



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Sistemas

**Rediseño de arquitectura de software para atender la
alta carga de depósitos en una empresa de medio de
pagos**

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Sistemas

AUTOR

Jose Carlos BUSTAMANTE FALCÓN

ASESOR

Mg. Augusto Parcemon CORTEZ VÁSQUEZ

Lima, Perú

2025



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Bustamante, J. (2025). *Rediseño de arquitectura de software para atender la alta carga de depósitos en una empresa de medio de pagos*. [Trabajo de suficiencia profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Sistemas]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Jose Carlos Bustamante Falcón
Tipo de documento de identidad	DNI
Número de documento de identidad	71387562
URL de ORCID	https://orcid.org/0009-0000-8522-0130
Datos de asesor	
Nombres y apellidos	Augusto Parcemon Cortez Vásquez
Tipo de documento de identidad	DNI
Número de documento de identidad	08634618
URL de ORCID	https://orcid.org/0000-0002-5188-7962
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Jorge Raúl Díaz Muñante
Tipo de documento	DNI
Número de documento de identidad	07216161
Miembro del jurado 1	
Nombres y apellidos	César Augusto Alcántara Loayza
Tipo de documento	DNI
Número de documento de identidad	09132297
Datos de investigación	
Línea de investigación	No aplica
Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento

Ubicación geográfica de la investigación	País: Perú Departamento: Lima Provincia: Lima
	Distrito: Cercado de Lima Jr. Carlos Amezaga No. 375 Universidad Nacional Mayor de San Marcos Latitud: -12.0564232 Longitud: -77.0843327
Año o rango de años en que se realizó la investigación	2024
URL de disciplinas OCDE	2.02.04 -- Ingeniería de sistemas y comunicaciones https://purl.org/pe-repo/ocde/ford#2.02.04



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
Escuela Profesional de Ingeniería de Sistemas

Acta Virtual de Sustentación
del Trabajo de Suficiencia Profesional

Siendo las 16:02 horas del día 15 de enero del año 2025, se reunieron virtualmente los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Mg. Díaz Muñante Jorge Raúl (Presidente), Mg. Alcántara Loayza César Augusto (Miembro) y el Mg. Cortez Vásquez Augusto Parcemon (Miembro Asesor), usando la plataforma Meet (<https://meet.google.com/jtg-ystx-bsz>), para la sustentación virtual del Trabajo de Suficiencia Profesional intitulado: **“REDISEÑO DE ARQUITECTURA DE SOFTWARE PARA ATENDER LA ALTA CARGA DE DEPÓSITOS EN UNA EMPRESA DE MEDIO DE PAGOS”**, por el Bachiller **Bustamante Falcón Jose Carlos**; para obtener el Título Profesional de Ingeniero de Sistemas.

Acto seguido de la exposición del Trabajo de Suficiencia Profesional, el Presidente invitó al Bachiller a dar las respuestas a las preguntas establecidas por los miembros del Jurado.

El Bachiller en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez a las observaciones y preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el Bachiller obtuvo la nota de **18 (DIECIOCHO)**.

A continuación el Presidente de Jurados el Mg. Díaz Muñante Jorge Raúl, declara al Bachiller **Ingeniero de Sistemas**.

Siendo las 16:52 horas, se levantó la sesión.

Presidente

Mg. Díaz Muñante Jorge Raúl

Miembro

Mg. Alcántara Loayza César Augusto

Miembro Asesor

Mg. Cortez Vásquez Augusto Parcemon



CERTIFICADO DE SIMILITUD

Yo, CORTEZ VASQUEZ AUGUSTO PARCEMON en mi condición de asesor acreditado con la Resolución Decanal N° 001059-2024-D-FISI/UNMSM y Oficio N° 0010-PTTSP-FISI-2024 del Trabajo de Suficiencia Profesional cuyo título es Rediseño de arquitectura de software para atender la alta carga de depósitos en una empresa de medio de pagos, presentado por el bachiller BUSTAMANTE FALCÓN JOSE CARLOS para optar el título de Ingeniero de Sistemas CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante la plataforma PIDS, el documento evaluado cuenta con el 4% de similitud, nivel **PERMITIDO** para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional.**

Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del título.

Lima, 8 de enero de 2025

Firma del Asesor

Nombres y Apellidos del asesor: CORTEZ VASQUEZ AUGUSTO PARCEMON

DNI 08634618

Dedicatoria

Dedico el presente trabajo al Señor Jesucristo quien me ayudó a formarme como un varón profesional, con ética y que aporta a la sociedad con sus conocimientos y valores.

Agradecimiento

Agradezco a mis padres por su amor y sustento durante todo el tiempo de mis estudios y formación.

Agradezco a mis hermanas por ser ese ejemplo de excelentes profesionales.

Agradezco a mi esposa por motivarme a ser un mejor hombre cada día y ser mi compañera de vida.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

**Rediseño de arquitectura de software para atender la alta carga de depósitos en
una empresa de medio de pagos**

Autor: Bustamante Falcón, Jose Carlos

Asesor: Cortez Vásquez, Augusto Parcemon

Título: Trabajo de Suficiencia Profesional para optar por el Título
Profesional de Ingeniero de Sistemas

Fecha: Octubre de 2024

RESUMEN

La empresa “Rapiditos” ofrece un sistema de depósitos que sirve para que plataformas de apuestas deportivas puedan tercerizar la generación de vouchers de pago, envío de notificaciones y acreditaciones de los depósitos que hacen sus clientes en sus cuentas. El sistema de depósitos que había antes, que procesaba los depósitos en lotes y secuencialmente, no era capaz de atender la alta carga, causando que el sistema realice sus procesos de manera lenta y por largos periodos, generando retrasos en la entrada de dinero para las plataformas de apuestas e insatisfacción en los clientes finales que no veían su dinero reflejado en sus cuentas para realizar sus apuestas. Para poder atender con éxito la creciente carga, realizamos un rediseño de la arquitectura de software del sistema de depósitos a una basada en eventos. Esta solución permitió poder atender exitosamente la alta carga durante La Copa América del 2024 mediante el escalado automático de los contenedores de las aplicaciones. También se agilizó el tiempo de acreditación de depósitos y envío de notificaciones.

Palabras clave: Arquitectura basada en eventos, Event Storming, Computación en la nube, Kubernetes, Escalamiento Automático.

MAJOR NATIONAL UNIVERSITY OF SAN MARCOS
FACULTY OF SYSTEMS ENGINEERING AND COMPUTER SCIENCE
PROFESSIONAL SCHOOL OF SYSTEMS ENGINEERING

**Redesign of software architecture to handle the high deposit load in a payment
method company**

Author: Bustamante Falcón, Jose Carlos

Advisor: Cortez Vásquez, Augusto Parcemon

Title: Professional Sufficiency Work to opt for the Professional Title of
Systems Engineer

Date: October 2024

ABSTRACT

The company “Rapiditos” offers a deposit system that allows sports betting platforms to outsource the generation of payment vouchers, sending notifications and crediting deposits made by their clients in their accounts. The previous deposit system, which processed deposits in batches and sequentially, was not able to handle the high load, causing the system to carry out its processes slowly and for long periods, generating delays in the entry of money for the betting platforms and dissatisfaction in end customers who did not see their money reflected in their accounts to place their bets. In order to successfully handle the increasing load, we redesigned the software architecture of the deposit system to one based on events. This solution allowed us to successfully handle the high load during the 2024 Copa América by automatically scaling the application containers. The time for crediting deposits and sending notifications was also expedited.

Keywords: Event driven architecture, Event Storming, Cloud Computing, Kubernetes, Autoscaling

TABLA DE CONTENIDO

RESUMEN.....	v
ABSTRACT	vi
INTRODUCCIÓN	1
CAPÍTULO I – TRAYECTORIA PROFESIONAL	4
CAPÍTULO II – CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA	9
2.1. Empresa – Actividad que realiza	9
2.2. Visión	9
2.3. Misión	9
2.4. Organización de la empresa	9
2.5. Área, cargo y funciones desempeñadas	10
2.6. Experiencia profesional realizada en la organización	11
CAPÍTULO III – ACTIVIDADES DESARROLLADAS	13
3.1. Situación problemática	13
3.1.1. Antecedentes.....	13
3.1.2. Definición del problema	15
3.2. Solución	17
3.2.1. Objetivos	17
3.2.2. Alcance.....	18
3.2.3. Etapas y metodología	18
3.2.4. Fundamentos utilizados	19
3.2.5. Implementación de las áreas, procesos, sistemas y buenas prácticas	71
1.5. Evaluación.....	112
1.5.1. Evaluación costo – beneficio	112
CAPÍTULO IV - REFLEXIÓN CRÍTICA DE LA EXPERIENCIA	116
CAPÍTULO V - CONCLUSIONES Y RECOMENDACIONES	117
5.1 Conclusiones	117
5.2. Recomendaciones	119
Fuentes de información	121
ANEXOS.....	125

Índice de Figuras

Figura 1 Organigrama de la empresa	10
Figura 2 Organigrama del área de Tecnología e Informática.....	11
Figura 3 Árbol de problemas	16
Figura 4 Árbol de objetivos	17
Figura 5 Etapas del Trabajo	18
Figura 6 Componentes de Event Storming	23
Figura 7 Fase 1 de Event Storming	24
Figura 8 Fase 2 de Event Storming	24
Figura 9 Fase 3 de Event Storming	25
Figura 10 Fase 4 de Event Storming	25
Figura 11 Fase 5 de Event Storming.....	26
Figura 12 Fase 6 de Event Storming	27
Figura 13 Fase 7 de Event Storming	27
Figura 14 Fase 8 de Event Storming	28
Figura 15 Pasos para el diagnóstico del problema	71
Figura 16 Arquitectura C4 As-Is Nivel 1	72
Figura 17 Arquitectura C4 As-Is Nivel 2	73
Figura 18 Creación de vouchers AS-IS	76
Figura 19 Expiración de vouchers AS-IS	77
Figura 20 Pago de vouchers AS-IS.....	78
Figura 21 Reporte de vouchers AS-IS.....	79
Figura 22 Lluvia de ideas para solución.....	84
Figura 23 Pasos del diseño de solución del problema.....	86
Figura 24 Fase 1 de Event Storming Aplicada	87
Figura 25 Fase 2 de Event Storming Aplicada	88
Figura 26 Fase 3 de Event Storming Aplicada	89
Figura 27 Fase 4 de Event Storming Aplicada	90
Figura 28 Fase 5 de Event Storming Aplicada	90
Figura 29 Fase 6 de Event Storming Aplicada	91
Figura 30 Fase 8 de Event Storming Aplicada	92
Figura 31 Diagrama C4 To-Be Nivel 2	94
Figura 32 Generación de vouchers TO-BE	97
Figura 33 Notificaciones TO-BE	98
Figura 34 Pagos TO-BE.....	99
Figura 35 Acreditaciones TO-BE	100
Figura 36 Expiraciones TO-BE	101
Figura 37 Publicación de voucher creado en API	105
Figura 38 Publicación de voucher pagado en Web Service.....	105
Figura 39 Fragmento de código de Notifications.....	106
Figura 40 Fragmento de código de Accreditation	106
Figura 41 Recurso de escalado automático en Kubernetes para Accreditation	107
Figura 42 Fragmento del plan de pase a producción	108
Figura 43 Fragmento de recurso Azure Service Bus en Producción	109
Figura 44 Fragmento de recurso Azure de API en Producción	109
Figura 45 Fragmento de manifiesto de despliegue de Web Service.....	110
Figura 46 Fragmento de manifiesto de despliegue de Notifications	110

Figura 47 Fragmento de manifiesto de despliegue de Accreditation	111
Figura 48 Fragmento de manifiesto de despliegue de Expiration	111

Índice de Tablas

Tabla 1 Experiencia profesional.....	4
Tabla 2 Estudios de Pregrado	6
Tabla 3 Idiomas	6
Tabla 4 Certificaciones	7
Tabla 5 Habilidades técnicas	7
Tabla 6 Artículos científicos	8
Tabla 7 Datos de final de Liga de Campeones 2023-2024	14
Tabla 8 Tiempo máximo de operaciones durante final de Liga de Campeones 2023-2024.....	15
Tabla 9 Componentes As-Is	75
Tabla 10 Problemas por cada componente.....	80
Tabla 11 Tiempos de procesos de Post Service	81
Tabla 12 Criticidad de componentes.....	82
Tabla 13 Alternativas de solución	85
Tabla 14 Componentes To-Be	95
Tabla 15 Publicador y subscriptores por evento.....	102
Tabla 16 Alternativas de patrones de evento	103
Tabla 17 Estrategias de escalamiento.....	104
Tabla 18 Evaluación de tiempos de procesamiento de notificaciones	113
Tabla 19 Evaluación de tiempos de procesamiento de acreditaciones.....	114

INTRODUCCIÓN

En Perú más del 50 por ciento tiene una enorme afición por los deportes, en especial por el fútbol masculino¹. Esto ha llevado que haya una tendencia a que en el país estos aficionados también apuesten durante los eventos deportivos, siendo aprovechado por las empresas de apuestas deportivas, que se han modernizado y facilitaron las apuestas por medio de plataformas en línea como aplicaciones móviles y aplicaciones web sin tener que ir a establecimientos físicos.

