UCAB Universidad Católica ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

MICROSERVICIOS PARA LA APLICACIÓN DE USUARIOS DE ESTACIONAMIENTOS CITYWALLET

TRABAJO INSTRUMENTAL DE GRADO

Presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR	**Alejandro Antonio García Acosta**
TUTOR ACADÉMICO	**Ing. Valeria León**
TUTOR EMPRESARIAL	**Ing. Félix Armando Fernández Pérez**
FECHA	**15/09/2017**

UCAB Universidad Católica ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

MICROSERVICIOS PARA LA APLICACIÓN DE USUARIOS DE ESTACIONAMIENTOS CITYWALLET

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado:

JURADO EXAMINADOR

Firma:	Firma:	Firma:
Nombre: Ana Fernandes	Nombre: Valeria León	Nombre: RAFAEL A. LARA CAMPOS

REALIZADO POR	**Alejandro Antonio García Acosta**
TUTOR ACADÉMICO	**Ing. Valeria León**
TUTOR EMPRESARIAL	**Ing. Félix Armando Fernández Pérez**
FECHA	**15/09/2017**

DEDICATORIA

Este logro se lo dedico a mis padres, quienes me han ayudado y soportado por cada trasnocho, exámenes, ataques nerviosos, dolores de cabeza y estómago gracias a esta carrera.

A mis abuelos, quienes me han tenido saludable todas las veces que he estado como "inquilino" en su casa, durante los días más difíciles de desarrollo de proyectos y tesis.

A mi hermano, quien me ha mantenido activo y despierto hasta altas horas de la noche pendiente de que llegara a la casa.

A Carlos Unda, uno de mis hermanos/mejores amigos, quien me ha sacado la pata del barro siempre que lo necesito, quien me ha mostrado que siempre hay un punto de vista válido distinto al mío, y quien me ha soportado en mis momentos de crisis más injustificados.

A mi novia, María Unda, de quien aprendo cada día más y la que ha estado conmigo en las buenas y no tan buenas, siempre, para sacarme una sonrisa y verle el lado positivo a las cosas.

A dos familias, Unda y Sifontes, quienes me hacen sentir bienvenido cada que nos encontramos para celebrar y compartir.

A CityWallet, por darme la oportunidad de crecer como persona y profesional a la vez, junto a los mejores emprendedores de Venezuela.

A mis panas, Ricardo Pérez, Enrique Nichols July Marval, Javier Pérez, Manuel Pacheco, Gabriela Sotelo y Santiago Bernal por ser mucho más que "compañeros de trabajo" y compartir los mejores momentos.

A los súper jefazos, Ramón Ginéz, Félix Fernández, Hernán García, Atilana Piñón, Liesl Hros y Fernándo Marcano, quienes me han apoyado en cada paso que damos y los que invitan las cenas luego de grandes éxitos.

A la profe Valeria León, nuestra tutora, quien nos atiende y guía en los momentos de más presión, para así poder hacer realidad nuestro gran sueño.

A Venezuela, mi gran hogar, para que juntos podamos salir de los momentos más difíciles y hacerte el país que soñamos.

AGRADECIMIENTOS

Agradezco a mis padres Vladimir y Sor Angel, por estar siempre a mi lado apoyándome y guiándome en cada decisión que he tomado.

A mis abuelos Norma y Alfonso, por haberme enseñado los valores que hoy me hacen la persona que soy.

A mi hermano Andrés, por enseñarme que con esfuerzo y cariño se logra lo que nos proponemos.

A mis tíos Norma y Raúl, que a pesar de la distancia siempre están ahí para escucharme y aconsejarme.

A mis primas y a mi tía, Alanis, Joengly, Andreina y Estela, por cuidarme y mantenerme animado en esta carrera de resistencia que está por culminar.

A Dios, por darme salud y la familia que tengo.

Alejandro Antonio García Acosta

ÍNDICE

DEDICATORIA i
AGRADECIMIENTOS ii
ÍNDICE iii
ÍNDICE DE TABLAS v
ÍNDICE DE FIGURAS vi
GLOSARIO vii
1. Mantenibilidad del software vii
2. Acoplamiento y cohesión vii
3. Escalabilidad vii
4. API viii
5. RESTful viii
SINOPSIS ix
CAPÍTULO I – PLATEAMIENDTO DEL PROBLEMA 1
1. NECESIDADES DE LA EMPRESA 1
2. SOLUCIÓN PROPUESTA 3
1. OBJETIVO GENERAL 6
2. OBJETIVOS ESPECÍFICOS 6
3. ALCANCE 7
4. LIMITACIONES 13
5. JUSTIFICACIÓN 14
CAPÍTULO II - MARCO REFERENCIAL 16
1. FinTech 16
2. Arquitectura de software 17
3. Cloud Computing 19
4. Transaction tool 24
CAPITULO III 26
1. MARCO METODOLÓGICO 26
1.1. Scrum 26
1.2. Marco de trabajo ágil de CityWallet 32
CAPÍTULO IV - DESARROLLO 43
1. Sprints 43
CAPÍTULO V 76

1. Resultados 76
2. Conclusiones 82
3. Recomendaciones 83
REFERENCIAS BIBLIOGRÁFICAS 86
ANEXOS 94
Anexo A – APIs 94
1. Recursos 94
2. Estructura del API 95
Anexo B – Microservicios 110
Anexo C – Tablas de base de datos 111
Anexo D – *Sprints* 118
Anexo E – Diagramas de flujo 124
1. Procesos rediseñados 124
2. Procesos anteriores 132

ÍNDICE DE TABLAS

Tabla 1. Comparativa de semejanzas y diferencias entre plataformas en la nube. 46
Tabla 2. Matriz ponderación según criterios..... 47
Tabla 3. Recursos y métodos desarrollados. 94
Tabla 4. Clasificación de microservicios desarrollados. 110
Tabla 5. Tabla Accounts. 111
Tabla 6. Tabla Balance. 112
Tabla 7. Tabla Transactions..... 113
Tabla 8. Tabla TransactionTools..... 114
Tabla 9. Tabla TransactionToolTicket. 115
Tabla 10. Tabla Stock. 116
Tabla 11. Tabla Orders 117
Tabla 12. Backlog del producto..... 118
Tabla 13. Backlog del Sprint 1. 119
Tabla 14. Backlog del Sprint 2. 120
Tabla 15. Backlog del Sprint 3. 120
Tabla 16. Backlog del Sprint 4. 121
Tabla 17. Backlog del Sprint 5. 121
Tabla 18. Backlog del Sprint 6. 121
Tabla 19. Backlog del Sprint 7. 122
Tabla 20. Backlog del Sprint 8. 122
Tabla 21. Backlog del Sprint 9. 123

ÍNDICE DE FIGURAS

Figura 1. Arquitectura propuesta para la aplicación de CityWallet..... 5
Figura 2. Ejemplo de una arquitectura de microservicios. 18
Figura 3. Administración, autenticación y sincronización con Amazon Cognito..... 23
Figura 4. Arquitectura de la solución..... 51
Figura 5. Recurso transaction tools, método POST. 95
Figura 6. Recurso transaction tools, método GET..... 96
Figura 7. Recurso transaction tools, método PUT..... 97
Figura 8. Recurso accounts, método GET 98
Figura 9. Recurso balance, método GET. 99
Figura 10. Recurso transactions, método GET. 100
Figura 11. Recurso transactions, método GET por id..... 101
Figura 12. Recurso transactions, método POST. 102
Figura 13. Recurso orders, método POST. 103
Figura 14. Recurso orders, método GET. 104
Figura 15. Recurso orders, método GET por id..... 105
Figura 16. Recurso orders, método PUT..... 106
Figura 17. Recurso orders, sub recurso payment, método POST. 107
Figura 18. Recurso products, método GET..... 108
Figura 19. Recurso products, método GET por id. 109
Figura 20. Rediseño de asociación de un transaction tool. 124
Figura 21. Rediseño de consulta de transaction tools. 125
Figura 22. Rediseño de modificación de un transaction tool. 125
Figura 23. Rediseño de agregar de una transacción de recarga de saldo..... 126
Figura 24. Rediseño de consulta de transacciones. 127
Figura 25. Rediseño de consulta de una transacción. 127
Figura 26. Rediseño de agregar una cuenta de usuario..... 127
Figura 27. Rediseño de consultar cuenta de usuario. 128
Figura 28. Rediseño de consultar balance de una cuenta de usuario. 128
Figura 29. Rediseño de agregar una orden carro de compra. 128
Figura 30. Rediseño de consultar órdenes..... 129
Figura 31. Rediseño de consultar una orden o carro de compra. 129
Figura 32. Rediseño de modificar una orden o carro de compra. 129
Figura 33. Rediseño de consultar productos del inventario. 130
Figura 34. Rediseño de consultar un producto del inventario. 130
Figura 35. Rediseño de pago de orden o carro de compra. 131
Figura 36. Diseño de asociación de un cashout device..... 132
Figura 37. Diseño de consulta de cashout devices. 133
Figura 38. Diseño de modificación de un cashout device..... 133
Figura 39. Diseño de creación de cuenta..... 134
Figura 40. Diseño de consulta de cuenta. 134
Figura 41. Diseño de pago por InstaPago..... 135
Figura 42. Diseño de consulta de transacciones..... 136
Figura 43. Diseño de consulta de una transacción..... 137

GLOSARIO

1. Mantenibilidad del software

La facilidad con la que un sistema o software puede ser modificado para prevenir, mejorar y/o corregir fallos es denominado como mantenibilidad de software. Aplicado, es el proceso de modificar el funcionamiento y otros atributos para adaptarlos al entorno [16].

2. Acoplamiento y cohesión

Se define como cohesión al grado de cercanía entre dos o más elementos, es decir, el conjunto de características que los une y, por lo tanto, los agrupa bajo un mismo denominador definiendo su identidad. Por otro lado, El acoplamiento mide el grado de dependencia entre dos o más elementos, por lo que según su dependencia estarán más o menos acoplados; a mayor dependencia mayor acoplamiento [17].

El objetivo final del diseño de software y/o soluciones informáticas en relación al acoplamiento y cohesión es: reducir al máximo el acoplamiento entre componentes y aumentar la cohesión interna de cada componente [18].

3. Escalabilidad

La escalabilidad hace referencia para la informática y las telecomunicaciones a las propiedades de un sistema o red que hacen

que éste se mantenga funcional y operativamente activo cuando la demanda aumenta, también, puede significar la capacidad que tiene el sistema informático o red para cambiar su tamaño o configuración para ajustarse a demandas de servicio cambiantes sin afectar su funcionamiento [19].

4. API

El término API proviene de las siglas que hacen referencia a "Application Programable Interface" o "Interfaz Programable de Aplicación" en español, y consiste en una serie de especificaciones que detallan como componentes de software pueden realizar interacciones con otros componentes. En ella se muestran las maneras en que se deben de realizar las llamadas para poder ser procesadas por el elemento, los posibles errores que pueden ocurrir y los posibles resultados. Con el uso de esto, se facilita la integración entre dos programas diferentes, teniendo acceso a las operaciones básicas realizables por cada uno sin tener que revisar a detalle el código [20].

5. RESTful

Es un estilo de arquitectura de software basado en los principios que se refieren como recursos interconectados, definidos y direccionados; comúnmente usan HTTP para obtener datos o generar operaciones sobre esos datos en formatos posibles, como XML y JSON [21].

SINOPSIS

La realización del trabajo instrumental de grado intitulado "Microservicios para la aplicación de usuarios de estacionamientos CityWallet", se fundamenta en el desarrollo de una arquitectura basada en microservicios. Esta nueva plataforma permite soportar las operaciones dentro del modelo de negocio de CityWallet a través de nuevos procesos definidos junto a los ya existentes acoplados a un paradigma serverless dentro de una plataforma en la nube.

El desarrollo se basó en el marco de trabajo Scrum junto a herramientas pertenecientes al desarrollo ágil, donde se llevaron a cabo 10 *sprints.* Dentro de los procesos cubiertos, se realizó el rediseño de las funcionalidades referentes a la gestión de usuarios, donde el usuario puede crear una cuenta en la plataforma y consultar su saldo actual. Se rediseñó el proceso de realización de transacciones de recarga y la consulta consumos, permitiendo a los usuarios del sistema, realizar recargas de tarjeta de crédito y consultar sus movimientos en estacionamientos. También se realizó el rediseño para gestión de calcomanía CityWallet usadas en los estacionamientos, por último se diseñó el proceso no existente de compra dentro de la tienda de productos

CAPÍTULO I – PLANTEAMIENTO DEL PROBLEMA

1. NECESIDADES DE LA EMPRESA

A lo largo de los dos últimos años CityWallet se ha convertido en una de las primeras empresas de emprendimiento que ofrece servicios de cartera digital y micropagos en Venezuela, actualmente su modelo de negocio se centra en el pago de estacionamientos con calcomanías que poseen tecnología NFC[1].

Hoy en día esta empresa se vincula estrechamente con el crecimiento tecnológico y desea aprovechar los distintos beneficios de adoptar tendencias actuales relacionadas al desarrollo de software, con el objetivo de obtener un óptimo rendimiento, monitoreo y ejecución de sus procesos para ofrecer servicios de alta calidad.

CityWallet cuenta con distintas aplicaciones que soportan diversas funcionalidades, dentro de este grupo existe una aplicación de estacionamiento, la cual permite a través de una cuenta CityWallet consultar las transacciones realizadas en los estacionamientos, consulta de saldo disponible, asociar transaction tools (dispositivo que posee la tecnología NFC y se encuentra asociado a una cuenta CityWallet, el cual permite realizar transacciones de entrada y salida en estacionamientos), entre otros.

[1] Near Field Communication: comunicación a campo corto en español, y se trata de una tecnología de comunicación inalámbrica de corto alcance y alta frecuencia que permite el intercambio de datos entre dos (2) dispositivos que cuenten con un chip NFC, en el marco referencial se amplía la idea.

Actualmente la empresa desea ampliar las funcionalidades de la aplicación de estacionamiento con una tienda de productos y servicios, entre ellos transaction tools[2] y planes para los distintos estacionamientos, donde el usuario pueda consultar y realizar la compra de dichos productos con el manejo de un carrito de compra.

El backend [3]de la aplicación de estacionamiento se encuentra desarrollado en una arquitectura monolítica, la cual se basa en la división por niveles o capas, de los mecanismos que soportan las diversas operaciones dentro de su lógica de negocio. La arquitectura monolítica es actualmente la más usada por las empresas, donde estas sacan ventaja debido a su simple desarrollo basado en solo un ciclo de vida junto con los entornos de desarrollo que tienen la capacidad de manejarse como una sola aplicación.

Uno de los principales problemas de esta arquitectura se evidencia cuando el sistema se vuelve más complejo al pasar de los años, el aumento de sus funcionalidades puede llegar a ser un factor negativo para su escalabilidad, existiendo la posibilidad de que un módulo en particular necesite escalar de manera diferente a su actual implementación, representando un cambio para toda la aplicación y de esta manera un incremento en el riesgo a fallar.

---

[2] Transaction tools: su término en español "herramienta de transacción" y hace referencia a la identificación de los dispositivos que pueden operar para hacer consumos dentro de la plataforma.
[3] Backend: conjunto de componentes dentro del sistema que se encarga de procesar la data que ingresa al sistema.

2. SOLUCIÓN PROPUESTA

Como solución se propone el rediseño de las funcionalidades existentes y el diseño de las nuevas funcionalidades a incluir en la aplicación de estacionamiento de CityWallet.

Las funcionalidades soportadas actualmente de la aplicación de estacionamiento se enfocan en brindar al usuario una herramienta de manejo y monitoreo, tanto de las transacciones realizadas y consulta de su cuenta, así como de los transaction tools asociados. Uno de los nuevos requerimientos a contemplar dentro de esta aplicación es la inclusión de una tienda mediante la cual exista la posibilidad de comercializar diversos productos; gracias a la necesidad de incluir nuevas funcionalidades y abarcar nuevas verticales de negocio, se propone el desarrollo del backend de la aplicación de estacionamientos de CityWallet basándose en una arquitectura de microservicios, la cual se basa en estructuras modulares y autónomas que cumplen una función en específico. Para el desarrollo de estas funcionalidades se requiere del análisis y definición de los procesos que cubren las necesidades actuales de esta vertical de negocio para su futura expansión.

La implantación de esta arquitectura tiene como objetivo colocar cada elemento de funcionalidad como un servicio totalmente separado donde se escala distribuyendo estos según se requiera para garantizar alta cohesión y bajo acoplamiento entre los componentes del sistema. Para el desarrollo de estas

funcionalidades se requiere del análisis y definición de los procesos que cubren las necesidades actuales del negocio de la empresa, así como también su futura expansión por medio de distintas verticales como pagos en transporte público, locales de venta y máquinas expendedoras.

Adicionalmente el desarrollo de los microservicios que sirven a la aplicación de estacionamiento, tiene como consecuencia la posibilidad de integración a la plataforma de CityWallet a través de APIs[4] públicas, trayendo como beneficio la posible interacción entre los distintos componentes de los sistemas externos.

El desarrollo de los microservicios y la integración con los distintos servicios se encontrará soportado por diversos servicios en la nube que hoy en día poseen una posición relevante en el mercado, estos deben soportar herramientas de desarrollo de funciones o microservicios en la nube, herramientas de APIs públicas, herramientas de monitoreo, entre otros.

A continuación se presenta un diagrama de la arquitectura con los componentes de la solución propuesta:

[4] Application Programming Interface: el término en español "Interfaz de Programación de Aplicación", actúa como la capa intermedia entre las peticiones que realizan las aplicaciones cliente y el sistema, se expone la idea en el marco referencial.

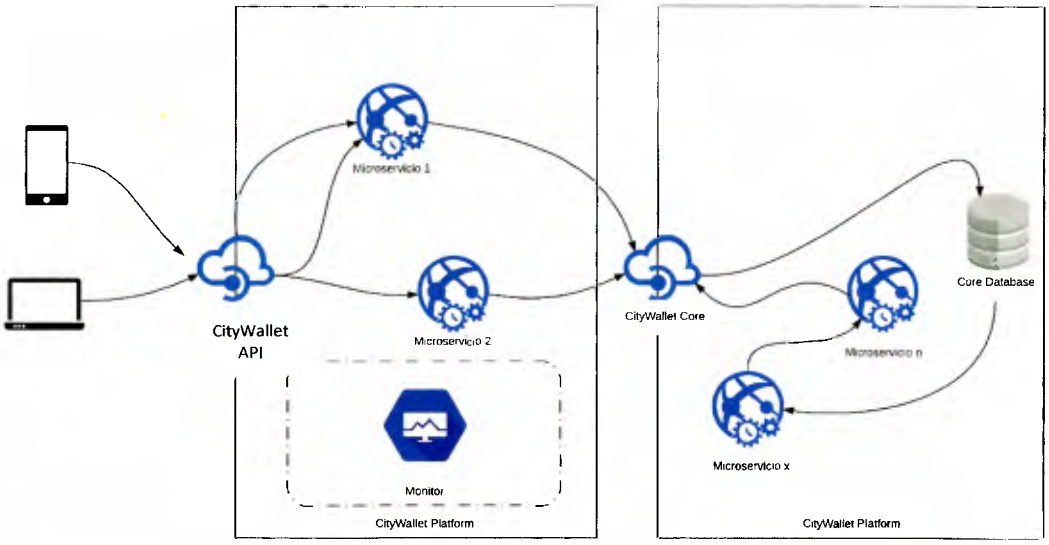


Figura 1. Arquitectura propuesta para la aplicación de CityWallet.

Fuente: [Elaboración propia]

Dentro de la arquitectura propuesta, se exponen distintos elementos que interactúan entre sí con el objetivo de satisfacer las operaciones de la empresa. En este caso se propone tener un primer elemento denominado CityWallet API que se comporte como capa de middleware, recibiendo las peticiones de los clientes de CityWallet, validando su sesión basada en un pool de usuarios existente y ejecutando el microservicio. Este segundo componente llamado microservicio, tiene como fin validar y procesar la petición enviada por la API, según la lógica del negocio del proceso. Así mismo, se maneja la base de datos por medio de una librería interna de la empresa que contiene implementaciones de operaciones básicas de lectura y escritura sobre las tablas. Durante el flujo de los

elementos, se plantea tener un ente de monitoreo para registrar el comportamiento de las distintas ejecuciones.

1. OBJETIVO GENERAL

Desarrollar los microservicios para la aplicación de usuarios de estacionamientos CityWallet.

2. OBJETIVOS ESPECÍFICOS

- Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios asociados a la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Desarrollar los microservicios para la gestión y reportes de los transaction tools asociados a cuentas CityWallet.
- Desarrollar los microservicios para la gestión y reportes de cuentas CityWallet.
- Desarrollar los microservicios para reportes de transacciones realizadas en los estacionamientos.
- Desarrollar los microservicios para la gestión y reportes de transacciones de recarga de saldo a cuentas CityWallet.
- Desarrollar los microservicios para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet.

2.1. Aportes funcionales

- Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet.
- Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet.

2.2. Aportes tecnológicos

- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y de su ejecución.
- Evaluar plataformas de computación en la nube que mejor se adapten al diseño y construcción de sistemas bajo una arquitectura de microservicios.

3. ALCANCE

Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios asociados a la aplicación de usuarios de estacionamientos CityWallet.

El desarrollo de las APIs se basa en la definición de mecanismos de comunicación a través de peticiones HTTP[5] o HTTPS[6] entre los distintos componentes del sistema. Este objetivo se enfoca en el desarrollo de cada una de las APIs que actuarán como capa de middleware entre los microservicios a desarrollar mencionados en los objetivos anteriores, estas interfaces de comunicación garantizarán el completo encapsulamiento de las llamadas entre los distintos aplicativos de CityWallet, en este caso, cubriendo con las necesidades en la manera de comunicarse dentro de las funcionalidades de gestión y reportes de los transaction tools de cuentas de usuario, gestión y reportes de cuentas de usuario, reportes de transacciones realizadas en los estacionamientos, gestión y reportes de transacciones de recarga de saldo a cuentas de usuario y tienda de productos. Estas APIs serán consumidas principalmente por aplicaciones de frontend.

Integrar la plataforma de microservicios a desarrollar con la API de CityWallet Core para sustentar sus necesidades.

En los microservicios que se van desarrollar se debe realizar una integración con un componente actual del sistema de CityWallet, este se denomina Core, esta integración tiene como objetivo encapsular todos los métodos de manejo de la base de datos principal de la empresa.

---

[5] Hypertext Transfer Protocol: protocolo que permite la transferencia de información entre diversos servicios y las aplicaciones cliente.

[6] Hypertext Transfer Protocol Secure: protocolo que permite la transferencia segura de información entre diversos servicios y las aplicaciones cliente.

Desarrollar los microservicios para la gestión y reportes de los transaction tools asociados a cuentas CityWallet.

Actualmente los usuarios pueden asociar varios transaction tools a sus cuentas como clientes, estos transaction tools son definidos como dispositivos con tecnología NFC que permiten realizar transacciones con los puntos de servicios que ofrece CityWallet. Este objetivo se enfoca en el desarrollo de los microservicios para la gestión y reportes de los transaction tools asociados a cuentas de usuario, los cuales cubran requerimientos de asociar un nuevo transaction tool a la cuenta del usuario, consultar los transaction tools asociados a dicha cuenta y cambiar el estatus un transaction tool (activo o inactivo).

Desarrollar los microservicios para la gestión y reportes de cuentas CityWallet.

El desarrollo de los microservicios para la gestión y reportes de cuentas CityWallet, tiene como objetivo a través de la aplicación de frontend, brindarle al usuario la posibilidad de registrar una nueva cuenta en el sistema, la consulta de su cuenta con información relevante como saldo disponible y realizar cambio de contraseña de su cuenta.

Desarrollar los microservicios para reportes de transacciones realizadas en los estacionamientos.

El sistema actual de CityWallet soporta operaciones que los usuarios realizan con respecto a las transacciones de estacionamiento, estas operaciones son de entrada y salida. Los microservicios a desarrollar en este objetivo deben

soportar funcionalidades de consulta de transacciones realizadas en estacionamientos y el detalle de cada una de ellas.

Desarrollar los microservicios para la gestión y reportes de transacciones de recarga de saldo a cuentas CityWallet.

Dentro de las necesidades de la aplicación de estacionamiento de CityWallet se encuentra el manejo de pagos y recargas de saldo a la cuenta del usuario. El manejo de pagos y recargas se realizará a través de la integración con el servicio proveedor de pagos digitales InstaPago[7], estas funcionalidades soportan las operaciones de recargas de saldo por tarjeta de crédito, así como reportes de consulta de recargas realizadas y el detalle de cada una de ellas. La realización de este objetivo se centra en el desarrollo de los microservicios que cubran los requerimientos anteriormente mencionados.

Desarrollar los microservicios para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet.

La tienda de productos y servicios, con el carrito de compra es la funcionalidad que ofrece la posibilidad de comprar planes y transaction tools dentro de la aplicación de CityWallet. El desarrollo de los microservicios para la tienda de la aplicación, engloban funcionalidades de consulta de lista de productos y el detalle de cada uno. Este objetivo también propone el desarrollo de los microservicios para gestión del carrito, que debe soportar operaciones para

[7] InstaPago: pasarela de pago de Banesco, que permite realizar operaciones por tarjeta de crédito a través de aplicaciones clientes.

agregar un producto al carrito, modificar la cantidad a comprar de los productos, remover productos del carrito de compra, remover la orden completa del carrito, así como también cumplir con reportes donde se muestre la lista de productos asociados a un carrito y consultar el detalle de un producto en específico del carrito. Adicionalmente estos microservicios a desarrollar cumplirán con la operación de checkout o confirmación de compra y la realización de la compra por tarjeta de crédito.

Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet.

CityWallet se ha visto en la necesidad de expandir sus funcionalidades gracias a las oportunidades de negocio que se han presentados en verticales de consumos en locales, máquinas expendedoras y estacionamientos. Debido a que la arquitectura monolítica del backend de la aplicación de estacionamientos de CityWallet no posee la capacidad de garantizar y cubrir las necesidades en cuanto a su escalabilidad y expansión según las necesidades de negocio mencionadas anteriormente, se plantea analizar y rediseñar los procesos existentes con el objetivo de acoplarlos de la mejor manera a la nueva arquitectura propuesta basada en microservicios.

Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet.

Hoy en día, CityWallet no posee un proceso para que los usuarios finales puedan consultar y comprar productos dentro de la tienda en la aplicación de

estacionamiento. Es por esto que se plantea analizar y diseñar un nuevo proceso que abarque la interacción del usuario final, en este caso, la compra de transaction tools y planes dentro de la aplicación de estacionamiento de CityWallet, con el objetivo que se pueda acoplar a la arquitectura de microservicios para soportar las funcionalidades mencionadas anteriormente y brindar una posibilidad de expansión en el mercado en el que se desenvuelven.

Integrar la plataforma de microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.

Con el cambio de arquitectura a microservicios, uno de los principales objetivos de CityWallet es integrar los distintos sistemas independientes que ya poseen. La integración que se propone busca estandarizar y acoplar el sistema propuesto junto con el ya existente, que se basa en el control de usuarios con roles y políticas de acceso bajo el servicios de Amazon Cognito. Este sistema de seguridad tiene como objetivo garantizar la seguridad de los microservicios a desarrollar.

Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y de su ejecución.

Dentro del desarrollo de los microservicios propuestos en este Trabajo Instrumental de Grado, CityWallet se ve en la necesidad de integrar estos con su sistema de monitoreo actual soportado por Amazon CloudWatch. El principal

aporte que busca esta integración es el monitoreo constante de la ejecución de los microservicios, trayendo como beneficio la personalización de métricas importantes para el desempeño operacional del sistema, registro en bitácora de los estados de los procesos que se ejecutan y la generación de gráficos y estadísticas para consultar data histórica.

Evaluar plataformas de computación en la nube que mejor se adapten al diseño y construcción de sistemas bajo una arquitectura de microservicios.

En la actualidad existen diversas plataformas de computación en la nube q ue apoyan el desarrollo de aplicaciones basadas en microservicios junto a herramientas necesarias para su integración, monitoreo, mantenimiento y seguridad. Este objetivo busca realizar una evaluación de distintas plataformas que ofrece el mercado para hacer uso de la que mejor se adapte a las necesidades de la empresa, mitigando el riesgo de elegir una que no se acople a los requerimientos y brindar recomendaciones a basadas en la evaluación realizada.

4. LIMITACIONES

- Los microservicios se desarrollarán utilizando lenguajes soportados por la plataforma de computación en la nube a elegir.
- Los microservicios para los reportes sólo cumplirán con las funcionalidades de backend.

- El desarrollo de los microservicios y la integración a sus distintas herramientas quedará soportado por la plataforma de computación en la nube a elegir.
- La integración para acoplar las políticas de acceso y roles de ejecución se realizará utilizando la plataforma Amazon Web Services: Cognito, que posee actualmente CityWallet.
- El manejo de las operaciones con la base de datos principal de CityWallet se encontrará encapsulada por la API de CityWallet Core.

5. JUSTIFICACIÓN

El desarrollo de la plataforma de CityWallet basado en servicios en la nube, es de gran importancia debido a que a través de esta se soporta la lógica de negocio de los distintos procesos que garantizan el valor creado por la empresa, apoyándose en las bondades de una arquitectura soportada en microservicios.

Gracias a las necesidades mencionadas, este trabajo instrumental de grado tiene el objetivo de rediseñar sus procesos de negocio y acoplarse a las nuevas tecnologías que manejan las plataformas en la nube, trayendo como beneficio los siguientes aportes tecnológicos:

- Resiliencia y flexibilidad: el desarrollo de un sistema basado en microservicios se apoya en la modularidad de sus funcionalidades, gracias a esta cualidad, se consigue el aislamiento de los errores, teniendo como consecuencia un aumento en la tolerancia a fallas del sistema en sí.

- Alta escalabilidad: la distribución de los componentes en la arquitectura de un sistema es un factor clave que determina su capacidad de crecer en el tiempo, debido a que los microservicios cumplen con un principio de descentralización no es necesario escalar el sistema como un todo sino como subcomponentes.
- Alta mantenibilidad: la independencia de las funcionalidades al desarrollar un sistema basado en microservicios permite que la realización de su mantenimiento a futuro, bien sea preventivo, correctivo o perfectivo resulta una tarea más sencilla pues el cambio o el ajuste de una funcionalidad no afecta en el comportamiento de las demás.
- Diversidad tecnológica: la naturaleza de los microservicios permite que en un mismo sistema existan funcionalidades desarrolladas en distintos lenguajes de programación, utilizando distintas librerías y accediendo a fuentes de datos distintas, otorgando libertad para desarrollar los requerimientos según sus necesidades. Gracias a esta importante cualidad, un cambio de lenguaje de programación o de tecnología utilizada, no resultaría tan crítico como podría resultar en un sistema monolítico.
- Despliegue independiente: ya que lo microservicios son representados como pequeñas porciones de funcionalidad trae consigo la facilidad de ser desplegables como unidades, esto genera la disminución de riesgos al momento de realizar procesos de despliegue con nuevas funcionalidades en el sistema.

CAPÍTULO II - MARCO REFERENCIAL

En el presente capítulo, se muestra detalladamente la base teórica fundamental para el completo entendimiento del presente trabajo, se exponen los conceptos requeridos para el desarrollo de los objetivos así como la metodología empleada para el cumplimiento de los mismos.

1. FinTech

La palabra FinTech es un término procedente del idioma inglés y compuesto por las palabras Finance (finanzas) y Technology (tecnología), y que por lo tanto resume y define todas aquellas empresas de servicios financieros que se apalancan en las últimas tendencias tecnológicas para ofrecer productos y servicios financieros innovadores que facilitan la vida cotidiana de los bancos y usuarios. Dentro del portafolio de servicios financieros que engloban la disrupción de las FinTech se encuentran: banca digital, créditos online, intercambio de divisas online, pagos móviles, entre otros [1].

Entre los principales características de las iniciativas FinTech está la intermediación en aspectos financieros relacionados a: transferencias de dinero, préstamos, compra y venta de títulos, asesoramiento financiero e inversiones; sin embargo, la oferta de estos productos está atado al objetivo principal de toda FinTech, el cual es diferenciarse de los servicios financieros tradicionales ofrecidos por la banca y las telecomunicaciones, es por ello que las iniciativas FinTech enfocan sus esfuerzos a [1]:

- Ofrecer nuevos servicios financieros con el apoyo tecnológico. Así, sirven la comodidad de una aplicación o la atractiva interfaz similar a una red social para brindar a los clientes productos anteriormente reservados a la banca tradicional.
- Añadir valor a los servicios financieros actuales, con un trato más personalizado, detallado e inmediato.
- Ofrecer productos financieros totalmente online, sin necesidad de desplazamiento del consumidor, utilizando los dispositivos inteligentes como intermediadores.
- Reducir de modo eficiente a nivel de costes los servicios actuales.
- Democratizar el acceso a una mayor cantidad de servicios financieros, así como favorecer una mayor transparencia en la oferta financiera
- Utilizar la tecnología para dar mejor soluciones a los servicios financieros.

2. Arquitectura de software

La arquitectura de software se define como la estructura del sistema, la cual incluye de manera macro los componentes principales del mismo, la conducta de esos componentes y su interacción para alcanzar el objetivo por el cual fue creado el sistema [3].

2.1. Microservicios

El término microservicios ha surgido entre los últimos años para describir la forma de diseñar pequeños servicios de aplicaciones centrados en el cliente, escalables y con control de versiones independiente que establecen comunicación

entre sí mediante protocolos estándar con interfaces previamente definidas, la comunicación ocurre en su mayoría mediante una API de recursos HTTP. [4].

En la **Figura 2**, se detalla el funcionamiento de los microservicios y como dentro de su arquitectura cada uno de estos pequeños servicios ejecutan su propio proceso y comunicación.

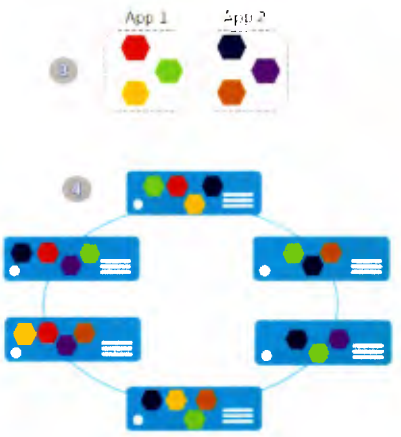


Figura 2. Ejemplo de una arquitectura de microservicios.

Fuente: Microsoft Azure (2017) [4]

Los servicios son definidos en función a las capacidades empresariales y son implementados con despliegue completamente automatizado. La gestión de estos servicios es centralizada y puede escribirse en diversos lenguajes de programación, así como utilizar diferentes tecnologías de almacenamiento de datos [5].

2.2. Serverless Computing

Serverless Computing o su término en español "informática sin servidor"

hace referencia a un modelo informático sin la existencia de servidores, aún cuando estos existen, no están presentes para los desarrolladores, de forma que ni la empresa ni el equipo de desarrolladores tiene que preocuparse por su funcionamiento. La computación o informática sin servidores se basa básicamente en la reducción de esfuerzos de mantenimiento y costos por posesión de servidores [6].

3. Cloud Computing

Cloud Computing o en su término en español "Informática en la Nube" es la entrega de trabajo bajo demanda de: potencia informática, almacenamiento de bases de datos, aplicaciones y otros recursos de tecnología a través de Internet bajo un esquema de precios por consumo [7].

La informática en la nube se compone de tres (3) tipos, que se definen como [8]:

- Infraestructura como servicio (IaaS): contiene los bloques de creación fundamentales para la tecnología informática (TI) en la nube y proporciona acceso a las características de redes, a los equipos (virtuales o en software dedicado) así como espacio de almacenamiento de datos.
- Plataformas como servicio (PaaS): administran la infraestructura subyacente (normalmente hardware y sistemas operativos) de las compañías, permitiéndoles centrarse en la implementación y la administración de sus aplicaciones.
- Software como servicio (SaaS): proporciona un producto completo que el

proveedor del servicio ejecuta y administra. En la mayoría de los casos, el Software como servicio se refiere a aplicaciones de usuario final.

3.1. AWS

Amazon Web Services (AWS) o en su término en español "Servicios Web Amazon" son un conjunto de productos globales basados en la nube, incluidas aplicaciones de informática, almacenamiento, bases de datos, análisis, redes, móviles, herramientas para desarrolladores, herramientas de administración, IoT, seguridad y empresariales. Estos servicios ayudan a las empresas a avanzar con mayor rapidez, reducir los costos de TI y escalar; respaldando una amplia variedad de cargas de trabajo, como las aplicaciones web y móviles, el desarrollo de juegos, el almacenamiento y procesamiento de datos, el almacenamiento en general, el archivado, entre otras [9].

3.1.1. AWS Lambda

AWS Lambda es un servicio de informática sin servidores que ejecuta el código como respuesta a eventos y administra automáticamente los recursos informáticos subyacentes. Lambda ejecuta el código en una infraestructura informática de alta disponibilidad y ejecuta la administración integral de los recursos informáticos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, la implementación de parches de seguridad y código, así como la monitorización y los registros. Lo único que tiene que hacer es proporcionar el código [10].

3.1.2. Amazon DynamoDB

Amazon DynamoDB es un servicio en la nube que se encarga de la gestión de base de datos NoSQL, su manejo interno permite almacenar modelos de datos flexibles para garantizar alta escalabilidad a medida que las necesidades del negocio crecen y mantener latencias constantes para no afectar el rendimiento del manejador de datos.[11].

3.1.3. Amazon API Gateway

Amazon API Gateway es un servicio totalmente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, la monitorización y la protección de la API a cualquier escala. Se debe crear una API que haga las veces de "puerta delantera" para que las aplicaciones obtengan acceso a datos, lógica de negocio o funcionalidades desde los servicios de back-end, como cargas de trabajo ejecutadas en Amazon Elastic Compute Cloud (Amazon EC2), código ejecutado en AWS Lambda o cualquier aplicación web. Amazon API Gateway administra todas las tareas involucradas en la aceptación y el proceso de hasta cientos de miles de llamadas de API simultáneas, entre ellas la administración del tráfico, el control de autorizaciones y acceso, la monitorización y la administración de la versión de API [12].

3.1.4. Amazon Cloudwatch

Amazon CloudWatch es un servicio de monitorización de los recursos de la nube de AWS y de las aplicaciones que se ejecutan en AWS. Sirve para recopilar y realizar el seguimiento de métricas y logs, establecer alarmas y reaccionar

automáticamente a los cambios en recursos, adicional, se utiliza para obtener visibilidad para todo el sistema sobre la utilización de recursos, el desempeño de las aplicaciones y el estado de funcionamiento. Las principales características de Amazon Cloudwatch son [13]:

- Monitorización de Amazon EC2
- Monitorización de otros recursos de AWS
- Monitorización de métricas personalizadas
- Monitorización y almacenamiento de logs
- Definición de alarmas
- Visualización de gráficos y estadísticas
- Monitorización y reacción ante cambios en los recursos

3.1.5. Amazon Cognito

Amazon Cognito permite agregar el registro y el inicio de sesión de forma sencilla en aplicaciones web y móviles; brinda la opción de autenticar a los usuarios a través de proveedores de identidad social, como Facebook, Twitter o Amazon, con soluciones de identidad SAML o a través de su propio sistema de identidad. Además, permite guardar datos de forma local en los dispositivos de los usuarios, lo que permite que las aplicaciones funcionen incluso cuando los dispositivos están fuera de línea [14]. En la **Figura 3**, se muestra cómo ocurre la administración, autenticación y sincronización de los usuarios entre dispositivos.

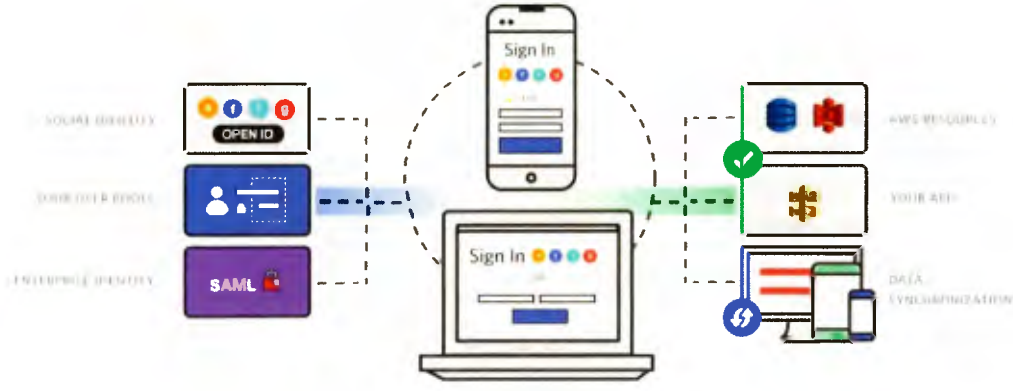


Figura 3. Administración, autenticación y sincronización con Amazon Cognito.

Fuente: Amazon Web Services 2017 [15]

3.2. Microsoft Azure

Azure es un conjunto de servicios integrados en la nube donde los desarrolladores y profesionales de tecnología de información pueden construir, desplegar y gestionar sus aplicaciones. Microsoft Azure ofrece distintas herramientas de desarrollo que garantizan a sistemas externos una integración segura y escalable para el funcionamiento de su lógica de negocio [35].

3.3. GCP

Google Cloud Platform se encarga de ofrecer distintas herramientas en la nube para el desarrollo de aplicaciones y sitios web, gestión y análisis de datos, tiene como objetivo sustentar de la mejor manera los procesos de las empresas en el mercado [36].

4. Transaction tool

El Transaction tool o su término en español "herramienta de transacción" es una terminología interna dentro de la empresa creadora de la aplicación CityWallet, que hace referencia a la identificación de los dispositivos que pueden operar para hacer consumos dentro de la plataforma y efectuar un cash-out dentro de la misma. La herramienta permite: activar, desactivar, asociar y consultar la lista de dispositivos.

4.1. Cash-in

El cash-in es una terminología interna dentro de la empresa creadora de la aplicación CityWallet que hace referencia a las recargas de saldo por parte de los usuarios, las cuales, pueden provenir de los siguientes canales para ser consumidos dentro de la cartera digital:

- Pasarela de pago: para recargas vía tarjeta de crédito.
- Transferencia bancaria: para las recargas efectuadas mediante depósitos o transferencias electrónicas a las cuentas bancarias mencionadas en la aplicación (web y móvil).
- Efectivo: para depósitos en moneda local efectuados en los puntos de la red afiliados a CityWallet (aún no disponible).
- SMS: para transferencia de saldo "aire" disponible por los usuarios en sus operadoras de telefonía celular y el cual, mediante un mensaje de texto se convierte en saldo disponible para consumo en la aplicación CityWallet (aún

no disponible).

4.2. Pasarela de pago

Una pasarela de pago es un proveedor de servicios de aplicación de comercio electrónico, que valida pagos en todo tipo de negocio virtual mediante un punto adquirente electrónico que procesa tarjetas de crédito y débito de diferentes emisores.

Las pasarelas de pago se encargan de cifrar la información sensible del pago (como el número de tarjeta de crédito) para garantizar la seguridad de la transacción, y transfiere la información de forma segura entre el portal de pago (que puede ser un sitio web donde se vende algo) y el procesador de interfaz o banco adquiriente [22].

4.3. Cash-out

El cash-out es una terminología interna dentro de la empresa creadora de la aplicación CityWallet que hace referencia a las salidas de dinero provenientes del saldo depositado dentro de la cartera digital, éstas salidas provienen de las siguientes modalidades:

- Consumos en comercios.
- Transferencias entre personas.
- Avances de efectivo: según las políticas establecidas en los términos y condiciones de uso de la plataforma al momento de afiliación.

CAPITULO III

1. MARCO METODOLÓGICO

Las técnicas y herramientas que se emplean a la hora de organizar un trabajo dependen directamente de la naturaleza de sus requerimientos. Es por esto que para efectos de éste trabajo instrumental de grado se decidió optar por una metodología ágil que permita cierta flexibilidad en las fases del ciclo de vida del producto, para así poder realizar ajustes de detalles en los requerimientos, además de la reestructuración de trabajos ya realizados durante el proyecto. Otra característica importante es la necesidad de generar productos funcionales de manera incremental, que puedan ser probados y corregidos con la finalidad de obtener un producto final que cumpla con los estándares de calidad exigidos.

1.1. Scrum

Scrum es un efectivo enfoque en la gerencia de proyectos que fue pensado para desarrollo de software, pero que tiene aplicabilidad en otras disciplinas incluyendo manufactura y servicios. La orientación de Scrum es sencilla y está enmarcada en el trabajo en equipo, la comunicación y la auto-organización.

Las bondades de Scrum radican en que se crea balance entre aquellos solicitando el trabajo y los equipos produciendo los resultados. Además, con Scrum se establece una línea bien demarcada entre los clientes y los usuarios. Por si fuera poco, Scrum también construye un ambiente de trabajo que empodera a todos los equipos de una organización y/o a todos los miembros de un equipo.

No en vano muchas famosas startups deben su éxito a haber aplicado las reglas de Scrum. Por ejemplo, Zappo´s, el gigante del comercio electrónico de ropa y zapatos, desarrolló su primera propuesta de producto en tan sólo 12 semanas [23] [24]. El secreto de su éxito radicó en dividir los esfuerzos en períodos de una semana con objetivos concretos. Estos períodos, son lo que Scrum denomina *sprints*. La misma filosofía y la misma división de tiempo se aplicaron en CityWallet.

Otro gigante que ha probado el éxito de Scrum es Spotify. Para mantenerse adelante de Apple con Apple Music, Google con Google Play Music All Access, entre otros; Spotify se divide en pequeños grupos que se manejan como startups individuales y no como departamentos. En consecuencia, la frecuencia en la entrega de nuevos productos o actualizaciones que presenta Spotify es muchísimo más rápida que la de compañías con largas cadenas de aprobación o burocracias. Scrum da vida y autonomía a los equipos de trabajo para que sean estos los que diseñen la estrategia para alcanzar sus objetivos y, además, sorteen los eventuales bloqueos que puedan presentarse.

Con base en lo previamente expuesto podemos decir que haber trabajado con Scrum impulsó a CityWallet a desarrollar en tiempo record varias versiones de la aplicación de usuario de estacionamientos; variaciones importantes en la arquitectura de la plataforma transaccional; iteraciones en el modelo de negocio y además, mejoras sustanciales en el diseño de los cajetines lectores ubicados en los estacionamientos. Todo esto con un pequeño equipo de 6 personas.

Para garantizar la eficiencia del trabajo, Scrum establece varios pilares, procesos y técnicas que describiremos a continuación.

1.1.1. Pilares

Scrum es un marco de trabajo que empodera a equipos para que afronten problemas complejos con alta productividad y calidad. Para ello, el trabajo, la documentación, los reportes y las tareas deben cumplir tres principios:

- **Transparencia:** los aspectos más relevantes del trabajo deben ser visibles para los responsables de los resultados. Para que la transparencia realmente exista, es importante que el equipo determine parámetros para esquematizar la información de forma que sea comprensible por todos. Un concepto que toma gran importancia es el de las tareas "hechas"; consecuentemente, estas tareas deberán cumplir con una serie de requisitos para considerarse como culminadas y con la calidad exigida.
- **Inspección:** el progreso del desarrollo debe pasar por varias revisiones para garantizar que el ritmo de trabajo no se vea afectado por bloqueos que comprometan la consecución de los objetivos. Estas inspecciones son planificadas y constantes y no deben entorpecer el trabajo.
- **Adaptación:** es importante comprender cómo se están desarrollando las tareas para detectar de forma temprana potenciales desviaciones o bloqueos. Esto con la idea de ajustar el plan y alcanzar los objetivos planteados [25].

1.1.2. Eventos

Para poder inspeccionar y adaptar los *sprints*, Scrum establece varios sucesos o reuniones con duración limitada para revisar el avance del trabajo. La idea es que estos eventos no entorpezcan las labores de desarrollo, sino que sean reuniones dinámicas y eficientes que brinden un diagnóstico de grupal de las labores.

Todos los eventos suceden alrededor de *sprints*, que no es más que el período del tiempo en el cual se supone que el equipo de desarrollo alcanzará metas determinadas. La duración de los *sprints* varía dependiendo de cada empresa.

Los eventos que establece Scrum son:

- **Planificación del *sprint*:** en este evento se determinan los objetivos del *sprint*; estos objetivos están directamente asociados con la capacidad de crecimiento del producto, es decir, el producto final se desglosa en piezas más pequeñas que se desarrollarán de distintos momentos [26].
- **Scrum diario**: es una reunión corta y diaria que tiene como fin entender si el ritmo del desarrollo va acorde a la planificación establecida. Durante esta junta se examinan las acciones que el día anterior contribuyeron positivamente con el trabajo; también se determina qué se hará durante ese día para alcanzar las metas y si hay algún impedimento frenando las acciones del equipo [27].
- **Revisión del *sprint*:** el evento se celebra al final de cada *sprint*. En la

reunión se junta al equipo de desarrollo con el maestro scrum y se hace un cierre formal de todas las tareas. El resultado de esta reunión arroja un estatus claro del desarrollo del producto. Los incrementos registrados en ese *sprint* se suman a todas las funcionalidades previamente hechas.

- **Retrospectiva del *sprint*:** en esta ocasión, el equipo se auto inspecciona en compañía del maestro scrum. El objetivo es entender:
    - La calidad e interrelación entre las personas, los procesos y las herramientas.
    - Determinar cuáles tareas y/o elementos fueron los más exitosos y cuáles tienen oportunidad de mejora.
    - Implementar las mejoras en el próximo *sprint* [28].

1.1.3. Artefactos

Este componente de Scrum representa cada tarea o funcionalidad que debe desarrollarse para completar el producto final. Los artefactos son [29]:

- **El *backlog* de producto:** es un listado de todo lo que el producto pudiera necesitar. Pueden tener la forma de funcionalidades específicas, asignaciones o tareas. Esta es una lista que muta con el tiempo, pues se nutre de la retroalimentación con el cliente, del contacto con los usuarios y del criterios de Dueño del Producto, quien es el responsable de mantener esta lista actualizada y eficiente a efectos conseguir los objetivos planteados por quienes demanden el producto final

- **El *backlog* del *sprint*:** esta lista contiene las tareas y/objetivos del *sprint* y la misma se alimenta del *backlog* del producto. La lista se conforma con la colaboración activa del equipo de desarrollo y del maestro scrum, siendo este último el responsable de hacer que el trabajo del equipo de desarrollo fluya hacia los objetivos.

