UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

GENERACIÓN DE ENLACES DE PAGO UTILIZANDO UN CHATBOT EN WHATSAPP: INTELIGENCIA ARTIFICIAL APLICADA EN LA INDUSTRIA DE TELECOMUNICACIONES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

ALEJANDRO ANTONIO VERAGUA ALBORNOZ

PROFESOR GUÍA:
MANUEL VILLARROEL ROMERO

PROFESOR CO-GUÍA:
DAVID VALENZUELA URRUTIA

COMISIÓN:
FRANCISCO RIVERA SERRANO

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: **ALEJANDRO ANTONIO VERAGUA ALBORNOZ**
FECHA: 2023
PROF. GUÍA: MANUEL VILLARROEL ROMERO

GENERACIÓN DE ENLACES DE PAGO UTILIZANDO UN CHATBOT EN WHATSAPP: INTELIGENCIA ARTIFICIAL APLICADA EN LA INDUSTRIA DE TELECOMUNICACIONES

El presente proyecto de titulación se enfoca en mejorar el *chatbot* existente en WhatsApp de una empresa de telecomunicaciones con el fin de incrementar la recaudación financiera y mejorar la experiencia del cliente. Se han identificado dos casos de uso principales: el pago de boletas de servicios y las recargas de saldos, ambos mediante la generación de enlaces de pago en línea.

Los objetivos del proyecto comprenden el diseño e implementación de opciones de pago a través del *chatbot*, específicamente para los casos de uso mencionados. Para lograrlo, se llevó a cabo la automatización del proceso de generación de enlaces de pago, lo cual facilitó a los clientes realizar pagos de manera rápida y segura. Esta solución busca mejorar la experiencia del cliente, optimizar los procesos de pago y generar ingresos adicionales para la empresa.

Se han establecido restricciones clave para el desarrollo del proyecto, entre las cuales se encuentra el uso de WhatsApp como canal de *chat*, la adopción de Watson Assistant como solución de Inteligencia Artificial conversacional, la selección de Amazon Web Services (AWS) como plataforma de computación en la nube y la utilización de las plataformas definidas por la empresa para obtener información de los clientes y realizar transacciones de pago.

La solución propuesta se basa en aprovechar las capacidades de AWS *serverless* y Watson Assistant. Mediante la arquitectura *serverless* de AWS, el *chatbot* podrá ejecutarse de manera escalable y eficiente, adaptándose a la demanda de los usuarios sin necesidad de redefinir la infraestructura subyacente. Por su parte, Watson Assistant proporcionará una experiencia de conversación inteligente y contextualizada para los clientes.

La implementación de este avance tecnológico en el canal ha permitido que un 13,35 % y 19,39 % de los enlaces de pago generados se cursen como un pago efectivo, con una tasa de fallos del 3,41 % y 2,89 %, para recargas de saldo y pago de boleta de servicios, respectivamente. Por otro lado, con este trabajo también se ha podido caracterizar el funcionamiento del sistema en cuanto a los tiempos de respuesta (RTT), confiabilidad y estimación de costos.

En conclusión, la mejora del *chatbot* en WhatsApp a través de la generación de enlaces de pago para recargas de saldo y pago de boletas de servicios, logró ofrecer una experiencia mejorada al cliente y permitir a la empresa aumentar su recaudación financiera y contribuyendo en mejorar los servicios de la empresa.

Agradecimientos

Deseo expresar mi más sincero agradecimiento a todos los que han sido un sólido pilar a lo largo de mi trayectoria: mi amada familia, mis entrañables amigos y mis dedicados profesores.

En este momento tan especial, quiero extender mi luminosa gratitud a la Universidad de Chile y a la Facultad de Ciencias Físicas y Matemáticas por abrirme sus puertas a través de programas inclusivos. Su generosidad y confianza al permitirme ingresar mediante el Sistema de Ingreso Prioritario de Equidad Educativa (SIPEE) han sido un regalo invaluable. Además, no puedo dejar de mencionar el incansable respaldo que me han brindado a lo largo de mi recorrido, tanto en el ámbito académico como personal. Sin su constante apoyo y aliento, este magnífico logro nunca habría sido posible. Tengo la certeza de que el constante trabajo que realizan a diario en sus programas seguirá cambiando la realidad de muchas familias.

Deseo dedicar un momento cargado de amor a mi querida familia. A ti, madre mía, eres la fuerza que siempre ha creído en mí y me ha sostenido desde mis primeros pasos. A mis amados hermanos y hermana, su compañía ha sido un regalo durante toda mi vida.

Quiero expresar mi profunda gratitud a Manuel Villarroel. Desde lo más hondo de mi ser, agradezco tu impulso para llevar a cabo este trabajo. A David Valenzuela, mi gratitud rebosa de alegría por compartir tu sabiduría en esta etapa tan trascendental.

Envío un cálido abrazo a los profesores de la Facultad de Ciencias Físicas y Matemáticas, quienes han sido mis inspiradores y mentores. Al profesor Felipe Tobar, mi gratitud y admiración, ya que encendiste la chispa de mi interés por el aprendizaje del *Machine Learning*. Al profesor Marchos Orchard, mi corazón se llena de alegría al recordar los fundamentos del modelamiento de sistemas que me transmitiste con paciencia y sabiduría. Y al profesor Marcelo Matus, mi gratitud, pues gracias a ti surgió en mí la curiosidad insaciable por la ingeniería de software.

Una vez más, quiero expresar mi profundo y sincero agradecimiento a todos aquellos que han contribuido a mi desarrollo académico y personal. Su amor, apoyo y amistad han sido las raíces que han nutrido mi camino hacia el éxito. Me siento muy agradecido por todo lo que han hecho por mí: Felipe Gutierrez, Matias Castro, Pedro Pablo Jofré, Raúl Quintana, Alexy Ángulo, Nicolás Romero, Manuel Sachez, Juan Daniel Tolosa, Fabián Camacho, David Arancibia, Francisco Díaz, Fabian Araya y Rodrigo Seguel.

Con amor e inmensa alegría.

Tabla de Contenido

Índice de Ilustraciones

Índice de Instrucciones

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

La internet ha trascendido desde lo físico a lo digital, llegando a ser parte fundamental del mundo, alcanzando casi 5.000 millones de usuarios [1]. Las empresas de telecomunicaciones juegan un rol fundamental en este tema porque proveen de conectividad móvil y fija, permitiendo conectar dispositivos móviles y proveer de internet a hogares, oficinas y espacios públicos.

En Chile, para fines de Junio del 2022, el sector de las telecomunicaciones alcanzó 59 millones de servicios [2], entre ellos: Televisión Pagada, Internet Móvil, Internet Fijo, Voz Móvil y Voz Fija, los cuales se realizan mediante la contratación de servicios. Esto requiere de esfuerzos para mantener los contratos activos, en cumplimiento con los derechos de los usuarios [3], regulados por la Subsecretaria de Telecomunicaciones (SUBTEL). Por reglamento, las empresas del rubro deben dar asistencia a los clientes para conocer los precios y coberturas; contratar servicios individuales, conocer el contrato, reclamar, entre otros. Todo lo cual se conoce como posventa.

La automatización es la aplicación de máquinas o de procedimientos automáticos en la realización de un proceso, y ha permitido generar eficiencias en procesos industriales y comerciales. Por ello, ha demostrado ser un camino factible para permitir a los usuarios realizar sus consultas de manera autónoma (vale decir, sin la necesidad de asistencia humana). Por esta razón existen canales de atención telefónicos, páginas web y aplicaciones móviles, que permiten a los clientes gestionar sus dudas, recibir asistencia técnica o comercial, ingresar reclamos, entre otros.

La Inteligencia Artificial (AI) es toda máquina con la capacidad de lograr rendimiento a nivel humano en las tareas cognitivas suficientes para engañar a un interrogador [4]. La aplicación de Inteligencia Artificial en conversaciones entre humano y computadora, ha derivado en la implementación de dos canales de atención que permiten aprovechar la automatización en un lenguaje natural para los clientes: los asistentes virtuales de voz (*voicebot*) y los asistentes virtuales de *chat* (*chatbot*). Algunos de los asistentes virtuales más conocidos son: Google Assistant, Siri, Cortana y Alexa [5].

Tabla 1.1: *Chatbots* en el canal de WhatsApp encontrados para la industria de telecomuicaciones en Chile, 2023.

Empresa	***Chatbot***	**Nro. WhatsApp**
Wom	Wom Chile	+(56) 9 3522 3070
Wom	Wom Chile Televenta	+(56) 9 3773 3992
Wom	Wom Chile Fibra	+(56) 9 3735 2538
Entel	Entel Posventa	+(56) 9 3407 2243
Entel	Entel Ventas	+(56) 9 756 30928
Entel	Entel Empresas	+(56) 9 3413 6117
Movistar	Movistar Chile	+(56) 9 4704 4226
Claro	Claro Posventa	+(56) 9 9000 0171

En la dirección de los canales de *chat*, WhatsApp permite mantenerse en contacto con amigos y familiares en cualquier momento y lugar, conectando a más de 2.000 millones de personas en más de 180 países [6]. Es gratuito y permite enviar mensajes y hacer llamadas de manera simple, segura y confiable en teléfonos de todo el mundo. Esta aplicación se lanzó al público general en 2009, fue adquirida en 2014 por Meta Platforms (ex Facebook) y su crecimiento, desde entonces, ha tenido gran impacto mundial dentro de las telecomunicaciones.

En Chile, según el estudio *WhatsApp, un canal clave en las compras del retail en Chile*, de la firma Accenture [7], un 97 % de los encuestados utiliza el canal de manera frecuente, un 83 % ya lo utiliza como soporte de compra en linea, junto a 49 % que estaría dispuesto a utilizarlo en el corto plazo. Todo ello, en contraposición a un solo 7 % que declara sentirse inseguro respecto a su uso.

El presente Trabajo de Titulo está motivado por contribuir con la empresa de telecomunicaciones, en el ámbito de sus procesos de recaudación. Esta es una tarea importante en la posventa, puesto que se relaciona directamente con el ingreso de capital, permitiendo a los clientes generar por si mismos enlaces de pago para sus servicios, combinando automatización, inteligencia artificial y computación en la nube.

Actualmente, se pueden encontrar 8 *chatbots* en WhatsApp para las principales empresas de telecomunicaciones de Chile: Wom, Entel, Movistar y Claro. Los números de contacto se pueden ver en la tabla 1.1.

1.2. Antecedentes

La compañía de telecomunicaciones posee un asistente virtual de atención masiva para posventa, con presencia en el canal de WhatsApp. Este fue lanzado en 2018 y, desde entonces, ha crecido como canal de atención, llegando a atender a más de medio millón de usuarios mensuales con poco más de mil ejecutivos comerciales. En este canal, los clientes pueden

escribir para solicitar atenciones de forma autónoma, automática y remota, tales como: información de la boleta, detalle de planes, consumo de planes, resolver problemas de internet y señal, transferencias a ejecutivos comerciales, entre otras.

Un proceso comercial se refiere a una serie de actividades y etapas secuenciales diseñadas para alcanzar un objetivo comercial específico, en el contexto de la posventa, es un conjunto de actividades que la empresa determina para atender las consultas de los clientes sobre cierto tema, las cuales pueden ser realizadas de manera automática o bien, por un ejecutivo comercial especialista en el tema. De esta manera, los mensajes que los usuarios envían por WhatsApp son analizados por el *chatbot* para determinar y ejecutar de manera automática el proceso comercial, buscando responder a la preguntas específicas del usuario.

Se puede clasificar a los clientes de acuerdo al tipo de servicio que contratan: prepago y pospago. Los primeros deben realizar el pago anticipado para poder llamar y navegar en internet, lo cual se conoce como recarga de saldo. En los pospago, en cambio, la empresa aprovisiona el servicio móvil u hogar por adelantado y en determinada fecha emite una factura o boleta que debe ser pagada por el cliente antes de la fecha de vencimiento. Esto es análogo a los modelos de pago por suscripción, en donde, cada cierto tiempo el cliente debe pagar para continuar utilizando los servicios [1].

Los mensajes de texto enviados por los usuarios a través de WhatsApp son dirigidos hacia el sistema interno del *chatbot*, el cual se encarga de procesar dichos mensajes. Cada mensaje es enviado a un motor de inteligencia artificial capaz de comprender y transformar el texto en un formato legible por la máquina. En este caso, se utiliza Watson Assistant [8], una solución de inteligencia artificial conversacional proporcionada por IBM. Esta tecnología permite abordar las frases de los clientes y determinar las acciones a tomar. Al finalizar el procesamiento de un mensaje por parte de Watson Assistant, se obtiene una de dos posibles respuestas: una respuesta dirigida al usuario o una respuesta destinada al sistema (considerando las instrucciones para ejecutar una acción automática).

Sin embargo, Watson Assistant no ofrece soporte nativo para la integración con dos plataformas cruciales para el *chatbot*: WhatsApp y los sistemas propios de la empresa. Por lo tanto, se requiere diseñar y desarrollar un sistema que permita la comunicación entre estas dos plataformas. Esto puede lograrse mediante el uso de servicios de computación en la nube.

Existen varias opciones de nubes disponibles, como Amazon Web Services (AWS) [9], Google Cloud Platform [10] y Microsoft Azure [11]. La empresa de telecomunicaciones ha seleccionado AWS como la plataforma para implementar todas las funcionalidades del *chatbot*.

En cuanto a la comunicación con WhatsApp, se utiliza infraestructura como servicio en forma de servidores y bases de datos dentro de AWS, para manejar los mensajes y responder a los clientes con texto, botones, multimedia y otros elementos interactivos. De manera similar, se emplean servicios específicos de AWS para orquestar la consulta de información de los usuarios en los sistemas internos de la empresa, solo cuando Watson Assistant responde al sistema con una acción automática.

[1] Spotify, Netflix y Youtube, son ejemplos de empresas que utilizan un modelo de pago por suscripción.

1.3. Descripción del Problema

El *chatbot* en WhatsApp no contaba con opciones de pago habilitadas (ver figuras 1.1 y 1.2). Al permitir a los clientes realizar pagos a través del *chatbot* en WhatsApp, se habilitó un nuevo canal de recaudación para el pago de servicios por parte de los clientes. Esto potenció significativamente el canal de WhatsApp, como una plataforma adicional para realizar transacciones financieras y mejorar la experiencia del cliente, siendo su implementación beneficioso para la empresa de telecomunicaciones.

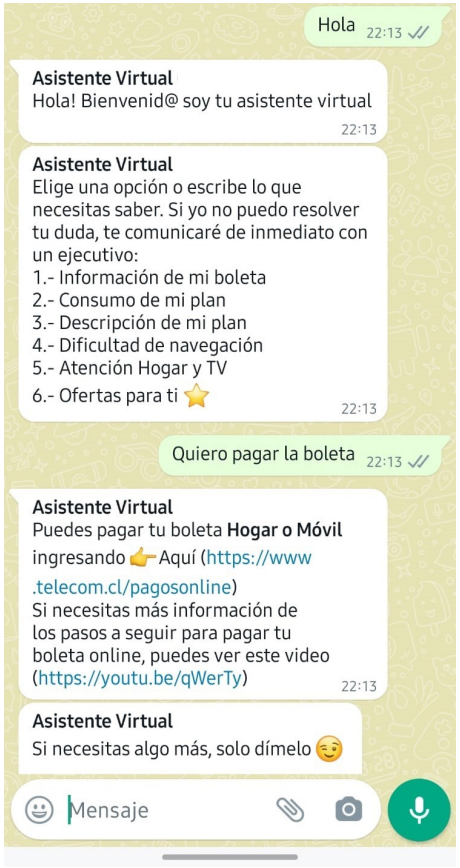

Figura 1.1: Respuesta del asistente virtual para una intención de pago de boletas, antes de la implementación.

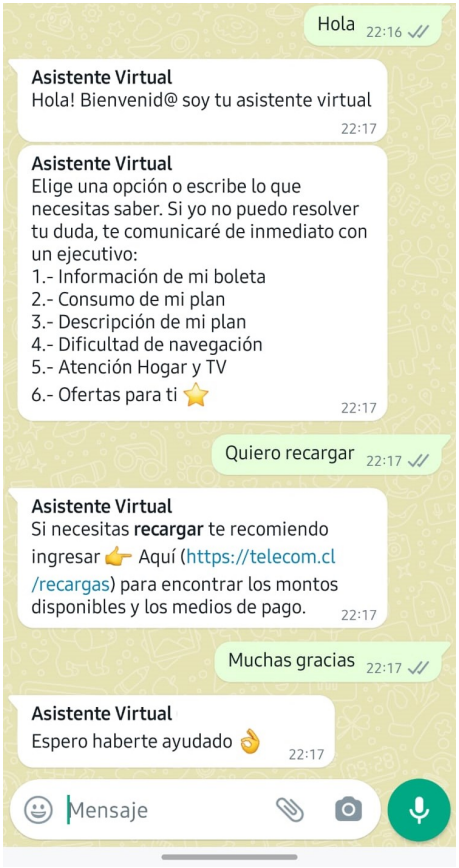

Figura 1.2: Respuesta del asistente virtual para una intención de pago de boletas, antes de la implementación.

Existen dos casos de uso específicos para la integración de enlaces de pago. En primer lugar, se encuentra el pago de la boleta de servicios móvil y hogar (ver sección 3.4), y en segundo lugar, las recargas de saldo con métodos de pago digital (ver sección 3.5).

Restricciones

La figura 1.3 presenta una visión simplificada del funcionamiento del *chatbot*. En la capa superior, se encuentra WhatsApp, que sirve como interfaz principal para la interacción con los clientes. Por otro lado, AWS actúa como una capa intermedia que conecta a Watson

Assistant con los demás sistemas empresariales de la compañía, facilitando la comunicación y la integración de datos.

Considerando diversos factores, como la adquisición de licencias, los conocimientos del equipo de trabajo y desarrolladores, así como la arquitectura actual del sistema, se ha determinado que es recomendable incorporar la solución en la arquitectura existente para el *chatbot*, lo cual implica utilizar las siguientes plataformas y tecnologías:

1. WhatsApp ha demostrado, como canal de *chat*, ser una plataforma de mensajería muy popular y ampliamente adoptada por los usuarios de la empresa. Utilizar este canal de *chat* proporciona una interfaz familiar y conveniente para los clientes, lo que facilita la interacción y la adopción del *chatbot*.
2. Watson Assistant como Inteligencia Artificial Conversacional, el equipo de trabajo ya cuenta con experiencia en el uso de esta plataforma; por cuanto, mantenerla proporcionará una transición más fluida y permitirá aprovechar el conocimiento y la infraestructura existente.
3. Amazon Web Services (AWS) como plataforma de computación en la nube. La arquitectura actual del sistema se basa en AWS y los equipos de desarrollo están familiarizados con esta plataforma. Mantenerla resulta conveniente y compatible con la infraestructura existente, optimizando la escalabilidad, seguridad y flexibilidad necesarias para el funcionamiento del *chatbot*.
4. Utilizar los sistemas definidos por la empresa para obtener información de clientes, es crucial utilizar las plataformas definidas por la empresa para acceder a la información de los clientes y llevar a cabo transacciones de pago. Esto garantiza la coherencia, la seguridad y el cumplimiento normativo, al tiempo que se aprovechan los sistemas y los procesos establecidos por la organización.

Figura 1.3: Diagrama de alto nivel del *chatbot* en el canal de WhatsApp.

Al mantener estas restricciones, se aprovechan los conocimientos existentes, se optimiza la compatibilidad con la arquitectura actual y se fomenta la continuidad de los sistemas y procesos ya implementados. Además, se asegura una experiencia de *chatbot* coherente y segura para los usuarios, minimizando los desafíos asociados con la adquisición de nuevas licencias o la adopción de plataformas desconocidas. Asimismo, se garantiza la integración adecuada con los sistemas existentes y se cumplen los requisitos de seguridad y cumplimiento normativo de la empresa.

1.4. Objetivos

1.4.1. Objetivo General

El objetivo general de este proyecto es diseñar e implementar opciones en el *chatbot* de WhatsApp que permitan a los clientes de la empresa de telecomunicaciones solicitar enlaces de pago para recargas de saldos y pago de boletas de servicios. Estas opciones deben ser desarrolladas teniendo en cuenta las restricciones establecidas en la sección 1.3, que incluyen el uso de WhatsApp como canal de *chat* (*frontend* [2]); la utilización de Watson Assistant como solución de Inteligencia Artificial, la elección de Amazon Web Services (AWS) y la integración con los sistemas internos de la empresa (*backend* [3]).

1.4.2. Objetivos Específicos

1. Diseñar e implementar un proceso automatizado para la creación de enlaces de pago destinados a recargas de saldo. Estos enlaces estarán habilitados para el pago electrónico o digital y se integrarán en el canal de WhatsApp.
2. Diseñar e implementar un proceso automatizado para la creación de enlaces de pago destinados al pago de boletas de servicio móvil y hogar. Estos enlaces se generarán de manera automática y estarán disponibles en el canal de WhatsApp.

1.5. Estructura de la Memoria

A continuación se resumen los capítulos para la presente Memoria de Titulo:

- Capítulo 1 - Introducción 1: en este capítulo se proporcionan los antecedentes relevantes sobre la industria de las telecomunicaciones y su vinculación con los *chatbot* en Chile. Asimismo, se plantea el problema a abordar, junto con sus restricciones.
- Capítulo 2 - Marco teórico y Estado del Arte 2: este capítulo abarca conceptos fundamentales sobre arquitectura de sistemas, enfocándose en las definiciones de *Round Trip*

[2]Interfaz Web o Mobile, con la cual interactúan los usuarios de WhatsApp.
[3]Infraestructura tecnológica detrás de aplicaciones, gestionando bases de datos, lógica y comunicaciones.

Time (RTT) y confiabilidad. Posteriormente, se expone el estado del arte en cuanto a arquitecturas sin servidor, computación en la nube, inteligencia artificial conversacional y la solución WhatsApp Business Solution.

- Capítulo 3 - Diseño e Implementación de la Solución Tecnológica 3: en este capítulo se detalla el diseño de la solución, destacando los diagramas que definen el flujo conversacional; los diagramas de componentes que identifican las tecnologías utilizadas en cada componente y su relación, y los diagramas de secuencia que especifican el orden de integración, las condiciones y las API a proveer o consumir. Asimismo, se presentan los resultados de la implementación mediante pruebas reales.
- Capítulo 4 - Análisis y Resultados 3: en este capítulo se ofrece un análisis detallado de las mediciones realizadas para el RTT, la confiabilidad teórica y la estimación de costos en un escenario en el cual un millón de usuarios emplean el servicio de pago de boletas de servicios o recargas de saldo.
- Capítulo 5 - Conclusiones y Trabajos Futuros 5: este capítulo recoge las conclusiones obtenidas a partir del trabajo desarrollado, así como las propuestas de mejoras en términos de tiempos de respuesta (RTT), confiabilidad y reducción de costos.

Capítulo 2

Marco Teórico y Estado del Arte

2.1. Conceptos Generales sobre Arquitectura de Sistemas

La arquitectura de sistemas en tecnologías de la información, es una disciplina que se enfoca en diseñar y estructurar sistemas informáticos complejos, con el objetivo de asegurar que se cumplan los requerimientos tanto del negocio como de los usuarios, de manera eficiente y efectiva. Esto implica la definición de los componentes del mismo sistema, sus interacciones y las reglas de integración entre ellos. Además, incluye la identificación y selección de las tecnologías adecuadas para la implementación, así como la definición de las políticas y estándares de seguridad y operación [12] [13] [14].

2.1.1. Infraestructura TI

La infraestructura de sistemas se refiere a todos los componentes físicos y virtuales que componen un sistema de tecnología de la información. Esto incluye servidores, redes, dispositivos de almacenamiento, *softwares* de gestión y otros recursos necesarios para el funcionamiento de aplicaciones y sistemas empresariales. Es esencial para el funcionamiento de la tecnología de la información de una organización, ya que proporciona los medios para almacenar, procesar y transmitir datos y aplicaciones. Además, garantiza la disponibilidad, escalabilidad y seguridad de los sistemas empresariales. Para llevarlo a cabo, se utilizan herramientas y tecnologías de gestión de infraestructura, tales como la monitorización de sistemas, la virtualización de servidores, la gestión de redes y la gestión de seguridad de la información.

2.1.2. Cliente y Servidor

Cliente-servidor es un modelo de diseño de sistemas informáticos en el que una parte del *software* llamado "cliente", se ejecuta en relación a un equipo local y se comunica con otra parte del *software*, llamada "servidor"; la cual, a su vez, se ejecuta en un equipo remoto. El cliente-servidor se basa en el concepto de división de tareas entre dos componentes: Por un

lado, el cliente que solicita recursos o servicios al servidor y, por otro, el servidor que procesa las solicitudes y responde al cliente proporcionando los recursos o servicios solicitados.

El paradigma arquitectónico cliente-servidor, comúnmente referido como *frontend* y *backend*, en la jerga técnica, establece que la interacción del usuario ocurre a través del *frontend*.

En el contexto de esta investigación, dicho *frontend* se identifica con la aplicación de mensajería WhatsApp, la cual se encuentra instalada en el dispositivo móvil del usuario. En contraste, el *backend* constituye la infraestructura subyacente, responsable de la implementación de los procesos en este trabajo.

2.1.3. Protocolo HTTP

HTTP *(Hypertext Transfer Protocol)*[15] es un protocolo de aplicación utilizado para la comunicación en la World Wide Web. Define la forma en que los clientes (como los navegadores web) solicitan recursos, como páginas web, a servidores web, y cómo estos servidores responden a esas solicitudes. HTTP se basa en un modelo cliente-servidor sin estado, lo que significa que cada solicitud se procesa de forma independiente, sin recordar las solicitudes anteriores.

HTTPS (HTTP Secure) es una extensión del protocolo HTTP que agrega una capa de seguridad mediante el uso de un protocolo de cifrado, como SSL (Secure Sockets Layer) o su sucesor, TLS (Transport Layer Security). HTTPS utiliza un certificado digital para establecer una conexión cifrada entre el cliente y el servidor web, protegiendo así la confidencialidad e integridad de los datos transmitidos. Esto proporciona una mayor seguridad y privacidad al navegar por sitios web, evitando que los datos sean interceptados o modificados por terceros malintencionados.

2.1.4. Separación de Ambientes

La separación de ambientes de desarrollo y producción es una práctica común en el desarrollo de *software*; lo cual busca mantener un entorno de prueba y experimentación, separado del entorno de producción en el que se ejecutan las aplicaciones en vivo. Aunque, de acuerdo a las necesidades, ciertas arquitecturas podrían ser definidas por mas de dos ambientes. Esta separación garantiza que los cambios y actualizaciones realizados en el ambiente de desarrollo se prueben adecuadamente antes de ser implementados en el entorno de producción, minimizando así el impacto de errores y fallas.

En el contexto de este proyecto, implica tener dos entornos separados: uno dedicado al desarrollo y otro al entorno de producción. El ambiente de desarrollo se utiliza para probar nuevas funcionalidades, realizar pruebas y depurar errores. El ambiente de producción, es donde se ejecuta el *chatbot* en vivo, atendiendo a los usuarios y proporcionando respuestas reales.

2.1.5. Confiabilidad

La confiabilidad de un dispositivo se refiere a la probabilidad de que funcione correctamente a lo largo del tiempo dentro de su entorno operativo (producción), tal como se describe en la referencia bibliográfica de Nachlas et al. (2017) sobre confiabilidad [16]. Esta medida de confiabilidad se utiliza para evaluar la fiabilidad y el tiempo en el que un servicio opera sin interrupciones ni fallos. La confiabilidad se expresa típicamente como un porcentaje y también se puede interpretar como la probabilidad de que ocurra una falla. Ambas representaciones indican la proporción de tiempo en que el servicio está activo y disponible en comparación con el tiempo total de referencia. Por ejemplo, una disponibilidad del 99.9 % indica que el servicio está disponible el 99.9 % del tiempo, mientras que el 0.1 % restante corresponde al tiempo de inactividad o falla.

Cuando se diseña un sistema, es importante considerar la confiabilidad de cada componente individual ($\mathbb{R}_{\mathrm{i}}$) y la configuración del sistema, ya sea en serie o en paralelo.

En el caso de un sistema en serie, la confiabilidad del sistema se calcula como el producto de las confiabilidades de los componentes individuales, como se muestra en la ecuación 2.1. Esta configuración implica que la confiabilidad del sistema depende de la confiabilidad de cada componente $\mathbb{R}_{\mathrm{i}}$, y se suele decir que el sistema es tan fuerte como el eslabón más débil.

Por otro lado, en un sistema en paralelo, la confiabilidad del sistema se calcula considerando el complemento de las confiabilidades de los componentes individuales, como se muestra en la ecuación 2.2. En esta configuración, la confiabilidad del sistema se explica por la probabilidad de que no ocurra una falla en ninguno de los componentes. Así, un sistema puede aumentar su confiabilidad agregando redundancias en paralelo.

Para calcular la confiabilidad de un servicio en la nube, los proveedores, como AWS, suelen ofrecer "Acuerdos de Nivel de Servicio" (SLA[1]) que especifican la disponibilidad mínima garantizada para sus servicios.

$$\mathbb{R}_{serie} = \prod_{\mathrm{i}=1}^{N} \mathbb{R}_{\mathrm{i}} \tag{2.1}$$

$$\mathbb{R}_{paralelo} = 1 - \prod_{\mathrm{i}=1}^{N} (1 - \mathbb{R}_{\mathrm{i}}) \tag{2.2}$$

2.1.6. Round Trip Time (RTT)

El *Round Trip Time (RTT)* hace referencia al tiempo transcurrido desde que un sistema cliente envía una solicitud, hasta que se obtiene una respuesta del servidor. Este tiempo representa la demora experimentada debido a la comunicación entre diferentes componentes de una infraestructura, como servidores, redes y dispositivos de almacenamiento. El RTT

[1]Service Level Agreement

puede verse afectado por diversos factores, como la distancia física entre los puntos de origen y destino, la capacidad de procesamiento de los servidores, la congestión de la red y el tipo de protocolo de comunicación utilizado [17].