En el año 2024 se estima que el mercado de apuestas deportivas alcance ingresos de hasta 169 millones de dólares, con una participación mayoritaria de jóvenes de entre 18 y 30 años. Esto debido a lo que se vio reflejado en el año 2023², en el cual, un 85% de los ciudadanos peruanos admitieron haber hecho al menos una apuesta en lo que llevaban del año.

En estas plataformas de apuestas cada cliente tiene una cuenta virtual, donde necesita recargar dinero para poder apostar en el deporte de su preferencia y donde también se le deposita el dinero de sus ganancias para que luego pueda solicitar retirarlo a la cuenta de banco de su preferencia.

“Rapiditos” es una de las empresas en el Perú que ofrece su servicio de sistema de depósitos, encargándose completamente de la generación de vouchers de pago, la integración con los bancos de los clientes para confirmar los pagos, acreditar en las cuentas virtuales los pagos confirmados y enviar notificaciones vía correo electrónico.

“Rapiditos” se ha posicionado como una de las más importantes opciones de medio de pago en plataformas de apuestas, apuntando a ser la primera en generar más tráfico entre sus competidores para el año 2025. Para el mantenimiento del sistema de depósitos, “Rapiditos” subcontrata su equipo de TI con la empresa “FakeIt”.

El diseño del sistema de depósitos de “Rapiditos” influye directamente en la rapidez con la que sus procesos se llevan a cabo e impactan directamente en la satisfacción del cliente,

¹ Según encuesta realizada por IPSOS Perú en el 2017

² Según encuesta realizada por Statista en el 2024

por lo cual en el presente trabajo se abordara el problema de la capacidad de atender altas demandas.

La estructura del presente trabajo está detallada a continuación:

1. En el CAPÍTULO I, se expone cronológicamente la trayectoria profesional del aspirante al título de Ingeniero de Sistemas, demostrando la experiencia conseguida durante los años.
2. En el CAPÍTULO II, se detalla el contexto en el cual se desarrolló la experiencia del presente trabajo, incluyendo todo lo relacionado a la empresa “Rapiditos” y el ejercicio profesional en la organización.
3. En el CAPÍTULO III, se detalla las actividades desarrolladas, definiendo el problema y cómo se solucionó, la metodología y la evaluación de resultados.
4. En el CAPÍTULO IV, se expone la reflexión crítica de la experiencia y cuál fue el aporte del autor y necesidades atendidas.
5. En el CAPÍTULO V, se detallan las conclusiones del presente trabajo y las recomendaciones basadas en la experiencia.

Finalmente se enumeran las referencias usadas en el presente trabajo con sus respectivos autores y año de publicación.

CAPÍTULO I – TRAYECTORIA PROFESIONAL

El autor del presente documento es bachiller en ingeniería de sistemas de la Universidad Nacional Mayor de San Marcos, con poco más de 6 años de experiencia en la industria del software, trabajando en la modernización de sistemas de software, análisis de requerimientos, diseño de soluciones, gestión de base de datos, gestión de infraestructura de aplicaciones y desarrollo en la nube. Dentro de todos los años de experiencia ha aprendido diferentes tecnologías, siempre estando a la vanguardia con los estándares de software y aplicando lo aprendido, tanto en la formación profesional como en la experiencia laboral, de la mejor manera en el centro de labores. Le gusta ejercitar su creatividad y trabajo en equipo para diseñar y construir soluciones de software, dedicado a la atención de requerimientos en el mejor tiempo posible y ofrecer soluciones de calidad y bien cimentadas.

A continuación, se expone la trayectoria profesional del autor del presente trabajo de suficiencia profesional:

Tabla 1

Experiencia profesional

Experiencia laboral	Periodo
Sercosta	
Consultor de TI	
Funciones:	
1. Gestión de todo el proceso de desarrollo de una aplicación de escritorio para gestión de Kardex.	2017-2018
2. Despliegue de la aplicación en el servidor de la empresa.	
3. Mantenimiento de la aplicación de gestión de Kardex, otorgando accesos y exportando reportes.	
Luz del Sur	
Practicante de sistemas	2018
Funciones:	

-
1. Revisión de soluciones de migración de aplicaciones legacy realizadas por empresas tercerizadas.
 2. Desarrollo de aplicaciones frontend y backend para los sistemas comerciales.

SES

Programador .Net Full Stack

Funciones:

1. Desarrollo de aplicaciones frontend y backend para los sistemas comerciales para el cliente Luz del Sur como consultora. 2018-2019
2. Desarrollo de aplicaciones del módulo de notificaciones electrónicas.
3. Mantenimiento y migración de aplicaciones heredadas de Visual Basic 6 a .NET 4.

Stefanini

Programador .Net Full Stack

Funciones:

1. Desarrollo de aplicaciones frontend y backend para los sistemas comerciales para el cliente Luz del Sur como tercero.
2. Desarrollo de aplicaciones del módulo de notificaciones electrónicas.
3. Desarrollo de aplicaciones del módulo de envío de correos electrónicos. 2020-2023
4. Desarrollo de aplicaciones del módulo de reclamos.
5. Desarrollo de aplicaciones del módulo de medidores.
6. Mantenimiento y migración de aplicaciones heredadas de Visual Basic 6 a .NET 4.

“FakeIt”

.NET Backend Developer

2023-Actualidad

Funciones:

1. Migración de las aplicaciones de NET Framework a NET Core.
 2. Desarrollo de aplicaciones del módulo de depósitos.
 3. Desarrollo de aplicaciones del módulo de retiros.
 4. Desarrollo de aplicaciones del módulo de KYC.
 5. Rediseñar la arquitectura de software del módulo de retiros.
 6. Rediseñar la arquitectura de software del módulo de depósitos.
 7. Mantener el clúster de Kubernetes.
-

Nota. Elaboración propia

Tabla 2

Estudios de Pregrado

Estudios de Pregrado	Periodo
Bachiller en Ingeniería de Sistemas Universidad Nacional Mayor de San Marcos	2015-2020

Nota. Elaboración propia

Tabla 3

Idiomas

Idiomas	Periodo
Inglés – Avanzado ICPNA	2011-2014

Nota. Elaboración propia

Tabla 4*Certificaciones*

Certificaciones	Periodo
Curso de Machine Learning	2021
Sistemas de Gestión de la Seguridad de la Información	2021
Entrenamiento en scrum	2021
Curso Ingeniería de requerimientos	2021
Emprendedurismo para los negocios	2022
Inglés TOEFL	2014

Nota. Elaboración propia

Tabla 5*Habilidades técnicas*

Habilidades Técnicas	Herramientas
Lenguajes de programación	Visual Basic 6.0, Python, C++, C#, Java, Javascript
Bases de datos	Informix, Oracle, SQL Server, MySQL, PostgreSQL, MongoDB, CosmosDB
Infraestructura de aplicaciones	VMs, Kubernetes, Azure Functions, Azure App Services
Sistemas Operativos	Linux, Ubuntu, Windows
Otros	Git, Docker, Kubernetes, Azure, Bitbucket, React, REST, Linux, RabbitMQ, Open Telemetry

Nota. Elaboración propia

Tabla 6

Artículos científicos

Artículo	Publicación
Real time facial expression recognition system based on deep learning	Publicado el 1 setiembre del 2019 en la revista International Journal of Recent Technology and Engineering

Nota. Elaboración propia

CAPÍTULO II – CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA

2.1. Empresa – Actividad que realiza

“Rapiditos” es una empresa con oficinas en Lima (Perú) y Sofía (Bulgaria) creada en el 2017 que se dedica a crear soluciones de medios de pago que se adaptan a las necesidades de cada cliente empresarial, usando nuevas tecnologías que ayudan con el trabajo operativo, usando procesos automatizados y potenciando resultados.

Ofrece de manera general dos servicios:

1. Depósitos, que es un medio de pagos premium para comercios electrónicos en el Perú.
2. Retiros, que es una plataforma específica para la gestión y monitorea de transferencias bancarias nacionales en todo el Perú.

2.2. Visión

Mantenemos siempre a la vanguardia en los desarrollos tecnológicos del sector de pagos y automatizar procesos que faciliten la gestión transaccional del cliente, a través de un sistema integral de pagos que abarque las necesidades de los clientes.

2.3. Misión

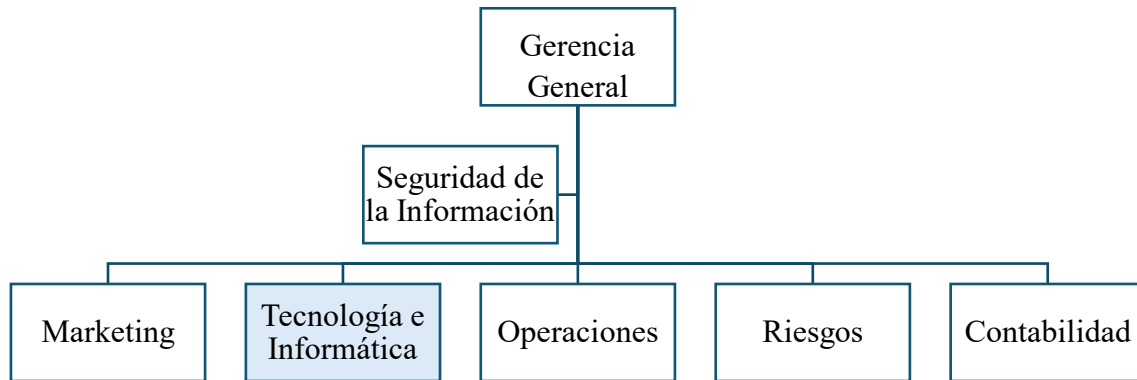
Ser una Fintech de pagos reconocida a nivel de América Latina por brindar una tecnología de primera calidad con máxima seguridad y velocidad, que permita a nuestros clientes obtener ventajas competitivas en el mercado.

2.4. Organización de la empresa

A continuación, se expone la estructura organizacional de la empresa actualizada en el 2024:

Figura 1

Organigrama de la empresa



Nota. Datos tomados de “Rapiditos” 2024.

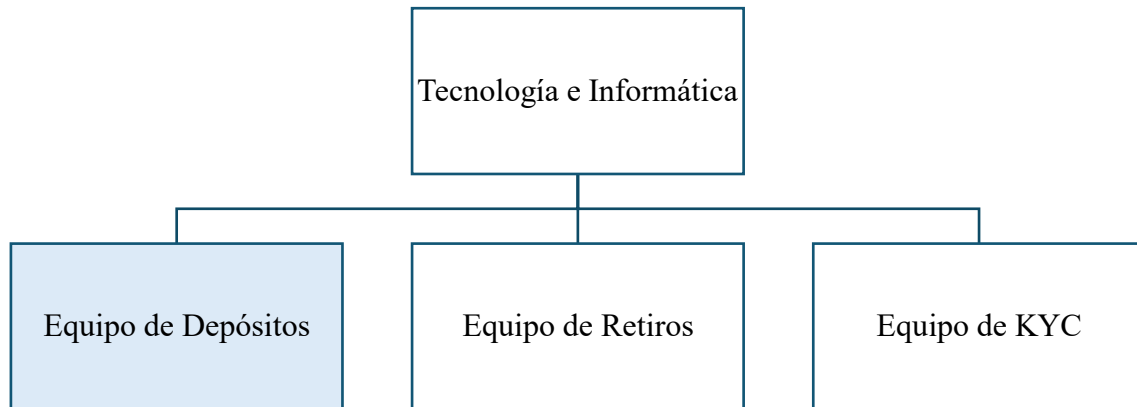
Fuente: Elaboración propia.

2.5. Área, cargo y funciones desempeñadas

El presente autor del documento se desempeña en el área de Tecnología e Informática que esta tercerizado con la empresa “FakeIt”, ocupando el puesto de .NET Backend Developer en el equipo de Depósitos.

Figura 2

Organigrama del área de Tecnología e Informática



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

2.6. Experiencia profesional realizada en la organización

Durante el tiempo que el presente autor trabajó en “FakeIt” se realizaron las siguientes funciones:

1. Migración de aplicaciones NET Framework a NET Core, esto se realizó para que las aplicaciones puedan ser desplegadas en contenedores Linux.
2. Migración de infraestructura de las aplicaciones de una máquina virtual a Kubernetes.
3. Elaboración de plan de separación de ambientes para cumplir con requerimientos de ISO 270001³.
4. Despliegue de recursos de Kubernetes para la correcta ejecución de aplicaciones en contenedores dentro de los clústeres de cada ambiente.
5. Migración de base de datos de máquinas virtuales al servicio en la nube Azure SQL Server Managed Instance.