1.1.1. Equipo

El equipo Scrum se conforma por el dueño del producto, el equipo de desarrollo y el maestro scrum, todos, deben contar con dos características indispensables: ser auto-organizativos e interfuncionales. Esta combinación es una apuesta directa al auto-liderazgo como base de la eficiencia. A continuación se describen sus características [30]:

- **El dueño del producto:** es la mente que maqueta el producto final y, por lo tanto, es quien tiene la capacidad de fragmentarlo en las pequeñas partes que construyen el *backlog* de producto. El dueño del producto interactúa con otros agentes interesados en el producto, como los clientes y los usuarios.
- **El equipo de desarrollo:** conformado por un grupo de profesionales de distintas áreas. Este equipo tiene la tarea de construir el producto, basándose en su propia visión para esquematizar los *sprints* de trabajo. Su objetivo es completar la mayor cantidad de tareas posibles durante el *sprint*.
- **El maestro scrum:** esta figura apoya al equipo de desarrollo en la consecución de sus objetivos. Puntualmente, el maestro Scrum se dedica a

asegurar que las tareas puedan completarse en el tiempo determinado y con la calidad exigida. El maestro Scrum está al servicio del Equipo de Desarrollo.

En el caso de CityWallet, el Ing. Ramón Ginéz fue quien asumió el rol del dueño del producto, por otro lado, el Ing. Félix Fernández cumplió el papel del maestro Scrum.

1.2. Marco de trabajo ágil de CityWallet

Como se mencionó anteriormente, CityWallet se basa en los fundamentos de Scrum, adoptando el desarrollo ágil para afrontar los requerimientos del negocio y cumplir con hitos en el mínimo tiempo y la mayor productividad posible. Gracias a esto, la empresa ha decidido adaptar Scrum a distintas aplicaciones del desarrollo ágil existentes hoy en día.

El marco de trabajo resultante que se aplica actualmente, cuenta con distintos componentes partiendo de la estructura de un *sprint*. Para CityWallet cada *sprint* se encuentra compuesto por un *backlog*, a su vez este se encuentra dividido en hitos, que se definen como puntos de referencia a lo largo del *sprint* y representan un logro o entregable según los requerimientos establecidos. Estos hitos poseen actividades con las cuales se deben cumplir para lograr hito.

Este marco de trabajo parte de los objetivos a desarrollar en el alcance de este trabajo instrumental de grado, listando y agrupando los procesos que poseen relación entre sí, con la finalidad de definir grupos de funcionalidades o procesos llamados "Épicas" [31]. Estas épicas se componen por hitos y pueden representar

a uno o más *sprints*; cada hito representa un logro en una fecha o momento determinado, especificando que es lo que se quiere lograr para obtener como resultado parte del producto.

1.2.1. Mapa

A continuación se listan los *sprint*s que se realizaron según los objetivos a cubrir:

- ***Sprint* 0:**
  - Evaluar plataformas de computación en la nube que mejor se adapten al diseño y construcción de sistemas bajo una arquitectura de microservicios.
- ***Sprint* 1:**
  - Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet, específicamente a las funcionalidades de agregar, consultar y modificar una herramienta de transacción.
  - Desarrollar los microservicios para la gestión y reportes de los transaction tools asociados a cuentas CityWallet, relacionados a los procesos de agregar, consultar y modificar.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de

acceso para los microservicios de agregar, consultar y modificar una herramienta de transacción.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 2:**
  - Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet, específicamente en la funcionalidad de agregar una transacción de recarga de saldo.
  - Desarrollar los microservicios para la gestión de transacciones de recarga de saldo a cuentas CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de agregar una transacción de recarga de saldo.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 3:**
  - Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet, específicamente en la funcionalidad de consultar la lista y el detalle de transacciones de recarga y estacionamientos.
  - Desarrollar los microservicios para los reportes de consulta y detalle de transacciones de recarga y estacionamientos de cuentas CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de consulta y detalle de transacciones.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 4:**
  - Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet, específicamente a la funcionalidad de agregar una cuenta de usuario.
  - Desarrollar los microservicios para la gestión de cuentas de usuarios CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de agregar una cuenta de usuario.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 5:**
  - Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet, específicamente a la funcionalidad de consultar una cuenta de usuario y su balance.
  - Desarrollar los microservicios para los reportes de cuentas de usuarios CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de consultar una cuenta de usuario y su balance.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 6**:
  - Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet, específicamente la funcionalidad de agregar un carro de compra.
  - Desarrollar los microservicios de agregar un carro de compra en la tienda de productos para aplicación de usuarios de estacionamientos CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de agregar un carro de compra.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 7**:
  - Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet, específicamente la funcionalidad de consulta y detalle de un carro de compra.
  - Desarrollar los microservicios de consulta y detalle de un carro de compra en la tienda de productos para aplicación de usuarios de estacionamientos CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de consulta y detalle de un carro de compra.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 8:**
  - Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet, específicamente las funcionalidades de modificación de un carro de compra, consulta y detalle de productos.
  - Desarrollar los microservicios de modificación de un carro de compra, consulta y detalle de productos de la tienda para aplicación de usuarios de estacionamientos CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de modificación de un carro de compra, consulta y detalle de productos.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

- ***Sprint* 9:**
  - Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet, específicamente las funcionalidades de pago de un carro de compra.
  - Desarrollar los microservicios de pago de un carro de compra en la tienda de productos para aplicación de usuarios de estacionamientos CityWallet.
  - Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios de pago de un carro de compra.

- Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.
- Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.
- Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y ejecución de los mismos.

Los objetivos en común como desarrollo de las APIs según el proceso, y la integración a CityWallet Core se realizaron en cada *sprint*, estos abarcan actividades de desarrollo progresivo. De igual manera se realizaron rediseños de procesos existentes o diseños de nuevos procesos en base al proceso a cubrir.

CAPÍTULO IV - DESARROLLO

El desarrollo se basó en el marco de trabajo expuesto anteriormente, donde se dividió en *sprint*s cada una de los objetivos a cubrir. El inicio de esta fase parte de un *sprint* 0, este se enfoca en la identificación, junto con el dueño del producto, los procesos según los objetivos planteados. Así mismo, dentro de este *sprint* inicial también se encuentra evaluación de plataformas, que indica cual es la mejor alternativa para adoptar una arquitectura basada en servicios en la nube.

1. Sprints

En esta fase de desarrollo cada *sprint* posee una duración de dos semanas y el esfuerzo asignado a cada historia es medido en horas productivas. En el proceso de planificación se calculan las horas de cada historia de usuario junto al equipo Scrum y se valida que los esfuerzos no superen las horas de dedicación del equipo de desarrollo, en el caso de CityWallet se definen un lapso tope de la duración del *sprint* (80 horas).

1.1. Sprint 0

Este *sprint* inicial, tuvo como objetivo planificar el desarrollo que se realizó posteriormente, así como también evaluar las plataformas según los criterios que se expondrán más adelante.

En el **Anexo D** en la **Tabla 12** se muestra el *backlog* del producto donde se encuentra la lista de las épicas junto a los hitos que la

componen, resultado de la planificación inicial que identificó el aspecto general del producto.

Evaluación de plataformas

Criterios de selección

En el presente trabajo instrumental de grado se expone una evaluación que se enfoca en la selección de una plataforma en la nube. Para esto se han tomado distintos criterios que determinan qué plataforma se adapta mejor a los requerimientos de la empresa.

En la primera fase de la evaluación, se ha realizado una búsqueda entre los aspectos con mayor relevancia que tienen impacto en el desarrollo de los microservicios sobre una plataforma en la nube. Dentro de los criterios evaluados, se han seleccionado ocho de ellos que se consideran claves para la implantación de esta nueva arquitectura.

A continuación se listan los criterios que se tomaron en cuenta para selección de la plataforma:

1. Soporte de almacenamiento de datos
2. Soporte de administración de APIs
3. Soporte de administración de microservicios
4. Soporte de administración de credenciales de usuarios
5. Soporte de lenguajes de programación
6. Estabilidad y tiempo en el mercado

Personal capacitado dentro de la CityWallet

Selección de plataformas

Previo al estudio de plataformas, en esta segunda fase se realizó una búsqueda entre las principales empresas que proveen diferentes servicios en la nube y promueven la migración de sistemas a este nuevo paradigma.

Dentro de esta búsqueda superficial se tomó en cuenta empresas que tienen alto posicionamiento en el mercado, junto a experiencias de empresas emprendedoras que han adoptado este tipo de arquitectura. Se debe mencionar también que parte de esta investigación fue orientada a un grupo de empresas en específico debido a que las necesidades de CityWallet ya se encontraban establecidas, obteniendo un conjunto de 3 plataformas en la nube.

A continuación se listan las plataformas resultantes para su próximo estudio:

- Amazon Web Services (AWS[8])
- Google Cloud Platform (GCP[9])
- Microsoft Azure (Azure[10])

[8] Amazon Web Services: plataforma de Amazon en la nube, sus servicios se listan en el marco referencial.
[9] Google Cloud Platform: plataforma de Google en la nube.
[10] Microsoft Azure: plataforma de Microsfot en la nube.

Estudio de plataformas

El estudio de plataformas en la nube es la última fase del proceso de evaluación, esta tiene como objetivo el análisis que se apoya en la herramienta de tabla comparativa[11] donde se expone las semejanzas y diferencias entre las plataformas en la nube y los criterios seleccionados en la **Tabla 1**.

Tabla 1. Comparativa de semejanzas y diferencias entre plataformas en la nube.

Fuente: [33] [34] [35] [36]

Plataformas	AWS	GCP	Azure
BD	Posee: • SQL[12] • No SQL[13]	Posee: • SQL (Versión de prueba) • No SQL	Posee: • SQL • No SQL
API	Posee	Posee	Posee
Credenciales	Posee	Posee	Posee
Microservicios	Posee	Posee	Posee
Lenguajes de programación	• Java • Node.js • C# • Javascript • Python	• Node.js	• C# • F# • Node.js • Python • PHP
Estabilidad y tiempo en el mercado	Establecida desde 2006, con un aproximado de 2000 aplicaciones en su cartera de clientes	Establecida desde 2011, con un aproximado de 160 aplicaciones en su cartera de clientes	Establecida desde 2010, con un aproximado de 700 aplicaciones en su cartera de clientes
Personal capacitado	1	0	0

[11] Tabla comparativa: herramienta que se utiliza para exponer, según aspectos o criterios, comparaciones entre alternativas.
[12] Structured Query Language: lenguaje que permite realizar operaciones sobre un sistema de gestión de base de datos relacional.
[13] No Structure Query Language: termino que refiere a las bases de datos no relacionales.

Otra de las herramientas en la cual se basa este análisis es a través de una matriz de ponderación[14], que busca estudiar los aspectos que se puedan adaptar al desarrollo de la plataforma de microservicios según las necesidades particulares de la empresa.

En la **Tabla 2**, se lista la valoración numérica con la cual se clasificó cada criterio, siendo 1 = Malo, 2 = Regular, 3 = Bueno, 4 = Muy bueno.

Tabla 2. Matriz ponderación según criterios.

Fuente: [Elaboración propia]

Plataformas	AWS	GCP	Azure
BD	4	2	4
API	4	4	4
Credenciales	4	4	4
Microservicios	4	3	4
Lenguajes de programación	4	3	4
Estabilidad y tiempo en el mercado	4	2	2
Personal capacitado	4	2	1
Puntaje total	28	20	23

Sobre el criterio relacionado al soporte de almacenamiento de datos, el puntaje otorgado a cada servicio se basó en Jess Panni [37], argumentando que AWS y Azure poseen servicios de bases de datos altamente gestionado,

[14] Matriz ponderación: herramienta que se utiliza para dar puntaje a opciones o alternativas según aplicación de criterios.

contando con funcionalidades para gestión de modelos relacionales, no relacionales, replicación y almacenamiento masivo de datos. Por último, a pesar que GCP también posee las funcionalidades de las otras plataformas, su gestión para bases de datos relacionales se encuentra actualmente en una versión de prueba.

Con respecto al soporte de administración de APIs, las tres plataformas poseen la misma ponderación, esto se justifica sobre lo mencionado por Juan María Fiz [38] en los servicios de AWS y GCP y las funcionalidades de AWS y Azure expuestas por Wilson Mar [39], indicando la semejanzas en el uso de sus servicios de APIs y posibilidad de su implementación, despliegue, gestión y monitoreo.

Según Rajat Bhargava [40] y Greg Keller [41], las tres plataformas ofrecen funcionalidades de identificación, control de acceso y monitoreo dentro de sus servicios referentes a gestión de credenciales de usuario, siendo la única diferencia entre estos servicios es el tiempo en el que fueron lanzados al mercado. Es por esto que cada plataforma posee la misma puntuación en este criterio.

En el criterio de administración de microservicios AWS y Azure tienen la misma puntuación debido a que sus servicios de cómputo son similares y cubren con necesidades básicas como, flexibilidad de pruebas e integración con los demás servicios y estabilidad en sus funcionalidades. Así mismo, se le colocó la menor ponderación a GCP, debido a que se están ampliando sus servicios y por ahora se mantiene en una fase de prueba estable. Este criterio se basó según

Jess Panni [42] y Stephen Grider [43]. Posteriormente se ponderaron los criterios de lenguajes soportados junto a estabilidad y tiempo en el mercado, esta puntuación se asignó según la **Tabla 1**.

Por último se le colocó el mayor puntaje a AWS en el criterio de personal capacitado dentro de la empresa, debido a que CityWallet posee un arquitecto en soluciones certificado por Amazon.

Recomendaciones

En el estudio realizado a través de la matriz ponderada, se puede observar cada plataforma se adaptan a las necesidades mínimas para el soporte de la plataforma de CityWallet en la nube.

A pesar que estas tres plataformas cuentan con servicios de integración para almacenamiento de datos, gestión de APIs y microservicios, cada uno de sus servicios se diferencian ligeramente en las especificaciones tecnológicas.

La gestión de almacenamiento de datos es un aspecto clave en el desarrollo de los microservicios, Google posee bases de datos no relacionales en su versión más estable que se encuentra disponible y solo soporta bases de datos relacionales en su versión de prueba, lo cual es una desventaja debido a que sería una limitante para la escalabilidad de los sistemas de CityWallet que se deban soportar en bases de datos relacionales.

Azure como alternativa posee alrededor de cuatro lenguajes de programación, dentro de ellos Java, Node.js, .Net y Python, teniendo como ventaja que la gestión de microservicios esté soportada sobre los lenguajes de

programación más demandados [44] y de mayor impacto en el mercado, posicionándose en la segunda alternativa más viable en el estudio. Esta plataforma en la nube posee, al igual que AWS, el manejo de bases de datos relacionales y no relacionales, acoplándose casi en su totalidad a los requerimientos para el desarrollo de los microservicios.

Amazon Web Services tiene la mayor ponderación según los criterios seleccionados indicando que cada uno de sus servicios posee las características y especificaciones requeridas para soportar los procesos a desarrollar. Parte de la puntuación a favor de AWS, se inclina a su trayectoria en el mercado desde 2006 [45] y en las empresas que soportan sus operaciones en su plataforma en la nube como Netflix, Spotify, MLB, Airbnb, entre otras [46]. Adicionalmente, CityWallet posee un especialista certificado en arquitecto de soluciones de AWS, este desempeña labores de configuración de ambientes, asesoría en caso de necesitarlo junto a soporte técnico directo con personal de Amazon.

Arquitectura de la solución

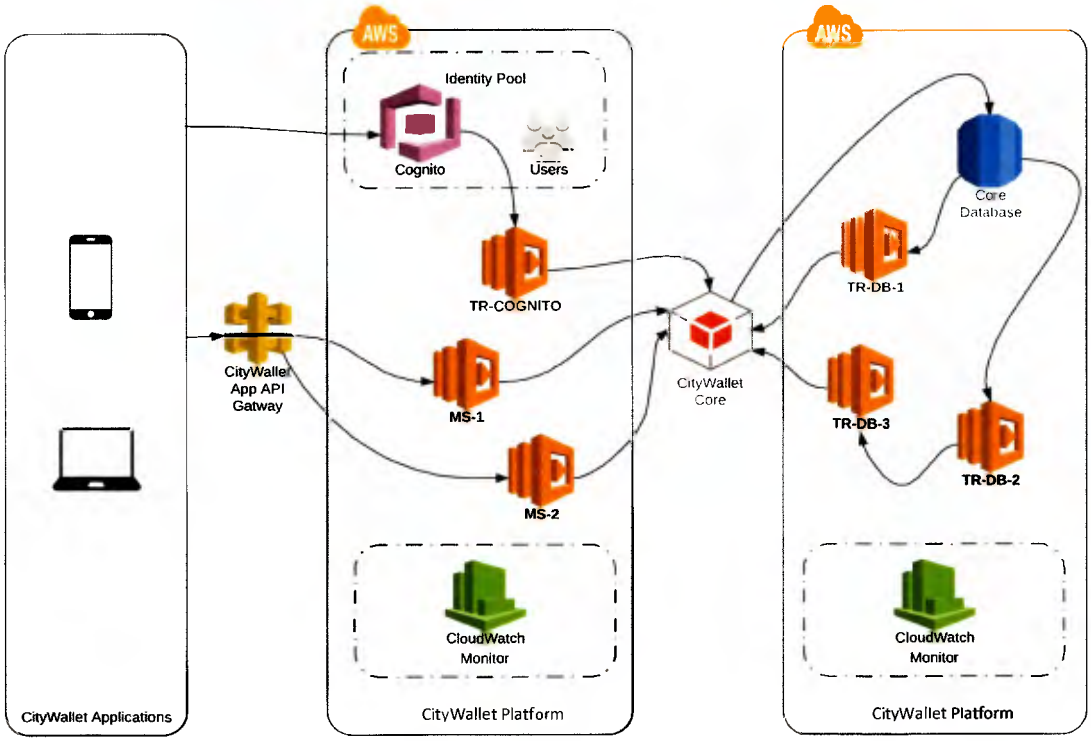


Figura 4. Arquitectura de la solución.

Fuente: [Elaboración propia]

Esta arquitectura expuesta en la **Tabla 4**, se encuentra compuesta por múltiples servicios, dentro de ellos:

- **AWS Cognito**: se encarga de almacenar los usuarios registrados y garantizarle el uso de las distintas funcionalidades que el sistema ofrece.

Este ente valida a los usuarios de la plataforma a través de un *pool* de usuarios[15] previamente configurado.

- **Amazon API Gateway:** este servicio soportó cada uno de los recursos y métodos desarrollados en cada objetivo, para recibir peticiones de los aplicativos cliente, estos se listan en el **Anexo A** en la **Tabla 3**.
- **AWS Lambda**: este servicio brinda soporte a los microservicios desarrollados en cada uno de los *sprints*, se muestran en el **Anexo B** en la **Tabla 4**. A continuación se listan las distintas funcionalidades que desempeñan los microservicios:
    - **Servicio web:** estos microservicios se caracterizan por ser invocados por la API de la aplicación de usuarios de CityWallet, están identificados por el estándar **MS** seguido del recurso que lo identifica.
    - **Evento de base de datos:** son microservicios que se disparan a partir de un evento de una tabla en la base de datos, actuando como disparador y se identifican con el estándar **TR-DB** seguido del recurso que lo identifica.
    - **Evento de AWS Cognito:** son disparados según el evento configurado en cognito como creación, modificación, verificación, autenticación, entre otros. Se identifican por el estándar **TR-COGNITO** seguido del recurso que lo identifica.

[15] Pool de usuarios: conjunto de usuarios pertenecientes a una plataforma.

- **Amazon DynamoDB:** las estructuras de almacenamiento que posee este servicio son manejadas por los métodos desarrollados en la librería de CityWallet Core que se muestran en el **Anexo C**.
- **CityWallet Core:** librería que se encuentra integrada a los microservicios para que estos realicen el manejo de base de datos según su lógica de negocio.
- **Amazon CloudWatch:** se encuentra integrado a los microservicios para almacenar las bitácoras generadas en sus ejecuciones.

1.2. Sprint 1

En este *sprint* se realizaron los objetivos referentes al aporte funcional, que en este caso, se rediseñaron los procesos asociados a la gestión de los transaction tools. Así mismo, se realizaron las integraciones respectivas para cumplir con el objetivo de aporte tecnológico. Ambos aportes se mencionan dentro del desarrollo de este *sprint*. Por otro lado, se desarrollaron las distintas actividades involucradas en los hitos pertenecientes a la épica **E1,** expuesta en el *backlog* del *sprint* en el **Anexo D** en la **Tabla 13**. Este desarrollo se enfocó en el cumplimiento de los siguientes hitos: **E1H1**, **E1H2**, **E1H3**, **E1H4**, **E1H5** y **E1H6**, estos cubren con los procesos involucrados en la gestión de transaction tools. Dentro de estas actividades de este *sprint* se desarrollaron los de los microservicios **MS-TT-1**, **MS-TT-2**, **MS-TT-3, TR-DB-TT-2** y la integración al **TR-DB-TT-1**, expuestos en el **Anexo B** en la **Tabla 4**. A su vez se realizó el desarrollo de los métodos en la librería CityWallet Core asociado a cada microservicios,

haciendo uso de las tablas de base de datos expuestas en el **Anexo C** en la **Tabla 5**, **8** y **9**. También se desarrolló del recurso **API-TT** de la API de la aplicación de usuarios que se muestran en el **Anexo A** en la **Tabla 3**.

Actualmente el transaction tool disponible para el uso en el sistema de CityWallet es de tipo calcomanía, está compuesto por un código PIN[16] y un código PAK[17], estos códigos se complementan entre sí identificando el transaction tool en el sistema. El proceso actual de asociación de un transaction tool a una cuenta de usuario pertenece a los hitos **E1H1** y **E1H2**, y se muestra reflejado en el rediseño resultante de este proceso que se muestra en el **Anexo E** en la **Figura 20**.

Una vez se encuentren registradas calcomanías válidas en el sistema, el usuario de estacionamiento podrá realizar la asociación de una de ellas, para esto necesitará ingresar el PIN y el PAK en la sección de "Mi calcomanía" en la aplicación de CityWallet. Al realizar exitosamente la asociación de la calcomanía, se registrará en la tabla **TBD-TT-1** los códigos del transaction tool con el número de identificación único de la cuenta del usuario y un estatus de calcomanía activa, junto a esto se actualizará la tabla del inventario interno de CityWallet, modificando este elemento asociado a un estatus de tipo reclamado.

Dentro de los microservicios involucrados a los hitos **E1H1** y **E1H2**, se desarrolló el microservicio **MS-TT-2** asociado a la actividad **E1H1A1,** este es invocado a través de la API desarrollada, para soportar la funcionalidad de registro

[16] PIN: código único alfanumérico de cuatro dígitos perteneciente a la calcomanía CityWallet.
[17] PAK: código numérico de cuatro dígitos perteneciente a la calcomanía CityWallet.

del transaction tool a una cuenta de usuario en la base de datos. Posteriormente se realizó la actividad **E1H1A3**, la integración al microservicio **TR-DB-TT-1**, este se ejecuta cuando se asocia un transaction tool a una cuenta de usuario, es un evento asociado a la tabla de los transaction tools y se encarga de invocar a su vez a un tercer microservicio **TR-DB-TT-2** perteneciente a la actividad **E1H1A3,** que tiene como funcionalidad el cambio de estatus de la calcomanía en el inventario. Así mismo, se desarrollaron los métodos de la librería de CityWallet Core, que se encuentran involucrados en el manejo de la base de datos que cumplieran con la asociación de la calcomanía y se integraron a los microservicios anteriormente desarrollados. Este último desarrollo se basó en las actividades **E1H1A4** y **E1H1A5**.

Para la ejecución del microservicio que asocia una calcomanía, se realizó la actividad **E1H2A1,** donde se desarrolló el recurso **API-TT** de la API de la aplicación de usuarios y el método **API-TT-1**, dentro de este se configuró los parámetros de entrada y salida del microservicio y los errores HTTP que este retornaría según el caso. Las estructuras de los recursos usados como parámetros de entrada y salida a través del API se muestran en el **Anexo A** en la **Figura 5**.

Otro de los procesos cubiertos en este *sprint*, es el de consulta de los transaction tools asociados a una cuenta. Para el cumplimiento de los hitos **E1H3** y **E1H4**, se realizó un rediseño que se muestra en el **Anexo E** en la **Figura 21**.

Para soportar este proceso, se comenzó por la actividad **E1H3A1**, relacionada al desarrollo del microservicio **MS-TT-1**. Este microservicio es el

encargado de obtener la lista los transaction tools que se encuentran asociados a una cuenta de usuario. La funcionalidad dentro del microservicio que se relaciona a la consulta de la lista, fue desarrollada dentro de CityWallet Core e integrada al microservicio para cubrir con este requerimiento. Estas últimas funcionalidades se muestran en las actividades **E1H3A2** y **E1H3A3**.

Para que esta funcionalidad pueda ser consumida por la aplicación de CityWallet, se realizó la actividad **E1H4A1**. La exposición de la API de la aplicación de usuarios para la consulta de los transaction tools, consistió en el desarrollo del recurso **API-TT**, específicamente el método **API-TT-2**, donde se configuró el modelo del recurso que el microservicio retorna al cliente. Esta configuración se muestra en el **Anexo A** en la **Figura 6.**

El último proceso que se cubrió en este *sprint* fue la modificación de un transaction tool, los hitos involucrados en este proceso son **E1H5** y **E1H6**; y se enfocan en el rediseño que se expone en el **Anexo E** en la **Figura 22**.

La actividad **E1H5A1,** fue la primera que se realizó para el hito **E1H5** y se enfocó en el desarrollo del microservicio **MS-TT-3**. El objetivo de este microservicio es la modificación del estado de un transaction tool, este puede pasar de estar de un estatus "Activo" a "Inactivo", o viceversa implicando que el usuario pueda realizar o no, respectivamente, operaciones de entrada y salida en los estacionamientos CityWallet. Luego del desarrollo del microservicio, se procedió a realizar la actividad **E1H5A2** donde se desarrollaron los métodos asociados al cambio de estado de un transaction tool dentro de la librería de

CityWallet Core, posteriormente se integraron estos métodos al microservicio correspondiente en la actividad **E1H5A3.**

Para que la aplicación de CityWallet pueda operar sobre esta funcionalidad, se hizo el desarrollo del método **API-TT-3** sobre el recurso **API-TT** en la API perteneciente a la aplicación de usuarios. Este desarrollo pertenece al hito **E1H6**, actividad **E1H6A1**. El diseño de los modelos de los recursos usados por la API se muestra en el **Anexo A** en la **Figura 7.**

La integración a herramientas de monitoreo a través del agente Cloudwatch, se utiliza en cada uno de los microservicios y en los métodos de core para registrar una bitácora en tiempo de ejecución. Dentro del servicio Amazon API Gateway, el cual soporta las APIs de los procesos, se realizó una integración con Amazon Cognito para, a través de políticas de acceso establecidas, autorizar o denegar la identidad o cuenta de usuario que busca ejecutar las distintas funcionalidades del sistema.

