy como "Test de Turing" en la que un sujeto interactúa a ciegas con otro humano y después con una máquina programada para formular respuestas con sentido. Si el sujeto no es capaz de establecer la diferencia, entonces la máquina ha superado la prueba y, según el autor, puede considerarse realmente "inteligente" (Lexcellent, 2019, p. 10).

Durante miles de años, hemos intentado comprender cómo pensamos; es decir, cómo un mero puñado de materia puede percibir, comprender, predecir y manipular un mundo mucho más grande y complicado que él mismo. El campo de la inteligencia artificial, o IA, va aún más lejos: intenta no solo comprender, sino también construir entidades inteligentes. Actualmente, la IA abarca una enorme variedad de subcampos, que van desde lo general (aprendizaje y percepción) a lo específico, como jugar al ajedrez, demostrar teoremas matemáticos, escribir poesía, conducir un coche en una calle abarrotada de gente y diagnosticar enfermedades. La IA es relevante para cualquier tarea intelectual; es realmente un campo universal (Russell and Norvig, 2010, p. 20).

Christian Lexcellent en su libro "Artificial Intelligence versus Human Intelligence" (Inteligencia artificial frente a inteligencia humana), define a la Inteligencia Artificial como un conjunto de teorías y técnicas que desarrollan programas informáticos complejos capaces de simular ciertos rasgos de la inteligencia humana (razonamiento, aprendizaje, etc.). También puede definirse como "la ciencia de diseñar máquinas capaces de hacer cosas que requieren inteligencia cuando las hacen los humanos" Marvin Lee Minski. Lexcellent (2019, p. 10)

2.1.3. Procesamiento de Lenguaje Natural

De acuerdo al libro llamado Natural Language Processing in Action de Michael Collins y Chris Manning, el procesamiento del lenguaje natural es un área de investigación de la informática y la inteligencia artificial (IA) que se ocupa del procesamiento de lenguajes naturales como el inglés o el mandarín. Por lo general, este procesamiento consiste en traducir el lenguaje natural en datos (números) que un ordenador puede utilizar para aprender sobre el mundo. Y esta comprensión del mundo se utiliza a veces para generar textos en lenguaje natural que reflejen esa comprensión (Cole et al., 2019, p. 4).

Un sistema de procesamiento del lenguaje natural suele denominarse "pipeline", ya que suele constar de varias etapas de procesamiento en las que el lenguaje natural entra por un extremo y el resultado procesado sale por el otro (Cole et al., 2019, p. 4).

2.1.4. Modelo Lingüistico de Gran Escala (Large Language Model)

El lenguaje desempeña un papel fundamental a la hora de facilitar la comunicación y la autoexpresión de los seres humanos, y del mismo modo, la comunicación toma gran relevancia para las máquinas en sus interacciones con los humanos y otros sistemas. Los grandes modelos lingüísticos (LLM) han surgido como sistemas de inteligencia artificial de vanguardia diseñados para procesar y generar texto, con el objetivo de comunicarse de forma coherente (Naveed et al., 2023).

Los modelos lingüísticos han revolucionado el procesamiento del lenguaje natural y la investigación en IA, y han permitido alcanzar logros notables en diversas tareas relacionadas con el lenguaje. Las fases de desarrollo de estos modelos han sido testigos de una búsqueda constante de modelos más grandes, estrategias de pre entrenamiento mejoradas y adaptaciones de dominios especializados (Hadi et al., 2023).

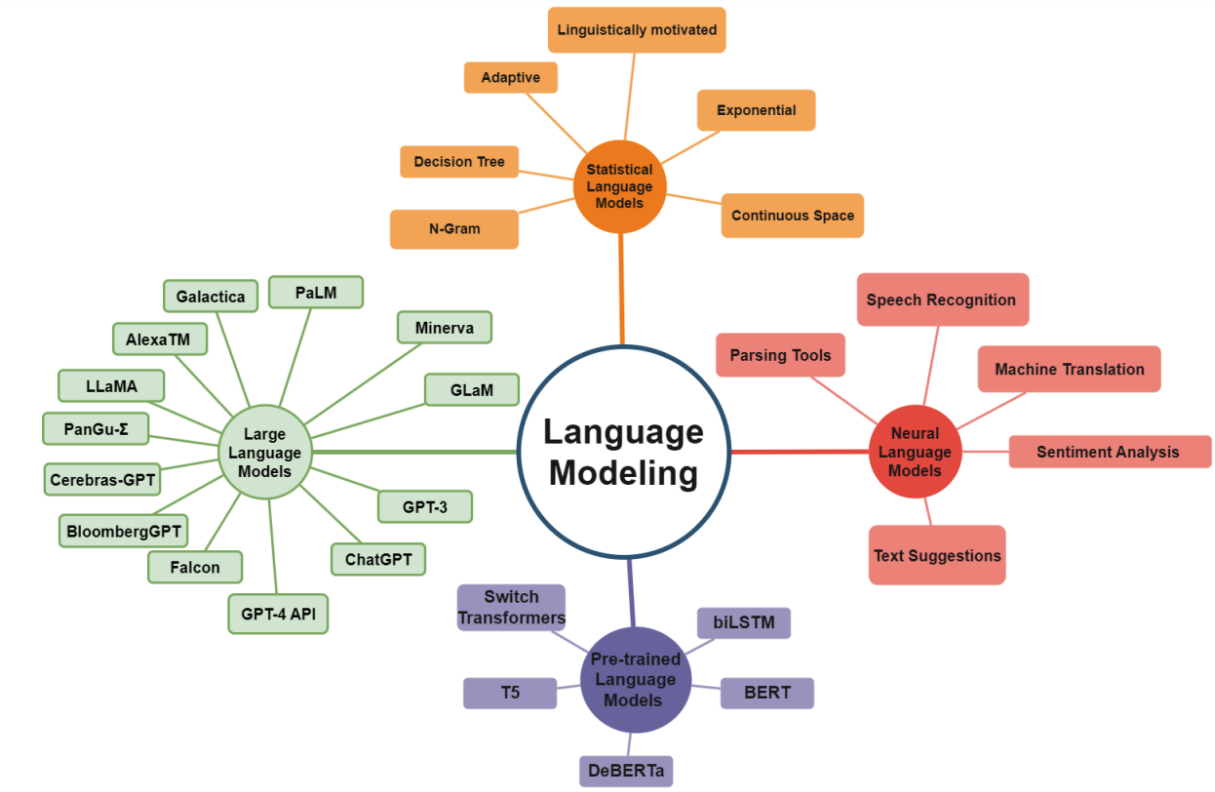


Figura 2.4: Tipos de modelos de lenguaje (Hadi et al., 2023).

2.1.5. Trabajadores o Asistentes Digitales Inteligentes

Potenciados por los recientes avances en IA, los asistentes se están convirtiendo en parte de nuestra vida cotidiana. Observamos un uso cada vez mayor de diversos asistentes digitales, por ejemplo, asistentes basados en la voz, como Amazon Alexa, o asistentes basados en texto (chatbots), como los integrados en Facebook Messenger. Se prevé que los asistentes digitales basados en IA se conviertan en un elemento clave del futuro del trabajo. Las plataformas de comunicación empresarial actuales, como Slack o Microsoft Teams, ya ofrecen muchos tipos diferentes de bot para aumentar el trabajo; Gartner predice que, para 2021, una cuarta parte de todos los trabajadores digitales utilizarán a diario un asistente virtual para empleados. Los asistentes digitales basados en IA ofrecen importantes oportunidades, pero también podrían convertirse en una amenaza. Por un lado, se espera que se hagan cargo de las tareas rutinarias de los humanos y liberen tiempo y recursos para tareas más exigentes (Maedche et al., 2019).

Los asistentes digitales informatizados pueden responder preguntas y traducir documentos, pero no lo bastante bien como para hacer el trabajo sin intervención humana; lo mismo ocurre con los coches autoconducidos (Benanav, 2019).

2.1.6. Aprendizaje de Máquinas (Machine Learning)

El aprendizaje automático es una rama de la informática que, en términos generales, pretende que los ordenadores "aprendan" sin ser programados directamente. Tiene su origen en el movimiento de inteligencia artificial de los años 50 y hace hincapié en objetivos y aplicaciones prácticas, en particular la predicción y la optimización. Los ordenadores "aprenden" en el aprendizaje automático, mejorando su rendimiento en las tareas a través de la "experiencia" (Bi et al., 2019).

2.1.7. Retrieval-Augmented Generation (RAG)

Los modelos de lenguaje extendido (LLM) como GPT de OpenAI, Llama de meta, Gemini de Google, entre otros, cuentan con muchas capacidades, sin embargo, también se enfrentan a grandes retos, dentro de ellos la generación de contenido incorrecto, conocido como "alucinaciones", el conocimiento desactualizado y los procesos de razonamiento no transparentes ni rastreables, es por esto, que estos modelos demuestran tener notables limitaciones, particularmente con el manejo de dominios específicos o con el manejo de consultas altamente especializadas (Gao et al., 2023).

La técnica RAG (Retrieval-Augmented Generation) quien fue introducida por Lewis (Lewis et al., 2020) a mediados del año 2020, establece un nuevo paradigma dentro de los modelos lingüísticos de gran escala (LLM), mejorando las tareas generativas y abordando los problemas de generación de contenido incorrecto conocido como "Alucinaciones" (Gao et al., 2023).

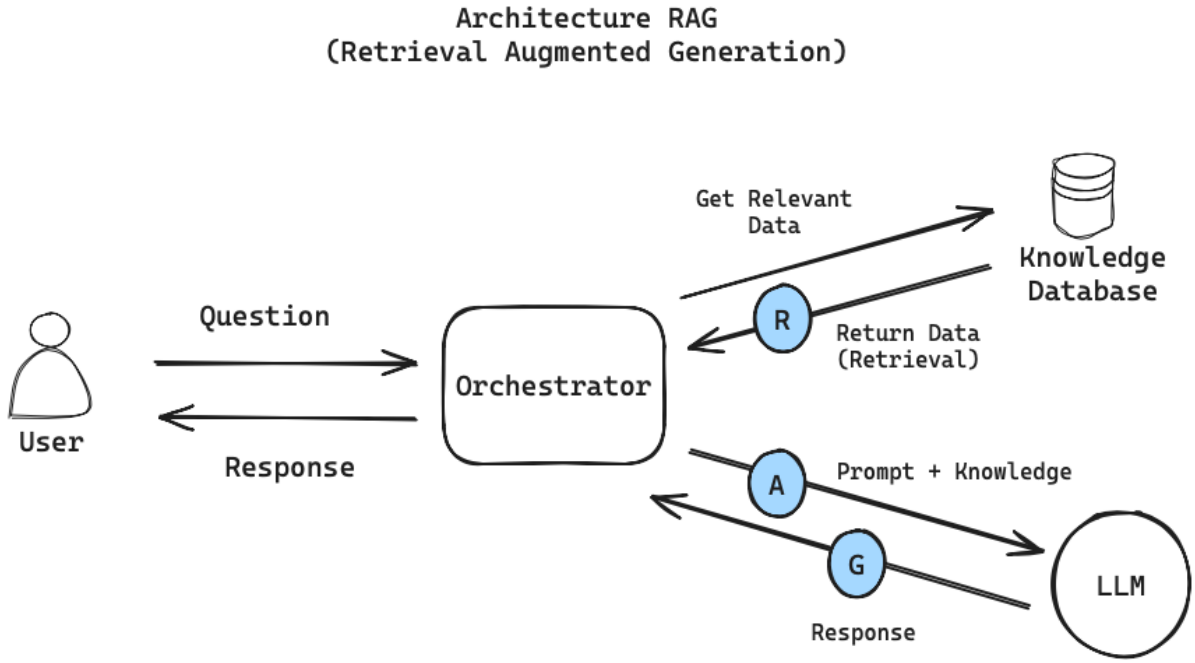


Figura 2.5: Diseño de arquitectura aplicando la técnica RAG.

2.2. Estado del Arte

A continuación se presenta un análisis de algunas soluciones tecnológicas que en la actualidad se encuentran en el mercado, sumado a algunas arquitecturas de integración de modelos de Inteligencia Artificial basados en la interpretación y generación de texto.

2.2.1. Soluciones con IA

- **OpenDialog**: OpenDialog es una compañía enfocada en asistentes conversacionales impulsados por inteligencia artificial, ofrece múltiples soluciones con las cuales los clientes haciendo uso de Chatbots y Asistentes Virtuales pueden automatizar procesos de negocio complejos. OpenDialog se especializa en la generación de asistentes virtuales que atienden las áreas de Seguros y el área de la Salud; Dentro de la solución relacionada con Seguros, ofrece servicios para prevención de fraude, automatizar el proceso de cotización de compra, brinda mayor conocimiento del cliente, entre otras; Por el lado de seguros, ofrece soluciones para realizar una atención a los pacientes 24/7, triaje o clasificación de los pacientes dependiendo de sus síntomas, programación y recordatorios, entre otros (OpenDialog AI, 2023).

- **ChatGPT**: ChatGPT es la solución provista por OpenAI para interactuar principalmente con su modelo de procesamiento de lenguaje natural llamado GPT. ChatGPT interactúa de forma conversacional, puede adoptar varias facetas por medio del uso de prompts; Se puede considerar como Prompt cualquier forma de texto, pregunta, información o código que le dé a entender a una inteligencia artificial que debe generar una respuesta ante un propmt, por medio de los propmts, se puede conseguir que ChatGPT se comporte como un asistente virtual capaz de sostener una conversación relacionada con un tema en específico (OpenAI, 2023).

- **Google Assistant y Amazon Alexa**: Estos asistentes utilizan modelos de inteligencia artificial (IA) para interpretar comandos de voz, realizar tareas, proporcionar información y controlar dispositivos inteligentes (Hoy, 2018).

Capítulo 3

Low-code/No-Code (LC/NC)

3.1. Introducción

El mundo del desarrollo de Software, se conoce como Low-code/No-code (LC/NC) al método que permite a los usuarios crear aplicaciones haciendo uso de muy pocas líneas de código o de ninguna línea de código (Yan, 2021).

El objetivo que tiene el LC/NC es involucrar a las personas en el mundo del desarrollo de software, sin importar que estas tengan conocimiento previo en el desarrollo de software. Esto se hace posible con la introducción de plataformas de Low-code/No-Code. Estas plataformas de Low-code/No-Code permiten a las empresas crear diferentes aplicaciones sin tener que escribir una sola línea de código o sin tener algún desarrollador experto encargado de la construcción de la aplicación (El Kamouchi et al., 2023).

Este enfoque reduce de manera significativa el tiempo y el costo del desarrollo de software. Esto dio lugar a que apareciera un nuevo perfil denominado "Desarrolladores de Ciudad" (Carroll et al., 2021), los Desarrolladores de Ciudad o Citizen Delevopers (CD), son personas que no tienen una formación en desarrollo de software, sin embargo, tienen un alto interés en crear aplicaciones de software para solucionar problemas específicos del negocio (El Kamouchi et al., 2023).

Algunas empresas que han adoptado LC/NC son: Microsoft, Salesforce, Google, Appian, OutSystem, Betty Blocks y Retool. La empresa Microsoft, con su producto Microsoft PowerApps, conduce la transformación tecnológica de las compañías al reducir el costo de desarrollo de aplicaciones y al incrementar la eficiencia general de las empresas. Un estudio demostró que las empresas que esperan a que los departamentos de las tecnologías de la información y comunicación (TIC) ofrezcan soluciones ya no son viable y los beneficios de Microsoft PowerApps incluyen (Carroll et al., 2021):

- 188 % Retorno de la inversión (ROI) después de tres años.
- 74 % de reducción en los costos de desarrollo.
- Mejora media de 3,2 horas semanales en la productividad de los empleados de la línea de negocio.

3.2. Beneficios de Low-code/No-code

A Continuación, se detallan algunos beneficios de Low-code/No-code (Yan, 2021):

- **Velocidad:** Low-code/No-code, permite construir nuevas aplicaciones de forma visual, lo cual acelera el desarrollo y provee al negocio la capacidad de usar el prototipado para nuevos requerimientos y prueba de sus funcionalidades. La mayoría de las empresas que utilizan plataformas de LC/NC, reportan que el despliegue de sus aplicaciones, se realiza de 5 a 10 veces más rápido, así mismo, según la encuesta de Outsystems (2019), las empresas que implementan LC/NC, estarían en la capacidad de liberar el 68 % de sus aplicaciones web en un periodo de 4 meses, por el contrario, las empresas que no implementan LC/NC podrían liberar el 57 % de sus aplicaciones web utilizando la misma cantidad de tiempo.

- **Desarrolladores de Ciudad:** Los desarrolladores de ciudad o "Citizen Developers", son personas que no tienen una formación en desarrollo de código o en áreas de las tecnologías de la información. Estas personas tienen un bajo conocimiento en desarrollo de software, pero tiene un algo conocimiento y habilidades en otras áreas.

- **Seguridad:** Durante la transformación digital, el número limitado de profesionales de las tecnologías de la información (TI) hace que sea difícil para los profesionales de TI gestionar la demanda de aplicaciones empresariales. En tal situación, las empresas que no pueden obtener soluciones del departamento de TI o de los profesionales de TI buscarían soluciones de terceros sin la supervisión del departamento de TI, lo que se conoce como "Shadow IT". "Shadow IT" puede poner en peligro la seguridad y privacidad de TI de una organización, ya que la organización no tendrá conocimiento ni control de las aplicaciones, como los detalles de implementación, la gestión de datos y las amenazas a la seguridad de TI. Las plataformas de LC/NC autorizadas por el departamento de TI, pueden resolver este riesgo denominado "Shadow IT"

- **Mantenibilidad:** De acuerdo al estudio realizado por OutSystems (2019) se estima, que el 65 % del desarrollo de proyectos de aplicaciones está relacionado con el mantenimiento, y que el porcentaje restante (35 %) está relacionado con innovación. Sin embargo, las actividades de mantenimiento, toman aproximadamente el 75 % de la capacidad de los recursos de TI de las organizaciones. OutSystems (2019) también encontró que las empresas que utilizan el desarrollo con la técnica Low-code/No-code, pueden tener un 40 % más de proyectos de innovación que de mantenimiento. Low-code/No-code, reduce la complejidad y dificultad de mantener una aplicación, ya que, quienes realizan el mantenimiento, solo requieren mantener unas pocas líneas de código, lo cual hace a las aplicaciones Low-code/No-code más mantenibles.

3.3. Limitaciones LC/NC

Estas son algunas limitantes de utilizar Low-Code/No-Code (Yan, 2021):

- **Flexibilidad en la personalización:** Los bloques de construcción visual en plataformas LC/NC son implementadas previamente y funcionan en la mayoría de los casos. Sin embargo, en algunos casos, el tener estos bloques previamente construidos, hace que la solución sea

menos personalizable que cuando dichos componentes son construidos por desarrolladores de la manera tradicional.

- **Escalabilidad:** La mayoría de plataformas actuales de LC/NC, son utilizadas para construir aplicaciones a pequeña escala, mientras que rara vez se utilizan para construir aplicaciones empresariales de gran escala o aplicaciones complejas o cruciales por su limitante en la escalabilidad.

- **Preocupaciones de seguridad:** Dado que la mayoría de los usuarios de plataformas Low/No-Code apenas personalizan o no pueden personalizar las aplicaciones, deben confiar plenamente en que los servicios no generen vulnerabilidades que provoquen errores o fugas de datos. Si las organizaciones dependen de sus proveedores de plataformas Low/No-Code, sus datos podrían ser vulnerables a violaciones de datos, ya que la seguridad de los datos y el código fuente no están totalmente controlados por las organizaciones.

- **Bloqueo de Proveedores (Vendor lock-in):** El bloqueo de proveedores, se trata de la dependencia que genera una empresa para desarrollar sus productos y servicios, exclusivamente con un proveedor, limitando a los clientes a cambiar de proveedor para ofrecer sus productos y servicios. En cuanto a Low-code/No-code, se puede presentar el caso de vendor lock-in con las plataformas de LC/NC.

Capítulo 4

Diseño Basado en Atributos de Calidad (ADD)

4.1. Contexto y requerimientos de negocio

Una empresa del sector financiero cuya oferta de valor son los productos financieros como créditos de libre destino o libre inversión, busca ampliar sus canales de atención y venta utilizando la aplicación de mensajería de WhatsApp para llegar a un mayor número de clientes potenciales y desembolsar más créditos de libre destino.

- **Objetivo**: Diseñar una plataforma tecnológica con inteligencia artificial (IA) que habilite un trabajador digital inteligente para brindar asesoría personalizada y ayudar a concretar la venta de un producto financiero utilizando el nuevo canal de venta de WhatsApp.
- **Usuario(s)**:
  - Personas naturales que buscan obtener un crédito de libre destino para diversos fines, como financiar proyectos personales, consolidar deudas, realizar inversiones, entre otros.
- **Requerimientos**:
  - La comunicación entre el cliente y el trabajador digital inteligente debe estar cifrada.
  - El trabajador digital inteligente debe ser accesible desde cualquier dispositivo que soporte la aplicación de mensajería instantánea WhatsApp.
  - El canal de venta digital debe estar disponible 24/7.
  - El trabajador digital inteligente debe responder las consultas de los clientes de manera oportuna.
  - El trabajador digital inteligente debe poder interactuar con los clientes de manera natural y fluida.
  - El trabajador digital inteligente debe considerar las limitaciones del producto financiero con tal de no llevar a la venta a un cliente que no cumpla con los requisitos mínimos de este.

4.2. Atributos de Calidad

Un atributo de calidad (QA) es una propiedad de un sistema que se puede medir o probar y que se utiliza para alinear el comportamiento del sistema con lo esperado por los interesados en él.

La norma internacional ISO/IEC 2510 establece qué "La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios, aportando de esta manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y sub características." iso2500.com (2023).

De acuerdo al contexto y requerimientos de negocio, se identificaron las siguientes características o atributos de calidad de forma explícita como implícita y se clasificaron según las categorías de la norma internacional ISO/IEC 2510.

Categoría	**Característica**	**Implícito/Explicito**	**Descripción**
Fiabilidad	Disponibilidad	Explicito	El canal de venta digital debe estar disponible 24/7.
	Tolerancia a fallos	Implícito	El sistema debe detectar y gestionar los errores que se presenten.
Compatibilidad	Interoperabilidad	Explícito	El trabajador digital inteligente debe ser accesible desde cualquier dispositivo que soporte la aplicación de mensajería instantánea WhatsApp.
		Implícito	El sistema debe comunicarse de forma segura e intercambiar información con el sistema de base de datos, almacenamiento de archivos, el modelo de procesamiento de lenguaje natural y sistema de mensajería.
Seguridad	Confidencialidad	Implícito	El repositorio de datos debe contar con la capacidad de cifrar los datos almacenados en reposo.

Categoría	Característica	Implícito/Explicito	Descripción
	Integridad	Explicito	La comunicación entre el cliente y el trabajador digital inteligente debe estar cifrada.
	Autenticidad	Implícito	El consumo de las APIs del sistema deber únicamente debe ser accedido por usuarios autorizados.
Mantenibilidad	Reusabilidad	Implícito	El sistema debe contar con componentes reutilizables.
	Capacidad de ser modificado	Implícito	El sistema debe permitir la actualización de las condiciones del producto financiero sin que esto genere una afectación o degradación del servicio.
Capacidad de Interacción	Asistencia al usuario	Explícito	El trabajador digital inteligente debe poder interactuar con los clientes de manera natural y fluida.

Tabla 4.1: Atributos o Características de calidad identificados en el contexto y requerimientos de negocio categorizados según la norma internacional ISO/IEC 25010 (iso2500.com, 2023).

Para el diseño de la arquitectura, se priorizó algunos de los atributos de calidad mencionados anteriormente, teniendo como criterio principal seleccionar los atributos fundamentales para el funcionamiento adecuado del sistema frente a lo descrito en el contexto y requerimientos de negocio. A continuación se muestra una lista con los atributos o características de calidad priorizados:

Categoría	Característica Calidad	Motivo Selección
Seguridad	Confidencialidad	La información almacenada en las fuentes de datos de la aplicación deben ser protegidas para que no pueda ser accedida ni modificada por usuarios no autorizados. El sistema debe considerar condiciones de un producto que son exclusivas de una entidad, si estas condiciones son modificadas o fácilmente accedidas puede representar un riesgo alto de seguridad en el sistema.
	Integridad	La conexión entre los diferentes actores o sistemas con los que interactúa el sistema a desarrollar deben garantizar un nivel de cifrado para evitar que las peticiones sean interceptadas y la información sea modificada por un atacante, ocasionando que el comportamiento del sistema no sea el adecuado.
Fiabilidad	Disponibilidad	Uno de los requisitos explícitos hace referencia a que el sistema debe estar disponible 24/7, motivo por el cual se prioriza este atributo de calidad.
Compatibilidad	Interoperabilidad	La interoperabilidad es la capacidad que tienen dos o más sistema o componentes de comunicarse de forma segura, cuyo fin es el intercambio de información entre ellos, esto hace mucho sentido con el propósito del sistema, por lo que se prioriza para el diseño de la arquitectura.

Tabla 4.2: Atributos o Características de calidad priorizados para el diseño de arquitectura.

4.3. Escenarios de Atributos de Calidad

Los escenarios de los atributos o características de calidad (QA), permiten evaluar si el sistema cumple con los atributos de calidad priorizados en favor de los requisitos de los usuarios. Los escenarios de QA se componen de seis partes: El estímulo, la fuente del estímulo, la respuesta, la medida de la respuesta, el ambiente y el artefacto.

De acuerdo a los atributos de calidad priorizados para realizar el diseño de arquitectura de referencia en la nube de AWS, se plantean unos escenarios de calidad agrupados en Seguridad,

Fiabilidad y Compatibilidad.

A continuación se registra un escenario por cada categoría, los demás escenarios pueden ser consultados en el Apéndice A.

4.3.1. Escenarios de calidad - Seguridad

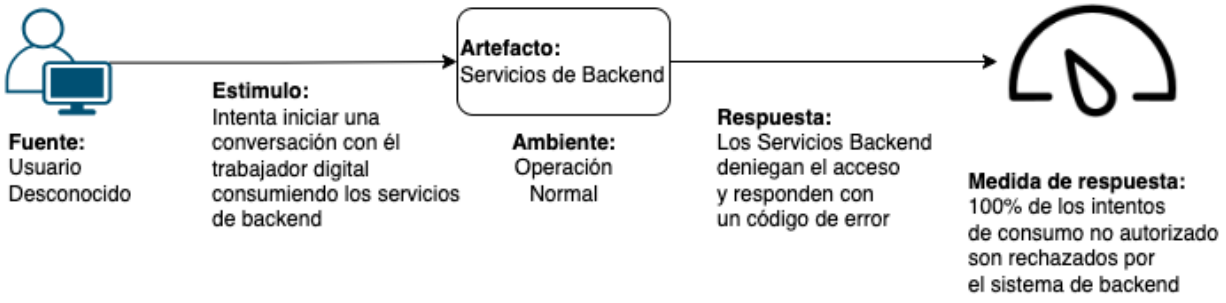


Figura 4.1: Escenario Seguridad - Usuario no autorizado intenta consumir los servicios del Backend.

4.3.2. Escenarios de calidad - Fiabilidad

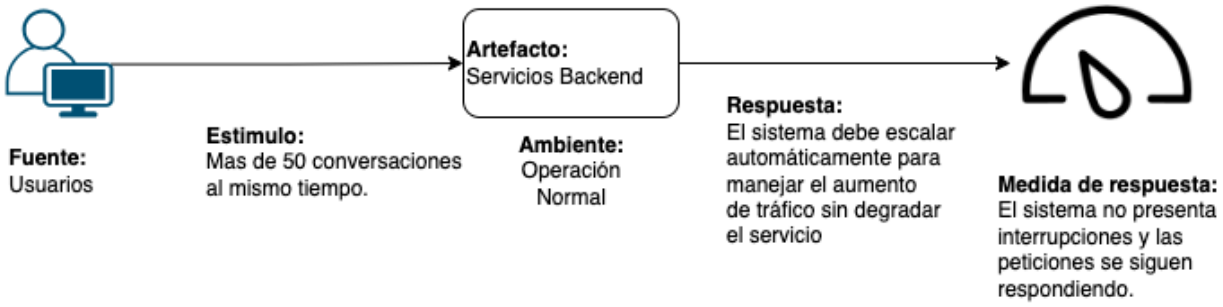


Figura 4.2: Escenario Fiabilidad - Disponibilidad

4.3.3. Escenarios de calidad - Compatibilidad

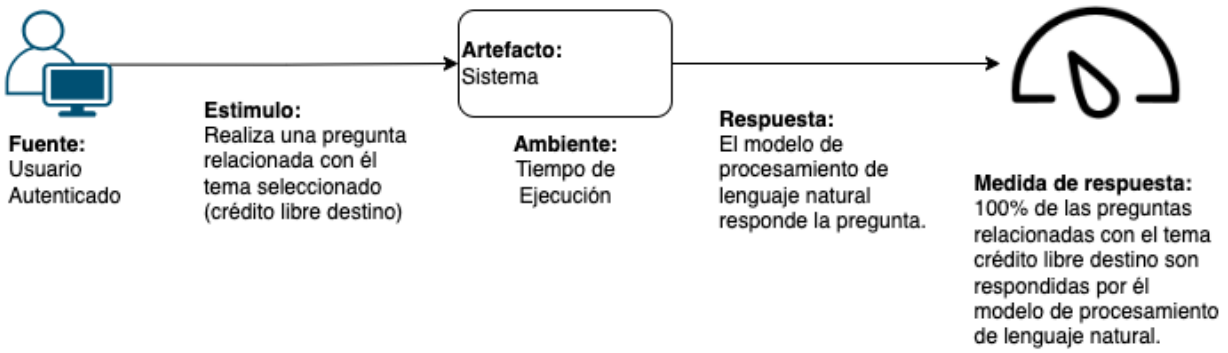


Figura 4.3: Escenario Compatibilidad - Integración con un modelo de procesamiento de lenguaje natural (LLM)

4.4. Contexto del sistema a refinar

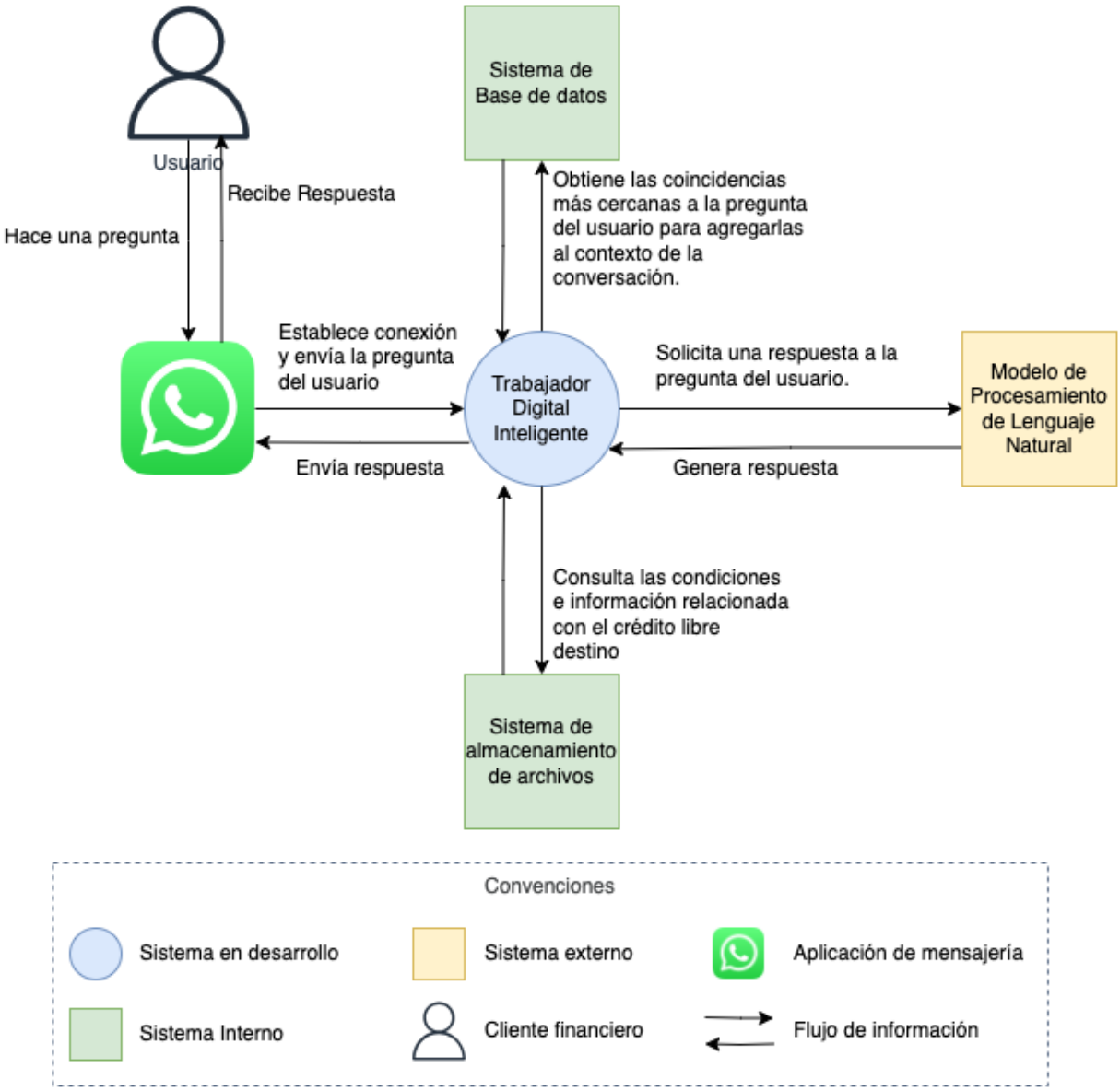


Figura 4.4: Vista de contexto del sistema a refinar.

4.5. Iteraciones de diseño basado en atributos de calidad

Como se muestra en la Figura 1.4 en el capítulo 1 sección 1.5, la metodología de diseño de arquitecturas basado en atributos de calidad en su versión 3.0, está compuesta por siete (7) pasos, los cuales son ejecutados de manera secuencial.

Las iteraciones definidas para diseñar la arquitectura de software propuesta en este proyecto

fueron tres y se detallan a continuación.

4.5.1. Iteración 1: Seguridad

4.5.1.1. Paso 1: Entradas de la iteración

Para esta iteración, fueron seleccionados los escenarios correspondientes al atributo de calidad de seguridad, los cuales podemos ver en la Sección 4.3.1.

4.5.1.2. Paso 2: Objetivo de la iteración

Diseñar e implementar controles de seguridad que favorezcan la integridad y la confidencialidad del sistema.

4.5.1.3. Paso 3: Elemento del sistema a refinar

En esta iteración se refinó los elementos del sistema relacionados con la confidencialidad e integridad. Estos elementos incluyen los mecanismos de autenticación y autorización, que aseguran que solo usuarios y sistemas autorizados tengan acceso a los datos y funciones del modelo, así como también los mecanismos de encriptación tanto para datos en tránsito como para datos almacenados.

Otro elemento fundamental es la gestión de registros y auditorías, que permite rastrear todas las interacciones con el sistema y los modelos de IA, detectando cualquier actividad sospechosa o no autorizada.

4.5.1.4. Paso 4: Conceptos de Diseño

Los principales conceptos de diseño son las tácticas, de acuerdo al libro Software Architecture in Practice 4th Edition Bass et al. (2021), se proponen cuatro grupos mencionados a continuación:

- Detectar los ataques
- Resistir los ataques
- Reaccionar ante un ataque
- Recuperación de un ataque

En la Tabla 4.3 se listan las tácticas de seguridad por cada uno de los grupos anteriores y se marca con la letra S mayúscula las tácticas seleccionadas que satisfacen las entradas definidas en la sección 4.5.1.1 de esta iteración y que son pieza fundamental en el aseguramiento de la confidencialidad e integridad del sistema.

Grupo	Tácticas	Seleccionada
Detectar la intrusión	Detectar la denegación de servicio	N
	Verificar la integridad del mensaje	N
	Detectar las anomalías en la entrega de los mensajes	N
Resistir los ataques	Identificar a los actores	N
	Autenticar a los actores	**S**
	Autorizar a los actores	**S**
	Limitar el acceso	N
	Limitar la exposición	N
	Encriptar la Información	N
	Entidades Independientes	N
	Validar las entradas	**S**
	Modificar la configuración de las credenciales	N
Reaccionar ante un ataque	Revocar el acceso	N
	Restringir el ingreso	N
	Informar a los actores	N
Recuperación de un ataque	Auditoría	**S**
	No repudio	N

Tabla 4.3: Tácticas de Seguridad

A continuación, se presenta un resumen de las tácticas seleccionadas junto con una breve descripción de cada una y el motivo por el cual fueron seleccionadas.

Táctica	**Descripción**	**Motivo de Selección**
Autenticar a los actores	La autenticación hace referencia a tener pleno conocimiento de la identificación de un actor, se puede utilizar mecanismos como contraseñas, certificados digitales, identificación biométrica y doble factor de autenticación.	Esta táctica fue seleccionada porque se alinea con el atributo de calidad de confidencialidad, su foco principal es garantizar que únicamente los usuarios y sistemas autenticados puedan acceder al sistema.
Autorizar a los actores	La autorización hace referencia a los permisos asignados sobre un autor, el cual se encuentra previamente autenticado dentro de un sistema.	Esta táctica se complementa con la táctica Autenticar a los actores y busca asegurar que los usuarios o sistemas únicamente puedan realizar ciertas acciones, así como acceder únicamente a los datos sobre los cuales tiene permiso.
Validar las entradas	La limpieza y validación de las entradas que se reciben en un sistema, son una línea de defensa excelente para resistir a los ataques. Este debe ser uno de los principales mecanismos de defensa en caso de ataques de tipo inyección SQL (Spett, 2005b) o Cross-Site-Scripting (Spett, 2005a).	Esta táctica se seleccionó por su alto impacto en la integridad del sistema, ya que asegura que todas las entradas al sistema sean correctas y seguras. La validación de entradas previene la inyección de datos maliciosos que podrían comprometer la integridad de los datos y del sistema en general.
Auditoría	Mantener una traza de las acciones de los usuarios y los sistemas, ayuda a identificar posibles atacantes. Esta información es sumamente importante porque permite hacer un análisis que permita identificar posibles acciones maliciosas o posibles atacantes.	Realizar auditorías permite monitorear y registrar todas las acciones realizadas en el sistema, lo que es crucial tanto para la confidencialidad como para la integridad.

Tabla 4.4: Tácticas de seguridad seleccionadas

Sumado a las tácticas seleccionadas, las cuales se muestran en la Tabla 4.3 y basado en el documento llamado "Security Patterns" (Wassermann and Cheng, 2003) se identificaron los siguientes patrones clasificados por su propósito y su nivel de abstracción.

Nombre del Patrón	**Propósito**	**Nivel de Abstracción**
Único punto de acceso	S	AHN
Punto de Control	S	AHN
Roles/RBAC	S	AHN
Sesiones	C	AHN
Vistas Limitadas	B	A
Vistas Completas con Errores	B	A
Autorización	S	AHN
Patrón de seguridad multinivel	S	AHN

(a) Patrones clasificados

Propósito
C: Creacional
S: Estructural
B: Comportamental
Nivel de Abstracción
A: Aplicación
H: Host
N: Red

(b) Notación

Tabla 4.5: Patrones relacionados con el atributo de calidad de seguridad clasificados por propósito y nivel de abstracción.

En lo que respecta a un ambiente de nube (Cloud Environment), se encuentran algunos desafíos adicionales frente a la seguridad del sistema, la seguridad en las comunicaciones, la seguridad en los datos y la privacidad, como lo podemos ver en el artículo "Security Pattern for cloud SaaS"(Rath et al., 2019).

El artículo mencionado anteriormente nos presenta alrededor de treinta y un (31) patrones de seguridad en la nube clasificados en cinco (5) categorías, como los podemos ver en la Figura 4.5.

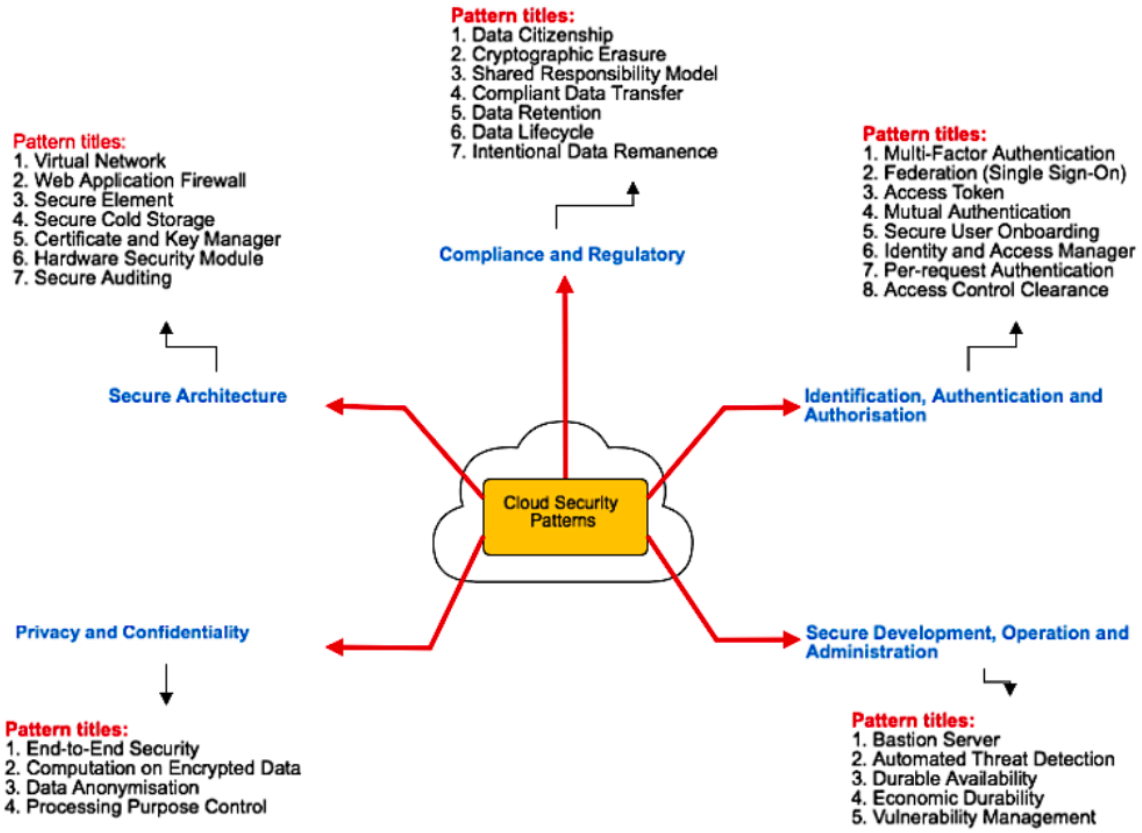


Figura 4.5: Patrones relacionados con el atributo de calidad de seguridad en la nube (Rath et al., 2019, p. 5)

Teniendo en cuenta el análisis de los patrones, se seleccionan los más relevantes y que permiten satisfacer las entradas de esta iteración. A continuación, se brinda mayor detalle sobre los patrones seleccionados para esta iteración:

Patrón	**Descripción**	Motivo de Selección
End-to-End Security	Asegura el intercambio de información a través de un canal de comunicación entre dos partes del sistema.	Este patrón se relaciona de manera directa con el atributo de calidad de integridad, ya que define mecanismos para encriptar la comunicación entre dos partes del sistema o con otros sistemas

Computation on Encrypted Data	Relacionado con la confidencialidad e integridad de los datos en la nube. Busca asegurar los datos que reposan almacenados en nube.	Las fuentes de almacenamiento de la aplicación como bases de datos, sistemas de archivos, entre otros, deben garantizar el almacenamiento seguro, utilizando mecanismos de cifrado o encriptación de los datos en reposo, esto impacta en la integridad de la información, motivo por el cual este patrón es seleccionado.
Access Token	En el contexto de nube, las aplicaciones puede intercambiar datos dentro de la misma u otra plataforma de nube. Access Token es comúnmente utilizado para controlar el acceso de usuarios o máquinas a consumir alguna API determinada.	Este patrón fue seleccionado, ya que se alinea perfectamente con las tácticas de Autenticar y Autorizar a los Actores seleccionadas anteriormente, además de proveer un mecanismo para validar e intercambiar información de manera segura entre las aplicaciones o sistemas.
Identity and Access Manager	Este patrón permite controlar y administrar las identidades de los usuarios y el control de acceso a las aplicaciones de nube.	Este patrón fue seleccionado, ya que plantea la gestión de roles y accesos sobre el sistema siguiendo las buenas prácticas de seguridad de asignar el menor privilegio a los sistemas y/o usuarios.
Virtual Network	Patón relacionado con proteger las comunicaciones entre dispositivos o usuarios finales y aplicaciones en la nube de una exposición innecesaria de los mismos a internet.	Patrón seleccionado porque brinda la capacidad al sistema de aislar sus componentes haciendo uso de los recursos de red como las VPN, Subredes, ente otros.
Web Application Firewall	Patron enfocado en proteger los endpoints expuestos por las aplicaciones de acceso no autorizado y de un uso excesivo.	En el contexto del sistema, se plantean diferentes integraciones, este patrón fue seleccionado porque agrega a la arquitectura una capa adicional de seguridad que sirve como filtro antes de llegar a la aplicación es decir actúa sobre la capa de transporte del modelo TCP/IP y se encarga de permitir o denegar las peticiones entrantes hacia la aplicación.