³ **ISO 270001:** Es un estándar internacional que establece los requisitos para la implementación, mantenimiento y mejora continua de un Sistema De Gestión de la Seguridad de la Información (SGSI).

6. Mantenimiento de repositorio GitOps para gestión del despliegue de recursos de Kubernetes mediante ArgoCD por cada ambiente.
7. Mantenimiento de repositorio Infrastructure As Code para gestión de recursos de suscripción de Azure por cada ambiente.
8. Migración de procesador de sistema de retiros a un sistema de eventos en tiempo real.
9. Rediseño de arquitectura de software del sistema de depósitos.
10. Implementación de TDD como metodología de desarrollo.
11. Desarrollo de módulo Open Banking para integración con pagos instantáneos de los bancos.
12. Desarrollo de aplicaciones del módulo de retiros.
13. Desarrollo de aplicaciones del módulo de depósitos.
14. Desarrollo de aplicaciones del módulo de KYC.

CAPÍTULO III – ACTIVIDADES DESARROLLADAS

3.1. Situación problemática

3.1.1. Antecedentes

El sistema de depósitos se encarga de la gestión de los vouchers de pago que son entidades con los siguientes atributos:

1. Número de voucher
2. Fecha de creación
3. Identificador público
4. Fecha de expiración
5. Moneda del pago
6. Monto del pago
7. Correo del cliente
8. Estado del voucher
9. Banco de pago
10. Canal de pago

El sistema de depósitos también cuenta con cuatro procesos para realizar operaciones en los vouchers de pago:

1. Creación de voucher:

Es el proceso que se encarga de generar el voucher de pago y mostrar las instrucciones de pago tanto por medio de una página HTML como por correo electrónico (v.

2. Expiración de voucher:

Es el proceso que se encarga de monitorear los vouchers de pago que caducaron y expirarlos para que no puedan ser pagados.

3. Pago de vouchers:

Es el proceso que se encarga de recibir los reportes de pagos desde ASBANC⁴ y marcar los vouchers como pagados. Luego, acreditar los pagos en las cuentas de los clientes y notificarlos por correo electrónico.

⁴ **Asociación de Bancos del Perú - ASBANC**, organización que representa a las instituciones financieras privadas del Perú.

4. Reporte de vouchers:

Es el proceso que se encarga de diariamente enviar un reporte de los vouchers y sus estados a las casas de apuestas.

Estos procesos también registran entidades llamadas operaciones que tienen los siguientes atributos:

1. Número de voucher
2. Evento realizado
3. Fecha del evento

En un flujo normal de un cliente es solicitar un depósito, que crea un voucher de pago (ver **Anexo A**) y obtener las instrucciones de pago tanto en la web (ver **Anexo B**) como por correo (ver **Anexo C**), luego con ese número de voucher procede a pagarlo en el canal del banco de su preferencia, por ejemplo, por la aplicación móvil de su banco (ver **Anexo D**). Finalmente, ASBANC reporta el pago a nuestro sistema para que se notifique su depósito (ver **Anexo E**) y se acredite en su cuenta virtual (ver **Anexo F**).

En el año 2024 durante la final de la Liga de Campeones temporada 2023-2024, el área de Tecnología e Informática pudo detectar que hubo un incremento en la cantidad de creación de vouchers por segundo, llegando a un total de aproximadamente 1200 vouchers creados por hora y 510 vouchers pagados por hora según se puede observar en la **Tabla 7**.

Tabla 7

Datos de final de Liga de Campeones 2023-2024

Tipo de operación	Cantidad de vouchers en el día	Promedio de cantidad de vouchers por hora
Vouchers creados	28443	1185.125
Vouchers pagados	12237	509.87

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Base de datos de Depósitos.

3.1.2. Definición del problema

El incremento de la creación de vouchers causó que las notificaciones por correo con las instrucciones de pagos, las acreditaciones en las cuentas de los clientes y las notificaciones por correo con los pagos confirmados se retrasaran debido a que se formaron grandes colas de vouchers a ser procesados por el sistema.

Como se puede observar en la **Tabla 8** había notificaciones de instrucciones de pago que llegaban hasta más de una hora y media después de crear el voucher o que luego que el cliente pagara no se viera su dinero reflejado hasta aproximadamente una hora después de hacer el pago en el banco.

Tabla 8

Tiempo máximo de operaciones durante final de Liga de Campeones 2023-2024

Tipo de operación	Máximo tiempo en segundos	Máximo tiempo en horas
Envío de notificaciones	6266	1.74
Acreditación de pagos	3245	0.90

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Base de datos de Depósitos y registros con fecha 01/06/2024.

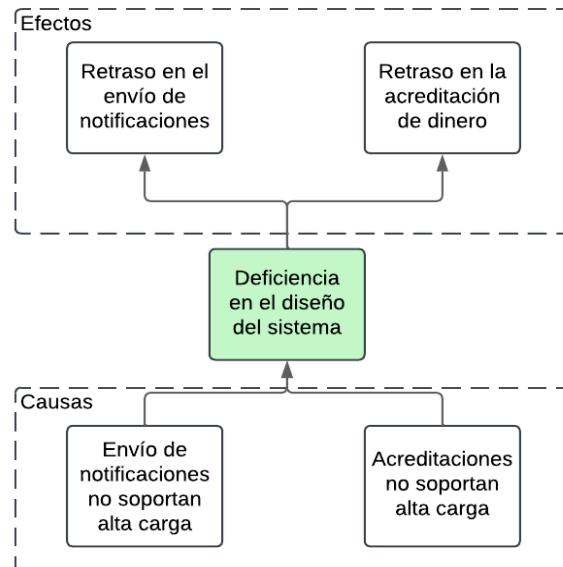
Esto causó que los clientes se sientan insatisfechos ya que al haber un retraso en las notificaciones de instrucciones de pago no podrían pagar sus vouchers, y al haber un retraso en las acreditaciones y notificaciones de confirmación de pago no verían sus pagos reflejado en sus cuentas.

La preocupación de la empresa “Rapiditos” era debido a que durante el próximo evento de La Copa América 2024 se estimaba que la cantidad de depósitos fuera aún mayor que lo observado en la final de La Liga de Campeones y por un tiempo más prolongado, por lo cual se debía migrar el sistema para que los problemas anteriormente mencionados no ocurran nuevamente.

Para hallar las causas del problema se hizo un análisis de la problemática estableciendo una relación causa-efecto entre los problemas como se muestra en la **Figura 3**. A partir de este análisis se elaboró un árbol de problemas para determinar el problema general:

Figura 3

Árbol de problemas



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

Problema general

Deficiencia en el diseño del sistema de depósitos cuando hay alta carga.

Problemas específicos:

1. Envío de notificaciones no soportan alta carga.
2. Acreditaciones no soportan alta carga.

Efectos:

1. Retraso en el envío de notificaciones cuando hay alta carga.
2. Retraso en la acreditación de dinero cuando hay alta carga.

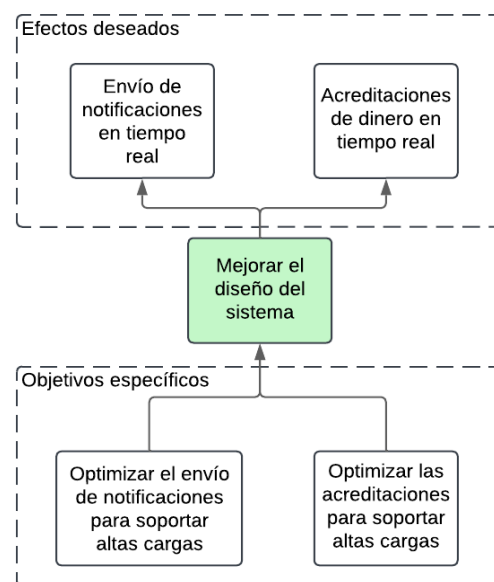
3.2. Solución

3.2.1. Objetivos

Para dar solución a los problemas detallados anteriormente se definieron los siguientes objetivos:

Figura 4

Árbol de objetivos



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

Objetivo general

Mejorar el diseño del sistema de depósitos para atender la alta carga durante La Copa América 2024.

Objetivos específicos

1. Optimizar el envío de notificaciones para soportar altas cargas.
2. Optimizar las acreditaciones para soportar altas cargas.

Efectos deseados

1. Envío de notificaciones en tiempo real.
2. Acreditaciones en tiempo real.

3.2.2. Alcance

La solución se estimó para desarrollarse en tres semanas con el inicio de La Copa América como fecha límite de entrega y se limitó a migrar solo los componentes con más alta prioridad.

La solución fue desarrollada por el equipo de depósitos, conformado por 4 desarrolladores, tanto en la oficina como en trabajo remoto.

3.2.3. Etapas y metodología

Con el propósito de lograr los objetivos y lograr la solución de mayor efectividad, el equipo de depósitos organizó el trabajo en cuatro etapas, como se muestra en la **Figura 5**. Primero se diagnosticó el problema, esta etapa fue muy importante, ya que permitió profundizar en los problemas y establecer órdenes de criticidad o prioridades, como así también que problemas son causa de otros y cuales consecuencia.

Luego a partir de ese diagnóstico se diseñó la solución, se implementó y finalmente fue desplegada en producción.

Figura 5

Etapas del Trabajo



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Elaboración propia.

3.2.4. Fundamentos utilizados

A. Event Storming

Es una metodología de modelado colaborativo para el diseño de software que se centra en la identificación y análisis de eventos dentro de un sistema. Creada por Alberto Brandolini en 2013, esta metodología se caracteriza por su enfoque visual y participativo, lo que permite a los equipos de desarrollo de software, analistas de negocio y partes interesadas del sistema crear conjuntamente una representación clara y comprensible de los eventos clave que ocurren en el dominio de negocio (Brandolini, 2013). El principal objetivo de Event Storming es lograr un entendimiento común y compartido del comportamiento del sistema a través de la identificación de eventos relevantes, lo que facilita la identificación de problemas, oportunidades y posibles soluciones desde una perspectiva técnica y de negocio

La popularidad de Event Storming radica en su capacidad para descomponer la complejidad de los sistemas mediante la visualización de los eventos. En lugar de centrarse en diagramas técnicos o UML, Event Storming promueve la colaboración entre personas con diferentes roles y niveles de conocimiento, utilizando tableros y notas adhesivas para crear una vista integral de los procesos de negocio. Esto facilita el descubrimiento de las interacciones críticas del sistema y de cómo los eventos fluyen a través de él, lo que es especialmente útil en proyectos de gran envergadura o donde los sistemas existentes necesitan ser rediseñados o migrados.

Características

Algunas de las características de la metodología son:

1. ***Enfoque en eventos***: Los eventos son acciones o cambios de estado significativos que ocurren en el sistema y son de interés para el negocio.
2. ***Colaboración interdisciplinaria***: La metodología fomenta la participación de desarrolladores, arquitectos de software, analistas de negocio y usuarios finales.
3. ***Visualización dinámica***: Se utilizan elementos físicos, como pizarras y notas adhesivas, para representar los eventos y los flujos del sistema, facilitando la comprensión y la iteración.

4. **Flexibilidad:** Event Storming no sigue reglas estrictas en cuanto a diagramación, lo que permite que el equipo pueda experimentar y ajustar el enfoque según sus necesidades específicas.

Niveles

Existen varios niveles para abordar la metodología de Event Storming, cada uno diseñado para diferentes niveles de detalle y objetivos:

1. **Big Picture:** Es la forma más amplia de Event Storming, que se utiliza para explorar el flujo de eventos en todo un dominio empresarial sin entrar en detalles técnicos.
2. **Process Level:** Focalizado en el flujo de un proceso o servicio en particular. Aquí se pueden analizar las interacciones entre eventos, comandos y reglas del negocio en detalle.
3. **Design Level:** Un enfoque más técnico, donde se diseña una solución tecnológica basada en los eventos descubiertos en los niveles anteriores.

Componentes de Event Storming

1. **Eventos del Dominio:** Los eventos son el componente central de la metodología Event Storming. Un evento del dominio es una acción o cambio significativo que ocurre dentro del sistema o en el dominio del negocio. Estos eventos son relevantes porque representan situaciones que el sistema debe gestionar y responder.
 - **Ejemplos:** "Pedido realizado", "Pago completado", "Producto añadido al carrito".
 - **Color:** Los eventos suelen representarse con notas adhesivas de color naranja en la pizarra o tablero.