1.3. Sprint 2

Este *sprint* estuvo compuesto por las actividades del hito **E2H1** y **E2H2** perteneciente a la épica **E2,** mostradas en el **Anexo D** en la **Tabla 14.** Para cubrir con las necesidades del hito **E2H1**, se desarrollaron las actividades **E2H1A1**, **E2H1A2** y **E2H1A3**, correspondientes al proceso de recarga de saldo a una cuenta de usuario por medio de la pasarela de pago Instapago. Cada una de las actividades realizadas en este *sprint* corresponde a los microservicios desarrollados **MS-TSN-1**, **MS-TSN-2** y **MS-TSN-3**. Adicionalmente se mencionan

microservicios como **TR-DB-TSN-1**, **TR-DB-TSN-2**, **TR-DB-TSN-3** y **TR-DB-TSN-4**, que se ejecutan en procesos posteriores no comprendidos en el alcance de este trabajo instrumental de grado. Estos microservicios se listan en el **Anexo B** en la **Tabla 4**.

Las tablas de base de datos que son utilizadas por los métodos de la librería CityWallet Core y el recurso **API-TSN** desarrollado de la API de la aplicación de usuario son expuestas en el **Anexo C** en la **Tabla 5, 6** y **7,** y en el **Anexo A** en la **Tabla 3** respectivamente.

El proceso que se cubrió en este *sprint* se centra en la recarga de saldo a través de la pasarela de pago Instapago, este permite al usuario usar su tarjeta de crédito para sumar saldo a su balance. Una vez el usuario realiza esta recarga, el microservicio desarrollado **MS-TSN-2** en la actividad **E2H1A1**, realiza una petición de creación de pago, donde es enviada la información básica de la tarjeta de crédito y es validada por Instapago, retornando una respuesta al microservicio que lo invocó. Posteriormente la información devuelta por el ente externo, es validada por el microservicio y este procede a generar la transacción en la tabla **TBD-TSN-1** con un estatus "Creada" a través de la librería CityWallet Core. Una vez la transacción fue creada en el sistema, el microservicio realiza una segunda petición a la pasarela de pago para que el pago creado sea completado y descontado su cuenta bancaria. En el **Anexo E**, **Figura 23** se muestra el diagrama del proceso cubierto en este *sprint*. En el proceso de creación del pago por tarjeta de crédito y su registro en la tabla **TBD-TSN-1.** Paralelo a su registro, se dispara un evento de

la tabla anterior, asociado al microservicio **TR-DB-TSN-1,** que tiene como objetivo invocar a **TR-DB-TSN-2,** este es un ente interno de CityWallet encargado de realizar validaciones sobre los usuarios a quienes se le registra una transacción y crear un nuevo balance en la tabla **TBD-BLC-1,** este proceso se encuentra resaltado en la figura anterior por el color azul. Posterior al registro de un nuevo balance, se dispara otro evento asociado al microservicio **TR-DB-TSN-3,** que se encarga de recibir los datos de este balance generado e invocar al microservicio **TR-DB-TSN-4.** Este último tiene como objetivo cambiar el estatus de una transacción para indicar que fue debitada, acreditada o errónea, este proceso está señalado por el color verde.

Una vez el microservicio de creación de pago fue desarrollado, se realizaron las actividades **E2H1A2 y E2H1A3,** donde se desarrollaron los métodos de la librería CityWallet Core asociados al registro de una transacción de tipo consumo por recarga, al igual que su integración al microservicio **MS-TSN-2.**

Por último se desarrolló el recurso **API-TSN** de la API junto al método **API-TSN-2** perteneciente a la actividad **E2H2A1** del hito **E2H2,** configurando el modelo de los objetos que actúan como parámetros de entrada y de salida en el proceso cubierto. La estructura de la API se muestra en el **Anexo A** en la **Figura 12.**

Para finalizar, dentro de este *sprint* se realizó el rediseño del proceso de realizar una transacción de recarga de saldo cumpliendo con el aporte funcional, adicionalmente se realizaron los objetivos asociados al aporte tecnológico, donde

en desarrollo de los microservicios, se realizó la integración de Amazon Cloudwatch y tuvo como objetivo registrar archivos de bitácora para obtener una traza una vez se ejecute el microservicio. Así mismo, se realizó la integración de las APIs con Amazon Cognito, este es el ente autorizador de las cuentas de usuario de CityWallet, permitiendo o denegando el acceso a funcionalidades de la plataforma.

1.4. Sprint 3

El *backlog* del *sprint* 3 cubrió con las necesidades del proceso de consulta de transacciones así como su detalle, se trabajó sobre la épica **E2**, cumpliendo con los hitos **E2H3**, **E2H4**, **E2H5** y **E2H6** del **Anexo 9** en la **Tabla 15**.

El proceso que abarca la consulta de la lista de las transacciones está relacionado a las actividades **E2H3A1**, **E2H3A2** y **E2H3A3**, donde el proceso quedó definido en el **Anexo E**, **Figura 24**.

Para el cumplimiento del hito **E2H3**, se desarrolló el microservicio **MS-TSN-1** del **Anexo B** en la **Tabla 4**, que se encarga de consultar las transacciones de consumo por recargas a través su integración a la librería de CityWallet Core con los métodos desarrollados asociados a la consulta de transacciones, que se almacenan en la tabla **TBD-TSN-1** y consultan **TBD-ACC-1** en la tabla expuestas en el **Anexo C** en la **Tabla 5** y **7**.

Para el consumo de este microservicio se desarrolló el método **API-TSN-1** de la API en el recurso **API-TSN,** incluyendo la configuración de sus modelos

dentro de la API expuesta en el **Anexo A** en la **Figura 10** perteneciente al hito **E2H4** y a la actividad **E2H4A1.**

Al igual que se desarrollaron los componentes de la consulta de transacciones por consumo de recargas, se cumplió la consulta del detalle de cada una de estas transacciones. Este segundo proceso cubierto se expone en el **Anexo E, Figura 25.**

Para este proceso se realizaron las actividades **E2H5A1**, **E2H5A2, E2H5A3,** donde se desarrolló el microservicio **MS-TSN-3** que se encarga de la consulta de una transacción almacenada en la tabla de base de datos **TBD-TSN-1.** Esto se soportaba gracias a la integración que se realizó sobre el desarrollo de los métodos del detalle de una transacción que se realizó dentro de la librería de CityWallet Core.

Este *sprint* tuvo su cierre con el cumplimiento de la actividad **E2H6A1** del hito **E2H6,** que consistió en el desarrollo del método **API-TSN-3** dentro del recurso **API-TSN** en la API que consume la aplicación de usuario. Las estructuras que se diseñaron para los modelos dentro de este recurso se muestran en el **Anexo A** en la **Figura 11.**

Por último se cumplió con el aporte tecnológico, donde se realizaron las integraciones necesarias en el desarrollo de los microservicios **MS-TSN-1** y **MS-TSN-3.** Se hizo el manejo del registro de bitácora a través del servicio Amazon Cloudwatch, permitiendo obtener una traza de las distintas operaciones realizadas por el microservicio. Dentro de las APIs desarrolladas, se establecio el grupo de

usuarios que serían autenticados a través de la integración con Amazon Cognito para la posterior ejecución de cada mircroservicio.

Anteriormente, se indicó el rediseño de los procesos existentes en este *sprint*, cubriendo con el aporte funcional de este trabajo instrumental de grado.

1.5. Sprint 4

La épica **E3** tuvo su desarrollo en este *sprint*, se realizaron las actividades **E3H1A1, E3H1A2** y **E3H1A3,** listadas en el **Anexo D** en la **Tabla 10,** asociadas al hito **E3H1** tienen el objetivo de soportar el proceso referente a la creación de un usuario dentro de la plataforma CityWallet. En el **Anexo E**, **Figura 26,** se muestra el diagrama del proceso al crear una cuenta.

Este proceso cuenta Cognito, es un ente que se encarga de gestionar los usuario que creados a través de la aplicación de CityWallet. Cognito además de almacenar los usuarios creados, maneja los procesos asociados a recuperación y cambio de contraseña, inicio y cierre de sesión y verificación de creación de cuenta; estos procesos son soportados gracias a la integración que se realizó a través del SDK público de AWS. Para el alcance de este trabajo instrumental de grado se hizo la integración de la funcionalidad en específico de cambio de contraseña directamente en la aplicación.

El proceso comienza cuando se dispara un evento de Cognito asociado al microservicio **TR-CGT-ACC-1,** mostrado en el **Anexo B** en la **Tabla 4,** este se desarrolló en función al evento que dispara cuando el usuario es creado y verificado en Cognito. Una vez es disparado este microservicio, este procede a

validar y a registrar la cuenta creada, en la tabla **TBD-ACC-1** con los datos básicos de Cognito, expuestos en el **Anexo C** en la **Tabla 5.**

Para el desarrollo de este microservicio se tomó en cuenta el modelo del objeto que envía posteriormente Cognito en el evento de post-confirmación; se realizó también el desarrollo y la integración de los métodos de la librería CityWallet Core para el registro de la cuenta. Por último se integró Amazon Cloudwatch con el objetivo de registrar en archivos de bitácora la ejecución del microservicio desarrollado.

Los aportes tecnológicos y funcionales, se realizaron a través de las integraciones a Amazon Cloudwatch y Amazon Cognito, junto al rediseño del proceso de agregar una cuenta de tipo usuario, respectivamente.

1.6. Sprint 5

Este *sprint* tuvo como objetivo complementar los procesos sobre la gestión y reportes de usuarios realizados en el *sprint* pasado, por lo tanto se realizaron las actividades **E4H1A1, E4H1A2** y **E4H1A3** del hito **H1,** listados en el **Anexo D** en la **Tabla 17.** Adicionalmente se realizaron los objetivos de aporte funcional y tecnológico, relacionados al rediseño de los procesos de consulta de cuenta y balance, y la integración a servicios de monitoreo y políticas de acceso respectivamente. Estos objetivos se explican en el desarrollo del *sprint.*

El proceso que soporta las operaciones de reportes de una cuenta CityWallet, se encuentra compuesto por los microservicios **MS-ACC-1 y MS-ACC-**

2 detallados en el **Anexo B** en la **Tabla 4.** El primer microservicio se encarga de obtener los datos básicos de la cuenta y se basa en el diagrama del **Anexo E**, **Figura 27**.

El proceso de ia consulta del usuario CityWallet resultó del desarrollo del microservicio **MS-ACC-1**, este se encarga de manejar la invocación realizada desde la API para consultar la información del usuario a través de los métodos de la librería de CityWallet core que cubren con este requerimientos. Los métodos pertenecientes a core, se integraron en la actividad **E4H1A3,** estos realizan operaciones sobre la tabla **TBD-ACC-1** y sus atributos son especificados en el **Anexo C** en la **Tabla 5.**

Para los reportes de los usuarios, también se desarrolló el microservicio **MS-ACC-2** donde se consulta el balance más actual del usuario según sus transacciones realizadas con estatus "Procesada", esta consulta se realiza a través de los métodos de la librería core a los cuales se integró este lambda en la actividad **E4H1A3,** utilizando la tabla **TBD-BLC-1** detallada en el **Anexo C**, **Tabla 6** y consultando el campo "CurrentBalance" detallado en la estructura de esta tabla.

El desarrollo de este segundo microservicio se soportó por el siguiente diagrama detallado en el **Anexo E**, **Figura 28.**

Para la culminación de este *sprint* se cumplió con el hito **H2** de la épica **E4,** este hito comprende las actividades **E4H2A1** y **E4H2A2** donde se realizó el desarrollo de los distintos recursos de la API listados en el **Anexo A** en la **Tabla 3,**

específicamente el recurso **API-ACC** con el método **API-ACC-1** para la cuenta de usuario y el recurso **API-BLC** con el método **API-BLC-1** para su balance correspondiente. Los recursos de los métodos desarrollados pertenecientes a la API de la aplicación de usuarios son detallados en el **Anexo A** en la **Figura 8** y en el **Anexo A** en la **Figura 9** respectivamente. Así mismo, se realizó la integración de Amazon Cognito a través del servicio API Gateway encargado de gestionar las APIs; Cognito se encarga de verificar y autorizar los usuarios que hacen uso de estos APIs, permitiendo o no el uso de la funcionalidad en la plataforma.

Amazon Cloudwatch se integró a los microservicios desarrollados, garantizando el registro de bitácoras en archivos para el monitoreo de su comportamiento

1.7. Sprint 6

Este *sprint* tuvo como objetivo el desarrollo del proceso de agregar de un carro de compra en la tienda de productos, para esto se cumplió con los hitos **H1** y **H2** de la épica **E5** referente al objetivo de la gestión de la tienda de productos; se realizaron las actividades **E5H1A1, E5H1A2, E5H1A3** y **E5H2A1** del *backlog* del *sprint* ubicado en el **Anexo D** en la **Tabla 18.**

Anteriormente la funcionalidad de agregar un carro no se encontraba soportada dentro de los procesos de CityWallet. Para esto se realizó el diseño del proceso expuesto en el **Anexo E** en la **Figura 29**, basándose en los requerimientos de la tienda de productos y cumpliendo con el aporte funcional.

La creación de un carro de compra parte de los productos que se encuentran disponibles en la tienda. Al agregar el primer producto al carro de compra se crea una orden en la base de datos, una vez se agregue un segundo producto, la orden se modifica y agrega este último a la lista de productos de dicha orden. Gracias a como se encuentra soportada esta funcionalidad, la creación de una orden o carro de compra se limita a que el usuario podrá tener una sola orden creada, una vez esa orden sea cancelada o pagada, el usuario podrá optar a crear una nueva.

La primera actividad que se realizó fue **E5H1A1,** en ella se desarrolló el microservicio **MS-ORD-2**, detallado en el **Anexo B** en la **Tabla 4**, encargado de crear la orden a través de la integración de la librería CityWallet Core realizada en la actividad **E5H1A3**. Antes de esta integración se realizó el desarrollo de los métodos core en la actividad **E5H1A2,** estos métodos se encargan de realizar distintas operaciones, desde la validación enviada del microservicio hasta la consulta del usuario a quien se le está creando la orden. Dentro del desarrollo del microservicio se integró Amazon Cloudwatch que se encarga de realizar trazas información dentro de los archivos de bitácora.

Al crearse la orden en la tabla **TDB-ORD-1** de base de datos reflejada en el **Anexo C** en la **Tabla 11**, esta se inserta estatus de creada y sus datos básicos detallados en el **Anexo C** en la **Tabla 11.**

Por último se realizó la actividad **E5H2A1,** en ella se desarrolló el recurso **API-ORD** de la API de la aplicación de usuarios y el método **API-ORD-2** listado en

el **Anexo A** en la **Tabla 3**, permitiendo al microservicio ser invocado desde los distintos clientes. También dentro de este desarrollo se cumplió con los objetivos de aporte tecnológico, donde realizó la integración de API Gateway con Amazon Cognito, el cual tiene como objetivo validar y autorizar la cuenta de usuario que está realizando la petición a este API. En el **Anexo A** en la **Figura 13** se muestran los modelos que se usan en la petición o respuesta de la API.

1.8. Sprint 7

En el *sprint* pasado se realizaron los primeros hitos pertenecientes a la épica **E5,** en este se cubrieron los procesos de reporte para el carro de compra y realizando las actividades de los hitos **H3, H4, H5** y **H6,** expuestos en el **Anexo D** en la **Tabla 19.** Previo a la realización de los hitos **H3** y **H4,** se realizó el diseño de los dos procesos que soportan los reportes de las órdenes, en el **Anexo E** en la **Figura 30,** se muestra el diagrama asociados a la consulta de las órdenes

Para este proceso, se realizaron las actividades **E5H3A1, E5H3A2** y **E5H3A3** donde se desarrolló el microservicio **MS-ORD-1** expuesto en el **Anexo B** en la **Tabla 4,** que tiene como objetivo retornar la lista de las órdenes asociadas a un usuario. Este microservicio se hace uso de los métodos de consulta de órdenes de CityWallet Core, estos métodos manejas distintas operaciones para validar la cuenta de usuario existente junto con la obtención sus órdenes asociadas. La librería de core hace uso de la tabla **TBD-ORD-1** listada en el **Anexo C** en la **Tabla 11,** detallando la estructura almacenada de una orden.

Como última actividad se realizó **E5H4A1**, donde se desarrolla el método **API-ORD-1** del recurso **API-ORD** de la API, especificando en el **Anexo A** en la **Figura 14** la estructura de los valores de retorno del microservicio.

Al igual que la consulta de la lista de órdenes, se realizó el diagrama del proceso de consulta del detalle de una orden, en el **Anexo E, Figura 31** se muestra gráficamente el diseño resultante.

Al igual que en el hito **H3,** se realizaron las actividades **E5H5A1, E5H5A2** y **E5H5A3** del hito **H5,** en la primera actividad se realizó el desarrollo del microservicio **MS-ORD-4,** encargado de retornar el detalle de alguna orden asociada a la cuenta de usuario. A través de las actividades **E5H5A2** y **E5H5A3,** se realizó el desarrollo de los métodos para el detalle de una orden y su integración al microservicio respectivamente. Así mismo, se realizó la actividad **E5H6A1** del hito **H6** en donde se desarrolló el método **API-ORD-3** de la API para la aplicación de usuario y los modelos pertenecientes al recurso de órdenes mostrados en el **Anexo A** en la **Figura 15**.

Cumpliendo con los aportes tecnológicos, se realizaron integraciones dentro del conjunto de microservicios desarrollados, donde se integró Amazon Cloudwatch, un ente de monitoreo que se utilizó principalmente para el registro de bitácoras según la ejecución del microservicio. Adicionalmente, dentro del desarrollo de las APIs se integró con Amazon Cognito que se encarga validar la sesión activa del usuario y permitir el acceso a las funcionalidades de la plataforma.

Al igual que se menciona los aportes tecnológicos realizados en este *sprint*, cada uno de los diseños de procesos no existentes cubre con el aporte funcional según los objetivos propuestos.

1.9. Sprint 8

En este *sprint* se realizaron los hitos **H7, H8, H9, H10, H11** y **H12** asociados a la épica **E5.** Los hitos **H7** y **H8** abarcan la funcionalidad de modificación de carro de compra, para estos hitos se realizaron las siguientes actividades **E5H7A1, E5H7A2, E5H7A3** y **E5H8A1**. Estas actividades se listaron en el **Anexo D** en la **Tabla 20** del *backlog* del *sprint*.

Para el desarrollo de cada una de las actividades se realizó el diseño de la funcionalidad anteriormente mencionada que se muestra en el **Anexo E** en la **Figura 32.**

Este proceso soporta la operación de modificación de carro de compra, para esto el usuario debió haber asociado previamente productos a un carro de compra, creando la orden en base de datos. Una vez la orden está creada, el usuario será capaz de modificar las cantidades de los productos de la orden o remover algún producto. Otra de las funcionalidades soportadas por este proceso es la desactivación o cancelación de una orden, para esto la orden debe tener un estatus creada.

El conjunto de actividades **E5H7A1, E5H7A2** y **E5H7A3** se enfocó en el desarrollo del microservicio **MS-ORD-3** detallado en el **Anexo B** en la **Tabla 4** junto al desarrollo e integración de los métodos de la librería CityWallet Core.

El microservicio desarrollado se encarga de una serie de validaciones con respecto al usuario que tiene asociada esa orden a modificar, obtener los datos de la orden existente a modificar e integrarse a los métodos desarrollados de la librería. Los métodos soportados por la librería realizan operaciones de consulta al inventario con las cantidades más actuales de los productos almacenados en la tabla **TBD-STK-1**, mostrada en el **Anexo C** en la **Tabla 10** CityWallet Core también realiza la operación de modificación de la orden en la tabla **TBD-ORD-1** del **Anexo C** en la **Tabla 11,** según los cambios realizados por el usuario.

En el último hito se realizó la actividad **E5H8A1,** donde se realizó el desarrollo del método **API-ORD-4** expuesto en el **Anexo A** en la **Tabla 3,** configurando la asociación del microservicio a la API junto a los modelos de petición y de retorno al cliente, detallados en el **Anexo A** en la **Figura 16.**

Los últimos hitos sobre los que se desarrollaron en este *sprint* fueron: **H9, H10, H11** y **H12.** Estos hitos soportan los procesos de reportes de los productos y detalle de un producto para la tienda. En el **Anexo E** en la **Figura 33,** y en la **Figura 34** se muestran los diseños de los dos procesos respectivamente, asociados a las actividades realizadas.

Al definirse el proceso de la **Figura 33** se realizaron las actividades **E5H9A1, E5H9A2** y **E5H9A3,** donde se desarrolló el microservicio **MS-STK-1** que se encarga de validar la data entrante que se envía a través de la API de la aplicación, al recurso **API-PRD** utilizando el método **API-PRD-1**. Cumpliendo con la actividad **E5H9A1,** se realizó el desarrollo de los métodos en la librería core

asociados a la actividad **E5H9A2,** estos están encargados de consultar la lista de los productos disponibles almacenados en la tabla **TBD-STK-1**. Posteriormente en la actividad **E5H9A3,** se realizó la integración de CityWallet Core al microservicio para que este pueda cumplir con las funcionalidades requeridas en el proceso. Adicionalmente se cumplió con la actividad **E5H10A1** del hito **H10,** donde se desarrolló el recurso de la API anteriormente mencionada, en el **Anexo A** en la **Figura 18** se muestra en detalle la estructura del recurso desarrollado.

De igual forma se cumplió con las actividades del hito **E11** y **E12,** referente al desarrollo del proceso de detalle de un producto expuesto en la **Figura 34.** Primero se realizó el grupo de actividades **E5H11A1, E5H11A2** y **E5H11A3** relacionadas al microservicios, donde se realizó su desarrollo basado en cómo es invocado por la método del recurso **API-PRD-2,** realizando validaciones según la data entrante. Adicionalmente se desarrollaron los métodos de la librería para consultar el detalle de un producto en específico y se integraron al microservicio.

Para garantizar el funcionamiento de entre el recurso **API-PRD** de la API y el microservicio, se desarrolló la actividad **E5H12A1** que se enfocó en la estructuración del recurso **API-PRD-2,** mostrado en el **Anexo A** en la **Figura 19.** Ambos procesos desarrollados permiten al usuario consultar la información de los productos disponibles con sus cantidades más actualizadas, para así contar con la posibilidad de crear un carro de compra en la tienda de productos.

Todos los microservicios desarrollados se integraron al registro de archivos de bitácora de Amazon Cloudwatch; también cada API desarrollado se integró con

un agente autorizador de Amazon Cognito, que gestiona las sesiones de los usuarios, permitiendo el acceso a los recursos en los APIs para la ejecución de las funcionalidades. Estas integraciones a los servicios de Amazon son pertenecientes a al aporte tecnológico del trabajo instrumental de grado, al igual que los diseños de procesos realizados anteriormente.

1.10. Sprint 9

Este último *sprint* tuvo como objetivo cumplir con los hitos **H13** y **H14** de la épica **E5**, como se muestra en el **Anexo D** en la **Tabla 21** Al cumplir con los hitos mencionados anteriormente se cubrió con el proceso de pago en la tienda de productos. El diseño de este proceso no existente se expone en el **Anexo E** en la **Figura 35.**

Este proceso comienza a partir de la creación de una orden o carro de compra; una vez el usuario eligió los productos que componen la orden, este tiene la posibilidad de realizar el pago con tarjeta de crédito a través de la pasarela Instapago.

La primera actividad que se realizó fue **E5H13A1** donde se desarrolló el microservicio **MS-ORD-5** listado en el **Anexo B** en la **Tabla 4,** este se integra a Instapago a través de su API pública y realiza una primera petición enviando los datos básicos del usuario junto a su tarjeta de crédito. La pasarela de pago al recibir y validar los datos necesarios para generar el pago, retorna al microservicio una respuesta con información de que el pago fue aprobado. Al obtener esta información retornada al microservicio, se realiza el cambio de estatus de la orden

de creada a un estatus “Procesando” en la tabla **TBD-ORD-1** a través de la integración con CityWallet Core, la estructura de la tabla de órdenes se detalla en el **Anexo C** en la **Tabla 11**

Como se observa en el diagrama del proceso, una vez se realice una modificación en el estatus de la orden en base de datos se dispara un evento asociado al microservicio **TR-DB-ORD-1** desarrollado en la actividad **E5H13A2.** El objetivo de este microservicio es recibir el evento ejecutado y recibir la orden que cambió de estatus, dentro de las validaciones que se realizan dentro de este subproceso se consulta al inventario almacenado en la tabla **TBD-STK-1** mostrada en el **Anexo C** en la **Tabla 10,** con el fin de obtener las unidades disponibles, en ese momento, de cada producto perteneciente a dicha orden. Basado en el retorno del método que consulta el inventario el microservicio procede a invocar dos microservicios, si el inventario posee cantidades disponibles para cada producto de la orden se invoca al microservicio **TR-DB-ORD-2**, de forma contraria se invoca a **TR-DB-ORD-3.**

La actividad **E5H13A3** consistió en el desarrollo de los microservicios que se invocan una vez se consulte el inventario disponible, el primer microservicio **TR-DB-ORD-2,** realiza la modificación del inventario según las cantidades de los productos demandados en la orden a pagar, la funcionalidad de este microservicio se encuentra resaltada de color amarillo en el diagrama del proceso. Al realizar el descuento exitoso de las cantidades respectivas, se dispara un evento asociado a la tabla **TBD-STK-1** del inventario, donde se da inicio a otro subproceso asociado

a la actividad **E5H13A5.** El segundo microservicio que se desarrolló fue **TR-DB-ORD-3** asociado a la actividad **E5H13A4,** el flujo de la manera en que este interactúa entre los distintos componentes se encuentra diagramada en color rojo y se encarga de cambiar el estatus de la orden de procesando a "Error" en caso de que no exista inventario disponible al realizar la compra de las orden.