Este concepto en general se mide y aplica mediante *ping* (ICMP), que indica en capa de red el tiempo transcurrido en "'ida y vuelta"; vale decir, entre el cliente y el servidor. No obstante, y debido a que el *ping* se mide a nivel de red, no contempla el tiempo incurrido en otras capas, como transporte (TCP) y aplicación (HTTP). De esta manera, en el contexto de las comunicaciones HTTP, se puede utilizar el concepto de RTT para medir el tiempo transcurrido en procesar completamente las solicitudes entre el cliente y servidor. El estudio del *RTT* se torna relevante para indicar aquellos elementos con mayor RTT y, en consecuencia identificar mejoras y optimizaciones al sistema.

Cuando un cliente solicita una petición HTTP, K componentes son ejecutados en serie. Así, es posible modelar el tiempo de respuesta total de una petición i, sumando las contribuciones individuales de cada uno de los subcomponentes $j \in \{0, K\}$ involucrados en la petición, como se puede ver en la ecuación 2.3. Cabe destacar que, como existen diversos componentes involucrados en la misma petición, el RTT también se puede medir en los componentes intermedios.

$$RTT_{\mathrm{i}} = \sum_{j=1}^{K} RTT_j \tag{2.3}$$

2.2. Estado del Arte

2.2.1. WhatsApp Business Solutions

WhatsApp es una aplicación de mensajería instantánea y llamadas de voz y video que se utiliza ampliamente en todo el mundo. Fue desarrollada originalmente en 2009 por Brian Acton y Jan Koum, dos antiguos empleados de Yahoo. En 2014, WhatsApp fue adquirida por Facebook y, desde entonces ha experimentado un crecimiento masivo en popularidad.

Esta aplicación permite a los usuarios enviar mensajes de texto, imágenes, videos, audios, documentos y ubicaciones en tiempo real a través de una conexión a Internet. La aplicación está disponible para dispositivos móviles (iOS, Android, Windows Phone) y también puede ser utilizada en computadoras a través de la versión web o de escritorio.

Existe una versión específica de WhatsApp para empresas, llamada WhatsApp Business, que ofrece características adicionales para empresas y emprendedores, como perfiles comerciales, respuestas automáticas y estadísticas de mensajes. En cambio, WhatsApp Business API [2] [18] es una solución más avanzada y escalable, ofrecida por WhatsApp para empresas que desean integrar WhatsApp directamente en sus sistemas. A diferencia de la versión estándar de WhatsApp Business, que se ejecuta en dispositivos móviles, la API permite el

[2]Interfaz de Programación de Aplicaciones, por sus siglas en inglés.

intercambio de mensajes entre aplicaciones. Esta API ofrece a las empresas la capacidad de enviar notificaciones, alertas, actualizaciones de servicios y mensajes transaccionales a sus clientes de manera automatizada y personalizada. También permite a las empresas recibir mensajes de los usuarios y responder a ellos de forma programática.

Para utilizarla, las empresas deben cumplir con ciertos requisitos y políticas establecidas por WhatsApp, como por ejemplo, contar con la aprobación de WhatsApp y tener una cuenta de negocio verificada por la misma. Además, las empresas deben cumplir con las leyes de protección de datos aplicables y garantizar la privacidad y seguridad de los datos del usuario.

Las empresas que quieren utilizar WhatsApp pueden convertirse en un partner de Facebook WhatsApp o contratar servicios profesionales a un BSP[3]. El estado del arte en WhatsApp Business API consiste de dos posibles conectores o "sabores":

1. WhatsApp On-Premise API [20] es una solución de WhatsApp Business que se debe instalar en infraestructura de la empresa. WhatsApp proporciona documentación e instalables para distintas nubes. Por ejemplo, se podría ejecutar la instalación en la plataforma Amazon Web Services, Google Cloud o Microsoft Azure. Al utilizar esta solución, las empresas tienen un mayor control sobre sus datos y pueden personalizar la solución de acuerdo a sus necesidades específicas. De esta forma, la solución On-Premise se ejecuta en una infraestructura en la nube, escalable y segura, lo cual garantiza la disponibilidad y confidencialidad de los datos. No obstante, para implementarla, se necesitan habilidades y conocimientos técnicos especializados, puesto que se trata de una solución personalizada que requiere de una configuración y mantenimiento cuidadosos. Además, se debe incurrir en el costo de infraestructura para servidores, bases de datos y balanceadores de carga.
2. WhatsApp Cloud API [21] es una solución de WhatsApp Business API que se ejecuta en la nube de WhatsApp, vale decir, no requiere de infraestructura de la empresa como en el caso de WhatsApp On-Premise API. En lugar de eso, se accede a la API a través de la nube de WhatsApp, lo que simplifica su implementación y mantenimiento. Esta API permite la comunicación bidireccional entre los usuarios y las aplicaciones a través de mensajes de WhatsApp, y ofrece una serie de características y herramientas para las empresas, como el envío de mensajes personalizados y automatizados a los clientes; la integración con sistemas empresariales y las herramientas de análisis, así como el seguimiento del rendimiento de las campañas.

2.2.2. Inteligencia Artificial

Fue el matemático Alan Turing, quien en 1950 planteó la pregunta "*¿Pueden las máquinas pensar?*", la persona a quien se suele considerar como la primera en concebir los *chatbots*.

[3]Business Solution Partner, por sus siglas en inglés, forman parte de una comunidad internacional de proveedores independientes de soluciones empresariales con experiencia en la plataforma de WhatsApp Business, y pueden ayudar a las empresas a comunicarse con sus clientes mediante la plataforma de WhatsApp Business en los casos prácticos aprobados, como lo son la entrega de notificaciones personalizadas y la prestación de servicios de atención al cliente [19]

Alan Turing define Inteligencia Artificial (IA) como la habilidad de una máquina para llevar a cabo tareas que, si fueran realizadas por seres humanos, requerirían de inteligencia. Es decir, se refiere a la capacidad de una máquina para imitar o simular procesos de pensamiento y aprendizaje que normalmente asociamos con la mente humana [22].

2.2.3. Inteligencia Artificial Conversacional

Los sistemas de conversación hombre-computadora en línea que utilizan técnicas de procesamiento del lenguaje natural (NLP) remiten a lo que se conoce como agentes de conversación. La tecnología de Inteligencia Artificial Conversacional (IAC) ha avanzado significativa y velozmente desde el trabajo de Turing, gracias a los desarrollos en NLP y la inteligencia artificial. De forma similar, el uso de *chatbots* también ha aumentado, especialmente desde que las principales aplicaciones de mensajería han ido introduciendo sus propios sistemas de *chatbots* como: WhatsApp [23], Facebook Messenger [24] y Telegram [25].

Un *chatbot* es un programa de *software* que utiliza el procesamiento del lenguaje natural para interactuar con los usuarios a través de una conversación de texto. Estos pueden ser programados para realizar una variedad de tareas, como responder preguntas comunes; proporcionar información, realizar transacciones, brindar soporte técnico, entre otras funciones. También, se utilizan a menudo en sitios web, aplicaciones de mensajería y redes sociales para automatizar el proceso de atención al cliente y mejorar la experiencia del usuario [26].

2.2.4. Alternativas Comerciales de IAC

El mercado de las IAC es la aplicación de *software* que se pueda utilizar para construir, orquestar y respaldar el desarrollo de múltiples casos de uso de automatización conversacional. A continuación, se presentan algunos de los líderes en el mercado para esta materia [27]:

1. IBM Watson Assistant [8]: es una plataforma de inteligencia artificial de IBM que permite crear *chatbots* y asistentes virtuales personalizados. Utiliza técnicas avanzadas de procesamiento de lenguaje natural y aprendizaje automático para brindar una experiencia de atención al cliente eficiente y consistente a través de diferentes canales de comunicación.
2. Google Dialogflow [28]: es una plataforma de Google que utiliza el procesamiento del lenguaje natural para crear *chatbots* y asistentes virtuales. Ofrece herramientas para crear flujos de conversación personalizados y soporta múltiples canales de comunicación, tales como Facebook, Messenger, Slack, etc.
3. Amazon Lex [29]: es una herramienta de inteligencia artificial de Amazon Web Services que permite crear *chatbots* y asistentes virtuales. Ofrece integración con múltiples canales y herramientas para crear flujos de conversación personalizados.
4. Microsoft Bot Framework [30]: es una plataforma de desarrollo de *chatbots* de Microsoft que permite crear *chatbots* y asistentes virtuales. Ofrece herramientas para crear flujos de conversación personalizados y soporta, asimismo, múltiples canales de comunicación.

IBM Watson Assistant

Watson Assistant es una plataforma de asistentes virtuales desarrollada por IBM. La plataforma utiliza inteligencia artificial y aprendizaje automático para permitir a los desarrolladores diseñar, desarrollar y desplegar *chatbots* personalizados para diferentes aplicaciones y dispositivos. Por ejemplo, la plataforma permite a los desarrolladores crear flujos de conversación; definir entidades y habilidades, integración de sistemas mediante la API y analizar la retroalimentación de los usuarios para mejorar la experiencia del usuario [8] [31].

Al desarrollar en Watson Assistant hay tres elementos claves que deben tenerse en cuenta: habilidades, entidades, intenciones y diálogo. A continuación, se describen estos elementos con más detalle [32]:

- La habilidad de diálogo (*skill*) es un espacio de trabajo que se utiliza para definir tanto los datos de entrenamiento (intenciones y entidades) como el diálogo. La conversación se representa gráficamente como un árbol. Utiliza el editor de diálogo gráfico para crear una especie de secuencia de comandos para que su asistente la lea cuando interactúa con sus clientes. El cuadro de diálogo teclea los objetivos comunes del cliente que le enseñas a reconocer y proporciona respuestas útiles.

- Las entidades son objetos estructurados y relevantes para la conversación. Por ejemplo, al utilizar el *chatbot* para recargas de saldo, las entidades podrían incluir el monto de recarga y el número de teléfono destino de la recarga, entre otros. Estas entidades ayudan al *chatbot* a extraer los datos específicos que debe obtener desde el usuario para la recarga.

- Las intenciones clasifican las frases que el usuario expresa durante la conversación. Por ejemplo, las intenciones pueden ser: solicitar una recarga, verificar el saldo actual, conocer las promociones disponibles, obtener información sobre los métodos de pago aceptados, entre otros.

- El dialogo define qué se responderá al usuario bajo determinadas condiciones. La creación del diálogo consiste en diseñar la estructura y el flujo de la conversación del *chatbot*. Esto implica definir cómo el *chatbot* responderá a diferentes preguntas de los usuarios y qué información o transacciones se deben realizar para dar la atención. Por ejemplo, al recibir una solicitud de recarga de saldo por parte del usuario (intención), el *chatbot* puede comenzar por solicitar al usuario el número de teléfono destino de la recarga (entidad) y luego validarlo en el sistema (acción). Luego, puede solicitar al usuario elegir el monto deseado (entidad) desde una lista de posibles montos. A continuación, el *chatbot* puede proporcionar información sobre los métodos de pago aceptados y guiar al usuario a través de los pasos necesarios para completar la recarga. Además, el diálogo puede incluir ramificaciones y condiciones para adaptarse a diferentes situaciones. Por ejemplo, si el usuario desea cancelar la recarga o tiene preguntas adicionales sobre el proceso, el *chatbot* puede ofrecer respuestas y opciones específicas para abordar esas situaciones.

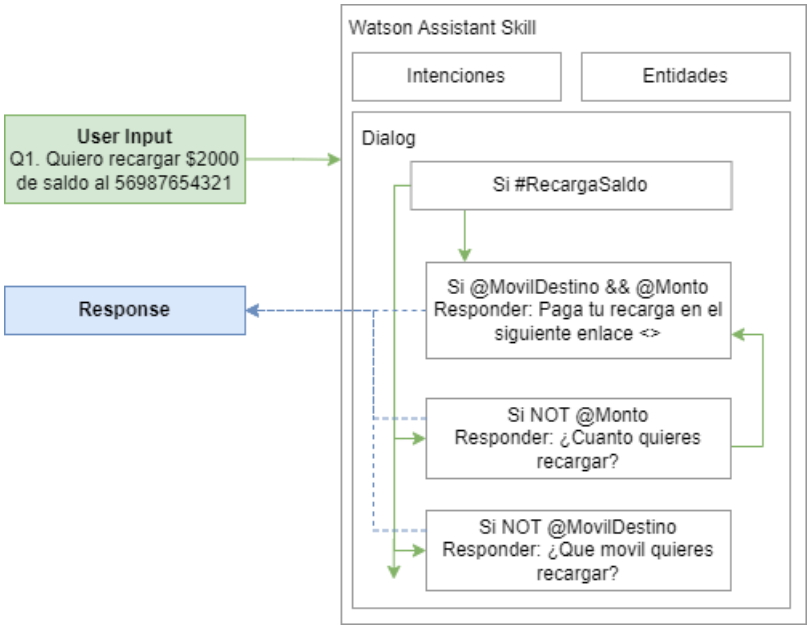

Figura 2.1: Esquema de una habilidad de dialogo, intenciones, entidades, y respuestas en IBM Watson Assistant.

Es importante destacar que estos elementos están interconectados entre sí y es preciso trabajarlos en su conjunto para crear un *chatbot* efectivo y preciso. Por ejemplo, al definir una intención, en el dialogo se puede especificar qué entidades están relacionadas con esa intención, para que el *chatbot* pueda identificar las entidades relevantes y responder de manera precisa.

Lo expuesto se ilustra en la figura 2.1, donde el usuario emite el mensaje "Quiero una recarga de $2000 de saldo al número 56987654321". En el sistema de Watson Assistant, esta interacción se ha configurado dentro de la habilidad de diálogo, asignando la intención #RecargaSaldo, seguida por la validación de las entidades @MovilDestino y @Monto, a fin de proporcionar una respuesta acorde a la solicitud realizada.

2.2.5. Inteligencia Artificial Generativa

La Inteligencia Artificial Generativa (IAG) refiere a un dominio de la IA que se centra en los sistemas capaces de generar contenido por sí mismos, como imágenes, música, texto e incluso videos. A diferencia de los enfoques tradicionales de la IA, que se centran en resolver problemas basados en reglas y patrones predefinidos (por ejemplo, IAC), la IAG busca crear sistemas que puedan producir resultados que parezcan haber sido creados por seres humanos. Durante los últimos dos años ha habido una gran cantidad de modelos generativos como OpenAI GPT[33] (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers) [34] de Google, u OpenAI DALLE-2 [35]. Estos modelos son capaces de realizar tareas, al modo de un sistema general de preguntas y respuestas, crear automáticamente imágenes artísticas a partir del texto, entre otros [36]. A la fecha, *chat* GPT cuenta con mas de 100 millones de usuarios mensuales, 13 millones de visitantes al día y con una inversión de Microsoft por 10 billones de dólares [37].

2.2.6. Computación en la Nube

La computación en la nube, también conocida como *cloud computing* en inglés, es un modelo de entrega de servicios de tecnologías de la información y comunicaciones, a través de Internet. En lugar de alojar y administrar aplicaciones y servicios en servidores físicos locales, la computación en la nube permite a los usuarios crear y administrar recursos informáticos remotos, como servidores, almacenamiento y bases de datos. Este modelo de entrega de servicios informáticos a través de Internet ofrece flexibilidad, escalabilidad y acceso a los recursos informáticos desde cualquier lugar y en cualquier momento.

Las tres nubes más utilizadas son Amazon Web Services, Microsoft Azure y Google Cloud Platform [27]. Cada una de estas plataformas de servicios en la nube tiene sus propias fortalezas y debilidades. En consecuencia, la elección de la plataforma adecuada depende de las necesidades y objetivos específicos de la empresa.

1. Amazon Web Services (AWS) [9]: es la plataforma de servicios en la nube ofrecida por Amazon. Se lanzó oficialmente en 2006, pero en realidad, los servicios de AWS comenzaron a desarrollarse en 2003 cuando Amazon necesitaba una infraestructura escalable y rentable para respaldar su propio negocio de comercio electrónico. Amazon decidió ofrecer sus servicios de infraestructura en la nube al público en general, lo que resultó en el nacimiento de AWS. Actualmente ofrece una amplia gama de servicios en la nube, que incluyen almacenamiento; computación, bases de datos, redes, análisis, inteligencia artificial, aprendizaje automático y más.
2. Google Cloud Platform (GCP) [10]: es la plataforma de servicios en la nube ofrecida por Google. Se comenzó a desarrollar su infraestructura en la nube para respaldar sus propios productos y servicios, como la búsqueda en línea y YouTube. Luego, en 2008, Google decidió abrir su plataforma en la nube al público, dando lugar al lanzamiento de GCP. Esta nube proporciona servicios para computación; almacenamiento, bases de datos, redes, análisis, aprendizaje automático y más.
3. Microsoft Azure [11]: es la plataforma de servicios en la nube ofrecida por Microsoft. Fue lanzado en 2010 y se originó como una extensión de los servicios de Microsoft existentes, como Windows Server y SQL Server. Este proveedor ofrece una amplia gama de servicios en la nube, que incluyen computación; almacenamiento, bases de datos, redes, análisis, inteligencia artificial, IoT y más.

Unos de los enfoques más utilizados en la industria tecnológica para la entrega de servicios está estandarizado por el Instituto Nacional de Estándares y Tecnología (NIST) [38] e incluye los siguientes:

- IaaS (Infraestructura como Servicio): es un modelo en el que los proveedores de servicios en la nube ofrecen infraestructura básica de TI a través de Internet. Esto incluye servidores virtuales, almacenamiento, redes y otros recursos de computación. Los usuarios tienen control total sobre el sistema operativo y las aplicaciones que se ejecutan en la infraestructura proporcionada, lo que les permite gestionar y controlar su entorno

de manera más completa. Un ejemplo de IaaS es permitir que otros usuarios puedan instalar máquinas virtuales con requerimientos específicos, como Amazon EC2 [39].

- PaaS (Plataforma como Servicio): en este modelo, los proveedores ofrecen una plataforma completa para el desarrollo, ejecución y gestión de aplicaciones. Esto incluye entornos de desarrollo, herramientas de programación, bibliotecas y servicios para crear, probar y desplegar aplicaciones en la nube. Los usuarios se encargan del desarrollo de sus aplicaciones, mientras que la infraestructura subyacente es gestionada por el proveedor de servicios en la nube. Un ejemplo de PaaS es AWS ElastiCache [40], el cual es una base de datos en memoria completamente administrada por AWS y accesible desde su API.
- SaaS (Software como Servicio): en este modelo, los usuarios pueden acceder a aplicaciones y *software* completos a través de Internet, sin necesidad de descargar o instalar nada en sus propios dispositivos. Los proveedores de servicios en la nube gestionan la infraestructura, la plataforma y el *software*, y los usuarios solo necesitan acceder a través de un navegador web u otra interfaz para utilizar la aplicación. Los ejemplos comunes de SaaS incluyen aplicaciones de correo electrónico, gestión de relaciones con los clientes (CRM), mantenedores de páginas web, creadores para tiendas en linea y otras herramientas de gestión empresarial (ERP).

2.2.7. Function as a Service (FaaS)

La computación sin servidor, a menudo conocida como *Function as a Service* (FaaS [4]), es el paradigma más reciente de la computación en la nube que surgió del modelo SaaS. Con este nuevo paradigma de computación, los desarrolladores despliegan pequeños fragmentos de código, llamados funciones, y el proveedor de la nube se encarga de aprovisionar toda la infraestructura y recursos necesarios para ejecutar estas funciones. A diferencia de IaaS o PaaS, donde los administradores de la nube aún deben aprovisionar sus servidores, FaaS no requiere aprovisionamiento de recursos ni configuraciones complejas. El proveedor de la nube es responsable de administrar la infraestructura de FaaS, lo que permite a los clientes desplegar y ejecutar sus funciones de manera eficiente y más rápida en comparación con los modelos de computación más antiguos. Con FaaS, los proveedores de la nube cobran por solicitud, no por el uso de recursos. El valor de cada ejecución varía según la configuración de la función y la región en la que se ejecute [41] [42].

En este modelo, las aplicaciones se construyen como funciones que se ejecutan en respuesta a eventos específicos, como solicitudes HTTP o cambios en bases de datos. Esto proporciona una implementación rápida y flexible, ya que los recursos se asignan según los recursos que necesite la función. Además, el modelo de precios se basa en el consumo, lo que significa que los usuarios solo pagan por el tiempo de ejecución y los recursos utilizados, a diferencia de otros enfoques como arquitecturas monolíticas[43] o microservicios [13], en donde se requiere desplegar múltiples instancias de servidor que podrían estar en constante funcionamiento.

Uno de los desafíos más significativos que enfrentan todas las arquitecturas de FaaS es el retraso provocado por el inicio en frío, también conocido como "*cold start*". Al invocar por

[4]También se emplea el término *serverless*, debido a la tecnología *serverless framework*.

primera vez una función en la nube, se desencadenan procesos secuenciales. En el caso de un despliegue comprimido (zip) [42], estos procesos se dividen en las siguientes etapas:

1. Tiempo de inicialización en frío (cold start): esta fase abarca el tiempo empleado en descargar el código fuente de la función desde el almacenamiento, descomprimirlo, copiarlo en un contenedor y luego inicializarlo. Una vez que se maneja la primera solicitud, las siguientes solicitudes se ejecutan más rápidamente durante un cierto período.

2. Tiempo de ejecución: dado que el contenedor ya está inicializado, la función entra en modo de reposo y todas las solicitudes siguientes se ejecutan instantáneamente. Estas etapas comprenden el tiempo de ejecución. Sin embargo, los proveedores de la nube pueden desactivar el contenedor de la función en cualquier momento y, cada vez que lo hacen, los clientes experimentarán nuevamente un inicio en frío para sus solicitudes.

2.2.8. Amazon Web Services

En la actualidad, se ha convertido en una herramienta esencial para empresas de todos los tamaños, desde *startups* hasta grandes corporaciones, ya que permite a los usuarios alojar y administrar sus aplicaciones y servicios en la nube de forma segura, escalable y económica. Además, ofrece un modelo de pago por uso, con arreglo al cual, los usuarios sólo pagan por los servicios que utilizan, sin tener que invertir en costosas infraestructuras de TI [44] [27].

En síntesis, AWS es una plataforma en la nube que ofrece una amplia gama de servicios informáticos; de almacenamiento, redes y bases de datos para alojar y administrar aplicaciones y servicios, los cuales se mencionan a continuación:

1. Amazon Lambda [45] es un servicio de computación sin servidor (*serverless*) que permite ejecutar código sin necesidad de aprovisionar o administrar servidores. El desarrollador configura y carga la función de código, ajustándose automáticamente al volumen de tráfico de la función.

2. Amazon DynamoDB [46] es un servicio de base de datos NoSQL completamente administrado por AWS. Permite al desarrollador almacenar y recuperar grandes cantidades de datos con baja latencia. También ofrece escalabilidad y flexibilidad, ajustándose automáticamente al volumen de tráfico y tamaño de los datos almacenados.

3. Amazon Step Functions [47] permite orquestar y monitorear el resultado de flujos de trabajo. Con esta tecnología, los desarrolladores ejecutan procesos visuales utilizando estados, tareas y transiciones. Además, ayuda en la automatización de procesos y la gestión de flujos de trabajo.

4. Amazon API Gateway [48] permite al desarrollador crear, publicar y gestionar las API. Facilita la creación de interfaces de programación seguras y escalables, y se encarga de la autenticación, autorización y monitoreo del tráfico, mejorando el rendimiento y la seguridad de las aplicaciones en la nube.

5. Amazon Virtual Private Cloud (VPC) [49] permite al usuario crear y gestionar una red virtual personalizada en un entorno de nube. Ofrece control sobre la configuración de red, segmentos de IP, subredes y *gateways*. Garantiza la privacidad y el aislamiento de los recursos, proporcionando un entorno escalable y seguro.

6. Amazon Identity and Access Management (IAM) [50] permite al usuario controlar el acceso a los recursos de la nube. Facilita la creación y gestión de usuarios, grupos y permisos, asegurando que solo las personas y servicios autorizados puedan realizar acciones específicas. De esta forma, mejora la seguridad y el cumplimiento de las políticas de acceso en la nube.

7. Amazon CloudWatch [51] es un servicio de monitoreo y observabilidad que recopila y rastrea métricas, registros y eventos en tiempo real en AWS. Permite analizar y visualizar el rendimiento de aplicaciones, recursos y servicios en AWS. CloudWatch ofrece una amplia gama de métricas, incluyendo el uso de CPU, el rendimiento de la red, el consumo de almacenamiento y la latencia de la base de datos.

2.2.9. NodeJS

JavaScript es un lenguaje de programación, interpretado y orientado a objetos que se utilizan principalmente en el desarrollo web. Fue creado por Brendan Eich en 1995 mientras trabajaba en Netscape Communications Corporation. Se utiliza para agregar dinamismo a las páginas web, facilitando a los desarrolladores crear efectos visuales, animaciones, validaciones de formularios y otras características dinámicas que mejoran la experiencia del usuario.

Este lenguaje se ejecuta en el lado del cliente (*frontend*), específicamente en el navegador web del usuario. Ha evolucionado significativamente desde su creación original y, hoy en día, cuenta con una gran cantidad de *frameworks* y bibliotecas que facilitan su uso y desarrollo.

En cambio, NodeJS [52] es un entorno de ejecución de JavaScript que permite ejecutar código JavaScript en el servidor (*backend*). Esto se consigue mediante el motor V8 de Google. Se utiliza principalmente para crear aplicaciones web escalables y en tiempo real, como aplicaciones de *chat*, juegos en línea y aplicaciones en redes sociales.

La principal diferencia con otros entornos de ejecución es que utiliza un modelo de programación asíncrona y no bloqueante, permitiendo con ello manejar múltiples solicitudes de manera simultánea, sin bloquear la ejecución del código. Esto lo hace especialmente adecuado para aplicaciones que requieren una gran cantidad de entrada/salida, como la lectura y escritura de datos de la base de datos o la manipulación de archivos.

Capítulo 3

Diseño e Implementación de la Solución Tecnológica

3.1. Definición de los casos de uso

A continuación se describen las condiciones e interacciones entre el *chatbot* y el usuario para dos casos de uso fundamentales: el pago de boletas de servicios móvil y hogar, y las recargas de saldos con pago digital. Los diagramas de flujo se puede ver en los anexos A.1 y A.2, respectivamente.

En el caso del pago de boletas de servicios móvil y hogar (ver diagrama de flujo A.1), el usuario inicia la conversación manifestando la intención de pagar su boleta a través de una frase o seleccionando una opción del menú. El *chatbot*, basándose en la información personal y el perfil de facturación del usuario, obtiene el detalle actualizado sobre las deudas pendientes y presenta al usuario las opciones de pago disponibles. Si el usuario tiene más de una boleta pendiente, se muestra el desglose por servicios móvil y hogar, incluyendo el monto total pendiente de pago. Cuando el usuario selecciona la deuda a pagar, el *chatbot* genera de forma automática un enlace de pago que redirige al usuario hacia WebPay, donde puede visualizar el monto a pagar y completar el proceso de pago. Finalmente el flujo termina notificando al usuario el resultado del pago y se ofrecen otras atenciones.

En el caso de las recargas de saldos (ver diagrama de flujo A.2), el usuario expresa la intención de realizar una recarga a través de una frase o seleccionando una opción del menú. El *chatbot* solicita al usuario que ingrese el monto de recarga deseado, el móvil destino y el medio de pago (WebPay o Banco Estado). Como respuesta genera un enlace que direcciona al usuario hacia la pasarela de pagos que corresponda. El usuario puede seguir las instrucciones de la web para completar el proceso de pago y realizar la recarga de saldo. Una vez que se confirma el pago exitoso, se actualiza el saldo del usuario y se notifica al usuario por WhatsApp el estado de la transacción, ofreciendo otras atenciones, según sea el caso.

3.2. Situación Antes de la Implementación

La plataforma actual del *chatbot* se basa en una conversación asíncrona con los clientes, a través de la plataforma de mensajería de WhatsApp Cloud API. En el diagrama de bloques presentado en la figura 3.1, la capa *Core* se encarga de recibir y orquestar los mensajes y transacciones internas del sistema, asegurando una interacción fluida. Una componente de Inteligencia Artificial Conversacional es responsable de reconocer las intenciones y entidades de los clientes, y proporcionar respuestas adecuadas. La información sobre los usuarios se obtiene ejecutando acciones automáticas que consultan un único punto de acceso para los datos sobre clientes, productos, facturas, órdenes, entre otros.

Los usuarios interactúan con el *chatbot* en la aplicación oficial de WhatsApp, en todas las versiones disponibles y para todos los dispositivos soportados por el distribuidor oficial.