Los eventos del dominio permiten modelar el comportamiento del sistema desde una perspectiva de negocio, ya que reflejan momentos clave en el flujo de trabajo. A medida que se identifican y ordenan cronológicamente, ofrecen una visión general de cómo interactúan los diferentes componentes del sistema.

2. **Comandos:** Un comando es una acción que solicita que algo suceda en el sistema, es decir, que un evento sea disparado. Los comandos son acciones que los usuarios u otros sistemas externos ejecutan para provocar cambios en el sistema.

- **Ejemplos:** "Crear un pedido", "Procesar un pago", "Enviar notificación".
- **Color:** Los comandos se suelen representar con notas adhesivas de color azul.

Los comandos son fundamentales para identificar las interacciones entre los actores y el sistema, ya que permiten entender quién o qué está generando los eventos. Además, ayudan a definir qué operaciones deben ser soportadas por el sistema en respuesta a los eventos del dominio.

3. **Agregados:** Un agregado es un conjunto de entidades relacionadas que deben tratarse como una única unidad dentro del sistema. Los agregados son un concepto proveniente del diseño orientado a dominios (DDD, Domain-Driven Design) y se utilizan para organizar los datos y comportamientos de manera coherente.

- **Ejemplos:** Un "Pedido" puede ser un agregado que incluye las entidades "Producto", "Cliente", "Dirección de envío".
- **Color:** Los agregados se suelen representar con notas adhesivas de color amarillo oscuro.

Los agregados sirven para encapsular el estado y la lógica relacionada con un grupo de entidades, y definen los límites de consistencia en el sistema. Cada agregado maneja comandos y genera eventos dentro de su contexto, lo que permite una organización más clara y mantenible del código y los datos.

4. **Actores:** Los actores son las personas o sistemas que interactúan con el sistema y generan comandos que luego provocan eventos. Un actor puede ser un usuario, un sistema externo o cualquier entidad que interfiera con el sistema desde fuera.

- **Ejemplos:** "Cliente", "Administrador del sistema", "Sistema de pagos".

- **Color:** Los actores se representan con notas adhesivas de color amarillo claro.

La identificación de actores permite entender quién está generando los comandos y qué partes del sistema están involucradas en la ejecución de procesos clave. Los actores son fundamentales para modelar las interacciones humanas o de sistemas externos con el sistema interno.

5. **Proyecciones o Vistas:** Las proyecciones (también conocidas como vistas) son vistas del sistema que muestran información derivada de los eventos que han ocurrido. Estas proyecciones permiten visualizar el estado del sistema de manera amigable para los usuarios o sistemas externos, proporcionando datos procesados y actualizados.

- **Ejemplos:** "Lista de productos en stock", "Resumen de pedidos realizados".
- **Color:** Las proyecciones suelen representarse con notas adhesivas de color verde.

Las proyecciones permiten una visión materializada del estado del sistema que se genera a partir de eventos previos. Son cruciales en arquitecturas basadas en eventos, donde los datos se procesan de manera asíncrona, y el estado del sistema se reconstruye a partir de los eventos emitidos.

6. **Políticas:** Las políticas son reglas de negocio que definen cómo deben manejarse los eventos. Estas reglas dictan las acciones que deben ocurrir cuando un determinado evento tiene lugar. Las políticas permiten automatizar la respuesta del sistema a eventos del dominio de forma eficiente.

- **Ejemplos:** "Si el pago es aprobado, enviar notificación al cliente", "Si el pedido es mayor a cierta cantidad, aplicar descuento".
- **Color:** Las políticas se suelen representar con notas adhesivas de color morado.

Las políticas se usan para definir comportamientos automáticos en respuesta a eventos, lo que es útil para gestionar flujos de trabajo complejos en los que se deben ejecutar ciertas acciones sin intervención manual.

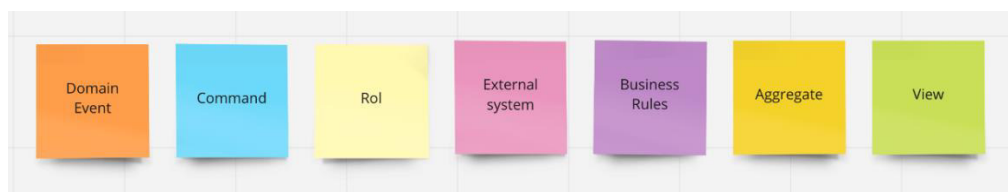
7. **Sistemas externos:** Un sistema externo es una entidad o servicio fuera del dominio de la aplicación que recibe o envía información, y puede interactuar con el sistema en base a ciertos eventos. Estos sistemas pueden ser otras aplicaciones, servicios de terceros, o componentes independientes que proporcionan o consumen datos del sistema principal.

- **Ejemplos:** "Sistema de pago de terceros, servicios de mensajería, API de terceros".
- **Color:** Los sistemas externos se representan con notas adhesivas de color rosado.

La inclusión de sistemas externos es fundamental en la metodología Event Storming, ya que refleja la interacción del sistema modelado con el mundo exterior. Estos componentes destacan la dependencia de la aplicación con servicios de terceros, la gestión de integraciones y la necesidad de coordinar flujos de trabajo que abarcan múltiples sistemas.

Figura 6

Componentes de Event Storming



Nota. Imagen tomada de libro.

Fuente: Brandolini, A. (2013). *Introducing Event Storming: An unconventional approach to domain driven design*.

Fases del Event Storming

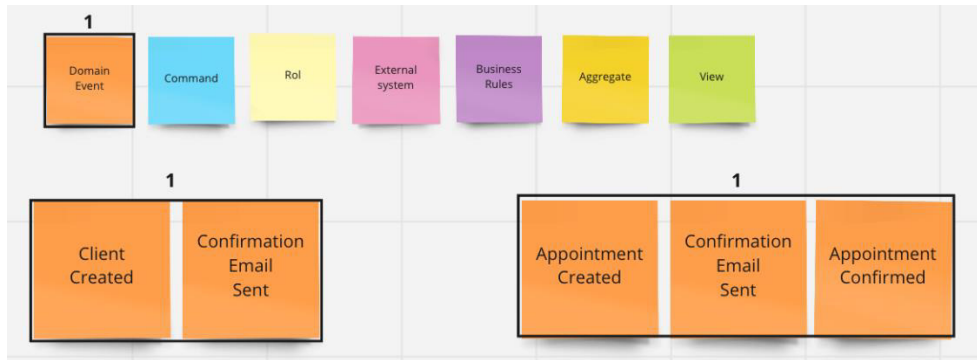
La metodología Event Storming se compone de varias fases que guían el proceso de modelado y descubrimiento de eventos. Estas fases pueden variar ligeramente dependiendo del enfoque que se tome (Big Picture, Process Level o Design Level), pero en general se puede considerar el siguiente flujo de trabajo:

1. **Descubrimiento de eventos clave:** El primer paso es identificar los eventos más importantes que ocurren dentro del sistema o el dominio de negocio (ver **Figura 7**). Estos eventos deben ser acciones significativas que cambien el estado del sistema, y se representan típicamente como notas

adhesivas de color naranja pegadas en una pizarra o tablero. En esta fase, no se entra en detalles técnicos, sino que se busca obtener una visión general del sistema desde la perspectiva del negocio.

Figura 7

Fase 1 de Event Storming



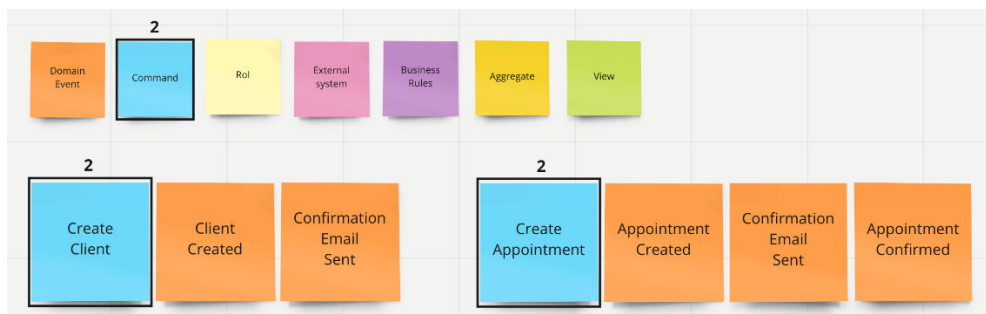
Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

2. **Identificación de comandos:** Después de mapear los eventos, se identifican los comandos que los activan (ver **Figura 8**). Los comandos son acciones que los usuarios o sistemas externos ejecutan para provocar cambios en el sistema.

Figura 8

Fase 2 de Event Storming



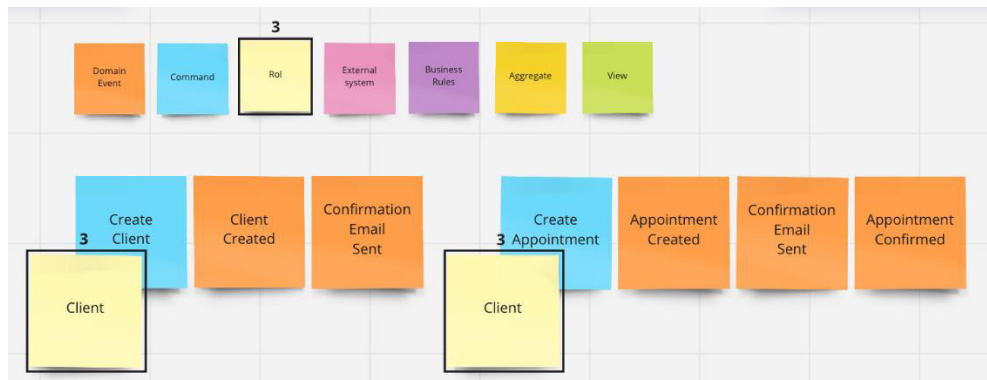
Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

3. **Identificación de actores:** Después de mapear los comandos que provocan los eventos, se identifican los actores (personas o sistemas) que accionan esos comandos (ver **Figura 9**).

Figura 9

Fase 3 de Event Storming



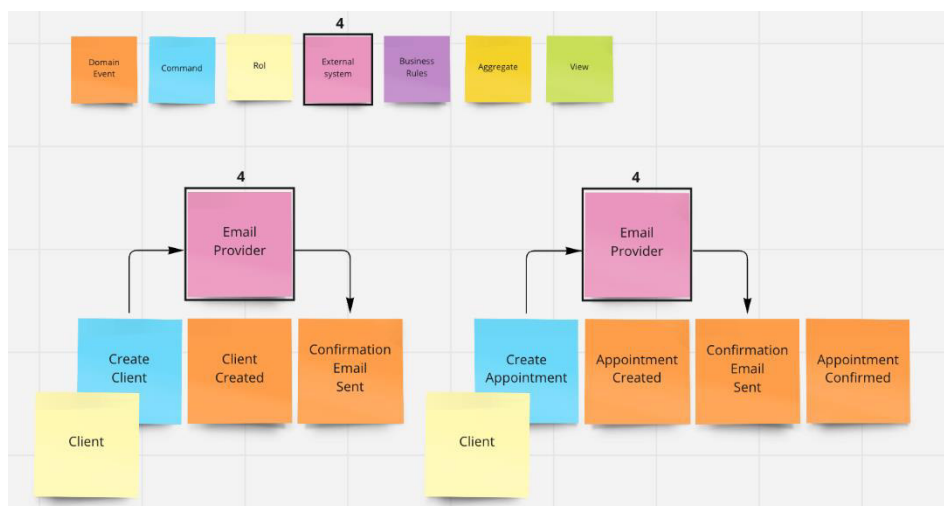
Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

4. **Identificación de sistemas externos:** En esta etapa se identifica los sistemas externos, que están fuera del dominio, pero pueden recibir o accionar un comando (ver **Figura 10**).

Figura 10

Fase 4 de Event Storming



Nota. Imagen tomada de una web.

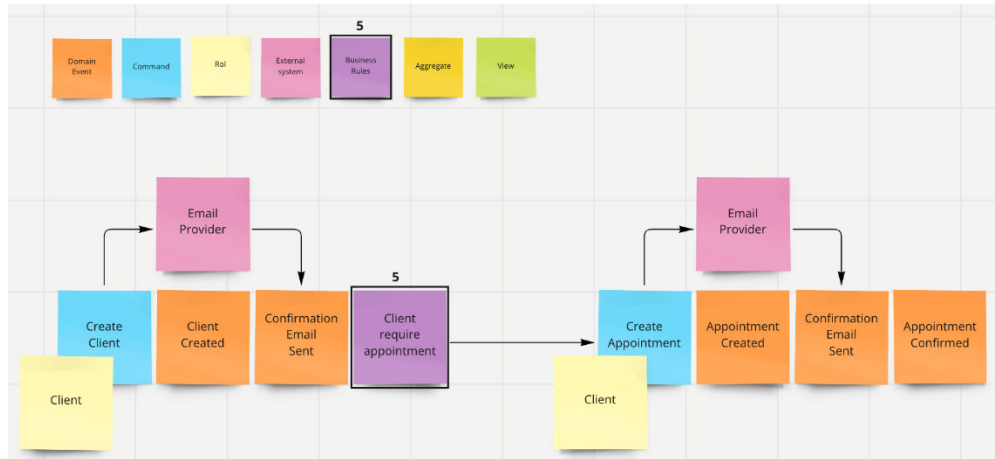
Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

5. **Identificación de políticas:** Las políticas son reglas de negocio que dictan cómo deben manejarse los eventos, mientras que las proyecciones son representaciones de los datos procesados y almacenados que pueden ser consultados por los usuarios (ver **Figura 11**). Aquí es donde se definen las

reglas y las vistas de datos necesarias para soportar los procesos de negocio.