Para soportar el subproceso que comienza una vez que se descontaron las cantidades en el inventario, se realizó la actividad **E5H13A5** donde se desarrolló el microservicio **TR-DB-STK-1** responsable de obtener y validar la información que se generó del evento de modificación del inventario en la tabla **TBD-STK-1,** en él se recibe la orden que modificó el inventario, la cuenta a quién pertenece esa orden y el monto total de ese carro de compra. Posteriormente el microservicio invoca a los microservicios **TR-DB-STK-1** y **TR-DB-ORD-3** de forma paralela y asíncrona. El microservicio **TR-DB-STK-1** desarrolló en la actividad **E5H13A6,** este a través de una petición al API de Instapago se realiza la confirmación del pago por tarjeta de crédito previamente aprobado en el microservicio **MS-ORD-5.** Una vez el pago se haya realizado por completo, se procede a generar una nueva transacción de tipo pago de tienda en la tabla **TBD-TSN-1** del **Anexo C** en la **Tabla 7,** perteneciente a las transacciones de CityWallet, dando la posibilidad al usuario de consultar su balance y ver reflejada esta compra. Este subproceso para completar el pago se muestra en color verde dentro del diagrama. Así mismo, el microservicio **TR-DB-ORD-3** se ejecuta paralelamente como se indicó

anteriormente y este tiene como objetivo cambiar de estatus procesando a procesada culminando con el proceso de pago de un carro de compra.

Para hacer posible el manejo de la base de datos dentro de los microservicios desarrollados, en la actividad **E5H13A7** se realizó el desarrollo de los métodos pertenecientes a las operaciones para registrar un pago a un carro de compra dentro de CityWallet Core, al contar con estos métodos se realizó la integración de esta librería a los microservicios respectivos en la actividad **E5H13A8.**

Por último se cumplió con la actividad **E5H14A1,** donde se desarrolló el método **API-ORD-5** del recurso **API-ORD**. Este desarrollo le permite a los clientes a través de la API de la aplicación de usuarios invocar la funcionalidad del pago de un carro de compra. La integración de Amazon Cognito dentro del API desarrollado, permite verificar la validez de la sesión en la plataforma que posee la cuenta de usuario antes de la ejecución del microservicio. En el **Anexo A** en la **Figura 17** se detalla la estructura soportada que tiene esta petición basándose en REST.

Dentro del desarrollo de los microservicios, se cumplieron los objetivos respectivos al aporte tecnológico, realizando la integración a Amazon Cloudwatch que permite registrar archivos de bitácoras, dejando evidencia del comportamiento de la ejecución del microservicios. Por último, se realizaron los objetivos de aporte funcional expuestos en los rediseños realizados anteriormente.

CAPÍTULO V

1. Resultados

Al haber finalizado con la fase de implementación de cada componente del sistema, se presentan los resultados obtenidos como consecuencia del cumplimiento de cada objetivo expuesto en este trabajo instrumental de grado:

Desarrollar las APIs como una capa independiente, con funcionalidades específicas de comunicación y control de acceso para los microservicios asociados a la aplicación de usuarios de estacionamientos CityWallet.

A través del desarrollo de las distintas APIs dentro del servicio Amazon API Gateway basado en REST, se obtuvo como resultado la capa intermedia o middleware que permite la integración a la plataforma de CityWallet, este desarrollo se hizo progresivamente en los *sprints* 1, 2, 3, 5, 6, 7, 8 y 9.

Integrar los microservicios a desarrollar con CityWallet Core para sustentar sus necesidades de la aplicación de usuarios de estacionamientos CityWallet.

Gracias a la integración de CityWallet Core que se realizó durante la implementación de los microservicios en los *sprints* 1, 2, 3, 4, 5, 6, 7, 8 y 9, se obtuvo el acceso a las distintas implementaciones dentro de esta librería, que permite operar y almacenar estructuras en una base de datos, trayendo como

beneficio la integridad de la información del usuario para su correcto manejo dentro de las aplicaciones cliente.

Desarrollar los microservicios para la gestión y reportes de los transaction tools asociados a cuentas CityWallet.

Una vez se realizó el desarrollo de los microservicios para la gestión y reportes de los transaction tools en el *sprint* 1, se obtuvo como resultado el manejo por parte del usuario de las distintas funcionalidades mostradas en las aplicaciones clientes, el usuario tiene la posibilidad de asociar una calcomanía CityWallet o transaction tool a su cuenta para operar dentro de los estacionamientos y consultar o desactivar en caso de pérdida sus transaction tools.

Desarrollar los microservicios para la gestión y reportes de cuentas CityWallet.

El desarrollo de la gestión y reportes de cuentas CityWallet en los *sprints* 4 y 5, tuvo como resultado los distintos microservicios que soportan las funcionalidades de creación de cuenta, consulta de cuenta y balance por parte de los usuarios.

Desarrollar los microservicios para reportes de transacciones realizadas en los estacionamientos.

Los microservicios desarrollados en este objetivo durante el *sprint* 3, cubrieron con el proceso de consulta de transacciones realizadas en los

estacionamientos CityWallet, brindándole al usuario obtener información sobre sus consumos al entrar o salir de un establecimiento.

Desarrollar los microservicios para la gestión y reportes de transacciones de recarga de saldo a cuentas CityWallet.

Los microservicios desarrollados en los *sprints* 2 y 3, y relacionados a la gestión y reportes de transacciones de recarga de saldo, traen como resultado, los procesos donde el usuario tiene la posibilidad de realizar una recarga por tarjeta de crédito; también este puede consultar los movimientos dentro de la plataforma CityWallet asociados a la recarga de saldo.

Desarrollar los microservicios para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet.

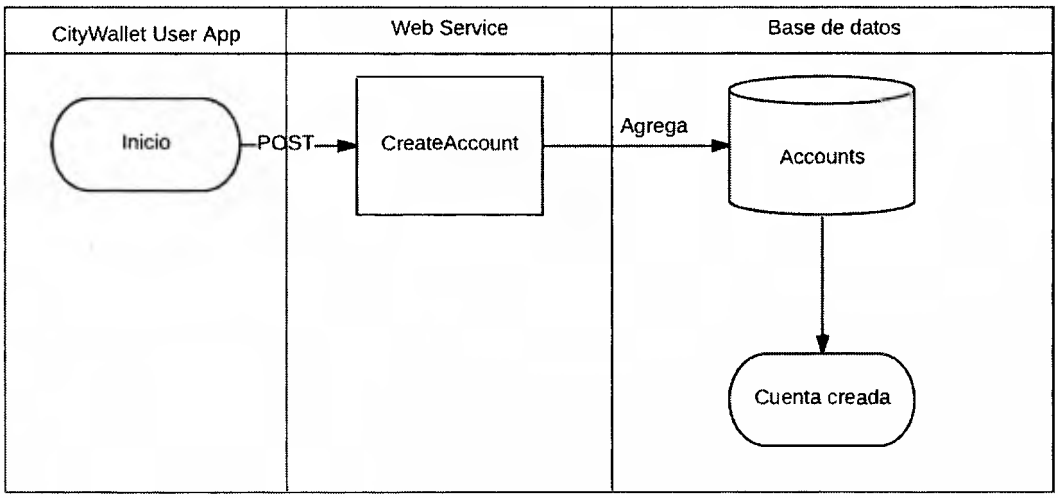
Como resultado del desarrollo de los microservicios que soportan la gestión de la tienda de productos realizados en los *sprint*s 6, 7, 8 y 9, se obtuvo el soporte de las funcionalidades de los procesos asociados a la creación, modificación y consulta de un carro de compra, la eliminación o desactivación de un carro de compra y por último la compra de un carro de compra por tarjeta de crédito. Una vez el usuario haya realizado la compra, podrá consultar el cobro de ese monto reflejado gracias a las funcionalidades obtenidas en el objetivo anterior.

Rediseñar los procesos existentes en el backend actual de la aplicación de usuarios de estacionamientos CityWallet.

Anteriormente, CityWallet contaba con procesos definidos que soportaban las funcionalidades en una arquitectura monolítica, gran parte del objetivo de este

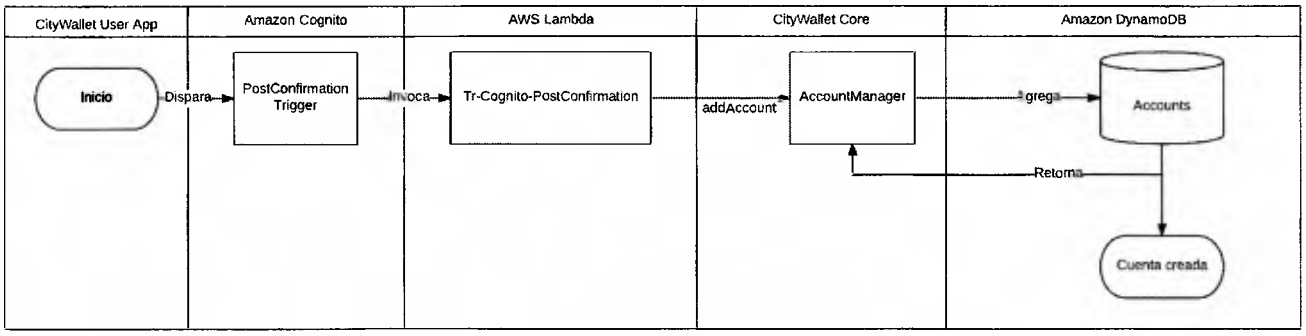
trabajo instrumental de grado fue el rediseño de los procesos pasados debido a la necesidad de desarrollar la plataforma de la empresa sobre una arquitectura basada microservicios dentro de una plataforma en la nube. Esto trajo como consecuencia la creación y adaptación de la lógica de negocio existente sobre una plataforma en la nube.

El rediseño de la gestión de cuentas CityWallet se basó en el proceso anterior de la plataforma monolítica y así se obtuvo como resultado el actual diseño dentro de la plataforma de microservicios, ambos diagramas se muestran a continuación:



Anexo E. Figura 39. Diseño de creación de cuenta.

Fuente: CityWallet



Anexo E. Figura 26. Rediseño de agregar una cuenta de usuario.

Fuente: [Elaboración propia]

Los procesos rediseñados se realizaron en los *sprints* 1, 2, 3, 4, y 5, y se muestran en el **Anexo E**, las **Figuras 20 – 35**. Al igual que los diseños de los procesos anteriores en el **Anexo E**, las **Figuras 36 – 43**.

Diseñar los procesos para la tienda de productos de la aplicación de usuarios de estacionamientos CityWallet.

La empresa no contaba con procesos existentes relacionado a la gestión de una tienda de productos, gracias a este diseño se obtuvo la definición de los procesos referentes a la consulta de productos disponibles, creación, modificación y desactivación de una orden o carro de compra y la compra a través de tarjeta de crédito. A través de los procesos diseñados en los *sprints* 6, 7, 8 y 9, el usuario podrá hacer uso de estas funcionalidades y ver reflejado sus movimientos en la aplicación cliente.

Integrar los microservicios a desarrollar con los sistemas de control y gestión de usuarios existentes en CityWallet para mantener las políticas de acceso.

Gracias a la integración de los microservicios a los sistemas de control de usuarios según políticas de acceso, el sistema cuenta con una capa de seguridad dentro del servicio Amazon API Gateway, permitiendo a los usuarios con sesión válida manejada por Amazon Cognito, hacer uso de las funcionalidades del sistema.

Integrar los microservicios a desarrollar con herramientas de monitoreo en tiempo real para proporcionar información de acceso y de su ejecución.

Al igual que se integró un servicio para controlar a los usuarios que ingresan al sistema, se realizó una integración con el servicio de Amazon Cloudwatch, trayendo como resultado una capa de monitoreo que genera archivos donde se evidencien las trazas durante las operaciones de un microservicio; esta integración se realizó durante su fase de implementación en los *sprints* 1, 2, 3, 4, 5, 6, 7, 8 y 9.

Evaluar plataformas de computación en la nube que mejor se adapten al diseño y construcción de sistemas bajo una arquitectura de microservicios.

A pesar de que la empresa poseía una decisión tomada sobre la plataforma en la nube a utilizar, con esta evaluación se obtuvo como resultado una

documentación donde se establecen varios criterios que definen las distintas alternativas dentro del paradigma serverless. Esta evaluación se realizó en el *sprint* 0.

2. Conclusiones

El objetivo principal de este trabajo instrumental de grado fue el desarrollo de una segunda versión de la plataforma de CityWallet, basándose en el paradigma serverless. A través de este desarrollo se logró evaluar y definir la arquitectura en la nube que soporta los distintos componentes responsables de soportar la lógica de negocio de la empresa.

A través del marco de trabajo Scrum modificado, se logró realizar el desarrollo de las funcionalidades a través de la organización en épicas e hitos basada en las necesidades de los procesos existentes y no existentes.

Con los procesos existentes, se realizó un rediseño asociado a las distintas funcionalidades dentro de la gestión de usuarios, gestión de transaction tools y gestión de transacciones; así mismo, se realizó el diseño de la gestión de tienda de productos. Con esto, se buscó mejorar las fallas del proceso, garantizar su funcionamiento óptimo y mejorar la calidad del servicio entregado al cliente.

Gracias a que la plataforma de CityWallet se soportó sobre microservicios, trae consigo beneficios como el desacoplamiento de sus funcionalidades con lo que se espera tener una alta mantenibilidad y escalabilidad. Debido a que existe un alto nivel de modularidad entre sus componentes, cada uno de estos es responsable

de una funcionalidad en específico, trayendo como consecuencia un posible sistema tolerante a fallos. Cada una de las bondades de la nueva plataforma no solo afecta el rendimiento del sistema en sí, sino también la manera que el equipo de TI maneja sobre este, esperando tiempos menores para resolución de problemas o expansión de las funcionalidades.

Por último, el objetivo principal de una empresa u organización es entregar valor a sus clientes, esto es posible a través del rendimiento positivo de sus procesos. Para esto se realizó el desarrollo de la plataforma bajo una arquitectura de microservicios, a través de un marco de trabajo asociado al desarrollo ágil, con el objetivo de soportar la lógica del negocio que sea capaz adaptarse a las necesidades y de esta manera poder satisfacer al cliente, en este caso, con las funcionalidades asociadas a la aplicación de usuarios de estacionamiento CityWallet.

3. Recomendaciones

Basándose en el desarrollo de la nueva plataforma de CityWallet, es importante resaltar que el lenguaje y la estructura utilizada en la implementación de cada microservicio estuvo soportada por Java bajo la creación de un proyecto Maven[18].

Este fenómeno se define como inicio en frío [47] y se encuentra expuesto en distintos foros de discusión, donde se argumenta que el servicio de AWS

[18] Maven: herramienta para la construcción de proyectos, donde se configura la compilación y creación de un jar. Esta generación parte de un archivo denominado POM, el cual contiene dependencias o referencias que garantizan el funcionamiento interno del proyecto.

Lambda posee una manera particular de operar cuando algún microservicio es invocado.

El primer paso que realiza este servicio es preguntar si existe algún contenedor que pueda manejar la petición realizada al microservicio. Estos contenedores son definidos como ambientes o instancias que poseen una configuración previa referente a los recursos que se consumen en la ejecución del microservicio, como por ejemplo cantidad memoria asignada o el máximo tiempo de ejecución que puede tardar. Al conocer si existe algún contenedor que pueda manejar esa petición, el microservicio se ejecuta; de ser negativa la respuesta, un nuevo contenedor es creado y el microservicio es invocado [48]. Gracias a que el contenedor se encarga de manejar los recursos donde se ejecuta el microservicio, este es el responsable de culminar la ejecución, si el tiempo máximo es cumplido o recibir una respuesta en un tiempo menor al máximo.

Como se puede observar, una de las grandes restricciones que existe en la arquitectura basada en serverless es el manejo del tiempo máximo en el cual se debe ejecutar un microservicio, implicando implementación que pueda satisfacer esta limitación.

Es recomendable tener eventos programados a través del servicio Amazon Cloudwatch que permite el "calentamiento" o preparación de los microservicios, para no comprometer el rendimiento al momento de la ejecución de la funcionalidad. Por otra parte, se recomienda evaluar para próximos desarrollos, el rendimiento que posee cada lenguaje de programación.

Por último, se recomienda realizar la configuración de políticas de alarmas a través de Amazon Cloudwatch, con esto el sistema contaría con mecanismos preventivos o correctivos basados en el comportamiento del tráfico entrante y saliente de CityWallet; este tipo de medidas busca que la plataforma sea lo sufientemente capaz de manejar los problemas que se pudiesen presentar en el registro o proceso de cada una de la transacciones de estacionamiento o recargas. Dentro de los problemas tentativos que se puedan presentar, existe la posibilidad que una transacción no sea modificada a lo largo de sus estados (creada, debitada o acreditada) dentro del proceso de su registro. Con la configuración propuesta, el sistema actuaría como una herramienta de reportes reflejando las transacciones no procesadas y ejecutando procesos de corrección sobre aquellas que presentan esa caractarística.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Nuñez-Torrón, «¿Qué es Fintech? Todas las claves para entenderlo,» Ticbeat España, Julio 2016. [En línea]. Disponible en: http://www.ticbeat.com/innovacion/que-es-fintech-claves-concepto/ [Último acceso: 27 de febrero de 2017].

[2] MediaTrends, «¿Qué es la tecnología NFC y que móviles lo contienen?,» MediaTrends, 2017. [En línea]. Disponible en: https://www.mediatrends.es/a/81355/que-es-la-tecnologia-nfc/ [Último acceso: 27 de febrero de 2017].

[3] Software Engineering Institute, «Community Software Architecture Definitions,» Software Engineering Institute, 2017. [En línea]. Disponible en: http://www.sei.cmu.edu/architecture/start/glossary/community.cfm. [Último acceso: 27 de febrero de 2017].

[4] Microsoft Azure, «¿Por qué usar un enfoque de microservicios para crear aplicaciones?,» Microsoft, 6 de febrero de 2017. [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/service-fabric/service-fabric-overview-microservices. [Último acceso: 27 de febrero de 2017].

[5] J. Lewis y M. Fowler, «Microservices,» 25 de marzo de 2014. [En línea]. Disponible en: https://martinfowler.com/articles/microservices.html#footnote-etymology. [Último acceso: 27 de febrero de 2017].

[6] IBM Developers Center, «What is serverless computing?,» IBM, 2017. [En

línea]. Disponible en: https://developer.ibm.com/openwhisk/what-is-serverless-computing/. [Último acceso: 27 de febrero de 2017].

[7] Amazon Web Services, «¿Qué es la informática en la nube?,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/what-is-cloud-computing/. [Último acceso: 27 de febrero de 2017]

[8] Amazon Web Services, «Tipos de cloud computing,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/types-of-cloud-computing/. [Último acceso: 27 de febrero de 2017]

[9] Amazon Web Services, «Productos de la nube,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/products/. [Último acceso: 27 de febrero de 2017]

[10] Amazon Web Services, «Detalles del producto AWS Lambda,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/lambda/details/. [Último acceso: 27 de febrero de 2017]

[11] Amazon Web Services, «Amazon DynamoBD,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/dynamodb/. [Último acceso: 1 de mayo de 2017].

[12] Amazon Web Services, «Amazon API Gateway,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/api-gateway/. [Último acceso: 27 de febrero de 2017]

[13] Amazon Web Services, «Amazon Cloudwatch,» Amazon, 2017. [En línea].

Disponible en: https://aws.amazon.com/es/cloudwatch/. [Último acceso: 27 de febrero de 2017]

[14] Amazon Web Services, «¿Qué Amazon Cognito?,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/cognito/faqs/. [Último acceso: 27 de febrero de 2017]

[15] Amazon Web Services, «Amazon Cognito,» Amazon, 2017. [En línea]. Disponible en: https://aws.amazon.com/es/cognito/. [Último acceso: 27 de febrero de 2017]

[16] Institute of Electrical and Electronics Engineers. (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.IEEE Std. 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, Los Alamitos, CA.

[17] L. De Seta, «Cohesión y Acoplamiento,» Dos Ideas, 8 de abril de 2009. [En línea]. Disponible en: https://dosideas.com/noticias/java/502-acoplamiento-y-cohesion. [Último acceso: 28 de febrero de 2017].

[18] Cunningham & Cunningham «Couping and Cohesion,» Cunningham & Cunningham, 2017. [En línea]. Disponible en: http://wiki.c2.com/?CouplingAndCohesion. [Último acceso: 28 de febrero de 2017].

[19] Club de Investigación Tecnológica «Sistemas escalables,» Club de Investigación Tecnológica, 2009. [En línea]. Disponible en:

http://www.clubdeinvestigacion.com/usuarios/articles/sistemas-escalables.html. [Último acceso: 28 de febrero de 2017].

[20] B. Proffitt, «What APIs Are And Why They're Important,» Readwrite, 19 Septiembre 2013. [En línea]. Disponivle en: http://readwrite.com/2013/09/19/api-defined/. [Último acceso: 19 Noviembre 2016].

[21] BBVA API Market, «API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos,» BBVA, 23 de marzo de 2016. [En línea]. Disponible en: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos. [Último acceso: 28 de febrero de 2017].

[22] A. Gómez, « Pasarelas de pago: todo lo que debe saber un CEO y como contratarlas.,» Reporte Digital, 12 de febrero de 2015. [En línea]. Disponible en: http://reportedigital.com/transformacion-digital/pasarelas-pago-todo-debe-saber-ceo-como-contratarlas/. [Último acceso: 28 de febrero de 2017].

[23] Fast Company, «¿How Zappos Uses One-Week Work Sprints to Laungh Big Projects Fast?», Fast Company & Inc, 2017 [En línea]. Available: https://www.fastcompany.com/3032947/how-zappos-uses-one-week-work-sprints-to-launch-big-projects-fast . [Último acceso: 10 de Abril de 2017].

[24] Tech Crunch, «Here´s How Spotify Scales Up And Stays Agile: It Runs "Squads" Like Lean Startups», AOL Inc. 2013-2017, [En línea]. Available: https://techcrunch.com/2012/11/17/heres-how-spotify-scales-up-and-stays-agile-it-

runs-squads-like-lean-startups/[Último acceso: 15 de Abril de 2017].