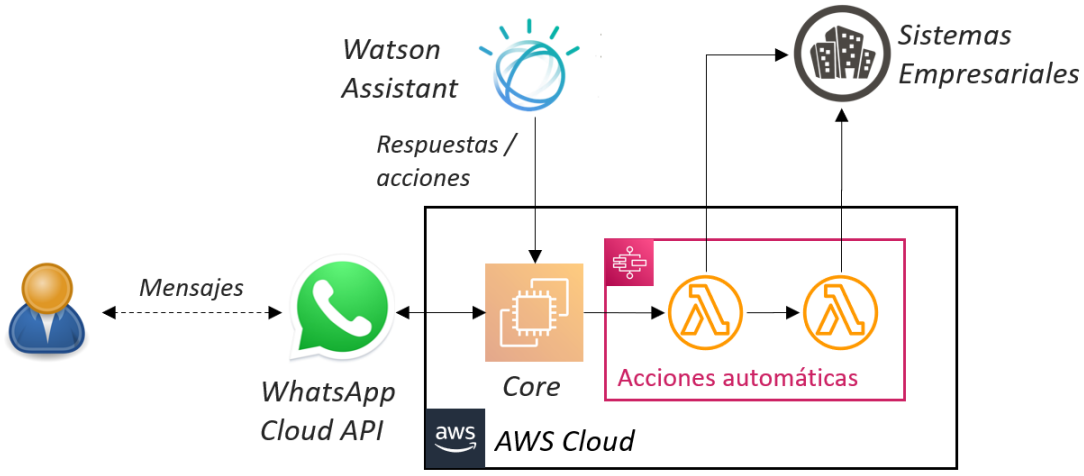

Figura 3.1: Diagrama de bloques para la plataforma actual del *chatbot*.

Capa Whatsapp Cloud API

Los usuarios interactúan con el *chatbot* mediante la plataforma de mensajería de WhatsApp. En cambio, la comunicación se consigue haciendo uso de WhatsApp Cloud API y, permitiendo con ello, el ingreso de mensajes desde los usuarios y hacia el *chatbot*. En la arquitectura actual, todos los mensajes provenientes de WhatsApp son enviados mediante peticiones HTTPs hacia un servicio central en la capa *Core*, denominado *webhook*. En cambio, para los mensajes que son originados por el *chatbot* hacia los usuarios, el *Core* implementa directamente la API Graph para WhatsApp Cloud API [23].

Con el fin de mantener un entorno separado para desarrollo y producción, se utilizan dos cuentas de WhatsApp que dirigen los mensajes hacia el *Core* correspondiente: uno para el ambiente de desarrollo y otro para el ambiente de producción. De esta manera, el desarrollo y las pruebas no afectan la aplicación en línea. Los mensajes enviados desde WhatsApp en el entorno de desarrollo son redirigidos al *Core* de desarrollo, mientras que los mensajes del entorno de producción se dirigen al *Core* de producción.

Capa de Inteligencia Artificial Conversacional

La capa de Inteligencia Artificial Conversacional desempeña un papel fundamental en la gestión de los diálogos con los usuarios: el reconocimiento de entidades, clasificación de intenciones y la implementación de dialogo, junto a la ejecución de acciones automáticas que obtienen información de los usuarios. Para llevar a cabo estas tareas, se utiliza la versión del API v1.0 de Watson Assistant [53] y se dispone de una instancia para el entorno de desarrollo y otra para el entorno de producción.

Cada instancia de Watson Assistant puede contener múltiples *skill*, aunque la implementación actual con el *Core* solo admite un *skill* por ambiente. Para implementar los cambios en el ambiente de producción, es necesario descargar manualmente el *skill* desde el entorno de desarrollo y cargarlo en el entorno productivo. Previamente, se debe realizar un respaldo de la versión anterior (*backup*), en caso de que sea necesario revertir los cambios (*rollback*).

Capa Core

La capa *Core* es la capa que tiene la responsabilidad de orquestar toda la plataforma de *chat* y se encuentra alojada en la nube de AWS. La arquitectura de la capa *Core* se basa en el concepto de microservicios, lo que implica el uso de servidores especializados que se encargan de tareas específicas, tales como: la comunicación bi-direccional con WhatsApp, comunicación bi-direccional con la plataforma de atención para los ejecutivos comerciales y repetición de un ciclo de interacción en línea con Watson Assistant (ver sección 3.2).

El *Core* delega la responsabilidad a la capa de acciones automáticas para implementar con mayor flexibilidad los distintos casos de uso. Para llevar a cabo esto, soporta la invocación de acciones automáticas implementadas con la tecnología de AWS Step Functions, que a su vez, ejecutan funciones de AWS Lambda.

Por otro lado, cuenta con una base de datos en memoria centralizada que almacena el contexto de la conversación con los usuarios. Este contexto se enriquece con datos a medida que se generan las interacciones con el usuario y el sistema, permitiendo a Watson Assistant aplicar condiciones y tomar decisiones basadas en este contexto. El contexto tiene una duración predeterminada y expira después de cierto tiempo de inactividad del usuario. Esta base de datos centralizada se encuentra implementada en AWS ElastiCache [40], mediante su despliegue de Redis [54].

Existen dos despliegues del *Core* que están implementados en cuentas de AWS independientes, esto separa la de infraestructura los ambientes de desarrollo y producción.

Capa de Acciones Automatizadas

La capa de acciones automatizadas engloba un conjunto de flujos implementados para consultar los sistemas internos de la empresa, y así proporcionar a Watson Assistant datos en el formato adecuado para su uso en el diálogo. Esta capa se basa en una arquitectura *FaaS*

y la elección se justifica principalmente por dos motivos:

1. Facilita la escalabilidad horizontal a medida que aumenta el volumen de transacciones realizadas por los usuarios y solo incurre en costos por tiempo de ejecución.
2. Las actualizaciones del código fuente solo afectan a la función en cuestión. Al existir un desacople entre los distintos componentes, cualquier falla afectará solo el flujo conversacional involucrado, aumentando la confiabilidad del sistema.

Las acciones se implementan utilizando AWS Step Functions. Cada tarea del flujo de trabajo se implementa en funciones de código (AWS Lambda), que realizan consultas a la capa de integración y aplican alguna transformación a los datos. Lo que también permite el uso de servicios adicionales de AWS para tablas de datos o almacenamiento de archivos. Existe un catalogo de acciones automáticas que incluyen: estado de contrato, detalle de productos contratados, detalles de facturas, el perfil del clientes, entre muchas otras. Pero no existen acciones para generar enlaces de pago ni consultar la deuda actualizada de los clientes.

Esta capa se encuentra aislada entre los ambientes de desarrollo y producción a nivel de infraestructura, mediante el uso de cuentas de AWS independientes. La cuales, se encuentran conectadas coherentemente con los ambiente de producción y desarrollo del *Core*.

Sistemas Empresariales

Son todos los sistemas que la empresa ha implementado para operar su negocio, los cuales contemplan: operación de la red móvil y fija, gestión y atención de clientes, contratos, productos, ventas, comercio electrónico, entre muchos más. Cada sistema posee ambientes de desarrollo y producción, separados a nivel de infraestructura, los cuales se encuentran instalados bajo esquemas de "nube híbrida", servidores propios de la empresa (*on-premise*) y distintas nubes (*multicloud*).

En el caso del *chatbot*, los sistemas internos de la empresa que proveen datos de clientes solo son accesibles mediante un componente denominado API Manager; el cual actúa como intermediario entre el cliente y el servidor de la petición, permitiendo comunicación HTTPs entre sistemas. Esta capa presenta tres principales ventajas: una documentación exhaustiva de los servicios ofrecidos, un punto de conexión con alta confiabilidad y soporte técnico.

La integración con esta capa se puede dar como consumidor de servicios (cliente) o sistema proveedor (servidor). En el primer caso, el sistema de origen envía una petición HTTPs hacia el API Manager, la cual dirige la transacción hacia otro sistema empresarial que retorna los datos. En el caso contrario, cuando un sistema proveedor de servicios se conecta a la capa de integraciones, este sistema debe ser capaz de atender las peticiones que generan los sistemas consumidores. En cualquier caso, se considera que los sistemas empresariales existentes funcionan correctamente de principio a fin (son ideales). Además, están preparados para contener una cantidad definida de transacciones por segundo, que se solicita a los administradores de la capa.

Esta capa posee una separación a nivel de infraestructura para los ambientes de desarrollo y producción, el cual está conectado coherentemente con los ambientes de acciones automáticas.

Ciclo de Interacción

En la situación presentada en las figuras 1.1 y 1.1, el usuario inicia la conversación con el *chatbot* enviando el mensaje "hola"; a lo cual el *chatbot* responde con un menú de bienvenida. Para responder al usuario, el *Core* llevó a cabo el ciclo de interacción presentado en la figura 3.2.

La interacción comienza por parte del usuario, al enviar un primer mensaje, el cual es procesado por el *Core* y enviado hacia Watson Assistant para su reconocimiento, interpretando el mensaje como una intención de saludo. Como resultado, Watson responde que se debe ejecutar una acción automática, para verificar si el remitente corresponde a un número de teléfono móvil activo en la compañía, entregando también los datos requeridos por la acción.

El *Core* ejecuta la acción `verificarCliente`, utilizando los parámetros enviados por Watson Assistant. El resultado de la ejecución se agrega a los datos de contexto y nuevamente se envía una petición hacia Watson Assistant. En esta segunda solicitud, Watson Assistant puede utilizar el contexto agregado para avanzar en el flujo, en particular el dato "cliente".

En base a lo anterior, determina el menú de atención a responder y queda a la espera de una nueva interacción por parte del usuario, repitiendo el ciclo cada vez. A medida que se desarrolla el diálogo, Watson Assistant aprovecha la información del contexto para dar continuidad a la conversación, responder a los usuarios y aguardar la siguiente interacción [1].

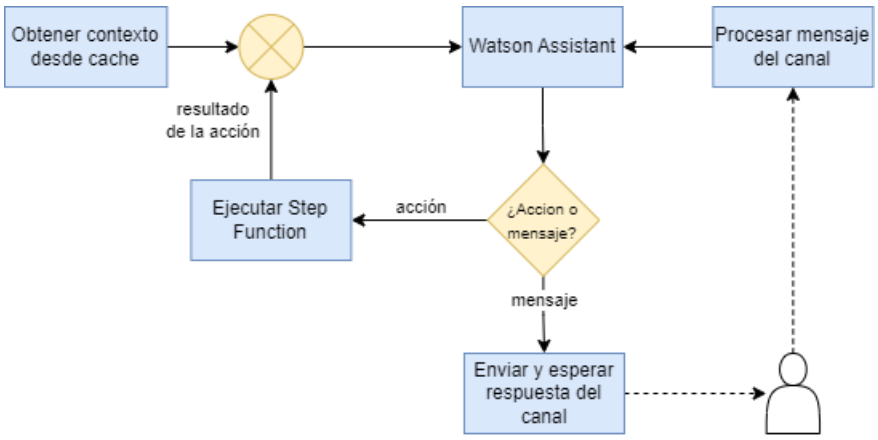

Figura 3.2: Ciclo de interacción por el cual es responsable la capa Core.

[1]Para simplificar los elementos que son necesarios para la solución, no se aborda una tercera decisión por parte de Watson Assistant: la acción "transfer". Esta acción es atendida por el *Core* y abre un canal de comunicación con un asesor humano.

3.3. Diseño de la Solución Tecnológica

En la siguiente sección se describe el diseño de la solución tecnológica para los casos de uso propuestos. Los usuarios inician la interacción en el canal de WhatsApp, cuyos mensajes son transportados desde WhatsApp Cloud API hacia la capa *Core*. Dado que el problema está estrechamente relacionado con el diseño de un flujo conversacional, no es necesario realizar modificación alguna a la capa *Core*.

Se debe tener presente la dualidad acción y respuesta en el ciclo de interacción, debido a que Watson Assistant invocará consecutivamente acciones automáticas hasta que logre cumplir con las condiciones para responder al usuario y, en consecuencia, quedar en espera de una nueva interacción.

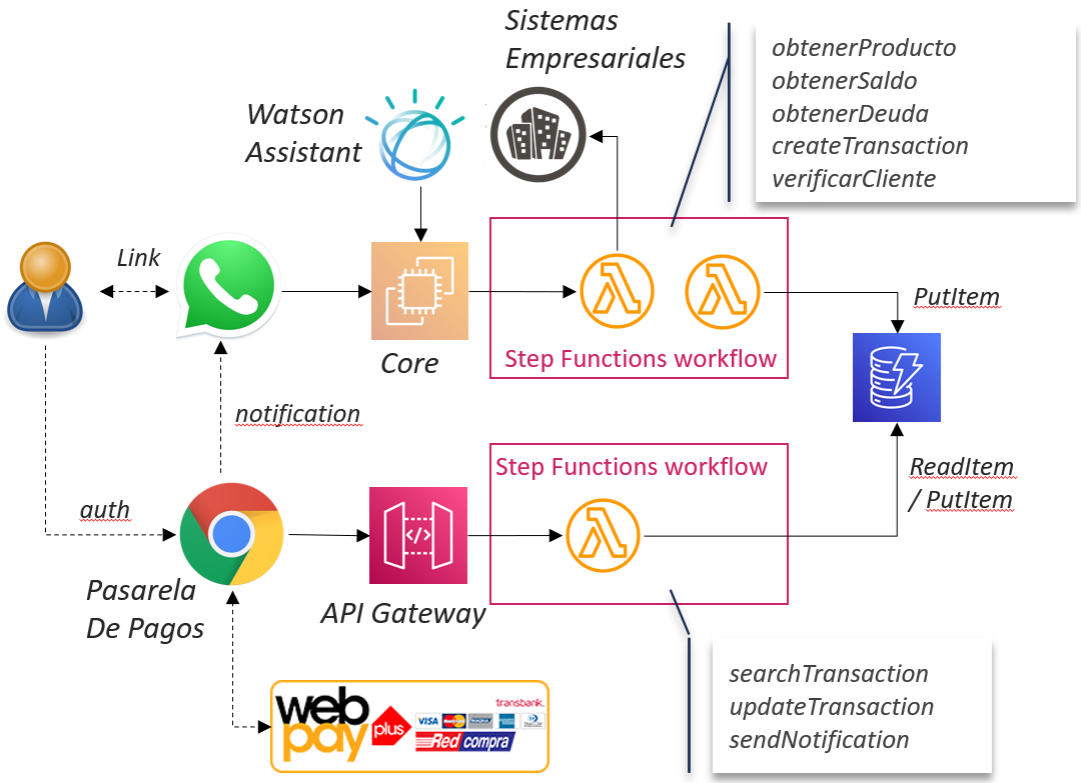

Figura 3.3: Diseño de la solución tecnológica implementada

En la figura 3.3 se puede ver la solución propuesta para los dos casos de uso. El usuario interactúa por medio del canal de WhatsApp, mientras Watson Assistant invoca las AWS Step Function consecutivamente hasta crear el enlace de pago (*link*). Cuando se ingresa al *link* de pago, el usuario es dirigido hacia el navegador web, en donde la pasarela de pagos consulta los datos asociado al enlace de pago para poder crear la transacción en WebPay. Finalmente, cuando el usuario autoriza la transacción, la pasarela de pago notifica por WhatsApp el resultado de la transacción, pudiendo ser exitoso o fallido.

En este contexto, AWS Step Function invoca funciones AWS Lambda para conseguir la tarea para la cual ha sido definida. Esta funciones Lambda contienen la lógica (código

fuente) para generar el enlace de pago y almacenar la información clave en la base de datos. Se utiliza AWS DynamoDB como almacenamiento persistente, lo que permite ser consultado posteriormente por la pasarela de pagos (ver esquema de datos C.12), utilizando como llave primaria un identificador único y universal de la transacción, UUID V4 [55].

3.3.1. Descripción de los Componentes

El diagrama C.1 indica todos los componentes involucrados en la solución tecnológica implementada. Sus especificaciones técnicas se pueden revisar en el anexo C.

Los siguientes componentes se encontraban disponibles en el catálogo de acciones automáticas AWS Step Function antes de la implementación de este trabajo, cuyas funciones Lambda configuradas en las acciones han sido programadas en NodeJS [52] para distintas versiones, que pueden variar entre la 14, 16 o 18 [56] según la fecha en la cual han sido implementadas (legados), por lo que fue factible su reutilización en el flujo conversacional de Watson Assistant:

- obtenerProducto: permite adquirir el producto asociado al número telefónico desde el cual el usuario se comunica y facilita obtener el estado del contrato para determinar si este se encuentra activo o suspendido.
- obtenerSaldo: permite obtener el saldo actual de un cliente prepago y así ofrecer la recarga cuando el usuario no tiene saldo ni bolsas disponibles.
- verificarCliente: valida si el numero de teléfono corresponde a un cliente de la empresa de telecomunicacines. Este componente existe en el catálogo y se puede reutilizar.
- sendNotification: en cuanto a las notificaciones de éxito o falla del pago de boleta o recarga de saldo, está disponible el modulo para notificaciones de WhatsApp; el cual se encuentra habilitado como sistema proveedor de servicio en API Manager y AWS Notification API Gateway, para que otros sistemas de la compañía puedan enviar notificaciones a usuarios. Si bien, funcionalmente el módulo permanece intacto, se hace necesario registrar los datos para las plantillas de notificación: exitosa para el pago de boleta, fallida para el pago de boleta, exitosa para la recarga de saldo y fallida para la recarga de saldo.

La generación de los enlaces de pago implica dos aspectos críticos: incorporar lógicas de generación del enlace que son distintas para cada caso de uso y la necesidad de una API para su integración con la pasarela de pagos. No fue posible reutilizar ningún componente existente, lo que implicó el desarrollo del código fuente y sus configuraciones desde cero.

Para el desarrollo del código fuente se utilizó la última versión estable para NodeJS en AWS Lambda (v18). Para lograr esto se han seguido tres prácticas de desarrollo: versionamiento del código por medio de GIT (Atlassian Bitbucket [57]), documentación de código fuente con JSDoc [58] e implementación de pruebas unitarias automáticas con JestJS [59].

En este contexto, el diseño contempla tres operaciones fundamentales: la creación, búsqueda y actualización de enlaces de pago. Cada una de estas operaciones se lleva a cabo mediante el uso de una AWS Step Function. Específicamente, la creación de enlaces es consumida por Watson Assistant. Por otro lado, las operaciones de búsqueda y actualización utilizan las AWS Step Function, las cuales se implementan en el Internal API Gateway que se ha publicado en el API Manager, logrando una integración efectiva con la pasarela de pagos de la compañía.

Los siguientes componentes tuvieron que ser implementados desde cero debido a que no existían en el catálogo de acciones automáticas:

- obtenerDeuda: se encarga de obtener la deuda actual del usuario y aplicar la agrupación por: móvil, hogar y total. De esta forma, el *chatbot* le puede solicitar al cliente si desea pagar una o todas ellas, junto a presentar los datos en tipo moneda. Esta acción no está disponible en el catálogo, por lo que fue necesario su diseño e implementación utilizando la API en la capa de integraciones.
- createTransaction: tiene como responsabilidad la generación del enlace de pago para la transacción. Para el pago de boleta de servicios, el vínculo se compone de una cadena cifrada en RC4, que comprende datos tales como el identificador único de la transacción e identificación del usuario. En cuanto a las recargas de saldo, el enlace se compone exclusivamente por el identificador único de la transacción. En ambos escenarios, el enlace posee la estructura "https://telecom.cl?q=token", donde el campo "q=" hace alusión a la cadena cifrada o el identificador único, según corresponda. El identificador único de la transacción siempre se corresponde con una cadena de texto UUID V4. Los datos de entrada para este servicio se puede ver en C.4 y C.3, los datos de repuesta se pueden ver en C.5.
- searchTransaction: es utilizada por la Pasarela de Pagos y se requiere para obtener el detalle de la transacción de pago. Esta consulta se basa en el identificador único de la transacción. Los datos de entrada se pueden ver en C.8.
- updateTransaction: se encarga de actualizar el estado de la transacción de pago basado en su identificador único; la utiliza la pasarela de pagos para declarar una transacción como pagada y así facilitar el seguimiento de las operaciones. La transacción tiene cuatro estados posibles: "CREATED", "PAID", "CLOSED" y "ERROR".

3.3.2. Diagramas de Secuencia

El diagrama de arquitectura de la solución indica los componentes, pero no el orden y las condiciones en que se deben ejecutar las consultas hacia las acciones automáticas y a cuales servicios empresariales. Para lograr esto, se construyeron dos diagramas de secuencia que se limitan hasta la generación del enlace de pago. Así, las figuras B.1 y B.2 presentan en detalle la secuencia de invocación para las acciones automáticas y las condiciones en las cuales se debe entregar una respuesta al usuario, para cada caso de uso.

Posterior a la generación del enlace de pago, el control se encuentra en la pasarela de

pagos (navegador web) hasta que se finaliza la transacción, la cual invoca la misma secuencia de integración para ambos casos de uso y se expone en la figura B.3.

3.4. Implementación pago de boleta de servicios móvil y hogar

A continuación, se exponen y describen los resultados obtenidos tras la implementación del caso de uso concerniente al pago de boletas de servicios móviles y de hogar. Esta implementación se ha realizado en estricta concordancia con los diseños previamente presentados en la sección anterior. Con el propósito de verificar su correcto funcionamiento en condiciones reales, se presentan los resultados en el entorno productivo, es decir, aquel que los usuarios utilizan durante su operación en tiempo real.

Figura 3.4: Bienvenida e invitación a pagar para clientes suspendidos.

Figura 3.5: Generación del enlace de pago para el pago de ambas cuentas.

El flujo de ejecución puede dar inicio mediante un saludo emitido por parte del usuario, lo cual desencadena una secuencia de acciones para perfilar al usuario (que no se detallan en este contexto). Una de las acciones incluidas en el flujo es la función `obtenerProducto`, en la cual Watson Assistant emplea el campo denominado "estadoCuenta" para determinar si el cliente se encuentra en situación de suspensión debido a falta de pago, y de ser así, le

ofrece directamente la opción de generar el enlace de pago correspondiente. El resultado de este proceso se puede observar en la figura 3.4, invitando al usuario a "pagar ahora".

En el caso de que el cliente opte por la alternativa "Pagar Ahora" o manifieste su intención de efectuar el pago de la boleta mediante un diálogo abierto, Watson Assistant solicita la ejecución de la acción automática `obtenerDeuda`. Dicha acción arroja información sobre las deudas pendientes. Si no existiesen deudas pendientes, el flujo de ejecución finaliza. No obstante, si las hubiera, se le informa al cliente sobre dichas obligaciones, agrupadas según el tipo de servicio, como se puede apreciar en las botoneras de la figura 3.4.

A continuación, el cliente tiene la opción de seleccionar la deuda móvil, la deuda hogar o ambas, utilizando botones interactivos. En cualquiera de los casos, se genera el enlace de pago correspondiente a la deuda seleccionada, utilizando para ello la acción `createTransaction`. El enlace de pago es extenso por lo que ha sido ocultado detrás del botón "Pagar mi boleta", como se ve en la figura 3.5. En la prueba expuesta, el usuario se encuentra suspendido por no pago, por lo que el *chatbot* agrega también a la conversación que la reposición del servicio podría tardar hasta dos horas. Es importante destacar que los datos asociados al pago fueron almacenados en el momento de crear el enlace y serán utilizados posteriormente por la web.

Cuando el usuario hace *click* en el enlace de pago, este direcciona hacia la pasarela de pagos de la compañía. A partir de este punto el control se encuentra completamente en el navegador web. Este descifra el identificador único de la transacción desde la cadena "q" y lo utiliza para obtener el detalle de la misma. Obtenido el detalle de la transacción, la pasarela de pagos posee la información suficiente para mostrar en pantalla el monto y cursar la transacción en WebPay (ver figura 3.6).

En la vista de WebPay, el cliente debe seleccionar el medio de pago (Tarjeta o OnePay) y, posteriormente, autorizar el pago a través de su entidad bancaria. Una vez completada la autorización por parte del usuario, el control regresa a la pasarela de pago, la cual procede a procesar el pago (ver figura 3.7). Durante esta etapa, el proceso puede culminar con éxito o con un fallo, y en ambos casos, la pasarela de pagos actualiza el estado de la transacción según corresponda, "state=error" o "state=next".

En el caso de un pago exitoso, el resultado se visualiza en la figura 3.8, donde el *chatbot* notifica al usuario que su boleta ha sido pagada y le invita a descubrir más beneficios o a finalizar la interacción. Posteriormente, en la figura 3.9 se muestra una nueva consulta sobre la intención de pago de boletas, a lo cual el *chatbot* responde que el usuario no posee deudas pendientes. Si ocurre alguna falla en el diálogo de Watson Assistant, el *chatbot* informa al cliente que ha habido un error en la transacción e invita a intentarlo nuevamente más tarde. Asimismo, si se presenta un error en la pasarela de pagos, se notifica al usuario sobre el inconveniente y se le sugiere intentar nuevamente en otro momento.

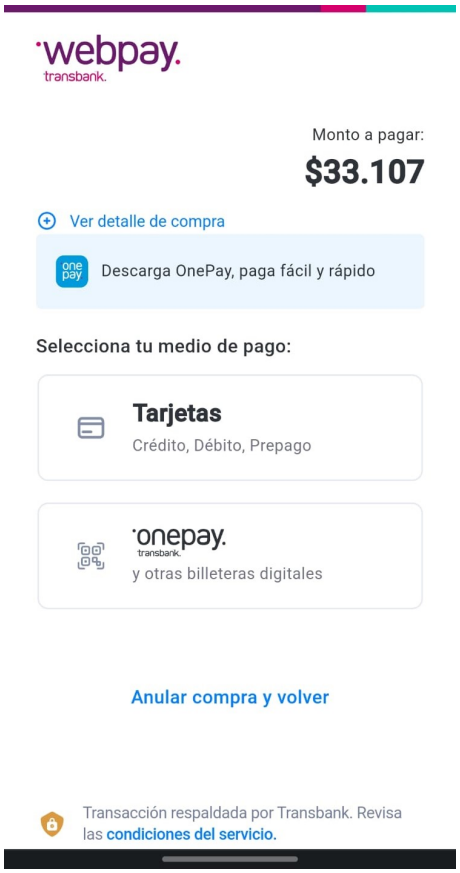

Figura 3.6: Vista Webpay antes del pago de una boleta de servicios.

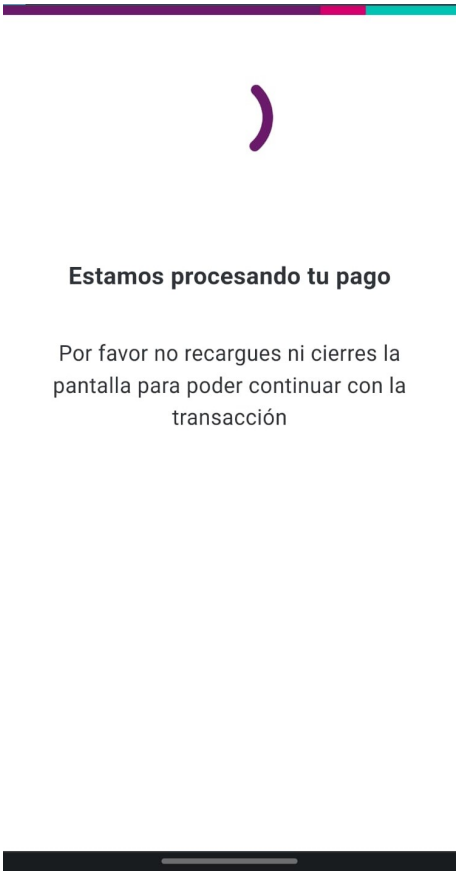

Figura 3.7: Vista Webpay después del pago de una boleta de servicios.

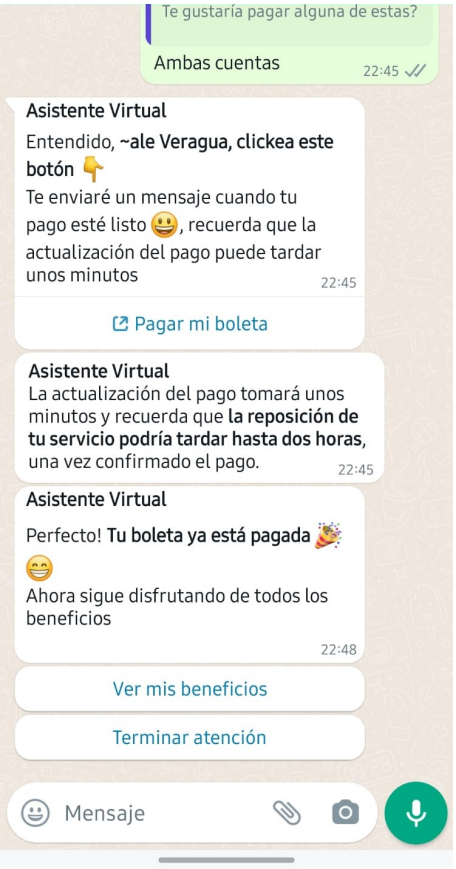

Figura 3.8: Notificación de transacción exitosa para el pago de boleta.

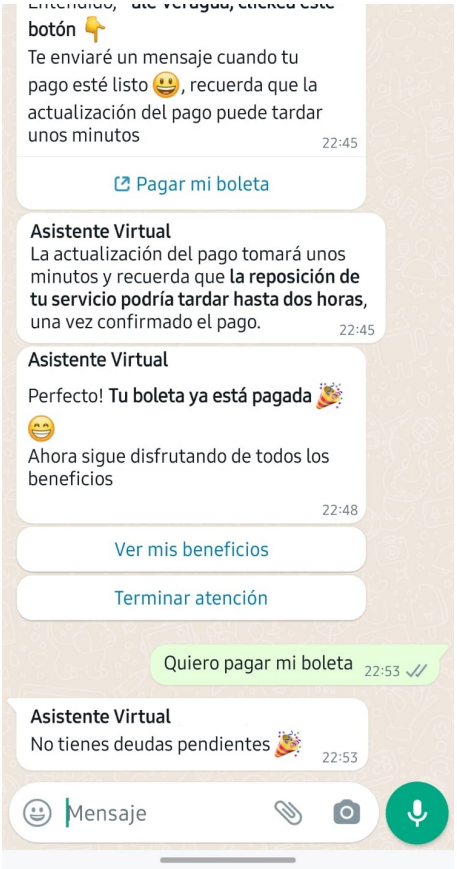

Figura 3.9: Confirmación de cliente sin deuda.