Figura 11

Fase 5 de Event Storming



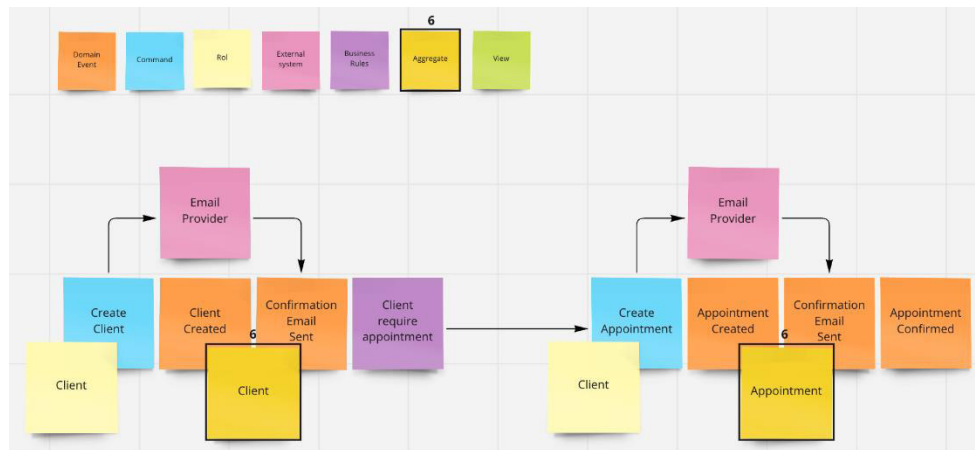
Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

6. **Determinación de agregados y entidades:** Los agregados son clústeres lógicos de datos y comportamientos que representan unidades coherentes en el sistema. En esta fase, se identifican los agregados o entidades que responden a los comandos y generan eventos (ver **Figura 12**). Esta fase suele implicar una mayor profundidad técnica y está relacionada con conceptos del diseño orientado a dominios (DDD, Domain-Driven Design).

Figura 12

Fase 6 de Event Storming



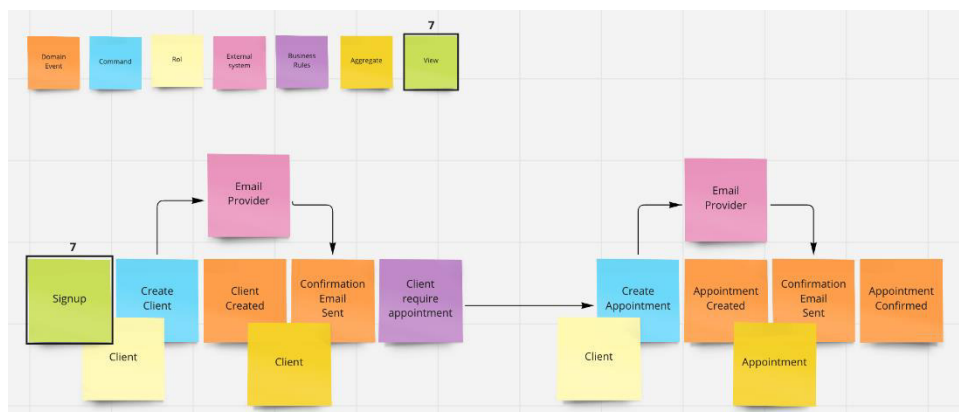
Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

7. **Determinación de proyecciones:** Se puede añadir las proyecciones o vistas involucradas para reforzar la idea de interacción (ver **Figura 13**).

Figura 13

Fase 7 de Event Storming



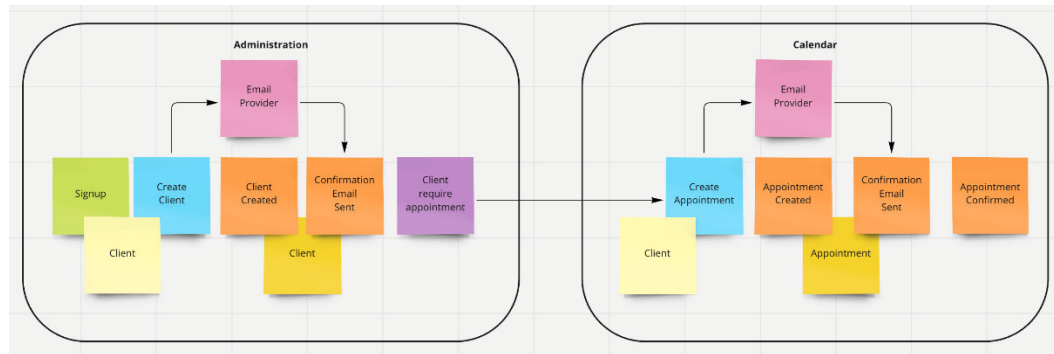
Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

8. **Organización de contextos:** El objetivo es definir límites entre ellos. En resumidas cuentas, es agrupar las responsabilidades de los agregados (ver **Figura 14**). Podemos ir añadiendo más y más agregados en base a las necesidades del negocio y cada uno estará clasificado en un contexto.

Figura 14

Fase 8 de Event Storming



Nota. Imagen tomada de una web.

Fuente: <https://adictosaltrabajo.com/2022/09/27/event-storming/>.

B. Arquitectura de Software

Al igual que ocurre en construcción civil, la arquitectura de software es como un plano de construcción: si no se realiza una planificación adecuada, el edificio puede derrumbarse. Lo mismo ocurre con el desarrollo de aplicaciones, sin una arquitectura de software de calidad, el programa puede presentar errores a largo plazo y generar retrasos en la operatividad. La arquitectura de software es el conjunto de decisiones estructurales fundamentales sobre el diseño y la organización de un sistema de software. Según Bass et al. (2021), esta disciplina define la estructura general y el comportamiento de un sistema al especificar sus componentes, las interacciones entre ellos y las propiedades emergentes que resultan de dichas interacciones. Además, como mencionan Rozanski y Woods (2020), una arquitectura bien diseñada proporciona un marco sólido para el desarrollo y evolución del sistema, permitiendo que se mantenga un equilibrio entre las necesidades técnicas y los objetivos de negocio, así como entre otros factores críticos como el rendimiento, la escalabilidad y la seguridad.

Importancia de la Arquitectura de Software

La arquitectura de software es fundamental para el desarrollo eficiente de sistemas complejos, ya que define la estructura sobre la cual se construyen, implementan y mantienen dichos sistemas. Según Bass, Clements y Kazman (2021), una arquitectura bien planificada puede reducir los riesgos a lo largo del ciclo de vida del software y ayudar a los equipos a responder de manera ágil ante cambios en las necesidades del negocio. Como afirman Rozanski y Woods (2020), una arquitectura sólida no solo favorece la toma de decisiones técnicas, sino que también promueve cualidades esenciales como la resiliencia y la escalabilidad, siempre manteniendo el enfoque en los objetivos del negocio.

Principios de Arquitectura de Software

El diseño de una arquitectura de software efectiva requiere la aplicación de principios clave que aseguren la calidad y mantenibilidad del sistema como la modularidad, el bajo acoplamiento, la alta cohesión y la separación de responsabilidades. Rozanski y Woods (2020) destacan que la modularidad permite dividir el sistema en componentes manejables que se pueden desarrollar y mantener de manera independiente. Por su parte, Taylor et al. (2020) señalan que

el bajo acoplamiento y la alta cohesión son esenciales para que los módulos sean independientes y tengan una función clara, lo cual facilita su reutilización y mantenimiento. Asimismo, Richards y Ford (2020) enfatizan que la separación de responsabilidades mejora la escalabilidad y mantenibilidad del sistema al asignar tareas específicas a componentes especializados.

Estilos y Patrones de Arquitectura

Existen varios estilos para diseñar la arquitectura de software:

1. ***Arquitectura en capas***: La arquitectura en capas es una estructura organizativa de software que separa el sistema en distintos niveles de abstracción, asignando a cada capa una responsabilidad específica y limitando la comunicación a las capas adyacentes. Como explican Bass et al. (2021), este estilo arquitectónico simplifica el mantenimiento al dividir la lógica de negocio, la lógica de presentación y la gestión de datos en capas independientes. Además, según Richards y Ford (2020), esta segmentación facilita la escalabilidad y la flexibilidad del sistema, permitiendo a los equipos realizar cambios en una capa sin afectar directamente a las demás.
2. ***Monolitos***: La arquitectura monolítica es un estilo de diseño en el cual todos los componentes de una aplicación están integrados en una única unidad cohesiva. Según Richards y Ford (2020), este enfoque facilita la gestión y el despliegue, ya que todo el sistema se implementa como un solo bloque, lo que reduce la complejidad en la administración de sus componentes. Además, como mencionan Bass, Clements y Kazman (2021), las arquitecturas monolíticas son adecuadas para sistemas que requieren una comunicación interna rápida y coherente, siendo efectivas en términos de rendimiento en contextos de baja complejidad y escalabilidad moderada.
3. ***Microservicios***: La arquitectura de microservicios es un enfoque que permite dividir un sistema en servicios independientes, cada uno con responsabilidades específicas, lo que facilita su desarrollo, despliegue y escalabilidad autónoma. Como explica Newman (2021), los microservicios brindan una flexibilidad sobresaliente para la construcción de sistemas distribuidos al descomponer las funciones del sistema en

servicios más pequeños e independientes. Sin embargo, esta arquitectura introduce desafíos adicionales, como la gestión de la comunicación entre servicios, el despliegue continuo y la consistencia eventual en bases de datos distribuidas, lo cual exige una planificación cuidadosa para mantener la cohesión del sistema. Por su parte, Richards y Ford (2020) advierten que la adopción de microservicios requiere herramientas avanzadas para monitoreo y orquestación, esenciales para gestionar la complejidad inherente a los sistemas distribuidos.

4. ***Arquitectura basada en eventos (Event-Driven Architecture):*** La arquitectura basada en eventos (EDA) es una estructura que permite diseñar sistemas escalables, asíncronos y reactivos, lo que la convierte en una opción ideal para aplicaciones que requieren capacidad de respuesta en tiempo real. Como explican Richards y Ford (2020), en la EDA, los componentes del sistema se comunican a través de eventos, permitiendo que los servicios reaccionen sin establecer una comunicación directa entre ellos. Además, Bass et al. (2021) señalan que este enfoque de bajo acoplamiento fomenta la escalabilidad y adaptabilidad, ya que cada componente puede operar de manera autónoma, respondiendo a eventos según sea necesario y sin afectar directamente a otros componentes.

Documentación y Comunicación de la Arquitectura

La documentación de la arquitectura es esencial para que todas las partes interesadas comprendan a fondo el diseño y las decisiones técnicas del sistema. Rozanski y Woods (2020) destacan que la documentación debe capturar tanto las decisiones técnicas como las razones que las motivan, asegurando una comunicación efectiva entre los equipos. Además, el enfoque de "vistas y más allá" propuesto por Clements et al. (2010) continúa siendo una referencia útil para estructurar la documentación de manera comprensible, aunque hoy en día se apoya también en enfoques visuales modernos y herramientas de automatización que permiten mantener la documentación arquitectónica actualizada de forma más eficiente.

C. Arquitectura basada en eventos

La arquitectura basada en eventos (EDA) permite que los componentes de un sistema interactúen generando, detectando y respondiendo a eventos de forma autónoma, lo cual facilita la comunicación asincrónica y el desacoplamiento entre sus elementos. Richards y Ford (2020) explican que este enfoque arquitectónico es ideal para construir sistemas escalables y reactivos, ya que los eventos desencadenan procesos específicos sin necesidad de una comunicación directa entre los componentes. Además, como señalan Bass et al. (2021), los eventos en EDA permiten que las aplicaciones respondan de manera eficiente y flexible, adaptándose dinámicamente a las demandas de procesamiento y habilitando un flujo de datos ágil entre los servicios del sistema.