[25] Scrum Guides, «Scrum Guide», Scrum.Org & ScrumInc 2.016 [En línea]. Available:http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100. [Último acceso: 15 de Abril de 2017]

[26] Dr. Alistair Cockburn, «Using Both Incremental and Iterative Development», Software Ingeniering Technolgy [En línea]. Available http://static1.1.sqspcdn.com/static/f/702523/9242211/1288741989673/200805-Cockburn.pdf?token=b4O8EtA2%2F7Dyr8oBCXwjr54d%2BTE%3D. [Último acceso: 15 de Abril de 2017]

[27] Scrum Guides, «Scrum Guide», Scrum.Org & ScrumInc 2.016 [En línea]. Available: http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100. [Último acceso: 15 de Abril de 2017]

[28] Scrum Guides, «Scrum Guide», Scrum.Org & ScrumInc 2.016 [En línea]. Available: http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100. [Último acceso: 15 de Abril de 2017]

[29] Scrum Guides, «Scrum Guide», Scrum.Org & ScrumInc 2.016 [En línea]. Available: http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100. [Último acceso: 16 de Abril de 2017]

[30] Scrum Guides, «Scrum Guide», Scrum.Org & ScrumInc 2.016 [En línea]. Available: http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-

US.pdf#zoom=100. [Último acceso: 16 de Abril de 2017]

[31] Dan Radigan, «Epics, stories, versions, and sprints», Atlassian 2015 [En línea]. Available: http://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100. [Último acceso: 16 de Abril de 2017]

[32] Andersen, Erling S., «Milestone planning--a different planning approach», Project Management Institute 2006 [En línea]. Available: https://www.atlassian.com/agile/delivery-vehicles. [Último acceso: 08 de Julio de 2017]

[33] Ilyas F., «Public Cloud Services Comparison », Born on Cloud, 2017 [En línea]. Available: https://ilyas-it83.github.io/CloudComparer/. [Último acceso: 10 de Julio de 2017]

[34] Amazon Web Services, «AWS Lambda FAQs», Amazon, 2017 [En línea]. Available: https://aws.amazon.com/lambda/faqs/. [Último acceso: 10 de Julio de 2017]

[35] Microsoft Azure, «An introduction to Azure Functions», Microsoft, 2017 [En línea]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview. [Último acceso: 10 de Julio de 2017]

[36] Google Cloud Platform, «Quickstart», Google, 2017 [En línea]. Available: https://cloud.google.com/functions/docs/quickstart. [Último acceso: 10 de Julio de 2017]

[37] Jess Panni, «AWS vs Azure vs Google Cloud Platform – Database», endjin, 2016 [En línea]. Available: https://blogs.endjin.com/2016/08/aws-vs-azure-vs-google-cloud-platform-database/. [Último acceso: 10 de Julio de 2017]

[38] Juan María Fiz, «AWS vs. Google Cloud (2/2): bases de datos, redes, analytics y mucho más», Paradigma, 2016 [En línea]. Available: https://www.paradigmadigital.com/techbiz/aws-vs-google-cloud-22-bases-datos-redes-analytics-mucho-mas/. [Último acceso: 10 de Julio de 2017]

[39] Wilson Mar, «API Management Evaluation», Wilson Mar, 2016 [En línea]. Available: https://wilsonmar.github.io/api-management-evaluation/#TotalPoints. [Último acceso: 10 de Julio de 2017]

[40] Rajat Bhargav, «Google Cloud IAM vs Azure Active Directory», JumpCloud, 2017 [En línea]. Available: https://jumpcloud.com/blog/google-cloud-iam-vs-azure-active-directory/. [Último acceso: 10 de Julio de 2017]

[41] Greg Keller, «Google Cloud IAM vs AWS IAM», JumpCloud, 2017 [En línea]. Available: https://jumpcloud.com/blog/google-cloud-iam-vs-aws-iam/. [Último acceso: 10 de Julio de 2017]

[42] Jess Panni, «AWS vs Azure vs Google Cloud Platform – Compute», endjin, 2016 [En línea]. Available: https://blogs.endjin.com/2016/07/aws-vs-azure-vs-google-cloud-platform-compute/ [Último acceso: 10 de Julio de 2017]

[43] Stephen Grider, «Serverless Showdown: AWS Lambda vs Firebase Google

Cloud Functions», Medium, 2017 [En línea]. Available: https://medium.com/@ste.grider/serverless-showdown-aws-lambda-vs-firebase-google-cloud-functions-cc7529bcfa7d. [Último acceso: 10 de Julio de 2017]

[44] Larry Kim, «10 Most Popular Programming Languages Today», Medium, 2015 [En línea]. Available: https://www.inc.com/larry-kim/10-most-popular-programming-languages-today.html. [Último acceso: 10 de Julio de 2017]

[45] Ron Miller, « How AWS came to be », Techcrunch, 2016 [En línea]. Available: https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/. [Último acceso: 10 de Julio de 2017]

[46] Amazon Web Services, « All Customer Success Stories», Amazon, 2017 [En línea]. Available: https://aws.amazon.com/solutions/case-studies/all/ [Último acceso: 10 de Julio de 2017]

[47] New Relic Products, «Understanding AWS Lambda Performance—How Much Do Cold Starts Really Matter?», New Relic, 2017 [En línea]. Available: https://blog.newrelic.com/2017/01/11/aws-lambda-cold-start-optimization/ [Último acceso: 14 de Julio de 2017]

[48] Amazon Web Services, «How Does AWS Lambda Run My Code? The Container Model», Amazon, 2017 [En línea]. Available: http://docs.aws.amazon.com/lambda/latest/dg/lambda-introduction.html [Último acceso: 14 de Julio de 2017]

ANEXOS

Anexo A – APIs

1. Recursos

Tabla 3. Recursos y métodos desarrollados.

Fuente: [Elaboración propia]

Recurso	Nombre	Nombre de sub recurso	Método HTTP	Código
API-ACC	accounts	-	GET	API-ACC-1
API-TT	transactiontools	-	GET	API-TT-1
		-	POST	API-TT-2
		transactiontoolId	PUT	API-TT-3
API-TSN	transactions	-	GET	API-TSN-1
		-	POST	API-TSN-2
		transactionId	GET	API-TSN-3
API-BLC	balance	-	GET	API-BLC-1
API-ORD	orders	-	GET	API-ORD-1
		-	POST	API-ORD-2
		orderId	GET	API-ORD-3
		orderId	PUT	API-ORD-4
		payment	POST	API-ORD-5
API-PRD	products	-	GET	API-PRD-1
		productId	GET	API-PRD-2

2. Estructura del API

POST

```
/transactiontools
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X POST -H "Authorization: [[apiKey]]" "https://localhost/transactiontools"
```

Parameters

Body parameters

Name	Description

transactionToolPostRequestBody *	▾ {
	metadata: ▾ {
	pak: string
	pin: string
	transactionToolType: string

Responses

Status: 200 - 200 response

Schema Headers

▾ {
response: string

Status: 400 - 400 response

Schema

▾ {
code: integer
message: string

Figura 5. Recurso transaction tools, método POST.

Fuente: [Elaboración propia]

GET

```
/transactiontools
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/transactiontools"
```

Parameters

Responses

Status: 200 - 200 response

Schema Headers

```
▾ [
  ▾ {
      accountId:            string
      createdDate:          integer
      metadata:             string
      transactionToolId:    string
      transactionToolStatus: string
      transactionToolType:  string
      updatedDate:          integer
  }
```

Status: 400 - 400 response

Schema

```
▾ {
    code:    integer
    message: string
```

Figura 6. Recurso transaction tools, método GET.

Fuente: [Elaboración propia]

PUT

```
/transactiontools/{transactiontoolId}
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X PUT -H "Authorization: {{apiKey}}" "https://localhost/transactiontools/{transactiontoolId}"
```

Parameters

Path parameters

Name	Description
transactiontoolId*	string Required

Body parameters

Name	Description

transactionToolPutRequestBody
*
	▾ {
	metadata: ▾ {
	pak: integer
	pin: string
	transactionToolStatus: string
	transactionToolType: string

Responses

Status: 200 - 200 response

Schema Headers

▾ {
response: string

Status: 400 - 400 response

Schema

▾ {
code: integer
message: string

Figura 7. Recurso transaction tools, método PUT.

Fuente: [Elaboración propia]

GET

```
/accounts
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/accounts"
```

Parameters

Responses

Status: 200 - 200 response

Schema Headers

```
▾ {
  accountId:          string
  contactFirstName:   string
  contactLastName:    string
  contactMetadata:    string
  contactPicture:     string
  createdDate:        integer
  email:              string
  metadata:           string
  status:             string
  type:               string
  userExternalKey:    string
  username:           string
```

Status: 400 - 400 response

Schema

```
▾ {
  code:     integer
  message:  string
```

Figura 8. Recurso accounts, método GET

Fuente: [Elaboración propia]

GET

/balance

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/balance"
```

Parameters

Responses

Status: 200 - 200 response

Schema Headers

```
▾ {
  balance: string
```

Status: 400 - 400 response

Schema

```
▾ {
  code: integer
  message: string
```

Figura 9. Recurso balance, método GET.

Fuente: [Elaboración propia]

GET

```
/transactions
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/transactions"
```

Parameters

Responses

Status: 200 - 200 response

Schema Headers

```
▾ [
  ▾ {
      accountIdTo:      string
      amount:           string
      metadata:         string
      transactionDate:  string
      transactionId:    string
      transactionStatus: string
      transactionType:  string
  }
]
```

Status: 400 - 400 response

Schema

```
▾ {
    code:    integer
    message: string
}
```

Figura 10. Recurso transactions, método GET.

Fuente: [Elaboración propia]

GET

/transactions/{transactionId}

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/transactions/{transactionId}"
```

Parameters

Path parameters

Name	Description
transactionId*	string Required

Responses

Status: 200 - 200 response

Schema Headers

```
▾ {
  accountIdTo:      string
  amount:           string
  metadata:         string
  transactionDate:  string
  transactionId:    string
  transactionStatus: string
  transactionType:  string
```

Status: 400 - 400 response

Schema

```
▾ {
  code:    integer
  message: string
```

Figura 11. Recurso transactions, método GET por id.

Fuente: [Elaboración propia]

POST

/transactions

Usage and SDK Samples

curl -X POST -H "Authorization: {apiKey}" "https://localhost/transactions"

Parameters

Body parameters

Name	Description
transactionFromDiscountBody *	* { address: string amount: string cardHolderId: string cardHolderName: string cardNumber: string city: string cvv: string expirationDate: string state: string zipCode: string

Responses

Status: 200 - 200 response

Schema Example

* {
amount: string
approved: string
bank: string
dateTime: string
deferred: string
id: string
lat: string
message: string
orderNumber: string
payment * {
address: string
amount: string
cardHolderId: string
cardHolderName: string
cardNumber: string
city: string
cvv: string
expirationDate: string
state: string
zipCode: string
reference: string
responseCode: string
sessionId: string
voucher: string

Status: 400 - 400 response

Schema

* {
code: integer
message: string

Figura 12. Recurso transactions, método POST.

Fuente: [Elaboración propia]

POST

/orders

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X POST -H "Authorization: [[apiKey]]" "https://localhost/orders"
```

Parameters

Body parameters

Name	Description
orderPostRequestBody *	▾ { products: ▾ [▾ { productId: string quantity: integer }]

Responses

Status: 200 - 200 response

Schema

```
▾ {
  response: string
```

Status: 400 - 400 response

Schema

```
▾ {
  code: integer
  message: string
```

Figura 13. Recurso orders, método POST.

Fuente: [Elaboración propia]

GET

```
/orders
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/orders"
```

Parameters

Responses

Status: 200 - 200 response

Schema

```
▾ [
  ▾ {
      accountId:    string
      createdDate:  integer
      metadata:     string
      orderDate:    integer
      orderId:      string
      orderStatus:  string
      productList:  ▾ [
                      ▸ {}
                    ]
  }
```

Status: 400 - 400 response

Schema

```
▾ {
    code:    integer
    message: string
```

Figura 14. Recurso orders, método GET.

Fuente: [Elaboración propia]

GET

/orders/{orderId}

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [apiKey]]" "https://localhost/orders/{orderId}"
```

Parameters

Path parameters

Name	Description
orderId*	string Required

Responses

Status: 200 - 200 response

Schema

```
+ {
    accountId: string
    createdDate: integer
    metadata: string
    orderDate: integer
    orderId: string
    orderStatus: string
    productList: + [
                   + {
                       createdDate: integer
                       productDescription: string
                       productId: string
                       productImageUrl: string
                       productName: string
                       productPrice: string
                       productQuantity: integer
                       productStatus: string
                       productType: string
                   }
```

Status: 400 - 400 response

Schema

```
+ {
    code: integer
    message: string
```

Figura 15. Recurso orders, método GET por id.

Fuente: [Elaboración propia]

PUT

```
/orders/{orderId}
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X PUT -H "Authorization: [[apiKey]]" "https://localhost/orders/{orderId}"
```

Parameters

Path parameters

Name	Description
orderId*	string Required

Body parameters

Name	Description
orderPutRequestBody *	

Responses

Status: 200 - 200 response

Schema

```
▾ {
  response: string
```

Status: 400 - 400 response

Schema

```
▾ {
  code: integer
  message: string
```

Figura 16. Recurso orders, método PUT.

Fuente: [Elaboración propia]

Figura 17. Recurso orders, sub recurso payment, método POST.

Fuente: [Elaboración propia]

GET

/products

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: [[apiKey]]" "https://localhost/products"
```

Parameters

Responses

Status: 200 - 200 response

Schema

```
▾ [
  ▾ {
    createdDate:          integer
    productDescription:   string
    productId:            string
    productImageUrl:      string
    productName:          string
    productPrice:         string
    productQuantity:      integer
    productStatus:        string
    productType:          string
  }
```

Status: 400 - 400 response

Schema

```
▾ {
  code:     integer
  message:  string
```

Figura 18. Recurso products, método GET.

Fuente: [Elaboración propia]

GET

```
/products/{productId}
```

Usage and SDK Samples

Curl Java Android Obj-C JavaScript C# PHP Perl Python

```
curl -X GET -H "Authorization: {[apiKey]}" "https://localhost/products/{productId}"
```

Parameters

Path parameters

Name	Description
productId*	string Required

Responses

Status: 200 - 200 response

Schema

```
▾ {
  createdDate:        integer
  productDescription: string
  productId:          string
  productImageUrl:    string
  productName:        string
  productPrice:       string
  productQuantity:    integer
  productStatus:      string
  productType:        string
```

Status: 400 - 400 response

Schema