3.5. Implementación recargas de saldos con pago digital

En la siguiente sección se exponen los resultados de la implementación correspondientes al caso de uso sobre recargas de saldos mediante pago digital. Esta implementación ha sido ejecutada en conformidad con los diagramas presentados en la sección dedicada al diseño. Con el propósito de validar su funcionamiento en un entorno real, los resultados mencionados a continuación han sido obtenidos en un ambiente productivo, es decir, aquel que los clientes utilizan en tiempo real.

La primera interacción se origina a partir de un saludo por parte de un usuario prepago, lo cual desencadena una serie de acciones, entre las que se incluye la operación `obtenerSaldo`. En el ejemplo representado en la figura 3.10, el usuario no cuenta con saldo disponible ni dispone de bolsas de navegación (*sms*, minutos, *gigas*, entre otros). Este contexto es identificado por el *chatbot*, el cual responde al usuario sugiriéndole realizar una recarga. En caso de que el cliente confirme su intención de "Recargar saldo", el *chatbot* le proporciona una lista desplegable con diferentes opciones de montos de recargas y los beneficios asociados a cada uno de ellos (ver figura 3.11). El usuario tiene la posibilidad de seleccionar alguna de estas opciones, y posteriormente, dicho valor es almacenado en la entidad *@MontoRecarga*.

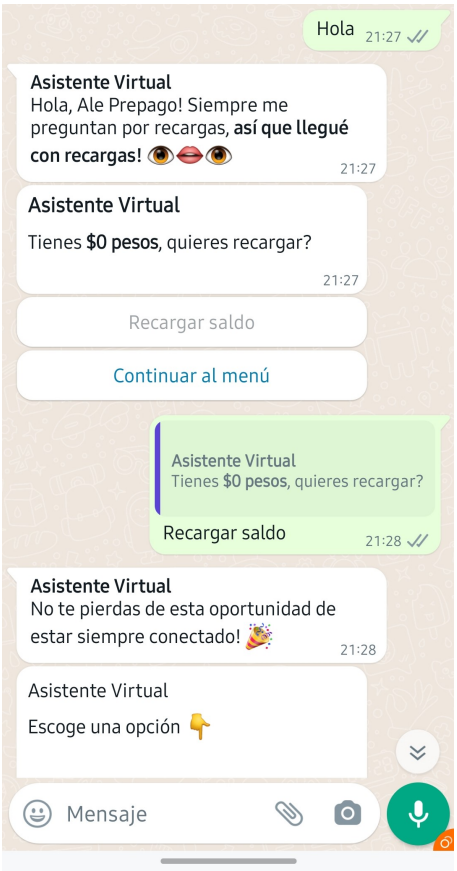

Figura 3.10: Bienvenida e invitación a recarga para clientes sin saldo.

Figura 3.11: Menú desplegable con posibles montos de recargas.

El *chatbot*, en esta situación, deduce que el número de teléfono móvil al que se realizará la recarga corresponde al mismo usuario que está interactuando a través de WhatsApp. Sin embargo, también existe la posibilidad de que el usuario desee efectuar la recarga para otro número, y en tal caso, se le solicita el número de móvil destino. Luego, se extrae la entidad *@MovilDestino* y se procede a validar si dicho número pertenece a un cliente de la empresa mediante la acción `verificarCliente`.

Una vez que el cliente ha seleccionado el monto de la recarga, Watson Assistant solicita el medio de pago, que puede ser: WebPay o Banco Estado. Al seleccionar el medio de pago, Watson Assistant extrae la entidad *@MedioPago* y genera el enlace de pago utilizando la acción `createTransaction`. Este enlace de pago incluye los detalles necesarios para que la pasarela de pagos pueda llevar a cabo la transacción, tales como el monto de la recarga, el número de móvil destino, el medio de pago utilizado y los datos comerciales relevantes (ver tabla A.1). Es importante destacar que las bonificaciones asociadas a los montos seleccionados serán activadas una vez se haya realizado la provisión de la recarga. Por ende, es crucial almacenar el código de comercio al generar el enlace de pago.

Cuando el cliente accede al enlace de pago, el control se transfiere a la pasarela de pagos en el navegador web. Esta pasarela obtiene el identificador único de la transacción, que se

encuentra explícitamente en la cadena de texto "q" y lo utiliza para obtener los detalles de la transacción mediante una llamada al servicio `searchTransaction`. Con estos datos disponibles, la pasarela redirige al usuario hacia WebPay, donde se muestra el monto a pagar y se crea el proceso de pago en WebPay (ver figura 3.14). En este punto, el usuario debe autorizar el pago a través de su entidad bancaria y, posteriormente, el control regresa a la pasarela de pagos para procesar el resultado (ver figura 3.15).

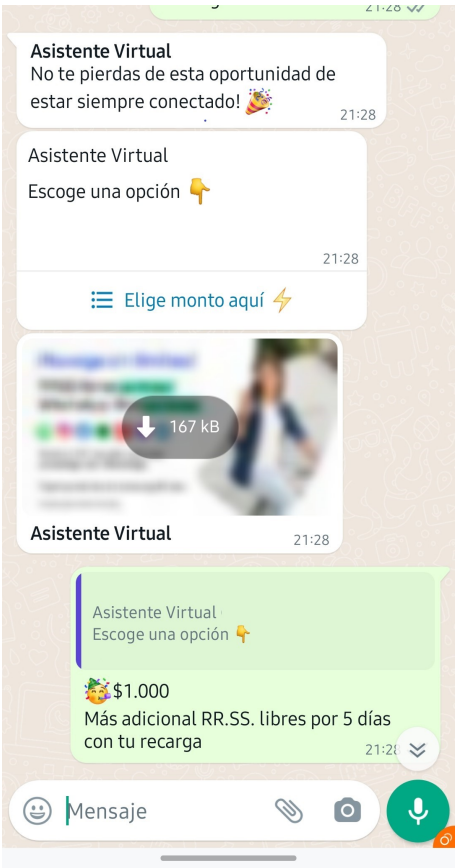

Figura 3.12: Selección del monto a recargar.

Figura 3.13: Enlace para pagar una recarga.

Si el pago se realiza con éxito, la pasarela de pagos de la empresa procede a realizar la recarga. Tras ello, actualiza el estado final del enlace de pagos mediante una llamada al servicio `updateTransaction`. Finalmente, la misma pasarela notifica por WhatsApp el resultado de la transacción, ya sea exitosa o fallida, utilizando el servicio `sendNotification`. La notificación mostrada en la figura 3.16 corresponde a un caso de éxito, en la que se invita al cliente a realizar una compra de bolsa adicional, consultar el saldo o acceder al menú principal.

Para verificar que la recarga ha sido exitosa, en la figura 3.16 se muestra una interacción por parte del usuario donde manifiesta su intención de realizar una "Consulta de saldo". En respuesta, el *chatbot* proporciona los detalles del saldo, incluyendo el monto disponible, la fecha de la consulta y la fecha de vencimiento. Esta acción se realiza mediante una nueva invocación a la acción `obtenerSaldo`.

En caso de que ocurra algún fallo en el diálogo con Watson Assistant, el *chatbot* informa

al cliente que ha habido un error en la transacción e invita a intentarlo nuevamente más tarde. Asimismo, si se presenta un error en la pasarela de pagos, se notifica al usuario sobre el inconveniente y se le sugiere intentar nuevamente en otro momento.

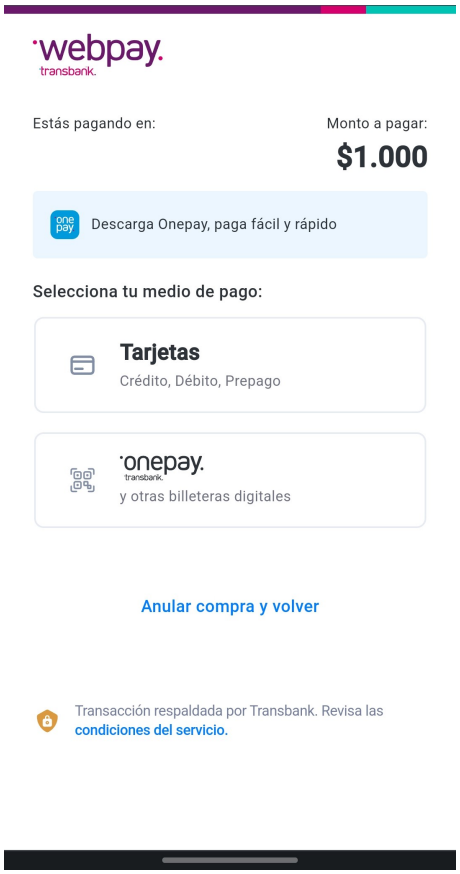

Figura 3.14: Vista Webpay antes del pago de una recarga.

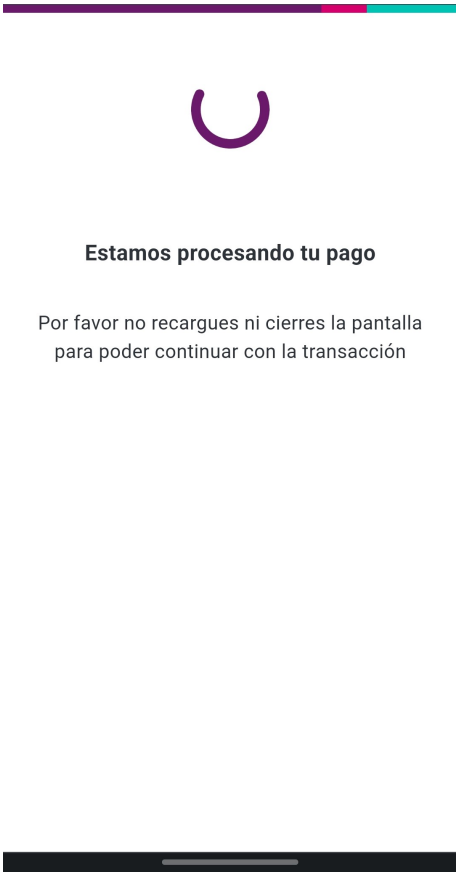

Figura 3.15: Vista Webpay después del pago de una recarga.

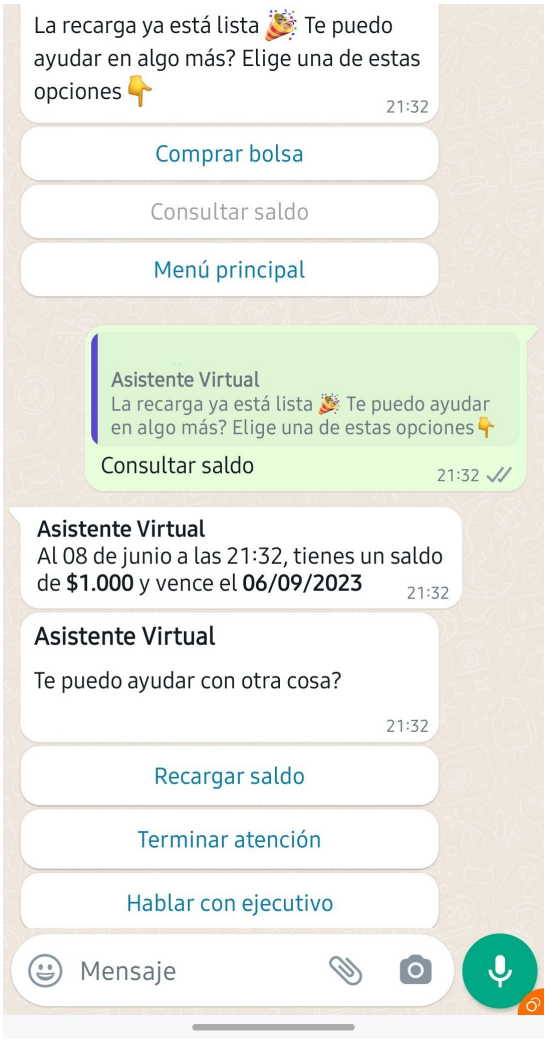

Figura 3.16: Consulta y confirmación del saldo después de la recarga.

Capítulo 4

Resultados y Análisis

4.1. Corrección del efecto Cold Start

En la sección 2.2.7, se ha abordado previamente el fenómeno conocido como inicialización en frío que se presenta en las arquitecturas de tipo *serverless* o en los modelos de ejecución "Function as a Service" (FaaS). Por otra parte, se ha observado que el fenómeno del *cold start* es menos acentuado en el lenguaje de programación NodeJS en comparación con los lenguajes Python y Java. Asimismo, se ha comprobado experimentalmente que la utilización de despliegues comprimidos en formato *zip* para NodeJS, también proporciona tiempos de arranque más rápidos en comparación con aquellos basados en contenedores [42]. Por consiguiente, el resultado minimiza en gran medida el impacto de este efecto en la arquitectura de la presente solución. Es importante destacar que, si se realiza algún cambio en el lenguaje de programación o en el tipo de despliegue, resultaría en un incremento del fenómeno del *cold start* en mayor medida de la que se describe a continuación.

Para abordar la mitigación del efecto en AWS Lambda, el proveedor recomienda implementar el enfoque de "capacidades provisionadas" [60]. La simultaneidad aprovisionada es la cantidad de entornos de ejecución pre-inicializados que se asignan a la función. Estos entornos de ejecución están preparados para responder inmediatamente a las solicitudes de funciones entrantes, más aún teniendo en cuenta que la configuración de la simultaneidad aprovisionada genera cargos en la cuenta de AWS. Esto ayuda a mitigar el efecto de "inicialización en frío" al mantener cierta capacidad siempre disponible, reduciendo así los tiempos de arranque de las funciones. Al establecer un nivel de concurrencia, la función se beneficia de una respuesta más rápida y predecible, lo que resulta útil en casos donde el tiempo de respuesta es crítico, mejorando así la experiencia del usuario y los RTT de la aplicación.

Esta estrategia implica reservar una cantidad específica de entornos de ejecución para la función. En la figura 4.1, tras implementar esta medida en el día 29 durante la operación, se logró reducir el RTT promedio diario de la función en 15 milisegundos, con tan solo una concurrencia aprovisionada para cada función. Esta configuración se aplicó en dos componentes críticos donde el tiempo de respuesta es de suma importancia, ya que son percibidos directamente por el usuario en el contexto de la pasarela de pagos: `searchTransaction` y

`updateTransaction`.

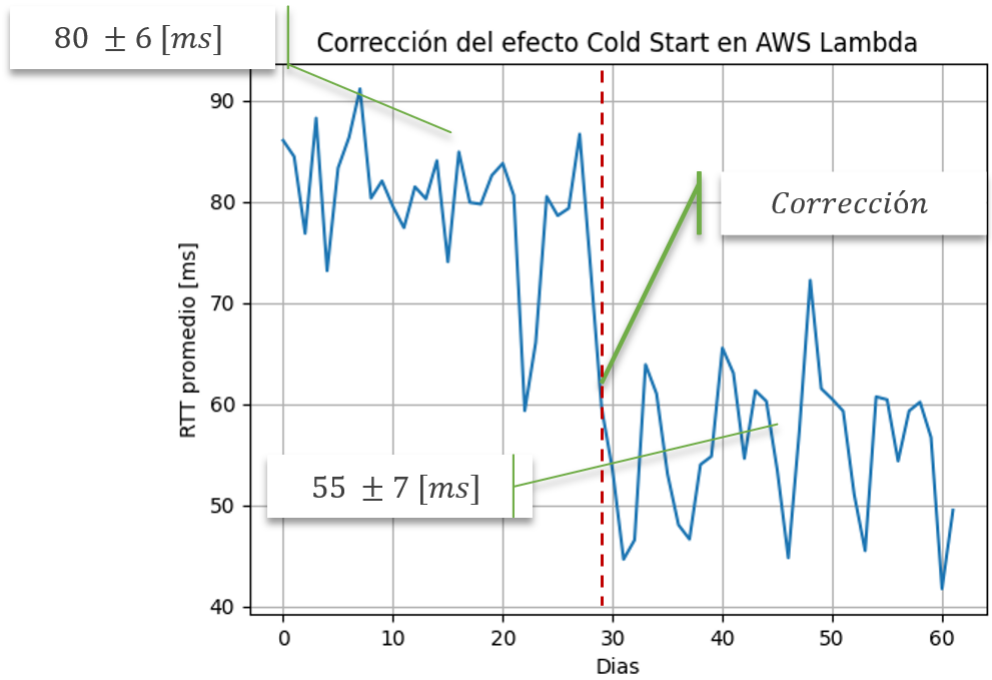

Figura 4.1: Corrección del efecto *cold start* para AWS Lambda "searchTransaction"

4.2. Resultados y Análisis Estadístico RTT

4.2.1. Formulación del Método Matemático

En AWS CloudWatch Metrics es posible obtener el promedio de las peticiones consideradas en una ventana de tiempo fija [1]. La muestra $R\hat{T}T_{\mathrm{i}}^{j}$ observada en un componente j cualquiera, corresponde al promedio de las observaciones en una ventana de tiempo de 5 segundos. Bajo el supuesto de que las observaciones i sobre un componente j siguen una distribución normal 4.1, cada componente puede caracterizarse mediante el promedio de las observaciones μ^j 4.2 y su desviación estándar σ^j 4.3. Entonces, calcular experimentalmente μ^j y σ^j es suficiente para caracterizar los tiempos de respuesta bajo un comportamiento normal.

$$R\hat{T}T_j \sim \mathcal{N}(\mu_j, \sigma_j) \tag{4.1}$$

$$\mu_j = \frac{1}{N}\sum_{\mathrm{i}=0}^{N} R\hat{T}T_j^{\mathrm{i}} \tag{4.2}$$

[1] También es posible obtener otros estadísticos desde la ventana de tiempo, como máximos, mínimos y percentiles.

$$\sigma_j = \frac{1}{N}\sqrt{\sum_{\mathrm{i}=0}^{N}\left(\hat{RTT}_j^{\mathrm{i}} - \mu_j\right)^2} \tag{4.3}$$

En adición, si se supone que el componente l se puede medir directamente, compuesto por subcomponentes medibles j, y otros k que no lo son , se puede modelar el error de los componentes k no observables $RTT^k \sim \mathcal{N}(\mu_k, \sigma_k)$ basándose en la expresión 4.4. Cuyo error considera elementos como el ancho de banda disponible en la red, el tiempo de transporte entre componentes y los tiempos de inicialización en frío. Como el componente l es posible medirlo y también los subcomponentes j, se puede despejar la expresión para la distribución del error $\mathcal{N}(\mu_k, \sigma_k)$ resultando 4.5. Finalmente, la media y la desviación del error se ve en las ecuaciones 4.6 y 4.7, respectivamente.

$$\hat{RTT}_l \sim \mathcal{N}(\mu_l, \sigma_l) = \mathcal{N}(\mu_k, \sigma_k) + \sum_{j=0, j\neq k}^{K} \mathcal{N}\left(\mu_j, \sigma_j\right) \tag{4.4}$$

$$\Rightarrow \mathcal{N}(\mu_k, \sigma_k) = \mathcal{N}(\mu_l, \sigma_l) - \sum_{j=0, j\neq k}^{K} \mathcal{N}\left(\mu_j, \sigma_j\right) \tag{4.5}$$

$$\mu_k = \mu_l - \sum_{j=0, j\neq k}^{K} \mu_j \tag{4.6}$$

$$\sigma_k = \sqrt{\sigma_l^2 - \sum_{j=0, j\neq k}^{K} \sigma_j^2} \tag{4.7}$$

4.2.2. Mediciones sobre el RTT

En la siguiente sección se exhibe una estadística descriptiva para cada uno de los componentes involucrados en la solución. Cada uno de los resultados presentados ha sido obtenido mediante el uso de AWS CloudWatch, específicamente empleando las opciones de métricas que se activaron durante la creación de los recursos respectivos. Es importante destacar que CloudWatch Metrics no proporciona datos en estado bruto. En su lugar, calcula automáticamente el promedio (u otro estadístico) dentro de una ventana de tiempo predefinida. Por lo tanto, cada muestra RTT_{i}^j, correspondiente al componente j, representa el promedio calculado dentro de una ventana temporal fija, que para estos resultados es de $\Delta t = 5$ minutos. No se dispone de muestras para los tiempos de ejecución para solicitudes independientes de cada componente.

Con base en lo expuesto anteriormente, es factible obtener los valores promedio RTT_{i} para un componente j específico y calcular el histograma, junto con diversos estadísticos: media, desviación estándar, mínimo, máximo y percentiles 25, 50, y 75. El histograma se puede

graficar con el fin de comprender empíricamente el tipo de distribución y proponer mejoras al sistema. En el anexo E se encuentran los estadísticos y las densidades de probabilidad para todos los componentes involucrados en la solución.

Por otro lado, de la ecuación 2.3 se puede considerar un componente que agrupe todos aquellos elementos no medidos, como el ancho de banda disponible, transporte entre componentes y los tiempos de inicialización de AWS. Así, el RTT de un componente posee un "error" de medición (ver ecuación 4.5), aún después de considerar los RTT de los componentes conocidos.

La tabla resumen 4.1 incluye los componentes que han sido habilitados como proveedores de servicios, y que son utilizados por Watson Assistant y API Manager. Posteriormente, cada uno de estos componentes puede ser estudiado de manera independiente en cuanto a los tiempos de respuesta (RTT) de sus subcomponentes.

En relación a la acción `obtenerDeuda`, `obtenerProducto` y `obtenerSaldo`, el tiempo está principalmente asociado al período que demanda la petición hacia el sistema interno de la empresa, el cual obtiene las deudas en línea. En cuanto las funciones Lambda que se ejecutan como subtareas de estas acciones, presentan un error promedio de $\mu_k = 238$ $[ms]$, respecto al RTT de cada Step Function. Además requieren invocar la función `obtenerToken` para autenticarse con los sistemas internos de la empresa, aunque su contribución es prácticamente despreciable, con un RTT de 5 ± 17 $[ms]$. Se puede atribuir este μ_k al entorno de ejecución AWS Step Function.

En cuanto a la acción `verificarCliente`, el tiempo de inicialización de la Step Function es $\mu_k = 62$ $[ms]$. Esto se puede explicar debido a que esta acción forma parte del perfilamiento inicial para todos los clientes que interactúan en el canal, por lo tanto tiene un consumo mayor, y al ser invocada en el flujo de recargas, ya existe un ambiente activo para atender la solicitud, expresando un menor tiempo de inicialización.

Para la acción `createTransaction`, utilizada por Watson Assistant, se observa en la tabla E.7 que la Step Function tiene un RTT de 283 ± 449 $[ms]$, lo cual resulta en promedio 4,15 veces más lento que la función Lambda que invoca ($RTT = 62 \pm 44$ $[ms]$). En cuanto a la consulta para insertar el registro en la base de datos, el tiempo es de 21 ± 2 $[ms]$.

A continuación, se presentan los estadísticos obtenidos para los servicios empleados por la pasarela de pagos. En primer lugar, se encuentra el Internal API Gateway, como se muestra en la tabla E.6, el cual posee un RTT de 526 ± 462 $[ms]$. Esto incluye conjuntamente todos los componentes conectados al *gateway*, por lo que resulta relevante estudiar de forma independiente las Step Function de búsqueda y actualización de los enlaces de pago. Cabe mencionar, que a partir de las gráficas obtenidas, el Intenal API Gateway E.11 no posee un comportamiento normal. Esto, debido a que las incorporadas consideran, en conjunto los servicios de búsqueda y actualización de los enlaces de pago.

El componente `searchTransaction` posee un $RTT = 321 \pm 465$ $[ms]$, lo cual es $5,2$ superior que el tiempo demorado por la lambda invocada, siendo este $RTT = 62 \pm 55$ $[ms]$. Otro componente involucrado en esta acción es la base de datos, en particular la consulta *ReadItem*, la cual demora 18 ± 3 $[ms]$.

Tabla 4.1: Resumen de los estadísticos para los principales componentes involucrados en la solución.

Componente l	$\mu_l \pm \sigma_l$ [ms]	$\mu_k \pm \sigma_k$ [ms]	**Consumidor**	**Subcomponentes**
VerificarCliente	1017 ± 265	62 ± 6	Watson Assistant	Lambdas: obtenerToken getMSISDN
ObtenerDeuda	1794 ± 513	235 ± 14	Watson Assistant	Lambdas: obtenerToken customerDebts
ObtenerSaldo	3593 ± 882	254 ± 14	Watson Assistant	Lambdas: obtenerToken getCustomerPrepayBalance getAvailableProductOffer getMinimalAssets
ObtenerProducto	1020 ± 624	224 ± 23	Watson Assistant	Lambdas: obtenerToken getMinimalAssets
Internal API Gateway	525 ± 461	NA	API Manager	Lambda: sendNotification
CreateTransaction	283 ± 450	149 ± 18	Watson Assistant	Lambda: createTransaction
SearchTransaction	321 ± 464	247 ± 21	AWS API Gateway	Lambda: searchTransaction
UpdateTransaction	954 ± 577	816 ± 23	AWS API Gateway	Lambda: updateTransaction
Notification API Gateway	70 ± 5	asíncrono	API Manager	SQS: FIFO Queue Lambda: sendNotification

El componente `updateTransaction` posee un $RTT = 946 \pm 576$ $[ms]$, esta vez demora en promedio $6,9$ veces mas que la lambda, siendo este de 137 ± 37 $[ms]$. El alza de esta consulta respecto a la creación y busqueda de enlaces de pago se explica porque, en este caso, se debe realizar una búsqueda y luego inserción del registro en la tabla. Debido a que esto no corresponde a un proceso nativo de DynamoDB, el RTT se ha modelado a partir de los resultados obtenidos para *PutItem* y *ReadItem*, ambos comportamientos han sido inferidos normales según las gráficas E.17 y E.20. Resultando, en la teoría, un $RTT = 42 \pm 5$ $[ms]$.

En cuanto a los tiempos para el envío de notificaciones, se ha medido que son inferiores a 100 milisegundos, con un valor exacto de 70 ± 5 $[ms]$. Esto se debe a que le petición se ejecuta de manera asíncrona, el Notification API Gateway responde inmediatamente un estado 200 a la consulta HTTP y posteriormente deposita los datos en una cola de mensajería SQS FIFO [61]. Luego los mensajes en la cola ejecutan la lambda `sendNotification` como eventos. De esta forma el consumidor del servicio puede continuar con el proceso y esperar que se envíe la notificación con el mejor esfuerzo posible.

4.3. Confiabilidad teórica

En la siguiente sección se presentan los cálculos teóricos para la confiabilidad de cada uno de los componentes implementados. AWS Lambda es el único componente cuya disponibilidad se aumenta utilizando dos zonas de disponibilidad, por lo tanto, al calcular la confiabilidad para una lambda, se debe tener en consideración esta configuración en paralelo, utilizando la ecuación descrita en 2.2.

Luego se debe tener en cuenta los elementos en serie para cada uno de los componentes construidos. La confiabilidad entregada por AWS se puede ver en la tabla D.1. Todos los demás componentes relacionados a las aplicaciones empresariales, como API Manager y también los sistemas proveedores de servicios, han sido considerados como un sistema confiable ideal (100 %).

Tabla 4.2: Confiabilidad de los componentes utilizados en la solución tecnológica.

Modulo	**Confiabilidad (%)**
obtenerProducto	99.8999
obtenerSaldo	99.8999
obtenerDeuda	99.8999
createTransaction	99.8999
searchTransaction	99.8
updateTransaction	99.8
sendNotification	99.8999
verificarCliente	99.8999

En la tabla 4.2 se presentan los resultados de las confiabilidades teóricas obtenidas para la configuración implementada en cada componente. En el caso de los servicios destinados a buscar el detalle de las transacciones, la confiabilidad se ve disminuida debido a la presencia del Internal API Gateway. Por otro lado, el servicio encargado de actualizar las transacciones se ve afectado por la existencia de dos consultas en serie hacia la base de datos.

En cuanto al proceso de pago de boleta de servicios, se requieren 5 peticiones a Watson Assistant, mientras que las recargas de saldo implican 8 peticiones hacia el mismo servicio. Considerando también los datos presentados en la tabla D.1, es factible calcular la confiabilidad teórica para cada uno de los casos de uso como un sistema en serie, debido a que cada uno de los componentes es ejecutado secuencialmente. Obteniendo valores de 98.7076 % y 98.5103 % para el pago de boleta de servicios y las recargas de saldo, respectivamente. Estos resultados indican que, teóricamente, el sistema no estará disponible durante un total de 9,43 y 10,87 horas mensuales, respectivamente.