Importancia de la Arquitectura Basada en Eventos

La arquitectura basada en eventos (EDA) está especialmente diseñada para aplicaciones que necesitan respuestas rápidas a eventos en tiempo real, aprovechando la flexibilidad y el desacoplamiento entre los componentes del sistema. Según Richards y Ford (2020), EDA permite que los eventos, como la creación de un nuevo pedido en un sistema de comercio electrónico o la actualización de datos en plataformas de IoT, desencadenen reacciones específicas sin la necesidad de que los componentes interactúen directamente. Como indican Bass et al. (2021), este enfoque es ideal para sistemas que deben manejar un flujo continuo de información, permitiendo respuestas ágiles y escalables ante diversos eventos.

La arquitectura basada en eventos (EDA) es fundamental para la creación de sistemas distribuidos resilientes y escalables. Como menciona Montgomery (2020), los sistemas que adoptan este enfoque permiten procesar datos en paralelo y desacoplar los componentes, evitando que estén directamente conectados, lo que facilita la escalabilidad horizontal y mejora la tolerancia a fallos. Además, Bass et al. (2021) señalan que esta arquitectura permite la comunicación asincrónica entre los componentes, lo que reduce la dependencia y mejora la capacidad de los sistemas para manejar fallos sin interrumpir el flujo de información. Esta flexibilidad también es clave para permitir que los sistemas crezcan de manera más eficiente y con un mayor rendimiento.

Principios de la Arquitectura Basada en Eventos

Los principios de la arquitectura basada en eventos son los siguientes:

1. ***Desacoplamiento***: En la arquitectura basada en eventos (EDA), los emisores y receptores de eventos están completamente desacoplados, lo que significa que un componente puede generar un evento sin necesidad de conocer a los consumidores de este. Según Newman (2021), esto reduce significativamente la dependencia entre los componentes y fomenta una mayor flexibilidad y escalabilidad. Además, Richards y Ford (2020) destacan que este desacoplamiento permite que los sistemas sean más resilientes y capaces de evolucionar sin afectar otras partes del sistema, ya que los cambios en un componente no requieren modificaciones en otros.
2. ***Asincronía***: En la arquitectura basada en eventos (EDA), los eventos se procesan de manera asincrónica, lo que permite que los sistemas respondan rápidamente a estímulos externos y distribuyan la carga de trabajo de manera eficiente. Según Richards y Ford (2020), la asincronía facilita que los componentes del sistema operen de manera independiente, lo que mejora la escalabilidad y la capacidad de respuesta. Además, Bass et al. (2021) mencionan que este enfoque reduce la congestión en el sistema al permitir que los eventos se manejen de manera paralela, lo que optimiza el rendimiento general y la capacidad para adaptarse a cambios en la carga de trabajo.
3. ***Escalabilidad y Elasticidad***: Al desacoplar los componentes y manejar los eventos de forma asincrónica, la arquitectura basada en eventos (EDA) facilita la escalabilidad horizontal, lo que permite que los sistemas manejen grandes cantidades de tráfico aumentando o disminuyendo la cantidad de instancias de procesamiento según sea necesario. Como señala Montgomery (2020), este enfoque optimiza el rendimiento y la flexibilidad del sistema al permitir que se ajuste dinámicamente a los cambios en la carga. Además, Newman (2021) destaca que la capacidad de escalar de manera independiente los componentes en EDA mejoran la resiliencia del sistema, ya que las instancias pueden ser agregadas o eliminadas sin interrumpir el funcionamiento del sistema en su totalidad.

4. **Consistencia Eventual:** Esto implica que, aunque puede haber retrasos en la propagación de los eventos, eventualmente todos los sistemas alcanzarán un estado coherente. Según Poniatoski (2021), este modelo es adecuado para aplicaciones que priorizan la disponibilidad y la escalabilidad sobre la consistencia inmediata. Además, como mencionan Richards y Ford (2020), la consistencia eventual es una característica fundamental en los sistemas distribuidos que utilizan EDA, ya que permite mantener un equilibrio entre la eficiencia en el procesamiento y la capacidad de respuesta del sistema.

Componentes de la Arquitectura Basada en Eventos

Los componentes de la arquitectura se detallan a continuación:

1. **Productores de eventos:** Los productores de eventos son componentes que son responsables de crear y emitir los eventos a un canal de eventos para su posterior procesamiento. Según Richards y Ford (2020), los productores de eventos permiten que el sistema reaccione a cambios en tiempo real, actuando como el punto de origen de los datos que se distribuyen a través del sistema. Además, Newman (2021) resalta que la capacidad de los productores de emitir eventos de manera eficiente es crucial para garantizar el rendimiento y la escalabilidad en sistemas distribuidos basados en eventos.
2. **Canales de eventos:** El canal de eventos es el medio a través del cual se transmiten los eventos de los productores a los consumidores y asegura que los eventos lleguen a los consumidores adecuados, garantizando un acoplamiento bajo entre las partes. Como señala Newman (2021), el uso de canales de eventos permite la comunicación asíncrona y desacoplada entre los componentes del sistema. Además, Richards y Ford (2020) destacan que los canales de eventos son esenciales para garantizar la escalabilidad y flexibilidad de los sistemas basados en eventos, ya que permiten distribuir la carga de trabajo de manera eficiente entre los consumidores.
3. **Consumidores de eventos:** Los consumidores de eventos son los componentes que reciben y reaccionan a los eventos generados por los productores que pueden actuar de inmediato en función del evento recibido

o delegar tareas adicionales a otros servicios, manteniendo la arquitectura desacoplada. Según Newman (2021), los consumidores son fundamentales para procesar los eventos en tiempo real, permitiendo que los sistemas respondan rápidamente a los cambios. Además, Richards y Ford (2020) afirman que los consumidores desempeñan un papel crucial en la adaptabilidad de los sistemas distribuidos, ya que pueden procesar los eventos de manera independiente, sin depender directamente de otros componentes.

4. **Procesadores de eventos:** Los procesadores de eventos son componentes intermedios que actúan sobre los eventos, transformándolos o agregando información antes de entregarlos a los consumidores. Como menciona Montgomery (2020), los procesadores de eventos permiten una mayor flexibilidad y eficiencia en el manejo de eventos, asegurando que los consumidores reciban los datos en el formato adecuado. Además, Newman (2021) destaca que la inclusión de procesadores de eventos en sistemas distribuidos puede mejorar el rendimiento y la escalabilidad, ya que permite delegar tareas específicas antes de que los eventos lleguen a su destino final.

Patrones en la Arquitectura Basada en Eventos

Hay varios patrones en la arquitectura, algunos de ellos son:

1. **Event Notification:** Es uno de los patrones fundamentales de la Arquitectura Basada en Eventos (EDA), donde un evento notifica a los consumidores que algo ha sucedido en el sistema sin proporcionar el estado completo del sistema, lo que permite a los consumidores reaccionar y tomar las acciones necesarias según el contexto de su propio dominio. Según Richards y Ford (2020), el patrón de Event Notification promueve un desacoplamiento entre los productores de eventos y los consumidores, permitiendo una mayor flexibilidad y escalabilidad. Además, Newman (2021) resalta que este patrón es ampliamente utilizado en aplicaciones que requieren una respuesta rápida a eventos sin la necesidad de transmitir grandes volúmenes de datos adicionales.
2. **Event-Carried State Transfer:** En este patrón, un evento incluye todo el estado necesario para que los consumidores puedan reaccionar sin tener

que consultar otros servicios, permitiendo que los consumidores procesen el evento de manera autónoma, lo que reduce la carga sobre las bases de datos y mejora la eficiencia general del sistema. Según Poniatowski (2021), este patrón es útil para sistemas que requieren un alto rendimiento y una rápida toma de decisiones, ya que elimina la necesidad de hacer consultas adicionales. Newman (2021) también destaca que el uso de este patrón facilita la escalabilidad y la consistencia al permitir que los sistemas operen de manera más autónoma.

3. **Event Sourcing:** En este patrón en lugar de almacenar el estado actual de una entidad en una base de datos, se almacena una serie de eventos que describen las modificaciones realizadas. Según Newman (2021), este patrón mejora la auditabilidad del sistema al proporcionar un registro completo y detallado de todos los cambios, lo que facilita la trazabilidad y el análisis histórico. Richards y Ford (2020) añaden que Event Sourcing también mejora la flexibilidad del sistema, ya que los eventos almacenados pueden ser procesados de diferentes maneras para adaptarse a nuevas necesidades sin cambiar el modelo de datos.

Ventajas de la Arquitectura Basada en Eventos

Dentro de las ventajas de usar la arquitectura basada en eventos tenemos:

1. **Escalabilidad:** EDA permite que los sistemas manejen grandes volúmenes de datos distribuyendo el procesamiento de eventos entre múltiples consumidores, lo que facilita la expansión de los sistemas sin comprometer el rendimiento (Richards & Ford, 2020). Al distribuir la carga de trabajo de esta manera, los sistemas pueden escalar de forma eficiente y adaptarse a las demandas cambiantes (Newman, 2021). Según Poniatowski (2021), esta capacidad de escalar horizontalmente hace que EDA sea especialmente útil para aplicaciones que deben gestionar altos volúmenes de transacciones o interacciones en tiempo real.
2. **Bajo acoplamiento:** La separación de componentes, que se comunican únicamente a través de eventos, reduce la dependencia entre las partes del sistema. Este desacoplamiento facilita la evolución y el mantenimiento del sistema, ya que los cambios en un componente no afectan directamente a

los demás (Poniatowski, 2021). Además, según Richards y Ford (2020), este bajo acoplamiento permite una mayor flexibilidad en el diseño del sistema, al facilitar la sustitución y actualización de componentes sin afectar el funcionamiento general.

3. **Capacidad de respuesta:** La arquitectura basada en eventos permite que los sistemas respondan en tiempo real a los eventos, mejorando la experiencia del usuario al ofrecer una interacción más ágil (Poniatowski, 2021). Según Richards y Ford (2020), esta capacidad de respuesta también facilita la rápida integración de nuevos servicios y funcionalidades, lo que permite a los sistemas adaptarse de manera flexible a los cambios en los requisitos del negocio.

Desafíos de la Arquitectura Basada en Eventos

Sin embargo, hay algunos desafíos que deben ser tomados en cuenta al optar la arquitectura basada en eventos:

1. **Complejidad operativa:** La gestión de eventos distribuidos en arquitecturas basadas en eventos puede volverse compleja, especialmente en sistemas grandes y distribuidos, debido al tráfico intenso que generan los eventos y a la dificultad de garantizar la coherencia entre múltiples componentes (Newman, 2021). Según Poniatowski (2021), esta complejidad se incrementa al intentar implementar la consistencia eventual, ya que los sistemas deben alcanzar un estado coherente sin una sincronización estricta, lo cual añade desafíos en términos de monitoreo y mantenimiento.
2. **Depuración y monitoreo:** La depuración y el monitoreo en arquitecturas basadas en eventos pueden ser especialmente desafiantes debido a la naturaleza asincrónica y el desacoplamiento de sus componentes. A diferencia de las arquitecturas tradicionales, en las que la comunicación es más directa y secuencial, los eventos pueden dispararse en cualquier momento, haciendo más complejo rastrear la causa raíz de un problema (Richards & Ford, 2020). Como destaca Newman (2021), la identificación de problemas en sistemas distribuidos basados en eventos requiere

herramientas avanzadas que permitan observar y correlacionar eventos de forma efectiva para asegurar la coherencia y disponibilidad del sistema.

3. ***Consistencia eventual:*** La consistencia eventual es común en sistemas basados en eventos, ya que permite que los componentes del sistema lleguen a un estado coherente con el tiempo, sin necesidad de sincronización inmediata. Sin embargo, en aplicaciones críticas, donde la precisión de los datos y la sincronización son esenciales, este modelo puede no ser suficiente, y podría ser necesario implementar una consistencia más estricta, lo cual añade complejidades técnicas y operativas (Newman, 2021). Richards y Ford (2020) señalan que esta necesidad de coherencia rápida en sistemas distribuidos puede impactar en el rendimiento, ya que requiere mecanismos adicionales para mantener la exactitud y reducir la latencia en la replicación de datos.

D. Patrón Notificación de Evento

Este patrón permite que un componente avise a otros sobre la ocurrencia de un evento, sin necesidad de transmitir detalles específicos del cambio o del estado actual del sistema, haciendo que los eventos sean ligeros y eficaces para transmitir solo lo esencial. Hohpe y Woolf (2020) destacan que esta simplicidad facilita la implementación y reduce la carga de los sistemas. Además, según Richards y Ford (2020), este patrón puede ser útil para notificaciones donde los componentes que reciben los eventos pueden decidir si requieren más información o deben tomar acciones adicionales basadas en su propio contexto.