```
▾ {
  code:    integer
  message: string
```

Figura 19. Recurso products, método GET por id.

Fuente: [Elaboración propia]

Anexo B – Microservicios

Tabla 4. Clasificación de microservicios desarrollados.

Fuente: [Elaboración propia]

Tipo	Microservicio	Código
Evento de Cognito	Tr-Cognito-PostConfirmation	TR-CGT-ACC-1
Servicio web	Ws-Account-Get	MS-ACC-1
Servicio web	Ws-Balance-Get	MS-ACC-2
Servicio web	Ws-TransactionTool-Get	MS-TT-1
Servicio web	Ws-TransactionTool-Post	MS-TT-2
Servicio web	Ws-TransactionTool-Put	MS-TT-3
Evento de DynamoDB	Tr-DB-TransactionTool-Manager	TR-DB-TT-1
Evento de DynamoDB	Tr-DB-TransactionTool-TransactionToolTicket	TR-DB-TT-2
Servicio web	Ws-Transaction-Get	MS-TSN-1
Servicio web	Ws-TransactionInstapago-Post	MS-TSN-2
Servicio web	Ws-Transaction-Get-Id	MS-TSN-3
Evento de DynamoDB	Tr-DB-Transaction-Manager	TR-DB-TSN-1
Evento de DynamoDB	Tr-DB-Transaction-Authorizer	TR-DB-TSN-2
Evento de DynamoDB	Tr-DB-Balance-Manager	TR-DB-TSN-3
Evento de DynamoDB	Tr-DB-Balance-Transaction	TR-DB-TSN-4
Servicio web	Ws-Product-Get	MS-STK-1
Servicio web	Ws-Product-Get-Id	MS-STK-2
Servicio web	Ws-Order-Get	MS-ORD-1
Servicio web	Ws-Order-Post	MS-ORD-2
Servicio web	Ws-Order-Put	MS-ORD-3
Servicio web	Ws-Order-Get-Id	MS-ORD-4
Servicio web	Ws-OrderPayment-Post	MS-ORD-5
Evento de DynamoDB	Tr-DB-Order-Manager	TR-DB-ORD-1
Evento de DynamoDB	Tr-DB-Order-Stock	TR-DB-ORD-2

Evento de DynamoDB	Tr-DB-Order-Put	TR-DB-ORD-3
Evento de DynamoDB	Tr-DB-Stock-Manager	TR-DB-STK-1
Evento de DynamoDB	Tr-DB-OrderPayment-Complete	TR-DB-STK-2

Anexo C – Tablas de base de datos

Tabla 5. Tabla Accounts.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
Accounts	TBD-ACC-1	AccountId	Identificación autogenerada de la cuenta de usuario	{ "AccountId": "31f82bb4-b041-43aa-ad34-1fdaeea01e5d", "ContactEmail": "aucab21@gmail.com", "ContactFirstName": "Alejandro", "ContactImageUrl": "NO_IMAGE", "ContactLastName": "Garcia", "ContactMetadata":{"description":" NO_DESCRIPTION", "documentId":"V-20800824"}, "CreatedDate": "2017-05-06T18:27:43.840Z", "Metadata": "NO_METADATA", "Status": "ACTIVE", "Type": "CLIENT", "UserExternalKey": "f2a9edf6-2696-4acb-8758-fc5fd0c26232", "Username": "adosale" }
		ContactEmail	Correo electrónico de la cuenta	
		ContactFirstName	Nombre del usuario	
		ContactLastName	Apellido del usuario	
		ContactImageUrl	Url de la imagen perteneciente al usuario	
		ContactMetadata	Información adicional del usuario	
		CreatedDate	Fecha de creación de la cuenta	
		Metadata	Información adicional según el tipo de cuenta	
		Status	Estatus de la cuenta: activo o inactivo	
		Type	Tipo de cuenta: cliente o comercio	
		UserExternalKey	Identificación externa de usuario en Cognito	

		Username	Nombre de usuario de la cuenta	

Tabla 6. Tabla Balance.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
Balance	TBD-BLC-1	AccountId	Identificación autogenerada de la cuenta de usuario	{ "AccountId": "31f82bb4-b041-43aa-ad34-1fdaeea01e5d", "CreatedDate": "2017-06-02T19:07:56.225Z", "CurrentBalance": 500, "Sequence": 1, "TransactionId": "2c7fb26b-682b-4a6e-b65d-ca2afb04054f" }
		Sequence	Número incremental que representa el balance registrado. Su valor inicial es 0, indicando la primera transacción que se le registro a la cuenta	
		CreatedDate	Fecha de creación del balance	
		CurrentBalance	Valor balance actual del usuario	
		TransactionId	Identificación autogenerada de la transacción, representa la transacción que generó el balance	

Tabla 7. Tabla Transactions.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
Transactions	TBD-TSN-1	TransactionId	Identificación autogenerada de la transacción, representa la transacción que generó el balance	{ "AccountIdTo": "31f82bb4-b041-43aa-ad34-1fdaeea01e5d", "Amount": 5000, "CreatedDate": "2017-06-14T21:29:47.046Z", "Metadata": {"username":"adosale"}, "SearchKey": "::31f82bb4-b041-43aa-ad34-1fdaeea01e5d", "TransactionDate": "2017-06-14T21:29:47.045Z", "TransactionId": "a8af67c7-2ac5-4d11-8b4d-740a091b3c70", "TransactionStatus": "ACCREDITED", "TransactionType": "CASHIN_INSTAPAGO", "UpdatedDate": "2017-06-14T21:29:47.046Z" }
		TransactionType	Tipo de la transacción: Consumo: estacionamiento, recarga de saldo, transferencia de dinero. Sistema: impuesto sobre transacciones, reembolsos	
		TransactionStatus	Estatus de la transacción: creada, acreditada, debitada	
		CreatedDate	Fecha de creación del objeto a base de datos	
		UpdatedDate	Fecha de modificación de la transacción	
		Metadata	Información adicional según el tipo de la transacción	
		TransactionDate	Fecha de la transacción	

		Amount	Monto de la transacción	
		AccountIdTo	Identificación autogenerada de la cuenta destinatario de usuario	
		AccountIdFrom	Identificación autogenerada de la cuenta emisora de usuario	

Tabla 8. Tabla TransactionTools.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
Transaction Tools	TBD-TT-1	AccountId	Identificación autogenerada de la cuenta de usuario	{ "AccountId": "31f82bb4-b041-43aa-ad34-1fdaeea01e5d", "CreatedDate": "2017-07-03T16:29:05.736Z", "Metadata": {"pin":"AAZZ","pak":1199,"nfcTag":"0457AF5AD84981"}, "TransactionToolId": "9f5f434d-fba0-40b2-9ddc-5446307cf261", "TransactionToolStatus": "ACTIVE", "TransactionToolType": "NFC_TAG", "UpdatedDate": "2017-07-03T19:03:19.531Z" }
		Transaction ToolId	Identificación autogenerada del transaction tool	
		CreatedDate	Fecha de creación del transaction tool	
		Metadata	Información adicional según el tipo de transaction tool	
		Transaction ToolStatus	Estatus del transaction tool: activo o inactivo	

		Transaction ToolType	Tipo del transaction tool: tag NFC o punto de servicio CityWallet	
		UpdatedDat e	Fecha de modificación del transaction tool	

Tabla 9. Tabla TransactionToolTicket.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
TransactionTool sTicket	TBD-TT-2	TransactionTool TicketId	Identificación autogenerada del transaction tool ticket	{ "Metadata": {"pin":"AAZZ","pak":1199,"nfc Tag":"0457AF5AD84981"}, "TransactionToolStatus": "UNCLAIMED", "TransactionToolTicketId": "033", "TransactionToolType": "NFC_TAG" }
		Metadata	Información adicional según el tipo de transaction tool ticket según el tipo	
		TransactionTool Status	Estatus del transaction tool ticket: reclamado o no reclamado	
		TransactionTool Type	Tipo del transaction tool ticket: tag NFC	

Tabla 10. Tabla Stock.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
Stock	TBD-STK-1	StockId	Identificación autogenerada del inventario de productos	{ "CreatedDate": "2017-06-06T21:44:41.070Z", "LastUpdateId": "b899a3bc-525f-4dd9-ac66-7a14b58d3bfd::5cc3170d-6212-47ca-8987-5804db96bec0", "Products": [{"productId":"1111","productName":"Sticker","productDescription":"Test description", "productStatus":"ACTIVE", "productType":"STICKER", "productQuantity":190, "productPrice":200.00, "productImageUrl":"test_image", "createdDate":1495719740238}], "StockId": "0000", "UpdatedDate": "2017-07-03T19:30:59.117Z" }
		Products	Lista de productos existentes en el sistema.	
		CreatedDate	Fecha de creación del inventario de productos	
		LastUpdateId	Identificación autogenerada de la orden, representando la última orden que modificó el inventario de productos	
		UpdatedDate	Fecha de modificación del inventario de productos	

Tabla 11. Tabla Orders.

Fuente: [Elaboración propia]

Nombre	Código	Atributo	Descripción	Modelo
Orders	TBD-ORD-1	OrderId	Identificación autogenerada de la orden	{ "AccountId": "5cc3170d-6212-47ca-8987-5804db96bec0", "CreatedDate": "2017-07-03T19:28:25.123Z", "Metadata": "{\"amount\":\"400\",\"id\":\"000dae19-005b-4384-b2f5-4d76e8bd2132\"}", "OrderDate": "2017-07-03T19:28:25.106Z", "OrderId": "b899a3bc-525f-4dd9-ac66-7a14b58d3bfd", "OrderStatus": "PROCESSED", "Products": [{"productId":"1111","productName":"Sticker","productDescription":"Test description", "productStatus":"ACTIVE", "productType":"STICKER", "productQuantity":2, "productPrice":200.00, "productImageUrl":"test_image", "createdDate":1495719740238}] }
		Products	Lista de productos provenientes del inventario	
		AccountId	Identificación autogenerada de la cuenta de usuario	
		Metadata	Información adicional de la orden	
		OrderStatus	Estatus de la order: creada, procesando y procesada	
		OrderDate	Fecha de la orden	
		CreatedDate	Fecha de creación del objeto a base de datos	

Anexo D – *Sprints*

1. *Backlog* del producto

Tabla 12. Backlog del producto.

Fuente: [Elaboración propia]

Épica	Hito	Código
E1	Implementar proceso de agregar una herramienta de transacción como función Lambda	E1H1
E1	Exponer como Restful API el proceso de agregar una herramienta de transacción	E1H2
E1	Implementar proceso de consulta de herramientas de transacción como función Lambda	E1H3
E1	Exponer como Restful API el proceso de consulta de herramientas de transacción	E1H4
E1	Implementar proceso de modificación de herramientas de transacción como función Lambda	E1H5
E1	Exponer como Restful API el proceso de modificación de herramientas de transacción	E1H6
E2	Implementar proceso de agregar una transacción de recarga como función Lambda	E2H1
E2	Exponer como Restful API el proceso de agregar una transacción de recarga	E2H2
E2	Implementar proceso de consulta de transacciones como función Lambda	E2H3
E2	Exponer como Restful API el proceso de consulta de transacciones	E2H4
E2	Implementar proceso de consulta de transacciones como función Lambda	E2H5
E2	Exponer como Restful API el proceso de consulta de transacciones	E2H6
E3	Implementar proceso de agregar una cuenta de usuario como función Lambda	E3H1
E4	Implementar proceso de consulta de cuenta como función Lambda	E4H1
E4	Exponer como Restful API el proceso de reporte de cuentas	E4H2
E5	Implementar proceso de agregar carro de compra como función Lambda	E5H1
E5	Exponer como Restfull API el proceso de agregar carro de compra	E5H2