4.4. Estimación de costos

Un usuario con la intención de realizar el pago de una boleta consumirá la totalidad de los recursos implicados en el proceso de pago. Suponiendo que, la solución propuesta debe ser ampliada para dar cobertura de pago a un millón de usuarios activos mensuales (*Monthly Active Users*, MAU). El costo total se puede modelar como la contribución de Watson Assistant, componentes de AWS para la solución, WhatsApp Business y el costo por la capa *Core*, tal como se ve en la ecuación 4.8. Es necesario recalcar que el costo del *Core* se puede considerar un costo fijo, puesto que es necesario costearlo incluso si no existen usuarios activos. Desde ahora, el costo del *Core* se ignorará para centrarse solo en los componentes de la solución propuesta.

$$C_{\text{total}} = C_{\text{Assistant}} + C_{\text{componentes}} + C_{\text{WhatsApp}} + C_{\text{Core}} \tag{4.8}$$

- El modelo de precios de WhatsApp Business API es por conversación, que puede ser iniciada por un usuario o por la empresa. En este proyecto, las conversaciones son inicializadas por el usuario, por lo que el costo corresponde al precio por conversación de servicio [2]. Para Chile, el costo por conversación de servicio es $0,0454 USD. En total, para un millón de MAU es $45.400 USD/Mensual.
- El costo para AWS debe ser cotizado mediante la calculadora en línea [3], el desglose de esta cotización se puede ver en la tabla F.1. Siendo en total, para el consumo esperado por un millón de MAU en total $35,91 USD/Mensual.
- En cuanto a Watson Assistant[4], este implica un costo inicial de $140 USD, con un límite de 1.000 MAU y un costo adicional de $14 USD por cada bloque adicional de 1.000

[2] `https://developers.facebook.com/docs/whatsapp/pricing/#tarifas`
[3] `https://calculator.aws`
[4] `https://www.ibm.com/products/watson-assistant/pricing`

MAU. Por lo tanto, el costo total por el uso de Watson Assistant, correspondiente al manejo de un millón de MAU, sería de $140.000 USD.

Finalmente, el costo total por operar un millón de MAU en el canal es $ 185.431 USD/-Mensual. Equivalente a un costo unitario por MAU de $0,1854 USD/MAU

Capítulo 5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

El presente informe de título ha abordado exhaustivamente el diseño, la implementación y el análisis de una solución tecnológica para un *chatbot* en el ámbito de las telecomunicaciones. Con ello, la empresa cuenta actualmente con un nuevo medio para la recaudación financiera en los procesos comerciales de pago de servicios móviles y hogar, así como para las recargas de saldo mediante pago digital. Además, esta solución proporciona una experiencia novedosa a los usuarios a través del canal de WhatsApp. En la actualidad, el *chatbot* se encuentra en pleno funcionamiento y los usuarios pueden hacer uso de sus funcionalidades. La elección de tecnologías basadas en AWS ha sido acertada, permitiendo cumplir satisfactoriamente con los requerimientos funcionales de cada caso de uso.

Los principales desafíos en este trabajo están relacionados con el diseño. El primer desafío consistió en escoger adecuadamente a qué sistemas empresariales se debe consultar la información de los clientes, lo cual se logró involucrando a los ingenieros especialistas en los sistemas internos de la empresa de telecomunicaciones. Un segundo desafío fue especificar la API y los datos a almacenar para la generación, consulta y actualización de los enlaces de pago, lo cual no se encontraba disponible en la empresa. Para superar este desafío ha sido necesario revisar exhaustivamente documentación de los servicios de AWS y su implementación mediante el entorno de ejecución NodeJS, siguiendo tres prácticas de desarrollo: control de versiones, documentación y pruebas unitarias automáticas.

En relación a la confiabilidad, se ha determinado que el nivel mínimo de servicio garantizado para la solución propuesta es del 98.7076 % y 98.5103 % para el pago de boletas y recargas de saldo, respectivamente. Si bien es posible que pueda resultar sorprendente el hecho de que el sistema no esté disponible durante un máximo de 10 horas al mes, es crucial tener en cuenta que los valores representan la *confiabilidad mínima garantizada* por el proveedor de servicios, AWS.

Antes de este trabajo, no se disponía de datos empíricos acerca de los tiempos de respuesta (RTT) del *chatbot*. Esto ha habilitado una nueva forma de análisis para detectar posibles mejoras en la arquitectura del sistema. En relación a las acciones utilizadas por Watson

Assistant, es importante mencionar que es tolerable la demora de unos segundos, sin que ello afecte la experiencia del usuario. Esto se debe a que la naturaleza asincrónica del canal de WhatsApp permite que el *chatbot* pueda tardar algunos segundos para responder al usuario. Incluso, el usuario podría responder varios minutos después y el *chatbot* de igual forma retomará la conversación. No obstante, es fundamental no llevar al sistema al extremo de demorar la ejecución de acciones automáticas durante varios minutos, ya que la percepción del usuario debe asemejarse lo máximo posible al tiempo que experimentaría en una conversación en línea entre seres humanos.

Respecto a los servicios utilizados por la pasarela de pagos, el tiempo de respuesta (RTT) resulta relevante debido a que el usuario se encuentra en el navegador web y no puede tolerar demoras prolongadas al cargar una página. Este tipo de comportamientos pueden generar frustración en el usuario y afectar la recaudación financiera. Sin embargo, cabe resaltar que este problema no se ha presentado en la implementación del proyecto, dado que los tiempos promedio de ejecución siguen siendo aceptables, y se han aplicado estrategias para mitigar el efecto *cold start* de AWS Lambda.

Finalmente, se ha medido experimentalmente que la implementación de este avance tecnológico en el canal ha permitido que un 13,35 % y 19,39 % de los enlaces de pago generados se cursen como pago efectivo, con una tasa de fallos del 3,41 % y 2,89 %, para recargas de saldo y pago de boleta de servicios, respectivamente. Si bien la tasa de fallos es superior a la confiabilidad teórica calculada, esta desviación se puede atribuir a que los sistemas empresariales se han considerado "ideales" para todo efecto en el calculo.

5.2. Trabajos Futuros

A continuación se proponen algunos trabajos futuros para mejorar el sistema.

Si se requiere un nivel de servicio aún más elevado, se deberá considerar la adopción de otras soluciones, como contar con instalaciones en varias zonas geográficas de la infraestructura (por ejemplo, us-east-1 y us-east-2). Es importante mencionar que esto conllevará un aumento en los costos asociados a la instalación de la capa *Core* y la adición de más componentes para manejar instalaciones en dos regiones separadas. La solución propuesta se destaca por presentar una ventaja significativa en cuanto a disponibilidad y costos asociados con AWS, sin implicar un incremento en la capa *Core*.

Se podría obtener una mejora al prescindir de AWS Step Function para los servicios de búsqueda y actualización de enlaces de pago. Los flujos de trabajo que ejecutan las Step Function están compuestos cada uno por una sola función y es técnicamente factible conectar dicha función directamente en el Internal API Gateway, quitando la Step Function del medio, lo que permitiría reducir el RTT según el margen de error calculado en 247 ± 21 (5,1 veces mas rápido) y 816 ± 23 (6,9 veces más rápido) milisegundos, para las operaciones de búsqueda y actualización de enlaces de pago, respectivamente. Esto también aumentaría la confiabilidad, aunque marginal, en un 0,19 %, o el equivalente a 1,44 horas mensuales adicionales de disponibilidad.

En busca de mejorar aún más el tiempo de respuesta (RTT) para el servicio que actualiza el enlace de pago, se debe mencionar que esta se compone de una petición a DynamoDB que involucra tanto *ReadItem* como *PutItem*. Para optimizar el código fuente, es factible realizar únicamente la petición *PutItem*, gestionando adecuadamente las excepciones en caso de que el registro no exista en la base de datos y retornando esta información al consumidor. Esta modificación permitiría agilizar el proceso y contribuir a una experiencia más fluida para el usuario. Disminuyendo el RTT en 18 ± 2 $[ms]$, el equivalente a una consulta *ReadItem*.

En relación a los costos en el entorno de AWS, se ha constatado que son bajos en comparación con el volumen de transacciones propuesto. Para el caso de WhatsApp, los costos presentados corresponden a los precios de lista. Una alternativa potencial para reducir costos sería establecer contacto con algún BSP que disponga de precios preferenciales con Meta, lo que permitiría acceder a economías de escala. Sin embargo, es importante mencionar que esta posibilidad se encuentra sujeta a acuerdos contractuales entre la empresa y el BSP, aspectos que no han sido considerados en el alcance de este trabajo.

Por ende, los costos que podrían ser objeto de disminución corresponden a la plataforma de Inteligencia Artificial Conversacional: Watson Assistant. Como ejercicio, una cotización del servicio AWS Amazon Lex, para la cantidad de solicitudes requeridas en este proyecto, resulta en un costo de $10,000 USD por cada millón de MAU. Esto equivale a un valor de $0.055435 USD/MAU o $45,45 CLP/MAU [1]. No obstante, es relevante destacar que para lograr esta eficiencia en los costos, sería necesario actualizar el *Core* de todo el sistema, lo que implicaría costos de inversión relacionados con horas de desarrollo y los riesgos asociados a una actualización de esta envergadura.

[1] Valor del dólar en pesos chilenos al 23 de Julio 2023: $816.48

Bibliografía

[1] I. H. Media, *Digital 2022 Global Overview Report*, 2022. dirección: https://hootsuite.widen.net/s/kd6qgn9rwx/digital2022globaloverview_report_en.

[2] U. de Estudios Subtel – División Política Regulatoria y Estudios, *Evolución Internet Fija: Diciembre 2017 a Diciembre 2021*, 2022. dirección: https://www.subtel.gob.cl/wp-content/uploads/2022/03/PPT_Series_DICIEMBRE_2021_V2.pdf.

[3] S. de Telecomunicaciones de Chile, *Derechos de los Usuarios de Telecomunicaciones*, 2015. dirección: https://www.subtel.gob.cl/15derechos/.

[4] A. M. Turing, *Computing Machinery and Intelligence*. Dordrecht: Springer Netherlands, 2009, ISBN: 978-1-4020-6710-5. DOI: 10.1007/978-1-4020-6710-5_3.

[5] S. D. A. Tulshan, *Survey on Virtual Assistant: Google Assistant, Siri, Cortana, Alexa*. Singapore: Springer Singapore, 2019, págs. 190-201, ISBN: 978-981-13-5758-9. DOI: 10.1007/978-981-13-5758-9_17.

[6] W. LLC, *About WhatsApp*, 2023. dirección: https://www.whatsapp.com/about/.

[7] A. PLC, *WhatsApp, un canal clave en las compras del retail en Chile*, Report, 2020. dirección: https://www.anda.cl/wp-content/uploads/2020/09/Estudio-Chile-Whatsapp-OK.pdf.

[8] IBM, *IBM Watson Assistant Virtual Agent*, 2023. dirección: https://www.ibm.com/products/watson-assistant.

[9] I. Amazon.com, *Amazon Web Services*, 2023. dirección: https://aws.amazon.com.

[10] G. LLC, *Cloud Computing Services: Google Cloud Platform*, 2023. dirección: https://cloud.google.com.

[11] M. Corporation, *Cloud Computing Services: Microsoft Azure*, 2023. dirección: https://azure.microsoft.com.

[12] N. F. M. Richards, *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2020, pág. 426, ISBN: 9781492043454. DOI: 10.1007/978-1-492-04345-4.

[13] A. Barashkov, *Microservices vs. Monolith Architecture*, 2018. dirección: https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m.

[14] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Professional, 2012, ISBN: 9780133065213.

[15] J. R. R. T. Fielding M. Nottingham, *RFC 9110: HTTP Semantics*, 2022. dirección: https://www.rfc-editor.org/info/rfc9110.

[16] J. A. Nachlas, *Reliability Engineering, Probabilistic Models and Maintenance Methods, Second Edition*. CRC Press, mar. de 2017, págs. 1-394, ISBN: 9781315307596. DOI: 10.1201/9781315307596.

[17] T. ANDREW S y W. DAVID J, *COMPUTER NETWORKS FIFTH EDITION*, 2011.

[18] M. P. Inc., *WhatsApp Business Management API*, 2018. dirección: https://developers.facebook.com/docs/whatsapp/business-management-api.

[19] W. LLC, *What are Business Solution Providers and how can you work with them?: WhatsApp Help Center*, 2023. dirección: https://faq.whatsapp.com/695500918177858/?helpref=uf_share&cms_id=695500918177858.

[20] M. P. Inc., *Overview: WhatsApp On-Premise API*, 2018. dirección: https://developers.facebook.com/docs/whatsapp/on-premises/overview.

[21] M. P. Inc., *Overview: WhatsApp Cloud API*, 2018. dirección: https://developers.facebook.com/docs/whatsapp/cloud-api.

[22] A. M. Turing, *I. Computing Machinery and Intelligence*, 1950. DOI: 10.1093/mind/LIX.236.433.

[23] M. P. Inc., *WhatsApp Business Platform*, 2018. dirección: https://developers.facebook.com/docs/whatsapp?locale=en_US.

[24] M. for Developers, *Messenger Platform*, 2023. dirección: https://developers.facebook.com/docs/messenger-platform.

[25] T. LLC, *Bots: An introduction for developers*, 2016. dirección: https://core.telegram.org/bots.

[26] B. E. M. Revang A. Mullen, *Magic Quadrant for Enterprise Conversational AI Platforms*, 2022. dirección: https://www.gartner.com/en/documents/4154599.

[27] G. Research, *Magic Quadrant for Cloud Infrastructure and Platform Services*, 2022. dirección: https://www.gartner.com/en/documents/4020235.

[28] G. LLC, *Dialogflow: Google Cloud*, 2023. dirección: https://cloud.google.com/dialogflow?hl=es-419.

[29] A. Inc., *AWS: Conversational AI and Chatbots*, 2023. dirección: https://aws.amazon.com/lex/?nc1=h_ls.

[30] M. Corporation, *Microsoft Bot Framework*, 2019. dirección: https://dev.botframework.com/.

[31] e. a. N. Sabharwal, *Building Your First Bot Using Watson Assistant*, Berkeley, CA, 2019. DOI: 10.1007/978-1-4842-5555-1_4.

[32] IBM, *Dialog Skill*, 2023. dirección: https://cloud.ibm.com/docs/assistant?topic=assistant-skill-add#skill-add-dialog-skill.

[33] T. Eloundou, S. Manning, P. Mishkin y D. Rock, *"GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models"*, mar. de 2023. DOI: 10.48550/arXiv.2303.10130.

[34] e. a. J. Devlin, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, oct. de 2018. DOI: 10.48550/arXiv.1810.04805.

[35] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu y M. Chen, «Hierarchical text-conditional image generation with clip latents,» *arXiv preprint arXiv:2204.06125*, 2022.

[36] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. dirección: http://probml.github.io/book2.

[37] Q. .-. P. a Personal Wealth Movement, *Microsoft Confirms Its $10 Billion Investment Into* ChatGPT*: Changing How Microsoft Competes With Google, Apple And Other Tech Giants*, ene. de 2023. dirección: https://www.forbes.com/sites/qai/2023/01/27/microsoft-confirms-its-10-billion-investment-into-chatgpt-changing-how-microsoft-competes-with-google-apple-and-other-tech-giants/?sh=151122f33624.

[38] T. G. P. Mell, *The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology*, 2011. DOI: 10.6028/NIST.SP.800-145.

[39] A. Inc., *AWS: Secure and Resizable Cloud Compute*, 2023. dirección: https://aws.amazon.com/ec2.

[40] A. Inc., *AWS: Amazon ElastiCache*, 2023. dirección: https://aws.amazon.com/es/elasticache.

[41] P. Sbarski, *Serverless Architecture on AWS: Architectures and Patterns*. Manning Publications, 2017, ISBN: 9781617293825.

[42] L. M. D. Jaime K. Hamzeh, *Application Deployment Strategies for Reducing the Cold Start Delay of AWS Lambda*, 2022. DOI: 10.1109/CLOUD55607.2022.00016.

[43] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, 2019, ISBN: 9781492047810.

[44] A. Inc., *AWS: Customer Success Stories: Case Studies, Videos, Podcasts, Innovator stories*, 2023. dirección: https://aws.amazon.com/solutions/case-studies.

[45] A. Inc., *AWS: Serverless Computing*, 2023. dirección: https://aws.amazon.com/lambda.

[46] A. Inc., *AWS: Fast NoSQL Key-Value Database*, 2023. dirección: https://aws.amazon.com/dynamodb.

[47] A. Inc., *AWS: Serverless Workflow Orchestration Services*, 2023. dirección: https://aws.amazon.com/step-functions.

[48] A. Inc., *AWS: Amazon API Gateway*, 2023. dirección: https://aws.amazon.com/es/api-gateway/.

[49] A. Inc., *AWS: Logically Isolated Virtual Private Cloud*, 2023. dirección: https://aws.amazon.com/vpc.

[50] A. Inc., *AWS: Access Management- AWS Identity and Access Management -IAM*, 2023. dirección: https://aws.amazon.com/iam.

[51] A. Inc., *AWS: Application and Infrastructure Monitoring*, 2023. dirección: https://aws.amazon.com/cloudwatch.

[52] T. S. Committee, *NodeJS Project*, 2023. dirección: https://nodejs.org/en.

[53] IBM, *IBM Cloud API Docs, Watson Assistant v1*, 2023. dirección: https://cloud.ibm.com/apidocs/assistant-v1.

[54] R. Ltd., *Redis*, 2023. dirección: https://redis.io/.

[55] P. J. Leach, R. Salz y M. H. Mealling, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC 4122, 2005. DOI: 10.17487/RFC4122. dirección: https://www.rfc-editor.org/info/rfc4122.

[56] A. W. Services, *Lambda runtimes*, 2023. dirección: https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html.

[57] Bitbucket, *Git solution for teams using Jira*, 2023. dirección: https://bitbucket.org/product/.

[58] M. Mathews y Contrubutors, *Use JSDoc*, 2023. dirección: https://jsdoc.app/.

[59] *JestJS*, 2023. dirección: https://jestjs.io/.

[60] A. Inc., *AWS: Configuring reserved concurrency*, 2023. dirección: https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html.

[61] A. Inc., *JestJS*, 2023. dirección: https://aws.amazon.com/es/sqs/.

Anexo A

Diagramas de flujo para los casos de uso

En el presente anexo se describe con detalle el paso a paso para los dos casos de uso: pago de boletas de servicios móvil y hogar A.1 y, recargas de saldos con pago digital A.2.

A.1. Pago de boletas de servicios móvil y hogar

1. El dialogo comienza cuando el cliente escribe cualquier frase en el canal de WhatsApp.
2. En este punto ya se cuenta con la información personal del cliente y su perfil de facturación, mas no así con las deudas pendientes, en caso que las hubiera.
3. El cliente manifiesta la intención para pagar su boleta mediante una frase o mediante. alguna opción de menú. Por otro lado, el cliente puede iniciar la conversación manifestando directamente la intención del pago de su deuda. Por ejemplo, "quiero pagar mi boleta".
4. Se obtiene la deuda actual del cliente agrupado por móvil y hogar.
5. Si el cliente no tiene deudas pendientes, se debe informar adecuadamente al mismo. En caso contrario, continuar con el flujo.
6. El *chatbot* debe mostrar las deudas pendiente del cliente, desglosado por servicios móvil y hogar. Si el cliente presenta mas de una boleta pendiente por estos servicios, el monto desplegado debe ser la suma de los montos pendientes de pago por cada uno de los servicios. El pago total también debe ser una opción.
7. El cliente confirma su intención de pago seleccionando: La deuda para móvil, hogar o el pago total. Como resultado, el *chatbot* debe generar el enlace de pago de acuerdo a la deuda seleccionada, derivando a la web de pagos. En caso contrario, el proceso debe finalizar.
8. El cliente en la web de pago debe poder ver el monto seleccionado de pago, para poder continuar con el proceso.

9. Si el pago se realiza de forma exitosa, la web debe notificar al usuario por WhatsApp que el proceso se completó exitosamente, caducando el enlace de pago.

10. Si el proceso no se pudo realizar por causa de alguna falla, se debe notificar al usuario.

A.2. Recargas de saldo

1. El dialogo comienza cuando el cliente escribe cualquier frase en el canal de WhatsApp. En este punto ya se cuenta con la información personal del cliente.

2. El cliente manifiesta la intención de recargar un numero móvil (desde ahora móvil destino) mediante alguna frase o accediendo directamente desde alguna opción. El *chatbot* debe ser capaz de reconocer y extraer directamente desde la frase del usuario tanto el número como el monto de la recarga. Por ejemplo, "Quiero recargar el numero 56912345678 con $1000."

3. En caso que el cliente no indique el móvil destino a recargar, el *chatbot* debe consultar al usuario y obtener desde la respuesta del usuario, el numero móvil a recargar.

4. Si el móvil destino pertenece a la empresa, se debe validar que el mercado del cliente esté habilitado para recibir saldo. Esto siempre debe aplicar para el mercado de clientes prepago.

5. En caso que el cliente no indique el monto de la recarga, el *chatbot* debe consultar al usuario y extraer de la respuesta el monto a recargar.

6. El *chatbot* debe solicitar al cliente que confirme los datos (monto y móvil). El usuario también debe tener la opción de cambiar el monto o el numero.

7. Si el cliente confirma los datos de la recarga, entonces el *chatbot* genera y envía al usuario el enlace de pago. Considerando los datos de comercio en la tabla A.1,

8. Si el pago se realiza exitosamente, la web de recargas notifica al cliente la transacción exitosa.

9. Si el pago no se realiza exitosamente, el usuario debe tener la opción de finalizar la atención, reintentar o ser transferido a un ejecutivo comercial.

Tabla A.1: Configuraciones para habilitar como comercio al *chatbot* en el caso de uso recargas de saldo con pago digital.

Código Integrador	**Integrador**	**Código distribuidor**	**Distribuidor**	**Código comercio**	**Comercio**
51	TRANSBANK	45	WEBPAY	300	CHATBOT
51	TRANSBANK	576	BANCO ESTADO	300	CHATBOT

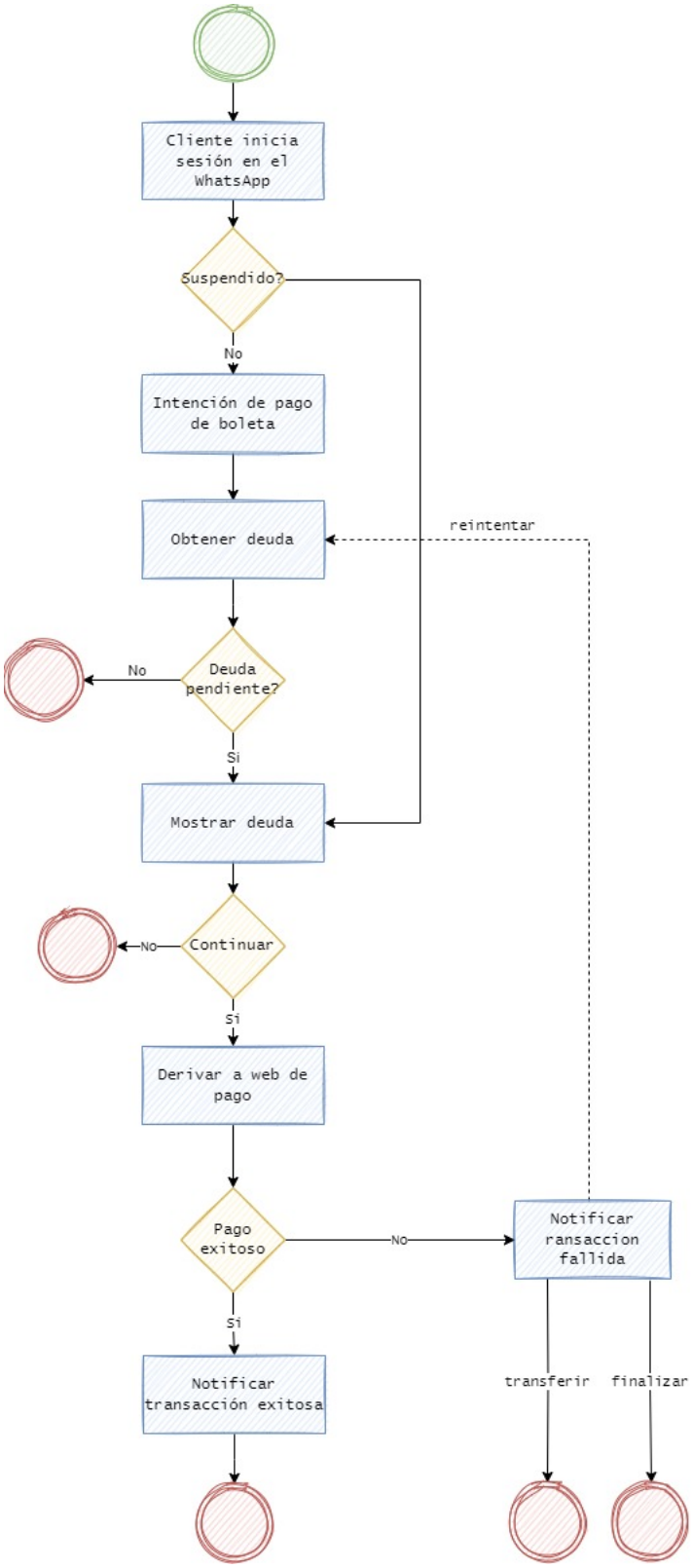

Figura A.1: Proceso Comercial para el pago de boletas de servicios a través de WhatsApp.

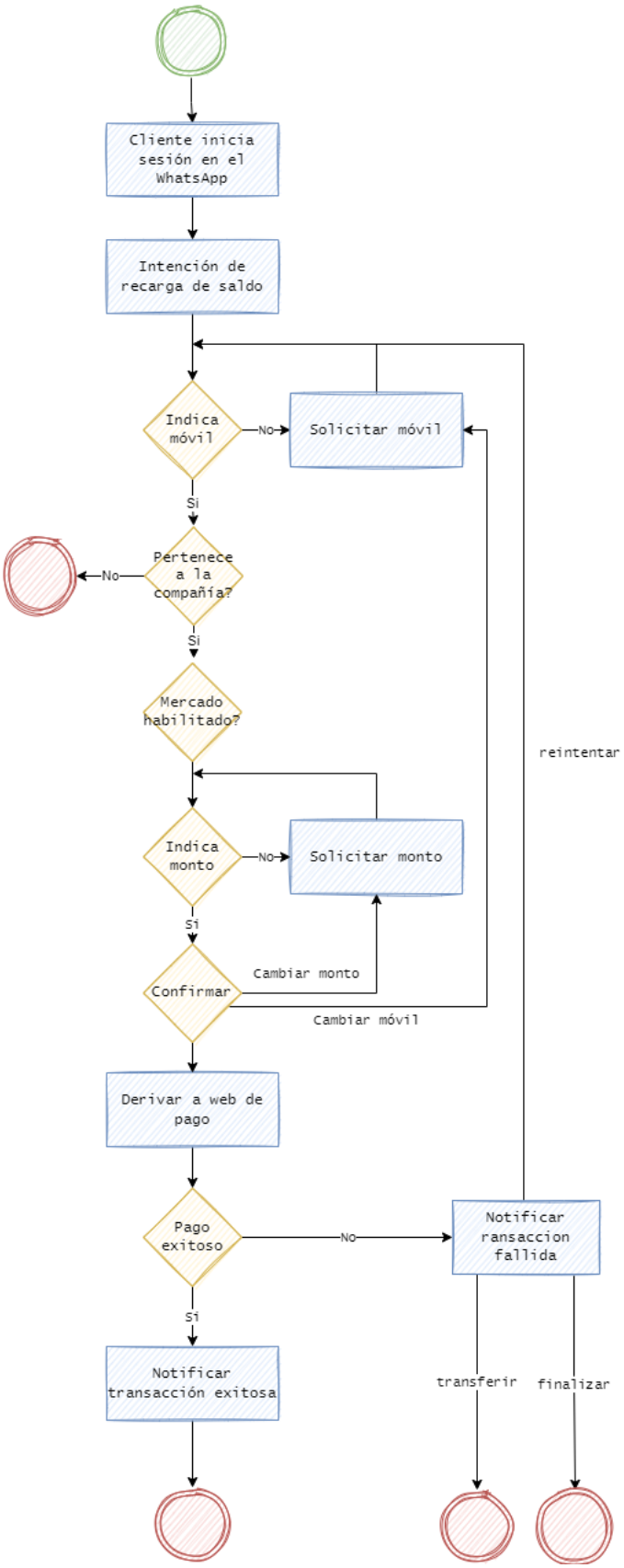

Figura A.2: Diagrama del proceso comercial para el pago de recargas de saldo en el *chatbot*

Anexo B

Diagramas de Secuencia

En el siguiente anexo se presentan los diagramas de secuencia para la solución diseñada, estas secuencias permite conocer en que orden y bajo que condiciones se deben invocar los componentes involucrados. Para una mejor comprensión se ha dividido la secuencia de integración en dos partes: antes y después de la generación del enlace de pago. Así, se tiene una secuencia de invocación diferente para el pago de boleta y las recargas de saldo, en cuanto la integración con la pasarela de pagos es transversal a los dos casos de uso.

En el diagrama de secuencia B.1 para el pago de boleta de servicios móvil, están involucrados los siguientes componentes: Watson Assistant, AWS Step Function, AWS Lambda y API Manager.

En el diagrama de secuencia B.1 para las recargas de saldos, están involucrados los siguientes componentes: Watson Assistant, AWS Step Function, AWS Lambda y API Manager.

El último diagrama de secuencia B.3 contempla la integración con la pasarela de pagos, en la cual están involucrados los siguientes componentes: Pasarela de pagos, API Manager, AWS Step Function, AWS Lambda y Watson Assistant.

Notar que a diferencia de los diagramas B.1 y B.2, el usuario interactúa primero con la pasarela de pagos, la cual obtiene y actualiza la información utilizando los servicios indicados. En cuanto Watson Assistant tiene participación solo cuando se finaliza el flujo para continuar con la atención en caso el usuario lo requiera.