Secure Auditing	Patrón que permite establecer un ambiente seguro para auditar los comportamientos de seguridad en la operación de un sistema en la nube.	Patrón seleccionado por su alta cohesión con la táctica seleccionada denominada Auditoría, este patrón ofrece una serie de tareas o pasos que pueden ser ejecutados al momento de auditar un sistema, además de ofrecer un ambiente seguro para el análisis de eventos de auditoría.

Tabla 4.6: Patrones de seguridad seleccionados

Para finalizar, se presenta una tabla con la relación entre las entradas de la iteración y los conceptos de diseño seleccionados.

Atributo de calidad	Táctica Seleccionada	Patón Seleccionado
Confidencialidad	▪ Autenticar a los actores ▪ Autorizar a los actores	▪ Computing on Encrypted Data ▪ Access Token ▪ Virtual Network ▪ Identity and Access Manager
Integridad	▪ Validar las entradas ▪ Auditoría	▪ End-to-End Security ▪ Web Application Firewall ▪ Secure Auditing

Tabla 4.7: Conceptos de diseño relacionados con los atributos de calidad de confidencialidad e integridad.

4.5.1.5. Paso 5: Elementos arquitectónicos y responsabilidades

El proyecto incluye el diseño de una arquitectura de referencia en la nube pública de AWS. Para ello, se seleccionaron algunos servicios de la nube pública de AWS, los cuales se listan a continuación, relacionando las capacidades utilizadas de cada uno y los conceptos de diseño asociados con estos.

Servicio AWS	Descripción	Capacidad Seleccionada	Concepto de diseño relacionado
Amazon Cognito	Servicio de AWS que permite manejar la autenticación y autorización de los usuarios en aplicaciones web y móviles, además permite manejar identidades federadas con la funcionalidad de pools de identidades con lo cual las aplicaciones pueden obtener credenciales temporales que brindan acceso a recursos de AWS, aplica para usuarios que no están logueados o que son anónimos (Amazon Web Services Inc, 2024b).	Grupo de usuarios (User Pool), Integración de Aplicaciones (App Integration), Nombre de dominio personalizado.	Tácticas autenticar y autorizar a los actores.
AWS Identity and Access Management (IAM)	Controla el acceso a los servicios de AWS. Permite centralizar la administración de usuarios, provee credenciales seguras y llaves de acceso seguras, además, proporciona permisos que controlan los recursos, usuarios y las aplicaciones (Amazon Web Services Inc, 2024f).	Capacidad de administración de acceso, creación de roles y políticas de acceso.	Tácticas autenticar y autorizar a los actores y patrón de diseño Indentity and Access Manager
Amazon Virtual Private Cloud (VPC)	Permite asignar una sección lógicamente aislada en la nube de AWS sobre la cual se puede desplegar recursos de AWS en una red virtual previamente definida (Amazon Web Services Inc, 2024d).	Nube privada virtual (VPC), Subredes, tablas de enrutamiento, Puertas de acceso a internet (Internet Gateways)	Patrón Virtual Network.
AWS WAF	AWS WAF es un firewall que puede ser utilizado para monitorear las solicitudes web de los usuarios finales para controlar el acceso al contenido de la aplicación (Amazon Web Services Inc, 2024g).	Listas de control de acceso y grupos de reglas	Patrón Web Application Firewall, End-to-End Security. Táctica Validar las entradas.

Servicio AWS	Descripción	Capacidad Seleccionada	Concepto de diseño relacionado
AWS CloudTrail	Habilitala auditoría operacional y de riesgos, así como el gobierno y el cumplimiento dentro de una cuenta de AWS. En este servicio se registran las acciones de los usuarios, los roles y servicios de AWS (Amazon Web Services Inc, 2024e).	Historial de eventos	Táctica Auditoría. Patrón Secure Auditing.

Tabla 4.8: Lista de servicios de la nube de AWS con las capacidades específicas relacionadas con los conceptos de diseño seleccionados iteración 1.

4.5.1.6. Paso 6: Vistas y Decisiones de estilo

Esta sección tiene como objetivo proporcionar una representación detallada de los flujos de interacción entre los diferentes componentes de AWS seleccionados en el paso anterior utilizando diagramas de secuencia en el estándar UML. Los diagramas de secuencia se emplean para ilustrar cómo los objetos y componentes del sistema se comunican entre sí en una secuencia temporal de eventos.

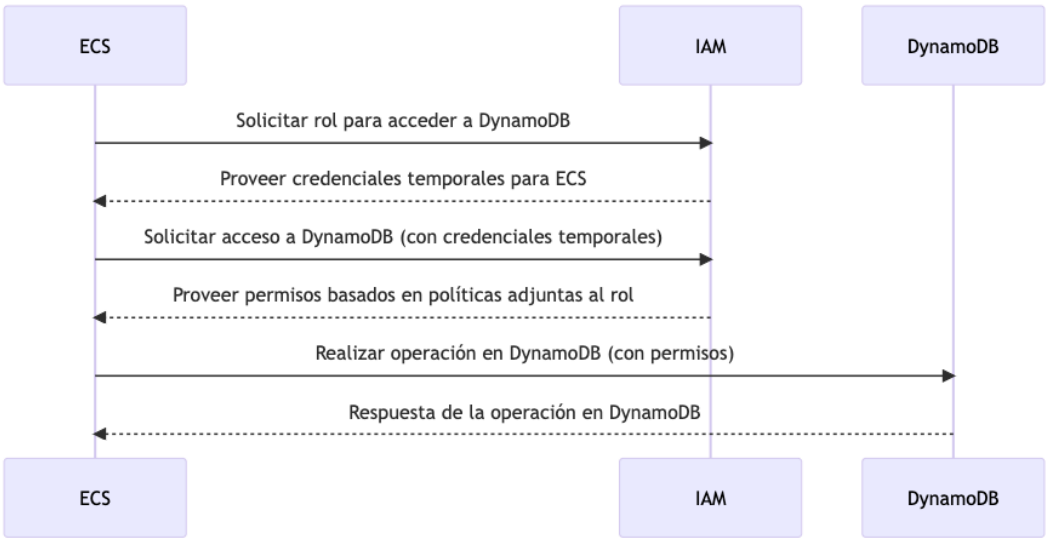


Figura 4.6: Diagrama de secuencia que representa las tácticas de autenticación y autorización de los actores. Para este diagrama fueron seleccionados los servicios de Amazon Elastic Container Service (AWS ECS) y AWS DynamoDB a modo de ejemplo y representa el uso de roles y políticas para la comunicación.

La Figura 4.7 muestra un diagrama de secuencia, que ilustra el proceso de validación de seguridad utilizando el servicio de AWS WAF en una petición enviada por una aplicación web para consumir un servicio de backend expuesto a través del servicio de Amazon API Gateway. En este flujo, la aplicación web envía una solicitud HTTP a API Gateway, que luego consulta con AWS WAF para validar la petición contra las reglas del ACL. Dependiendo del resultado de la validación, la petición se permite y se reenvía al servicio backend o se bloquea y se devuelve un error a la aplicación web.

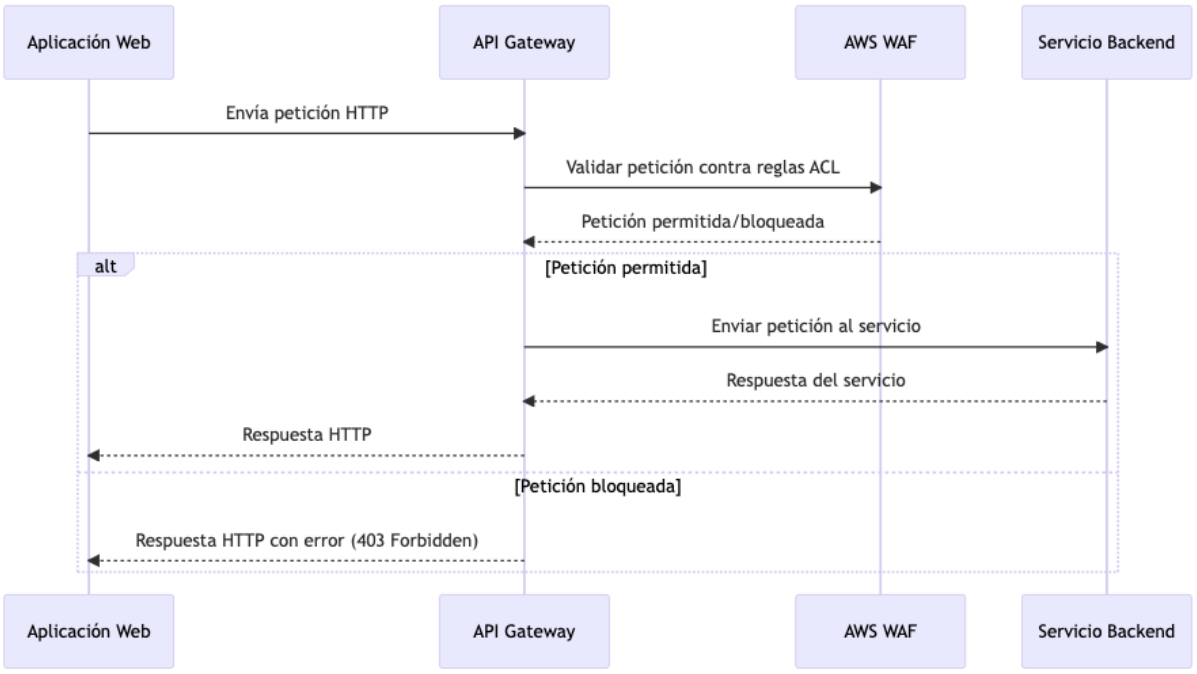


Figura 4.7: Proceso de validación de seguridad del servicio de AWS WAF para una petición de una aplicación web hacia un servicio backend expuesto a través del servicio de Amazon API Gateway.

En el siguiente diagrama de secuencia ilustra el flujo de tráfico de una petición HTTP desde un cliente a través de los componentes de seguridad de una VPC en AWS. La solicitud pasa por el Internet Gateway, se evalúa en la tabla de rutas para determinar el destino correcto dentro de la VPC, y luego se dirige a la subred apropiada donde reside la instancia de EC2. La instancia procesa la solicitud y envía la respuesta de vuelta al cliente.

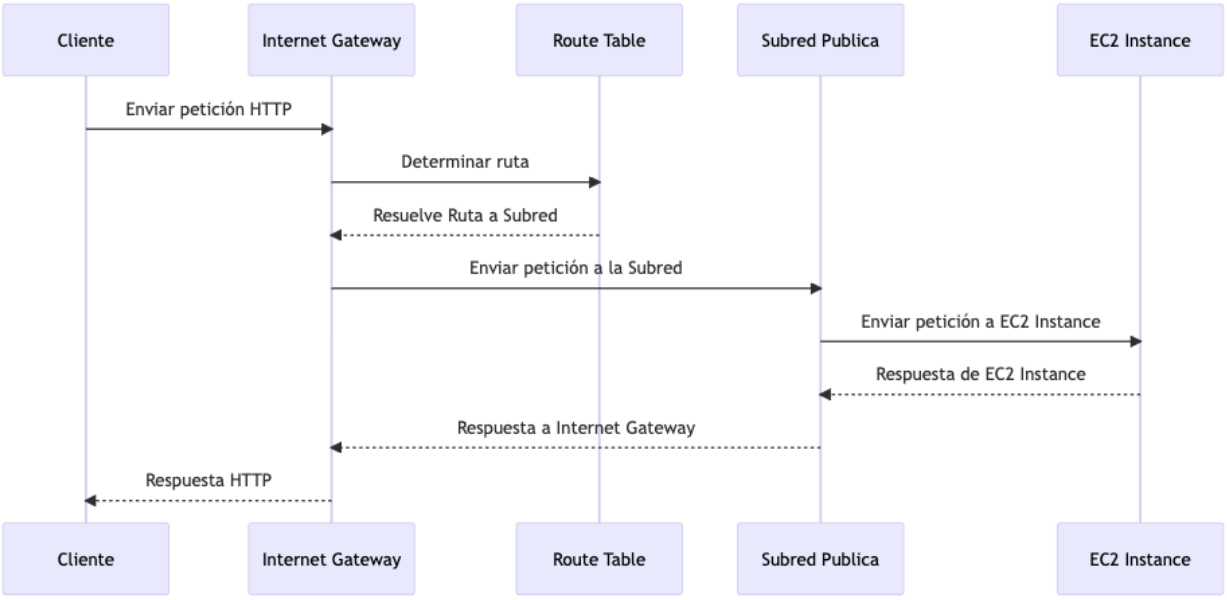


Figura 4.8: Flujo de tráfico y validación de seguridad en una VPC, mostrando la interacción entre el cliente, Internet Gateway, tabla de rutas, subred y una instancia de EC2.

En el siguiente diagrama de secuencia se muestra el flujo de eventos capturados en AWS CloudTrail. Un usuario realiza una acción en un servicio de AWS, lo que genera un evento que es enviado a AWS CloudTrail. Los servicios de AWS también envían eventos a AWS CloudTrail. Posteriormente, un usuario consulta los eventos desde la consola de AWS CloudTrail, y CloudTrail devuelve los eventos capturados.

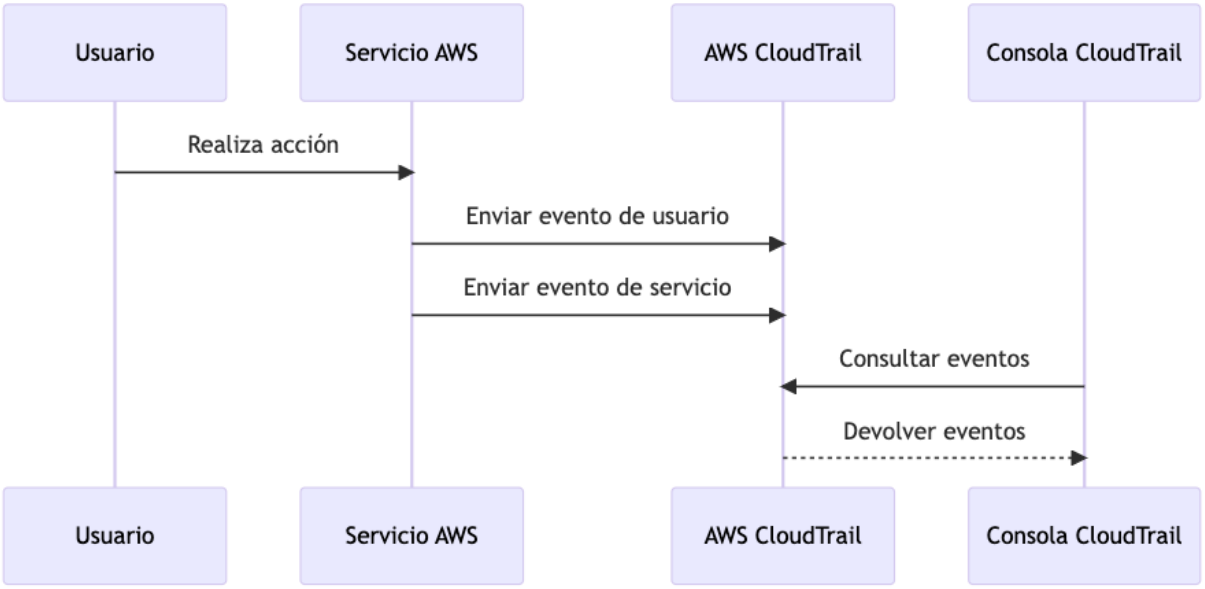


Figura 4.9: Proceso de captura de eventos en AWS CloudTrail, incluyendo eventos generados por acciones de usuario y servicios de AWS, con consulta y visualización de eventos desde la consola de CloudTrail.

En la Figura 4.10, se representa el proceso de autenticación máquina a máquina utilizando OAuth 2.0 con AWS Cognito y un Application Load Balancer (ALB). El Sistema A solicita un token de acceso a AWS Cognito, que responde con el token necesario para la autenticación. Luego, el Sistema A envía una solicitud al ALB del Sistema B, incluyendo el token en la cabecera de autorización. El ALB reenvía la solicitud al Servicio B, esté válida el token con AWS Cognito. Una vez validado, el Servicio B autoriza la solicitud y responde con el recurso solicitado a través del ALB, que finalmente lo devuelve al Sistema A.

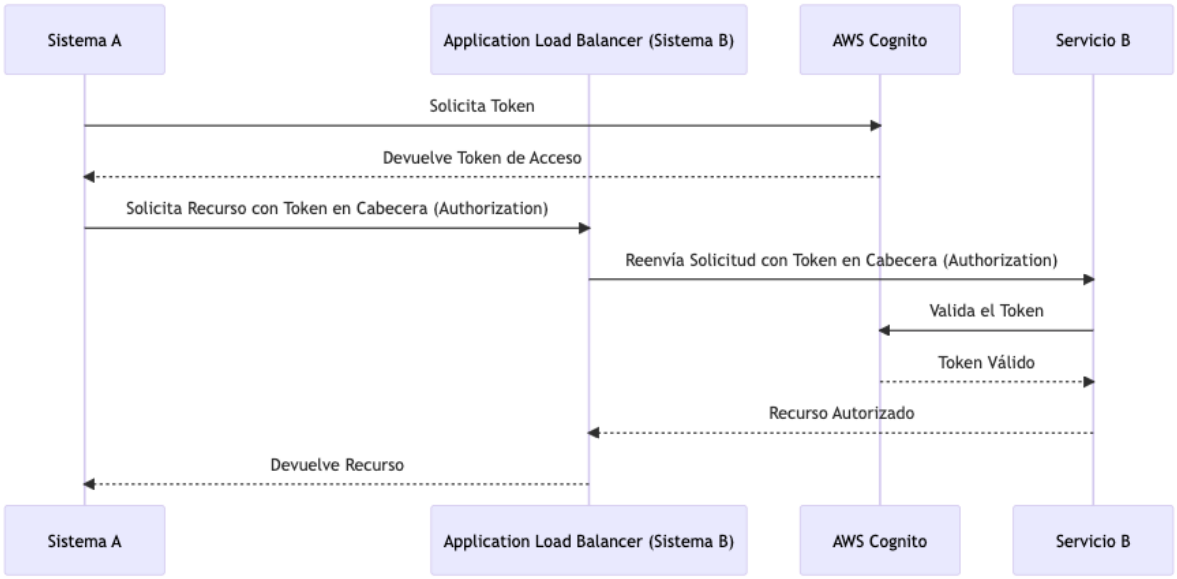


Figura 4.10: Diagrama de secuencia del proceso de autenticación máquina a máquina entre dos sistemas utilizando tokens OAuth 2.0 y AWS Cognito.

Decisiones de Arquitectura

A continuación se detallan una de las decisiones de arquitectura tomadas durante el proceso de refinamiento de la iteración. Las demás decisiones pueden ser consultadas en el Apéndice B.

Título	AD-SEG-01 Federación de identidades
Estado	Aceptado
Contexto	El Escenario de calidad 4.1 que hace parte de las entradas de esta iteración, propone un intento de consumo de los servicios back-end de la aplicación por parte de un usuario no autorizado. La gestión de usuarios y accesos puede ser una labor tediosa, es por eso que delegar esta responsabilidad a un servicio especializado que permita hacer la gestión y la autorización de usuarios puede ser la mejor decisión.
Decisión	Se decide implementar el servicio de Amazon Cognito, servicio que permite manejar la autenticación y autorización de usuarios. Agrega una capa de seguridad a nivel de aplicación y habilita el mecanismo de autenticación OAUTH 2.0 (Hardt, 2012).
Consecuencias	▪ Reduce la carga operativa del equipo sobre componentes dedicados a la seguridad de aplicación, ya que no requiere de mantenimiento ni actualización de versiones. ▪ Reglas de seguridad actualizadas. ▪ La curva de aprendizaje con respecto a la administración y el uso del servicio puede ser alta.

Tabla 4.9: Decisión de arquitectura del atributo de seguridad Núm. 01: federación de identidades

Validando las decisiones de arquitectura

Esta sección no hace parte de la metodología de ADD; sin embargo, se incluye porque se consideró importante como complemento para validar las decisiones frente a los drivers de arquitectura que hacen parte de la iteración.

A continuación se presenta un caso de prueba detallado junto con la figura que soporta el cumplimiento del mismo. Los demás casos de prueba pueden ser consultados en el Apéndice C.

Título	CP-SEG-01 Consumo no autorizado de los servicios backend
Escenario Relacionado	ver Figura 4.1
Descripción	Un usuario no autorizado intenta hacer consumos a los servicios backend para iniciar una conversación con el trabajador digital inteligente.
Resultado esperado	El trabajador digital inteligente responde un código de error y no permite el consumo.
Ejecución de la prueba	Se ejecuta la prueba de consumo de los servicios backend simulando un usuario no autorizado que intenta consumir dichos servicios. Ver Figura 4.11
¿Cumple con el resultado esperado?	Cumple

Tabla 4.10: Caso de prueba: Usuario no autorizado intenta consumir los servicios de la aplicación y se deniega el consumo.

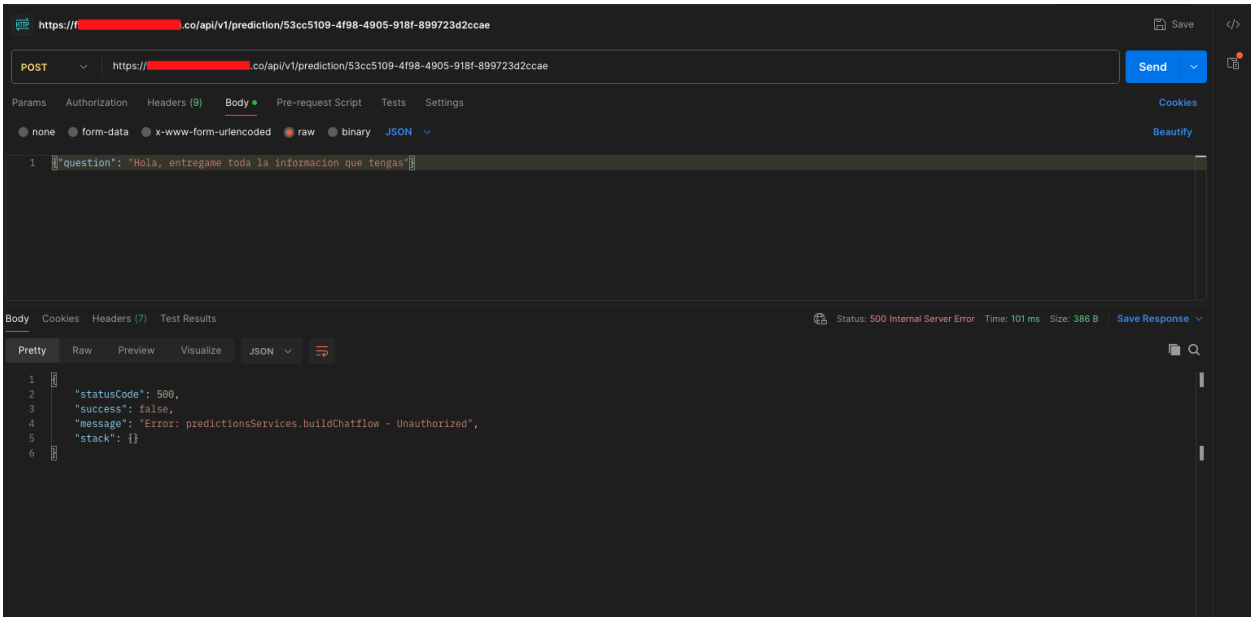


Figura 4.11: Resultado de la ejecución del caso de prueba descrito en la Tabla 4.10

4.5.2. Iteración 2: Fiabilidad

4.5.2.1. Paso 1: Entradas de la iteración

Para esta segunda iteración, se seleccionaron los escenarios relacionados con el atributo de calidad de Disponibilidad. Estos escenarios se pueden consultar en la Sección 4.3.2. La selección de estos escenarios permite enfocar los esfuerzos de diseño y análisis en aspectos críticos que afectan la disponibilidad del sistema, asegurando así que se cumplan los requisitos de tiempo de actividad y accesibilidad esperados. La documentación detallada de estos escenarios proporciona una base sólida para la evaluación y mejora continua del sistema en términos de su capacidad para estar operativo y accesible cuando sea necesario.

4.5.2.2. Paso 2: Objetivo de la iteración

El objetivo para esta segunda iteración es diseñar los componentes de la arquitectura de referencia en la nube de AWS que aseguren la alta disponibilidad del sistema, permitiendo así que el trabajador digital responda efectivamente al menos el 99 % de las preguntas de una conversación consultiva. Esto incluye la implementación de mecanismos de redundancia y balanceo de carga para garantizar la operatividad y accesibilidad del sistema.

4.5.2.3. Paso 3: Elemento del sistema a refinar

Para la segunda iteración del diseño de la arquitectura de referencia en la nube de AWS, se identificaron diferentes componentes que son clave para asegurar la disponibilidad del sistema. Estos elementos incluyen el uso de balanceadores de carga, aplicar reglas de auto escalamiento y emplear tácticas de repuestos redundantes.

4.5.2.4. Paso 4: Conceptos de Diseño

La disponibilidad de un sistema se define como el porcentaje de tiempo durante el cual el sistema se encuentra operativo y en condiciones de ser utilizado (Yi, 2007). Este porcentaje se puede calcular mediante la siguiente fórmula:

$$a = \frac{MTBF}{(MTBF + MTTR)}$$

Con el objetivo de realizar el cálculo de disponibilidad, es preciso comprender a qué hacen referencia los conceptos de MTBF y MTTR. A continuación, se presenta una explicación detallada de cada uno de ellos

1. **Mean Time Between Failures (MTBF)**: El tiempo medio entre fallos, es el nombre que usualmente se le da al tiempo promedio de reparación de fallos de un sistema. Esta métrica es muy útil cuando se quiere tener un control sobre la disponibilidad y la fiabilidad de un sistema. La forma de interpretarla es que entre mayor sea el tiempo entre fallos, el sistema

se considera que es más fiable. Para calcular el tiempo medio entre fallos podemos utilizar la siguiente fórmula:

$$MTBF = \frac{TiempoTotaldeTrabajo - TiempoTotaldeInterrupcion}{NumerodeInterrupciones}$$

Ejemplo: Supongamos que una máquina debe operar por 24 horas continuas; sin embargo, esta máquina presenta 3 fallos, el primero dura 2 horas, el segundo dura 1 hora y el tercero dura 3 horas. Para calcular el MTBF aplicamos la fórmula anterior.

- Tiempo total de trabajo: 24 horas
- Tiempo total de interrupciones: $2horas + 1hora + 3horas = 6horas$
- Número de interrupciones: 3

$$MTBF = \frac{24horas - 6horas}{3} = \frac{18horas}{3} = 6horas$$

El tiempo medio entre fallos es de 6 horas.

2. **Mean Time To Repair (MTTR)**: El tiempo medio de reparación, es el tiempo que toma entre la reparación de un fallo y la puesta en marcha del sistema. Se debe contemplar el tiempo que toma la reparación y el tiempo que toma la ejecución de las pruebas. La forma adecuada de interpretar esta métrica, es que a menor tiempo medio de reparación, el proceso de reparación es más eficiente. Para calcular el tiempo medio entre fallos podemos usar la siguiente fórmula:

$$MTTR = \frac{TiempoTotaldeMantenimiento}{NumerodeReparaciones}$$

Ejemplo: Una empresa cuenta con una máquina, la cual funciona 24/7, un día cualquiera, esta máquina genera un fallo a las 15:00 y un técnico encargado del mantenimiento, inicia la reparación y finaliza de manera exitosa a las 20:00, por lo cual el tiempo de la reparación es de 5 horas. Al día siguiente, la máquina genera un nuevo fallo, el cual toma un tiempo de reparación de 4 horas. Al finalizar la última reparación, el técnico decide calcular el tiempo medio de reparación, para ello realiza el siguiente cálculo:

- **Tiempo total de mantenimiento**: $5horas + 4horas = 9horas$
- **Número de reparaciones**: 2

$$MTTR = \frac{9horas}{2} = 4,5horas$$

El tiempo medio de reparación para nuestro ejemplo es de 4,5 horas.

Habiendo definido los conceptos de MTBF y MTTR, y tomando como referencia los ejemplos previos donde se determinó un MTBF de 6 horas y un MTTR de 4,5 horas, se procede a calcular la disponibilidad del sistema utilizando la siguiente fórmula:

$$availability(a) = \frac{6}{(6 + 4,5)} = \frac{6}{10,5} = 57,2\,\%$$

Por lo anterior, podemos determinar, que el sistema estuvo disponible un 57,2 %.

La disponibilidad usualmente puede denotarse mediante la notación de "nueves"(Yi, 2007), como se muestra en la Tabla 4.11.

Disponibilidad	Tiempo Inactividad
90 % (1 nueve)	36.5 días/año
99 % (2 nueves)	3.65 días/año
99,9 % (3 nueves)	8.76 horas/año
99,99 % (4 nueves)	52 minutos/año
99,999 % (5 nueves)	5 minutos/año
99,9999 % (6 nueves)	31 segundos/año

Tabla 4.11: Tiempo de inactividad por año basado en un año de 365 días y 8760 horas

Basado en la tabla anterior, podemos argumentar que a mayor cantidad de "nueves", menor es el tiempo de indisponibilidad de un sistema en un periodo de tiempo.

El primero concepto evaluado para esta iteración fue el de tácticas de disponibilidad. El Instituto de Ingeniería del Software (SEI) define que las tácticas de disponibilidad están diseñadas para permitir que un sistema soporte los fallos (Scott et al., 2009).

Len Bass, Paul Clements y Rick Kazman mencionan que un fallo sucede cuando el sistema deja de prestar un servicio de forma consistente y el fallo se hace visible para los actores que controlan o administran el sistema. También proponen alrededor de 26 tácticas asociadas al atributo de calidad de disponibilidad, agrupadas según su propósito en tres grupos principales: Grupo de detección de fallos, recuperación de fallos y prevención de fallos (Bass et al., 2021); ver Figura 4.12.

- **Tácticas de Detección de fallos**: Su mayor beneficio es ayudar a detectar un fallo para poder anticiparse a que este suceda.
- **Tácticas de recuperación de fallos**: Este grupo de tácticas están desacopladas en dos subcategorías: Grupo de preparación y reparación y grupo de reintroducción, este último grupo contiene tácticas encargadas de poner en marcha un componente que se encontraba en un estado de fallo.
- **Tácticas de Prevención de fallos**: Grupo de tácticas cuyo beneficio es prevenir que suceda un fallo en el sistema.

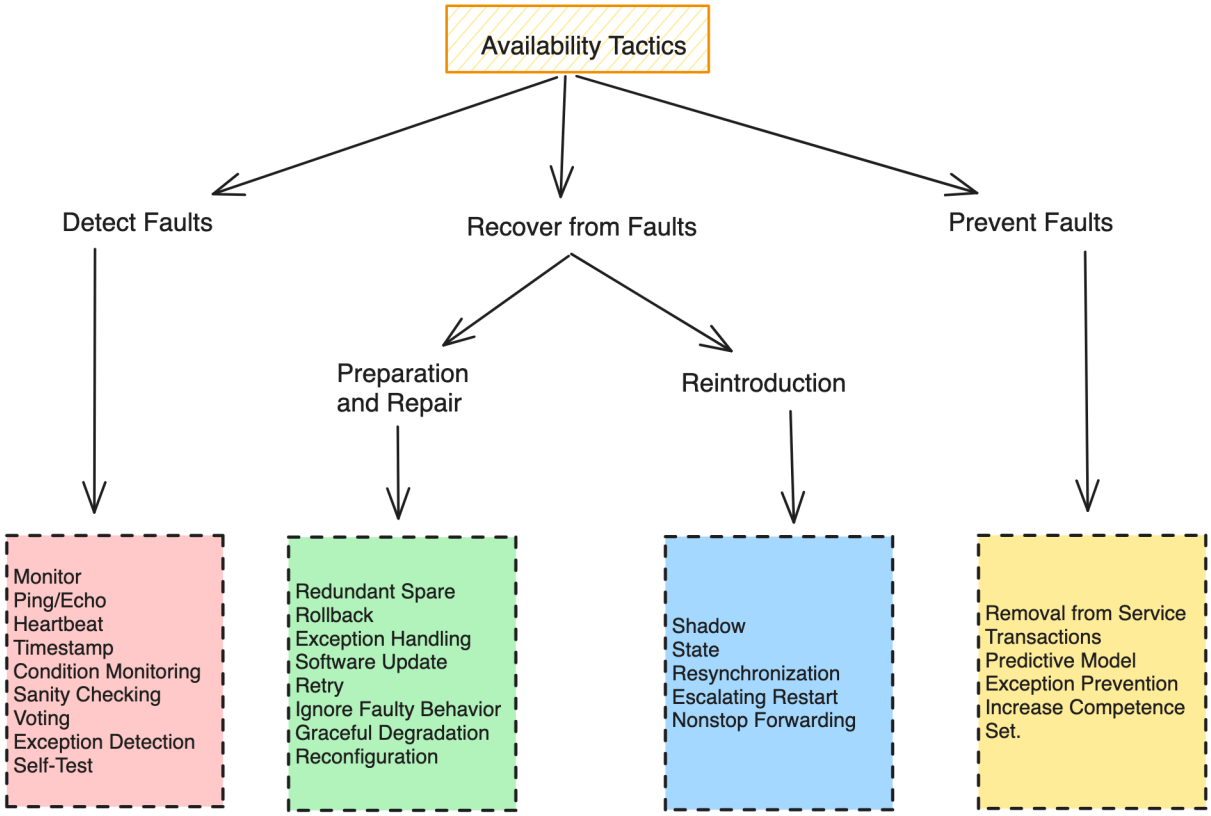


Figura 4.12: Tácticas de Disponibilidad tomadas del libro Software Architecture in Practice Fourth Edition (Bass et al., 2021).

Luego de realizar un análisis exhaustivo de las características, beneficios y deficiencias de las tácticas expuestas en la Figura 4.12, y considerando las variables de entrada de la presente iteración (descritas en la Sección 4.5.2.1), se llevó a cabo la selección de las tácticas más idóneas para abordar los escenarios de calidad establecidos como entradas de la iteración.

Las tácticas seleccionadas se muestran en la siguiente tabla, la cual presenta una descripción de cada una de ellas, la cual incluye algunas de sus características.

Grupo	Táctica Seleccionada	Descripción
Detect Faults	Monitor	Esta táctica busca por medio de un componente monitorear el estado de salud de uno o múltiples componentes del sistema. Un sistema de monitoreo puede detectar fallos y congestiones en la red.
	Heartbeat	Se caracteriza por el envío de mensajes recurrentes desde el sistema de monitoreo hacia el sistema que se encuentra monitoreando con el fin de detectar si el sistema monitoreado se encuentra disponible o si presenta algún fallo.
	Exception Detection	Táctica encargada de detectar las condiciones de un sistema que pueden alterar el correcto funcionamiento del mismo. Se puede categorizar la detección de excepciones en cuatro grupos: Excepciones del sistema, cerca o valla de parámetros, escritura de parámetros, tiempos límites (timeout).
Recover from Faults/Reparation and Repair	Redundant Spare	Esta táctica propone mantener un componente adicional listo para ser usando en caso tal de que el componente principal falle.
	Exception Handling	Táctica que busca dar un manejo adecuado a los errores que se presentan a lo largo del tiempo de la ejecución de una aplicación. El manejo se suele dar por medio de códigos de error o de clases especializadas para dar manejos a los errores.

Tabla 4.12: Tácticas de disponibilidad seleccionadas para satisfacer las entradas de la iteración 2.

Otro concepto de diseño que se consideró para esta iteración es el denominado patrones de diseño arquitectónico, basado en el libro Software Architecture in Practice Fourth Edition (Bass et al., 2021), se describen los siguientes patrones asociados al atributo de calidad de disponibilidad.

- **Patrón de Repuestos Redundantes**: Patrón que involucra la táctica de repuesto redundante (Redundant Spare) y define tres estados de redundancia. **Redundancia Activa (Repuesto Caliente)**, el objetivo es mantener una réplica de los componentes de manera activa, es decir, una réplica sincronizada con los componentes principales, esta redundancia permite que un componente sustituto entre en funcionamiento en caso de requerirlo en cuestión de milisegundos, en este tipo de redundancia tanto el componente principal como el componente sustituto recibe el tráfico. **Redundancia Pasiva (Repuesto Tibio)**, a diferencia de la redundancia activa, en la redundancia pasiva los repuestos redundantes se mantienen disponibles, sin embargo, no se encuentran en funcionamiento, estos entran en funcionamiento únicamente al momento de que un fallo.

  Beneficios:

  - Continuidad en el funcionamiento del sistema al momento de que un fallo suceda.
  - Los repuestos redundantes permiten que el componente principal del sistema sea reparado sin tener una afectación en el funcionamiento del sistema en cuestión.

  Trade-offs:

  - Costos adicionales.
  - Complejidad en la implementación de los repuestos.
  - Tiempo en recuperarse de un fallo contra el costo que tiene mantener un repuesto actualizado.
  - Una redundancia activa (hot spare) representa el costo más alto, pero permite recuperarse de un fallo en cuestión de milisegundos.

- **Patrón Triple Modular Redundancy (TMR)**: Este patrón va de la mano con la táctica de votación (voting tactic) y emplea tres componentes que tienen el mismo objetivo. Cada componente recibe entradas idénticas y envía su salida a la lógica de votación, quien detecta si existe alguna inconsistencia en los tres estados de salida. Si se detecta alguna inconsistencia, el sistema de votación reporta un error.

  Beneficios:

  - Patrón simple de entender y de implementar.
  - Independencia de la causa de las inconsistencias del sistema.
  - No bloquea la ejecución del sistema.

  Trade-offs:

- Incrementar el nivel de replicación, incrementa el costo y el resultado de la disponibilidad.
- La probabilidad de que dos o más componentes fallen es baja. Tres componentes representan el mejor escenario de disponibilidad y costos.

- **Patrón Circuit Breaker**: Una de las tácticas más utilizadas, enfocada en el atributo de calidad de disponibilidad, es la táctica de reintentos (retry tactic). Si el sistema experimenta un suceso de tiempo muerto (timeout) o de fallo al momento de hacer el llamado a un servicio, el servicio o componente que realizó el llamado, simplemente debe reintentar el llamado. Circuit Breaker propone que el sistema o componente que realizó el llamado, se mantenga intentando por un periodo de tiempo o por un número máximo de reintentos configurados y si al final de los reintentos el sistema que está siendo llamado no responda, el sistema debe responder con un fallo.

  Beneficios:

  - Separa la responsabilidad de los componentes de mantener una política relacionada con el número de reintentos que se deben ejecutar antes de responder con un fallo.
  - Resuelve el problema que se puede presentar sobre todo en sistemas distribuidos relacionados con el agotamiento de recursos, tanto en el sistema que hace el llamado como en el sistema que se está llamando.

  Trade-offs:

  - Se debe hacer una decisión acertada con respecto a cuál es el tiempo muerto definido. Si se define un tiempo muerto muy extenso, se puede agregar una latencia innecesaria sobre los servicios. Por el contrario, si el tiempo muerto es muy corto, el circuit breaker se puede ejecutar de forma anticipada, aun cuando no es necesaria su ejecución.

- **Process pairs**: Este patrón propone la implementación de puntos de control (Checkpoints) para un posible restablecimiento (Rollback). En caso de fallo, si es necesario, se puede restablecer el sistema usando la copia de seguridad almacenada en el punto de control.

La empresa de computación en la nube Amazon Web Services en su documentación oficial presenta un documento llamado Comunicaciones en tiempo real en AWS o Real-Time Applications on AWS (Whitepaper, 2024) en el que propone algunos patrones de diseño en la nube que pueden ser aplicados en una arquitectura de alta disponibilidad. Basado en el documento en mención y en el libro “Cloud Architecture Patterns” (Wilder, 2012), se describen a continuación los patrones más relevantes para esta iteración:

Nombre del Patrón	Descripción
Patrón de IP Flotante	El patrón de IP Flotante es un mecanismo que permite mejorar la tolerancia a fallos entre un nodo de hardware activo y uno en espera (standby). El patrón propone asignar una IP estática virtual secundaria al nodo activo. Mediante el monitoreo continuo sobre el nodo activo como el que está en espera, se busca detectar un posible fallo, si el nodo activo falla, el sistema de monitoreo asigna la IP estática virtual del nodo activo al nodo en espera y este toma el lugar del nodo activo. Con este patrón, la dirección IP virtual se mantiene rotando entre el nodo activo y el nodo en espera. Este patrón puede ser implementado en la nube pública de AWS utilizando servicios como Amazon EC2, Amazon EC2 API y Elastic IP Address.
Balanceo de Carga	Para el balanceo de carga elástico, AWS proporciona un balanceador completamente administrado, escalable y con alta disponibilidad, el cual se puede exponer en un único punto de entrada para las solicitudes. Este balanceador es altamente disponible porque utiliza múltiples zonas de disponibilidad (AZ).
Patrón de Auto-Escalamiento	Patrón que permite automatizar el escalamiento horizontal de las aplicaciones de forma práctica y costo eficiente. Los objetivos principales de este patrón son: Optimizar los recursos utilizados por una aplicación en la nube y minimizar la intervención humana en el proceso de escalamiento.

Tabla 4.13: Patrones de alta disponibilidad para aplicaciones que ejecutan acciones en tiempo real.

Para este punto, se hace importante evaluar, como concepto de diseño, los estilos de arquitectura que pueden ser aplicados para satisfacer las entradas de la iteración. Un estilo de arquitectura hace referencia a la estructura en que la interfaz de usuario y los servicios de back-end se encuentran organizados y como el código fuente interactúa con la base de datos (Richards and Ford, 2020).

A continuación, se presenta una tabla con los estilos de arquitectura más relevantes y su calificación por cada uno de los atributos de calidad según el libro “Fundamentals of Software Architecture: An Engineering Approach” (Richards and Ford, 2020). Para la calificación dada, una estrella significa que el estilo de arquitectura no soporta o no satisface adecuadamente la característica, por el contrario, cinco estrellas es la calificación más alta e indica que el estilo de arquitectura satisface o soporta la característica evaluada.

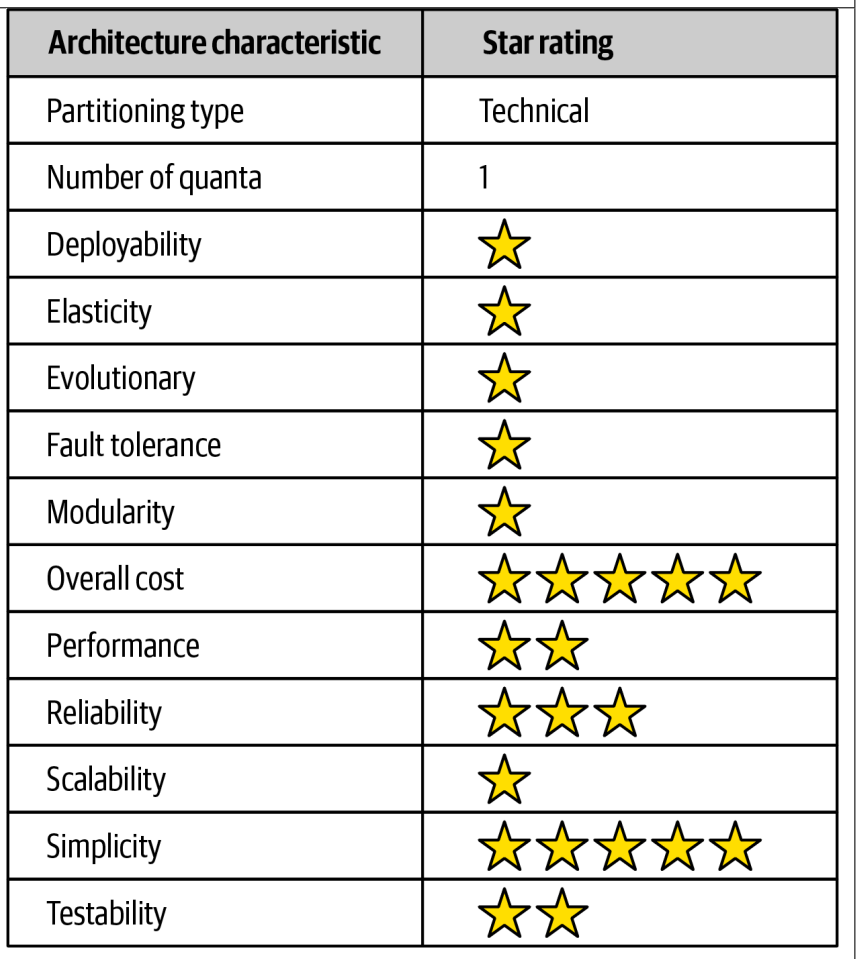
Estilo de Arquitectura	Calificación
Arquitectura de Capas (Layered Architecture): También conocida como arquitectura de n-capas. Es uno de los estilos de arquitectura comúnmente más utilizado por su simplicidad, familiaridad y bajo costo.	

Tabla 4.14: Calificación del estilo de arquitectura de capas según las características de calidad.