E5	Implementar proceso de consulta de carro de compra como función Lambda	E5H3
E5	Exponer como Restful API el proceso de consulta de carro de compra	E5H4
E5	Implementar proceso de consulta del detalle de carro de compra como función Lambda	E5H5

E5	Exponer como Restfull API el proceso de consulta del detalle de carro de compra	E5H6
E5	Implementar proceso de modificar carro de compra como función Lambda	E5H7
E5	Exponer como Restful API el proceso de modificar un carro de compra	E5H8
E5	Implementar proceso de consulta de productos de tienda como función Lambda	E5H9
E5	Exponer como Restful API el proceso de consulta de productos de tienda	E5H10
E5	Implementar proceso de consulta del detalle de productos de tienda como función Lambda	E5H11
E5	Exponer como Restful API el proceso de consulta del detalle de productos de tienda	E5H12
E5	Implementar proceso de pago de carro de compra como función Lambda	E5H13
E5	Exponer como Restful API el proceso de pago de un carro de compra	E5H14

1. Backlog dcl *Sprint*

1.1. *Sprint 1*

Tabla 13. Backlog del Sprint 1.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H1	Desarrollo de microservicio Ws-TransactionTool-Post	E1H1A1
	Desarrollo de microservicio Tr-DB-TransactionTool-TransactionToolTicket	E1H1A2
	Integración a microservicio de evento Tr-Db-TransactionTool-Manager	E1H1A3
	Desarrollo de métodos de librería CityWallet Core para agregar TransactionTools	E1H1A4
	Integración de librería CityWallet Core en microservicios desarrollados	E1H1A5
H2	Desarrollo de API Gw para recurso TransactionTool método POST	E1H2A1
H3	Desarrollo de microservicio Ws-TransactionTool-Get	E1H3A1
	Desarrollo de métodos de librería Core para obtener TransactionTools	E1H3A2
	Integración de librería CityWallet Core en microservicios desarrollados	E1H3A3
H4	Desarrollo de API Gw para recurso TransactionTool método GET	E1H4A1
	Desarrollo de microservicio Ws-TransactionTool-Put	E1H5A1
	Desarrollo de métodos de librería Core para modificar TransactionTools	E1H5A2

H5	Integración de librería CityWallet Core en microservicios desarrollados	E1H5A3
H6	Desarrollo de API Gw para recurso TransactionTool método PUT	E1H6A1

1.2. *Sprint* 2

Tabla 14. Backlog del Sprint 2.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H1	Desarrollo de microservicio Ws-TransactionInstapago-Post	E2H1A1
	Desarrollo de métodos de librería CityWallet Core para agregar una transacción de tipo consumo por recarga	E2H1A2
	Integración de librería CityWallet Core en microservicios desarrollados	E2H1A3
H2	Desarrollo de API Gw para recurso Transaction método POST	E2H2A1

1.3. *Sprint* 3

Tabla 15. Backlog del Sprint 3.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H3	Desarrollo de microservicio Ws-Transaction-Get	E2H3A1
	Desarrollo de métodos de librería Core para obtener la lista de transacciones	E2H3A2
	Integración de librería CityWallet Core en microservicios desarrollados	E2H3A3
H4	Desarrollo de API Gw para recurso Transaction método GET	E2H4A1
H5	Desarrollo de microservicio Ws-Transaction-Get-Id	E2H5A1
	Desarrollo de métodos de librería Core para obtener el detalle de una transacción	E2H5A2
	Integración de librería CityWallet Core en microservicios desarrollados	E2H5A3
H6	Desarrollo de API Gw para recurso Transaction método GET por id de transacción	E2H6A1

1.4. *Sprint* 4

Tabla 16. Backlog del Sprint 4.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H1	Desarrollo de microservicio Tr-Cognito-PostConfirmation	E3H1A1
	Desarrollo de métodos de librería Core para agregar una cuenta de tipo usuario	E3H1A2
	Integración de librería CityWallet Core en microservicios desarrollados	E3H1A3

1.5. *Sprint* 5

Tabla 17. Backlog del Sprint 5.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H1	Desarrollo de microservicio Ws-Account-Get	E4H1A1
	Desarrollo de microservicio Ws-Balance-Get	E4H1A2
	Integración de librería CityWallet Core en microservicios desarrollados	E4H1A3
H2	Desarrollo de API Gw para recurso Accounts método GET por id	E4H2A1
	Desarrollo de API Gw para recurso Balance método GET por id	E4H2A2

1.6. *Sprint* 6

Tabla 18. Backlog del Sprint 6.

Fuente: [Elaboración propia]

Hito	Actividad	Código
	Desarrollo de microservicio Ws-Order-Post	E5H1A1
	Desarrollo de métodos de librería Core para agregar carro de compra	E5H1A2

H1	Integración de librería CityWallet Core en microservicios desarrollados	E5H1A3
H2	Desarrollo de API Gw para recurso Orders método POST	E5H2A1

1.7. *Sprint* 7

Tabla 19. Backlog del Sprint 7.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H3	Desarrollo de microservicio Ws-Order-Get	E5H3A1
	Desarrollo de métodos de librería Core para obtener carro de compra	E5H3A2
	Integración de librería CityWallet Core en microservicios desarrollados	E5H3A3
H4	Desarrollo de API Gw para recurso Orders método GET	E5H4A1
H5	Desarrollo de microservicio Ws-Order-Get-Id	E5H5A1
	Desarrollo de métodos de librería Core para obtener detalle de carro de compra	E5H5A2
	Integración de librería CityWallet Core en microservicios desarrollados	E5H5A3
H6	Desarrollo de API Gw para recurso Orders método GET por id	E5H6A1

1.8. *Sprint* 8

Tabla 20. Backlog del Sprint 8.

Fuente: [Elaboración propia]

Hito	Actividad	Código
	Desarrollo de microservicio Ws-Order-Put	E5H7A1
	Desarrollo de métodos de librería Core para modificar carro de compra	E5H7A2

H7	Integración de librería CityWallet Core en microservicios desarrollados	E5H7A3
H8	Desarrollo de API Gw para recurso Orders método PUT	E5H8A1
H9	Desarrollo de microservicio Ws-Product-Get	E5H9A1
	Desarrollo de métodos de librería Core para obtener productos de tienda	E5H9A2
	Integración de librería CityWallet Core en microservicios desarrollados	E5H9A3
H10	Desarrollo de API Gw para recurso Stock método GET	E5H10A1
H11	Desarrollo de microservicio Ws-Product-Get-Id	E5H11A1
	Desarrollo de métodos de librería Core para obtener detalle de productos de tienda	E5H11A2
	Integración de librería CityWallet Core en microservicios desarrollados	E5H11A3
H12	Desarrollo de API Gw para recurso Stock método GET por id	E5H12A1

1.9. Sprint 9

Tabla 21. Backlog del Sprint 9.

Fuente: [Elaboración propia]

Hito	Actividad	Código
H13	Desarrollo de microservicio Ws-OrderPayment-Post	E5H13A1
	Desarrollo de microservicio Tr-DB-Order-Manager	E5H13A2
	Desarrollo de microservicio Tr-DB-Order-Stock	E5H13A3
	Desarrollo de microservicio Tr-DB-Order-Put	E5H13A4
	Desarrollo de microservicio Tr-DB-Stock-Manager	E5H13A5
	Desarrollo de microservicio Tr-DB-OrderPayment-Complete	E5H13A6
	Desarrollo de métodos de librería Core para agregar pago a carro de compra	E5H13A7
	Integración de librería CityWallet Core en microservicios desarrollados	E5H13A8
H14	Desarrollo de API Gw para recurso Orders subrecurso Payment método POST	E5H13A1

Anexo E – Diagramas de flujo

1. Procesos rediseñados

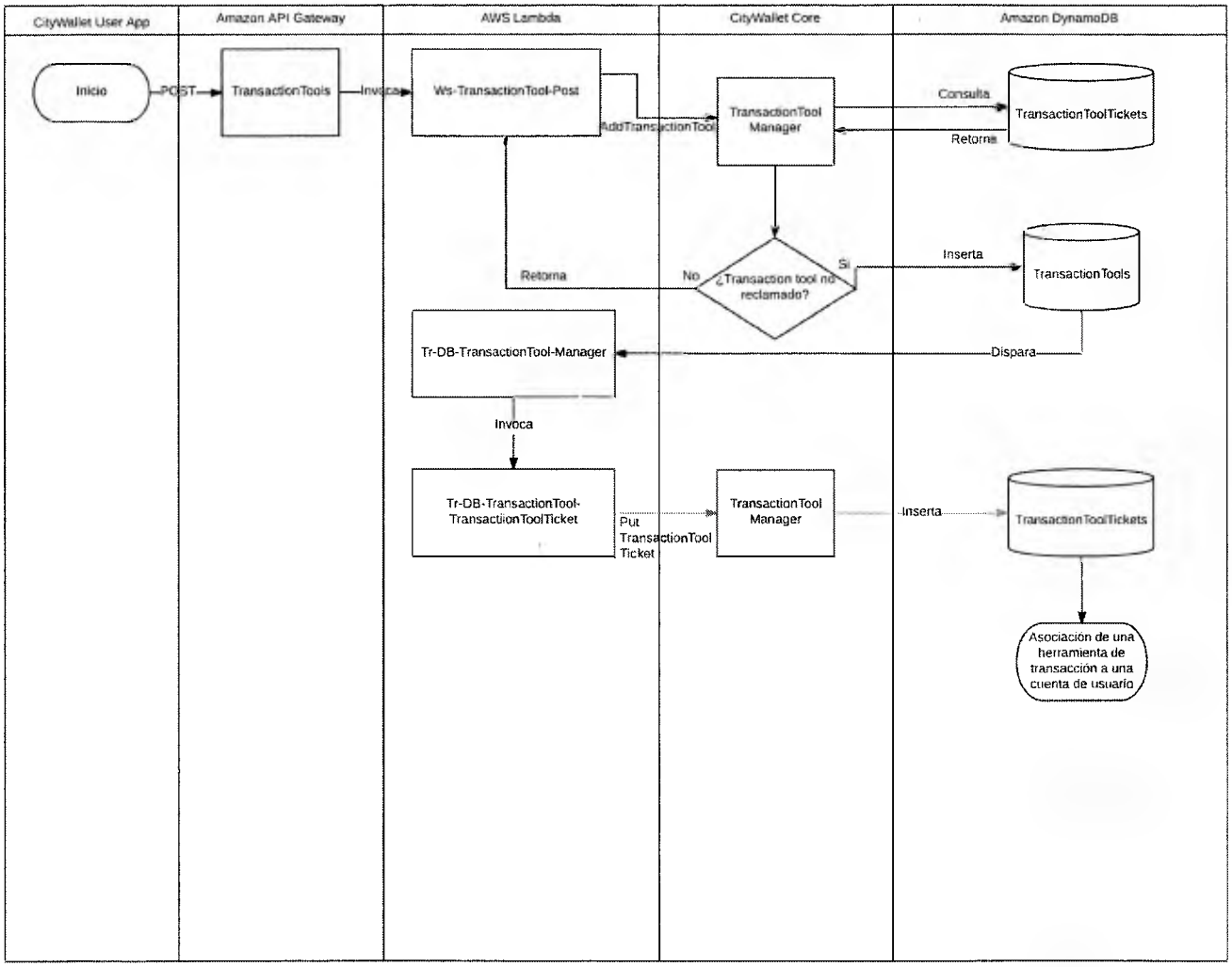


Figura 20. Rediseño de asociación de un transaction tool.

Fuente: [Elaboración propia]

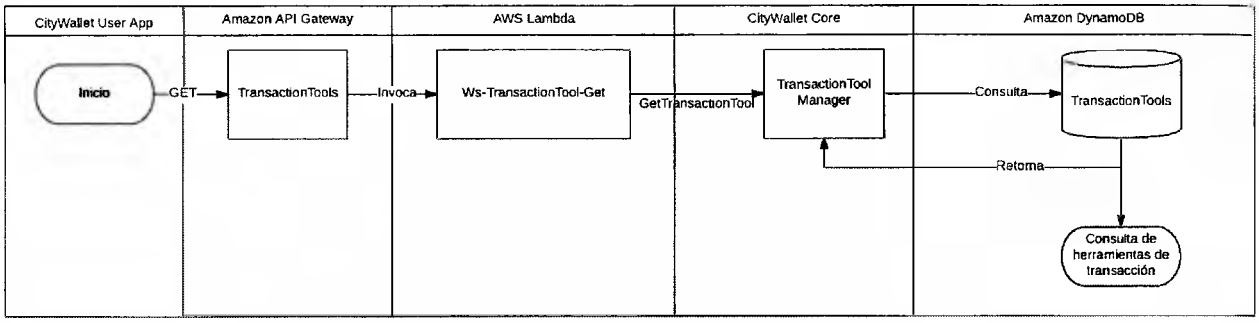


Figura 21. Rediseño de consulta de transaction tools.

Fuente: [Elaboración propia]

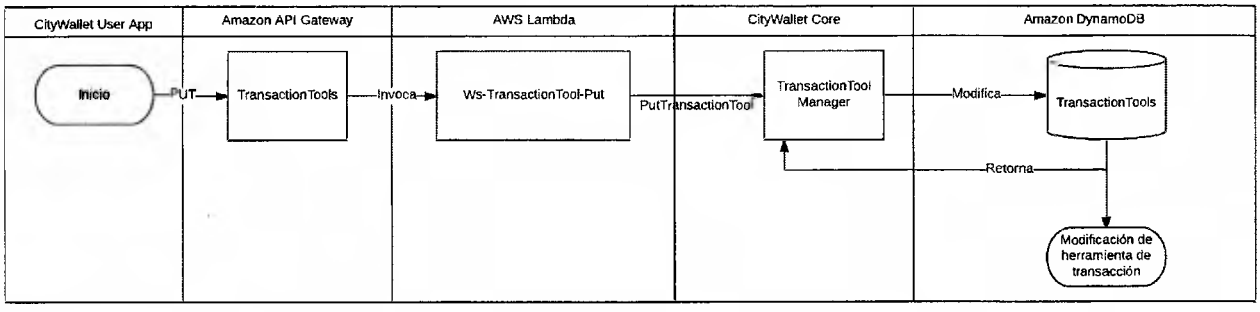


Figura 22. Rediseño de modificación de un transaction tool.

Fuente: [Elaboración propia]

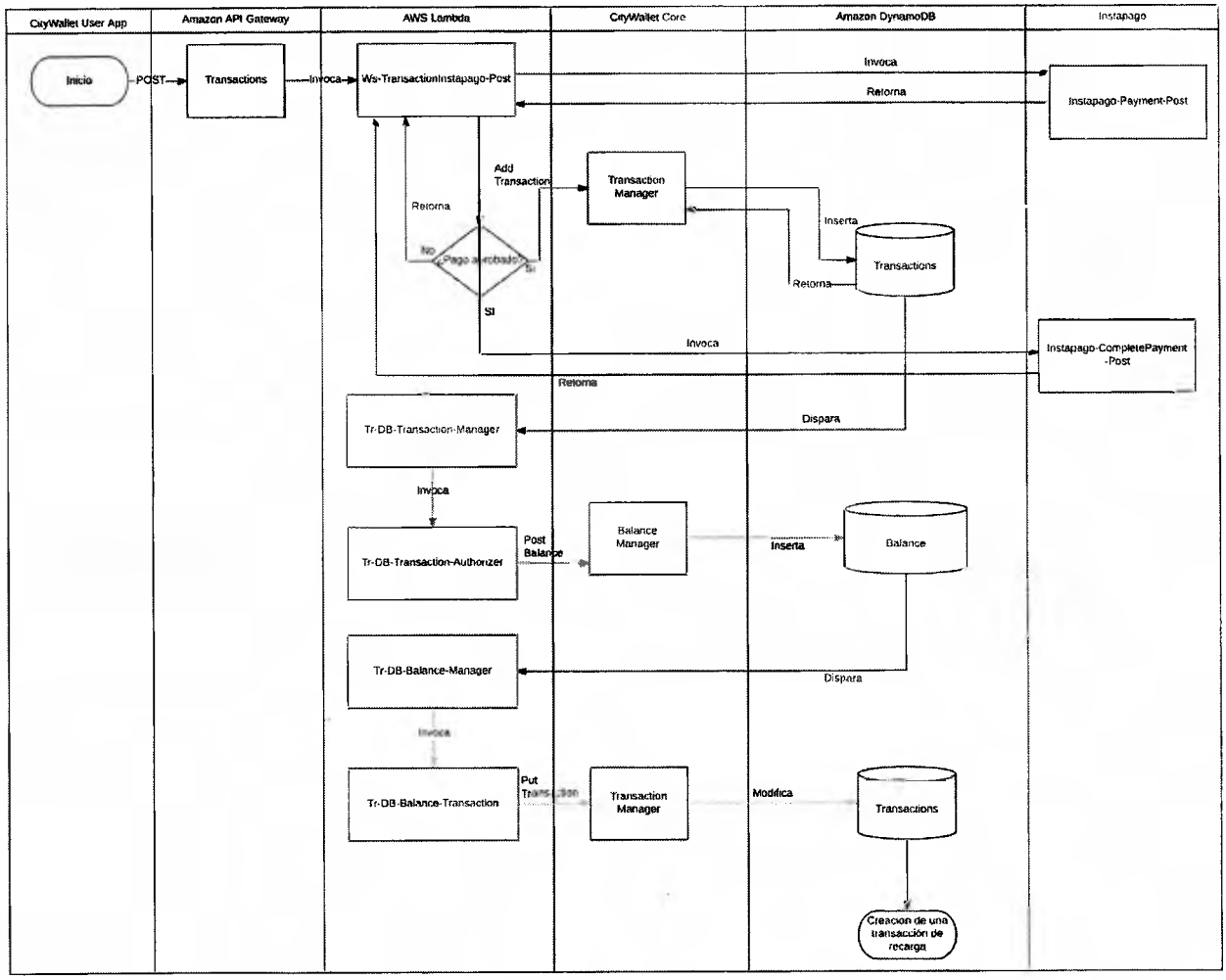


Figura 23. Rediseño de agregar de una transacción de recarga de saldo.

Fuente: [Elaboración propia]

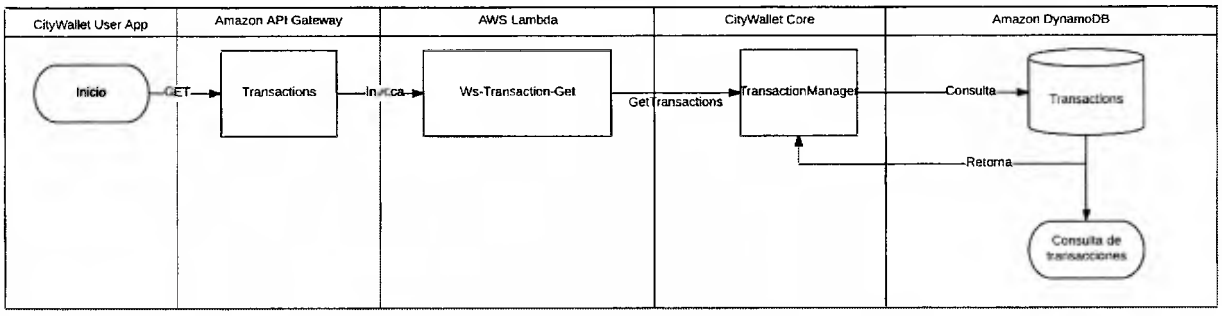


Figura 24. Rediseño de consulta de transacciones.

Fuente: [Elaboración propia]

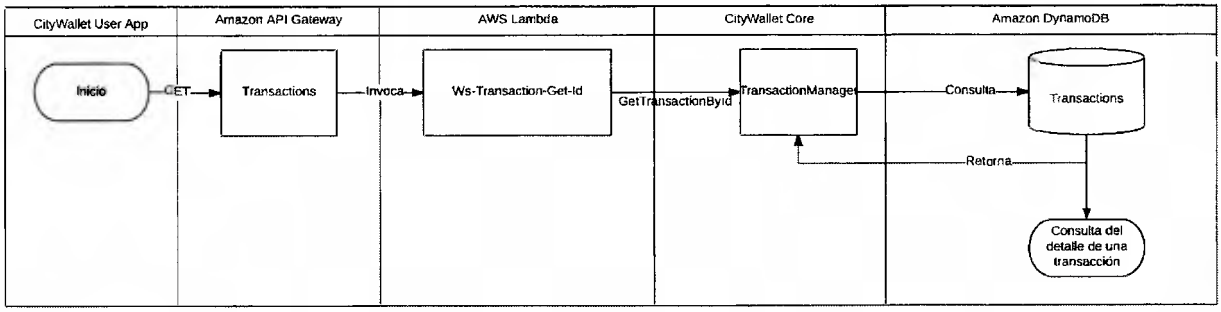


Figura 25. Rediseño de consulta de una transacción.

Fuente: [Elaboración propia]

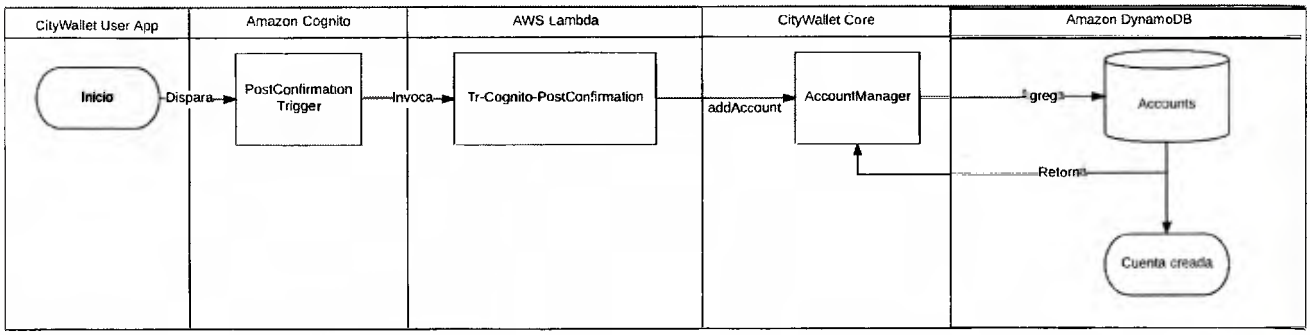


Figura 26. Rediseño de agregar una cuenta de usuario.

Fuente: [Elaboración propia]

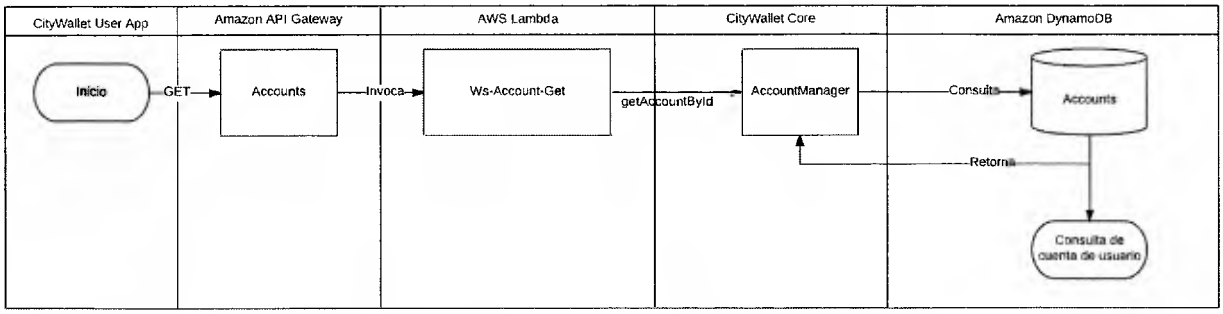


Figura 27. Rediseño de consultar cuenta de usuario.

Fuente: [Elaboración propia]

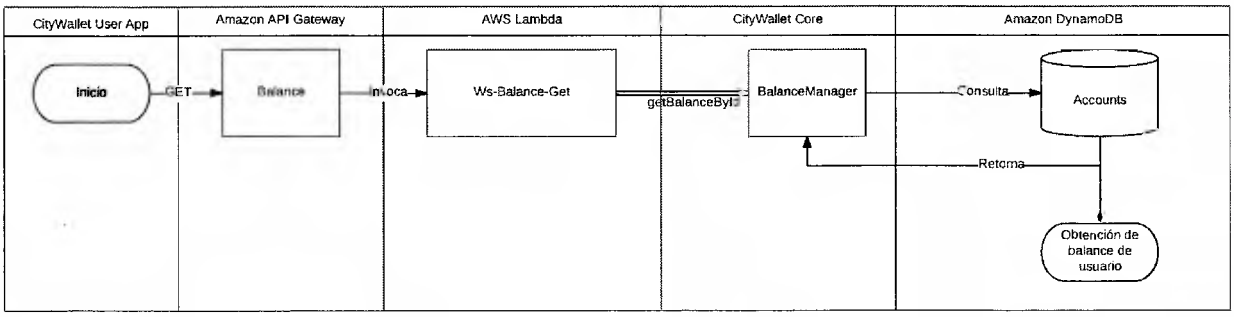


Figura 28. Rediseño de consultar balance de una cuenta de usuario.

Fuente: [Elaboración propia]

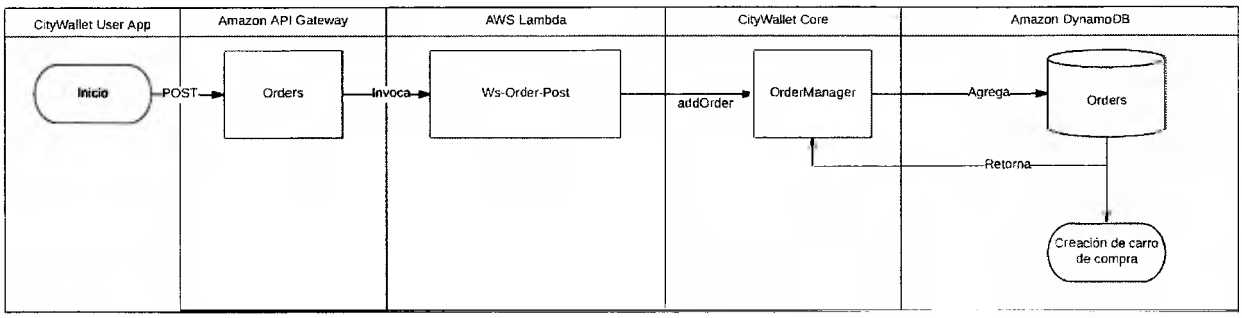


Figura 29. Rediseño de agregar una orden carro de compra.

Fuente: [Elaboración propia]

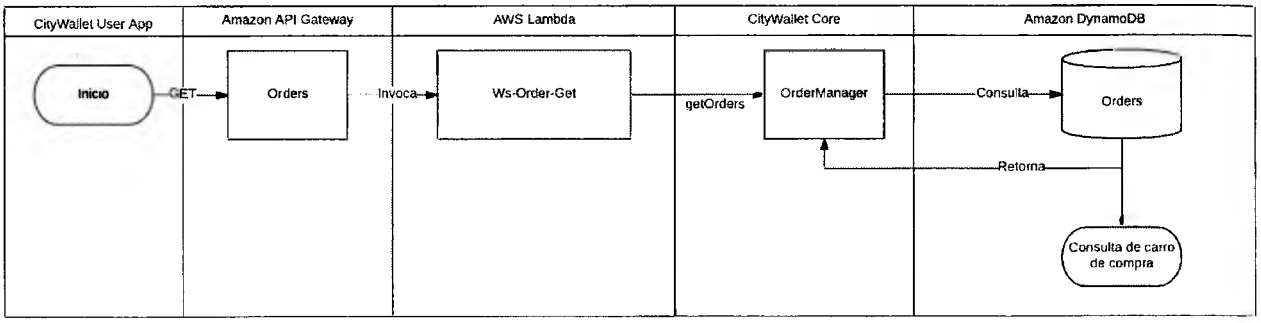


Figura 30. Rediseño de consultar órdenes.

Fuente: [Elaboración propia]

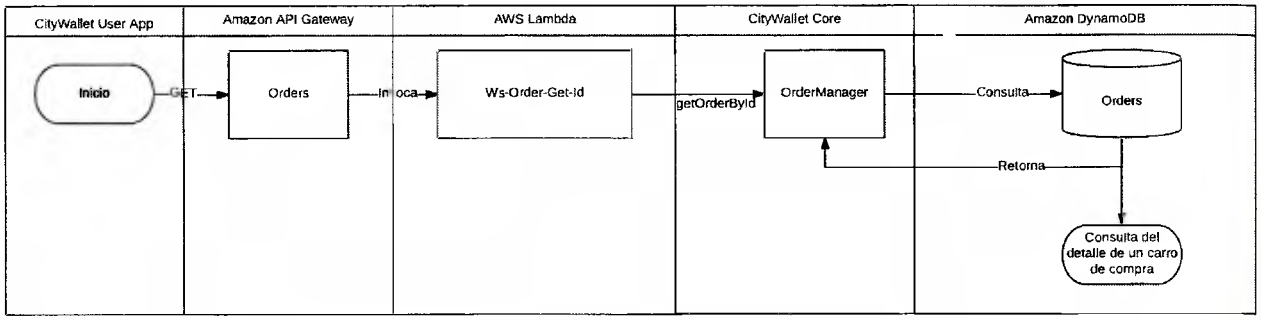


Figura 31. Rediseño de consultar una orden o carro de compra.

Fuente: [Elaboración propia]

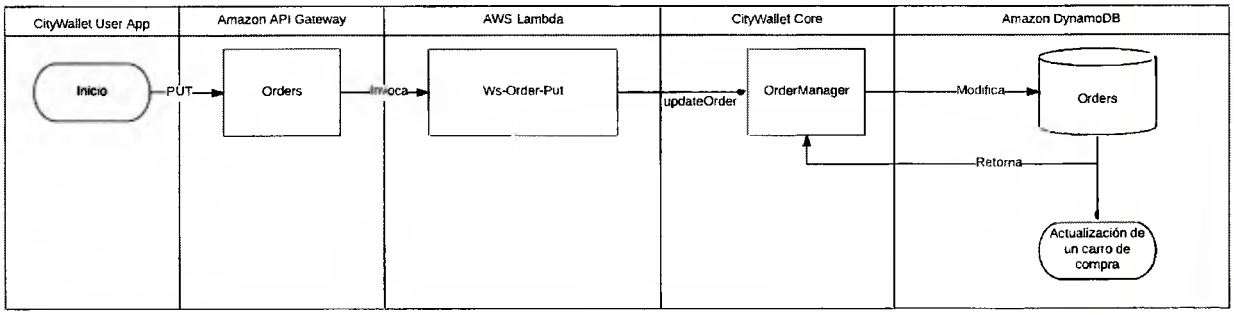


Figura 32. Rediseño de modificar una orden o carro de compra.

Fuente: [Elaboración propia]

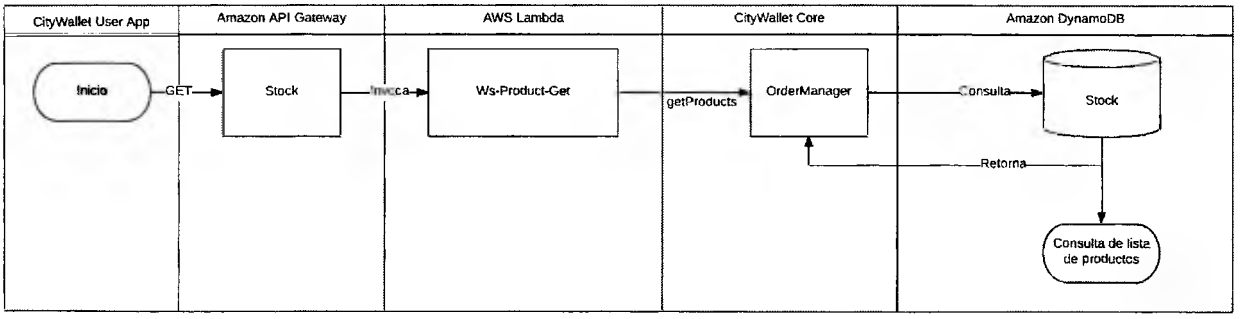


Figura 33. Rediseño de consultar productos del inventario.

Fuente: [Elaboración propia]

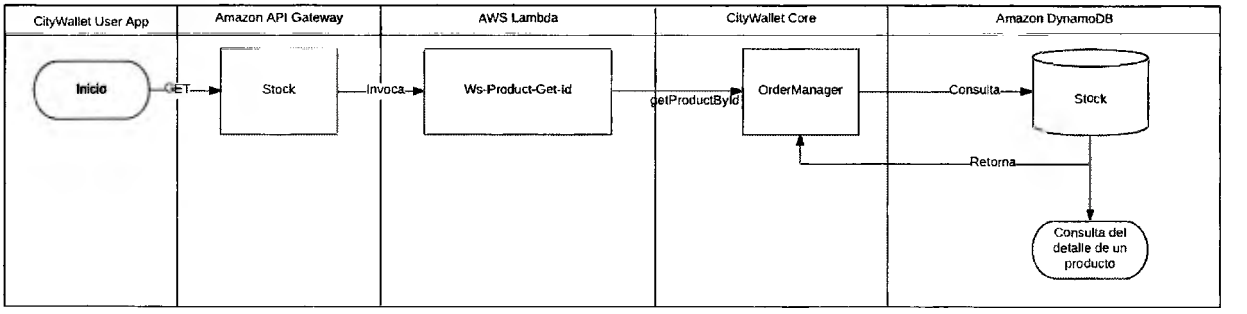


Figura 34. Rediseño de consultar un producto del inventario.

Fuente: [Elaboración propia]

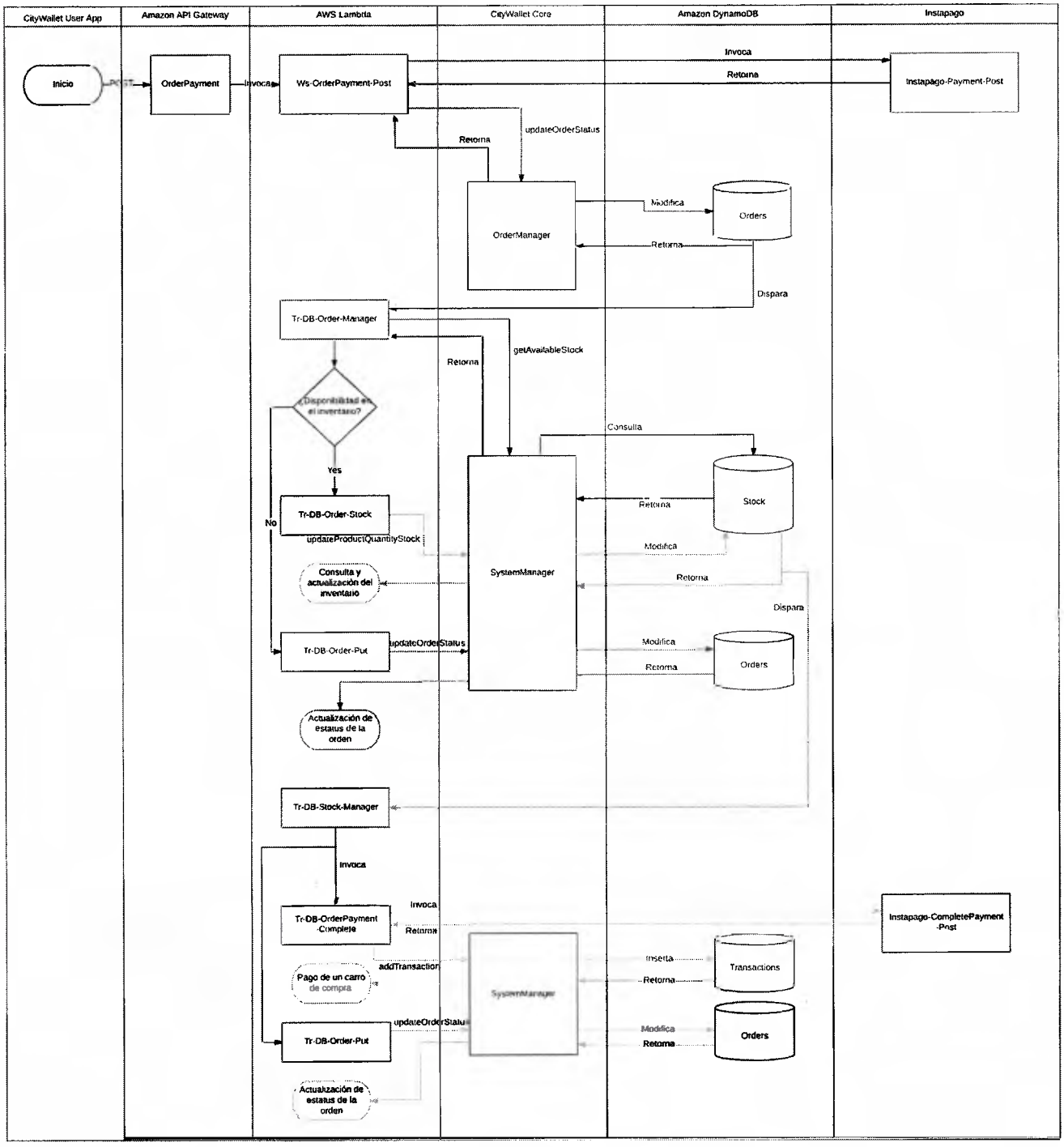


Figura 35. Rediseño de pago de orden o carro de compra.

Fuente: [Elaboración propia]

2. Procesos anteriores

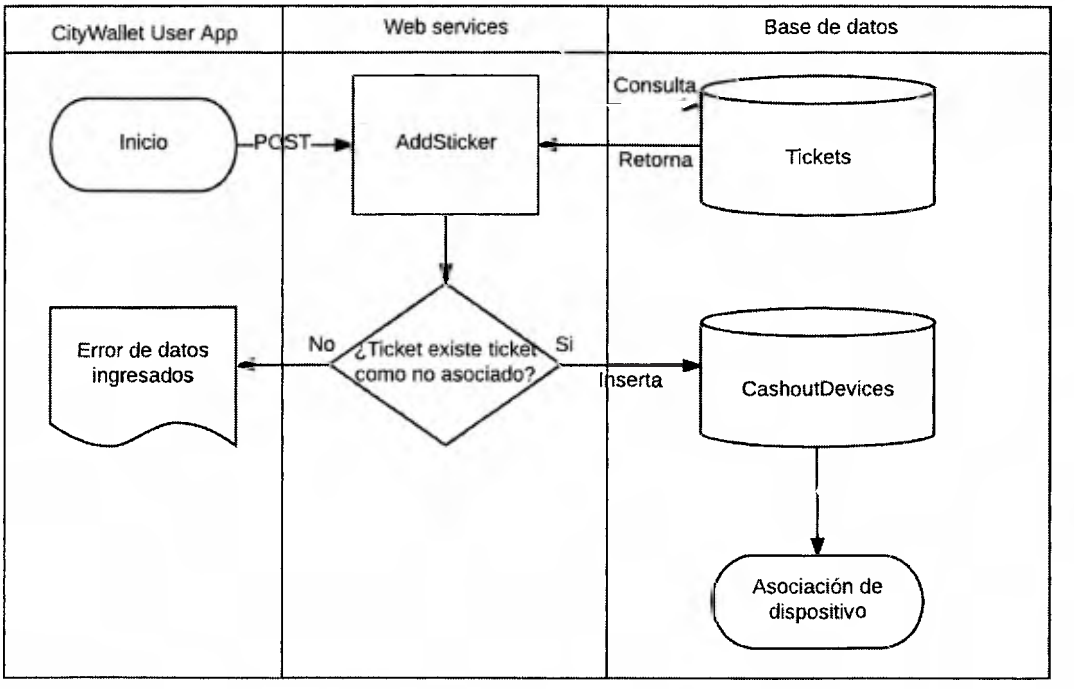


Figura 36. Diseño de asociación de un cashout device.

Fuente: CityWallet

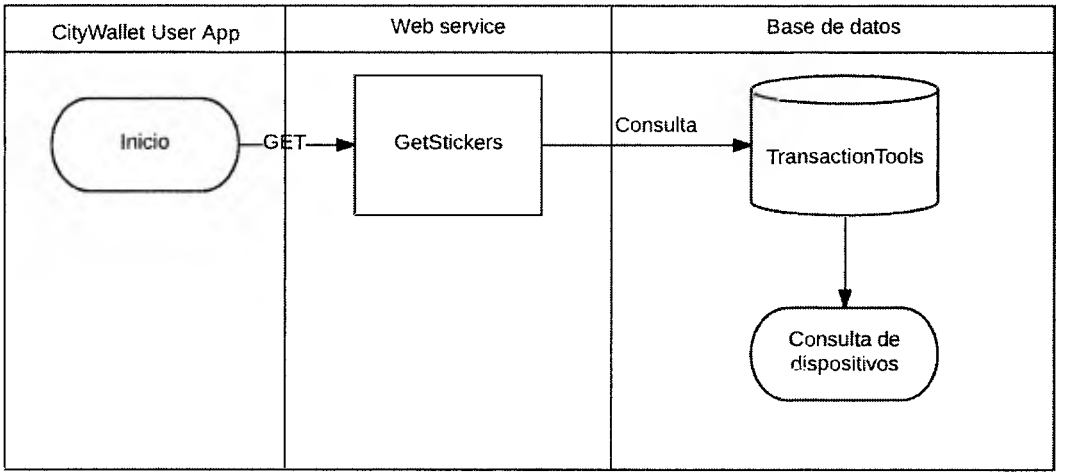


Figura 37. Diseño de consulta de cashout devices.

Fuente: CityWallet

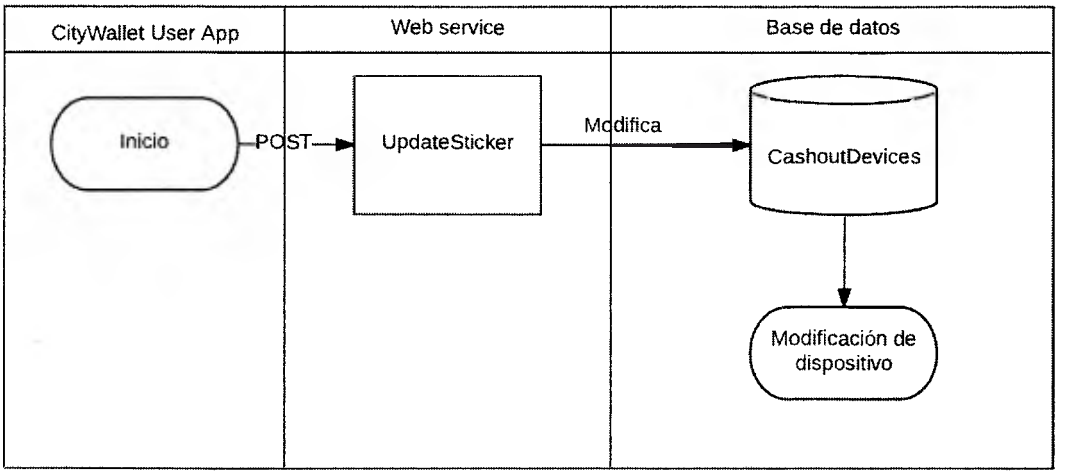


Figura 38. Diseño de modificación de un cashout device.

Fuente: CityWallet

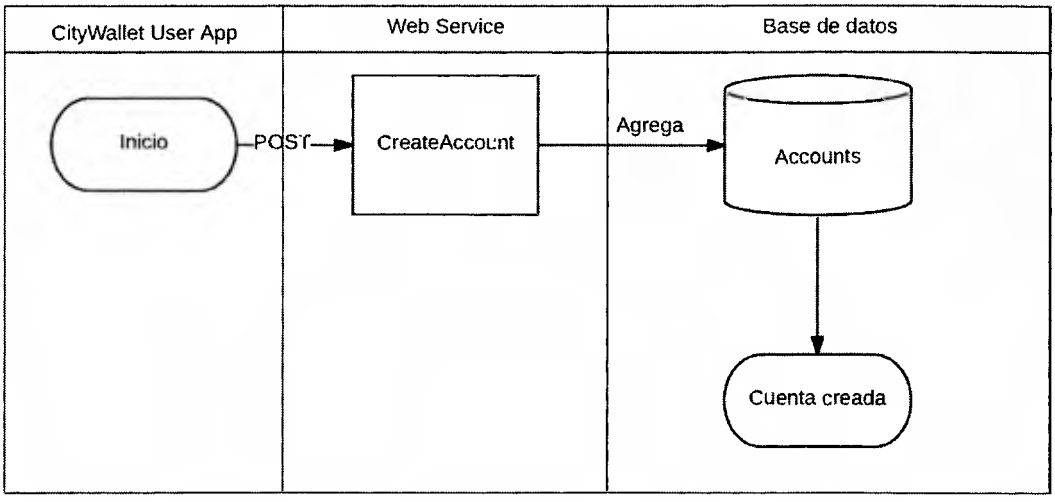


Figura 39. Diseño de creación de cuenta.

Fuente: CityWallet

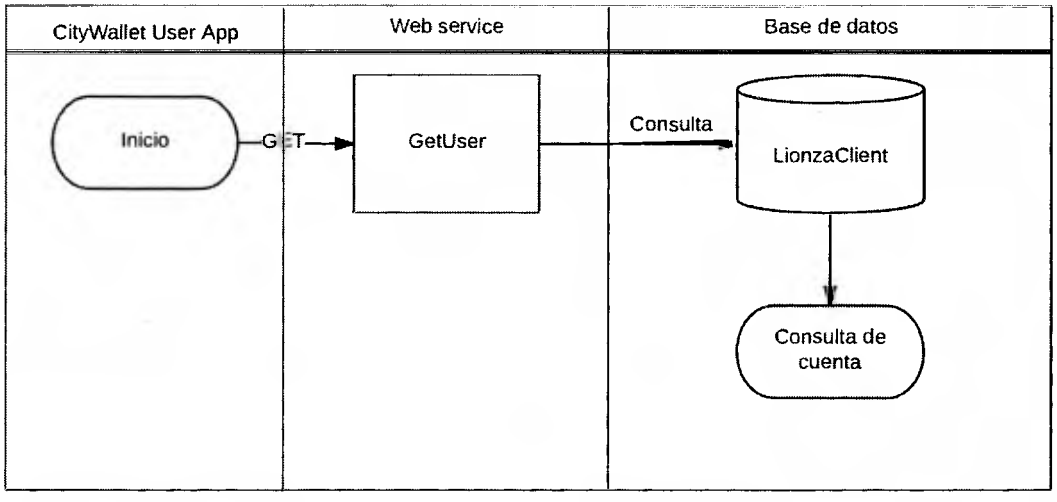


Figura 40. Diseño de consulta de cuenta.

Fuente: CityWallet

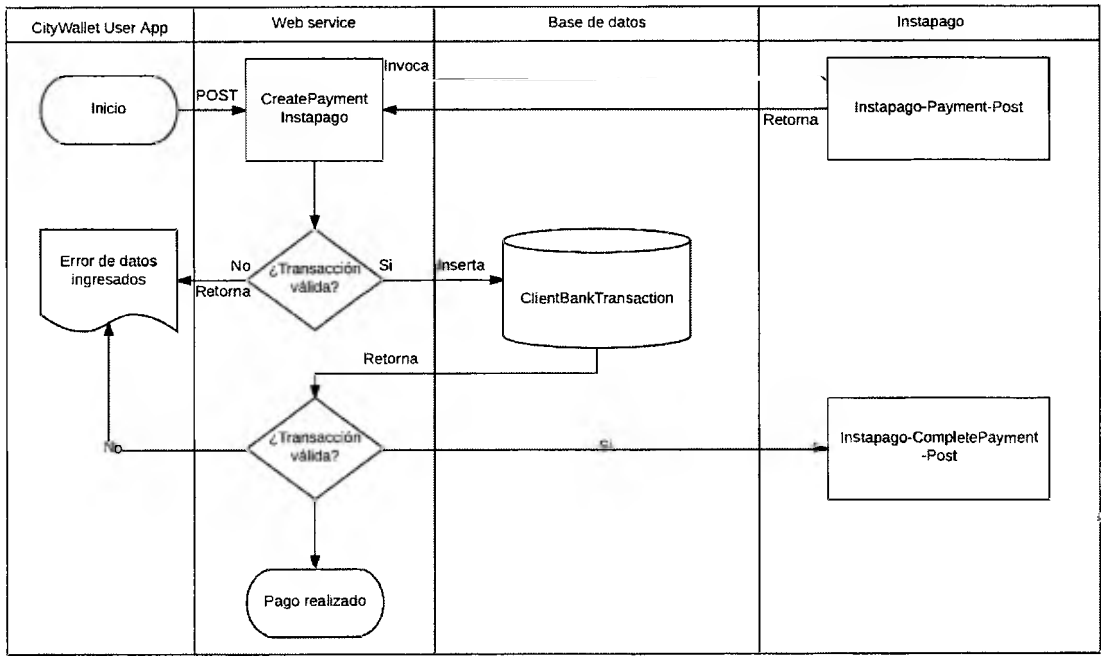


Figura 41. Diseño de pago por InstaPago.

Fuente: CityWallet

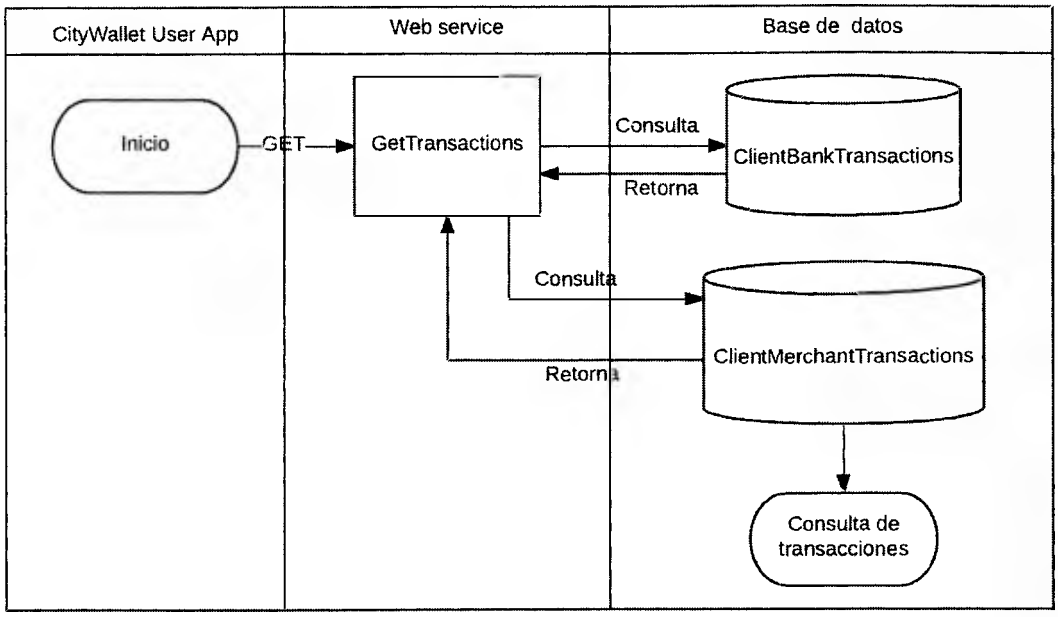


Figura 42. Diseño de consulta de transacciones.

Fuente: CityWallet

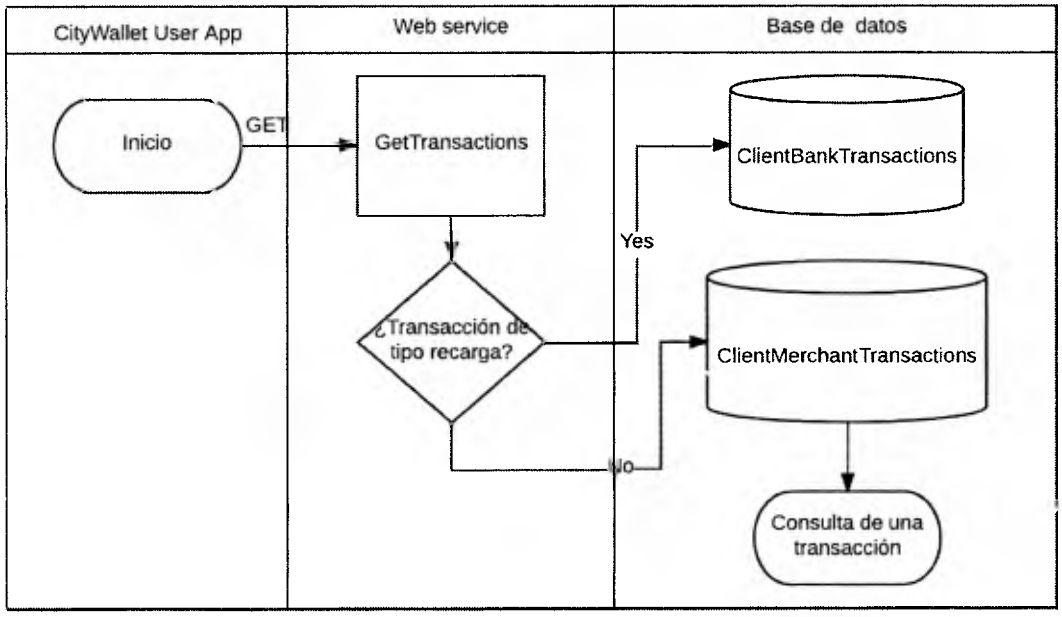


Figura 43. Diseño de consulta de una transacción.

Fuente: CityWallet