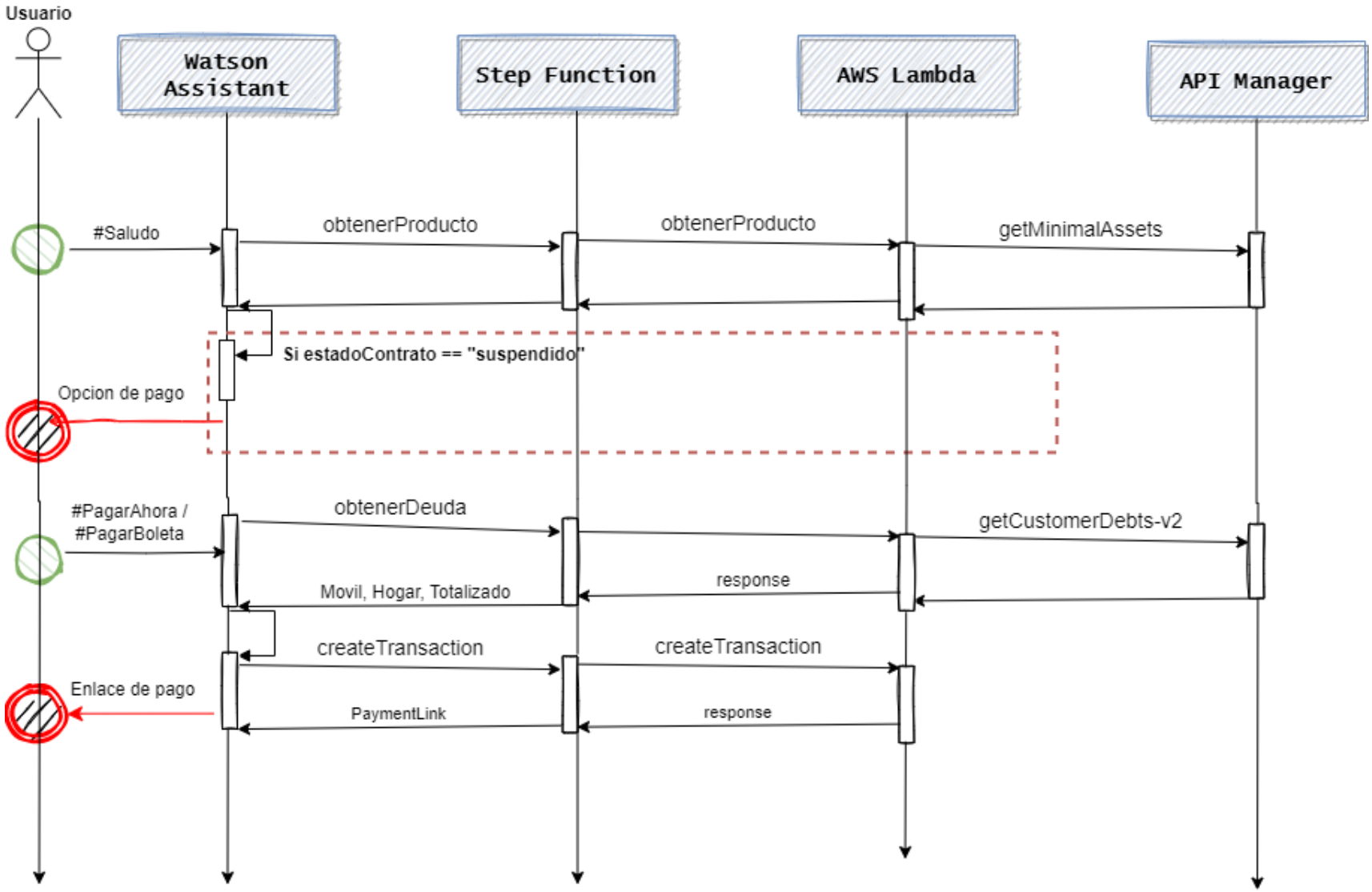

Figura B.1: Parte 1 de la secuencia de invocaciones para pago de boleta de servicios.

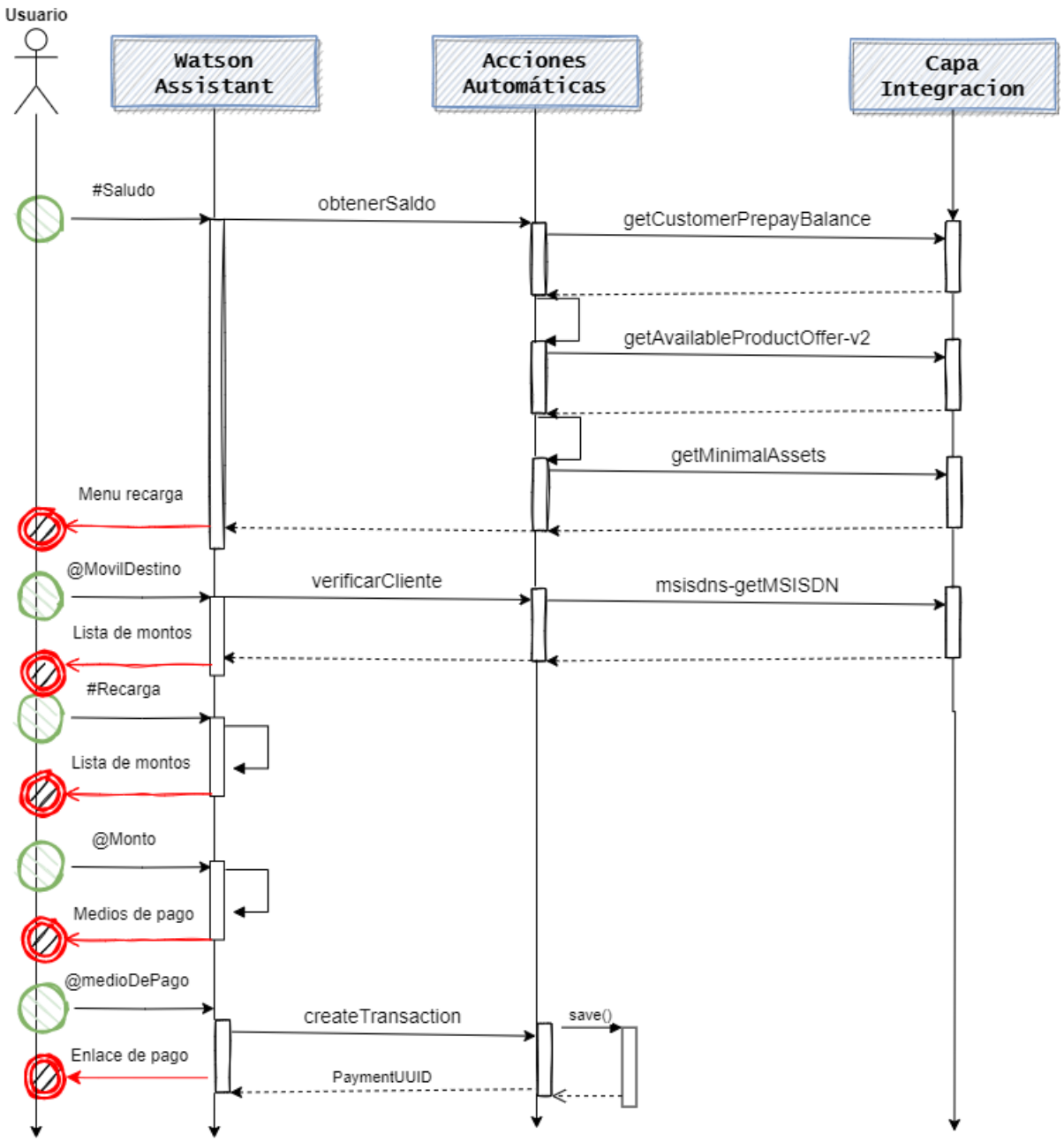

Figura B.2: Secuencia de invocaciones para recargas de saldo con pago digital.

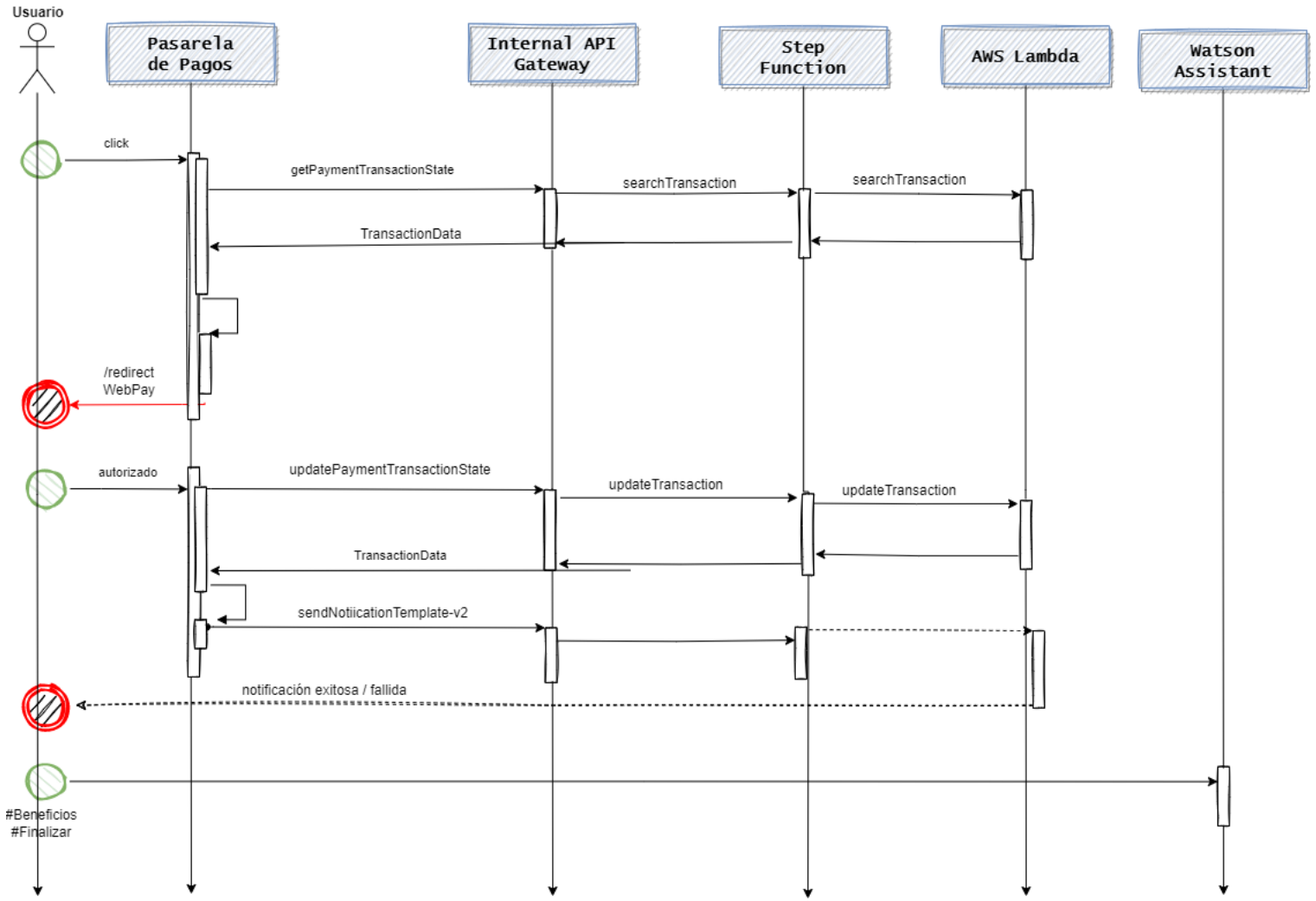

Figura B.3: Diagrama de secuencia de invocaciones que lleva a cabo la pasarela de pagos.

Anexo C

Especificaciones técnicas de los componentes creados

La arquitectura de los componentes involucrados en la solución implementada es representada por la figura C.1, en la cual cada una de las tecnologías involucradas y la relación que existe entre ellas son indicadas. Es importante destacar que la dirección de las flechas indica que la solicitud siempre es realizada desde el cliente hacia el servidor.

En lo que respecta a la creación de los enlaces de pago, las especificaciones para la Step Function encargada de la creación de transacciones de pago pueden ser observadas en los datos C.1. Las configuraciones para la Lambda encargada de crear las transacciones de pago pueden ser vistas en los datos C.2. Los datos de entrada para la creación de un enlace de pago del tipo boleta pueden ser visualizados en los datos C.3. Los datos de entrada para la creación de un enlace de pago de tipo recargas se pueden observar en los datos C.4. Los datos que resuelven la acción para crear transacciones de pago, y que también aplican para la búsqueda y actualización del enlace, se encuentran en los datos C.5.

En lo que respecta a la búsqueda de los enlaces de pago, la máquina de estados implementada en Step Function se muestra en los datos C.6. La definición de la Lambda encargada de buscar los enlaces de pago se puede ver en los datos C.7. Los datos de entrada para buscar una transacción de pago se pueden encontrar en C.8, mientras que la respuesta se halla en C.5.

Con respecto a la actualización de los enlaces de pago, la definición de la máquina de estados se muestra en los datos C.9. Las configuraciones relacionadas con la lambda encargada de actualizar los enlaces de pago en la base de datos se pueden encontrar en los datos C.10. Los datos de entrada necesarios para llevar a cabo una actualización de un enlace de pago se presentan en C.11, mientras que los datos de respuesta se encuentran en C.5.

En lo que respecta a la base de datos, un ejemplo de registro insertado en la base de datos DynamoDB para un enlace de tipo boleta de servicio se muestra en los datos C.13. Un ejemplo similar de registro insertado en la base de datos DynamoDB, esta vez para un enlace de tipo recarga, se presenta en los datos C.13. Los detalles de configuración de la tabla `transaction-state`, que almacena los enlaces de pago, se pueden encontrar en C.14.

En cuanto a la obtención de las deudas, la definición de la máquina de estados AWS Step Function se presenta en la figura C.15. Las configuraciones relacionadas con la función Lambda `obtenerDeuda` se pueden apreciar en los datos C.16.

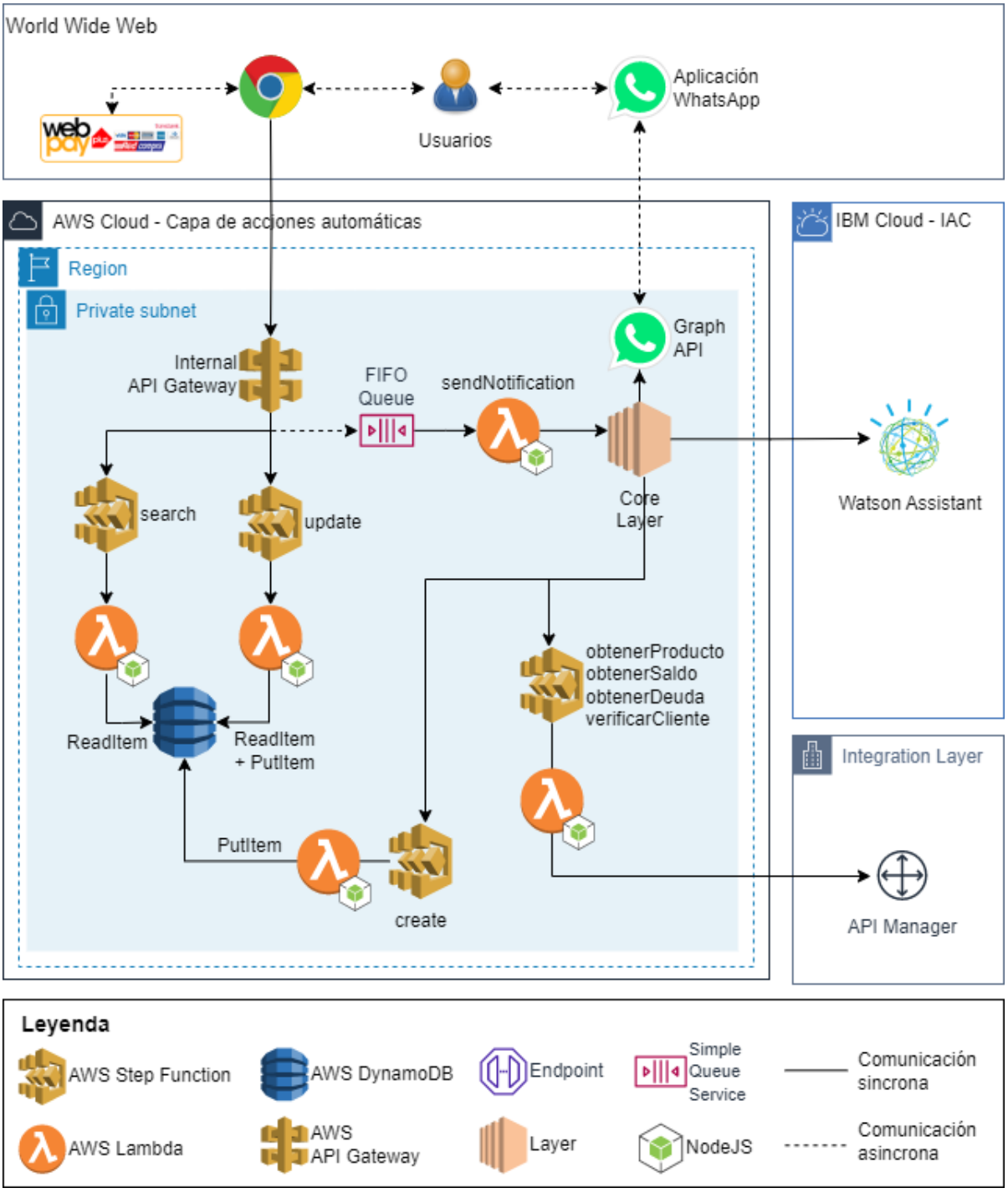

Figura C.1: Componentes de arquitectura para la solución tecnológica.

```
{
  "definition": {
    "Comment": "Crear ordenes para pagos y recargas",
    "StartAt": "_createTransaction",
    "States": {
      "_createTransaction": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-createTransaction",
        "End": true
      }
    }
  },
  "type": "EXPRESS",
  "loggingConfiguration": {
    "level": "ALL",
    "includeExecutionData": true,
    "destinations": [
      {
        "cloudWatchLogsLogGroup": {
          "logGroupArn": "arn:aws:logs:<region>:<accountId>:log-group:
    ↪ /aws/vendedlogs/states/chatbot-<environment>-createTransaction-Logs:*"
        }
      }
    ]
  }
}
```

Instrucciones C.1: Definición de la máquina de estados de AWS Step Function para createTransaction.

```
{
  "definition": {
    "FunctionName": "chatbot-<environment>-createTransaction",
    "FunctionArn": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-createTransaction",
    "Runtime": "nodejs14.x",
    "Role": "arn:aws:iam::<accountId>:role/chatbot-DynamoLogs",
    "Handler": "index.js",
    "Description": "",
    "Timeout": 60,
    "MemorySize": 512,
    "Version": "$LATEST",
    "Environment": {
      "Variables": {
        "AWS_LAMBDA_INITIALIZATION_TYPE": "provisioned-concurrency"
      }
    },
    "TracingConfig": {
      "Mode": "PassThrough"
```

```
    },
    "Layers": [
      {
        "Arn":
    ↪ "arn:aws:lambda:<region>:<accountId>:layer:<sourceCode>:<layerVersion>",
      }
    ],
    "VpcConfig": {
      "SubnetIds": [
        "<subNetAZ1>",
        "<subNetAZ2>"
      ],
      "SecurityGroupIds": [
        "<securityGroupId>"
      ],
      "VpcId": "<vpcId>"
    },
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip",
    "Architectures": [
      "x86_64"
    ],
    "EphemeralStorage": {
      "Size": 512
    },
    "SnapStart": {
      "ApplyOn": "None",
      "OptimizationStatus": "Off"
    }
  }
}
```

Instrucciones C.2: Definición de la lambda para createTransaction.

```
{
    "relatedPartyRef": {
        "id": "<RUT>",
        "msisdn": "<MSIDN>"
    },
    "channel": "BOT",
    "type": "billing",
    "characteristic": [
        { "key": "applicationName", "value": "CHATBOT" },
        { "key": "contractID", "value": "<productContractId>" },
        { "key": "society", "value": "TELECOM" }
  ]
}
```

Instrucciones C.3: Datos de entrada para generar enlaces para el pago de boleta de servcicios.

```
{
    "relatedPartyRef": {
        "id": "<RUT>",
        "msisdn": "<MSISDN>"
    },
    "channel": "BOT",
    "type": "refillment",
    "characteristic": [
        { "name": "msisdn", "value": "<MSISDN DESTINO>" },
        { "name": "amount", "value": "<MONTO RECARGA>" },
        { "name": "paymentMethodId", "value": "WPY | BES" },
        { "name": "integrationCode", "value": "300" },
        { "name": "distributionCode", "value": "45 | 576" },
        { "name": "commerceCode", "value": "300" },
        { "name": "commerceName", "value": "CHATBOT},
        { "name": "integrationId", "value": 51 },
    ]
}
```

Instrucciones C.4: Datos de entrada para generar enlaces de pago para recargas de saldos.

```
    {
        "createdAt": "2023-07-24T04:00:46.357Z",
        "reason": "Created successfully",
        "individualIdentification": {
            "RUT": "<RUT>",
            "MSISDN": "<MSISDN>"
        },
        "context": <RefillContextList> | <BillingContextList>,
        "id": "bc4fa2cf-c259-4ce3-8923-4e6fddc8a6fd",
        "state": {
            "current": "CREATED",
            "previous": "CREATED",
            "history": [
                "CREATED"
            ]
        },
        "paymentLink": "https://telecom.cl?q=bc4fa2cf-c259-4ce3-8923-4e6fddc8a6fd",
        "paymentBaseURL": "https://telecom.cl",
        "type": "billing",
        "ttl": 1690173046341,
        "consumer": "BOT-PRS",
        "updatedAt": "2023-07-24T04:00:46.357Z"
    }
```

Instrucciones C.5: Datos de salida para la creacion, busqueda y actualización de enlaces de pago

```
{
  "definition": {
```

```
    "Comment": "Buscar ordenes para pagos y recargas",
    "StartAt": "_searchTransaction",
    "States": {
      "_searchTransaction": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-searchTransaction",
        "End": true
      }
    }
  },
  "type": "EXPRESS",
  "loggingConfiguration": {
    "level": "ALL",
    "includeExecutionData": true,
    "destinations": [
      {
        "cloudWatchLogsLogGroup": {
          "logGroupArn": "arn:aws:logs:<region>:<accountId>:log-group:
    ↪ /aws/vendedlogs/states/chatbot-<environment>-searchTransaction-Logs:*"
        }
      }
    ]
  }
}
```

Instrucciones C.6: Definición de la máquina de estados de AWS Step Function para createTransaction

```
{
  "definition": {
    "FunctionName": "chatbot-<environment>-searchTransaction",
    "FunctionArn": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-searchTransaction",
    "Runtime": "nodejs14.x",
    "Role": "arn:aws:iam::<accountId>:role/chatbot-DynamoLogs",
    "Handler": "handlers/search.search",
    "Description": "Busca enlaces de pago en la base de datos",
    "Timeout": 3,
    "MemorySize": 512,
    "Version": "$LATEST",
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "Layers": [
      {
        "Arn":
    ↪ "arn:aws:lambda:<region>:<accountId>:layer:<sourceCode>:<layerVersion>",
      }
    ],
```

```
    "VpcConfig": {
      "SubnetIds": [
        "<subNetAZ1>",
        "<subNetAZ2>"
      ],
      "SecurityGroupIds": [
        "<securityGroupId>"
      ],
      "VpcId": "<vpcId>"
    },
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip",
    "Architectures": [
      "x86_64"
    ],
    "EphemeralStorage": {
      "Size": 512
    },
    "SnapStart": {
      "ApplyOn": "None",
      "OptimizationStatus": "Off"
    }
  }
}
```

Instrucciones C.7: Definición de la lambda para searchTransaction.

```
{
    "id": "0e0ce8da-1777-4186-8eb9-07df57b816c0"
}
```

Instrucciones C.8: Datos de entrada para la búsqueda de enlaces de pago

```
{
  "definition": {
    "Comment": "Actualizar enlaces de pago para pagos y recargas",
    "StartAt": "_updateTransaction",
    "States": {
      "_updateTransaction": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-updateTransaction",
        "End": true
      }
    }
  },
  "type": "EXPRESS",
  "loggingConfiguration": {
    "level": "ALL",
    "includeExecutionData": true,
```

```
    "destinations": [
      {
        "cloudWatchLogsLogGroup": {
          "logGroupArn": "arn:aws:logs:<region>:<accountId>:log-group:
    ↪ /aws/vendedlogs/states/chatbot-<environment>-updateTransaction-Logs:*"
        }
      }
    ]
  }
}
```

Instrucciones C.9: Definición de la máquina de estados de AWS Step Function para updateTransaction

```
{
  "definition": {
    "FunctionName": "chatbot-<environment>-updateTransaction",
    "FunctionArn": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-updateTransaction",
    "Runtime": "nodejs14.x",
    "Role": "arn:aws:iam::<accountId>:role/chatbot-DynamoLogs",
    "Handler": "handlers/update.update",
    "Description": "",
    "Timeout": 3,
    "MemorySize": 512,
    "Version": "$LATEST",
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "Layers": [
      {
        "Arn":
    ↪ "arn:aws:lambda:<region>:<accountId>:layer:<sourceCode>:<layerVersion>",
      }
    ],
    "VpcConfig": {
      "SubnetIds": [
        "<subNetAZ1>",
        "<subNetAZ2>"
      ],
      "SecurityGroupIds": [
        "<securityGroupId>"
      ],
      "VpcId": "<vpcId>"
    },
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip",
    "Architectures": [
      "x86_64"
```

```
    ],
    "EphemeralStorage": {
      "Size": 512
    },
    "SnapStart": {
      "ApplyOn": "None",
      "OptimizationStatus": "Off"
    }
  }
}
```

Instrucciones C.10: Definición de la lambda para updateTransaction.

```
{
    "channel": "BOT",
    "id": "<TOKEN>",
    "state": "next | error"
}
```

Instrucciones C.11: Datos de entrada para actualizar un enlace de pago

```

{
 "id": "<TOKEN>",
 "consumer": "BOT",
 "context": {
  "amount": "1000",
  "applicationName": "BOT",
  "commerceCode": "300",
  "commerceName": "CHATBOT",
  "distributionCode": "567",
  "failureCallback": "https://telecom.cl/failed",
  "failureTemplateName": "recarga_electronica_nok_v1",
  "integrationCode": "51",
  "isValid": false,
  "msisdn": "56987654321",
  "paymentMethodId": "WPY",
  "successCallback": "https://telecom.cl/success",
  "successTemplateName": "recarga_electronica_ok_v1"
 },
 "createdAt": 1686191042957,
 "individualIdentification": {
  "MSISDN": "56987654321",
  "RUT": "99999999-9"
 },
 "paymentBaseURL": "https://telecom.cl",
 "paymentLink": "https://telecom.cl?q=<TOKEN>",
 "reason": "Created successfully",
 "state": {
  "current": "CLOSED",
  "history": [
```

```
    "CLOSED",
    "PAID",
    "CREATED"
   ],
   "previous": "PAID"
  },
  "ttl": 1686192842956,
  "type": "billing",
  "updatedAt": 1686191042957
}
```

Instrucciones C.12: Ejemplo de registro insertado en la base de datos para un enlace de pago del tipo boleta

```
{
  "id": "<TOKEN>",
  "consumer": "BOT",
  "context": {
   "amount": "<AMOUNT>",
   "applicationName": "BOT",
   "commerceCode": "300",
   "commerceName": "CHATBOT",
   "distributionCode": "567",
   "failureCallback": "https://telecom.cl/failed",
   "failureTemplateName": "recarga_electronica_nok_v1",
   "integrationCode": "51",
   "isValid": false,
   "msisdn": "56987654321",
   "paymentMethodId": "WPY",
   "successCallback": "https://telecom.cl/success",
   "successTemplateName": "recarga_electronica_ok_v1"
  },
  "createdAt": 1686191042957,
  "individualIdentification": {
   "MSISDN": "<MSISDN>",
   "RUT": "<RUT>"
  },
  "paymentBaseURL": "https://telecom.cl/refillment",
  "paymentLink": "https://telecom.cl/refillment?q=<TOKEN>",
  "reason": "Created successfully",
  "state": {
   "current": "CLOSED",
   "history": [
    "CREATED",
    "PAID",
    "CLOSED"
   ],
   "previous": "CREATED"
  },
  "ttl": 1686192842956,
```

```
 "type": "refillment",
 "updatedAt": 1686191042957
}
```

Instrucciones C.13: Ejemplo de registro insertado en la base de datos para un enlace de pago del tipo recarga

```
    {
    "AttributeDefinitions": [
        {
            "AttributeName": "consumer",
            "AttributeType": "S"
        },
        {
            "AttributeName": "id",
            "AttributeType": "S"
        },
        {
            "AttributeName": "type",
            "AttributeType": "S"
        },
        {
            "AttributeName": "updatedAt",
            "AttributeType": "N"
        }
    ],
    "TableName": "transaction-state",
    "KeySchema": [
        {
            "AttributeName": "id",
            "KeyType": "HASH"
        }
    ],
    "TableStatus": "ACTIVE",
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    },
    "TableArn": "arn:aws:dynamodb:<region>:<accountId>:table/transaction-state",
    "GlobalSecondaryIndexes": [
        {
            "IndexName": "type-index",
            "KeySchema": [
                {
                    "AttributeName": "type",
                    "KeyType": "HASH"
                },
                {
                    "AttributeName": "updatedAt",
                    "KeyType": "RANGE"
```

```
                }
            ],
            "Projection": {
                "ProjectionType": "ALL"
            },
            "IndexStatus": "ACTIVE",
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 1,
                "WriteCapacityUnits": 1
            },
            "IndexArn": "arn:aws:dynamodb:<region>:<accountId>:
    ↪ table/transaction-state/index/type-index"
        },
        {
            "IndexName": "consumer-index",
            "KeySchema": [
                {
                    "AttributeName": "consumer",
                    "KeyType": "HASH"
                },
                {
                    "AttributeName": "updatedAt",
                    "KeyType": "RANGE"
                }
            ],
            "Projection": {
                "ProjectionType": "ALL"
            },
            "IndexStatus": "ACTIVE",
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 1,
                "WriteCapacityUnits": 1
            },
            "IndexArn": "arn:aws:dynamodb:<region>:<accountId>:
    ↪ table/transaction-state/index/consumer-index"
        },
        {
            "IndexName": "id-index",
            "KeySchema": [
                {
                    "AttributeName": "id",
                    "KeyType": "HASH"
                },
                {
                    "AttributeName": "updatedAt",
                    "KeyType": "RANGE"
                }
            ],
            "Projection": {
                "ProjectionType": "ALL"
```

```
            },
            "IndexStatus": "ACTIVE",
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 1,
                "WriteCapacityUnits": 1
            },
            "IndexArn": "arn:aws:dynamodb:<region>:<accountId>:
    ↪ table/transaction-state/index/id-index"
        }
    ]
}
```

Instrucciones C.14: Definición de la tabla transaction-state para la base de datos DynamoDB

```
    {
  "definition": {
    "Comment": "Accion automatica para obtener la deuda en linea de los clientes
    ↪ movil y hogar",
    "StartAt": "Obtener-token",
    "States": {
      "Obtener-token": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "OutputPath": "$.Payload",
        "Parameters": {
          "Payload.$": "$",
          "FunctionName": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-obtenerToken"
        },
        "Retry": [
          {
            "ErrorEquals": [
              "Lambda.ServiceException",
              "Lambda.AWSLambdaException",
              "Lambda.SdkClientException"
            ],
            "IntervalSeconds": 2,
            "MaxAttempts": 6,
            "BackoffRate": 2
          }
        ],
        "Next": "customerDebts"
      },
      "customerDebts": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "OutputPath": "$.Payload",
        "Parameters": {
          "Payload.$": "$",
          "FunctionName": "arn:aws:lambda:<region>:<accountId>:function:
```

```
    ↪ chatbot-<environment>-customerDebts"
        },
        "Retry": [
          {
            "ErrorEquals": [
              "Lambda.ServiceException",
              "Lambda.AWSLambdaException",
              "Lambda.SdkClientException"
            ],
            "IntervalSeconds": 2,
            "MaxAttempts": 6,
            "BackoffRate": 2
          }
        ],
        "End": true
      }
    }
  },
  "type": "EXPRESS",
  "loggingConfiguration": {
    "level": "ALL",
    "includeExecutionData": true,
    "destinations": [
      {
        "cloudWatchLogsLogGroup": {
          "logGroupArn": "arn:aws:logs:<region>:<accountId>:log-group:
    ↪ /aws/vendedlogs/states/chatbot-<environment>-obtenerDeuda-Logs:*"
        }
      }
    ]
  }
}
```

Instrucciones C.15: Definición de la máquina de estados AWS Step Function para la acción obtenerDeuda

```
{
  "definition": {
    "FunctionName": "chatbot-<environment>-customerDebts",
    "FunctionArn": "arn:aws:lambda:<region>:<accountId>:function:
    ↪ chatbot-<environment>-customerDebts",
    "Runtime": "nodejs14.x",
    "Role": "arn:aws:iam::<accountId>:role/chatbot-DynamoLogs",
    "Handler": "index.handler",
    "Description": "Obtiene la deuda en linea de los clientes",
    "Timeout": 90,
    "MemorySize": 128,
    "Version": "$LATEST",
    "VpcConfig": {
      "SubnetIds": [
```

```
        "<subNetAZ1>",
        "<subNetAZ2>"
      ],
      "SecurityGroupIds": [
        "<securityGroupId>"
      ],
      "VpcId": "<vpcId>"
    },
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip",
    "Architectures": [
      "x86_64"
    ],
    "EphemeralStorage": {
      "Size": 512
    },
    "SnapStart": {
      "ApplyOn": "None",
      "OptimizationStatus": "Off"
    }
  }
  }
}
```

Instrucciones C.16: Definición de la AWS Lambda para obtenerDeuda.