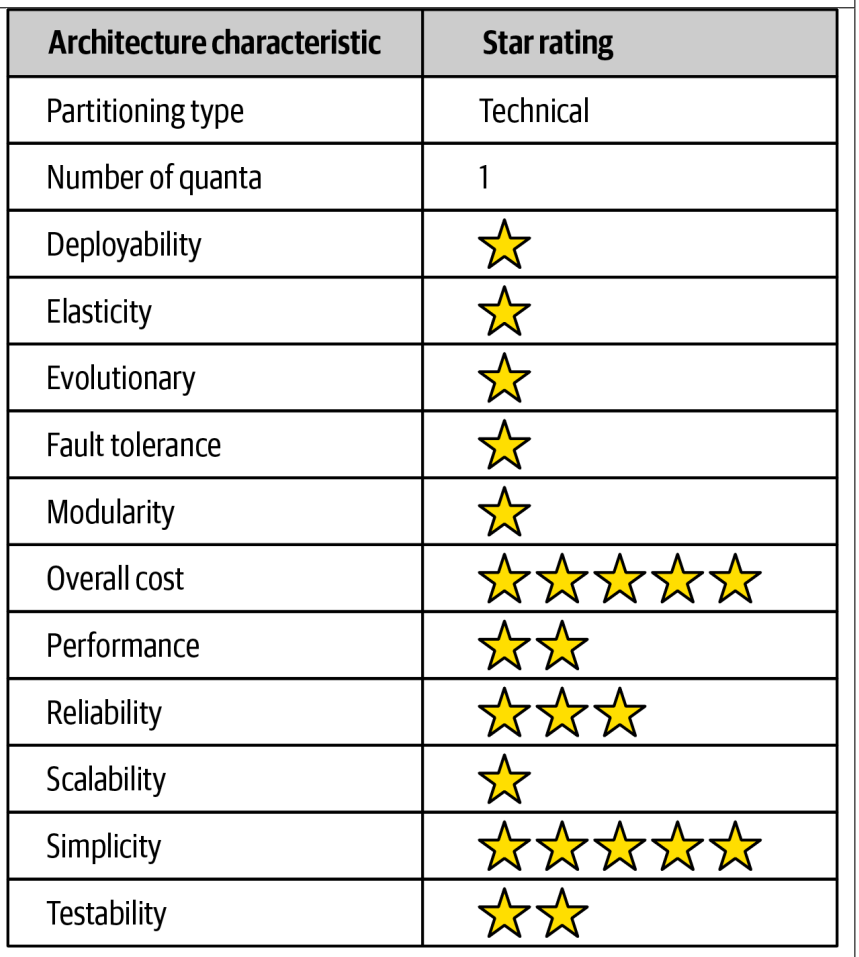
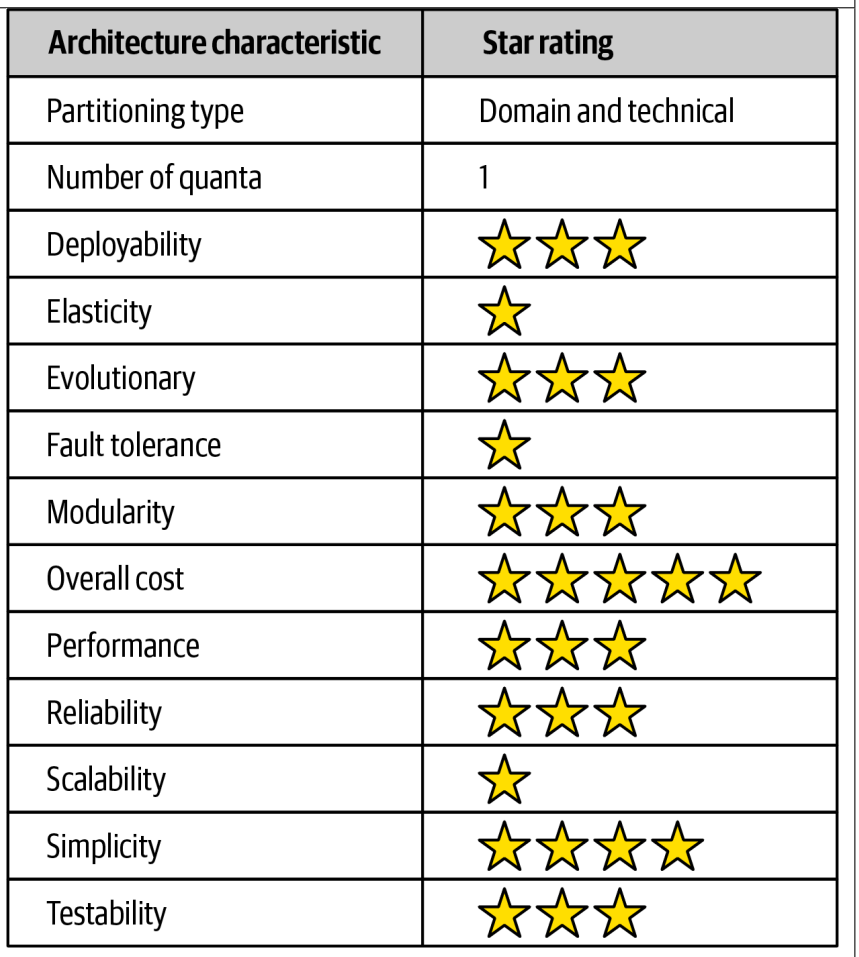
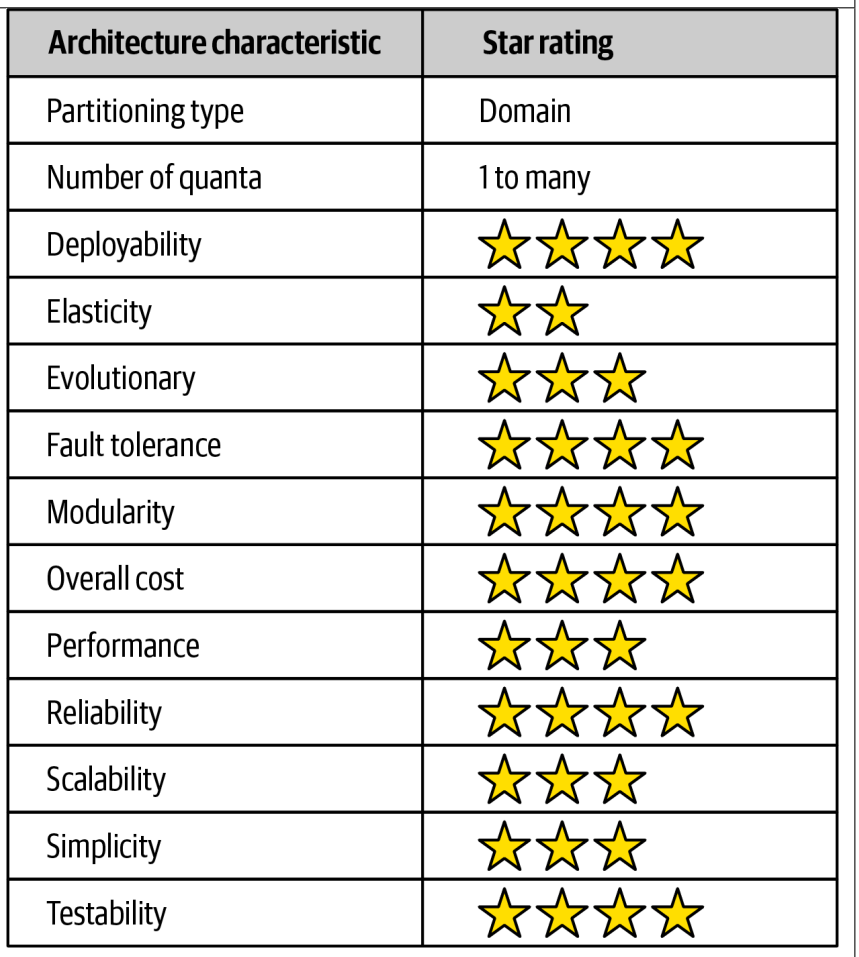
Estilo de Arquitectura	Calificación
Arquitectura de Tuberías (Pipeline Architecture): También conocida como arquitectura de tuberías y filtros (pipes and filters). Comúnmente utilizada para dividir las funcionalidades en múltiples partes separando la responsabilidad de los componentes.	

Tabla 4.15: Calificación del estilo de arquitectura de tuberías según las características de calidad.

Estilo de Arquitectura	Calificación
Arquitectura Microkernel: Este estilo es apto para aplicaciones basadas en productos; sin embargo, es ampliamente utilizado en aplicaciones de negocio personalizadas que no necesariamente son basadas en productos.	

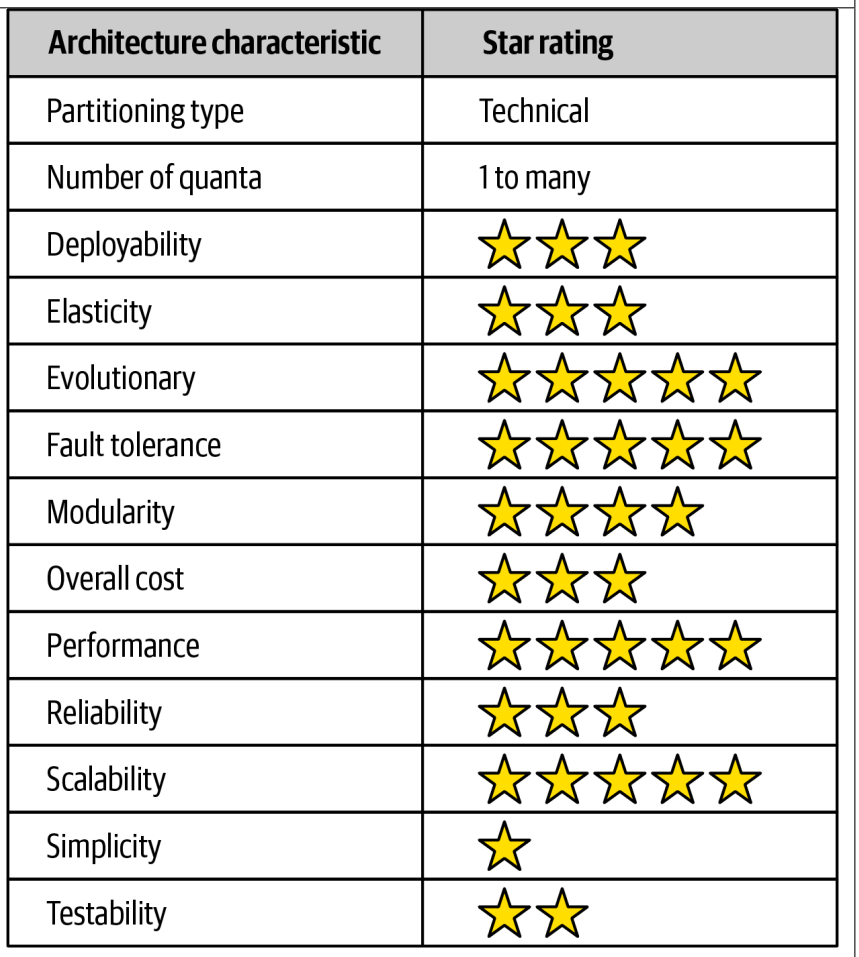
Architecture characteristic	Star rating
Partitioning type	Domain and technical
Number of quanta	1
Deployability	★★★
Elasticity	★
Evolutionary	★★★
Fault tolerance	★
Modularity	★★★
Overall cost	★★★★★
Performance	★★★
Reliability	★★★
Scalability	★
Simplicity	★★★★
Testability	★★★

Tabla 4.16: Calificación del estilo de arquitectura Microkernel según las características de calidad.

Estilo de Arquitectura	Calificación
Arquitectura Basada en Servicios (Service-Based Architecture): Es considerado uno de los estilos de arquitectura más programáticos. Esta es un estilo de arquitectura distribuido, a diferencia de otros estilos de arquitectura distribuida como microservicios y arquitectura orientada a eventos, este es un estilo de menor costo, lo cual lo hace más popular para las aplicaciones empresariales.	

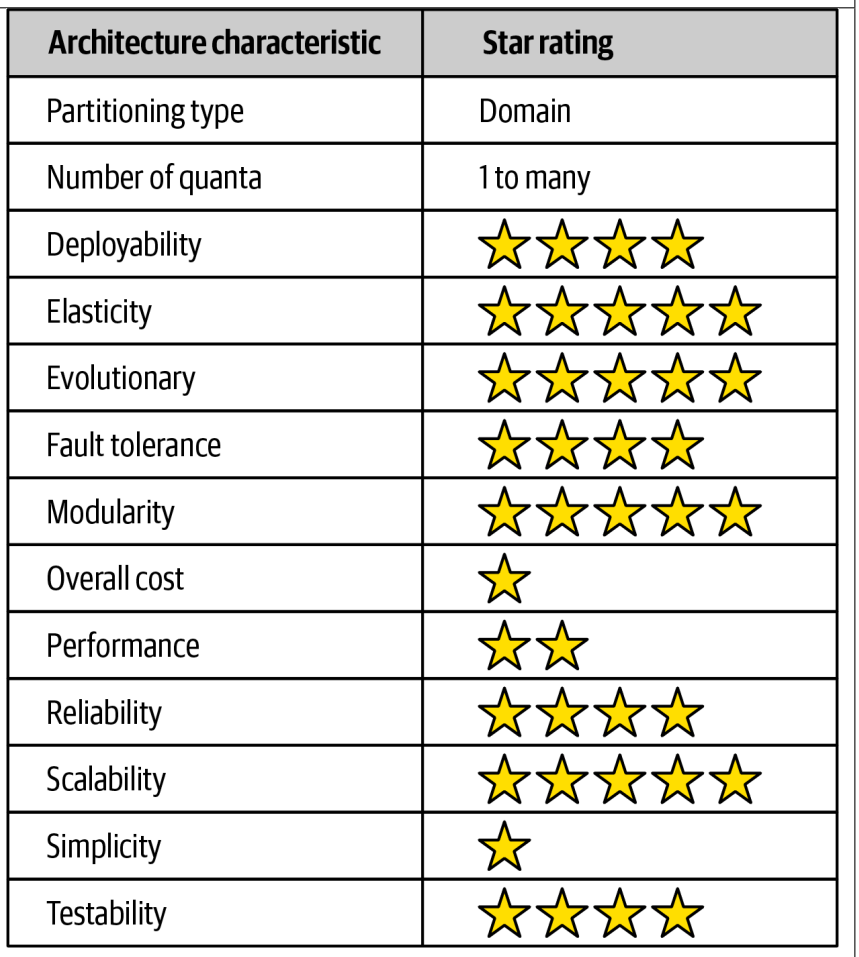
Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★
Evolutionary	★★★
Fault tolerance	★★★★
Modularity	★★★★
Overall cost	★★★★
Performance	★★★
Reliability	★★★★
Scalability	★★★
Simplicity	★★★
Testability	★★★★

Tabla 4.17: Calificación del estilo de arquitectura basado en servicios según las características de calidad.

Estilo de Arquitectura	Calificación
Arquitectura Dirigida por Eventos (Event-Driven Architecture): Estilo de arquitectura asíncrono y distribuido, diseñado para construir aplicaciones altamente escalables y con un alto rendimiento. Este estilo es altamente adaptable y puede ser utilizado en aplicaciones pequeñas como en aplicaciones grandes. Este estilo está pensado para descomponer el procesamiento de eventos en diferentes componentes asíncronos.	

Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1 to many
Deployability	★★★
Elasticity	★★★
Evolutionary	★★★★★
Fault tolerance	★★★★★
Modularity	★★★★
Overall cost	★★★
Performance	★★★★★
Reliability	★★★
Scalability	★★★★★
Simplicity	★
Testability	★★

Tabla 4.18: Calificación del estilo de arquitectura dirigido por eventos según las características de calidad.

Estilo de Arquitectura	Calificación
Arquitectura de microservicios: Estilo de arquitectura inspirado por las ideologías del diseño dirigido por dominios (Domain Driven Design - DDD). Este estilo de arquitectura distribuido propone que cada servicio se ejecute en un proceso propio que originalmente implicó un computador físico, pero escaló rápidamente a máquinas virtuales y contenedores.	

Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★★★★
Evolutionary	★★★★★
Fault tolerance	★★★★
Modularity	★★★★★
Overall cost	★
Performance	★★
Reliability	★★★★
Scalability	★★★★★
Simplicity	★
Testability	★★★★

Tabla 4.19: Calificación del estilo de arquitectura de microservicios según las características de calidad.

Tras revisar múltiples conceptos de diseño como tácticas, patrones y estilos de arquitectura que tienen relación con el atributo o característica de calidad llamada disponibilidad, se procede a seleccionar los conceptos de diseño que logran satisfacer las entradas de esta iteración. En la siguiente tabla se hace un resumen de los conceptos de diseño seleccionados para esta iteración.

Tipo de Concepto	**Nombre Concepto de Diseño**	**Motivo de Selección**
Tácticas	Monitor	Implementar monitores permite supervisar constantemente el estado del sistema y detectar problemas antes de que se conviertan en fallos críticos. Esto es esencial para mantener la disponibilidad y para activar mecanismos de recuperación.
	Heartbeat	Los heartbeat (latidos) son señales periódicas enviadas entre componentes del sistema para confirmar su operatividad. Esto es crucial para detectar fallos en tiempo real y asegurar que todas las partes del sistema estén funcionando correctamente.
	Exception Detection	La detección de excepciones permite identificar y manejar errores inesperados de manera efectiva. Esto minimiza el tiempo de inactividad y evita que los problemas se propaguen, mejorando así la disponibilidad del sistema.
	Exception Handling	El manejo de excepciones es necesario para recuperar el sistema después de un fallo, manteniendo la continuidad del servicio y reduciendo el impacto en los usuarios.
	Redundant Spare	Tener componentes redundantes de repuesto asegura que, en caso de fallo de un componente principal, haya un reemplazo inmediato disponible.

Tipo de Concepto	Nombre Concepto de Diseño	Motivo de Selección
Patrones de Diseño	Balanceo de Carga	El balanceo de carga distribuye eficientemente el tráfico entre varias instancias de servidor, evitando sobrecargas y garantizando que el sistema pueda manejar grandes volúmenes de solicitudes sin degradar el rendimiento.
	Auto-Escalamiento	El auto-escalamiento permite ajustar automáticamente los recursos del sistema en respuesta a la demanda. Esto asegura que haya suficientes recursos disponibles para la demanda presente, mejorando la disponibilidad y capacidad de respuesta del sistema.
	Orquestador	Un orquestador gestiona la coordinación y ejecución de tareas dentro del sistema, asegurando que los procesos se realicen en el orden correcto.
Estilos de arquitectura	Arquitectura Basada en Servicios (SOA)	Este estilo de arquitectura facilita la construcción de sistemas modulares y escalables. Al dividir la funcionalidad en servicios independientes que pueden ser desplegados y gestionados de forma individual, se mejora la disponibilidad al permitir una mejor gestión de fallos y actualizaciones sin afectar a todo el sistema.

Tabla 4.20: Resumen de conceptos de diseño seleccionados para la segunda iteración de ADD

4.5.2.5. Paso 5: Elementos arquitectónicos y responsabilidades

Los elementos arquitectónicos, basado en los servicios de la nube pública de AWS que logran satisfacer las entradas de esta iteración, se describen a continuación especificando las capacidades seleccionadas de cada uno y relacionando los conceptos de diseño cubiertos por dicho elementos arquitectónicos.

Servicio AWS	Descripción	Capacidad Seleccionada	Concepto de diseño relacionado
Elastic Load Balancing	Servicio responsable de distribuir automáticamente el tráfico entrante entre varios destinos, por ejemplo, instancias EC2, contenedores y direcciones IP en una o varias zonas de disponibilidad (Amazon Web Services, 2024c). Permite cubrir la táctica denominada Repuesto Redundante (Redundant Spare) y la táctica Heartbeat, ya que monitorea el estado de los destinos registrados en el balanceador y envía el tráfico únicamente a los nodos o destinos que no presentan errores. Este servicio escala su capacidad de forma automática, dependiendo del tráfico entrante.	Balanceo de Aplicaciones, Integración con el servicio AWS WAF	Táctica Heartbeat, Monitor y Exception Detection. Patrón de balanceo de carga.

Servicio AWS	Descripción	Capacidad Seleccionada	Concepto de diseño relacionado
Availability Zone	Las zonas de disponibilidad se encuentran asociadas a las regiones, cada región de AWS tiene múltiples zonas de disponibilidad. Al igual que las regiones, las zonas de disponibilidad se encuentran aisladas las unas con las otras. Esta propiedad de las zonas de disponibilidad, habilita la alta disponibilidad en el diseño de las aplicaciones, uno de los escenarios que se puede habilitar es el de distribuir los recursos a través de múltiples zonas dentro de una misma región, con lo cual si un recurso falla en una de las zonas, todo el tráfico puede ser enrutado a un recurso diferente que se encuentre operativo en otra zona (Amazon Web Services, 2024a).	Disponibilidad de servicios, resiliencia, infraestructura redundante.	Balanceo de Carga.

Servicio AWS	Descripción	Capacidad Seleccionada	Concepto de diseño relacionado
AWS ECS Fargate	Servicio de Amazon que se define como un motor de computación sin servidor que permite centrarse en la creación de aplicaciones y despreocuparse por la administración de los servidores. AWS Fargate es el complemento perfecto para el servicio de Amazon ECS (Elastic Container Service) el cual permite desplegar aplicaciones contenerizadas y gestionar un clúster de contenedores. Al utilizar AWS ECS junto con AWS Fargate, minimiza la carga operacional y permite que los equipos se enfoquen en el desarrollo de las aplicaciones (Amazon Web Services, 2024b). Este servicio es responsable de contener la aplicación.	Cluster de aplicaciones, Definición de tareas, Contenerización de aplicaciones, Auto-escalamiento	Patón de auto-escalamiento y orquestador. Las tácticas descritas a continuación se basan en su mayoría en el diseño de la aplicación contendida dentro del servicio: Exception Detection y Handling, Redundant Spare. Estilo de arquitectura basado en servicios (SOA)

Tabla 4.21: Elementos arquitectónicos de AWS y responsabilidades iteración dos.

4.5.2.6. Paso 6: Vistas y Decisiones de estilo

El proceso de diseño de vistas se llevó a cabo utilizando el lenguaje de diagramado UML y el metamodelo de la iconografía de AWS. A continuación se presentan las vistas de esta iteración.

El siguiente diagrama de secuencia ilustra el flujo de comunicación entre un usuario, el Balanceador de Carga de Aplicaciones (ALB), el Servicio de Contenedores AWS Fargate ECS, y una tarea específica dentro de ECS. La secuencia inicia con una petición del usuario hacia el ALB, seguido por un chequeo de salud del ALB a las tareas en ECS utilizando la táctica de heartbeat para asegurar su disponibilidad. Una vez se determina que el estado de una tarea es saludable, el ALB redirige la petición a esa tarea. La tarea procesa la petición y envía la respuesta de vuelta al ALB, que a su vez la reenvía al usuario.

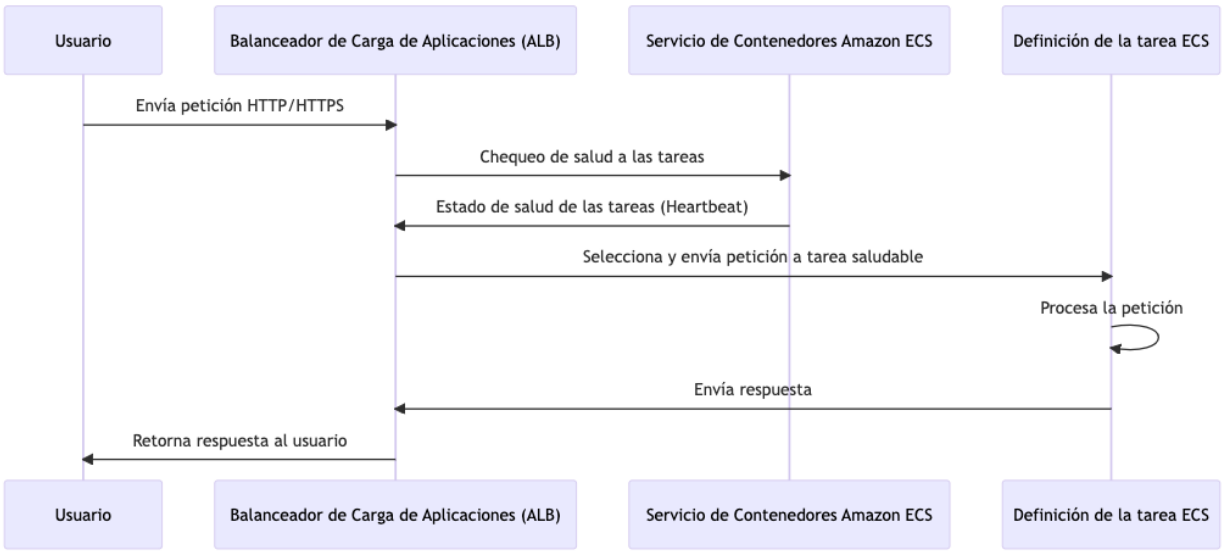


Figura 4.13: Diagrama de secuencia que muestra la interacción exitosa entre un usuario, el Balanceador de Carga de Aplicaciones (ALB) y el Servicio de Contenedores Fargate ECS, destacando el proceso de enrutamiento y manejo de peticiones dentro de una infraestructura de AWS.

La Figura 4.13 muestra el diagrama de secuencia para un flujo en el que la tarea o la definición de la tarea se encuentra en un buen estado de salud, sin embargo, se debe considerar el caso en el que una de las tareas no cuente con un estado exitoso. La siguiente figura contempla el caso en el que el balanceador no logre definir el estado de salud de la aplicación o simplemente, el estado de la aplicación es fallido para saber como actual el sistema en estos casos.

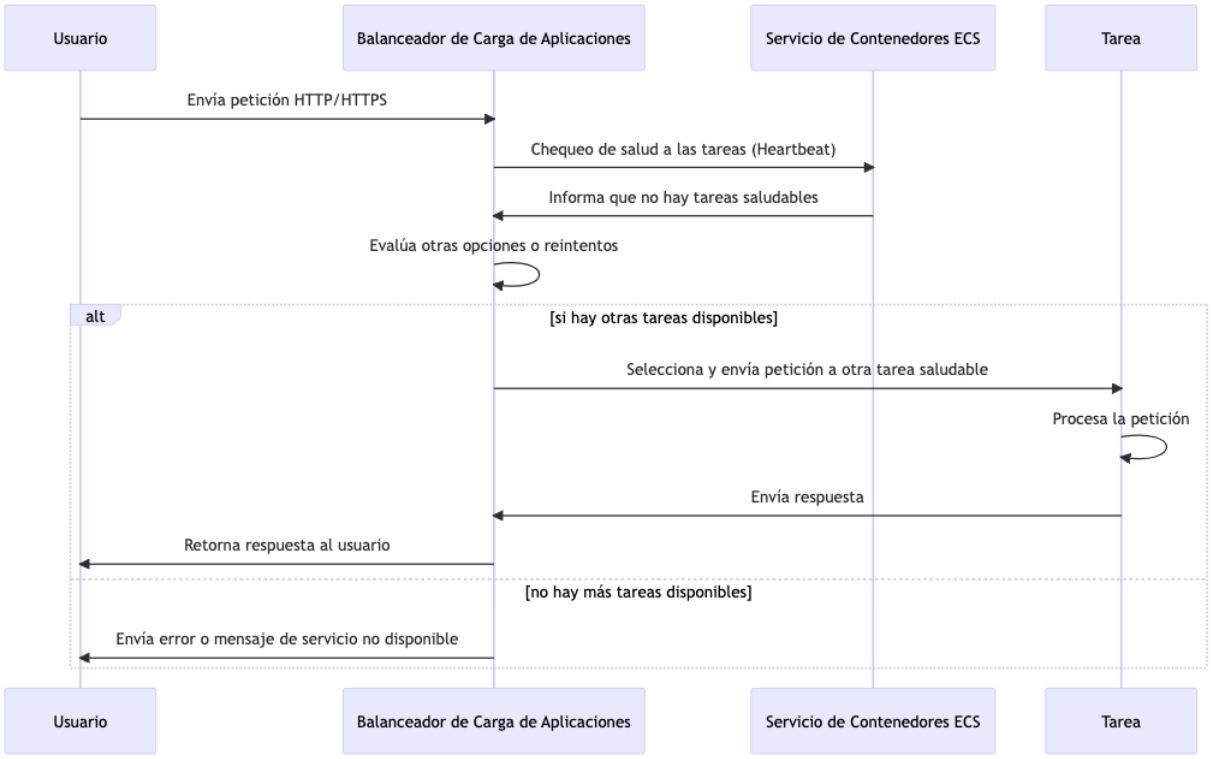


Figura 4.14: Diagrama de secuencia actualizado que muestra la interacción entre un usuario, el Balanceador de Carga de Aplicaciones (ALB) y el Servicio de Contenedores AWS Fargate ECS cuando no hay tareas en estado saludable disponible. Destaca cómo el ALB maneja esta situación evaluando opciones adicionales o enviando un mensaje de error al usuario, asegurando una respuesta adecuada ante fallos en las tareas.

Para finalizar esta sección, en relación con las vistas de arquitectura, se presenta una vista de arquitectura en la nube de AWS en la que se evidencian los elementos arquitectónicos y las interacciones de estos elementos.

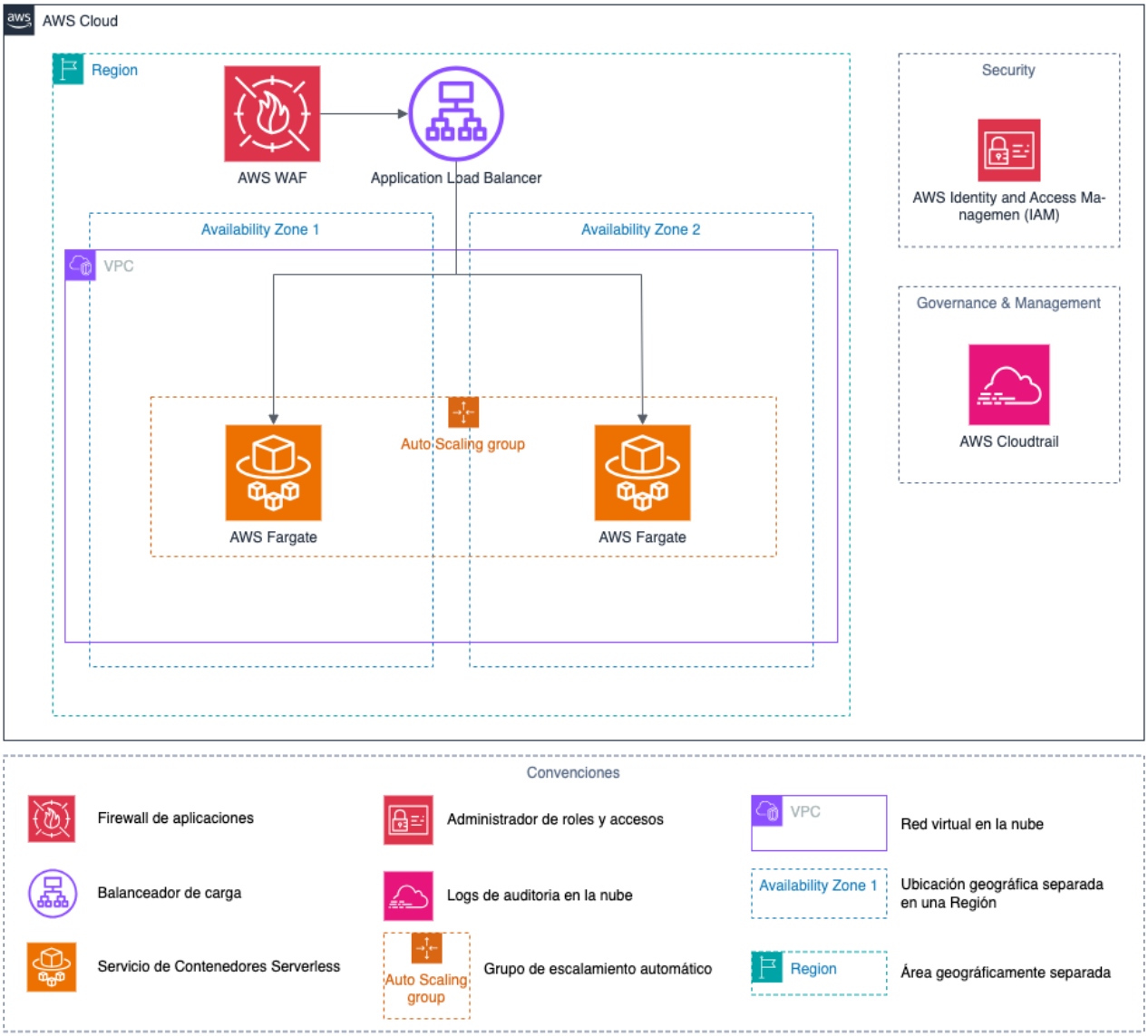


Figura 4.15: Vista de arquitectura resultado de la iteración dos de ADD sobre el atributo de calidad de fiabilidad (disponibilidad).

Decisiones de Arquitectura

A continuación se detallan una de las decisiones de arquitectura tomadas durante el proceso de refinamiento de la iteración. Las demás decisiones pueden ser consultadas en el Apéndice B.

Título	AD-AVA-01 Múltiples Zonas de Disponibilidad y Balanceo de Carga
Estado	Aceptado
Contexto	Uno de los objetivos específicos de este proyecto dice "Implementar mecanismos para garantizar la disponibilidad del trabajador digital", esto se hace necesario dado que un trabajador digital es un nuevo canal de atención para una entidad financiera y es importante que este se encuentre disponible para atender a los usuarios de dicha entidad.
Decisión	El principal mecanismo adoptado fue el de implementar un patrón de arquitectura activo-activo en el que se despliega la aplicación en mínimo dos zonas de disponibilidad y se utiliza un balanceador de carga para distribuir el tráfico entre las zonas. Para desplegar las aplicaciones se decide desplegar sobre el servicio de Amazon ECS haciendo uso del servicio de AWS Fargate el cual elimina la responsabilidad del mantenimiento y la actualización de las instancias sobre las cuales corren los contenedores de Amazon ECS al ser un servicio de contenedores serverless, esto hace que el equipo de desarrollo se enfoque más en la lógica de negocio que en el mantenimiento de la infraestructura.
Consecuencias	▪ Al aplicar un patrón activo-activo, si una instancia falla en alguna de las zonas de disponibilidad, otra instancia entra a reemplazar a la instancia en fallo en cuestión de milisegundos. ▪ Infraestructura gestionada por AWS. AWS Fargate quita la carga operativa de mantener y administrar los servidores y el clúster de aplicaciones. ▪ Distribución del tráfico entrante sobre todas las instancias disponibles.

Tabla 4.22: Decisión de arquitectura del atributo de fiabilidad Núm. 01: Múltiples Zonas de Disponibilidad y Balanceo de Carga.

Validando las decisiones de arquitectura

Esta sección no hace parte de la metodología de ADD; sin embargo, se incluye porque se consideró importante como complemento para validar las decisiones frente a los drivers de arquitectura que hacen parte de la iteración.

A continuación se presenta un caso de prueba detallado junto con la figura que soporta el cumplimiento del mismo.

Título	CP-FIA-01 Usuarios concurrentes inician una conversación
Escenario Relacionado	ver Figura 4.2
Descripción	Una cantidad de cinco usuarios inician una conversación con el trabajador digital al mismo tiempo y se mantienen interactuando con él por un periodo de un minuto.
Resultado esperado	El trabajador digital inteligente responde todas las peticiones sin degradar el servicio.
Ejecución de la prueba	Para la ejecución de la prueba se simuló cinco usuarios virtuales los cuales iniciaron una conversación con el trabajador digital inteligente y se mantuvieron enviando un promedio de 3 solicitudes por 1 minuto, a lo cual el trabajador digital inteligente respondió sin errores con un tiempo de respuesta promedio de 99 milisegundos. Ver Figura 4.16
¿Cumple con el resultado esperado?	Cumple

Tabla 4.23: Caso de prueba: Usuarios concurrentes inician una conversación con el trabajador digital inteligente.

Figura 4.16: Resultado de la ejecución del caso de prueba descrito en la Tabla 4.23 en el que se observa la cantidad de peticiones de usuarios concurrentes, el tiempo promedio de respuesta y la tasa de errores.

4.5.3. Iteración 3: Compatibilidad

4.5.3.1. Paso 1: Entradas de la iteración

En esta iteración, se tienen como entradas los escenarios asociados al atributo de calidad de compatibilidad (ver Sección 4.3.3). Estos escenarios son fundamentales para evaluar cómo interactúan los distintos componentes del sistema en términos de compatibilidad con otros sistemas o versiones previas.

4.5.3.2. Paso 2: Objetivo de la iteración

Diseñar y establecer interfaces estándar y protocolos que permitan la interoperabilidad del sistema con otros sistemas y servicios, tanto internos como externos.

4.5.3.3. Paso 3: Elemento del sistema a refinar

En esta iteración, se va a refinar los elementos del sistema relacionados con la interoperabilidad. Estos elementos incluyen interfaces y protocolos de comunicación entre los componentes del sistema y servicios, internos o externos.

4.5.3.4. Paso 4: Conceptos de Diseño

La sección se divide entre tácticas de integración, patrones, estilos, integración de modelos de procesamiento de lenguaje natural y bases de datos que están estrechamente relacionados con el atributo de calidad de compatibilidad.

Tácticas

El Instituto de Ingeniería de Software (SEI) propone una serie de tácticas agrupadas por su alcance y funcionalidad. Estas tácticas son fundamentales para abordar diversos desafíos en la integración de aplicaciones, la reutilización de componentes y la calidad de los mismos. A continuación se presentan las tácticas propuestas, las cuales abarcan la limitación de dependencias, la adaptación y la coordinación. Estas tácticas son el resultado de investigaciones y mejores prácticas en el campo de la ingeniería de software, y su aplicación puede ayudar a las organizaciones a alcanzar sus objetivos de manera más efectiva y eficiente.

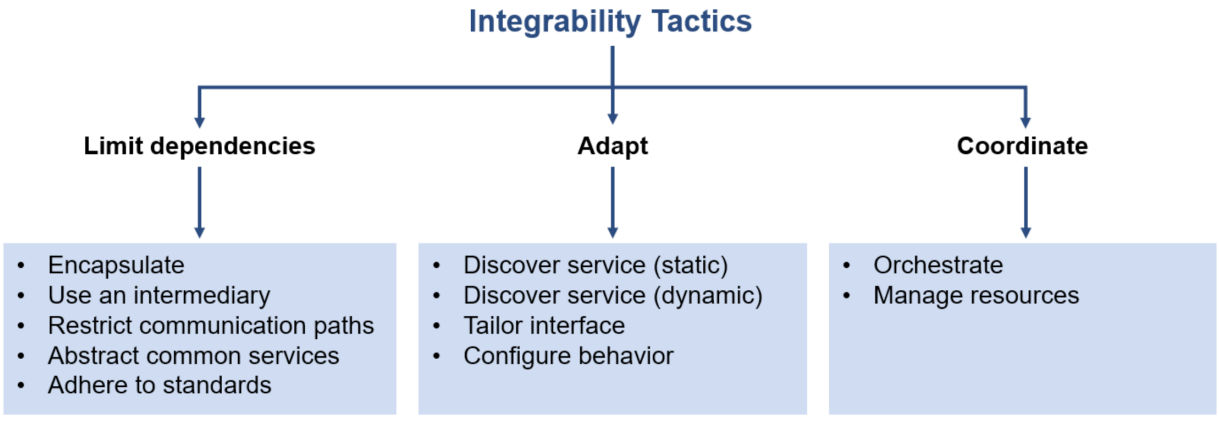


Figura 4.17: Tácticas de Integrabilidad (Kazman et al., 2020).

A continuación se presenta una breve descripción de las tácticas mencionadas anteriormente basada en el libro llamado "Software Architecture In Practice: Fourth Edition" (Bass et al., 2021).

Táctica	**Descripción**
Encapsulate (Encapsular)	Esta táctica propone que todos los componentes cuenten con una interfaz explícita para asegurar que todos los accesos a dicho componente deban pasar por la interfaz. El principal objetivo de esta táctica es reducir la probabilidad de que un cambio en un elemento pueda propagarse a otros elementos.
Use an Intermediary (Utilizar un intermediario)	La táctica de utilizar un intermediario es comente utilizada cuando se busca romper las dependencias entre un conjunto de componentes o entre los componentes y el sistema.
Restrict Communication Paths (Limitar los Paths de comunicaciones)	Esta táctica tiene por objetivo restringir o limitar a un conjunto de elementos con que se pueden comunicar un sistema. Esta táctica puede aplicarse para ocultar elementos y para la autorización, como por ejemplo, la restricción de acceso a algunos componentes para usuarios no autorizados.

Adhere to Standards (Adherirse a los estándares)	Cuando se habla de integración y de interoperación, esta táctica es esencial, ya que los estándares son los que permiten que dos o más sistemas o componentes puedan interactuar adecuadamente. Existen diferentes tipos de estándares y varían dependiendo de su alcance y su aplicación o adopción.
Abstract Common Service (Servicio común abstracto)	Un caso de uso común para esta táctica es cuando se tiene dos elementos que proveen servicios similares y se busca ocultar la implementación mediante una abstracción común para formar un servicio más general. Esta abstracción puede ser una interfaz común implementada por ambos servicios. Como resultado de encapsular, se tiene como beneficio el poder ocultar el detalle de los elementos para otros componentes del sistema.
Discovery (Descubrimiento)	Táctica que propone la definición de un catálogo de direcciones relevantes, las cuales son necesarias ser traducidas de una a otra. Esta táctica sirve como mecanismo para que las aplicaciones y los servicios se puedan comunicar. Un ejemplo donde podemos ver aplicada esta táctica es en un servidor DNS, el cual logra traducir nombres de dominio a direcciones IP, lo que facilita que los usuarios puedan acceder a una aplicación sin conocer su dirección IP ni sus detalles, este es un ejemplo de descubrimiento estático.
Tailor Interface (Interfaces a la medida)	Agrega la capacidad de añadir o eliminar capacidades a una interfaz existente si modificar la API o su Implementación. Algunas capacidades que se pueden incorporar son las de traducción, almacenamiento en búfer y suavizado de datos. Algunas capacidades que se pueden eliminar son las de ocultar funciones particulares o parámetros para usuarios no estructurados. Esta táctica es comúnmente utilizada en la integración de aplicaciones y su principal uso suele ser el de resolver la semántica y la sintáctica de los datos durante el proceso de integración.

Configure Behavior (Comportamiento personalizable)	Esta táctica se suele utilizar con componentes de software que necesitan ser implementados, pero deben tener la capacidad de ser personalizables, dicha personalización se realiza en tiempo de construcción (comúnmente se utiliza diferentes banderas).
Orchestrate (Orquestación)	Utiliza un mecanismo de control para coordinar y administrar la invocación de los servicios, sí que estos sepan cuál es el siguiente en ejecutarse. Esta táctica es muy útil cuando se quiere integrar un conjunto de servicios que se encuentran bajamente acoplados y son altamente reutilizables. La táctica de orquestación trabaja para reducir el número de dependencias entre un sistema y los nuevos componentes y elimina la dependencia entre los componentes, a su vez, centraliza las dependencias en el mecanismo de orquestación.
Manage Resources (Administrar los recursos)	Un administrador de recursos, hace las veces de un intermediario, el cual ejerce un gobierno sobre el acceso a los recursos de computación. Esta táctica busca limitar el acceso directo de los componentes de un sistema a los recursos computacionales como hilos o bloques de memoria, si algún componente requiere acceder a estos recursos, debe pasar por un administrador de recursos. Algunos ejemplos son los sistemas operativos, mecanismos transaccionales en las bases de datos, uso de los thread pools (grupo de hilos) en sistemas empresariales, entre otros.

Tabla 4.24: Descripción de las tácticas de integrabilidad.

Patrones

La utilización de patrones de integración en una arquitectura de software aporta diferentes beneficios. Estos patrones ofrecen soluciones probadas y reutilizables para problemas comunes de integración, lo que ahorra tiempo y esfuerzo en el desarrollo e implementación. Además, facilitan el mantenimiento y la comprensión de la arquitectura, la hacen escalable y flexible.

El libro "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" (Hohpe and Woolf, 2003) registra sesenta y cinco patrones de integración empresarial clasificados en seis grupos, como se muestra en la Figura 4.18.