Estructura

La estructura de este patrón contiene los siguientes elementos:

1. **Productor de eventos:** Es el responsable de generar y emitir el evento cuando ocurre una acción significativa. Este componente se encarga de detectar el evento y enviar una notificación a través de un canal adecuado (Hohpe & Woolf, 2020).
2. **Evento:** Representa el mensaje que se transmite, indicando que algo ha sucedido. Generalmente, es un mensaje ligero que contiene solo información básica sobre el evento y no necesariamente incluye el estado completo del sistema o entidad afectada (Richards & Ford, 2020).
3. **Consumidor de eventos:** Son los componentes que reciben el evento y reaccionan a él. Dependiendo del tipo de evento, los consumidores pueden necesitar o no acceder al estado completo de la entidad que lo generó. Este desacoplamiento permite que los consumidores tomen decisiones basadas solo en la información disponible en el evento (Hohpe & Woolf, 2020).

Beneficios

Algunos de los beneficios de usar este patrón son:

1. **Simplicidad:** La implementación de este patrón es relativamente sencilla, ya que los eventos no transportan mucha información. Esto reduce la carga de red y procesamiento, ya que los mensajes son livianos y se enfocan

únicamente en notificar que ha ocurrido un evento, sin la necesidad de transmitir grandes volúmenes de datos (Richards & Ford, 2020)

2. **Desacoplamiento:** Los productores y consumidores de eventos están desacoplados, lo que permite que cada uno evolucione de manera independiente. Los cambios en los productores no afectan directamente a los consumidores, facilitando la flexibilidad y la escalabilidad del sistema (Hohpe & Woolf, 2020).
3. **Eficiencia:** Al no incluir el estado completo del sistema en el evento, la transmisión de los eventos se realiza de manera rápida y eficiente. Este enfoque ayuda a reducir los costos de comunicación y a mejorar el rendimiento del sistema (Poniatowski, 2021).

Limitaciones

Sin embargo, algunas conocidas limitaciones de usar este patrón son:

1. **Carga adicional en sistemas externos:** Dado que los eventos no contienen el estado completo del sistema, los consumidores podrían necesitar hacer solicitudes adicionales a bases de datos o servicios externos para recuperar los detalles completos de la entidad que generó el evento. Esto puede generar una carga adicional en esos sistemas, afectando su rendimiento y escalabilidad, especialmente si el volumen de eventos es alto (Richards & Ford, 2020; Hohpe & Woolf, 2020).
2. **Latencia:** Esto se debe a que, a diferencia de otros patrones donde el estado se transfiere junto con el evento, los consumidores tienen que realizar solicitudes adicionales a otros servicios o bases de datos para obtener la información necesaria. Este retraso puede afectar la capacidad de respuesta del sistema, especialmente en aplicaciones donde la rapidez es crucial (Richards & Ford, 2020).

E. Patrón de Transferencia de Estado mediante Eventos

El patrón Event-Carried State Transfer (ECS) es un enfoque en las arquitecturas basadas en eventos donde no solo se notifica la ocurrencia de un evento, sino que también se incluye el estado completo o parcial de la entidad asociada a dicho evento. Este patrón es particularmente útil en contextos donde la sincronización de estado es crítica, ya que facilita la coherencia de la información sin necesidad de consultas adicionales (Fowler, 2020). Además, al integrar el estado directamente en el evento, ECS puede optimizar la eficiencia del sistema y reducir la sobrecarga de las interacciones entre los componentes (Richards & Ford, 2020).

Beneficios

Los principales beneficios de adoptar este patrón son:

1. ***Sincronización inmediata del estado:*** Este patrón proporciona una sincronización inmediata del estado, permitiendo que los consumidores reciban el estado completo en el evento y sincronicen su información sin hacer consultas adicionales al productor. Esto reduce la latencia y mejora la eficiencia en contextos de alta demanda de sincronización de datos (Fowler, 2020).
2. ***Desacoplamiento completo:*** Los componentes están completamente desacoplados, ya que los consumidores obtienen toda la información necesaria directamente en el evento sin tener que contactar al productor. Esto reduce la dependencia entre sistemas y aligerar la carga en los sistemas fuente, permitiendo que el sistema sea más escalable y eficiente (Fowler, 2020).
3. ***Mejora de la eficiencia:*** Dado que los eventos llevan el estado, se elimina la necesidad de realizar consultas adicionales al productor del evento. Esto minimiza la latencia y reduce la sobrecarga en la red, optimizando el flujo de datos en tiempo real y mejorando el rendimiento del sistema en general (Fowler, 2020).
4. ***Consistencia eventual:*** Esto es posible gracias a que los eventos incluyen el estado completo o parcial de la entidad, permitiendo que los consumidores actualicen su estado basado en el evento recibido y se

sincronicen gradualmente. Este enfoque ayuda a reducir la dependencia entre componentes y la sobrecarga en la comunicación entre sistemas, lo que mejora la escalabilidad en arquitecturas distribuidas (Fowler, 2020; Richards & Ford, 2020).

Limitaciones

Hay que tomar en cuenta las siguientes limitaciones al usar este patrón:

1. ***Posible duplicación de datos:*** Este patrón también trae consigo un posible riesgo de duplicación de datos en los diferentes sistemas que reciben los eventos, ya que cada uno podría mantener su propia copia del estado de la entidad. Esto puede llevar a un incremento en la complejidad de gestión de datos, especialmente en sistemas distribuidos que manejan grandes volúmenes de información (Newman, 2021; Fowler, 2020).
2. ***Tamaño de los eventos:*** Al incluir el estado completo en cada evento, puede resultar en un aumento considerable en el tamaño de los mensajes. Esto implica que, a medida que el sistema maneja eventos de mayor tamaño, se incrementa el consumo de ancho de banda y el procesamiento de red, lo cual puede afectar el rendimiento si no se optimiza adecuadamente (Fowler, 2020; Newman, 2021).
3. ***Gestión de la evolución del esquema:*** Si el estado de una entidad cambia, tanto el productor como los consumidores deben ser capaces de adaptarse a esos cambios de manera sincronizada. Sin embargo, si no se gestionan adecuadamente las versiones de los esquemas de datos, pueden surgir problemas como la incompatibilidad entre los sistemas que consumen los eventos, lo que añade complejidad a la arquitectura y a la implementación del patrón. En estos casos, tanto los productores como los consumidores deben tener mecanismos para manejar cambios en los esquemas sin afectar la integridad del sistema en su conjunto.

F. Modelo Publicador - Subscriptor

El patrón Publicador/Suscriptor es un patrón de diseño de software que sigue el principio de desacoplamiento entre productores y consumidores de mensajes donde un publicador emite eventos o mensajes sin conocer quién los recibirá. Los suscriptores, por otro lado, se registran para recibir mensajes específicos en función de sus intereses (Richards & Ford, 2020). Según Newman (2021), este patrón promueve la separación entre el productor y el consumidor de eventos, lo que permite una mayor escalabilidad y flexibilidad en sistemas distribuidos. Hohpe y Woolf (2020) destacan que, en este patrón, los publicadores no tienen que preocuparse por la forma en que los suscriptores procesan los eventos, lo que simplifica el diseño y la evolución de los sistemas.

Estructura

La estructura de este modelo contiene los siguientes elementos:

1. ***Publicador***: El publicador es el componente que emite eventos o mensajes en un sistema de mensajería sin tener conocimiento directo de los suscriptores. Según Newman (2021), el publicador envía eventos de manera asíncrona, y la lógica de negocio relacionada con el procesamiento de esos eventos recae en los suscriptores. Por su parte, Hohpe y Woolf (2020) explican que este patrón permite que los publicadores se centren únicamente en emitir eventos, lo que les otorga flexibilidad y les permite evolucionar sin preocuparse por el comportamiento de los consumidores.
2. ***Bróker de mensajes***: El bróker de mensajes actúa como intermediario entre los publicadores y suscriptores en un sistema de mensajería, facilitando la distribución de los mensajes a los suscriptores registrados para recibir ciertos eventos o tipos de mensajes. Según Richards y Ford (2020), el bróker asegura que los mensajes lleguen a los suscriptores adecuados, manteniendo un acoplamiento bajo entre productores y consumidores. Hohpe y Woolf (2020) también explican que el bróker de mensajes puede manejar la distribución de mensajes de manera eficiente y escalable, garantizando que la comunicación entre componentes del sistema distribuido sea flexible y sin dependencia directa entre ellos.

3. **Suscriptor:** Estos componentes reaccionan a los mensajes y pueden ejecutar acciones basadas en el contenido de los eventos que reciben. Según Richards y Ford (2020), los suscriptores se centran en actuar sobre los eventos que son relevantes para ellos, sin necesidad de conocer el origen del evento. Además, Hohpe y Woolf (2020) afirman que los suscriptores pueden desencadenar procesos adicionales o modificar el estado del sistema en función de la información contenida en los eventos recibidos.

Beneficios

Los principales beneficios de este modelo son:

1. **Escalabilidad:** El patrón es altamente escalable, ya que permite que múltiples suscriptores reciban el mismo mensaje sin que el publicador tenga que gestionar directamente a cada uno. Según Hohpe y Woolf (2020), este patrón se destaca por su capacidad de permitir una comunicación fluida y desacoplada entre los componentes. Además, Richards y Ford (2020) señalan que la escalabilidad del patrón se logra al permitir que los mensajes sean transmitidos a varios suscriptores sin que el publicador se vea afectado por el número de receptores.
2. **Desacoplamiento:** El patrón favorece el desacoplamiento entre los componentes del sistema, ya que los publicadores no necesitan conocer la identidad de los suscriptores, facilitando la extensión del sistema con nuevos componentes o suscriptores sin modificar la lógica existente. Según Hohpe y Woolf (2020), esta independencia entre los productores y consumidores de eventos mejora la flexibilidad del sistema, permitiendo que los suscriptores se agreguen o eliminen sin afectar al flujo general de mensajes. Además, Richards y Ford (2020) destacan que este desacoplamiento permite una mayor facilidad de mantenimiento y escalabilidad, ya que los componentes pueden evolucionar de forma independiente, sin que la introducción de nuevos suscriptores o cambios en los publicadores requiera modificaciones significativas en el sistema.
3. **Flexibilidad:** Este patrón facilita que los suscriptores reciban solo los mensajes relevantes para su operación, sin la necesidad de procesar

eventos no relacionados, lo que mejora la eficiencia y personalización. Según Hohpe y Woolf (2020), esta capacidad de personalizar los intereses de los suscriptores asegura que el sistema sea flexible y pueda evolucionar de acuerdo con las nuevas demandas sin complicar la lógica de los publicadores. Por su parte, Richards y Ford (2020) destacan que esta flexibilidad permite una mayor escalabilidad, ya que los nuevos tipos de eventos o suscriptores pueden ser añadidos fácilmente al sistema sin la necesidad de reestructuraciones complejas en la arquitectura existente.

Desafíos

Algunos desafíos para tomar en cuenta son:

1. ***Complejidad operativa:*** La gestión de sistemas de mensajería intermedios, como brókeres de mensajes, y el manejo de altos volúmenes de mensajes puede ser un desafío complejo que requiere una infraestructura robusta. Como se señala en la literatura sobre arquitectura basada en eventos, los sistemas de mensajería deben ser capaces de manejar tanto la escalabilidad como la resiliencia frente a fallos (Richards & Ford, 2020). Además, la necesidad de mantener la integridad y el orden de los mensajes, así como la gestión de posibles cuellos de botella, puede hacer que la infraestructura de mensajería sea un componente crítico en el rendimiento general del sistema (Hohpe & Woolf, 2020).
2. ***Garantías de entrega:*** Dependiendo del bróker de mensajes utilizado, pueden surgir problemas relacionados con la entrega en tiempo real de los eventos a todos los suscriptores. Algunos brókeres, como RabbitMQ y Kafka, manejan los mensajes de manera eficiente en sistemas distribuidos, pero pueden enfrentar dificultades con la latencia o la garantía de entrega en escenarios de alta concurrencia o de fallos en la red (Hohpe & Woolf, 2020). A pesar de que estos sistemas de mensajería permiten una distribución eficiente de eventos, mantener la sincronización en tiempo real entre productores y consumidores puede ser complicado cuando la infraestructura subyacente es compleja o se manejan grandes volúmenes de datos (Richards & Ford, 2020). La consistencia y la entrega garantizada de eventos también pueden depender de la configuración y las políticas de

reintentos del bróker, lo que podría introducir retrasos adicionales en la entrega de mensajes a los suscriptores.