Anexo D

Acuerdos de Nivel de Servicio AWS

La tabla D.1 indica la confiabilidad para cada una de las tecnologías involucradas en la solución tecnológica.

Tabla D.1: Confiabilidad de los componentes utilizados en la solución tecnológica.

Tecnología	**Confiabilidad (%)**
AWS Lambda	99.95
AWS Step Function	99.9
AWS DynamoDB	99.999
AWS API Gateway	99.9
IBM Watson Assisant	99.9

Anexo E

Mediciones Round Trip Time

En el siguiente anexo se presentan los estadísticos calculados para todos los componentes involucrados en la solución de arquitectura. También se presentan las gráficas para los histogramas normalizados de cada componente, para esto, se han filtrado todos los datos con percentil mayor a 99,99 %, con el objetivo de quitar valores fuera de rango y poder apreciar la forma de los histogramas.

El primer componente medido corresponde al componente AWS Lambda `obtenerToken`, cuyos Estadísticoss se pueden ver en la tabla E.1 y el histograma se ve en E.1.

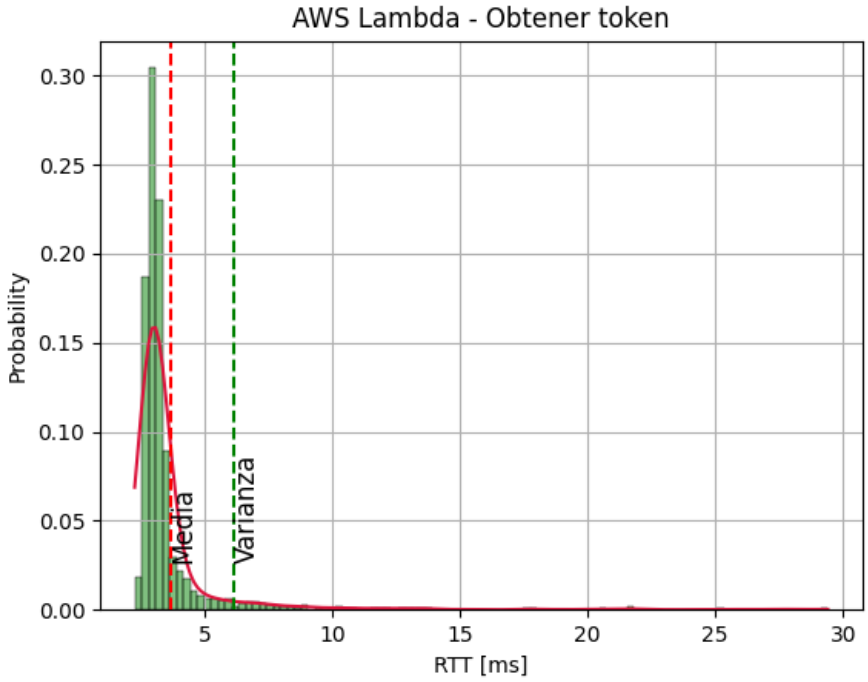

Figura E.1: Histograma normalizado del RTT para el componente AWS Lambda obtenerToken.

Tabla E.1: Estadísticos calculados para el RTT al autenticarse.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Lambda	obtenerToken	4	3	2	3	3	3	29

Los estadísticos calculados para la acción automática `obtenerDeuda` se puede ver en la tabla E.2, el histograma normalizado para la AWS Step Function `obtenerDeuda` se puede ver la figura E.2 y el histograma para la AWS Lambda `getCustomerDebts` se aprecia en la figura E.3.

Tabla E.2: Estadísticos obtenidos para los componentes involucrados al obtener deudas.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step Fn	obtenerDeuda	1794	419	742	1371	1550	2162	3164
Lambda	getCustomerDebts	1539	276	10	1328	1469	1712	2793

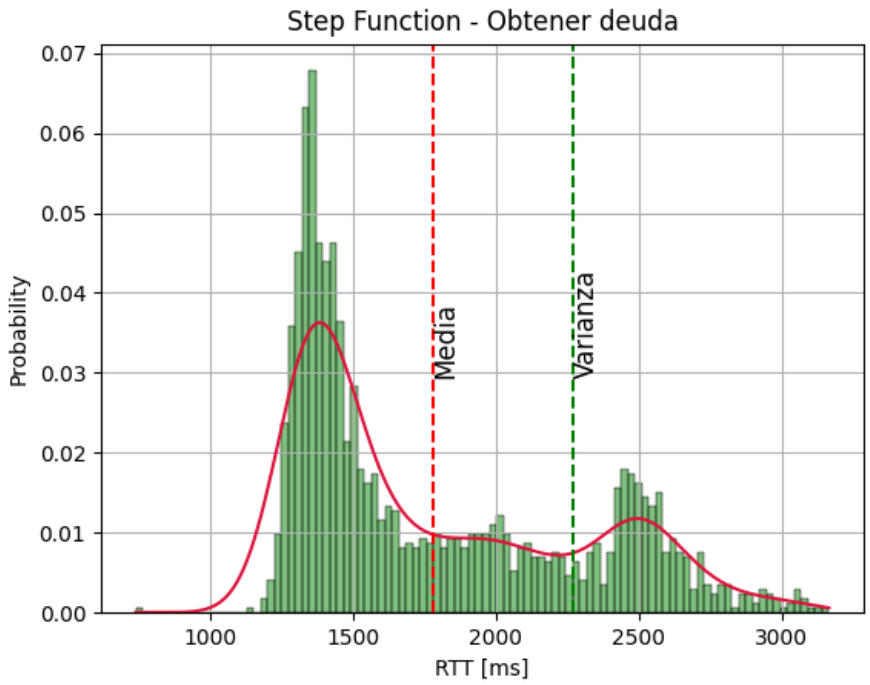

Figura E.2: Histograma normalizado del RTT para el componente AWS Step Function obtenerDeuda.

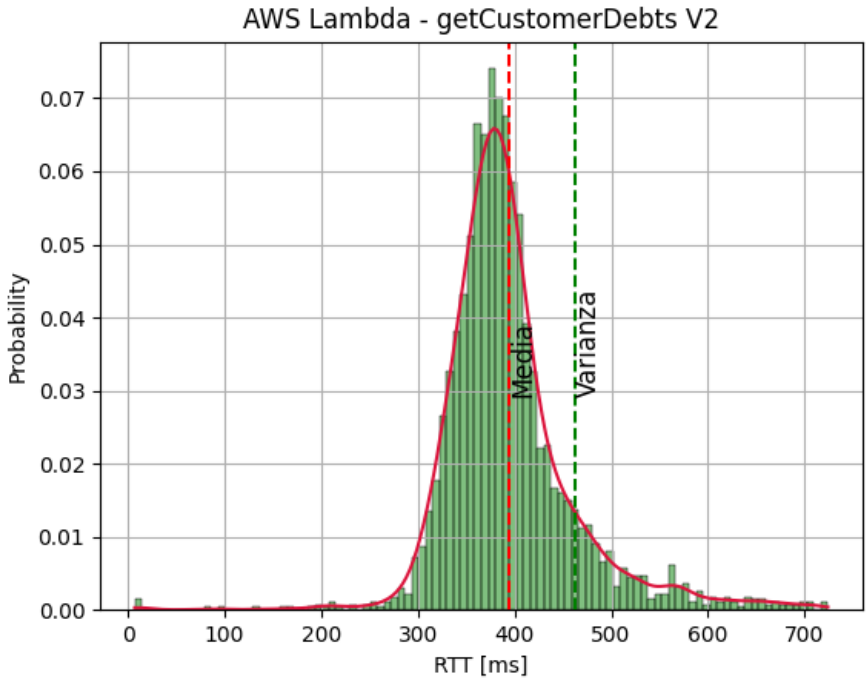

Figura E.3: Histograma normalizado del RTT para el componente AWS Step Function obtenerDeuda.

Los estadísticos calculados para la acción `verificarCliente` se muestra en la tabla E.3. En cuanto a los componentes involucrados, el histograma para la AWS Step Function `verificarCliente` se ve en la figura E.4 y su AWS Lambda en E.5.

Tabla E.3: Estadísticos obtenidos para los componentes involucrados al verificar un cliente.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step Fn	verificarCliente	1000	132	41	942	970	1025	1662
Lambda	getMSISDN	941	84	635	893	916	958	1460

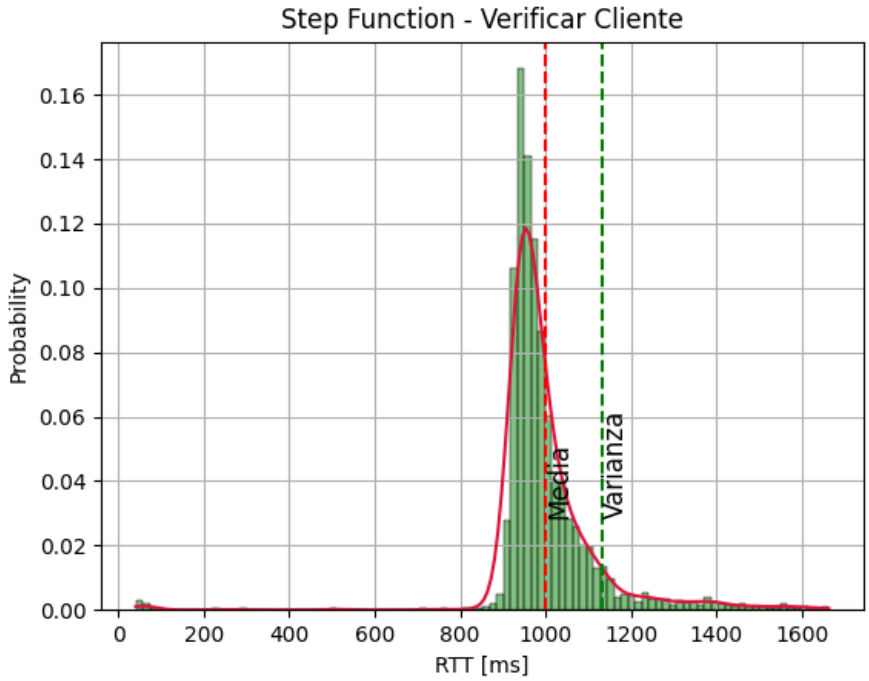

Figura E.4: Histograma normalizado del RTT para el componente AWS Step Function verificar-Cliente.

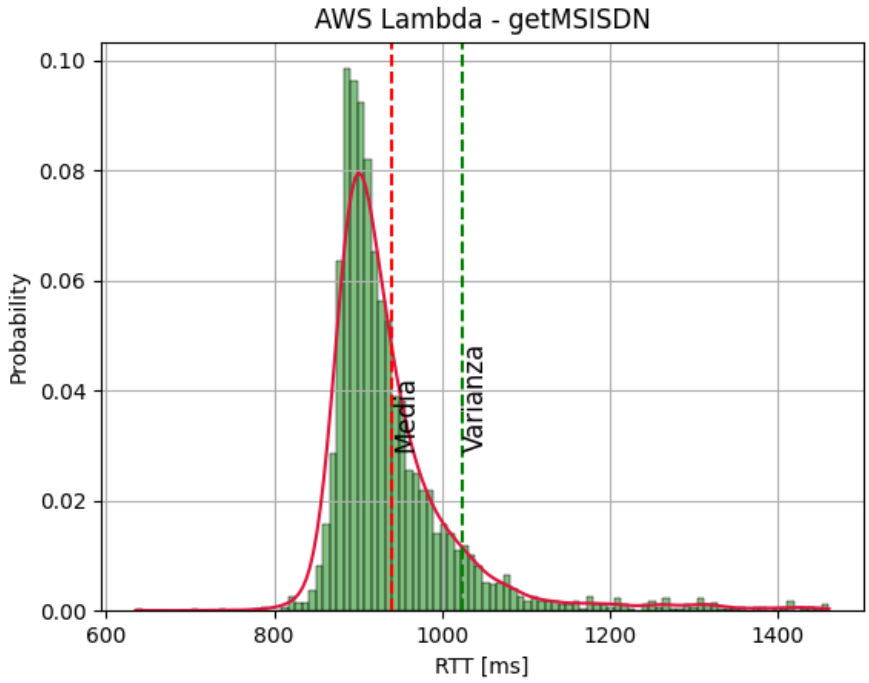

Figura E.5: Histograma normalizado del RTT para el componente AWS Lambda getMSISDN.

Los estadísticos calculados se pueden ver en la tabla E.4. El histograma para la AWS Step

Function `obtenerProducto` se ve en E.6 y la AWS Lambda `getMinimalAsset` en la figura E.7.

Tabla E.4: Estadísticos obtenidos para los componentes involucrados en obtener producto.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step Fn	obtenerProducto	917	115	19	848	891	953	1417
Lambda	getMinimalAsset	791	72	571	744	790	834	1028

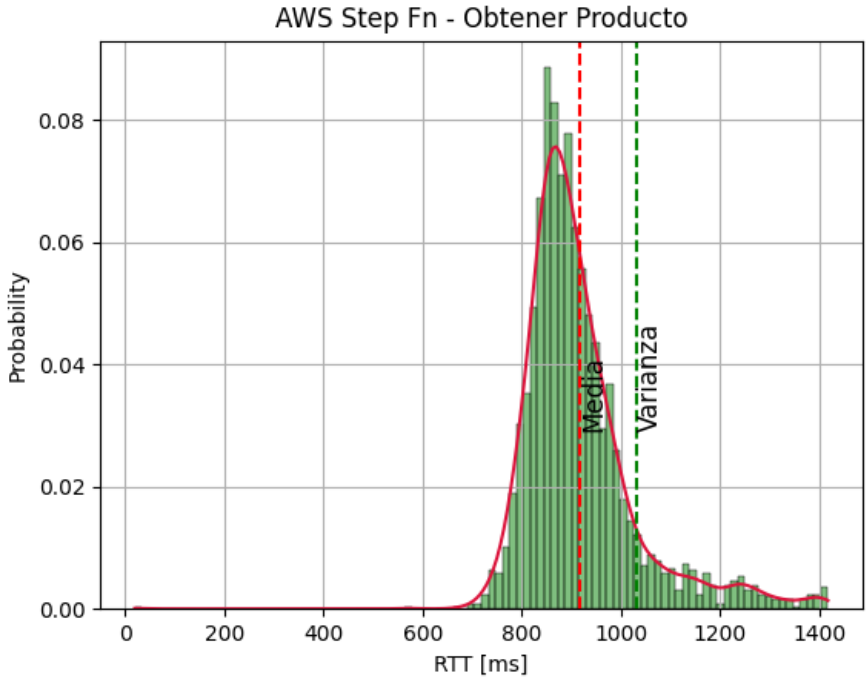

Figura E.6: Histograma normalizado del RTT para el componente AWS Step Function obtenerProducto.

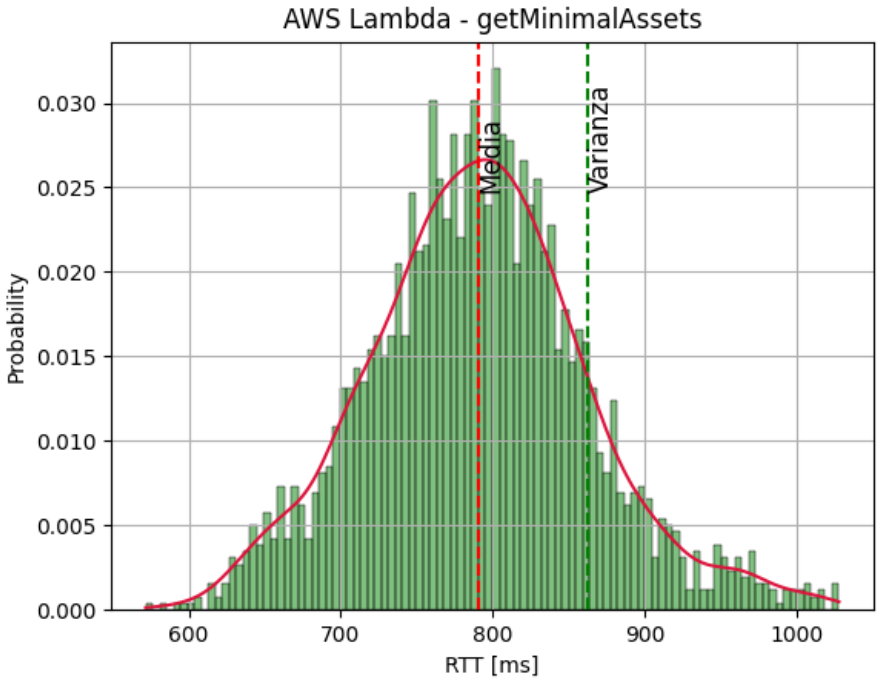

Figura E.7: Histograma normalizado del RTT para el componente AWS Lambda getMinimalAssets.

La acción automática `obtenerSaldo` está compuesta por tres AWS Lambda, los Estadísticoss para el RTT se presentan en la tabla E.5. En cuando a las AWS Lambda `getCustomer PrepayBalance`, `getMinimalAsset` y `getAvailableProductOffer` en las figuras E.9, E.7 y E.10, respectivamente.

Tabla E.5: Estadísticoss obtenidos para los componentes involucrados en obtener saldos.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step fn	obtenerSaldo	3433	535	1700	3103	3453	3771	4791
Lambda	getCustomer PrepayBalance	1434	112	1057	1353	1421	1508	1741
Lambda	getMinimalAsset	791	72	571	744	790	834	1028
Lambda	getAvailable ProductOffer	1110	485	1	864	1188	1439	1996

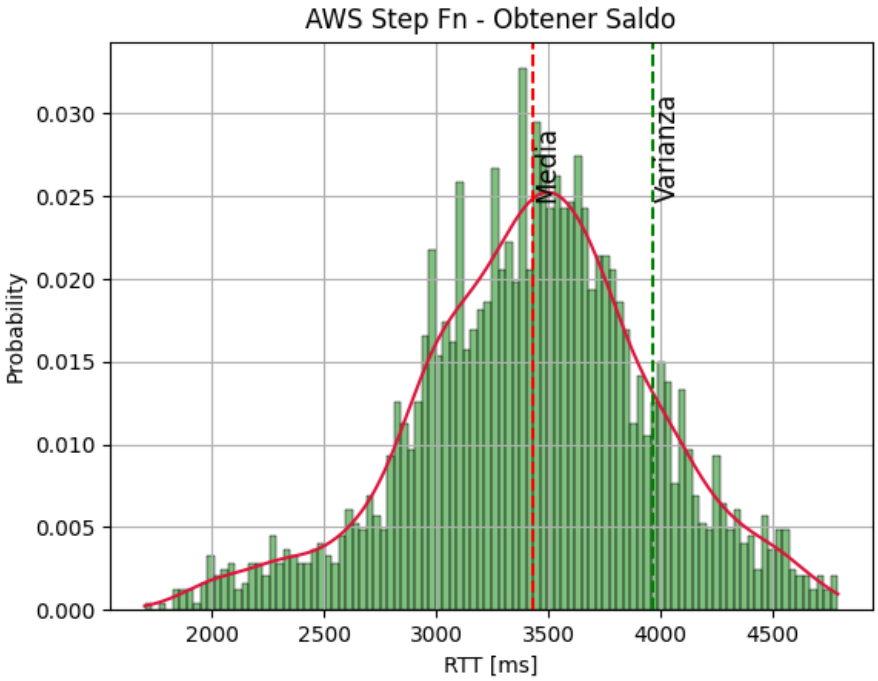

Figura E.8: Histograma normalizado del RTT para el componente AWS Step Function obtenerSaldo.

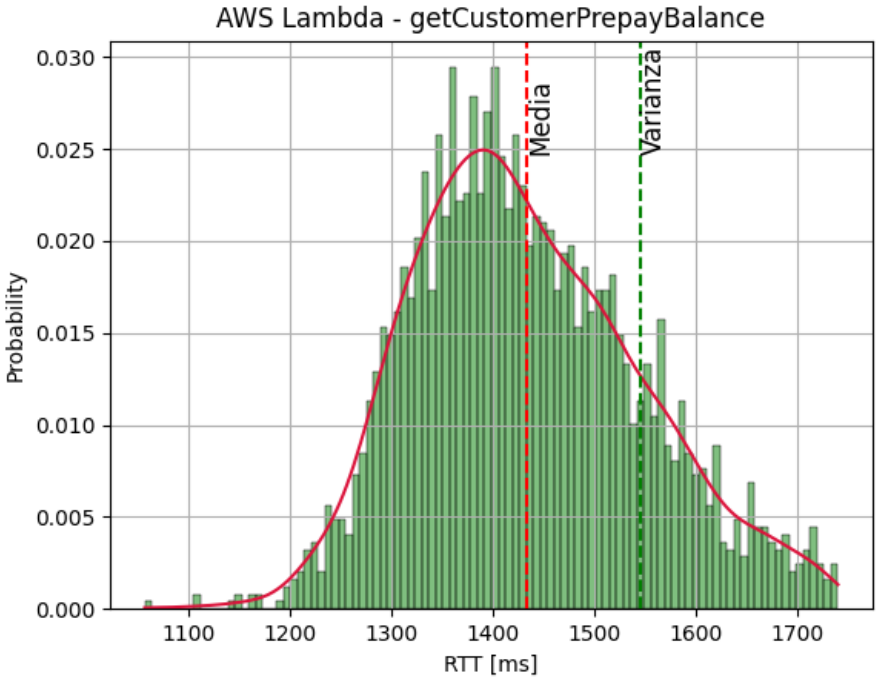

Figura E.9: Histograma normalizado del RTT para el componente AWS Lambda getCustomerPrepayBalance.

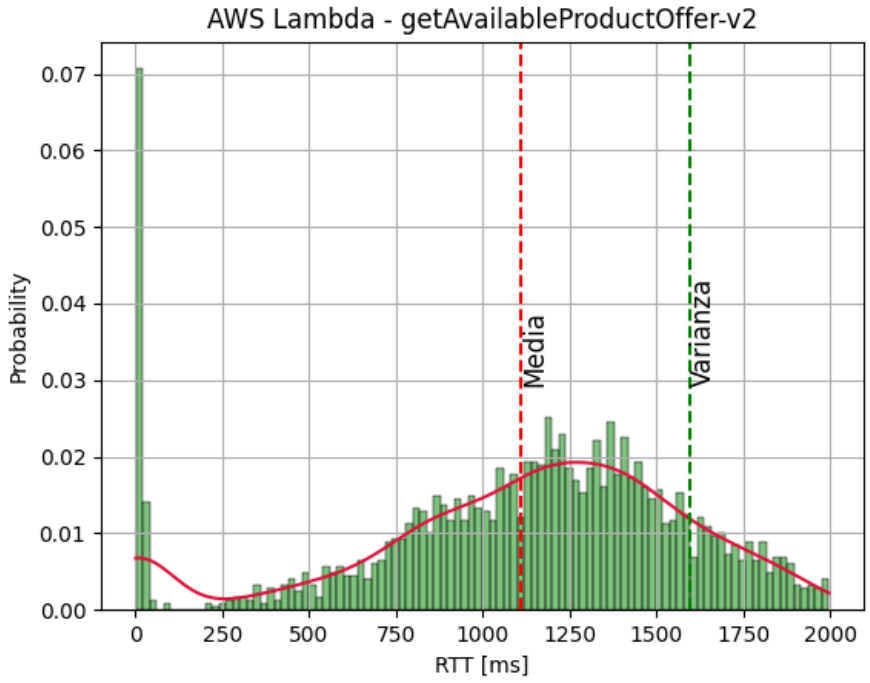

Figura E.10: Histograma normalizado del RTT para el componente AWS Lambda getAvailableProductOffer.

Los estadísticos calculados para el RTT medido en el componente AWS Internal API Gateway se pueden ver en la tabla E.6 y el histograma normalizado se puede ver en la figura E.11.

Tabla E.6: Estadísticos obtenidos AWS Internal API Gateway.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
API Gateway	Transaction API	511	443	123	175	325	675	1675
API Gateway	Template API	69	4	37	67	69	72	83

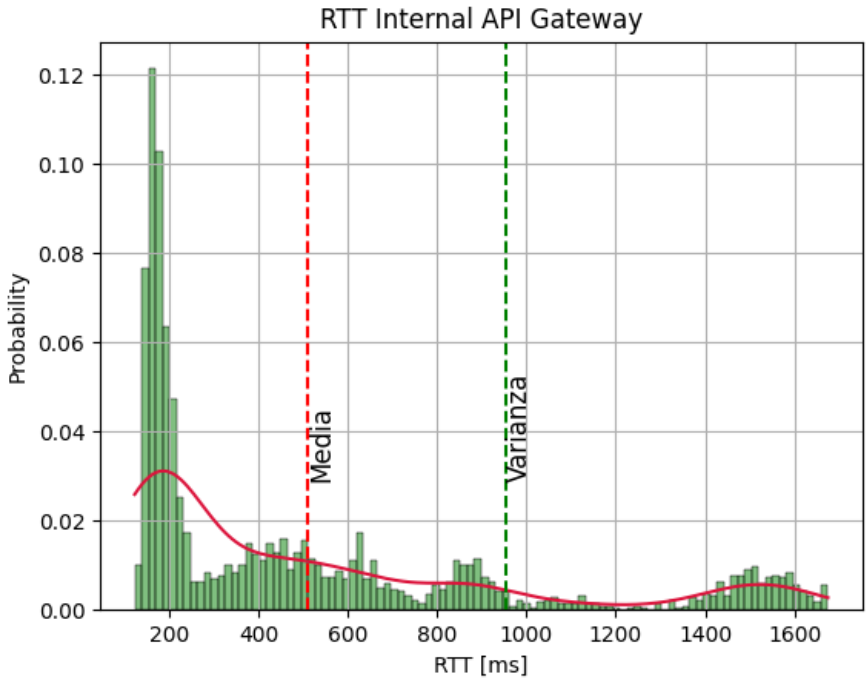

Figura E.11: Histograma normalizado del RTT para el componente AWS Internal API Gateway.