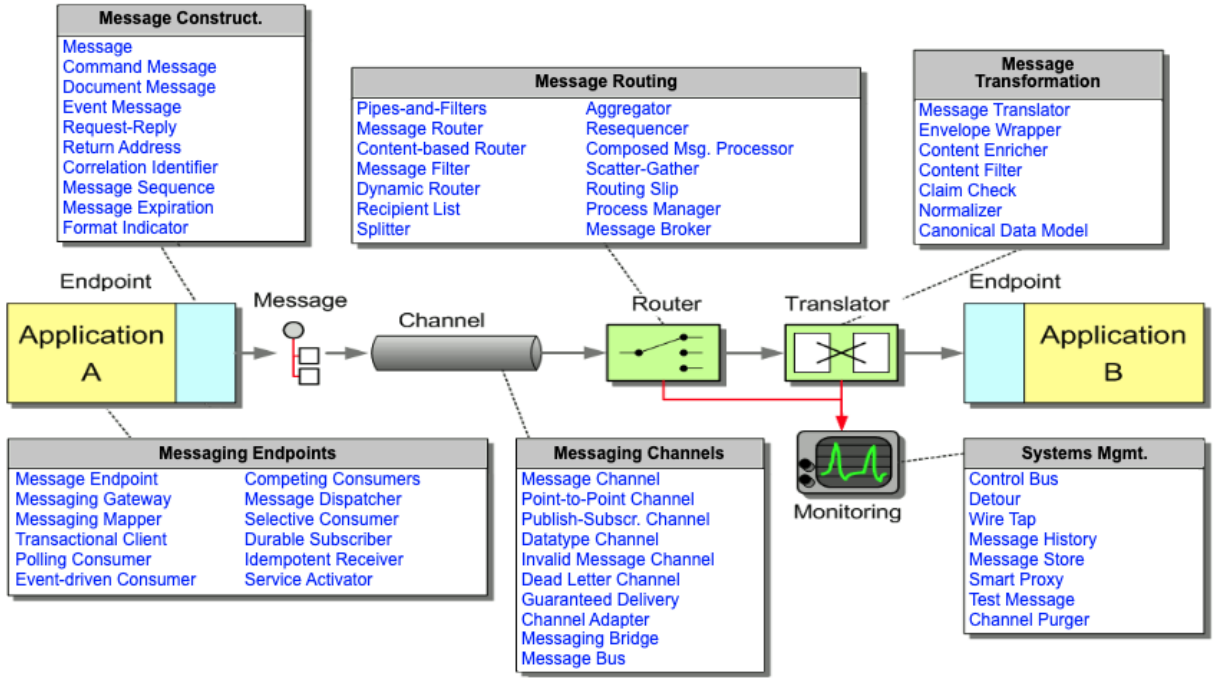


Figura 4.18: Patrones de integración empresarial (Hohpe and Woolf, 2003).

Sumado a lo anterior, Amazon Web Services plantea ocho patrones de diseño enfocados en la nube, en la guía de implementación llamada, "AWS Prescriptive Guidance: Cloud design patterns, architectures, and implementations" (Deenadayalan and Amazon Web Services Inc, 2024) los cuales están basados en el estilo de arquitectura de microservicios. A continuación se presenta un resumen de los patrones propuestos por AWS en la guía de implementación.

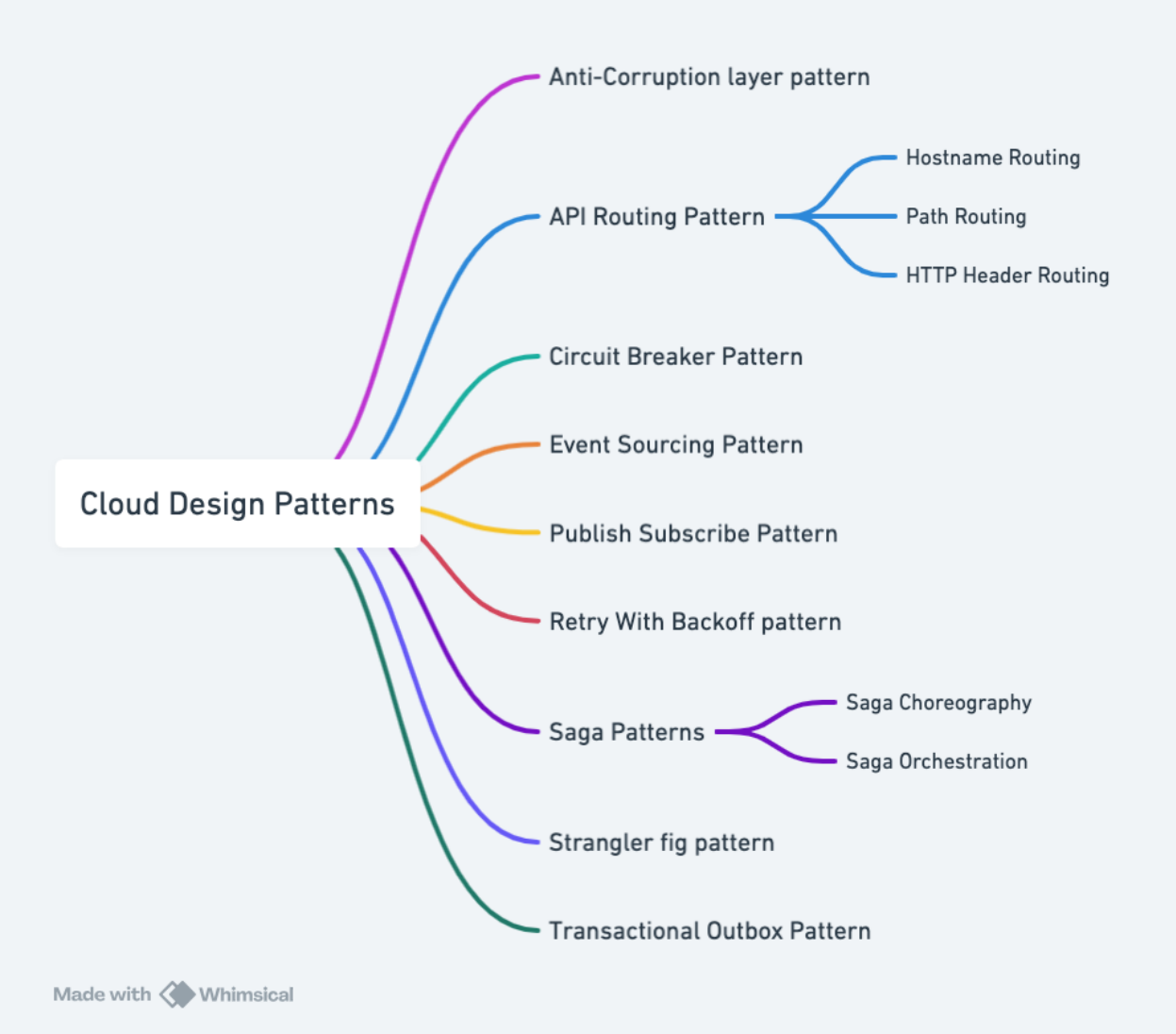


Figura 4.19: Patrones de diseño en la nube.

Después de revisar los patrones previamente mencionados, se eligieron una serie de patrones que cumplen con los requisitos de esta iteración. Estos patrones fueron cuidadosamente evaluados para garantizar que se alinearan adecuadamente con los objetivos y criterios establecidos para esta fase del proyecto. A continuación, se presenta una tabla que enumera los patrones seleccionados junto con sus características principales para una mejor comprensión y referencia.

Patrón	Descripción	Caso de uso
API Routing Pattern (Path Routing)	Este patrón plantea la exposición de cada uno de los servicios en un PATH diferente, ejemplo: my.domain.com/service-a, my.domain.com/service-b. Al enrutar por Paths, se puede agrupar múltiples o todos los servicios en un mismo nombre de host y separar los servicios con una URI de solicitud, como se muestra en el anterior ejemplo.	Método simple de enrutar las solicitudes por el protocolo HTTP que comúnmente utilizado por qué es de fácil acceso, ya que todos los servicios se encuentran expuestos por un mismo nombre de host (dominio) como my.domain.com.
Content Enricher Pattern	La interacción de dos sistemas puede ser algo compleja, es muy común que el sistema que recibe el mensaje, requiera más información de la que el sistema que envía tiene disponible. El patrón de contenido enriquecido utiliza la información dentro del mensaje entrante para agregar más información basado en este desde una fuente externa. La información original del mensaje puede estar contenida en el mismo o no necesariamente, depende del caso de uso.	Este patrón se suele utilizar para resolver las referencias contenidas en un mensaje, manteniendo el mensaje original lo más pequeño y fácil de manejar. Estas referencias usualmente toman la forma de llaves o de identificadores únicos.
Request-Reply Pattern	Generalmente la comunicación entre dos aplicaciones suele ser de una sola vía; sin embargo, este patrón propone que debe haber dos actores uno "Solicitante" y uno "Contestador". El solicitante envía el mensaje y espera por un mensaje de respuesta. El Contestador recibe el mensaje y responde con un nuevo mensaje. El Solicitante tiene dos formas de recibir el mensaje, la primera como forma sincrónica y la segunda de forma asincrónica	Esto se aplica en escenarios como la comunicación entre microservicios en arquitecturas de microservicios, sistemas cliente-servidor, integración de sistemas empresariales y comunicación web, donde la espera de respuestas es fundamental para coordinar las acciones entre los componentes involucrados.

Message Router (Enrutador de Mensajes)	El patrón Message Router se utiliza para enrutar mensajes desde una fuente hacia uno o más destinos, basándose en ciertos criterios de enrutamiento. Puede determinar el destino de un mensaje utilizando reglas definidas previamente o basándose en el contenido del mensaje.	Enrutar solicitudes de clientes a diferentes servicios en una arquitectura de microservicios.
Message Translator (Traductor de Mensajes)	Este patrón permite traducir mensajes entre diferentes formatos o protocolos para permitir la comunicación entre sistemas que utilizan tecnologías distintas.	Traducir mensajes entre diferentes formatos y protocolos para permitir la interoperabilidad entre sistemas heterogéneos. Por ejemplo, convertir mensajes en XML a JSON o viceversa para facilitar la comunicación entre sistemas que utilizan diferentes formatos de datos.

Tabla 4.25: Patrones de integración seleccionados.

Método de desarrollo

El método Low-code/No-code (LC/NC) permite el desarrollo de aplicaciones con un conocimiento básico o nulo de programación tradicional. Esta metodología fue evaluada en el Capítulo 3 y seleccionada para esta iteración debido a su capacidad para acelerar la construcción del prototipo funcional, el cual va a ser utilizado como mecanismo de evaluación de la arquitectura propuesta. Este prototipo requiere la integración de un modelo de procesamiento de lenguaje natural, el cual se incorpora utilizando el método de desarrollo de software LC/NC.

Técnicas para integrar e interactuar con un modelo de procesamiento de lenguaje natural (LLM)

Como entrada de esta iteración, se requiere de la integración con un modelo de inteligencia artificial que cuente con la capacidad de realizar un procesamiento de lenguaje natural, para ello como concepto de diseño se evaluaron tres técnicas para integración e interacción con modelos de inteligencia artificial que procesan lenguaje natural, estos estilos se describen a continuación.

Nombre	Descripción
Reasoning and Acting (ReAct)	Esta táctica se basa en la forma en la que los seres humanos razonamos y respondemos a una pregunta, para ello propone que los agentes cognitivos entiendan la pregunta que se les hace haciendo una descomposición de la misma para posteriormente generar una respuesta acorde a la pregunta realizada (Yao et al., 2022). En la Figura 4.20 se plantea un contraste en como un modelo de inteligencia artificial que procesa lenguaje natural puede responder utilizando la técnica ReAct y como responde sin utilizar dicha técnica.
Retrieval-Augmented Generation (RAG)	Técnica también considerada como un estilo de arquitectura que busca proporcionar un contexto mayor, también llamado conocimiento de una fuente externa a un modelo de procesamiento de lenguaje natural (LLM). Esta técnica busca reducir de manera efectiva el problema que presentan los LLM de generar respuestas incorrectas, para ello, utiliza fuentes de conocimiento externas, las cuales pueden ser públicas o privadas y son utilizadas para complementar la pregunta de un usuario antes de ser enviadas como contexto adicional al LLM. Esta técnica se divide en tres: RAG nativo, RAG avanzado y RAG modular (Gao et al., 2023). La Figura 4.21 representa la arquitectura RAG nativa.

MRKL (pronunciado "miracle") System	MRKL es la combinación de dos conceptos: **Modular Reasoning** (Rasonamiento Modular) y **Knowledge and Language** (Conocimiento y Lenguaje). La técnica de sistema MRKL consiste en un conjunto de módulos extensibles que son llamados "expertos", lo que busca es que por cada mensaje de entrada al modelo de procesamiento de lenguaje natural (NLP), se enrute dicha entrada a uno de los módulos el cual está en la capacidad de responder de mejor manera acerca de un tema en específico. La respuesta de un módulo puede ser el resultado de ejecutar un sistema MRKL o la entrada de otro módulo. Los módulos pueden ser Neurales o Simbólicos; Los módulos neurales pueden incluir un gran modelo de lenguaje de propósito general, como también modelos de lenguaje más pequeños y especializados. Los módulos simbólicos pueden ser por ejemplo una calculadora, un conversor de divisas o una llamada a una API (Karpas et al., 2022).

Tabla 4.26: Tácticas para interacción e integración de modelos de procesamiento de lenguaje natural para el desarrollo de agentes cognitivos.

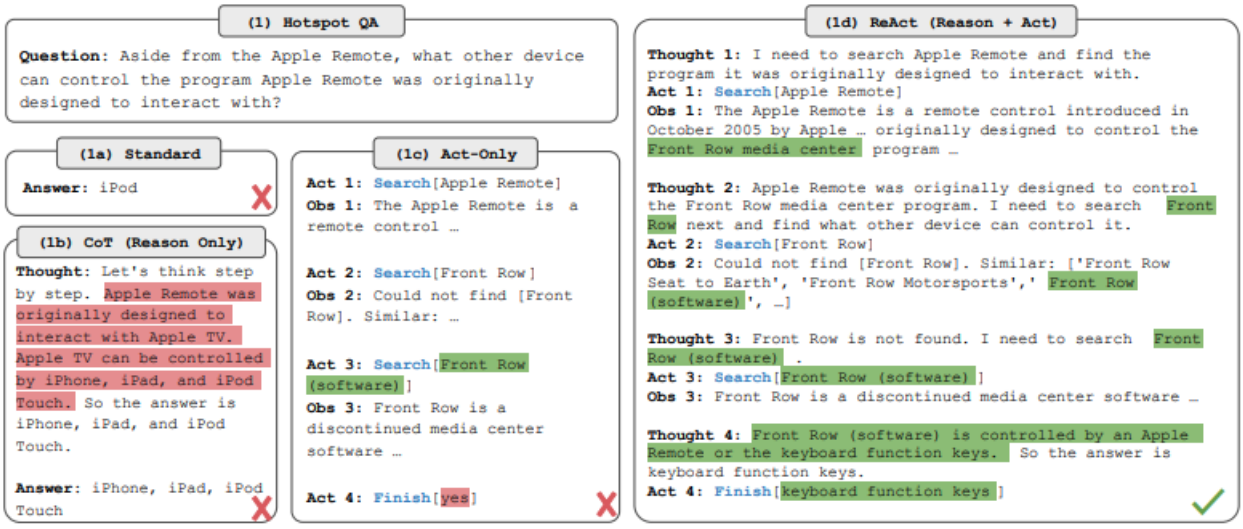


Figura 4.20: (1) Comparación entre cuatro métodos de Prompt (a) Standard, (b) Chain-of-Thought (Cadena de pensamiento) CoT, (c), Actuación únicamente y (d) Respuesta utilizando la técnica de razonamiento y actuación o ReAct (Yao et al., 2022).

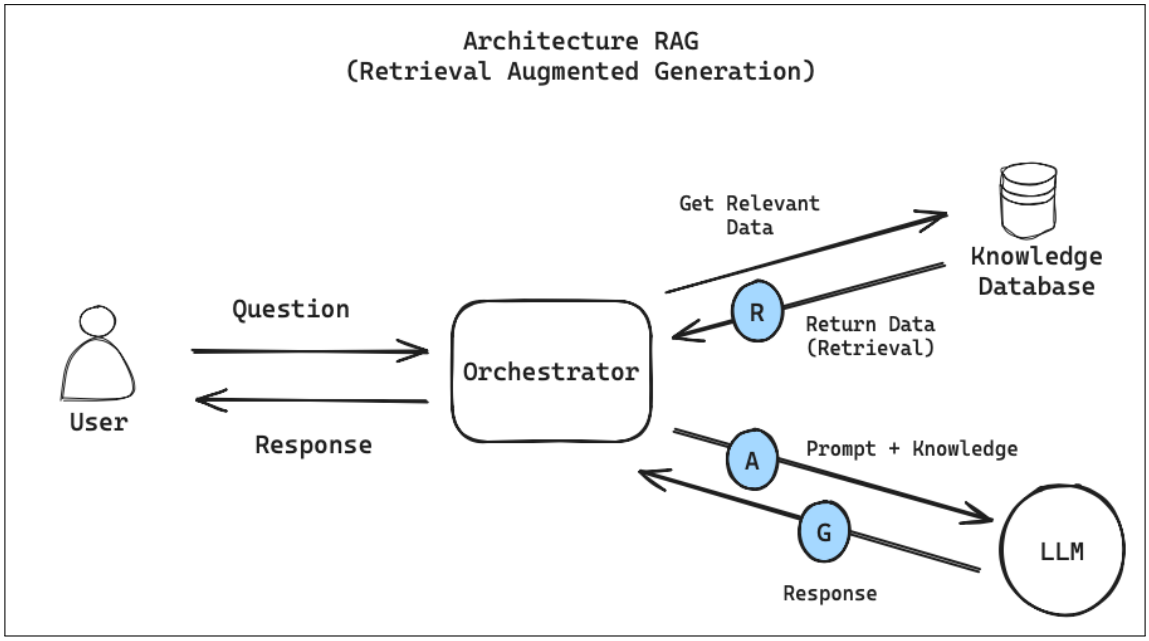


Figura 4.21: Representación de la técnica RAG que describe el flujo que sigue la comunicación con el LLM.

Bases de Datos

Otro concepto de diseño que es fundamental tener en cuenta para la integración en la arquitectura es el de bases de datos, las bases de datos más comunes son según su clasificación son: Bases de datos relacionales, bases de datos no relacionales y bases de datos orientadas a objetos.

Para esta iteración, considerando que la arquitectura, que está en proceso de refinamiento, emplea modelos de procesamiento de lenguaje natural (NLP) y que la mayoría de las técnicas previamente mencionadas para integrar e interactuar con modelos de NLP utilizan la representación vectorial del texto para llevar a cabo búsquedas basadas en un concepto conocido como "TOP K", donde K representa el número de vectores más cercanos al texto introducido por el usuario en cada pregunta, se enfocará esta sección en revisar brevemente los beneficios y características de las bases de datos vectoriales.

Yikun Han Han et al. (2023) define que las bases de datos vectoriales son bases de datos que se encuentran clasificadas como no relacionales, ya que no siguen el modelo relacional y son un tipo de bases de datos que almacenan datos como vectores de alta dimensión, que son la representación matemática de características o atributos. Cada vector tiene un cierto número de dimensiones, este puede ser un rango de diez o cientos dependiendo de la complejidad y granularidad de los datos.

En el mismo artículo científico, Yikun Han resalta tres beneficios de las bases de datos vectoriales. El principal y más importante es encontrar de manera rápida y precisa las palabras similares a una palabra que se esté buscando, esta es la funcionalidad más adecuada y utilizada para el procesamiento de lenguaje natural (NLP).

El segundo beneficio es el soporte que ofrecen para datos complejos y sin estructura como texto, imágenes, audio, video, etc. Un tercer beneficio es el de rendimiento y escalabilidad, estas bases de datos pueden manejar procesamiento de grandes volúmenes de información, incluso son capaces de procesar y analizar información en tiempo real, esta característica es esencial para aplicaciones de ciencias de datos y de inteligencia artificial.

A continuación, se enumeran las bases de datos vectoriales más relevantes, mayormente consideradas de código abierto u open source, junto con un caso de uso común, la relación entre el rendimiento y la escalabilidad. De manera similar, en la Figura 4.22 se presentan los logotipos de estas bases de datos para una mejor referencia.

Base de datos	**Caso de uso**	**Rendimiento/Escalabilidad**
Faiss	Base de datos en memoria con la capacidad de búsqueda semántica, análisis de similitud, recomendación de sistemas	Alto/Muy alto
Chroma	Búsqueda semántica y análisis de sentimientos	Alto/Alto
Vald	Búsqueda semántica y análisis de similitud	Alto/Alto
Pinecone	Búsqueda semántica y análisis de similitud	Alto/Alto
Vespa	Búsqueda semántica, análisis de similitud y análisis de texto	Alto/Alto
Milvus	Búsqueda semántica, análisis de similitud y recomendación de sistemas	Alto/Alto
Qdrant	Búsqueda semántica y análisis de similitud	Alto/Alto
Weaviate	Búsqueda semántica, análisis de similitud y recomendación de sistemas	Alto/Alto

Tabla 4.27: Lista de las bases de datos vectoriales más relevantes que describe un caso de uso común y referencia el nivel de rendimiento y escalabilidad.

Figura 4.22: Bases de datos vectoriales más relevantes.

Como resumen de esta sección, se presenta a continuación una tabla con el resumen de los conceptos de diseño seleccionados que satisfacen apropiadamente las entradas de esta iteración.

Categoría	**Nombre del concepto**	**Motivo de selección**
Técnicas de Integración	Discovery (Descubrimiento)	La táctica de descubrimiento cobra mucha relevancia en nuestra arquitectura, especialmente debido a su estrecha relación con el patrón de balanceador de carga seleccionado en la iteración anterior. Esta táctica fue seleccionada por su capacidad de proporcionar un componente central que conoce las direcciones de los demás servicios y actúa como un registro de servicios dinámico. Este componente no solo facilita la localización y comunicación entre servicios, sino que también permite una escalabilidad y flexibilidad mejoradas al adaptarse automáticamente a los cambios en la infraestructura.

Categoría	**Nombre del concepto**	**Motivo de selección**
	Configure Behavior	Permite la configuración dinámica de comportamientos en los servicios, lo que es esencial para la adaptabilidad y la personalización en tiempo real de la arquitectura basada en AWS. Esta técnica se abarca de manera efectiva con el método de desarrollo seleccionado.
	Orchestrate (Orquestación)	Coordina múltiples servicios y flujos de trabajo, mejorando la eficiencia y la gestión de procesos complejos. La orquestación es una pieza clave dentro de una arquitectura para asegurar que los servicios interactúen de manera efectiva.
	Adhere to Standards	Garantiza la interoperabilidad y la compatibilidad con otros sistemas, facilitando la integración y la conformidad con las mejores prácticas de la industria. Adherirse a estándares asegura que la arquitectura pueda comunicarse y funcionar correctamente dentro de un ecosistema más amplio.
Patrones de Integración	API Routing Pattern	Simplifica la gestión de las rutas y la lógica de direccionamiento de las solicitudes API. Este patrón es esencial para un enrutamiento eficiente y flexible, asegurando que las solicitudes lleguen a los servicios correctos sin complicaciones. Va de la mano con la táctica de discovery.

Categoría	Nombre del concepto	Motivo de selección
	Content Enricher Pattern	Agrega información adicional a los mensajes, mejorando la calidad y la utilidad de los datos transmitidos entre servicios. Esto permite que los servicios puedan tomar decisiones más informadas basadas en datos enriquecidos. Patrón que se relaciona con técnica de integración del modelo de procesamiento de lenguaje natural (LLM) seleccionada y llamada RAG.
	Request-Reply Pattern	Establece una comunicación síncrona entre servicios, permitiendo la solicitud y recepción de respuestas en tiempo real. Este patrón es fundamental para las interacciones que requieren una confirmación inmediata o una respuesta rápida.
	Message Translator Pattern	Transforma los mensajes entre diferentes formatos o protocolos, asegurando que los servicios puedan comunicarse sin problemas a pesar de las diferencias en sus interfaces. Este patrón es crucial para la interoperabilidad en un entorno de servicios heterogéneos.
Integración de modelo LLM	Retrieval-Augmented Generation (RAG)	Combina capacidades de recuperación y generación de información, mejorando la precisión y la relevancia de las respuestas proporcionadas por modelos de lenguaje. Esta técnica es fundamental para aplicaciones avanzadas de IA que requieren respuestas informadas y contextualmente relevantes.
Método de desarrollo de software	Low-code/No-code	Acelera el desarrollo de aplicaciones permitiendo a los usuarios crear soluciones con poco o ningún código.

Categoría	Nombre del concepto	Motivo de selección
Bases de datos	Bases de datos Vectoriales	Las bases de datos vectoriales son seleccionadas por su capacidad para almacenar y buscar datos vectoriales de alta dimensionalidad eficientemente. Son esenciales en aplicaciones de IA y machine learning que requieren búsquedas rápidas y precisas en grandes volúmenes de datos no estructurados, como texto, imágenes y otras formas de datos embebidos.

Tabla 4.28: Conceptos de diseño seleccionados para satisfacer las entradas de la iteración número tres (3) de ADD.

4.5.3.5. Paso 5: Elementos arquitectónicos y responsabilidades

Esta sección describe los elementos arquitectónicos seleccionados y se relacionan con los conceptos de diseño seleccionados en la sección anterior que incluyen técnicas de integración, patrones de integración, metodologías de desarrollo, integración de modelos de lenguaje natural y bases de datos especializadas, todos los cuales son esenciales para construir una arquitectura robusta y eficiente.

Elemento Arquitectónico	Descripción	Capacidad utilizada	Concepto de diseño relacionado
Plataforma LC/NC: FlowiseAI y Make	Plataforma Low-code/No-code que permite integrar un modelo de inteligencia artificial y orquestar un flujo basado en una arquitectura RAG.	Orquestación de nodos, integración de modelos, exposición de servicios tipo REST API, consumo de servicios REST API	Método de desarrollo de software. Tácticas como: configure behavior, orchestrate y Adhere to Standards.
Amazon Bedrock	Servicio Administrado por AWS que pone a disposición modelos de inteligencia artificial fundacionales con un alto rendimiento (Amazon Web Services Inc, 2024a). Este servicio provee el modelo de procesamiento de lenguaje natural para la arquitectura.	Despliegue de modelos fundacionales serverless	Modelo de procesamiento de lenguaje. Táctica de integración al modelo LLM llamada RAG. Patón message translator.

Elemento Arquitectónico	Descripción	Capacidad utilizada	Concepto de diseño relacionado
Amazon Simple Storage Service (Amazon S3)	"Servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento" Amazon Web Services Inc (2024c). Este servicio es responsable de almacenar la información relevante que se va a cargar en una base de datos vectorial para ser consultada como base de datos de conocimiento en la arquitectura RAG.	Creación de Buckets de propósito general, carga y consulta de información de los buckets.	Patrón Content Enricher. Táctica de integración al modelo LLM llamada RAG.
Base de datos vectorial	Para aplicar la arquitectura RAG, es necesario proveer una base de conocimiento, el proceso de carga de información para esta base es a manera de vectores (Gao et al., 2023), este elemento contiene las representaciones vectoriales de los documentos que van a servir como fuente de información para la arquitectura.	Creación de base de datos y creación de índices.	Bases de datos Vectoriales

Tabla 4.29: Elementos arquitectónicos y sus responsabilidades iteración tres.

4.5.3.6. Paso 6: Vistas y Decisiones de estilo

En esta sección se presentan las vistas de diseño como resultado del refinamiento junto con la documentación de las decisiones de arquitectura.

El siguiente diagrama de secuencia muestra cómo interactúan los servicios de AWS Fargate y Amazon Bedrock en un flujo de procesamiento de una conversación iniciada por un usuario. El usuario envía un mensaje a través de WhatsApp, que es recibido por la plataforma LC/NC llamada Make la cual se integra con WhatsApp por un webhook que utiliza el protocolo HTTP/HTTPS. Make automatiza el flujo de comunicación y envía el mensaje al orquestador desarrollado en la plataforma LC/NC llamada FlowiseAI la cual se encuentra contenerizada y desplegada en AWS Fargate. El componente orquestador obtiene información no estructurada (archivos de texto, PDF, imágenes, fotos, audios) del producto desde el repositorio de archivos Amazon S3, transforma esta

información en una representación vectorial y consulta Pinecone DB para obtener los vectores más cercanos. Con esta información, el orquestador envía una consulta a Amazon Bedrock, que procesa y devuelve una respuesta. Finalmente, el orquestador genera la respuesta y la envía a Make, que luego la envía a WhatsApp, que la entrega al usuario.

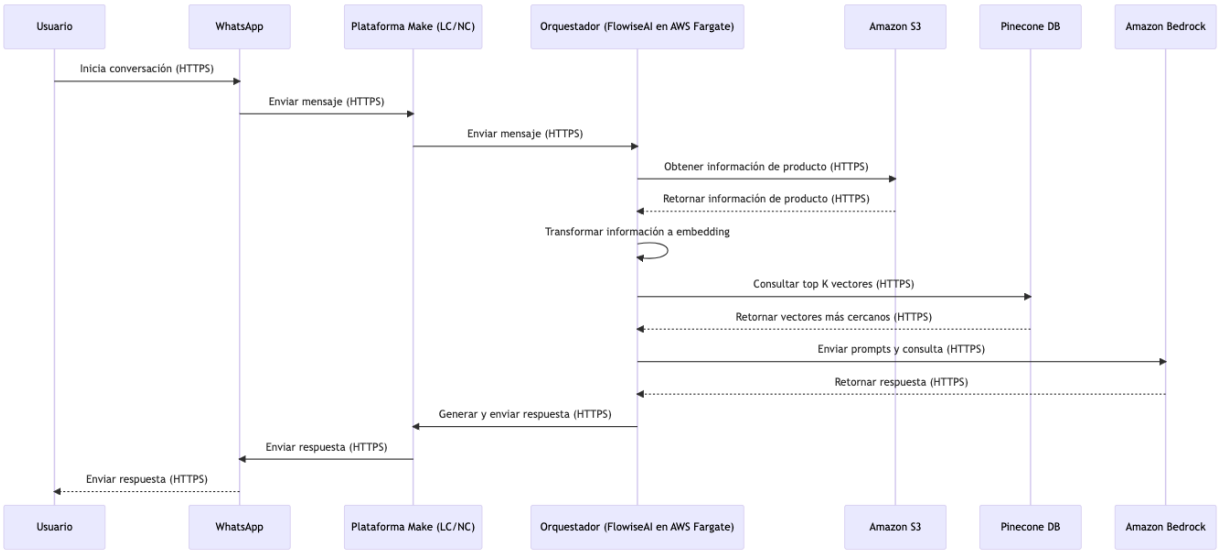


Figura 4.23: Diagrama de Secuencia que muestra la interacción entre el usuario, WhatsApp, la plataforma LC/NC Make, y el orquestador desarrollado en FlowiseAI desplegado en AWS Fargate con su respectivo protocolo de comunicación.

La siguiente es una vista de clases que representa la orquestación del flujo conversacional aplicando la técnica RAG (Retrieval Augmented Generation) en la plataforma de Low-Code/No-Code FlowiseAI. La clase “S3” es la encargada de la integración con el servicio llamado AWS S3 para consultar los documentos del producto, la clase “AWSBedrockEmbeddings” integra un modelo de lenguaje extendido (LLM) responsable de transformar el texto en lenguaje natural a una representación vectorial, estas dos clases en mención son utilizadas por la clase “Pinecone” la cual se integra con el proveedor de bases de datos vectoriales Pinecone seleccionado como concepto de diseño y hace uso de las clases “S3” y “AWSBedrockEmbeddings” para transformar los documentos, hacer una partición en los denominados “chunks” y almacenarlos en el índice de la base de datos. Por otro lado, la clase “AWSChatBedrock” realiza la integración del modelo de lenguaje extendido para el procesamiento de lenguaje natural y por último la clase

ConversationalRetrievalQAChain es la encargada de integrar la clase “Pinecone” y “AWSChatBedrock” con las cuales interactúa con el modelo de procesamiento de lenguaje natural y genera las respuestas.

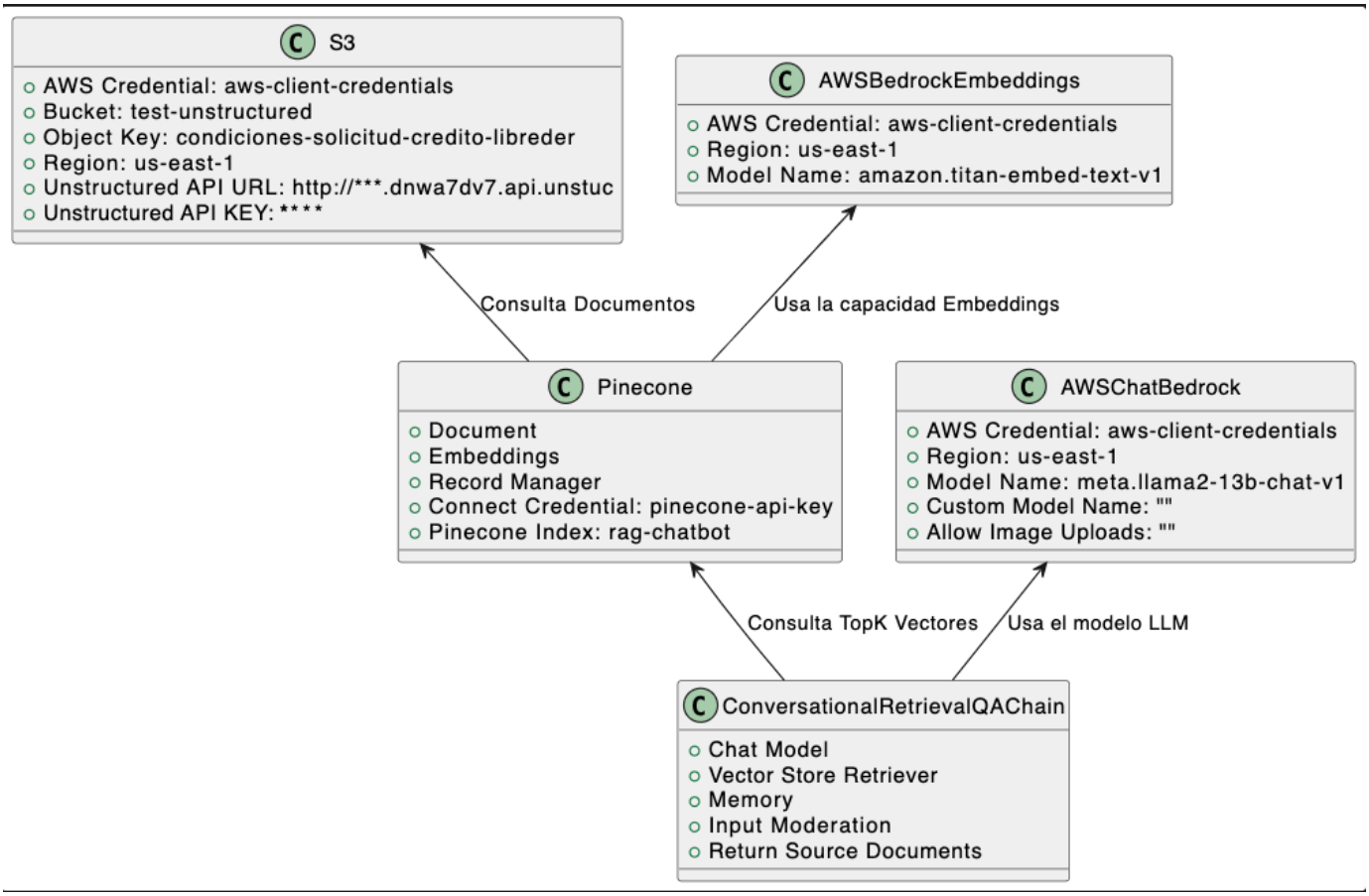


Figura 4.24: Diagrama de clases del modelo conversacional utilizando la técnica RAG en la plataforma LC/NC FlowiseAI.

La siguiente figura es una vista aterrizada sobre los componentes de AWS en la cual se ven aplicados los elementos arquitectónicos seleccionados en la iteración actual, además de los elementos arquitectónicos seleccionados en las iteraciones anteriores.

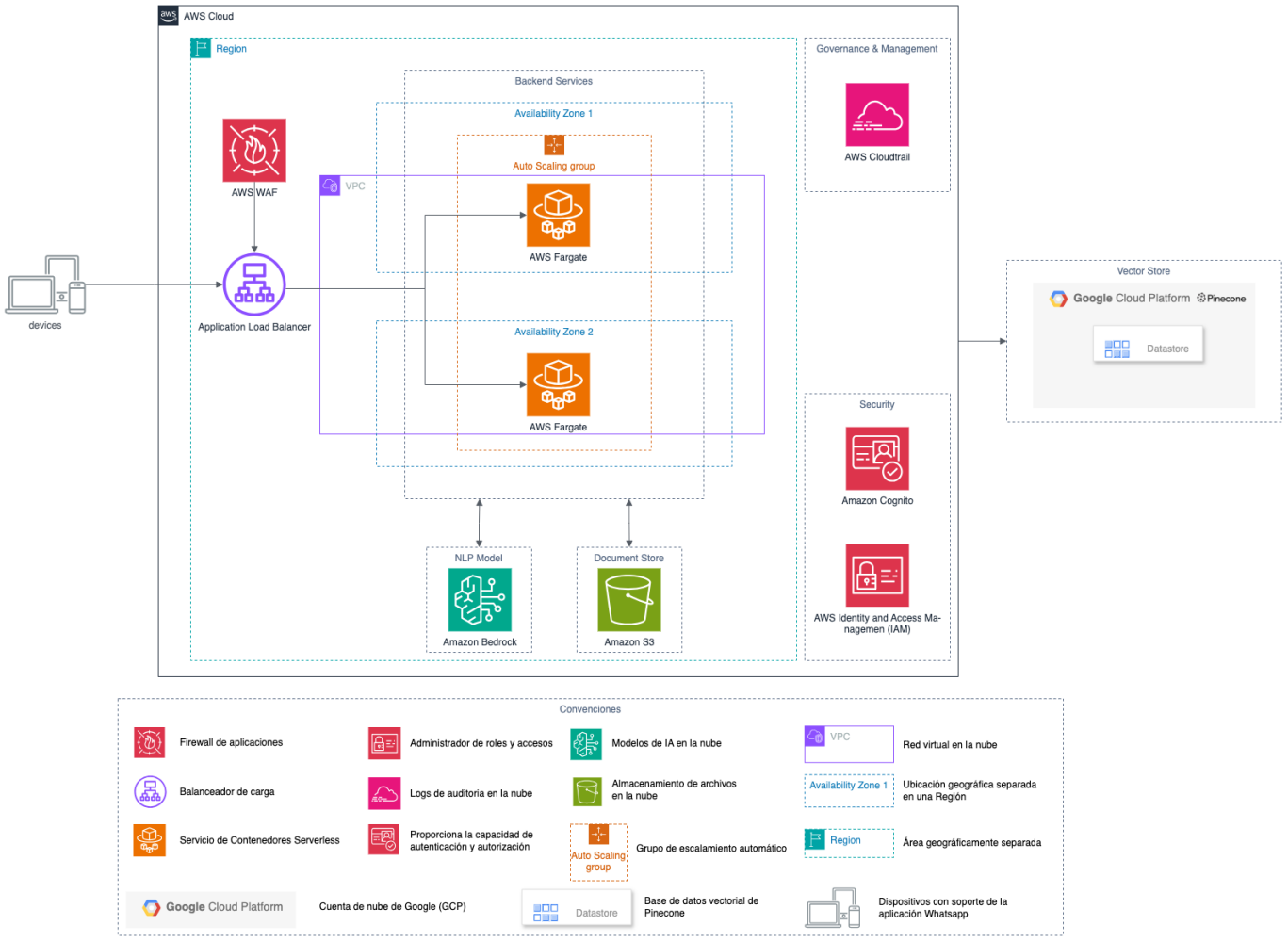


Figura 4.25: Vista de arquitectura resultado de la iteración tres de ADD sobre el atributo de calidad de Compatibilidad (Interoperabilidad).

Decisiones de Arquitectura

A continuación se detallan una de las decisiones de arquitectura tomadas durante el proceso de refinamiento de la iteración. Las demás decisiones pueden ser consultadas en el Apéndice B.

Título	AD-INT-01 Herramienta de desarrollo
Estado	Aceptado
Contexto	Parte de la etapa de evaluación del diseño de la arquitectura refinada mediante ADD es realizar la implementación de un prototipo funcional, dicho prototipo tiene un requerimiento principal relacionado con la infraestructura, la cual debe ser desplegada previamente en la nube pública de AWS, adicional a ello, requiere unos pasos de desarrollo para integrar los diferentes componentes que interactúan con el modelo de procesamiento de lenguaje natural.
Decisión	Tras revisar el Capítulo 3 donde se realiza una investigación exhaustiva de la metodología de desarrollo Low-code/No-code, se decide utilizar una herramienta que utiliza dicha metodología para acelerar el proceso de desarrollo e integración de los diferentes componentes, para ello, se utilizara la herramienta llamada "Flowise IA" la cual utiliza las librerías de Langchain para realizar la conexión y el orquestamiento de diferentes llamados a los modelos de procesamiento de lenguaje natural.
Consecuencias	▪ Acelera el proceso de desarrollo del prototipo funcional. ▪ Simplifica la interoperabilidad con el modelo de procesamiento de lenguaje natural. ▪ Limita la capacidad de personalización de las interacciones con el modelo de procesamiento natural. ▪ Habilita la capacidad de que personas con bajos conocimientos técnicos en desarrollo puedan modificar o agregar funcionalidades a los componentes que interactúan con el modelo de NLP.

Tabla 4.30: Decisión de arquitectura del atributo de compatibilidad Núm. 01: Herramienta de desarrollo.

Validando las decisiones de arquitectura

Esta sección no hace parte de la metodología de ADD; sin embargo, se incluye porque se consideró importante como complemento para validar las decisiones frente a los drivers de arquitectura que hacen parte de la iteración.

A continuación se presenta un caso de prueba detallado junto con la figura que soporta el cumplimiento del mismo. Los demás casos de prueba pueden ser consultados en el Apéndice C.

Título	CP-COMP-01 Integración con la base de datos vectorial
Escenario Relacionado	ver Figura A.7
Descripción	Componente conversacional requiere consultar a la base de datos para obtener el Top K de vectores más cercanos a la pregunta de un usuario.
Resultado esperado	La base de datos vectorial retorna la cantidad de vectores esperados.
Ejecución de la prueba	ver Figura 4.26
¿Cumple con el resultado esperado?	Cumple

Tabla 4.31: Caso de prueba: Integración con la base de datos vectorial.

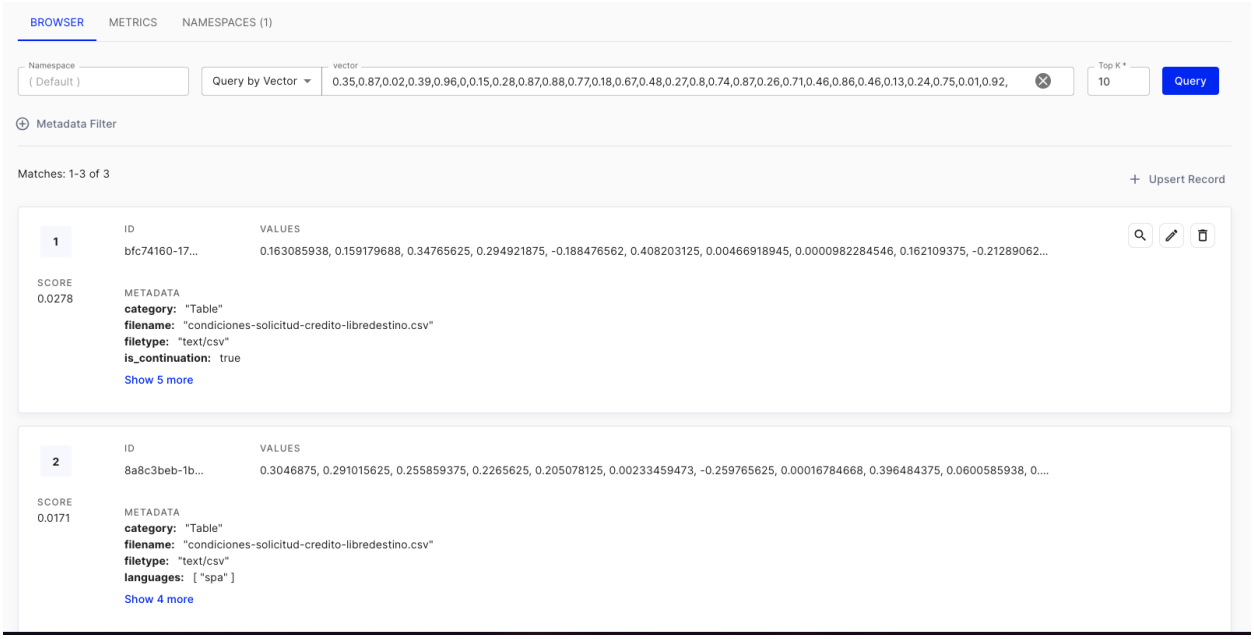


Figura 4.26: Resultado de la ejecución del caso de prueba descrito en la Tabla 4.31 donde se observa los vectores insertados en la base de datos como resultado de la integración.

4.6. Arquitectura final refinada por la metodología ADD

Para finalizar este capítulo, tras realizar un refinamiento exhaustivo de cada uno de los atributos de calidad priorizados, seleccionando cuidadosamente los conceptos de diseño y los elementos arquitectónicos, se presenta a continuación la arquitectura de referencia en la nube de AWS. Esta arquitectura soporta la plataforma destinada a la venta consultiva de un producto digital basado en inteligencia artificial.

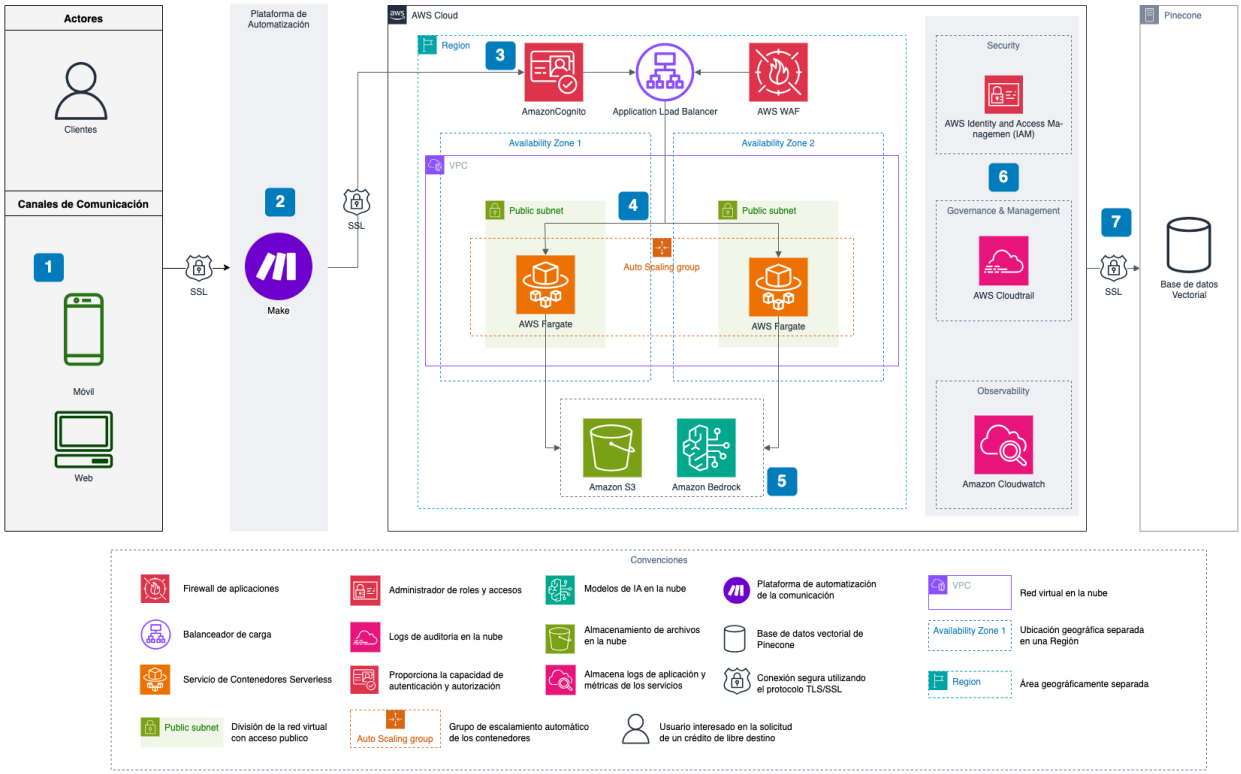


Figura 4.27: Plataforma tecnológica para habilitar la venta consultiva de un producto digital con inteligencia artificial

A continuación, se presentan las anotaciones respectivas a cada punto de la arquitectura de referencia representada en la Figura 4.27.

1. Canales de comunicación desde los cuales se puede iniciar una conversación con el trabajador digital inteligente, estos canales deben soportar la aplicación de mensajería instantánea WhatsApp.

2. Plataforma Low/No-Code llamada Make la cual disponibiliza un webhook para comunicarse con la aplicación de WhatsApp y automatiza la comunicación entre la arquitectura de referencia desplegada en la nube de AWS y la aplicación de mensajería WhatsApp.

3. Las peticiones pasan por un balanceador de aplicaciones, el cual tiene habilitada una capa de seguridad por medio del servicio AWS WAF el cual escanea las solicitudes entrantes y aplica una serie de reglas para determinar si la solicitud es segura, en este caso la solicitud pasa hacia los contenedores de AWS Fargate, en caso de que la solicitud no logre pasar las reglas de AWS WAF, la petición es bloqueada. Este balanceador también se encuentra protegido por cognito, para lo cual la aplicación consumidora, debe solicitar un token OAUTH 2.0 y posterior a esto pasar el token en una cabecera hacia el balanceador para que este valide si el token es válido, si es válido el consumo es permitido, de lo contrario se deniega el consumo.

4. Clúster de contenedores desplegado en un esquema de alta disponibilidad distribuido en dos zonas de disponibilidad y configurado con una regla de auto escalamiento para soportar las cargas de trabajo. Este clúster contiene la aplicación con el flujo conversacional desarrollado en la plataforma Low/No-Code llamada FlowiseAI.

5. Servicios necesarios para el funcionamiento de la aplicación. Amazon S3 provee el servicio de almacenamiento de archivos no estructurados sobre el cual se carga la información del producto para ser consultado por la aplicación y ser indexado en la base de datos vectorial. Amazon Bedrock provee los modelos de lenguaje extendido (LLM) requeridos por la aplicación para indexar los archivos obtenidos de Amazon S3 en la base de datos vectorial y para procesar el lenguaje natural para la comunicación con el usuario.

6. Servicios importantes para asegurar la seguridad, el gobierno y la observabilidad dentro de la nube.

7. Base de datos vectorial desplegada en la plataforma llamada Pinecone la cual se conecta con arquitectura en la nube usando protocolos de comunicación segura (SSL).

Capítulo 5

Desarrollo del Prototipo

5.1. Contexto general

Para este proyecto, se planteó el diseño de una arquitectura de software en la nube de AWS que habilite una plataforma tecnológica para la venta consultiva de un producto digital utilizando inteligencia artificial (IA). Como mecanismo de evaluación de la arquitectura diseñada, se propuso desarrollar un prototipo funcional.

Este capítulo se centra en la implementación del prototipo funcional, el cual permitió evaluar que la arquitectura diseñada esté acorde al contexto y a los requisitos de negocio planteados en el Capítulo 4.

5.2. Prototipo funcional

Para validar el diseño de arquitectura base refinado por la metodología de diseño por atributos de calidad (ADD) (ver Sección 4.2), se desarrolló un prototipo funcional, el cual tiene como alcance desplegar la infraestructura refinada en la nube pública de AWS, cargar la información base para el producto financiero denominado crédito de libre destino y exponer el trabajador digital inteligente de manera tal que pueda ser consumido desde la aplicación de WhatsApp.

5.2.1. Despliegue de la infraestructura en la nube

En la actualidad, es posible implementar la infraestructura en la nube por medio de dos métodos. El primero consiste en realizar un despliegue manual desde la consola de administración de la nube, mientras que el segundo implica una automatización mediante el uso de infraestructura como código (IaC). Para el presente proyecto, se optó por la utilización de infraestructura como código mediante un framework denominado Terraform. Esta herramienta es independiente del proveedor de servicios en la nube donde se desee implementar la infraestructura. Además, ofrece soporte para desplegar en los principales proveedores de servicios de nube pública, como AWS, Azure, GCP, Oracle, entre otros. Asimismo, permite versionar de la infraestructura en un repositorio de código y la gestión del estado de la misma.

A continuación, se muestra la estructura del proyecto de infraestructura como código desarrollado usando el framework de Terraform.

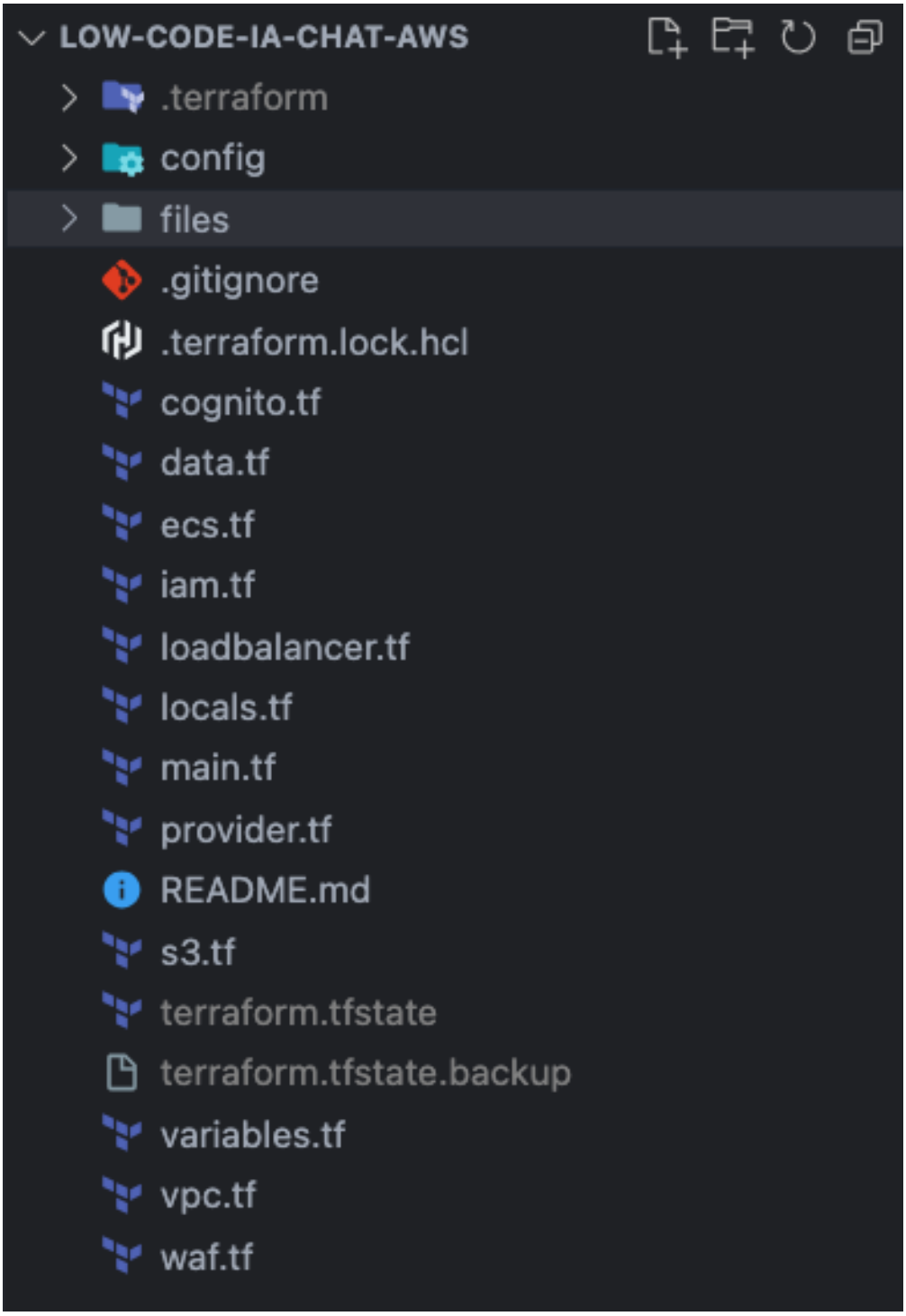


Figura 5.1: Estructura del proyecto de infraestructura utilizando Terraform como lenguaje de IaC.

Como se observa en la imagen anterior, el proyecto de infraestructura como código contiene los componentes de infraestructura en la nube separados en diferentes archivos como: `s3.tf`, `ecs.tf`, `main.tf` para separar las responsabilidades y facilitar la administración del proyecto, sin embargo, existen unos archivos de configuración sobre los cuales da más detalle a continuación.

- `environment.tfvars`: Contiene el archivo de configuración de variables, este archivo inicializa el valor de las variables definidas en el archivo `variables.tf`
- `variables.tf`: Contiene la definición de las variables utilizadas para aprovisionar la infraestructura, es muy útil para poder aprovisionar infraestructura en diferentes ambientes como: desarrollo, calidad y producción.
- `provider.tf`: Contiene la especificación del proveedor de nube, para este proyecto el proveedor de nube es AWS.
- `locals.tf`: Contiene las constantes utilizadas para el despliegue de la infraestructura, en este se encuentra la información de etiquetas comunes para los servicios correspondientes a este proyecto, algunos prefijos y sufijos, entre otros.
- `data.tf`: Contiene los denominados data sources o fuentes de datos, son recursos que sirven par obtener información o propiedades de recursos que ya se encuentran creado en la cuenta de nube, para este caso se obtiene en este archivo los valores de la VPC y de las subredes.
- `README.md`: Registra la documentación del proyecto de infraestructura, esta documentación contiene los comandos para validar, desplegar, actualizar y eliminar la infraestructura en la nube, a demás de los prerrequisitos para utilizar terraform para desplegar infraestructura como código.
- `iam.tf`: Archivo en el que se registran todos los roles y permisos que van a ser asociados a los recursos de infraestructura.

Para controlar las versiones del repositorio de infraestructura como código se utilizó la herramienta llamada Git. Git es una herramienta de código abierto u open source diseñada para manejar las versiones de grandes y pequeños proyectos de forma eficiente.

Las versiones del código en Git se pueden representar como “commits”, etiquetas (“tags”) o despliegues (“releases”), para este proyecto se manejaron las versiones como “commits”. En la figura 5.2 se evidencia las diferentes versiones generadas para el proyecto de infraestructura como código.

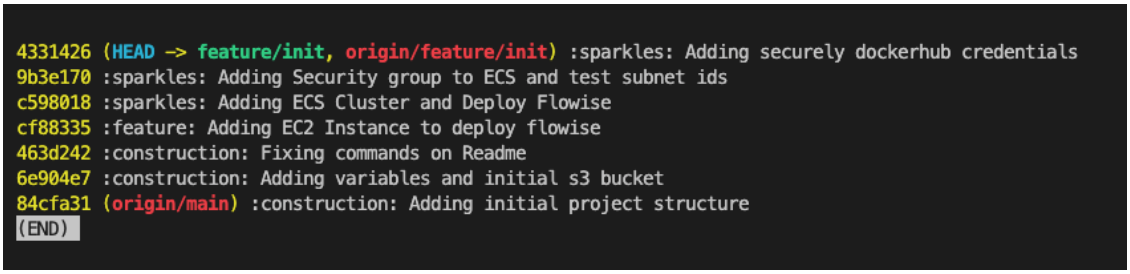


Figura 5.2: Versiones del proyecto de infraestructura como código.

5.2.2. Despliegue de herramienta Low-code/No-code: Flowise IA

Flowise IA fue la herramienta seleccionada para desarrollar la lógica de la aplicación y a su vez habilitar la integración con otros sistemas. Esta herramienta utiliza la metodología de desarrollo Low-code/No-code (LC/NC) para realizar las integraciones.

El despliegue de esta herramienta se hizo utilizando contenedores Serverless con el servicio de AWS Fargate y Amazon ECS, las arquitecturas Serverless permiten enfocarse en la aplicación, ya que eliminan la administración y gestión de servidores o de infraestructura.

Para desplegar servicio de AWS Fargate y Amazon ECS se utilizaron tres recursos de Terraform, el primero para desplegar el cluster de contenedores `aws_ecs_cluster`, el segundo para desplegar la definición de la tarea `aws_ecs_task-definition` y el tercero para desplegar el servicio `aws_ecs_service`. A continuación se describe brevemente la responsabilidad de cada uno de los artefactos mencionados.

- **Cluster**: El cluster provee una agrupación lógica de las tareas o contenedores, al igual que de los servicios, y brinda un ambiente aislado para ejecutar las aplicaciones.
- **Definición de la tarea**: Contiene todas las especificaciones para desplegar el contenedor, usualmente se especifica en una tarea (task definition) los recursos como cantidad de memoria, CPU, variables de entorno, imágenes con las que van a ser construidos los contenedores, entre otros.
- **Servicio**: Puede ser una tarea o un conjunto de tareas de larga ejecución administradas por Amazon ECS. Permite mantener un cierto número de tareas (task definitions) ejecutándose de forma simultánea en un clúster. Se utiliza comúnmente para escalar la aplicación de forma automática.

```
################
## ECS Cluster
################
resource "aws_ecs_cluster" "flowise_cluster" {
  name = "flowise-cluster-tf"

  setting {
    name  = "containerInsights"
    value = "enabled"
  }

  tags = local.tags
}
```

Figura 5.3: Cluster de contenedores en Amazon ECS.

El cluster de la Figura 5.3 contiene las configuraciones básicas para el despliegue, el bloque `setting` es opcional, se habilitó para poder visualizar la información de los contenedores.

```
##################################
## ECS Task Definition (Container)
##################################

resource "aws_ecs_task_definition" "flowise_task_definition" {
  family                   = "flowise-task-definition-tf"
  requires_compatibilities = ["FARGATE"]
  network_mode             = "awsvpc"
  cpu                      = 1024
  memory                   = 2048

  container_definitions = jsonencode([…
  ])

  runtime_platform {
    operating_system_family = "LINUX"
    cpu_architecture        = "X86_64"
  }

  task_role_arn = aws_iam_role.ecs_task_execution_role.arn

  tags = local.tags
}
```

Figura 5.4: Task Definition para desplegar el contenedor de Flowise IA con Amazon ECS.

La definición de la tarea de la Figura 5.4 contiene una serie de configuraciones necesarias para el despliegue de la herramienta Flowise IA. A continuación se describen las continuaciones más relevantes:

- `family`: nombre único para la definición de la tarea.
- `requires_compatibilities`: Específica si los contenedores se van a desplegar de forma serverless con AWS Fargate o si se van a desplegar en máquinas virtuales en Amazon EC2, para este caso se desplegará en AWS Fargate.
- `network_mode`: específica de qué manera el contenedor a desplegar va a utilizar los recursos de red, en este caso va a utilizar la VPC de la cuenta de AWS.
- `task_role_arn`: Role creado en Amazon IAM que habilita el acceso del contenedor a los

recursos de AWS por medio del service `sts:AssumeRole` como se muestra en la Figura 5.5.

- `container_definition`: Propiedad que contiene la definición del contenedor como imagen, variables de entorno, puertos, recursos como memoria y CPU. La definición del contenedor se especifica en la Figura

```
resource "aws_iam_role" "ecs_task_execution_role" {
  name = "ecs-task-execution-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Service = "ecs-tasks.amazonaws.com"
        }
        Action = "sts:AssumeRole"
      }
    ]
  })
}
```

Figura 5.5: Role que habilita el acceso a los recursos de nube al contenedor desplegado en Amazon ECS.

```
container_definitions = jsonencode([
  {
    name      = "flowise-container"
    image     = "flowiseai/flowise:1.6.0"
    cpu       = 1
    memory    = 2048
    essential = true
    command = [
      "flowise",
      "start"
    ]
    environment = [
      {
        name  = "DOCKER_USERNAME"
        value = var.docker_hub_username
      },
      {
        name  = "DOCKER_PASSWORD"
        value = var.docker_hub_password
      }
    ]
    portMappings = [
      {
        containerPort = 80
        hostPort      = 80
      }
    ]
  }
])
```

Figura 5.6: Definición del contenedor de Flowise IA.

A continuación, se especifican los puntos más relevantes de la configuración del contenedor:

- `container_definitions`: Define la configuración de un contenedor en formato JSON.

- `name:` Nombre del contenedor, en este caso "flowise-container".
- `image:` Imagen del contenedor a utilizar, en este caso "flowiseai/flowise:1.6.0".
- `cpu:` Especifica la cantidad de CPU asignada al contenedor (1 unidad).
- `memory:` Especifica la cantidad de memoria asignada al contenedor (2048 MB).
- `essential:` Indica si el contenedor es esencial para la tarea principal (true en este caso).
- `command:` Comando a ejecutar dentro del contenedor al iniciarlo.
- `environment:` Variables de entorno a pasar al contenedor, en este caso, se proporcionan las credenciales de Docker Hub.
- `portMappings:` Mapeo de puertos para el contenedor, en este caso, se mapea el puerto 80 del contenedor al puerto 80 del host.

5.2.3. Integración del repositorio de almacenamiento de información

La información que fue proporcionada como base de conocimiento específico al modelo de procesamiento de lenguaje natural es ficticia y no corresponde a datos verídicos para un producto financiero en particular (ver Figura 5.7). Está relacionada con el producto de crédito de libre destino. Esta información será cargada en formato tabular de CSV y contiene los siguientes detalles:

- Nombre del producto financiero.
- Tasa de interés vigente.
- Descripción del producto financiero.
- Montos mínimos y máximos del producto financiero.
- Restricciones del crédito.
- Plazos de desembolso.
- Medios de pago.

A	B
Nombre del Producto	Credito Libre Destino
Descripción del producto	préstamo ofrecido por
Tasa de Interes efectiva anual	1,50%
Edad Minima Solicitud (años)	18
Edad Maxima Solicitud (años)	60
Monto Minimo Solicitud	500000
Monto Maximo Solicitud	10000000
Las Personas Publicamente Expuestas (PEP) pueden solicitar est	NO
Plazo de desembolso	2 días habiles

Figura 5.7: Información tabular agregada al modelo NLP como base de conocimiento.

La información mencionada anteriormente, se cargó en un bucket del servicio de Amazon S3 en la nube de AWS. Este bucket se encuentra privado, por lo que no es accesible por ningún servicio. Para brindar acceso al bucket de Amazon S3, se creó un rol en el servicio de Amazon IAM siguiendo las buenas prácticas de la nube de AWS de establecer roles con el mínimo privilegio (ver Figura 5.8).

```
# IAM Policy for S3 access
resource "aws_iam_policy" "s3_access_policy" {
  name = "s3-access-policy"
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:GetObject",
          "s3:ListBucket"
        ]
        Resource = [
          aws_s3_bucket.documents_bucket.arn,
          "${aws_s3_bucket.documents_bucket.arn}/*"
        ]
      }
    ]
  })
}
```

Figura 5.8: Política de acceso al bucket de Amazon S3.

En la anterior política sé configuraron los permisos `s3:GetObject` y `s3:ListBucket` sobre el recurso específico que es el bucket de Amazon S3 y sobre todos los archivos contenidos en este. Esto permite a los servicios de la aplicación ver y recuperar los objetos en el bucket de S3, pero no

concede permisos para modificar o eliminarlos.

El proceso de cargue del documento al bucket de Amazon S3 se realizó por medio del recurso de Terraform llamado `aws_s3_object` ver Figura 5.9, el cual permite tomar un archivo de una ruta en específico y cargarlo dentro del bucket de Amazon S3 posterior a ser creado.

```
resource "aws_s3_object" "credit_information_csv" {
  bucket = aws_s3_bucket.documents_bucket.id
  key    = "condiciones-solicitud-credito-libredestino"
  source = "files/condiciones-solicitud-credito-libredestino.csv"

  etag = filemd5("files/condiciones-solicitud-credito-libredestino.csv")
}
```

Figura 5.9: Recurso de Terraform que carga el archivo con la información relacionada con el crédito de libre destino en el bucket de Amazon S3.

Para habilitar la integración con la aplicación, se asoció el rol del bucket al contenedor de aplicación despegado en la sección anterior en el servicio de Amazon ECS por medio del recurso de Terraform `aws_iam_role_policy_attachment` como se observa en la Figura 5.10.

```
# Attach S3 IAM Policy to ECS Task IAM Role
resource "aws_iam_role_policy_attachment" "s3_access_attachment" {
  role       = aws_iam_role.ecs_task_execution_role.name
  policy_arn = aws_iam_policy.s3_access_policy.arn
}
```

Figura 5.10: Role de Amazon S3 asociado al contenedor desplegado en el servicio Amazon ECS con Terraform.

El paso anterior garantiza que el contenedor con la aplicación tenga acceso a los recursos dentro del bucket. A continuación se configuró en la aplicación Flowise IA un nuevo nodo con las credenciales y parámetros para la integración con el servicio de Amazon S3.

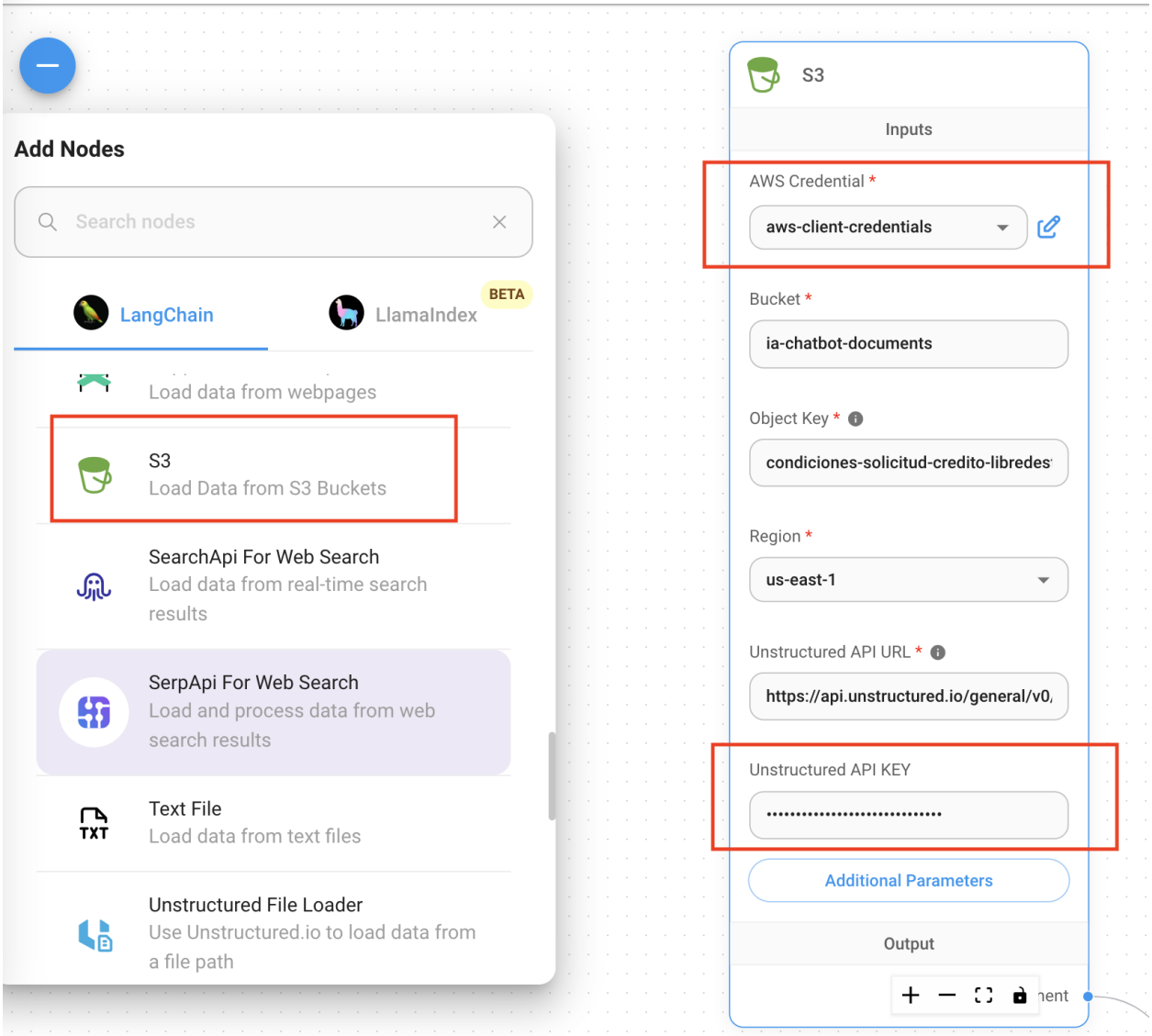


Figura 5.11: Nodo de Flowise IA que integra el documento que reposa en el bucket de Amazon S3.

Es importante aclarar que Amazon S3 almacena datos sin estructura, por lo que el nodo de Flowise IA necesita de un complemento para que el archivo recuperado sea un archivo estructurado, para esto se apoya de Unestructured.io (https://unstructured.io/) para generar dicha estructura, para ello se creó una cuenta en la aplicación web de unestructured.io y se creó una API KEY las cuales se asociaron al nodo de S3 en el flujo de Flowise IA como se muestra en la Figura 5.11.

5.2.4. Integración de la base de datos vectorial

Como se documentó en las decisiones de arquitectura de la iteración número tres de la metodología de diseño de arquitectura basado en atributos de calidad (ADD) para el atributo de compatibilidad (ver 4.5.3.1), se seleccionó la base de datos vectorial de Pinecone como base de conocimiento para la técnica RAG utilizada para interactuar con el modelo de procesamiento de lenguaje natural.

Como primera medida, se creó una cuenta en el sitio web de Pinecone https://www.pinecone.io/, la cuenta fue creada con la licencia Starter la cual aprovisiona una base de datos Serverless de manera gratuita con ciertas restricciones (ver Figura 5.12) que no afectan el desarrollo del proyecto.

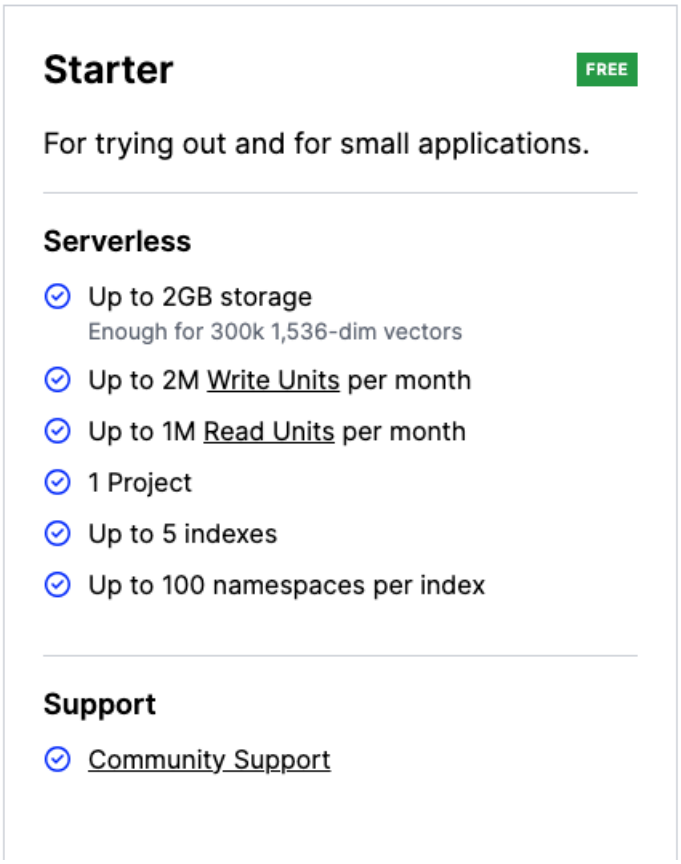


Figura 5.12: Características del plan básico y gratuito de Pinecone.

Pinecone en su versión Starter nos ofrece la capacidad de crear hasta cinco índices, estos pueden ser utilizados para organizar la base de conocimiento por temáticas separadas por cada uno de los índices, para este proyecto se creó un índice único.

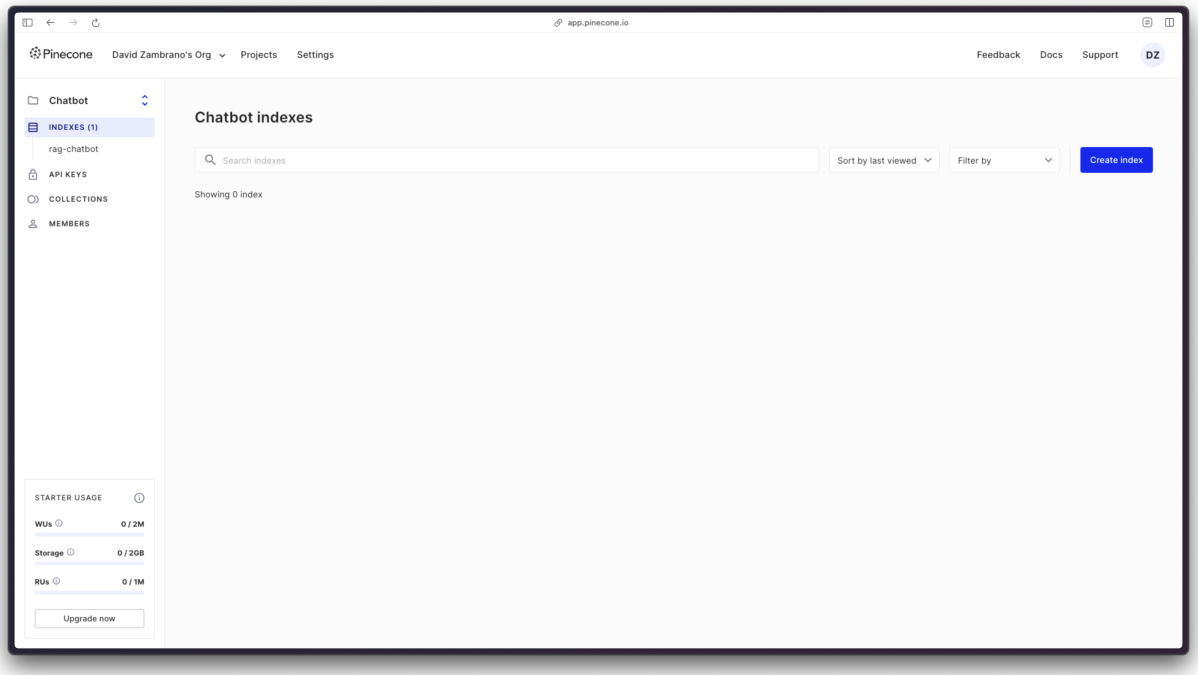


Figura 5.13: Cuenta creada en Pinecone.

Para crear el índice únicamente se especificó el nombre ("rag-chatbot"), ya que en el plan Starter las configuraciones las establece Pinecone por defecto, cabe resaltar que las bases de datos son Serverless y Pinecone decide en que proveedor de nube desplegar la base de datos y los índices, para este proyecto, el momento de la creación el índice el proveedor de nube habilitado fue el de Google Cloud Platform en la Región de Lowa ver Figura 5.14.

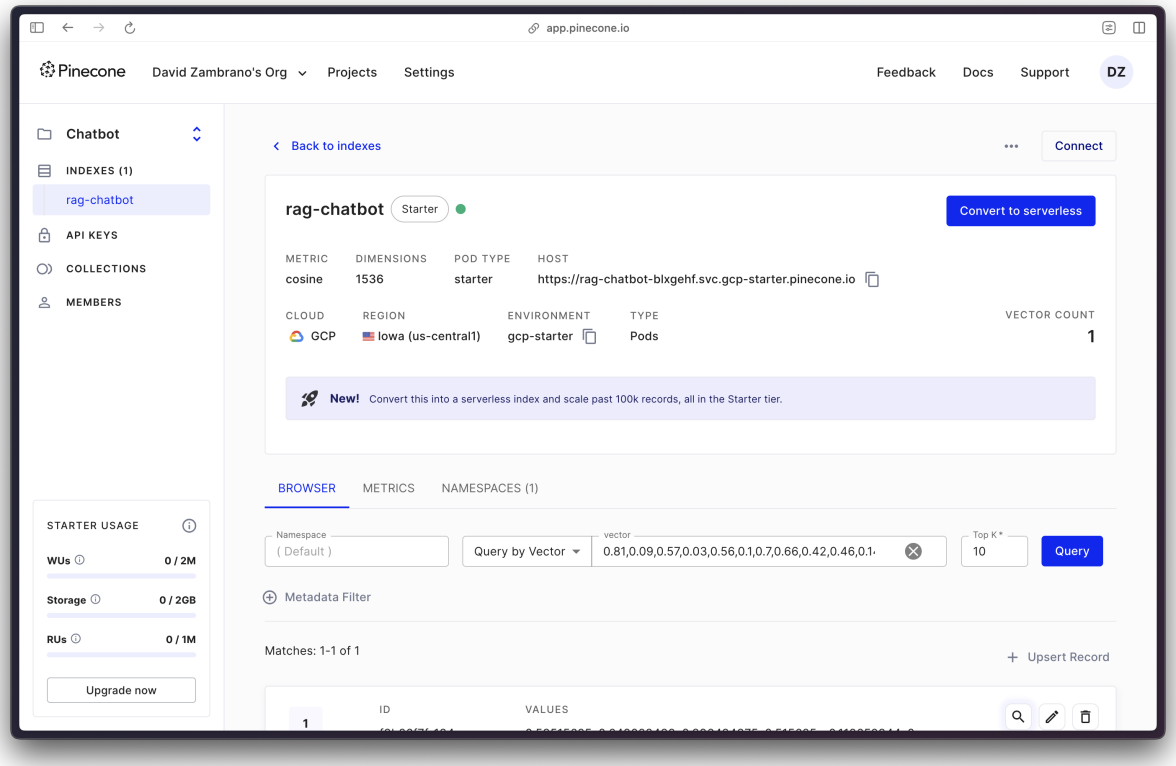


Figura 5.14: Índice creado en Pinecone en el proveedor de nube Google Cloud Platform en la Región de Lowa definido por defecto por Pinecone en el plan Starter.

Para integrar la base de datos de Pinecone a la aplicación con Flowise IA se requiere de una API KEY, la cual se creó en la cuenta de Pinecone para habilitar la conexión ver Figura 5.15.

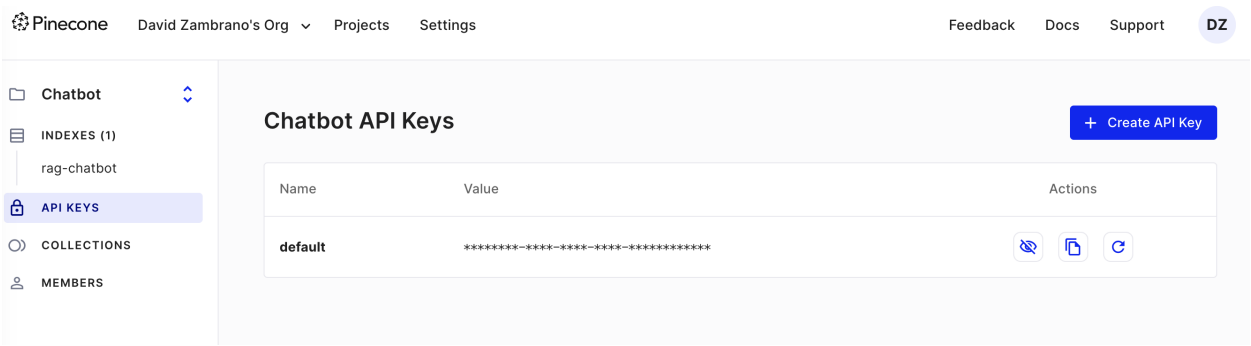


Figura 5.15: API KEY de Pinecone para integrar la aplicación.

Para este punto se integró la base de datos al flujo orquestado por Flowise IA, para ello se crearon las credenciales para conectarse con Pinecone en Flowise IA (ver Figura 5.16), posterior a

ello, se agregó un nuevo nodo al flujo en el cual se asoció las credenciales y se configuró el nombre del índice creado en Pinecone que para este proyecto es "rag-chatbot" y se configuró la salida como Pinecone Retriever para habilitar la recuperación de los vectores más cercanos a la pregunta generada por el usuario (ver Figura 5.17). Este nodo configura por defecto el TOP K de vectores en cuatro (4), este valor es personalizable, se recomienda que este valor no sea muy alto para que el contexto generado a partir de los vectores sea más preciso.

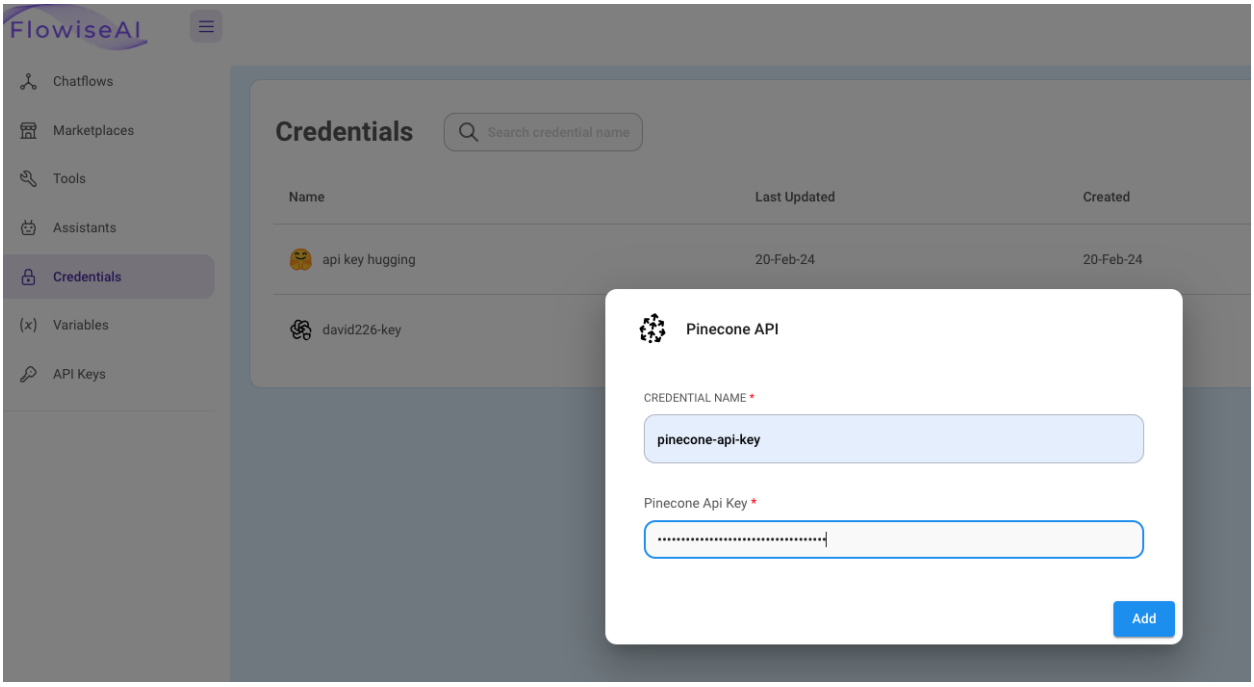


Figura 5.16: Creación de credenciales de Pinecone en Flowise IA.

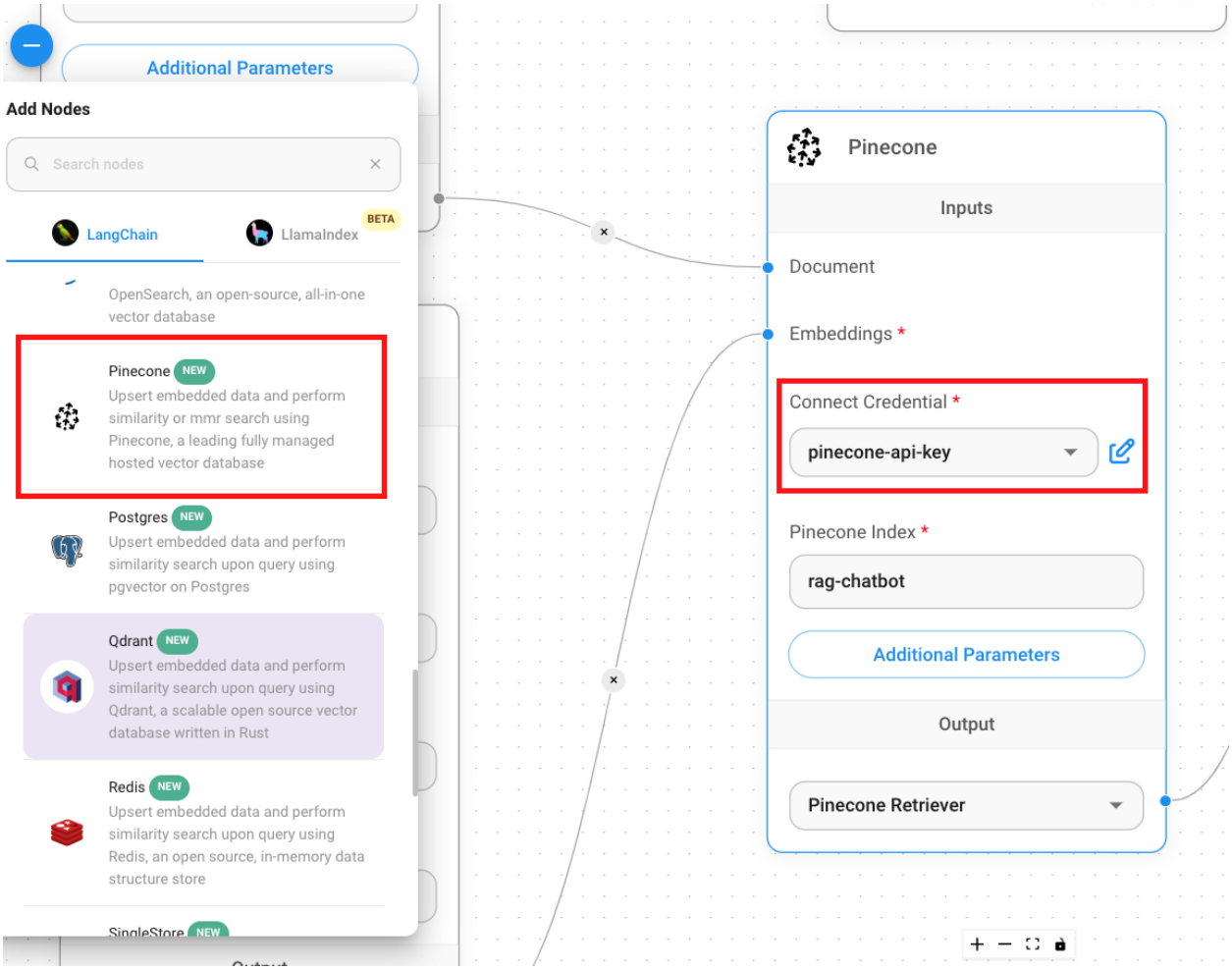


Figura 5.17: Nodo de Flowise IA que integra la base de datos vectorial de Pinecone.

5.2.5. Integrando el modelo de embeddings y de procesamiento de lenguaje natural de Amazon Bedrock

Para el proyecto, se integran dos modelos: el primero es el modelo encargado de convertir, a través de la técnica embedding function, la información obtenida desde Amazon S3 en vectores y cargarlo a la base de datos vectorial; y el segundo es el modelo encargado de procesar el lenguaje natural. Ambos modelos se encuentran desplegados en el servicio de Amazon Bedrock.