En cuanto a los RTT medidos para la acción automática `createTransactiom`, compuesta por la AWS Lambda `createTransaction` que consulta la tabla `transaction-state` mediante una consulta del tipo *PutItem*, se pueden ver todos en la tabla E.7. Los histogramas normalizados se muestran en las figuras E.12, E.13, E.14, respectivamente.

Tabla E.7: Estadísticos obtenidos para los componentes involucrados en crear enlaces de pago.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step Fn	createTransaction	204	331	46	70	79	102	1417
Lambda	createTransaction	56	19	16	43	47	62	117
DynamoDB	PutItem	21	2	10	20	21	22	29

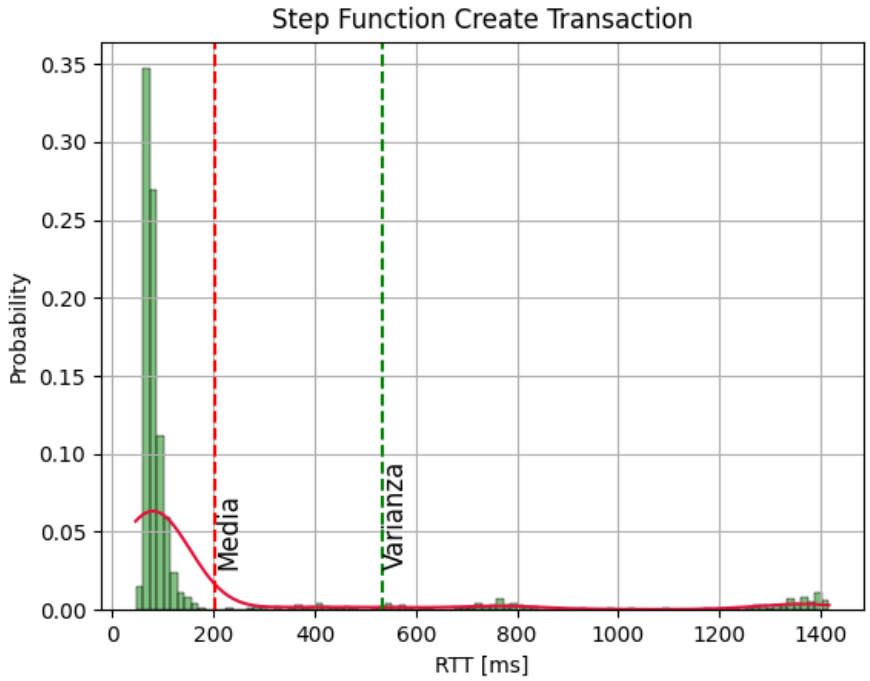

Figura E.12: Histograma normalizado del RTT para el componente AWS Step Function createTransaction.

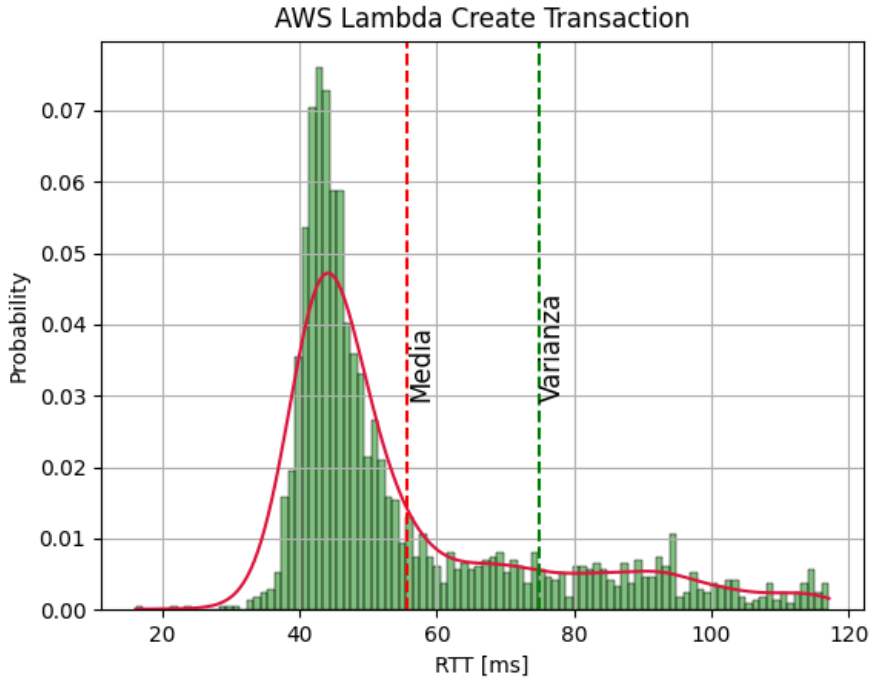

Figura E.13: Histograma normalizado del RTT para el componente AWS Lambda createTransaction.

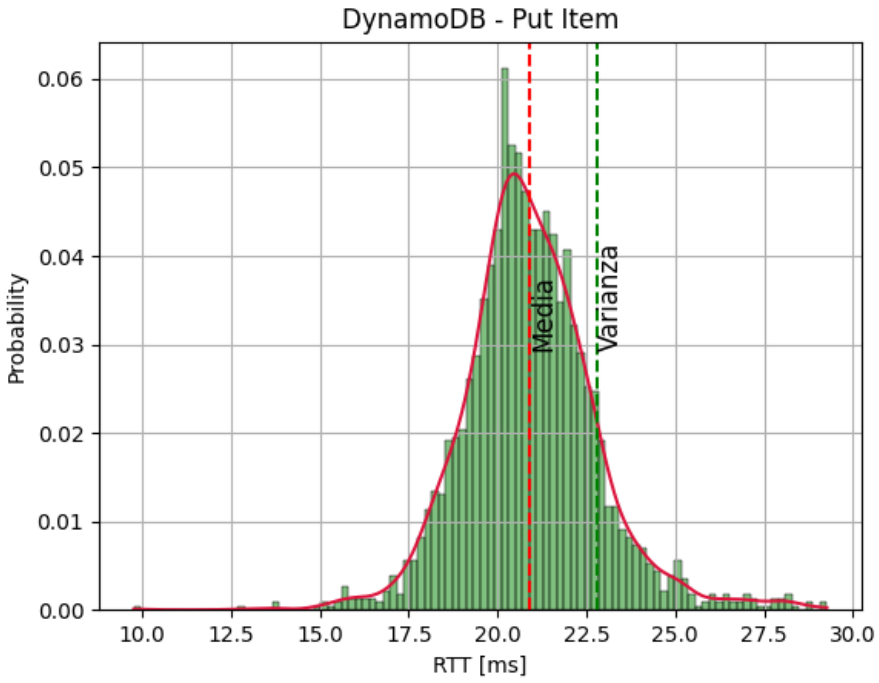

Figura E.14: Histograma normalizado del RTT para la petición AWS DynamoDB *PutItem*.

El servicio para buscar transacciones de pago es utilizado por la pasarela de pagos, a través del API Manager de la empresa. Este API Manager está conectado al Internal API Gateway, cuyos RTT se han presentado en E.6. No existen datos disponibles para los RTT en API Manager. Luego, el servicio que busca las transacciones de pago ejecuta la AWS Step Function `searchTransaction`, que a su vez, invoca la AWS Lambda `searchTransaction`, la cual realiza una consulta a DynamoDB del tipo *ReadItem* en la tabla `transaction-state`. Estos resultados se muestran en la tabla E.8 y sus histogramas en las figuras E.15, E.16 y E.17, para los componentes mencionados, respectivamente.

Tabla E.8: Estadísticos obtenidos para los componentes involucrados en buscar transacciones de pago.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step Fn.	searchTransaction	307	444	49	73	88	158	1591
Lambda	searchTransaction	60	26	28	39	48	79	137
DynamoDB	ReadItem	18	2	5	17	18	19	25

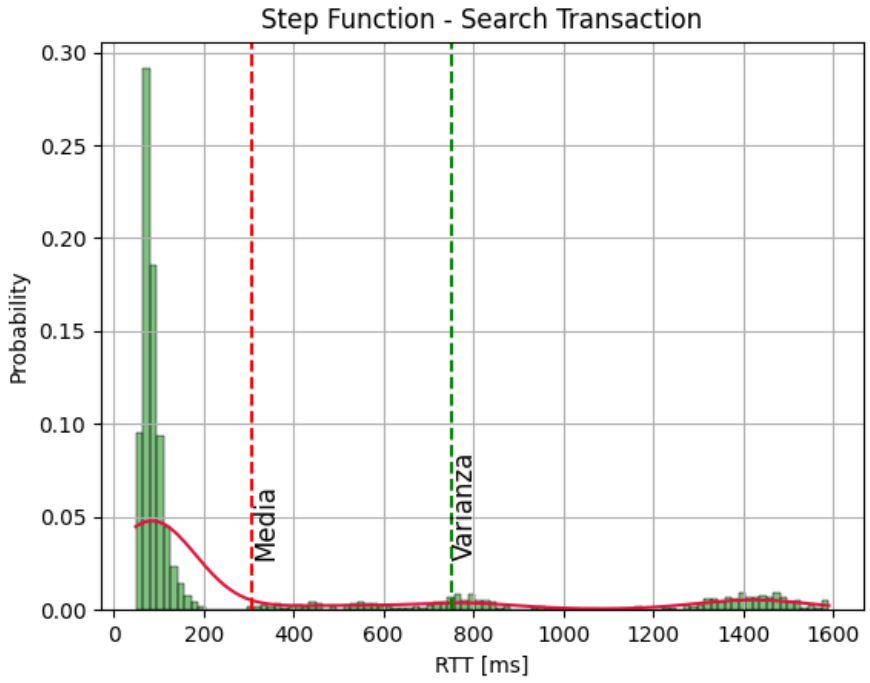

Figura E.15: Histograma normalizado del RTT para el componente AWS Step Function searchTransaction.

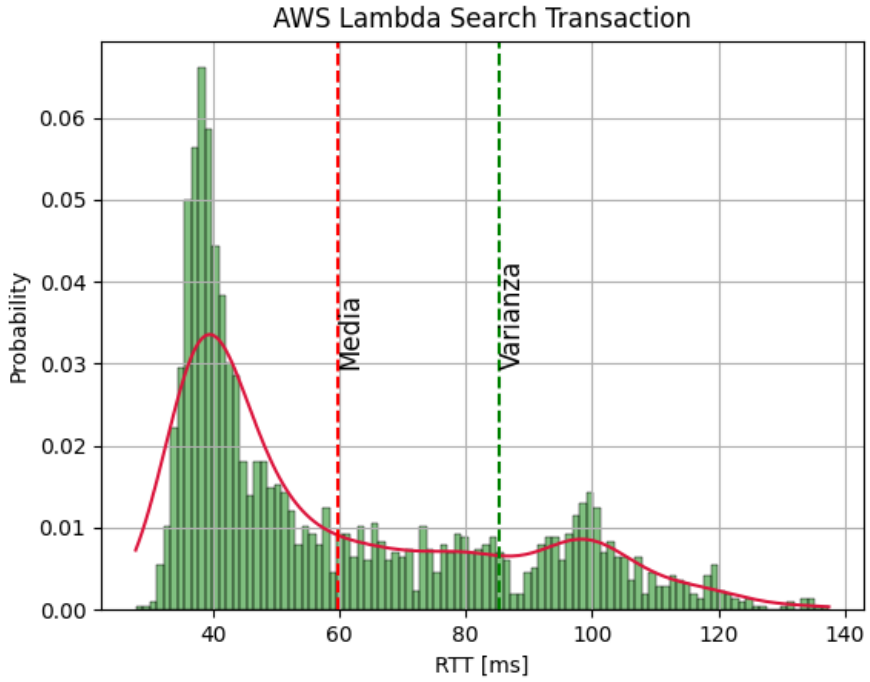

Figura E.16: Histograma normalizado del RTT para el componente AWS Lambda searchTransaction.

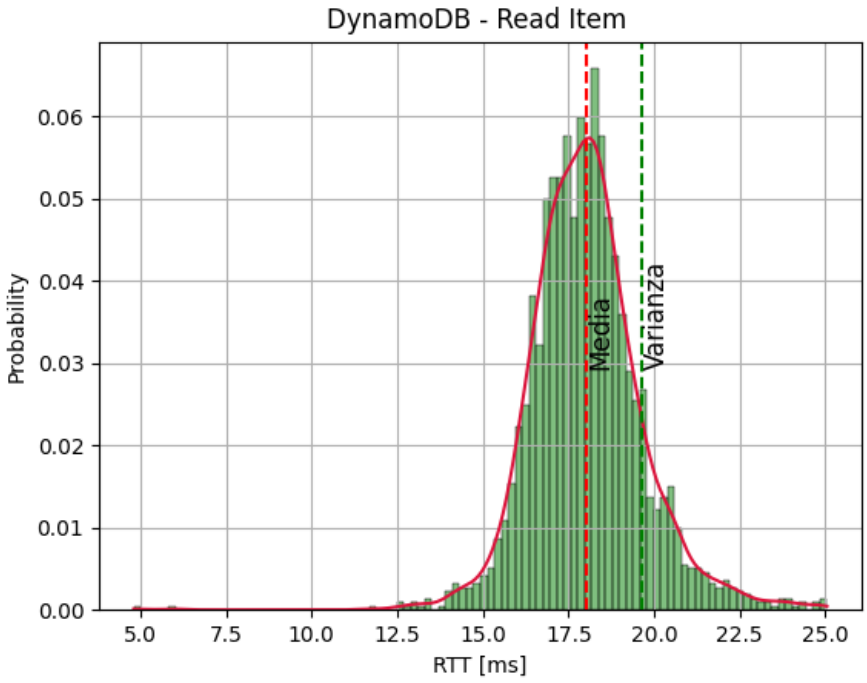

Figura E.17: Histograma normalizado del RTT para la petición AWS DynamoDB *ReadItem*.

El servicio para actualizar los enlaces de pago es utilizado por la pasarela de pagos de la compañía por medio del API Manager, el cual consulta al AWS Internal API Gateway. No hay datos de RTT disponibles para API Manager y los del Internal API Gateway han sido presentados en E.6. El servicio `updateTransaction` es una AWS Step Function que invoca la AWS Lambda `updateTransaction`, la cual realiza dos consultas a la base de datos DynamoDB mediante las operaciones *ReadItem* y *PutItem*, en la tabla `transaction-state`. Las mediciones para estos componentes se puede ver en la figura E.9 y los histogramas en las figuras E.18, E.19 y E.20, para los componentes mencionados, respectivamente.

Tabla E.9: Estadísticos obtenidos para los componentes involucrados en actualizar transacciones de pago.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
Step Fn	updateTransaction	947	571	76	267	1197	1460	1763
Lambda	updateTransaction	131	25	62	121	134	146	185
DynamoDB	ReadItem+PutItem	42	5	19	40	42	44	139

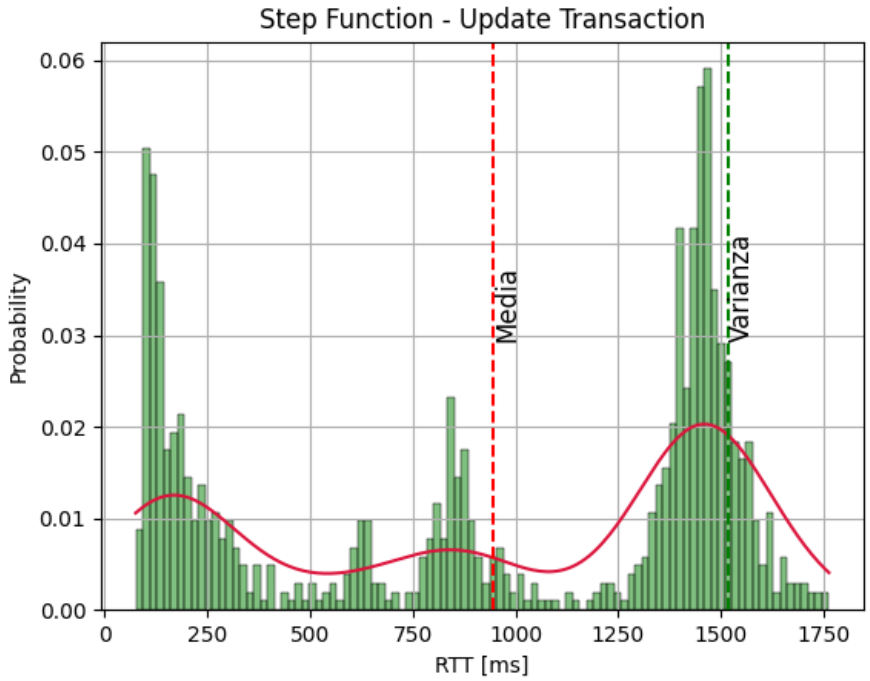

Figura E.18: Histograma normalizado del RTT para el componente AWS Step Function updateTransaction.

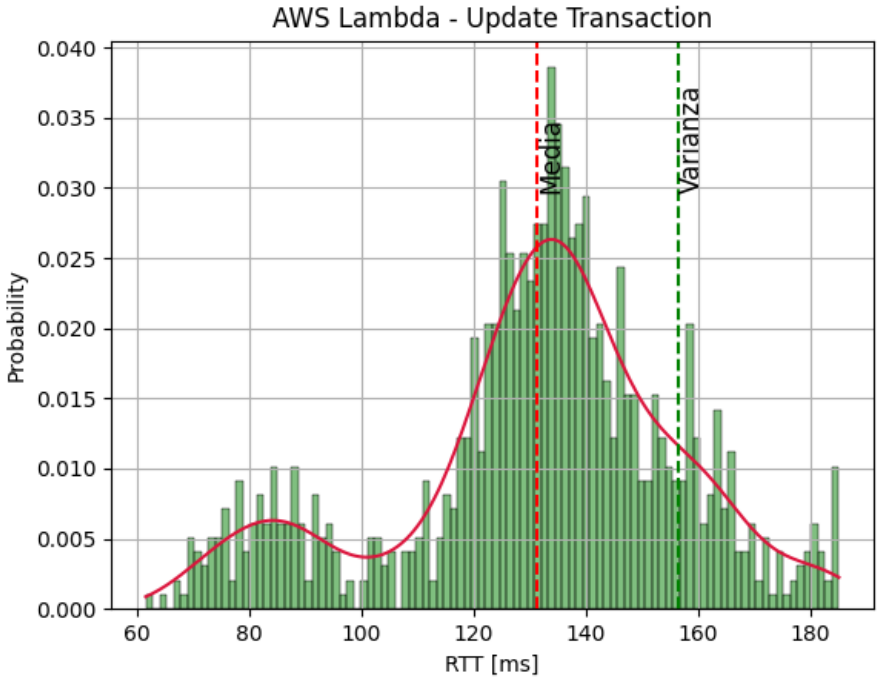

Figura E.19: Histograma normalizado del RTT para el componente AWS Lambda updateTransaction.

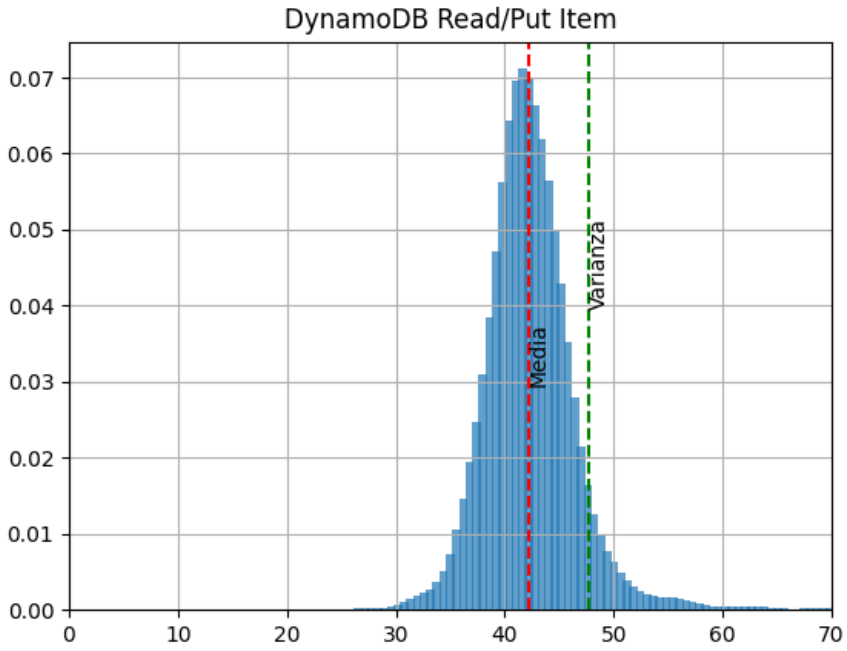

Figura E.20: Histograma normalizado del RTT para la petición AWS DynamoDB *PutItem* y *ReadItem*

El servicio de notificaciones por WhatsApp es utilizado por la pasarela de pagos de la compañía por medio del API Manager, el cual se conecta al AWS Template Internal API Gateway, este *gateway* cursa las solicitudes de manera asíncrona, a diferencia del AWS Internal API Gateway que las resuelve de forma síncrona, esto se logra mediante una cola de mensajería presentada en el diseño de componentes C.1. Solo se cuenta con mediciones para el *gateway* y la función Lambda `sendTemplate` que ejecuta el proceso de envío. Las mediciones se presentan en la tabla E.10 y son coherentes con el comportamiento asíncrono. Asimismo, los histogramas normalizados son presentados en las figuras E.21 y E.22, para el Notification API Gateway y AWS Lambda, respectivamente.

Tabla E.10: Estadísticos obtenidos para los componentes involucrados en las notificaciones por WhatsApp.

		Estadísticos RTT [milisegundos]						
Tecnología	**Componente**	**media**	**std**	**min**	**25 %**	**50 %**	**75 %**	**max**
API Gateway	Notification API	69	4	37	67	69	72	83
Lambda	sendNotification	395	68	7	358	384	416	724

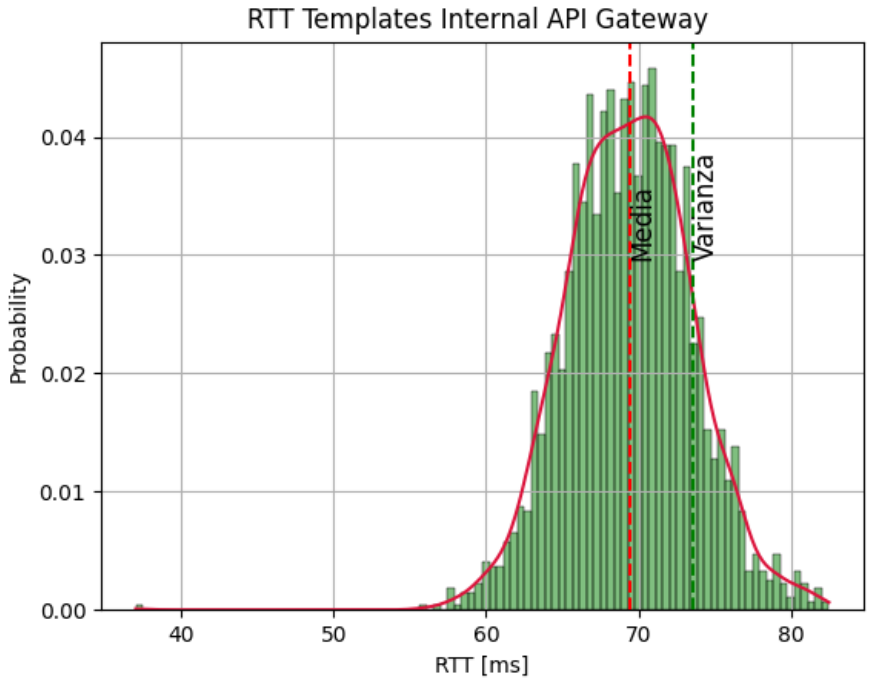

Figura E.21: Histograma normalizado del RTT para el componente AWS Template Internal API Gateway.

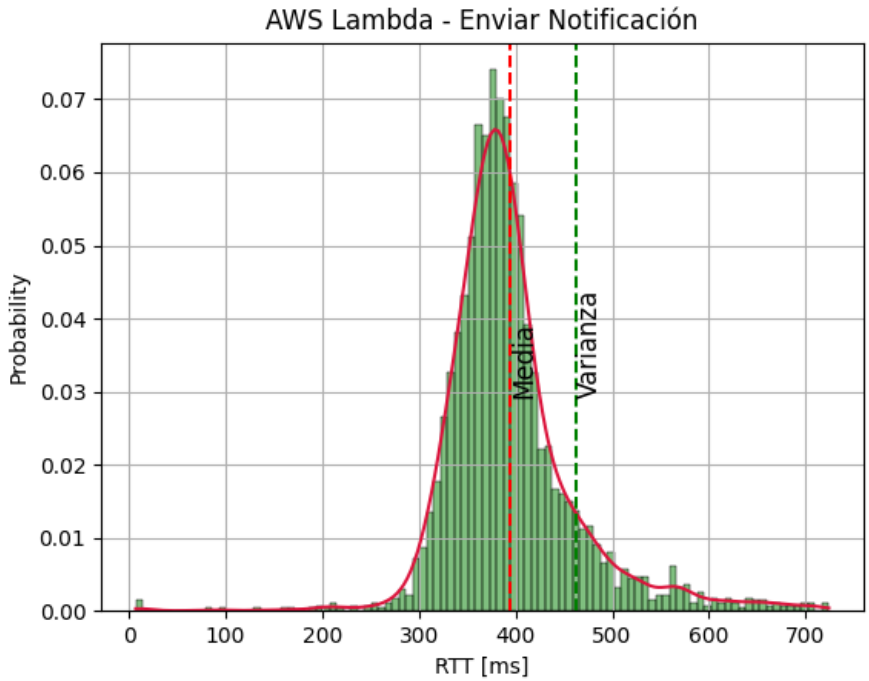

Figura E.22: Histograma normalizado del RTT para el componente AWS Lambda sendTemplate.

Anexo F

Detalle de Costos AWS

En la siguiente sección se presenta el desglose de la cotización realizada en AWS para todos los componentes involucrados en la solución tecnológica implementada, cuyo desglose de los costos se puede ver en la tabla F.1.

Tabla F.1: Detalle de los costos AWS para la solución tecnológica.

Servicio	**Componente**	**Costo (USD / Mes)**	**Resumen de configuración**
Step Functions Express Workflows	obtenerProducto	2,15	Duración de cada flujo de trabajo (1019 ms). Memoria consumida por cada flujo de trabajo (64 MB). Solicitudes de flujo de trabajo (1000000 por mes)
Step Functions Express Workflows	obtenerSaldo	4,75	Duración de cada flujo de trabajo (3592 ms). Memoria consumida por cada flujo de trabajo (64 MB). Solicitudes de flujo de trabajo (1000000 por mes)
Step Functions Express Workflows	obtenerDeuda	2,88	Duración de cada flujo de trabajo (1793 ms) Memoria consumida por cada flujo de trabajo (64 MB) Solicitudes de flujo de trabajo (1000000 por mes)
Step Functions Express Workflows	createTransaction	1,31	Duración de cada flujo de trabajo (283 ms) Memoria consumida por cada flujo de trabajo (64 MB) Solicitudes de flujo de trabajo (1000000 por mes)
Step Functions Express Workflows	searchTransaction	1,42	Duración de cada flujo de trabajo (321 ms) Memoria consumida por cada flujo de trabajo (64 MB) Solicitudes de flujo de trabajo (1000000 por mes)
Step Functions Express Workflows	updateTransaction	2,04	Duración de cada flujo de trabajo (956ms) Memoria consumida por cada flujo de trabajo (64 MB) Solicitudes de flujo de trabajo (1000000 por mes)
AWS Lambda	obtenerToken	0,21	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)
AWS Lambda	customerDebts	3,44	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)
AWS Lambda	createTransaction	0,33	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)

AWS Lambda	obtenerToken	0,21	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)
AWS Lambda	customerDebts	3,44	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)
AWS Lambda	createTransaction	0,33	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)
AWS Lambda	searchTransaction	0,65	Arquitectura (x86) Cantidad de solicitudes (1 millones por mes) Cantidad de almacenamiento efímero asignado (512 MB) Simultaneidad (1) Horas de simultaneidad aprovisionada (20 horas) Peticiones con simultaneidad aprovisionada (1000000 por mes)
AWS Lambda	updateTransaction	4,05	Arquitectura (x86) Cantidad de solicitudes (1 millones por mes) Cantidad de almacenamiento efímero asignado (512 MB) Simultaneidad (1) Horas de simultaneidad aprovisionada (20 horas) Peticiones con simultaneidad aprovisionada (1000000 por mes)
AWS Lambda	sendNotification	1,04	Arquitectura (x86) Cantidad de almacenamiento efímero asignado (512 MB) Cantidad de solicitudes (1 millones por mes)
Amazon API Gateway	Internal REST API Gateway	10,5	Unidades de solicitud de la API REST (millones) Unidades de solicitudes de la API HTTP (millones) Solicitudes (3 millones por mes)
DynamoDB on-demand capacity	Table: transaction-state	3	Clase de tabla (Standard) Tamaño promedio del elemento (920 Byte) Tamaño del almacenamiento de datos (1 GB)