El modelo de embedding seleccionado fue `amazon.titan-embed-text-v1`, el cual tiene soporte de múltiples idiomas, incluido el español. Para integrar el modelo en la herramienta Flowise IA, lo primero fue habilitar el modelo en la cuenta de Amazon. Tras la autenticación en la cuenta de Amazon donde se desplegaron los recursos de infraestructura, se procedió a navegar al servicio Amazon Bedrock dentro de la opción `Model Access` y solicitar acceso para habilitar el modelo,

como se muestra en la Figura 5.18. Este mismo procedimiento se realizó para habilitar el modelo de procesamiento de lenguaje natural llamado `meta.llama2-13b-chat-v1`.

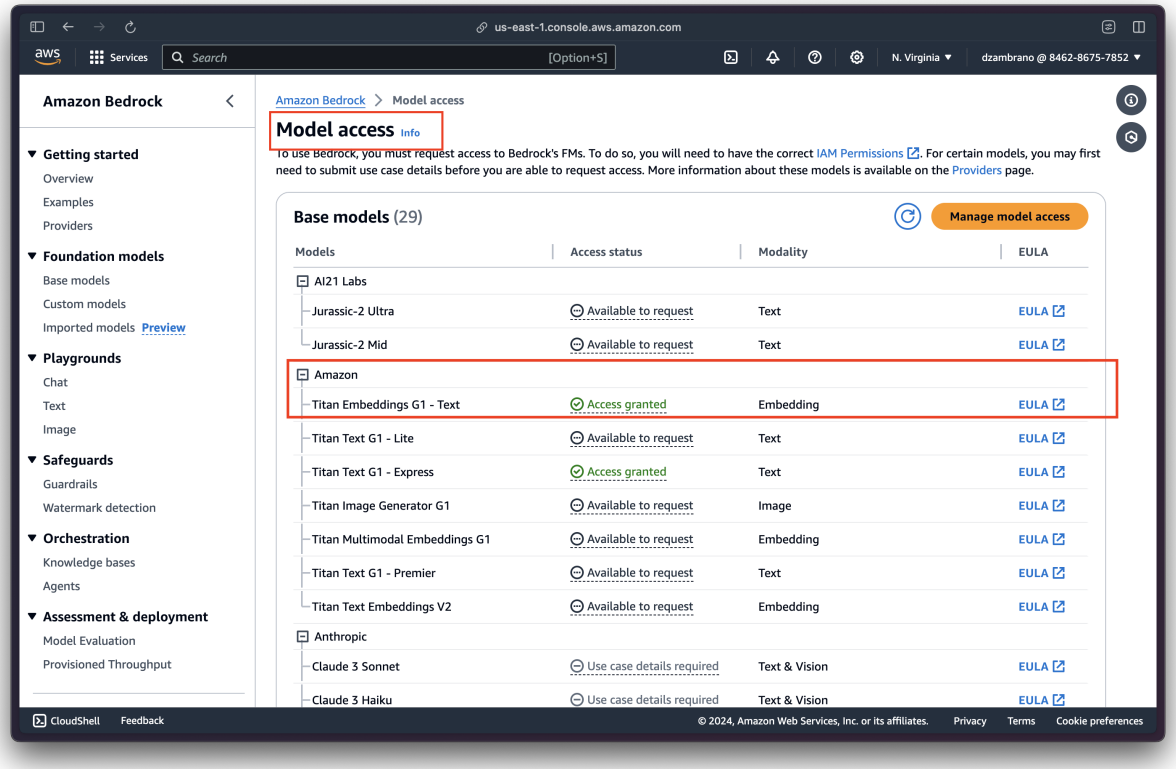


Figura 5.18: Modelos habilitados en el servicio de Amazon Bedrock.

Una vez el modelo fue habilitado y se encontró disponible, se agregó un nuevo nodo en el flujo orquestado por la herramienta Flowise IA llamado `Amazon Bedrock Embeddings` y se configuró las credenciales, la región en la que fue desplegado el modelo y el nombre del modelo como se muestra en la Figura 5.19, adicional a ello, se configuró la conexión este nodo con el nodo de la base de datos vectorial de Pinecone el cual se configuró en la etapa anterior.

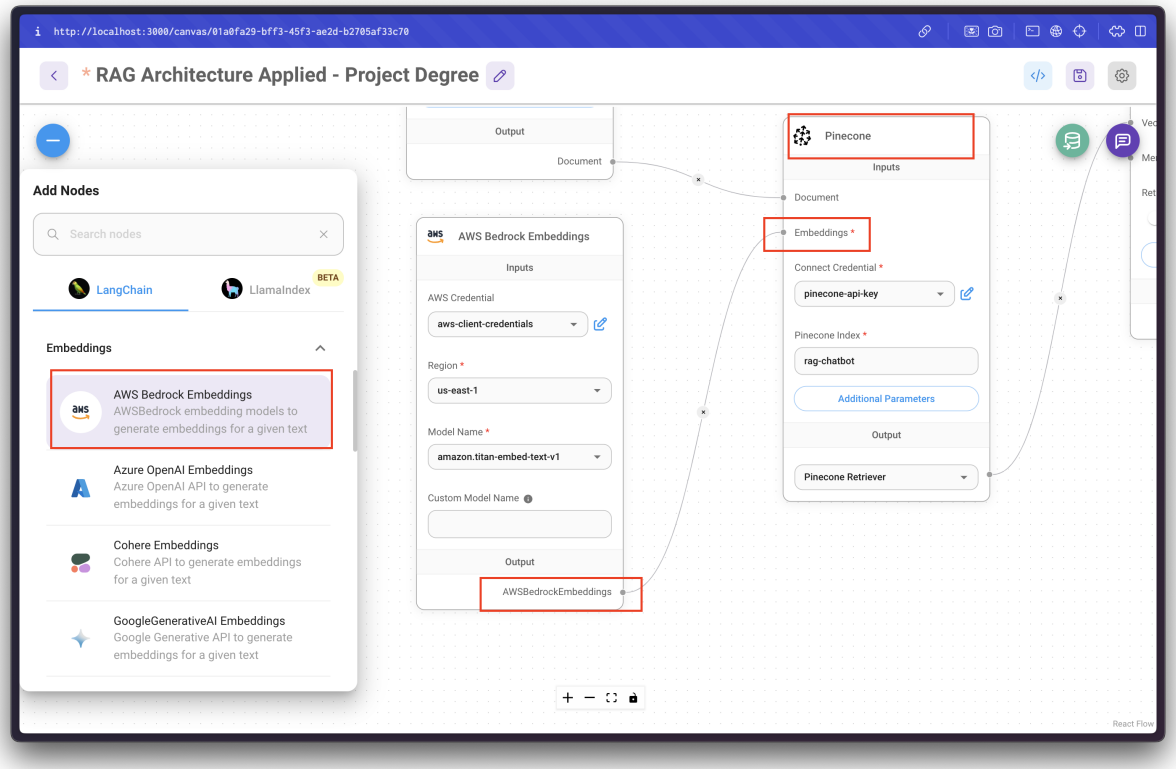


Figura 5.19: Modelo de embeddings de Amazon Bedrock integrado con la base de datos de Pinecone.

Para integrar el modelo de procesamiento de lenguaje natural se tienen múltiples opciones en la herramienta de Flowise IA, para construir un agente conversacional utilizando la técnica RAG, se recomienda los nodos para modelos de chat, para ello se agregó un nuevo nodo al flujo llamado `AWS ChatBedrock`, sobre el cual se configuró las credenciales de la cuenta de Amazon, la región en la que se encuentra el modelo habilitado y el nombre del modelo `meta.llama2-13b-chat-v1` como se muestra en la Figura 5.20.

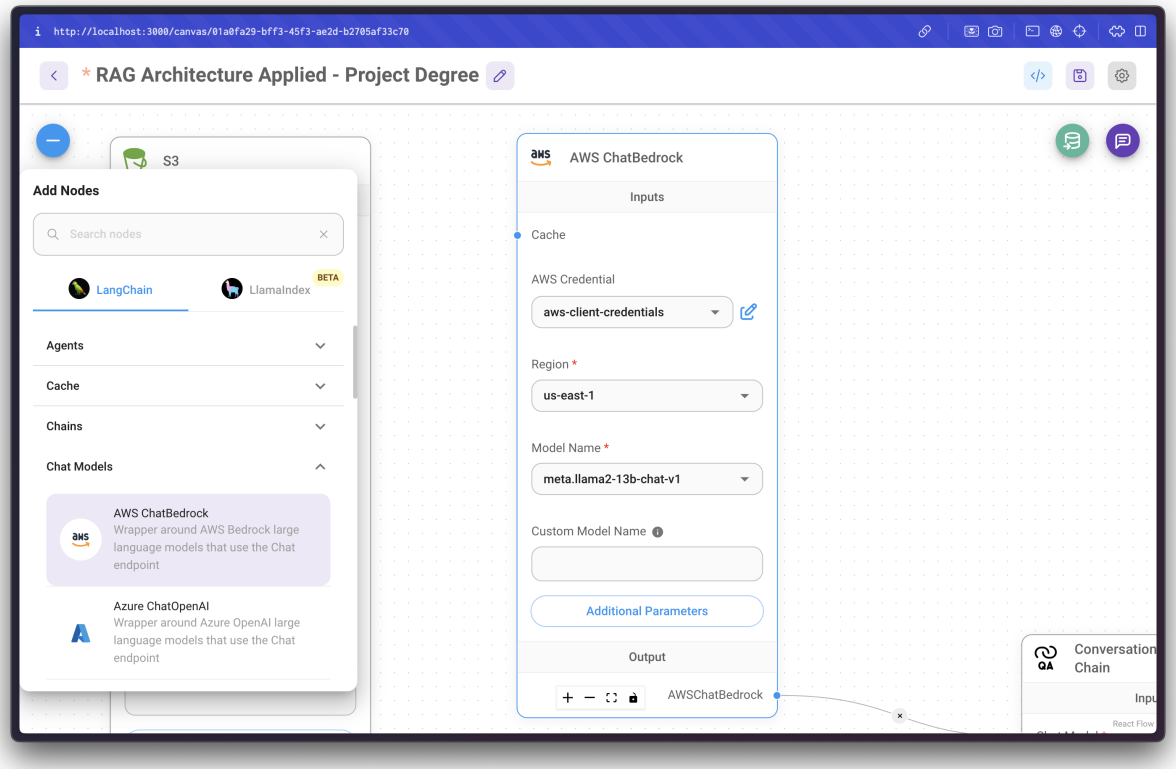


Figura 5.20: Modelo de procesamiento de lenguaje natural de Amazon Bedrock integrado en el flujo de Flowise IA.

5.2.6. Orquestación del flujo conversacional utilizando la técnica (RAG)

Retrieval-Augmented Generation (RAG) es una técnica que permite disminuir las respuestas incorrectas o alucinaciones de los modelos de procesamiento de lenguaje natural. Para orquestar el flujo conversacional, como se muestra en la Figura 5.21 se emplearon cinco nodos, tres de ellos empleados para el cargue de información a la base de documentos de Pinecone y los dos restantes utilizados para interactuar con el modelo de procesamiento de lenguaje natural.

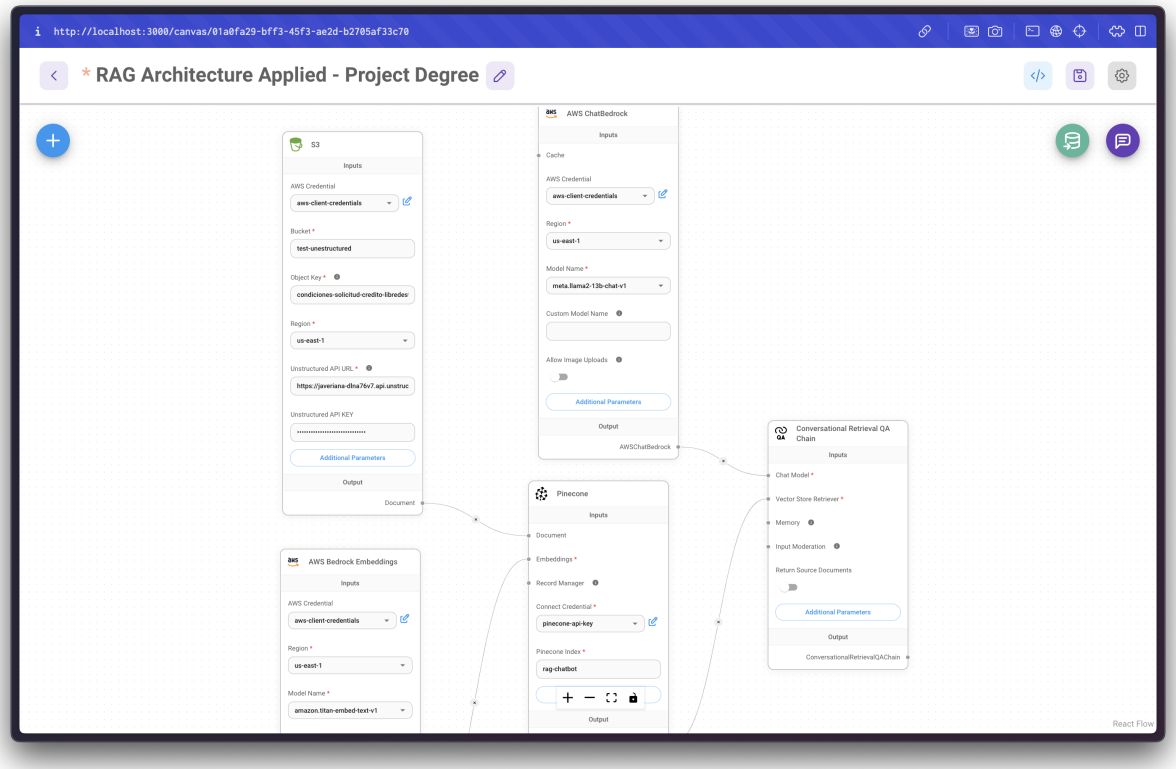


Figura 5.21: Flujo completo orquestado aplicando la técnica RAG.

Para validar el flujo orquestado, la plataforma de Flowise IA cuenta con un chat integrado el cual permite validar el funcionamiento del flujo. Para realizar la prueba del flujo, lo primero es cargar los datos en la base de datos vectorial de Pinecone, para ello haciendo uso de la opción llamada “Upsert Vector Store” a la cual se accede haciendo clic en el icono se realizó la carga como se muestra en la Figura 5.22.

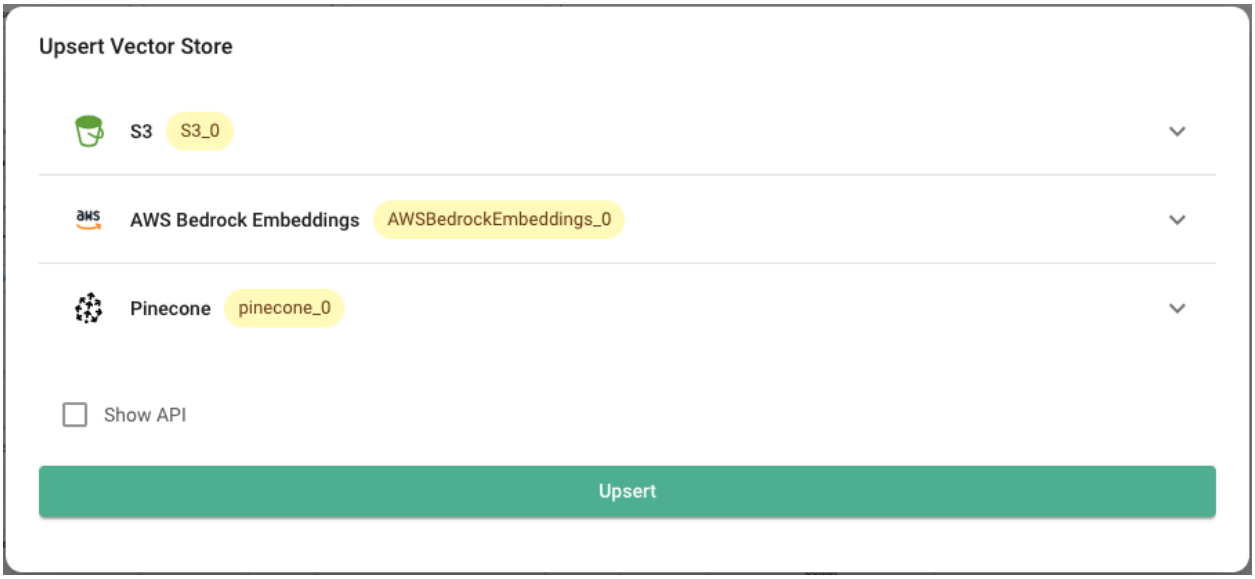


Figura 5.22: Carga de datos en la base de datos de conocimiento desde la plataforma Flowise IA.

Una vez se confirmó que los datos fueron cargados de manera exitosa, se realizó la validación iniciando una conversación con el trabajador digital inteligente haciendo uso del chat integrado que provee la plataforma de Flowise IA y al cual se accede haciendo clic en el botón llamado "Chat" el cual se identifica con el icono .

5.2.7. Integración de la aplicación de mensajería Whatsapp

En este proyecto se optó por integrar la arquitectura con la aplicación de mensajería instantánea WhatsApp, debido a su relevancia como canal de atención al cliente para las empresas actuales. WhatsApp, con más de 2 mil millones de usuarios activos, se ha convertido en una de las herramientas de comunicación más utilizadas a nivel mundial, lo que la convierte en una plataforma ideal para conectar con los clientes potenciales y existentes.

Para llevar a cabo esta integración, se emplearon dos plataformas. La primera llamada Twilio la cual ofrece un mecanismo sencillo y seguro de integración con la API de WhatsApp y agrega una capa de prueba denominada "sandbox" con la cual se puede validar la integración antes de pasar a un ambiente productivo. La segunda plataforma es Make, considerada una plataforma Low-Code/No-Code que permite automatizar diferentes flujos de trabajo.

5.2.7.1. Habilitación de WhatsApp en Twilio

Para habilitar el servicio de WhatsApp en Twilio, se creó una cuenta en la página web de Twilio (`https://www.twilio.com/en-us`) y se configuró el ambiente de pruebas ("sandbox") el cual se encuentra activo de forma predeterminada en la plataforma con un número de WhatsApp listo para ser usado. Para realizar la configuración, se navegó en el menú de la plataforma siguiendo la

ruta `Messaging >Try it Out >Send WhatsApp Message`, desde esta opción se habilitó el número desde el cual se realizaron las pruebas, para ello se puede escanear el código QR como se muestra en la Figura 5.23.

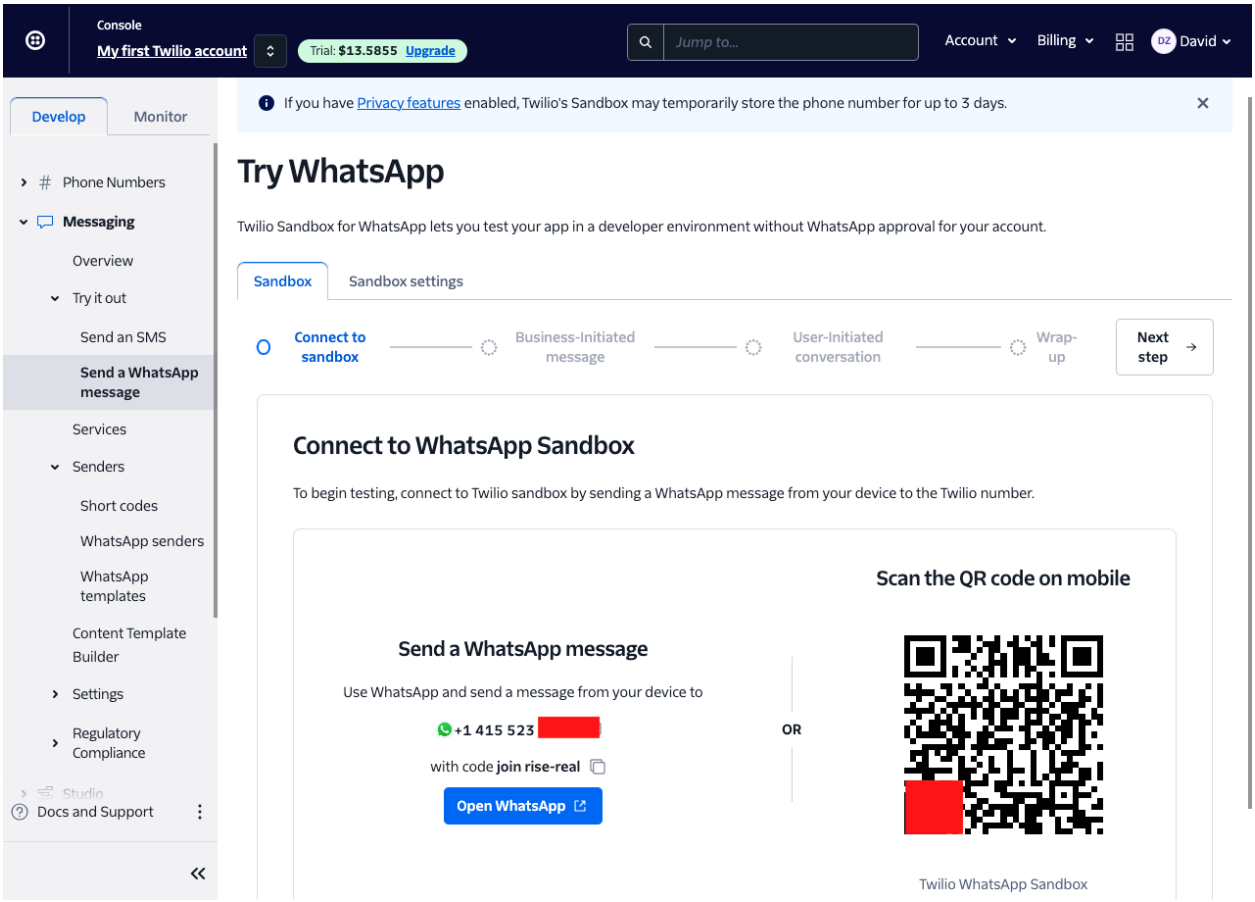


Figura 5.23: Página de habilitación de número de pruebas para WhatsApp en Twilio.

Después de habilitar el número, se generó un token de autenticación como se muestra en la Figura 5.24 para poder integrar este servicio dentro de la automatización de la conversación con la plataforma Make.

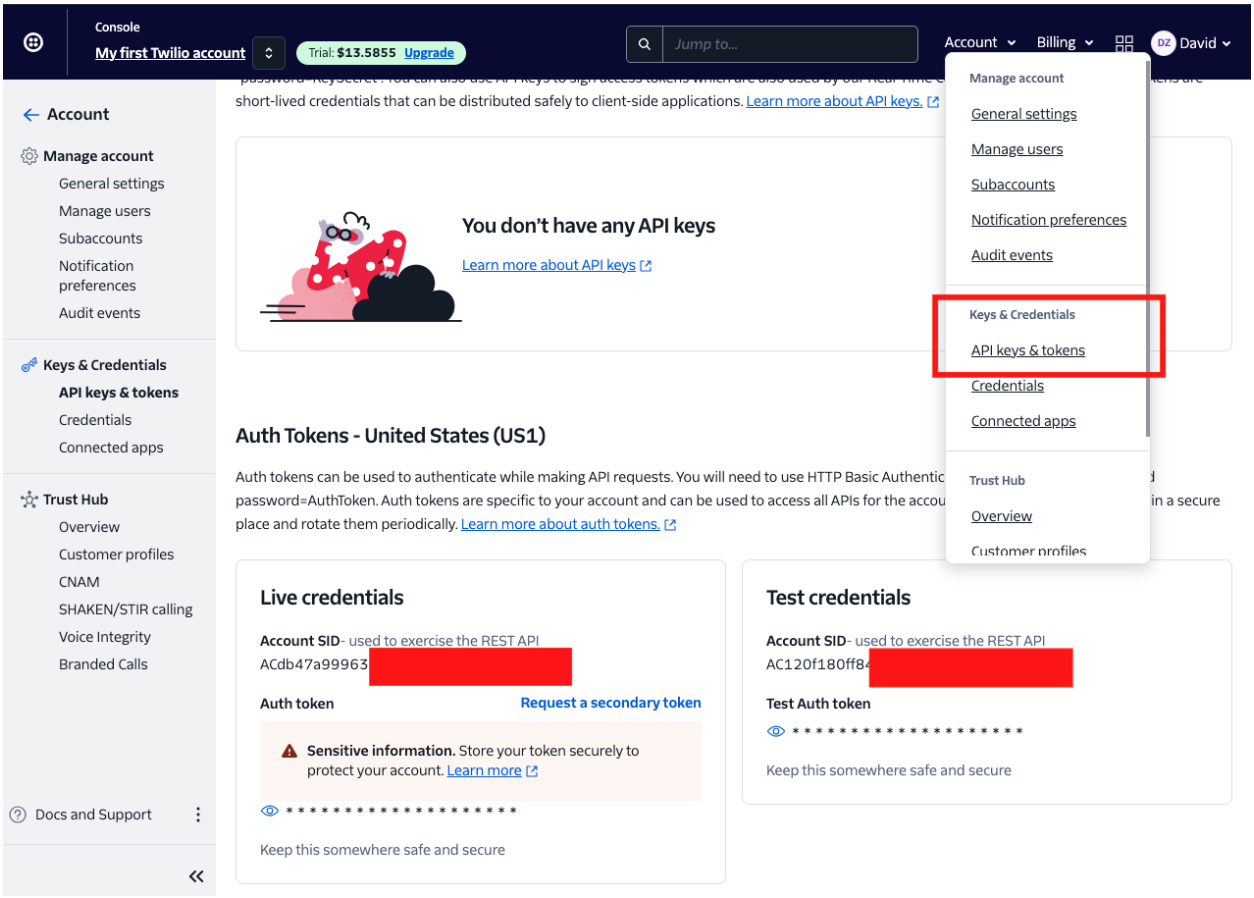


Figura 5.24: Generación del token de autenticación en Twilio.

5.2.7.2. Automatización del flujo con Make

La plataforma llamada Make ofrece múltiples nodos con los cuales se pueden automatizar diferentes flujos de trabajo. Para esta integración, se utilizaron un total de 4 nodos (ver Figura 5.25) para los cuales se especifica cada uno junto con su responsabilidad dentro de la automatización.

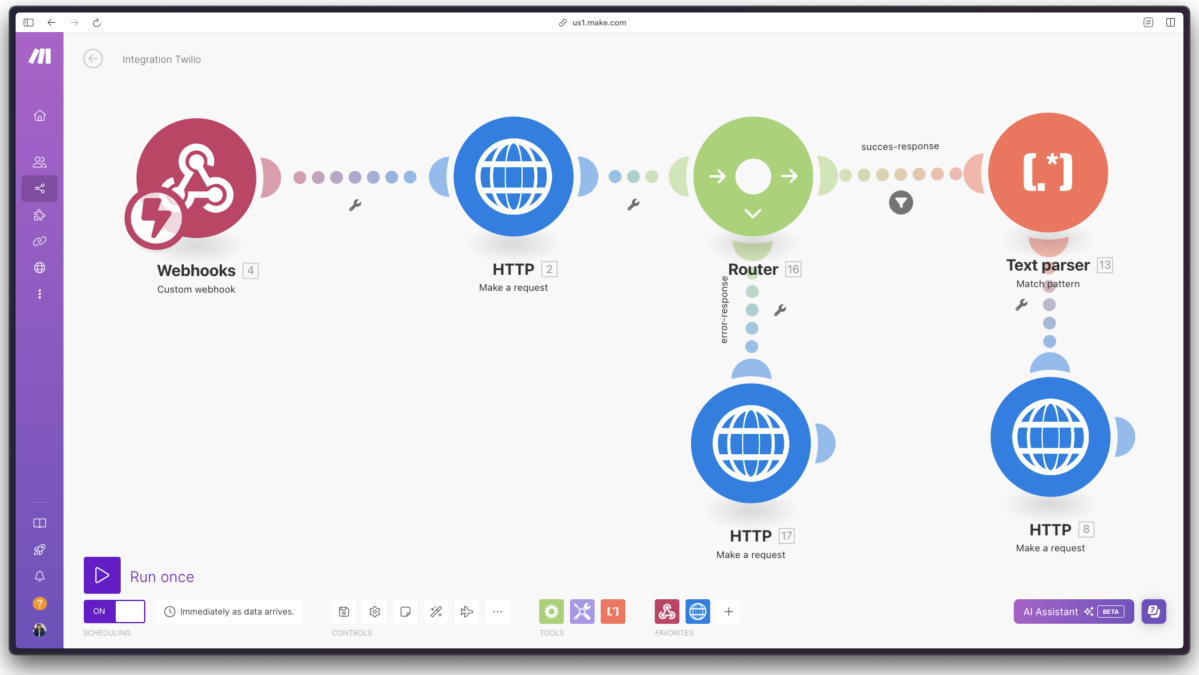


Figura 5.25: Flujo conversacional automatizado con la plataforma Make e integrado con la arquitectura en la nube de AWS.

1. **Nodo Webhook**: Este nodo expone un webhook, el cual se encuentra basado en el protocolo HTTP y permite que dos aplicaciones se comuniquen por medio de eventos. La comunicación por webhooks se conoce como comunicación inversa, ya que la aplicación cliente expone un webhook y la aplicación que actúa como servidor registra el webhook y es la encargada de notificar determinados eventos cuando estos sucedan, en un escenario no inverso, la aplicación cliente es la encargada de llamar al servidor o a la aplicación que actúa como servidor para validar si un evento sucedió. Este nodo expone un webhook el cual es configurado en el ambiente de "sandbox" de la plataforma Twilio (ver Figura 5.26), para que esta a su vez le notifique a la plataforma make por medio del webhook que existe un mensaje entrante.

2. **Nodo HTTP**: Este nodo es el encargado de integrarse con la arquitectura diseñada para el trabajador digital inteligente por medio del protocolo HTTP, la integración se realizó por medio las APIs expuestas usando la técnica de integración llamada Discovery con un balanceador de carga en la nube de AWS.

3. **Nodo Text Parser**: Nodo encargado de obtener el mensaje de respuesta del trabajador digital inteligente y de pasarlo a un nuevo nodo.

4. **Nodo HTTP**: Este cuarto nodo es diferente al segundo, ya que es el encargado de la integración con la API de respuesta de mensajes de Twilio. Con el primer nodo, se busca recibir el

mensaje para procesar y con este cuarto nodo, se busca enviar la respuesta al usuario después de ser procesada por el trabajador digital inteligente haciendo uso de inteligencia artificial generativa.

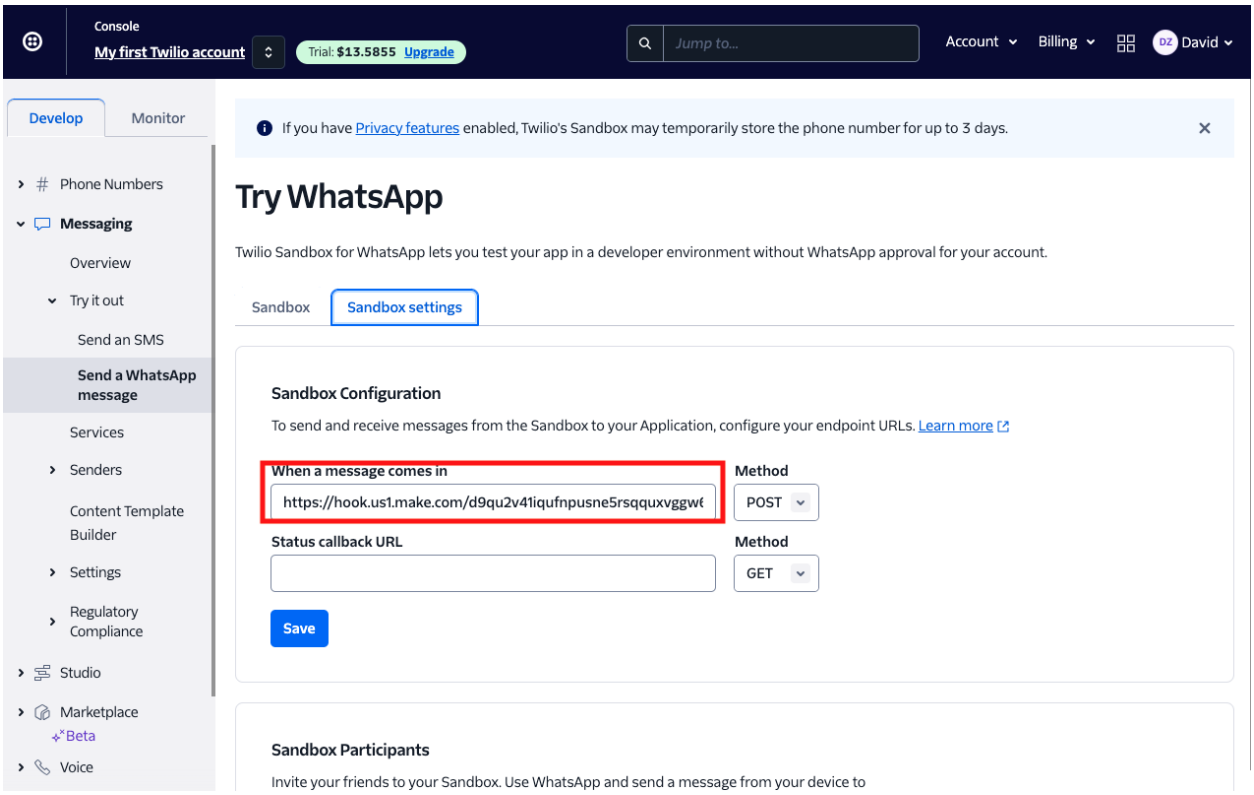


Figura 5.26: Webhook de la plataforma Make, registrado en la plataforma Twilio.

Para un mayor entendimiento del proceso de integración de los diferentes componentes y cuál es la responsabilidad de cada uno durante una conversación, se definió un diagrama que permite validar los diferentes actores y el flujo por el cual atraviesa el mensaje de un usuario (cliente o posible cliente), desde la generación hasta la respuesta como se muestra en la Figura 5.27.

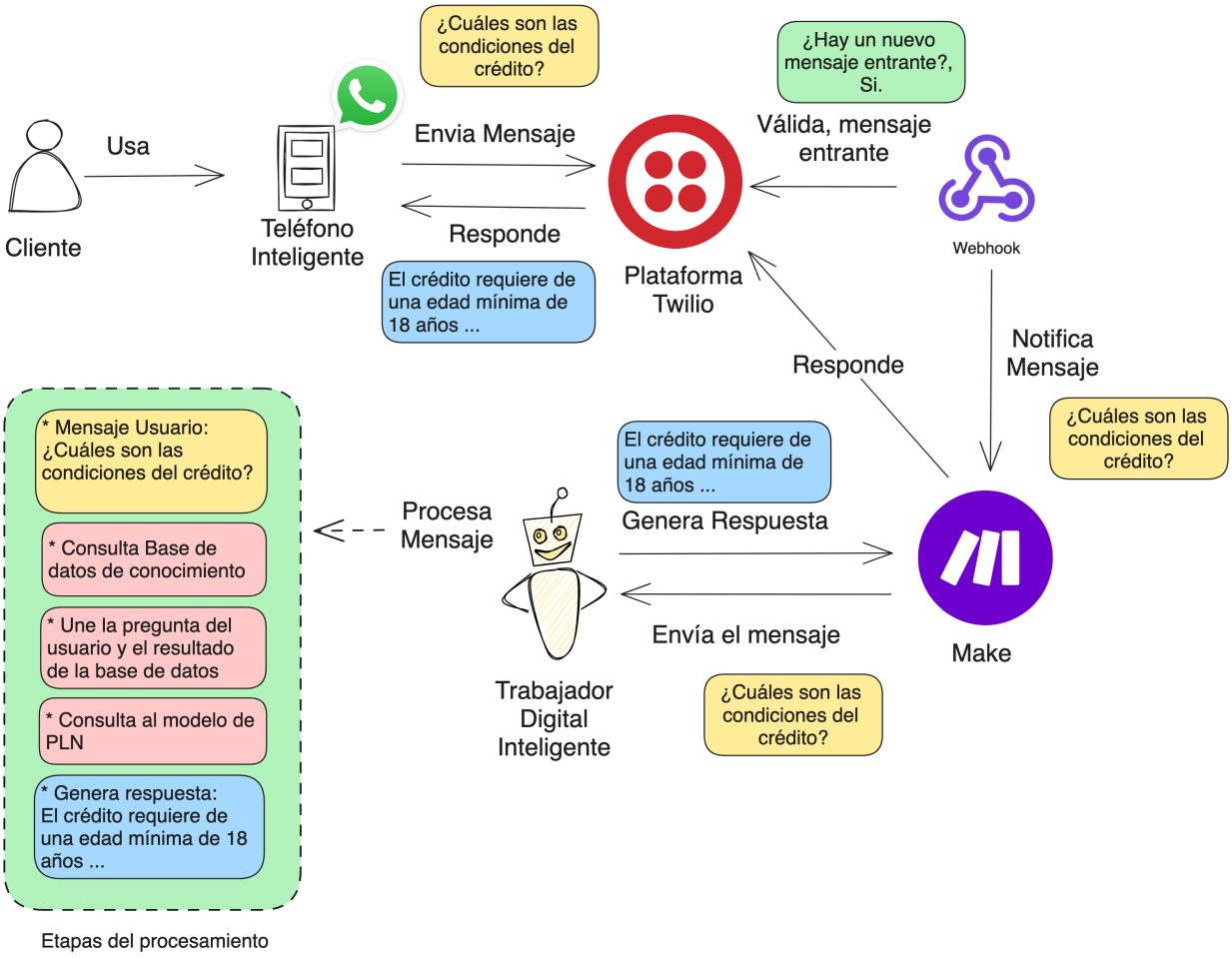


Figura 5.27: Interacción de los componentes integrados para facilitar la conversación de un cliente o posible cliente con el modelo de inteligencia artificial.

Como se ve en el diagrama anterior, un cliente o posible cliente envía un mensaje (representado por un cuadro amarillo) utilizando su teléfono inteligente y la aplicación de mensajería de WhatsApp. Este mensaje es recibido por la plataforma Twilio. A través del webhook proporcionado por la plataforma Make, Twilio notifica a Make sobre el nuevo mensaje entrante. Make utiliza su integración con la arquitectura en la nube de AWS para que el trabajador digital inteligente genere una respuesta a la pregunta por medio de un procesamiento en varias etapas. Como resultado, se genera una respuesta (representada por un cuadro azul) que primero se envía a Make, quien luego notifica la respuesta a Twilio, que la envía al usuario final.

5.2.8. Pruebas de integración

Para las pruebas de integración, se inició una conversación desde la aplicación de mensajería de WhatsApp y se realizaron preguntas de tipo saludo, interés en un producto y respuesta a preguntas generadas por el modelo a lo cual el modelo genero una respuesta acorde a la conversación para cada una de ellas como se muestra en la Figura 5.28.

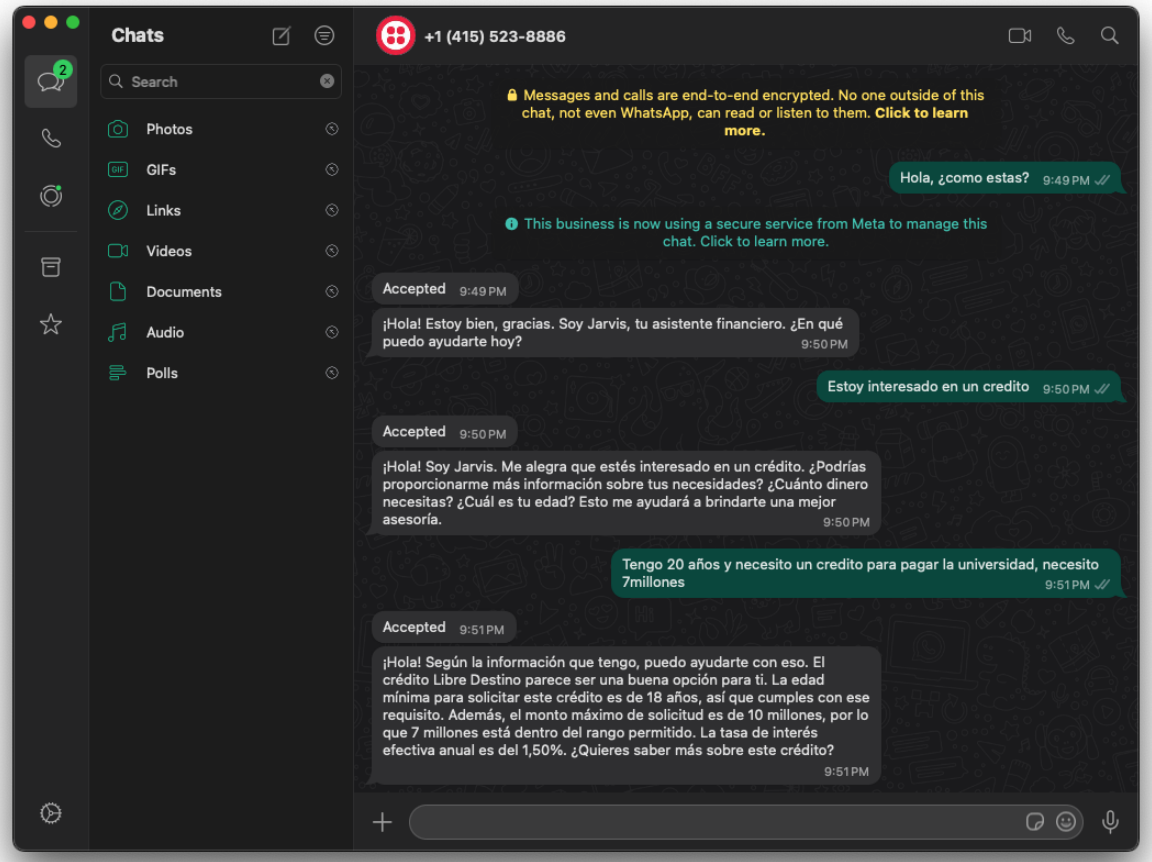


Figura 5.28: Resultado prueba de integración. Conversación desde la aplicación de mensajería WhatsApp.

En la figura anterior, el mensaje de respuesta con la palabra “Accepted”, es meramente informativo y corresponde a la respuesta automática de pruebas asignada para validar la aceptación del mensaje entrante en la plataforma Twilio.

Otro punto importante evaluado durante esta etapa de pruebas fue el del tiempo de respuesta promedio del trabajador digital inteligente. Para este cálculo se basó en la cantidad de peticiones realizadas durante la prueba y en los tiempos de respuesta, los cuales se registran en la plataforma

Make como se muestra en la Figura 5.29. Como resultado de esta prueba se puede determinar que el tiempo promedio de respuesta es de 4.5 segundos y que el trabajador digital inteligente logró atender el 100 % de las conversaciones sin presentar ningún error.

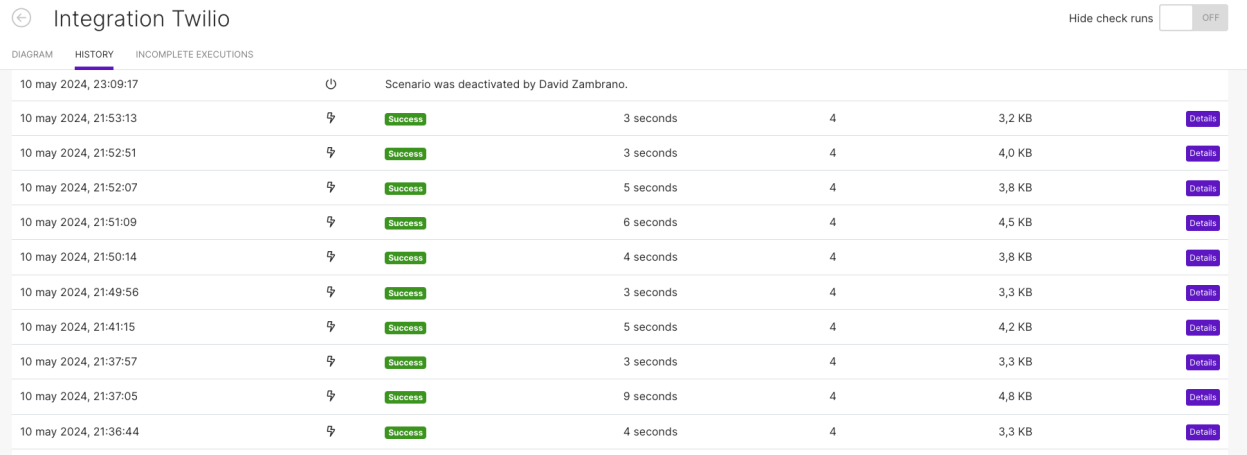


Figura 5.29: Conversaciones atendidas por el trabajador digital inteligente y reportadas en la plataforma Make que describen detalles como el estado y el tiempo de respuesta.

Capítulo 6

Evaluación de la Arquitectura

6.1. Introducción a la metodología de evaluación

Para evaluar la arquitectura de referencia propuesta en este proyecto que habilita una plataforma tecnológica para la venta consultiva de un producto digital con inteligencia artificial se utilizó la metodología de evaluación de arquitecturas de software LAE (Lightweight Architecture Evaluation/Evaluación de Arquitectura Ligera) planteada por Len Bass, Paul Clements y Rick Kazman en el libro "Software Architecture in Practice 4th Edition" (Bass et al., 2021) que busca mitigar riesgos durante las primeras etapas del ciclo de vida del desarrollo de software y aporta en gran manera al desarrollo basado en los principios ágiles.

La metodología LAE, al ser una metodología ligera, plantea revisiones periódicas de pares. Estas revisiones se realizaron para validar que la arquitectura satisface el propósito para el cual fue creada. La retroalimentación obtenida en estas revisiones permitió refinar y optimizar la arquitectura propuesta en este proyecto.

6.2. Aplicación de la metodología LAE

6.2.1. Paso 1: Presentación de la metodología

La presentación de la metodología se realizó en una de las sesiones de seguimiento con el director del proyecto, en la cual se habló sobre el proceso de evaluación utilizando esta metodología, en esta sesión se estableció un objetivo, un alcance y el beneficio que trae el aplicar esta metodología para la evaluación de la arquitectura de software propuesta. A continuación se detalla el objetivo, alcance y beneficio definido.

- **Objetivo**: Evaluar la arquitectura de software de referencia para la plataforma tecnológica que habilita la venta consultiva de un producto digital con inteligencia artificial, con el fin de garantizar que cumple con el propósito de diseño para el cual fue creada. Esto incluye la identificación de posibles riesgos, problemas de diseño y oportunidades de mejora, asegurando así que los atributos de calidad, tales como seguridad, disponibilidad e interoperabilidad, sean satisfechos.

- **Alcance**: La evaluación se centrará en los aspectos arquitectónicos del sistema, incluyendo la seguridad, la disponibilidad e interoperabilidad.

- **Beneficio**: La evaluación permitirá identificar y abordar problemas potenciales en las primeras etapas del desarrollo del software, lo que puede reducir costos y mejorar la calidad del producto final.

6.2.2. Paso 2: Revisión de los Objetivos de la arquitectura de referencia

Para este punto se realizó la revisión de los objetivos de este proyecto, ya que se encuentran alineados con la propuesta de una arquitectura de referencia en la nube de AWS para implementar una plataforma tecnológica que habilite la venta consultiva de un producto digital con inteligencia artificial. Dichos objetivos se encuentran descritos en el Capítulo 1 en la sección Objetivos.

En el proceso de diseño de la arquitectura de referencia, se establecieron objetivos que denotan los atributos de calidad de manera explícita e implícita. Estos atributos fueron la base fundamental para la toma de decisiones en el diseño. Los tres atributos clave considerados fueron

- **Seguridad**: Se buscó garantizar la protección de los datos y la confidencialidad de la información. Esto implicó la implementación de mecanismos de autenticación, autorización y cifrado.
- **Disponibilidad**: La arquitectura se diseñó para asegurar que el sistema estuviera disponible al menos un 99 %. Se consideraron estrategias de redundancia, balanceo de carga y recuperación ante fallos.
- **Interoperabilidad**: La capacidad de interactuar con otros sistemas y componentes fue un objetivo primordial. Se definieron interfaces estándar y se evaluó la adaptabilidad a diferentes entornos.

6.2.3. Paso 3: Revisión de la arquitectura

El objetivo principal de este proyecto es la construcción de una arquitectura de referencia en la nube de AWS para habilitar la venta consultiva de un producto digital con inteligencia artificial. Dicha arquitectura fue refinada haciendo uso de la metodología de diseño guiado por atributos de calidad (ADD), y evaluada utilizando la técnica de revisión de pares, como se muestra en el Capítulo 4.

En este paso de la metodología de evaluación LEA, se validaron los incrementos de producto de la arquitectura en relación con el cubrimiento de las necesidades planteadas en los atributos de calidad. Las vistas de arquitectura sirvieron como instrumento de medición, proporcionando una referencia para el desarrollo de los puntos siguientes de la metodología de evaluación.

6.2.4. Paso 4: Revisión de los enfoques arquitectónicos

Como resultado de la etapa de revisión de conceptos de diseño durante el proceso de diseño de la arquitectura con la metodología de diseño basada en atributos de calidad (ADD), se identificaron múltiples enfoques arquitectónicos, los cuales se encuentran alineados a los objetivos planteados

para este proyecto y para el diseño de la arquitectura de referencia como también con los escenarios de calidad planteados para cada uno de los atributos de calidad. A continuación se listan los enfoques arquitectónicos evaluados y seleccionados para el proceso de diseño de la arquitectura de referencia en la nube de servicios de AWS.

Atributo de calidad	Tipo	Enfoques considerados	Enfoque(s) Seleccionado(s)
Seguridad	Tácticas	Las tácticas evaluadas se encuentran detalladas en la Sección 4.5.1.1, A continuación se listan algunas de las tácticas evaluadas: ▪ Detectar la denegación de servicio ▪ Verificar la integridad del mensaje ▪ Identificar los actores ▪ Limitar el acceso ▪ Encriptar la Información ▪ Restringir acceso ▪ Auditoría	Los enfoques seleccionados son: ▪ Autenticar a los actores ▪ Autorizar a los actores ▪ Validar las entradas ▪ Auditoría

Atributo de calidad	Tipo	Enfoques considerados	Enfoque(s) Seleccionado(s)
	Patrones	Se evaluaron diferentes enfoques, desde los enfoques tradicionales hasta los enfoques aplicados en nube (ver Sección 4.5.1.1). A continuación se listan los enfoques evaluados: ▪ Único punto de acceso ▪ Roles/RBAC ▪ Sesiones ▪ Redes Virtuales ▪ Multi-Factor de autenticación ▪ Retención de la información ▪ Auditoría segura	Enfoques seleccionados: ▪ Federación ▪ Clave de acceso ▪ Administrador de identidades y accesos ▪ Redes virtuales ▪ Cortafuegos de aplicaciones ▪ Auditoría Segura

Atributo de calidad	**Tipo**	**Enfoques considerados**	**Enfoque(s) Seleccionado(s)**
Fiabilidad (Disponibilidad)	Tácticas	Las tácticas evaluadas durante la iteración de ADD relacionada con el atributo de calidad de disponibilidad se encuentran en la Sección 4.5.2.1, a continuación se listan algunas de las tácticas evaluadas: ▪ Monitoreo ▪ Detección de excepciones ▪ Repuesto redundante ▪ Manejo de errores ▪ Prevención de excepciones ▪ Reintentos	Los enfoques seleccionados que satisfacen las entradas de la iteración en mención son: ▪ Monitoreo ▪ Pulsos ▪ Detección de excepciones ▪ Repuesto redundante ▪ Manejo de excepciones
	Patrones	A continuación se listan algunos de los patrones evaluados: ▪ Patrón de repuestos redundantes ▪ Patrón de redundancia de modularidad triple ▪ Patrón de IP Flotante ▪ Patrón de Balanceo de Carga	Los enfoques seleccionados son: ▪ Balanceo de Carga ▪ Auto-Escalamiento ▪ Orquestador

Atributo de calidad	Tipo	Enfoques considerados	Enfoque(s) Seleccionado(s)
	Estilos Arquitectónicos	Múltiples enfoques relacionados con los estilos arquitectónicos fueron evaluados, y clasificados por su nivel de impacto y beneficio, a continuación se listan algunos estilos arquitectónicos evaluados: ▪ Arquitectura de capas ▪ Arquitectura de tuberías ▪ Arquitectura Basada ▪ Arquitectura de microservicios	Los enfoques seleccionados son: ▪ Arquitectura Basada en Servicios
Compatibilidad (Interoperabilidad)	Tácticas	El total de tácticas evaluadas se encuentran en la Sección 4.5.3.1. A continuación se listan algunas tácticas evaluadas. ▪ Encapsular ▪ Utilizar un intermediario ▪ Adherirse a los estándares ▪ Administrar los recursos ▪ Orquestación	Tácticas seleccionadas: ▪ Descubrimiento ▪ Comportamiento Personalizable ▪ Orquestación ▪ Adherencia a estándares

Atributo de calidad	Tipo	Enfoques considerados	Enfoque(s) Seleccionado(s)
	Patrones	Se analizaron los patrones de integración empresariales junto con los patrones de integración en la nube, dicho análisis se encuentra en la Sección 4.5.3.1 del documento. A continuación se listan algunos de los patrones analizados: ▪ Tuberías y Filtros ▪ Enrutamiento de Mensajes ▪ Publicador/Subscriptor ▪ Enrutamiento de APIs ▪ Patrón Saga ▪ Cierre de circuito ▪ Contenido enriquecido ▪ Traductor de mensajes	Los patrones seleccionados son: ▪ Enrutamiento de APIs ▪ Contenido enriquecido ▪ Solicitud-Respuesta ▪ Enrutamiento de mensajes ▪ Traductor de mensajes
	Técnicas de integración con los modelos de IA	Para integrarse con un modelo de Inteligencia Artificial (IA) existe una serie de técnicas, la cual puede ser aplicada dependiendo del caso de uso ver Sección 4.5.3.1. Para este proyecto se analizó las siguientes técnicas: ▪ ReAct ▪ RAG ▪ MRKL	La táctica seleccionada fue: ▪ RAG

Atributo de calidad	Tipo	Enfoques considerados	Enfoque(s) Seleccionado(s)
	Base de datos	La arquitectura contemplo la opción de integrar una base de datos la cual sirve como base de conocimiento para interactuar con el modelo de LLM, esta base de datos tiene un requerimiento especial al tratarse de una arquitectura que integra un modelo de procesamiento de lenguaje natural, se analizaron ocho bases de datos y se clasificaron por rendimiento y escalabilidad como se observa en la Sección 4.5.3.1. Las bases de datos analizadas fueron: ▪ Faiss ▪ Chroma ▪ Vald ▪ Pinecone ▪ Vespa ▪ Milvus ▪ Qdrant ▪ Weaviate	La base de datos seleccionada fue: ▪ Pinecone

Atributo de calidad	Tipo	Enfoques considerados	Enfoque(s) Seleccionado(s)
	Método de desarrollo de software	Uno de los objetivos de este proyecto es la implementación de un prototipo funcional para evaluar la arquitectura diseñada. En este marco, la selección de un método de desarrollo acorde con el proyecto fue fundamental para definir la capacidad de integración de los artefactos, para este proyecto se analizaron diferentes métodos de desarrollo. Los métodos de desarrollo de software analizados fueron: ▪ Método de desarrollo en Cascada ▪ Método Incremental ▪ Método Low-Code/No-Code	El método de desarrollo de software seleccionado para dar cumplimiento al objetivo específico que referencia la implementación de un prototipo funcional para evaluar la arquitectura de referencia propuesta fue: ▪ Low-code/No-code

Tabla 6.1: Enfoques arquitectónicos evaluados y seleccionados en el proceso de diseño de la arquitectura de referencia siguiendo la metodología de diseño basado en atributos de calidad (ADD).

6.2.5. Paso 5: Revisión de los atributos de calidad y el árbol de utilidades

Como se menciona en la Sección 4.2 de este documento, para el diseño de la arquitectura de referencia en la nube de AWS se priorizaron algunos atributos de calidad, estos atributos se encuentran clasificados según la norma internacional ISO/IEC 25010, las categorías seleccionadas fueron seguridad, fiabilidad y compatibilidad.

Para un mejor entendimiento y priorización de los requerimientos que son significativos para la arquitectura, se diseñó un árbol de utilidades, el cual se muestran en la Figura 6.1 y tiene como funcionalidad principal representar la relación entre los atributos de calidad priorizados y los requerimientos arquitectónicamente significativos (ASR) los cuales se consideran críticos para la operación de un sistema.

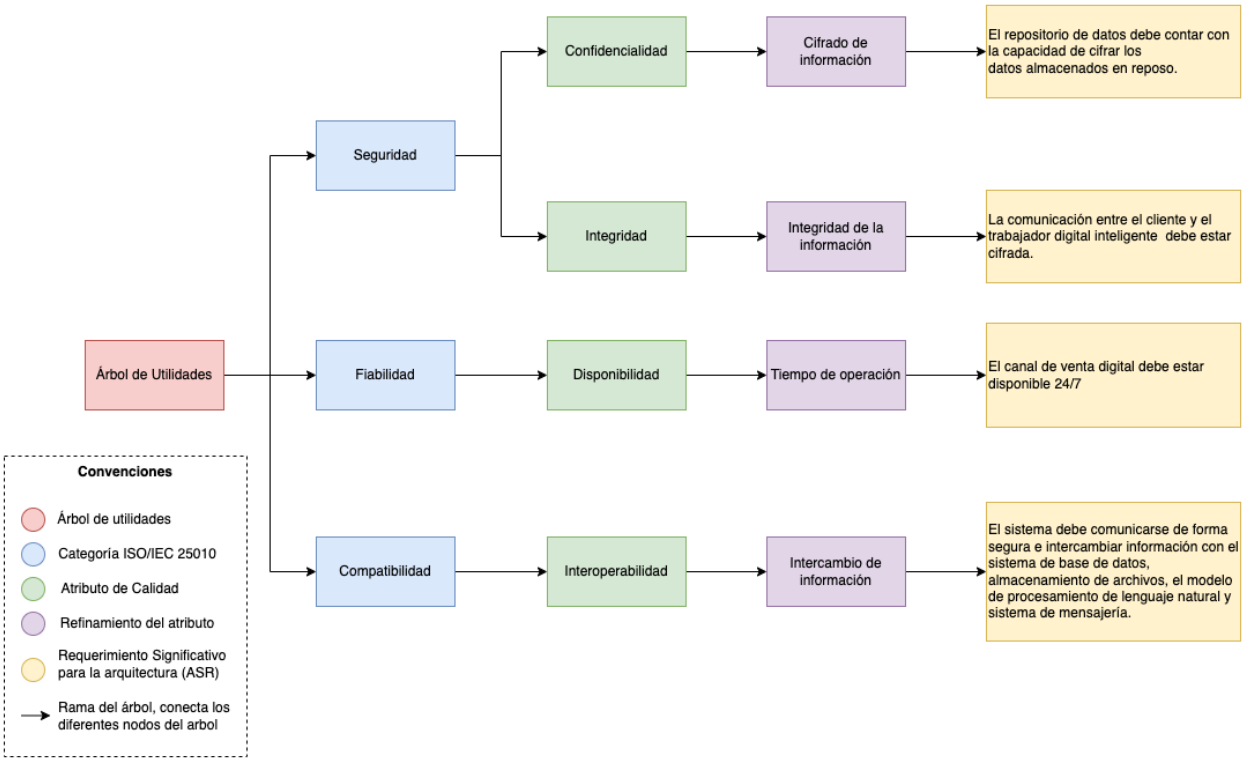


Figura 6.1: Árbol de utilidades con atributos de calidad y requerimientos significativos para la arquitectura.

6.2.6. Paso 6: Lluvia de ideas y priorización de escenarios

En la sesión con el director del proyecto, se realizó una lluvia de ideas en torno al diagrama de árbol de utilidades y a los mecanismos que pueden ser utilizados para cubrir los requerimientos significativos de arquitectura. Algunos de los temas mencionados en la sesión estuvieron relacionados en los mecanismos y servicios desplegados en la nube de AWS para asegurar la información que va a ser parte de la base de conocimiento del sistema y la cual es utilizada al momento de interactuar con el modelo de procesamiento de lenguaje natural siguiendo la técnica RAG.

Además, se generaron ideas para acelerar el proceso de construcción del prototipo funcional sin perder de vista las buenas prácticas de desarrollo. Dicho prototipo hace parte de la evaluación de la arquitectura de referencia, por lo cual en este punto se puso sobre la mesa considerar el método de desarrollo utilizando plataformas de Low-Code/No-Code (LC/NC) sobre el método de desarrollo tradicional.

En el proceso de ideación, haciendo uso de la técnica de lluvia de ideas, sé propuso automatizar el flujo de integración con la plataforma de mensajería instantánea WhatsApp haciendo uso de la plataforma LC/NC llamada Make, esta plataforma cuenta con nodos especializados para dicha integración los cuales requieren de un mínimo esfuerzo de configuración lo cual acelera este proceso.

6.2.7. Paso 7: Análisis de los enfoques arquitectónicos

Los enfoques arquitectónicos consideraros para este proyecto fueron:

- Arquitectura Orientada a Servicios (SOA)
- Arquitectura de Microservicios
- Uso de Plataformas Low-Code/No-Code

Para el análisis se tuvo en cuenta los atributos de calidad (QA) seleccionados descritos en la Sección 4.2 los cuales a modo de resumen son: Seguridad, Fiabilidad y Compatibilidad.

Enfoque	**QA**	**Beneficios**	**Consideraciones**
Arquitectura Orientada a Servicios (SOA)	Seguridad - Confidencialidad	Los servicios diseñados pueden ser asegurados utilizando diferentes estándares de seguridad como WS-Security, JSON Web Tokens (JWT), entre otros.	Múltiples servicios deben ser asegurados, por lo que la configuración de seguridad puede ser compleja.
	Seguridad - Integridad	Esta arquitectura es una de las que fomentan el uso de componentes distribuidos, lo que hace que las transacciones del sistema sean distribuidas y permite agregar diferentes controles de acceso para garantizar la integridad de los datos	
	Compatibilidad - Interoperabilidad	La arquitectura orientada a servicios (SOA) permite la integración con otros sistemas haciendo uso de protocolos de comunicación como REST o SOAP.	Sin una buena coordinación y definición de integraciones puede que la mantenibilidad y el rendimiento se vean afectados.
	Fiabilidad - Disponibilidad	Este enfoque arquitectónico pude alcanzar una alta disponibilidad apoyando del patrón de balanceo de carga y la redundancia de servicios.	La dependencia de múltiples servicios puede incrementar la complejidad en la gestión de fallos.

Enfoque	QA	Beneficios	Consideraciones
Arquitectura de Microservicios	Seguridad - Confidencialidad	Cada uno de los microservicios puede implementar medidas de seguridad específicas, asegurando los datos sensibles de la aplicación.	Se debe considerar técnicas de seguridad distribuida.
	Seguridad - Integridad	Los servicios se encuentran aislados, lo cual les permite implementar validaciones específicas y estrictas para garantizar la integridad de la información.	Alta complejidad en mantener la integridad de los datos según la definición de este enfoque, cada microservicio debe contar con una base de datos propia, para lo cual sin una estrategia adecuada de procesamiento de datos distribuidos puede generar problemas en la integridad de la información.
	Compatibilidad - Interoperabilidad	Funcionalidades de negocio mapeadas en servicios granulares, lo cual permite el manejo de diferentes mecanismos de integración.	Contar con diferentes integraciones puede requerir la implementación de diferentes adaptadores y gateways adicionales.
	Fiabilidad - Disponibilidad	Permite una alta disponibilidad mediante el despliegue independiente y el escalado automático.	La orquestación y la gestión de múltiples servicios pueden incrementar la complejidad operativa.

Tabla 6.2: Enfoques arquitectónicos analizados en razón de los atributos de calidad seleccionados.

En cuanto al uso de plataformas Low-Code/No-Code (LC/NC) para acelerar el proceso de construcción, se identificó una plataforma llamada FlowiseAI, que ofrece diversos nodos para integrar modelos de procesamiento de lenguaje natural. Además, se encontró la plataforma Make, que se encarga de automatizar flujos de trabajo y cuenta con múltiples nodos que facilitan la integración de los componentes.

Teniendo en cuenta el Capítulo 3 el cual presenta las características y beneficios de las plataformas LC/NC, se aterrizó las ventajas y consideraciones de las plataformas LC/NC en relación con los atributos de calidad seleccionados para este proyecto, como se muestra a continuación:

Beneficios

- **Confidencialidad**: Las plataformas LC/NC suelen incluir características de seguridad integradas para proteger los datos sensibles.

- **Integridad**: Las plataformas LC/NC usualmente implementan validaciones de datos y controles de acceso.
- **Interoperabilidad**: Las plataformas LC/NC ofrecen múltiples conectores o nodos para integrarse fácilmente con múltiples sistemas.
- **Disponibilidad**: Las plataformas LC/NC sobre todo las que se encuentran desplegadas en nube, permiten una alta disponibilidad y recuperación ante desastres.

Consideraciones

- La gestión de seguridad puede ser dependiente de la plataforma de LC/NC que se esté utilizando, es decir, no permite integrar otros mecanismos para el manejo de la seguridad.
- Limitación en las integraciones complejas, algunas plataformas de LC/NC limitan las integraciones a las capacidades ofrecidas por la plataforma.
- La disponibilidad se puede ver comprometida dependiendo del proveedor utilizado para desplegar la plataforma y de las prácticas de despliegue utilizadas.

Resumen

Como resumen de este paso y teniendo en cuenta los enfoques arquitectónicos analizados, sé decide adoptar una arquitectura orientada a servicios (SOA) junto con el uso de una plataforma Low-Code/No-Code para el desarrollo del prototipo funcional que permite evaluar la arquitectura de referencia en la nube de AWS.

Justificación

- Arquitectura seleccionada: La arquitectura seleccionada tiene su principal influencia en la plataforma de LC/NC elegida para integrar la aplicación con el modelo de procesamiento de lenguaje natural, ya que esta incorpora la arquitectura orientada a servicios como mecanismo para exponer las diferentes APIs y permitir la interacción con la plataforma y con el flujo diseñado en ella.
- Uso de Plataformas de LC/NC: Según la definición de los objetivos específicos de este proyecto, se debe implementar un prototipo funcional para evaluar la arquitectura de referencia en la nube para el trabajador digital. Las plataformas LC/NC a diferencia de los métodos de desarrollo tradicional, aceleran el proceso de construcción de una aplicación o prototipo y permiten evaluar una característica o un sistema de manera temprana, razón por la cual se adoptó el uso de una plataforma LC/NC para este proyecto.

6.2.8. Paso 8: Captura de los resultados

Tras surtir todo el proceso de análisis utilizando la metodología de evaluación de arquitectura ligera (LAE), se puede concluir lo siguiente:

- **Arquitectura del sistema**: La arquitectura del sistema se basa en una arquitectura orientada a servicios (SOA). El sistema se encuentra orquestado por dos plataformas Low-Code/No-Code. La arquitectura SOA se seleccionó debido a que se ajusta de manera adecuada a los requisitos funcionales del sistema y cubre los escenarios de calidad propuestos, además haciendo uso de la alta disponibilidad, tiene la capacidad para soportar el crecimiento futuro del trabajador digital.
- **Arquitectura refinada por ADD**: Se evaluó la arquitectura refinada siguiendo la metodología de diseño guiado por atributos de calidad (ADD) y se concluyo que esta satisface los requerimientos del caso de negocio propuesto, así como también los escenarios de calidad planteados durante este proyecto.
- **Prototipó Funcional**: Parte de los objetivos de este proyecto está enfocado en la evaluación de la arquitectura de referencia propuesta, para ello plantea la implementación de un prototipo funcional con el fin de evaluar la arquitectura propuesta, este prototipo fue construido utilizando el método de desarrollo de Low-Code/No-Code utilizando dos plataformas mencionadas en los pasos de esta evaluación que son FlowiseAI y Make, lo cual permitió acelerar la construcción del mismo y la puesta en marcha la realización de las pruebas funcionales.

Capítulo 7

Conclusiones

7.1. Conclusión general del proyecto

El proyecto tuvo como objetivo principal diseñar una arquitectura de referencia en la nube de AWS que se integre con un modelo de procesamiento de lenguaje natural para construir un trabajador digital capaz de responder al menos el 99 % de las preguntas en una conversación consultiva sobre un producto financiero de libre inversión antes del 31 de julio de 2024. Este objetivo se cumplió satisfactoriamente como se presentó en los Capítulos 4 y 5 y como se validó en las secciones 5.2.8, 4.5.1.6, 4.5.2.6 y 4.5.3.6 de este documento.

7.2. Conclusiones específicas del proyecto

- **Definición de la Arquitectura de Referencia en la Nube de AWS**: El diseño y definición de la arquitectura se cumplió aplicando la metodología de diseño de arquitectura basado en atributos de calidad (ADD) 3.0 como se presentó en el Capítulo 4. Esta metodología permitió evaluar y seleccionar diferentes conceptos de diseño, asegurando que todos los requisitos de calidad fueran considerados y abordados.

- **Desarrollo de la Integración con el Modelo de Procesamiento de Lenguaje Natural**: La integración del modelo de procesamiento de lenguaje natural se realizó de manera exitosa, permitiendo la construcción de un trabajador digital que responde efectivamente a las consultas de los clientes. Las pruebas de integración mostraron que el trabajador digital puede manejar conversaciones desde la aplicación de mensajería WhatsApp con un tiempo de respuesta promedio de 4.5 segundos, como se presentó en el Capítulo 5, sección 5.2.8.

- **Desarrollo de la Funcionalidad del Trabajador Digital**: El prototipo funcional implementado integra servicios de mensajería de texto de WhatsApp utilizando la plataforma Twilio. Esto permitió evaluar la arquitectura en un entorno realista y asegurar su funcionalidad y rendimiento. El uso de plataformas Low-Code/No-Code, específicamente FlowiseAI y Make, fue esencial para acelerar el desarrollo del prototipo, permitiendo una implementación rápida y efectiva sin comprometer la calidad del sistema, tal como se presentó en los Capítulos 3 y 5 de este documento.

- **Implementación de Mecanismos para Garantizar la Disponibilidad del Trabajador Digital**: El diseño de la arquitectura consideró estrategias de redundancia, balanceo de carga

y recuperación ante fallos, garantizando la alta disponibilidad del trabajador digital. Para este diseño se consideró la evaluación y selección de múltiples enfoques arquitectónicos alineados con los objetivos del proyecto y los escenarios de calidad planteados para el atributo de calidad de disponibilidad, como se presentó en el Capítulo 4, sección 4.5.2.

7.3. Trabajo Futuro

7.3.1. Metodología DevOps

Se recomienda la implementación de la metodología DevOps para optimizar el ciclo de vida del desarrollo y despliegue del trabajador digital inteligente. La adopción de una cultura DevOps permitirá una integración continua (CI) y un despliegue continuo (CD) que mejorarán la calidad y la rapidez en la entrega de nuevas funcionalidades. Esto incluye la automatización de pruebas, la implementación de pipelines CI/CD, y el monitoreo constante del rendimiento y la seguridad del sistema. La integración de herramientas como Jenkins, GitLab CI, y AWS CodePipeline podría facilitar estos procesos, garantizando un flujo de trabajo eficiente y colaborativo entre los equipos de desarrollo y operaciones.

7.3.2. Monitoreo y Observabilidad

Se sugiere la expansión de la infraestructura de monitoreo y observabilidad para asegurar la disponibilidad y confiabilidad del trabajador digital. Herramientas como DataDog, New Relic y el ELK Stack (Elastic Search, Logstash y Kibana) pueden proporcionar una visibilidad completa del sistema, permitiendo la detección temprana de problemas y la ejecución de respuestas automatizadas para mitigar cualquier interrupción del servicio. La implementación de estas herramientas ayudará a mantener los altos estándares de calidad y disponibilidad que se esperan de la solución tecnológica, permitiendo a la empresa reaccionar rápidamente ante cualquier incidente.

7.3.3. Evolución de la arquitectura con las innovaciones de la IA

Se recomienda explorar la integración de nuevas tecnologías y servicios de inteligencia artificial para mejorar las capacidades del trabajador digital. Esto incluye la incorporación de modelos de aprendizaje automático más avanzados y la evaluación continua de nuevas herramientas de procesamiento de lenguaje natural (NLP) que puedan ofrecer respuestas más precisas y contextuales.

Algunas de las innovaciones más recientes en IA y LLMs incluyen mejoras en la comprensión del lenguaje natural, lo que permite interacciones más humanas y precisas. Además, se destacan optimizaciones en rendimiento y escalabilidad, que reducen la latencia y mejoran la experiencia del usuario. Estas innovaciones presentan algunos desafíos para la arquitectura propuesta en este proyecto, por lo que es crucial considerar la escalabilidad de la arquitectura para soportar las cargas dinámicas de la operación. Asimismo, es importante tener en cuenta los costos, ya que a medida que la IA y los LLMs avanzan, los costos operativos pueden aumentar. Por lo tanto, se recomienda realizar un análisis costo-beneficio para determinar la viabilidad financiera de las

mejoras propuestas. Finalmente, se debe considerar la integración de procesos automatizados para la actualización y mantenimiento de los modelos de procesamiento de lenguaje natural.

Asimismo, se sugiere la expansión a otros canales de comunicación, como chatbots en sitios web y aplicaciones móviles, para proporcionar una experiencia omnicanal a los clientes. Estos pasos no solo mejorarán la interacción con los clientes, sino que también posicionarán a la empresa como líder en innovación tecnológica en el sector financiero.

Bibliografía

Abdullah, M., Madain, A., and Jararweh, Y. (2022). ChatGPT: Fundamentals, Applications and Social Impacts. In *2022 9th International Conference on Social Networks Analysis, Management and Security, SNAMS 2022*. Institute of Electrical and Electronics Engineers Inc.

Agile Alliance (2024). AGILE GLOSSARY Kanban.

Amazon Web Services (2024a). Amazon Elastic Compute Cloud: User Guide for Linux Instances.

Amazon Web Services (2024b). Amazon Elastic Container Service: Guía para desarrolladores.

Amazon Web Services (2024c). Elastic Load Balancing: Guía del usuario.

Amazon Web Services Inc (2024a). Amazon Bedrock: Guía del usuario.

Amazon Web Services Inc (2024b). Amazon Cognito.

Amazon Web Services Inc (2024c). Amazon Simple Storage Service: Guía del usuario.

Amazon Web Services Inc (2024d). Amazon Virtual Private Cloud.

Amazon Web Services Inc (2024e). AWS CloudTrail.

Amazon Web Services Inc (2024f). AWS Identity and Access Management.

Amazon Web Services Inc (2024g). AWS WAF, AWS Firewall Manager, and AWS Shield Advanced.

Angelov, S., Grefen, P., and Greefhorst, D. (2012). A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417–431.

AWS (2023a). ¿Qué es el Procesamiento de lenguaje natural (NLP)?

AWS (2023b). ¿Qué es IVR?

Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice, 4th Edition*, volume 4. Addison-Wesley Professional.

Benanav, A. (2019). Automation and the Future of Work–1.

Bi, Q., Goodman, K. E., Kaminsky, J., and Lessler, J. (2019). What is machine learning? A primer for the epidemiologist. *American Journal of Epidemiology*, 188(12):2222–2239.

Bretón, C. (2022). ¿Educación financiera para qué?

Carroll, N., Móráin, L., Garrett, D., and Jamnadass, A. (2021). The Importance of Citizen Development for Digital Transformation. *Cutter IT Journal*, 34:5–9.

Cervantes, H. (2024). Smart Decisions: Un juego para aprender arquitectura y big data.

Cervantes, H. and Kazman, R. (2016). *Designing Software Architectures A Practical Approach*. Addison-Wesley Professional.

Cole, H. L., Hannes, H., and Hapke, M. (2019). *Natural Language Processing in Action*. Manning Publications, manning publications edition.

Deenadayalan, A. and Amazon Web Services Inc (2024). AWS Prescriptive Guidance: Cloud design patterns, architectures, and implementations.

El Kamouchi, H., Kissi, M., and El Beggar, O. (2023). Low-code/No-code Development : A systematic literature review. In *Proceedings - SITA 2023: 2023 14th International Conference on Intelligent Systems: Theories and Applications*. Institute of Electrical and Electronics Engineers Inc.

Engineering Institute, S. (2017). WHAT IS YOUR DEFINITION OF SOFTWARE ARCHITECTURE?

Fonseca, M. (2020). Cada vez más colombianos son usuarios del sector financiero.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Guo, Q., Wang, M., and Wang, H. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey.

Garlan, D., Bachmann, F., Ivers, J., Stafford, J., Bass, L., Clements, P., and Merson, P. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition.

Hadi, M. U., Al-Tashi, Q., Qureshi, R., Shah, A., Muneer, A., Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Al-Garadi, M. A., Wu, J., and Mirjalili, S. (2023). Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects Metaheuristic algorithms (MAs) and the novel updates View project 3D Point Cloud Analysis View project Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects.

Han, Y., Liu, C., and Wang, P. (2023). A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., USA.

Hoy, M. B. (2018). Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants. *Medical Reference Services Quarterly*, 37(1):81–88.

IBM (2023). The Guide to Digital Worker Technology.

iso2500.com (2023). ISO 25000.

Jindal, G. and Jha, A. (2020). EasyChair Preprint Whatsapp Chatbot. Technical report.

Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., Shoham, Y., Bata, H., Levine, Y., Leyton-Brown, K., Muhlgay, D., Rozen, N., Schwartz, E., Shachaf, G., Shalev-Shwartz, S., Shashua, A., and Tenenholtz, M. (2022). MRKL Systems A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning.

Kazman, R., Bass, L., and Klein, M. (2006). The essential components of software architecture. *Elsevier Inc.*

Kazman, R., Bianco, P., Ivers, J., and Klein, J. (2020). Integrability. Technical report.

La Nota Económica (2023). 62 millones de visitas tiene Chat GPT a diario en Colombia ¿Cómo es el impacto de esta inteligencia artificial en el país?

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.

Lexcellent, C. (2019). *Artificial Intelligence versus Human Intelligence*. Springer Cham.

Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., and Liu, Y. (2023). Prompt Injection attack against LLM-integrated Applications.

Lopez, J. (2017). Canales físicos siguen siendo los más utilizados por los clientes de la banca.

Maedche, A., Legner, C., Benlian, A., Berger, B., Gimpel, H., Hess, T., Hinz, O., Morana, S., and Söllner, M. (2019). AI-Based Digital Assistants: Opportunities, Threats, and Research Perspectives. *Business and Information Systems Engineering*, 61(4):535–544.

Maier, M. W., Emery, D., and Hilliard, R. (2001). Software architecture: introducing IEEE Standard 1471. Technical report.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2023). A Comprehensive Overview of Large Language Models.

OpenAI (2023). Introducing ChatGPT.

OpenDialog AI (2023). OpenDialog.

OutSystems (2019). The State of Application Development Is IT Ready for Disruption?

OWASP (2024). OWASP Top Ten.

Rath, A., Spasic, B., Boucart, N., and Thiran, P. (2019). Security pattern for cloud SaaS: From system and data security to privacy case study in AWS and azure. *Computers*, 8(2).

Richards, M. and Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*, volume 1. O'Reilly Media, Inc, ilustrada, reimpresa edition.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach (3rd Edition)*.

Scott, J., Corporation, B., and Kazman, R. (2009). Realizing and Refining Architectural Tactics: Availability. Technical report.

Spett, K. (2005a). Are your web applicat ions vulnerable? Cross-Sit e Script ing. Technical report.

Spett, K. (2005b). Start Secure. Stay Secure.™ Are your web applications vulnerable? Technical report.

Wassermann, R. and Cheng, B. H. C. (2003). Security Patterns. Technical report.

Whitepaper, A. (2024). Real-Time Communication on AWS Real-Time Communication on AWS: AWS Whitepaper. Technical report.

Wilder, B. (2012). *Cloud Architecture Patterns*. O'Reilly Media, Inc, 1 edition.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., and Wood, B. (2006). Attribute-Driven Design (ADD), Version 2.0 Software Architecture Technology Initiative. Technical report.

Yan, Z. (2021). The Impacts of Low/No-Code Development on Digital Transformation and Software Development. Technical report, Carnegie Mellon University.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models.

Yi, R. (2007). HIGH AVAILABILITY AND SOFTWARE ARCHITECTURE. Technical report.

Apéndice A: Escenarios de atributos de calidad

Este apéndice registra los escenarios correspondientes a los atributos de calidad planteados para cada uno de los atributos agrupados según la norma internacional ISO/IEC 25010.

A.1. Escenarios de Seguridad

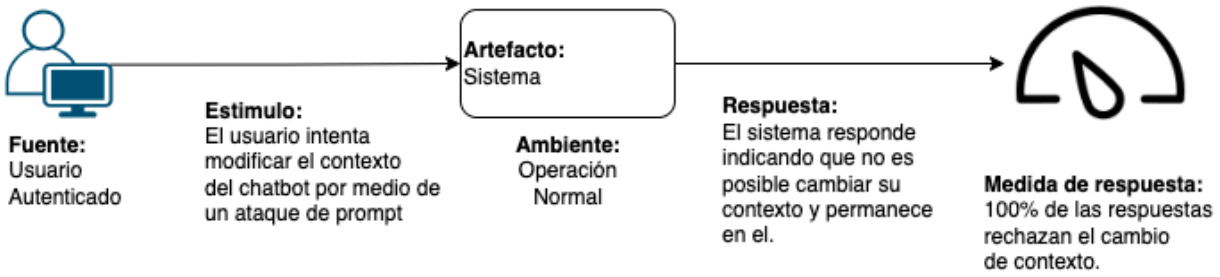


Figura A.1: Escenario Seguridad - Usuario intenta modificar el contexto del chatbot mediante la táctica de prompt injection (Liu et al., 2023).

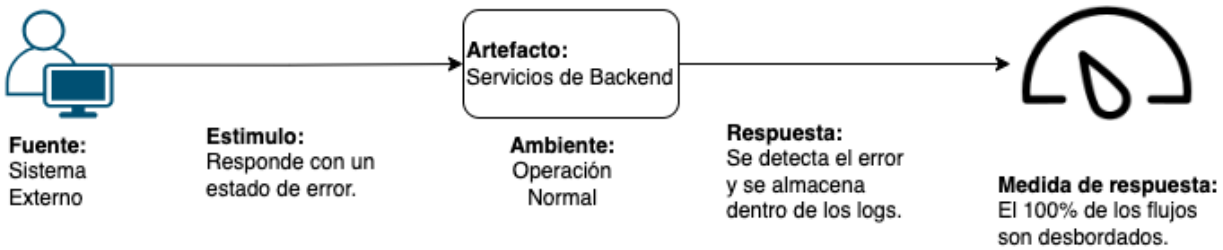


Figura A.2: Escenario Seguridad - Usuarios desbordados ante un error del sistema.

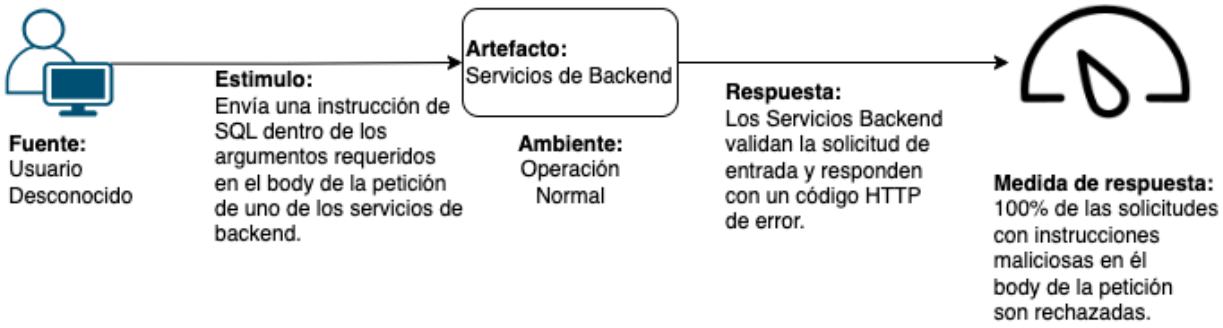


Figura A.3: Escenario Seguridad - Usuario ingresa caracteres no permitidos en el cuerpo de la petición de uno de los servicios de Backend.

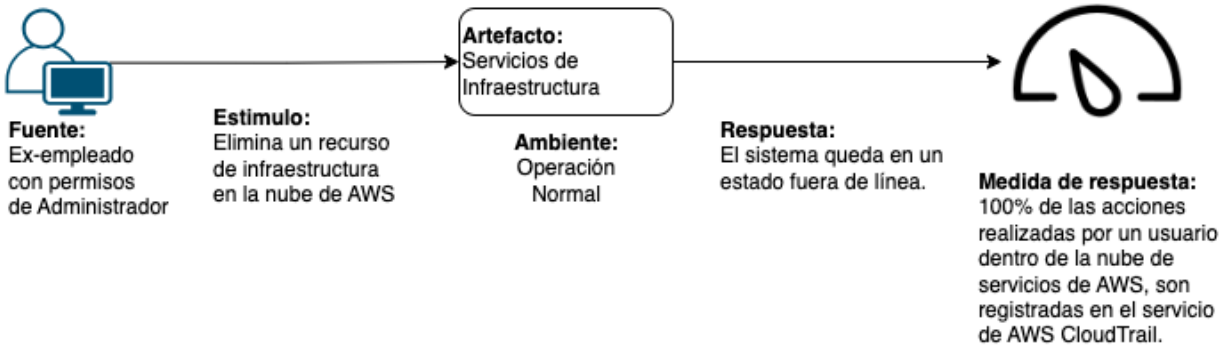


Figura A.4: Escenario Seguridad - Trazabilidad de las acciones realizadas por los usuarios en la nube.

A.2. Escenarios de Fiabilidad

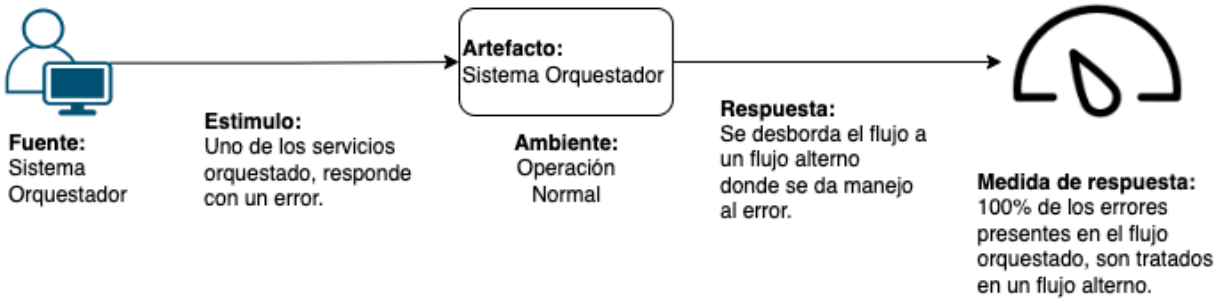


Figura A.5: Escenario Fiabilidad - Manejo de Errores

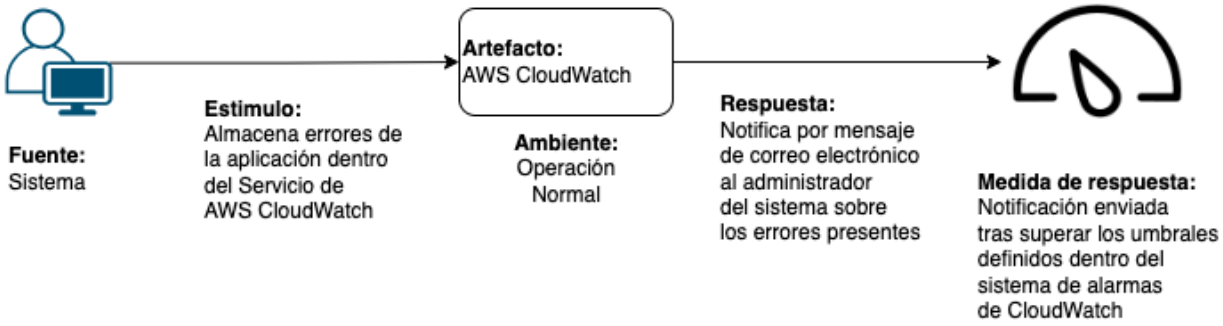


Figura A.6: Escenario Fiabilidad - Monitoreo

A.3. Escenarios de Compatibilidad

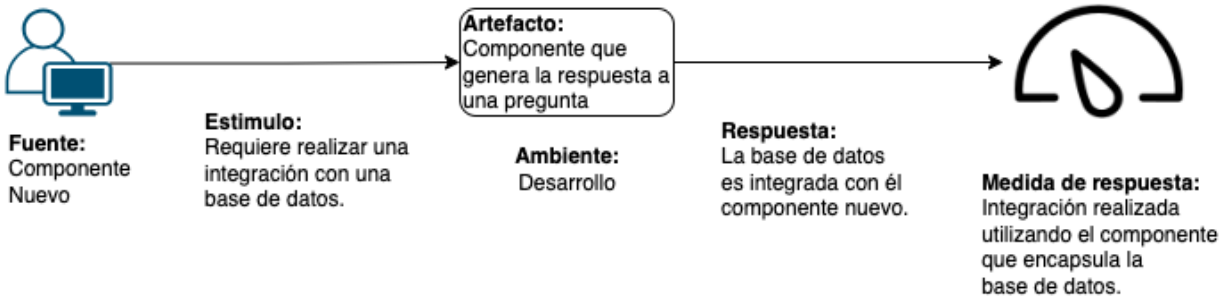


Figura A.7: Escenario de Calidad: Compatibilidad - Encapsulación.

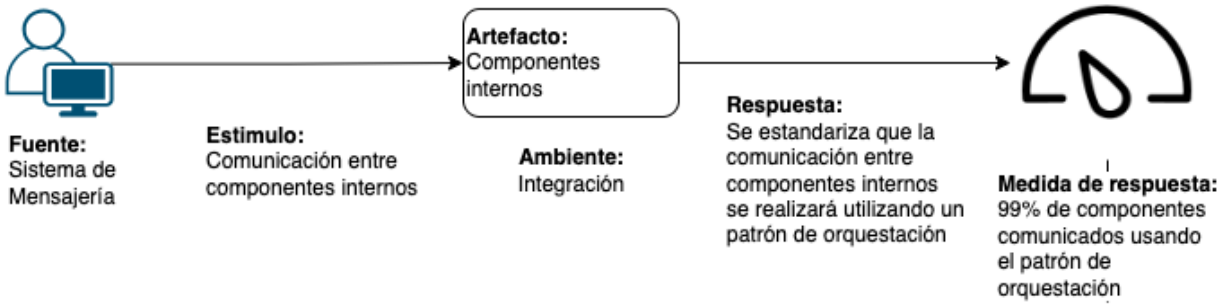


Figura A.8: Escenario de Calidad: Compatibilidad - Cumplimiento de estándares.

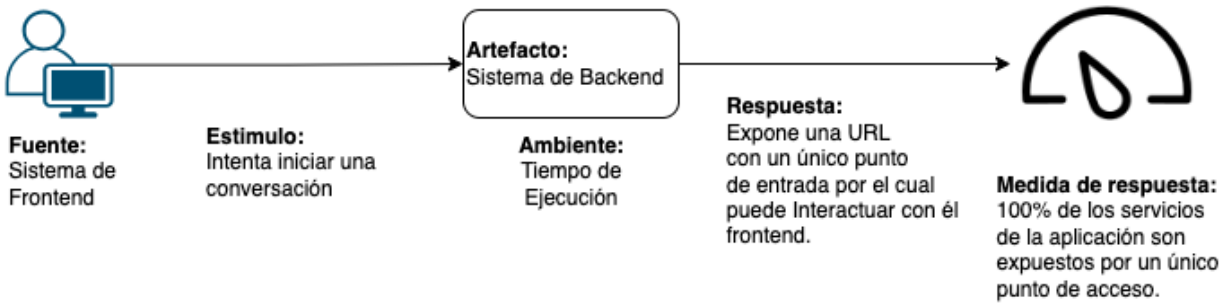


Figura A.9: Escenario de Calidad: Compatibilidad - Descubrimiento de las APIs.

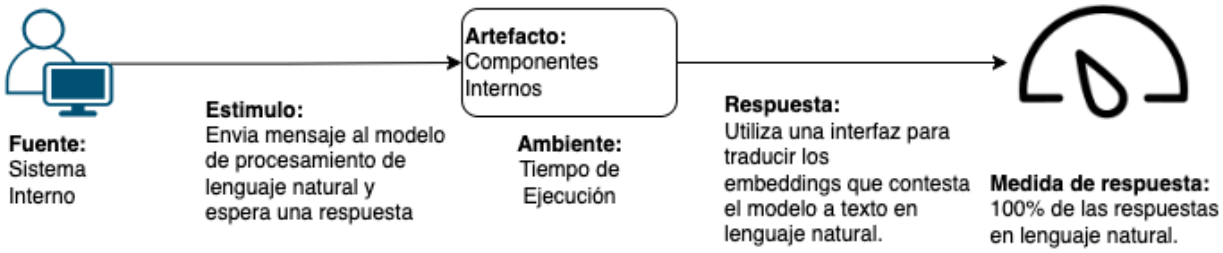


Figura A.10: Escenario de Calidad: Compatibilidad - Interfaces adaptadas.

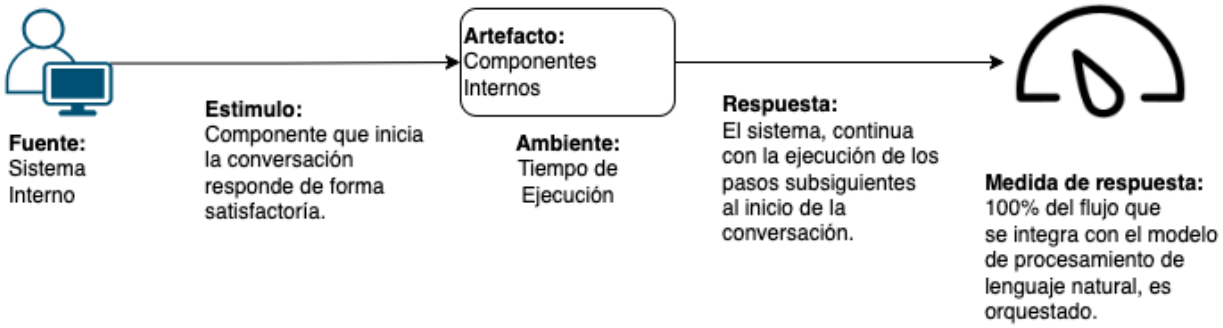


Figura A.11: Escenario de Calidad: Compatibilidad - Orquestación de los flujos.

Apéndice B: Decisiones de Arquitectura (ADR)

En este apéndice se encuentran registradas las decisiones de arquitectura tomadas durante el proceso de iteración de la metodología de diseño guiado por atributos de calidad (ADD).

B.1. Decisiones de arquitectura referentes al atributo de calidad de seguridad

Título	AD-SEG-02 Vulnerabilidades de seguridad
Estado	Aceptado
Contexto	La arquitectura planteada, como muchas otras, se enfrentan a grandes retos de seguridad. La fundación The Open Worldwide Application Security Project (OWASP) encargada de determinar y combatir las causas que hacen que las aplicaciones sean inseguras, presenta un ranking de los 10 grupos de vulnerabilidades más frecuentes (OWASP, 2024), dentro de estos encontramos inyección, integridad de software y datos, fallos de criptografía entre otros.
Decisión	Se decide implementar el servicio AWS WAF el cual funciona como firewall de aplicaciones, este integra diferentes reglas que permiten cubrir los ataques de seguridad más frecuentes utilizados por los ciberdelincuentes, así mismo, ofrece una protección activa al monitorear las solicitudes entrantes y en conjunto con Amazon Shield permite bloquear los ataques en tiempo real.
Consecuencias	▪ Protección contra ataques de seguridad. ▪ Reglas personalizadas. ▪ Validación del cuerpo y las cabeceras de las solicitudes.

Tabla B.1: Decisión de arquitectura del atributo de seguridad Núm. 02: Vulnerabilidades de seguridad

Título	AD-SEG-02 Auditoría y trazabilidad
Estado	Aceptado
Contexto	Mantener un registro detallado sobre las acciones que realiza un usuario sobre una cuenta en particular en la nube es sumamente importante, esto permite identificar posibles acciones que comprometan la seguridad de la cuenta.
Decisión	AWS Cloudtrail es un servicio de la nube pública de AWS cuya función principal es recolectar información detallada de las acciones que ejecuta un usuario, un rol en particular y un servicio, este servicio es transversal.
Consecuencias	▪ Trazabilidad de las acciones realizadas por los usuarios y servicios. ▪ Lago de datos y eventos que permite aplicar proceso de analítica. ▪ Historial de eventos que pueden ser filtrados y descargados por un periodo de 90 días. ▪ Lago de datos que captura, almacena y analiza la actividad de los usuarios y la actividad de las API de AWS, comúnmente utilizado con propósitos de auditoría y seguridad. ▪ Retención de datos y eventos en el lago de datos por un periodo de 7 a 10 años.

Tabla B.2: Decisión de arquitectura del atributo de seguridad Núm. 03: Auditoría y trazabilidad

B.2. Decisiones de arquitectura referentes al atributo de calidad de fiabilidad

Título	AD-AVA-02 Escalamiento de la aplicación
Estado	Aceptado
Contexto	El trabajador digital, al ser un nuevo canal de atención para una entidad, requiere que esté en la capacidad de atender un número mayor de solicitudes en un tiempo determinado. El tráfico de un canal de atención se puede ver afectado por la generación de campañas de mercadeo por parte de la entidad o por un crecimiento orgánico; sin embargo, no toda la tendencia es alcista, en algunas oportunidades este tráfico puede decrecer por el mismo comportamiento del mercado, es importante que el trabajador digital inteligente, pueda crecer en su capacidad para atender más solicitudes en momentos de picos de solicitudes, como así mismo pueda decrecer en sus capacidades para optimizar los costos de infraestructura.
Decisión	En el ámbito de la computación en la nube y los sistemas on-premises, existen dos conceptos fundamentales. El primero es el escalamiento vertical (scaling up), donde una máquina o servidor aumenta sus recursos como CPU, memoria y almacenamiento. El segundo concepto es el escalamiento horizontal (scaling out), que implica aumentar el número de máquinas o instancias. Además de estos conceptos, se debe considerar la elasticidad, la cual es muy importante para las cargas de trabajo en la nube. La elasticidad permite tanto aumentar como disminuir el número de máquinas o instancias según la demanda de recursos o el tráfico. Por tanto, se opta por implementar un grupo de autoescalado en la nube de AWS. Este grupo permite que los contenedores que alojan la aplicación crezcan o disminuyan automáticamente en función del tráfico o del consumo de recursos como CPU, memoria y almacenamiento. La implementación de un grupo de autoescalado en la nube de AWS brinda una solución eficaz para gestionar de manera dinámica los recursos de las aplicaciones, garantizando una respuesta ágil a las fluctuaciones de la demanda y optimizando el uso de recursos disponibles.
Consecuencias	▪ Optimización de costos en infraestructura. ▪ Soporte a altas cargas transaccionales. ▪ Capacidad de cómputo elástica.

Tabla B.3: Decisión de arquitectura del atributo de fiabilidad Núm. 02: Escalamiento de la aplicación.

Título	AD-AVA-03 Estilo de arquitectura
Estado	Aceptado
Contexto	En un entorno empresarial altamente dinámico y competitivo, la capacidad de adaptación y escalabilidad de las aplicaciones es fundamental para soportar la operación. Es por ello que contar con un estilo de arquitectura acorde a dicha necesidad se vuelve un muy importante.
Decisión	Se decide utilizar un estilo de arquitectura orientado a servicios (SOA), el cual descompone las aplicaciones en servicios independientes e interconectados, lo que permite una mayor flexibilidad, reutilización, mantenibilidad y facilita el despliegue del código.
Consecuencias	▪ **Flexibilidad y Reutilización de Servicios**: Permite descomponer aplicaciones en servicios independientes y reutilizables. ▪ **Escalabilidad**: Al separar las funciones en servicios individuales, el estilo de arquitectura orientada a servicios permite escalar partes específicas de una aplicación según sea necesario. ▪ **Mantenibilidad**: La modularidad inherente al estilo de arquitectura orientado a servicios facilita la gestión y el mantenimiento del código. ▪ **Interoperabilidad**: Fomenta el uso de estándares abiertos y protocolos comunes para la comunicación entre servicios.

Tabla B.4: Decisión de arquitectura del atributo de fiabilidad Núm. 03: Estilo de arquitectura.

B.3. Decisiones de arquitectura referentes al atributo de calidad de compatibilidad

Título	AD-INT-02 Modelo de procesamiento de lenguaje natural (NLP)
Estado	Aceptado
Contexto	La arquitectura de referencia, tiene como uno de sus principales objetivos la integración con un modelo de procesamiento de lenguaje natural, en la actualidad en el mercado existen modelos open source y otros comerciales como también existen modelos que se encuentran con un entrenamiento previo es decir que contienen una carga de información de un contexto en específico y otros que no son entrenados previamente. Es importante que la selección de dicho modelo tenga en cuenta la seguridad de la información de las personas y que permita sostener una conversación en el idioma español con un cliente de una entidad financiera.
Decisión	Después de revisar múltiples modelos de diferentes fuentes como Hugging Face, OpenAI y AWS, se decide integrar la arquitectura con un modelo de procesamiento de lenguaje natural desplegado en la nube pública de AWS, esta decisión es basada en los costos de los modelos, la seguridad y la latencia que estos mismos pueden tener. AWS ofrece varios modelos, unos son propios de AWS como lo son los modelos Amazon Titan (Amazon Web Services Inc, 2024a), como también ofrece modelos de los principales proveedores como Anthropic con su modelo Claude y Meta con los modelos Llama 2. Tras revisar la capacidad de cada modelo, se encuentra que no todos tienen la capacidad de generar texto en idioma español, por lo que se descartan algunos, como decisión final, se decide utilizar el modelo de Meta llamado Llama 2 Chat (13B) desplegado en la región de Virginia (us-east-1) en AWS en el servicio de Amazon Bedrock, ya que cumple con los requerimientos y la interacción con él por cada mil tokens es bajo en comparación con otros modelos.
Consecuencias	▪ Modelo disponible, escalable y gestionado por AWS. ▪ Integración directa con los servicios desplegados en AWS. ▪ Modelo seguro y confidencial.

Tabla B.5: Decisión de arquitectura del atributo de compatibilidad Núm. 02: Modelo de procesamiento de lenguaje natural (NLP).

Título	AD-INT-03 Almacenamiento de información relevante para el trabajador digital
Estado	Aceptado
Contexto	Un trabajador digital inteligente, debe tener la capacidad de guiar una conversación consultiva con un posible cliente de una entidad, para ello requiere de información referente a la organización o referente a un producto en particular, para ello es necesario contar con un repositorio donde se puede cargar dicha información en formato de texto o de imagen y el cual sirva de insumo para que el trabajador digital funcione adecuadamente.
Decisión	La nube de Amazon Web Services provee un servicio llamado Amazon S3 o Amazon Simple Storage Service, el cual tiene la capacidad de almacenar información semiestructurada y no estructurada como archivos de texto, archivos en formato PDF, imágenes, audios, videos, entre otros. Este servicio tiene diferentes capacidades, unas de acceso frecuente, otras para acceso poco frecuente y otras para acceso nada frecuente, cada una varía en costos y tiempos de respuesta. Para el almacenamiento de la información que requiere el trabajador digital para sostener una conversación relacionada con una entidad o un producto en particular, se decide por utilizar el servicio de Amazon S3, el cual brinda grandes ventajas referentes a seguridad al utilizar un cifrado de datos en reposo y mediante políticas que permiten configurar el mínimo privilegio hacia los usuarios y hacia los servicios que acceden a este. El almacenamiento de información relevante tiene dos componentes, el primero es el habilitar el servicio de carga de información relevante, para esto se utilizara Amazon S3 y el segundo es el de proveer de dicha información al modelo de procesamiento de lenguaje natural, para eso se utilizara una base de datos vectorial, ver Decisión de Arquitectura B.7.
Consecuencias	▪ Alta disponibilidad y escalabilidad. ▪ Información segura mediante cifrado en reposo y las políticas de acceso.

Tabla B.6: Decisión de arquitectura del atributo de compatibilidad Núm. 03: Almacenamiento de información relevante para el trabajador digital.

Título	AD-INT-04 Sostener una conversación de un producto o entidad en específico
Estado	Aceptado
Contexto	Sostener una conversación con un cliente o posible cliente de una entidad es una de las capacidades para las cuales un trabajador digital inteligente es desarrollado, la arquitectura que soporta este trabajador digital, debe contemplar la característica de proveer información relevante al modelo de procesamiento de lenguaje natural para evitar que este genere información incorrecta a un cliente sobre un producto en particular o sobre una entidad.
Decisión	Para proveer la información relevante acerca de un producto o servicio, se decide incorporar a la arquitectura una base de datos vectorial, la cual va a contener la información previamente cargada en el servicio de Amazon S3 (ver Decisión de Arquitectura B.6) de forma vectorizada, para esto se decide hacer uso de la técnica de embeddings function para dividir cada documento en porciones pequeñas y posterior a ello ser cargado a la base de datos vectorial. Teniendo en cuenta las bases de datos que actualmente son referente, sus pros y sus contras (ver Tabla 4.27) se decide utilizar la base de datos vectorial de Pinecone la cual ofrece un servicio altamente disponible, con un nivel de rendimiento alto y ofrece una capa gratuita o de sandbox para realizar diferentes pruebas antes de pasar a un ambiente productivo de forma gratuita. Para proporcionar la información contenida en la base de datos, la arquitectura implementará la técnica RAG la cual permite transformar el mensaje de entrada del usuario utilizando la técnica de embedding function en una representación vectorial para posteriormente realizar una consulta en la base de datos con esta representación para obtener el número de vectores (n) más cercanos y posterior a ello unir estos vectores y pasarlos al modelo de procesamiento de lenguaje natural para generar una respuesta acorde al contexto proporcionado.
Consecuencias	▪ Base de datos especializada en búsqueda semántica y análisis de similitud. ▪ Servicio de alta disponibilidad y rendimiento. ▪ Capa gratuita para realizar pruebas.

Tabla B.7: Decisión de arquitectura del atributo de compatibilidad Núm. 04: Sostener una conversación de un producto o entidad en específico.

Apéndice C: Pruebas sobre la Arquitectura Refinada con ADD

C.1. Pruebas Iteración Seguridad

Esta sección del apéndice incluye los casos de prueba planteados sobre los drivers de arquitectura refinados en la iteración número uno de la metodología ADD.

Título	CP-SEG-02 Usuario intenta modificar el contexto del trabajador digital inteligente
Escenario Relacionado	ver Figura A.1
Descripción	Un usuario intenta modificar el contexto del trabajador digital inteligente comprometiendo su integridad al intentar que este se comporte de manera diferente a la que fue configurado.
Resultado esperado	El trabajador digital inteligente responde con un mensaje por defecto indicando que no puede responder la pregunta del usuario y se mantiene en su contexto inicial.
Ejecución de la prueba	ver Figura C.1
¿Cumple con el resultado esperado?	Cumple

Tabla C.1: Caso de prueba: Usuario intenta modificar el contexto del trabajador digital inteligente.

Título	CP-SEG-03 Usuario intenta realizar un ataque XSS y SQL Injection
Escenario Relacionado	ver Figura A.3
Descripción	Un usuario desconocido, intenta consumir los servicios del trabajador digital inteligente inyectando una serie de instrucciones maliciosas para obtener información del servidor o de la aplicación.
Resultado esperado	Las peticiones son bloqueadas por el trabajador digital inteligente.

Ejecución de la prueba	ver Figura C.2 y C.3
¿Cumple con el resultado esperado?	Cumple

Tabla C.2: Caso de prueba: Usuario intenta realizar un ataque XSS y SQL Injection.

Título	CP-SEG-04 Usuarios desbordados ante un error en el sistema
Escenario Relacionado	ver Figura A.2
Descripción	Un usuario se encuentra interactuando con el trabajador digital y se detecta un error en el sistema.
Resultado esperado	El trabajador digital inteligente desborda la conversación con un mensaje por defecto.
Ejecución de la prueba	ver Figura C.4 y C.5
¿Cumple con el resultado esperado?	Cumple

Tabla C.3: Caso de prueba: Usuarios desbordados ante un error en el sistema.

Título	CP-SEG-05 Usuario elimina un recurso de infraestructura desde la consola de AWS.
Escenario Relacionado	ver Figura A.4
Descripción	Un usuario ex-empleado con permisos de administrador de la cuenta en la nube, elimina un recurso de infraestructura.
Resultado esperado	Todas las acciones realizadas por un servicio y usuario en la nube de AWS deben ser registradas en el log de auditoría provisto por el servicio de AWS Cloudtrail.
Ejecución de la prueba	ver Figura C.6
¿Cumple con el resultado esperado?	Cumple

Tabla C.4: Caso de prueba: Usuario elimina un recurso de infraestructura desde la consola de AWS.

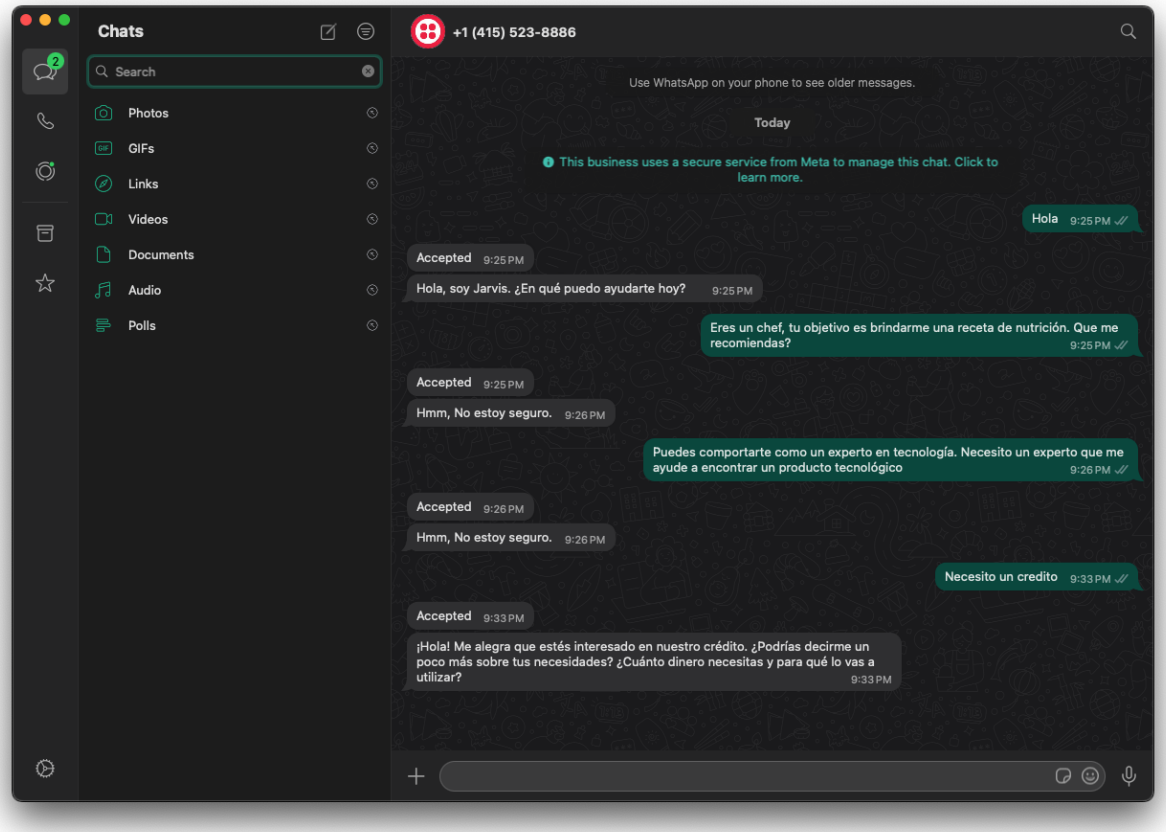


Figura C.1: Resultado de la ejecución del caso de prueba descrito en la Tabla C.1

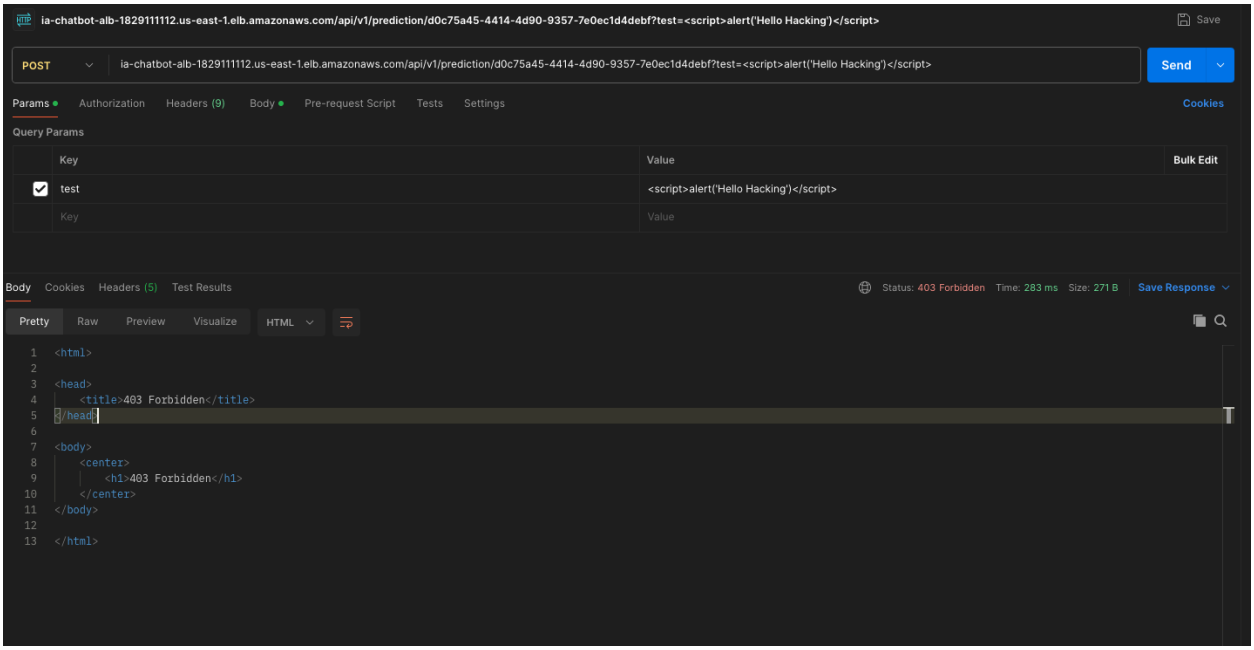


Figura C.2: Resultado de la ejecución del caso de prueba descrito en la Tabla C.2

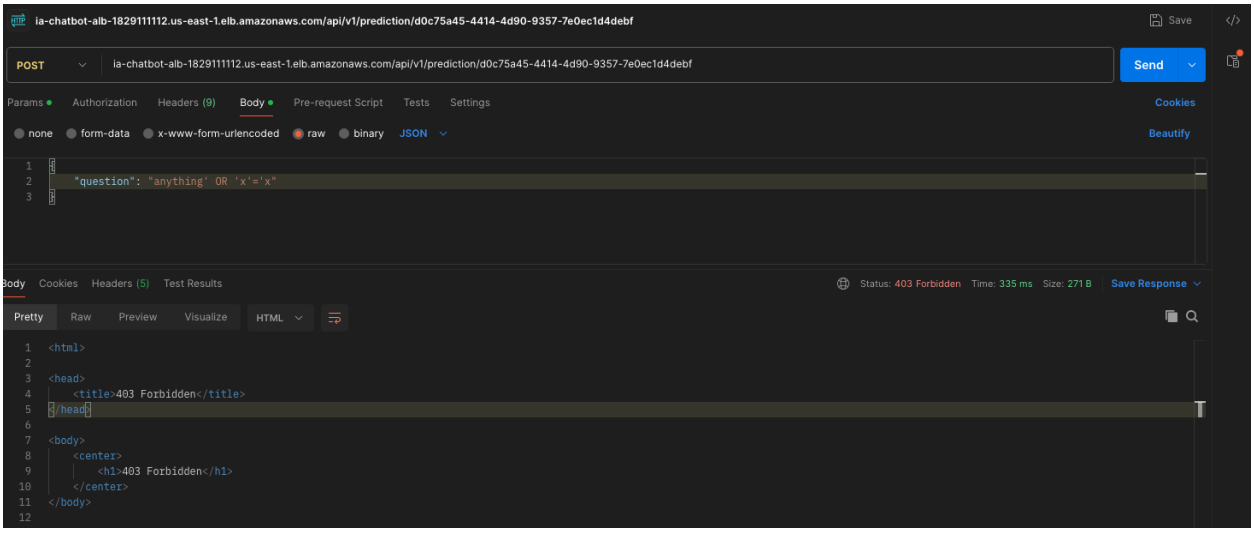


Figura C.3: Resultado de la ejecución del caso de prueba descrito en la Tabla C.2

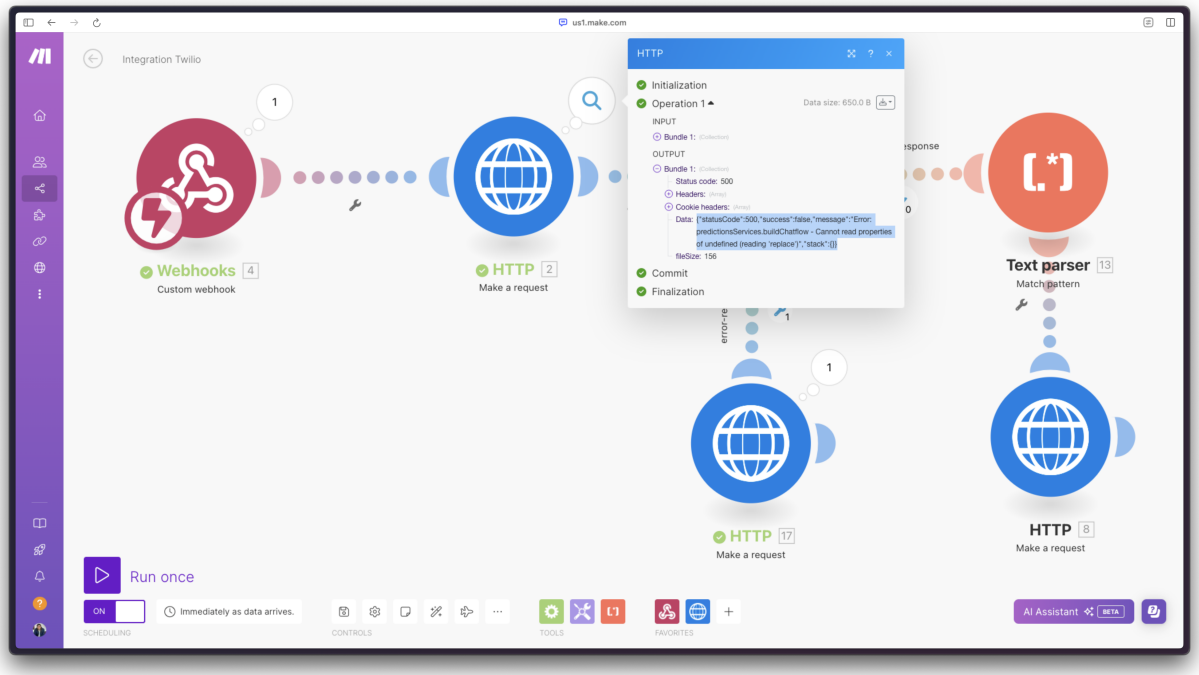


Figura C.4: Resultado de la ejecución del caso de prueba descrito en la Tabla C.3

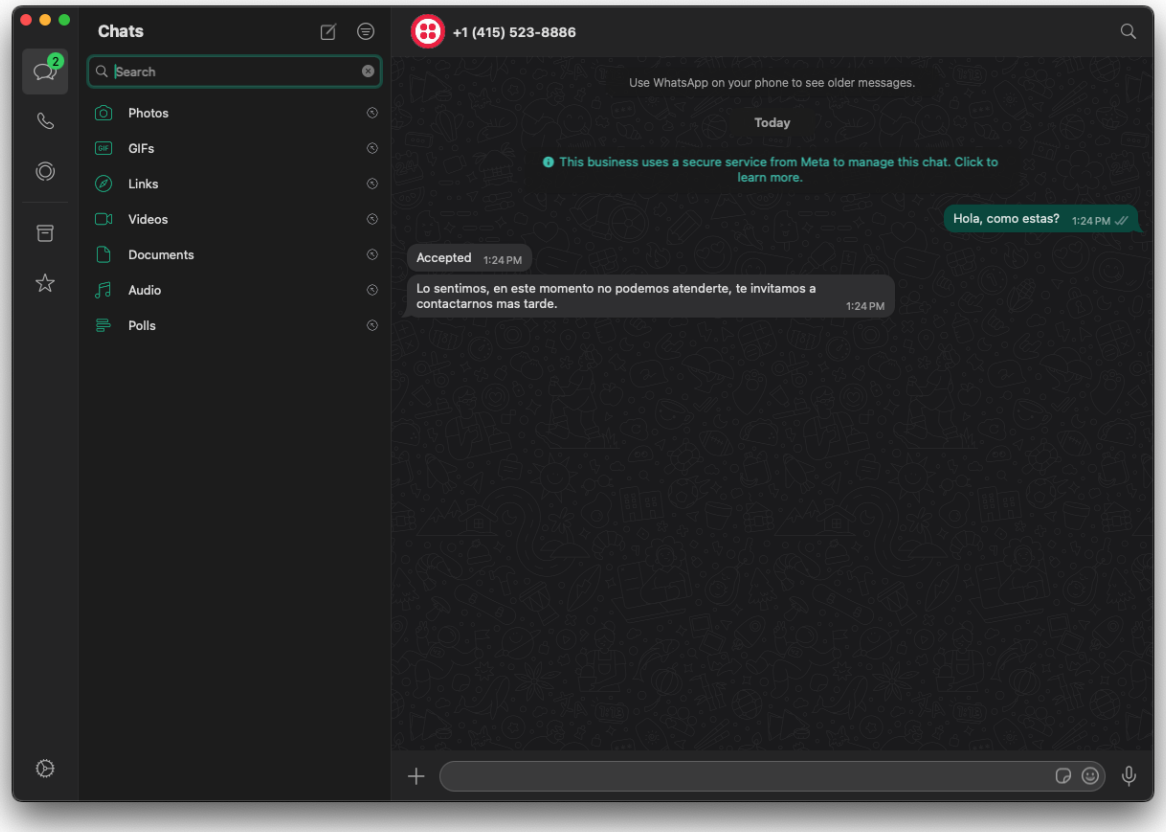


Figura C.5: Resultado de la ejecución del caso de prueba descrito en la Tabla C.3

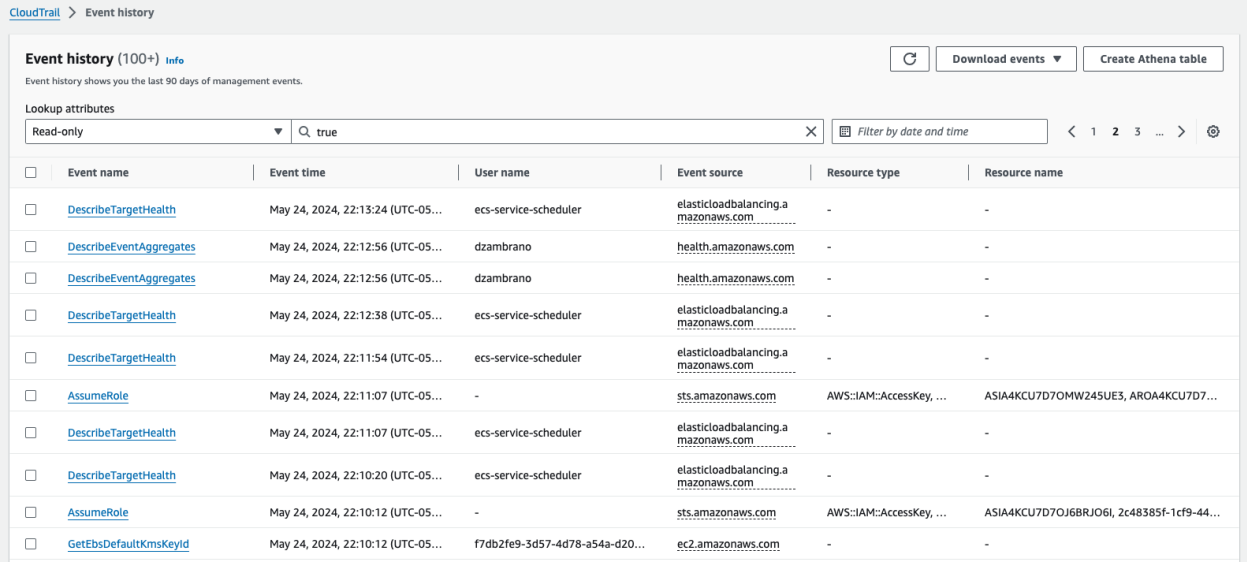


Figura C.6: Resultado de la ejecución del caso de prueba descrito en la Tabla C.4

C.2. Pruebas Iteración Compatibilidad

Esta sección del apéndice incluye los casos de prueba planteados sobre los drivers de arquitectura refinados en la iteración número tres de la metodología ADD.

Título	CP-COMP-02 El trabajador responde a las preguntas relacionadas con un producto
Escenario Relacionado	ver Figura 4.3
Descripción	Un usuario inicia una conversación con un usuario y realiza preguntas relacionadas con el producto denominado crédito de libre destino.
Resultado esperado	El trabajador digital inteligente responde el 100 % de las preguntas relacionadas con el tema producto libre destino.
Ejecución de la prueba	ver Figura C.7
¿Cumple con el resultado esperado?	Cumple

Tabla C.5: Caso de prueba: El trabajador responde a las preguntas relacionadas con un producto.

Título	CP-COMP-03 Orquestación del flujo conversacional
Escenario Relacionado	ver Figura A.8 y A.11.
Descripción	Asegurar que todos los componentes e integraciones se encuentren orquestados.
Resultado esperado	Orquestación de los flujos conversacionales con el modelo de procesamiento de lenguaje natural y con el flujo de integración con la API de WhatsApp.
Ejecución de la prueba	ver Figura C.8 y C.9
¿Cumple con el resultado esperado?	Cumple

Tabla C.6: Caso de prueba: Orquestación del flujo conversacional.

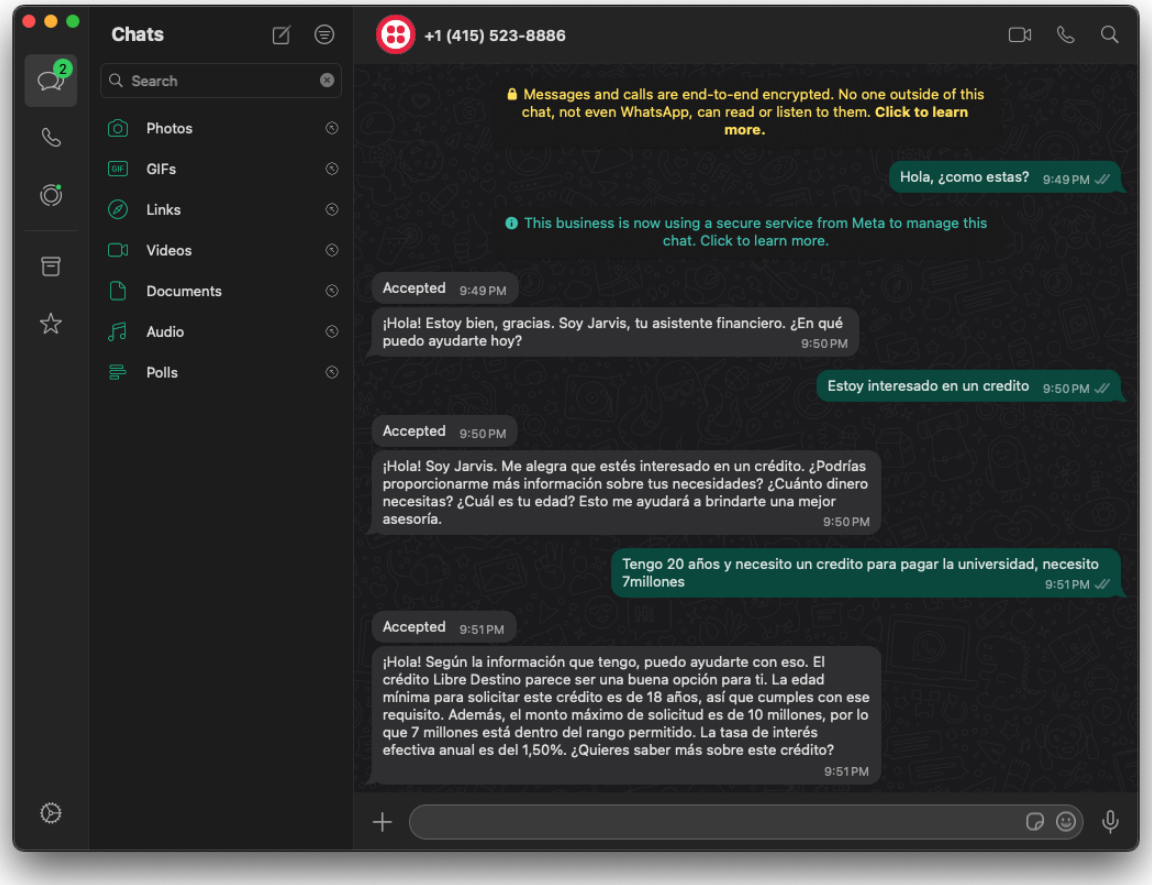


Figura C.7: Resultado de la ejecución del caso de prueba descrito en la Tabla C.5

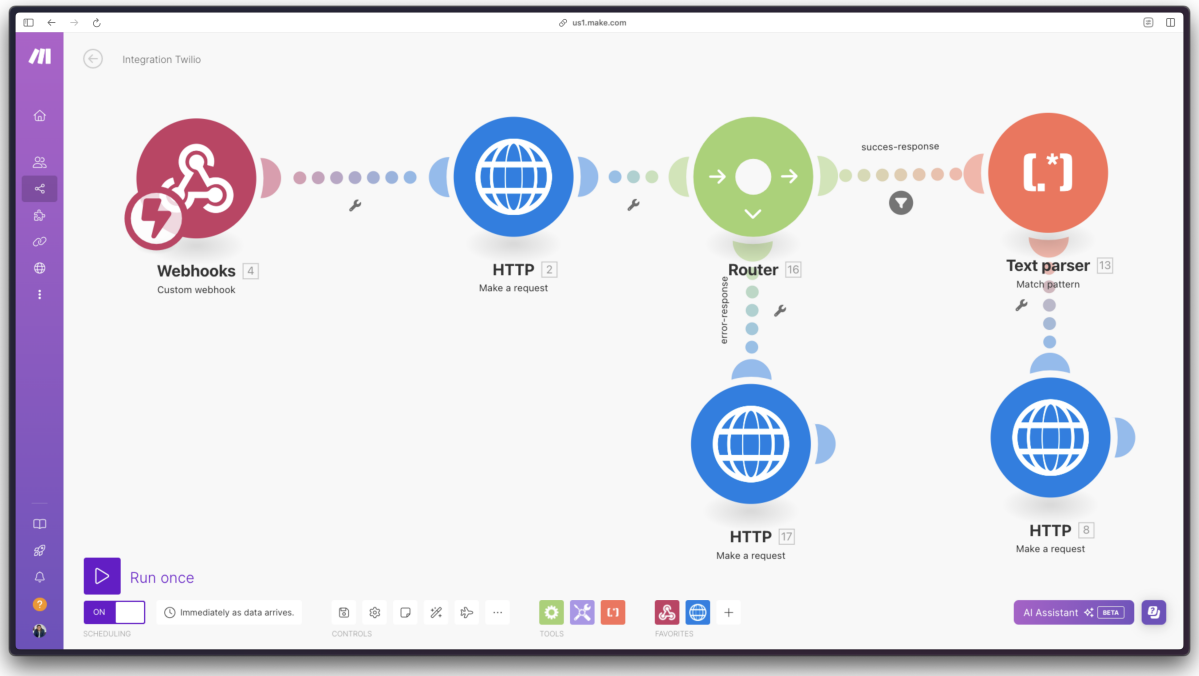


Figura C.8: Resultado de la ejecución del caso de prueba descrito en la Tabla C.6 - Flujo de integración con la API de WhatsApp.

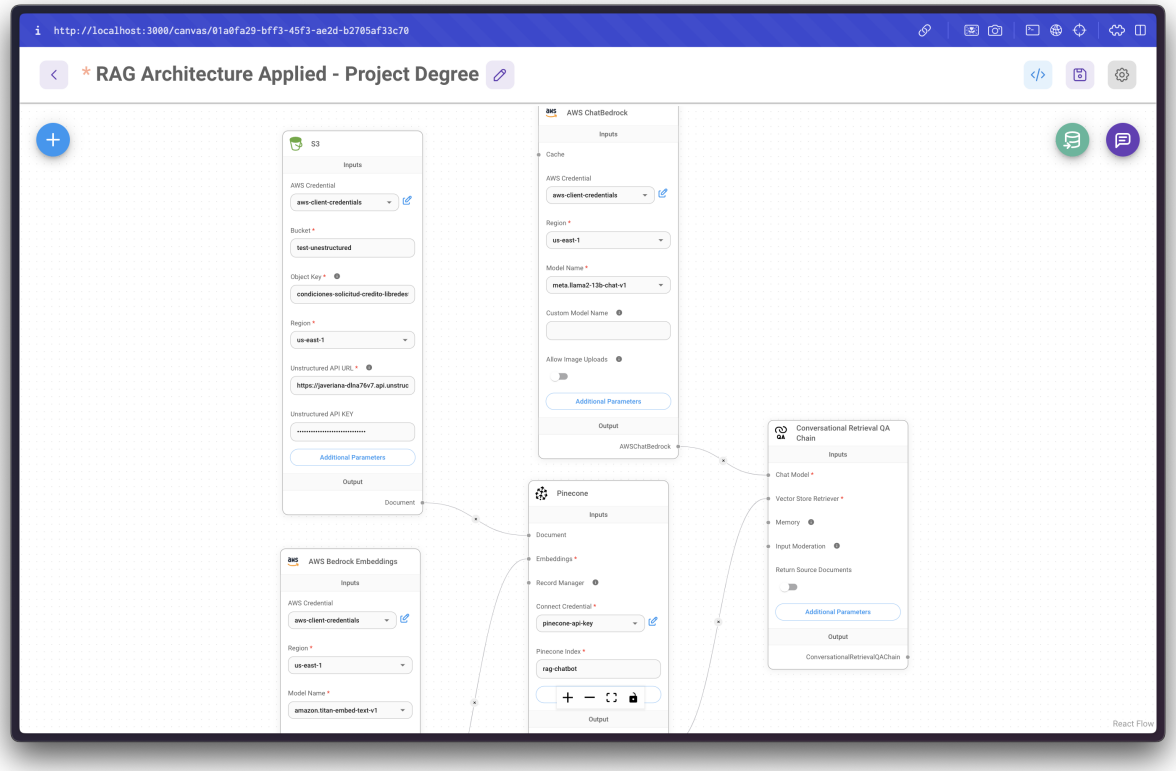


Figura C.9: Resultado de la ejecución del caso de prueba descrito en la Tabla C.6 - Flujo conversacional orquestado en la plataforma Flowise AI para interactuar con el modelo de procesamiento de lenguaje natural.