3. **Consistencia:** Al permitir que los suscriptores procesen los mensajes de manera independiente, pueden surgir problemas como retrasos o inconsistencias en el estado de los sistemas que reciben dichos mensajes. Dado que cada suscriptor puede procesar los eventos de forma autónoma, los sistemas pueden estar en diferentes estados en un momento dado, lo que dificulta mantener una sincronización perfecta entre ellos (Newman, 2021). Además, cuando se gestionan grandes volúmenes de eventos, las diferencias en los tiempos de procesamiento pueden hacer que algunos suscriptores reciban eventos antes que otros, generando una latencia perceptible en la actualización de sus estados (Hohpe & Woolf, 2020). Esta falta de coordinación en el procesamiento de eventos también puede complicar el manejo de la consistencia de los datos a lo largo del sistema (Richards & Ford, 2020).

G. Bróker de mensajería

Un bróker de mensajería es un componente esencial en arquitecturas de software modernas, facilitando la comunicación entre diferentes aplicaciones, servicios o componentes mediante el intercambio de mensajes. Actúa como intermediario para enviar, recibir y almacenar mensajes, permitiendo que los sistemas se comuniquen de manera asíncrona y desacoplada. Esta característica resulta particularmente útil en entornos con arquitecturas basadas en microservicios, donde la modularidad y la escalabilidad son factores clave. Según Richards y Ford (2015), los bróker de mensajería como RabbitMQ o Kafka permiten a los sistemas distribuir tareas y eventos de manera eficiente sin que los servicios dependan directamente unos de otros, lo que mejora la flexibilidad y resiliencia del sistema. Además, como argumentan Grolinger et al. (2014), este enfoque de mensajería asíncrona ayuda a reducir la carga sobre las bases de datos y los servicios, optimizando la escalabilidad en sistemas distribuidos.

Funcionalidades Clave de un Bróker de Mensajería

Un bróker de mensajería tiene las siguientes funcionalidades:

1. ***Intermediación de Mensajes:*** Los brókeres de mensajería facilitan la intermediación de mensajes, actuando como un puente entre productores y consumidores. Los productores son los componentes que envían mensajes, mientras que los consumidores son aquellos que los reciben y procesan. Esta intermediación permite que ambos componentes funcionen de manera independiente, sin necesidad de estar directamente conectados o conscientes de la existencia del otro. Según Shaw et al. (2009), esta independencia mejora la escalabilidad y flexibilidad de las aplicaciones, permitiendo que los servicios se comuniquen de manera asíncrona. Además, como mencionan Chakraborty y Hossain (2019), este desacoplamiento entre productores y consumidores es fundamental en arquitecturas modernas basadas en microservicios, donde cada componente puede escalar y actualizarse sin afectar a los demás.
2. ***Almacenamiento de Mensajes:*** En caso de que el consumidor no esté disponible, los brókeres de mensajería pueden almacenar los mensajes en una cola hasta que el consumidor esté listo para procesarlos, garantizando

así la entrega confiable de mensajes incluso en situaciones de fallo. Esta capacidad de almacenamiento temporal asegura que los mensajes no se pierdan si el sistema experimenta fallos o picos de carga, lo que proporciona una alta disponibilidad y resiliencia en la comunicación asíncrona. Según Kreps et al. (2011), este enfoque es fundamental para asegurar la robustez en sistemas distribuidos, ya que permite que los mensajes se procesen una vez que la infraestructura esté disponible nuevamente. Además, como subrayan O'Reilly y Weng (2020), la cola también permite a los consumidores procesar los mensajes a su propio ritmo, mejorando la eficiencia y reduciendo la posibilidad de sobrecargar los sistemas.

3. ***Distribución de Cargas:*** Los brókeres de mensajería permiten la distribución de cargas entre varios consumidores, lo que facilita el escalado horizontal de los sistemas y mejora la eficiencia en el procesamiento de mensajes. Esta capacidad de distribuir el trabajo entre múltiples consumidores asegura que los sistemas puedan manejar mayores volúmenes de mensajes sin que un solo consumidor se convierta en un cuello de botella. Según Kreps (2011), este enfoque ayuda a mantener un alto rendimiento y una baja latencia, permitiendo que las aplicaciones distribuidas escalen eficientemente conforme aumentan los requisitos de tráfico. Además, como mencionan Fowler y Lewis (2014), este tipo de distribución también facilita la tolerancia a fallos, ya que, si un consumidor no está disponible, otros pueden tomar su lugar sin interrumpir el procesamiento de mensajes, lo que mejora la resiliencia del sistema.
4. ***Soporte para Diferentes Protocolos:*** Muchos brókeres de mensajería, como RabbitMQ, Kafka y ActiveMQ, soportan una variedad de protocolos de comunicación tales como AMQP, MQTT, y HTTP, lo que les permite integrarse con diferentes tecnologías y lenguajes de programación. Esto es fundamental para garantizar que los sistemas puedan comunicarse sin importar las tecnologías subyacentes. Según Kreps (2011), el soporte para múltiples protocolos no solo facilita la integración, sino que también promueve la interoperabilidad entre sistemas dispares, lo cual es esencial para arquitecturas de microservicios donde diferentes componentes a

menudo usan diferentes tecnologías. Además, Fowler y Lewis (2014) señalan que la compatibilidad con diversos protocolos mejora la flexibilidad de los brókeres, permitiendo que las aplicaciones escalen y evolucionen conforme cambian las necesidades del negocio.

5. **Persistencia:** Algunos brókeres ofrecen opciones de persistencia, lo que significa que los mensajes pueden ser almacenados en un sistema de almacenamiento duradero para garantizar que no se pierdan en caso de fallos. Según Joshi (2016), la persistencia en los brókeres permite una recuperación ante fallos efectiva, lo que es esencial para sistemas críticos, donde la pérdida de mensajes puede tener consecuencias graves. Además, Muthukrishnan et al. (2018) argumentan que la persistencia mejora la fiabilidad del sistema en general, dado que los mensajes no se pierden, sino que se mantienen en cola hasta que puedan ser procesados correctamente. Sin esta capacidad, los sistemas serían mucho más vulnerables a fallos inesperados, lo que podría resultar en pérdidas de datos o interrupciones en los servicios.
6. **Seguridad y Autenticación:** Los brókeres de mensajería implementan mecanismos de seguridad como autenticación, autorización y encriptación para proteger la comunicación de datos entre servicios, asegurando tanto la integridad como la confidencialidad. Estas medidas son críticas en arquitecturas distribuidas y de microservicios, donde se requiere una comunicación segura entre componentes (Di Francesco et al., 2019). Además, la encriptación ayuda a prevenir accesos no autorizados y proteger la privacidad de los datos, especialmente en sistemas que manejan información sensible (Newman, 2015).

Tipos de Brókeres de Mensajería

Dentro de los tipos de brókeres de mensajería tenemos:

1. **Brókeres Basados en Colas:** Los brókeres de mensajería basados en colas operan bajo un modelo de First-In, First-Out (FIFO), en el cual los mensajes se almacenan en una cola y son procesados secuencialmente por los consumidores. Este enfoque garantiza que cada mensaje sea procesado en el orden en el que fue recibido, ideal para aplicaciones que requieren

procesamiento exacto y confiable de cada mensaje una sola vez (Hohpe y Woolf, 2004). Al mantener la secuencia y persistencia de los mensajes, los sistemas basados en colas son particularmente útiles en entornos de misión crítica y para aplicaciones que deben cumplir con estrictos requisitos de consistencia y orden (Di Francesco et al., 2019).

2. **Brókeres Basados en Publicación/Suscripción:** En el modelo de *publish-subscribe*, los productores envían mensajes a un "tema" en lugar de a una cola específica, y los consumidores interesados se suscriben a esos temas. Esto permite que múltiples consumidores reciban simultáneamente el mismo mensaje, lo cual es ventajoso para arquitecturas en las que varias aplicaciones o servicios necesitan procesar la misma información (Tarkoma et al., 2011). Además, este modelo es ideal para sistemas distribuidos y escalables, ya que desacopla la fuente de datos de los consumidores, facilitando así una mayor flexibilidad y resiliencia en la gestión de eventos (Eugster et al., 2003).
3. **Brókeres Híbridos:** Los brókeres de mensajería híbridos combinan elementos de los modelos de colas y de publicación/suscripción para proporcionar flexibilidad y adaptabilidad en la comunicación entre sistemas distribuidos. Este tipo de brókeres permite que un mismo mensaje sea procesado por múltiples consumidores (como en el modelo *publish-subscribe*) mientras asegura que ciertos mensajes se procesen solo una vez por cada consumidor (como en el modelo de cola), adaptándose a las necesidades de distintos tipos de aplicaciones y patrones de consumo de datos (Chockler et al., 2001; Balazinska et al., 2005).

H. Escalamiento de Aplicaciones

El escalamiento de aplicaciones se refiere a la capacidad de ajustar los recursos disponibles para una aplicación con el objetivo de adaptarse a variaciones en la demanda de los usuarios o las necesidades operativas. En el contexto de arquitecturas modernas, como las basadas en la nube, contenedores y microservicios, esta capacidad ha tomado una importancia crucial para asegurar que las aplicaciones puedan mantener un rendimiento adecuado, incluso frente a cambios en el tráfico o la carga de trabajo (Newman, 2021; Hohpe & Woolf, 2020). Según Richards y Ford (2020), el escalamiento dinámico permite a las aplicaciones responder a picos de demanda sin comprometer la disponibilidad o el tiempo de respuesta, mientras que Fowler (2020) resalta que la arquitectura distribuida es clave para gestionar este tipo de escalabilidad, facilitando la expansión sin sacrificar la eficiencia operativa.

El objetivo principal del escalamiento es garantizar que una aplicación pueda gestionar un número creciente de solicitudes o usuarios sin afectar su rendimiento, a la vez que se optimizan los costos al reducir los recursos cuando no son necesarios. Este enfoque de escalamiento horizontal o vertical es esencial para mantener la disponibilidad y eficiencia en entornos de alta demanda. Según Newman (2021), el escalamiento no solo mejora la capacidad de manejar tráfico adicional, sino que también permite una mayor resiliencia ante fallos al distribuir la carga entre múltiples instancias. Además, como señalan Richards y Ford (2020), la capacidad de ajustar recursos dinámicamente ayuda a evitar el sobre aprovisionamiento y los costos asociados con la infraestructura subutilizada, lo que contribuye a un uso más eficiente de los recursos.

Tipos de Escalamiento

El escalamiento de aplicaciones se puede dividir en los siguientes tipos:

1. ***Escalamiento Vertical:*** El escalamiento vertical implica agregar más recursos (como CPU, memoria, almacenamiento) a una única instancia de servidor o máquina virtual para manejar mayores cargas. Este tipo de escalamiento es relativamente simple y se puede implementar rápidamente mediante ajustes en la configuración de la infraestructura existente. Según Newman (2021), este tipo de escalamiento es adecuado para aplicaciones

que no requieren distribución, ya que no se necesitan modificaciones significativas en el código o en la arquitectura de la aplicación. Además, como señalan Richards y Ford (2020), el escalamiento vertical suele ser más fácil de gestionar, ya que no involucra la complejidad de la orquestación entre múltiples instancias.

Sin embargo, uno de los límites más importantes del escalamiento vertical es que cada máquina tiene un límite físico en cuanto a la cantidad de recursos que puede manejar. Es decir, aunque es posible agregar más CPU y memoria hasta cierto punto, llegará un momento en que la máquina alcanzará su capacidad máxima, y no se podrán añadir más recursos sin cambiar el hardware subyacente. Este desafío es uno de los factores que impulsa a las organizaciones a considerar el escalamiento horizontal cuando las cargas de trabajo continúan creciendo (Hohpe & Woolf, 2020).

2. ***Escalamiento Horizontal:*** El escalamiento horizontal consiste en añadir más instancias de servidores o máquinas virtuales para distribuir la carga de la aplicación, lo que permite manejar mejor el tráfico de manera más eficiente. En lugar de aumentar los recursos de una sola máquina, este enfoque permite crear múltiples instancias que trabajan juntas para atender las solicitudes de los usuarios (Newman, 2021).

Una de las principales características del escalamiento horizontal es la distribución de la carga, donde un balanceador de carga distribuye las solicitudes o transacciones entre varias instancias de servidores, lo que facilita la gestión de grandes volúmenes de tráfico (Hohpe & Woolf, 2020). Además, este enfoque ofrece lo que se considera una escalabilidad teóricamente infinita, ya que no existe un límite práctico sobre la cantidad de instancias que se pueden agregar, lo que lo hace ideal para aplicaciones con grandes necesidades de escalabilidad (Richards & Ford, 2020).

Sin embargo, el escalamiento horizontal requiere que las aplicaciones estén diseñadas para entornos distribuidos. Esto introduce complejidades adicionales en términos de gestión de estado, consistencia de los datos y la coordinación entre las distintas instancias, como afirman Hohpe & Woolf (2020) y Newman (2021).

3. ***Escalamiento Automático:*** El escalamiento automático es una estrategia avanzada que permite ajustar los recursos de una aplicación de manera dinámica según la demanda. Plataformas en la nube como AWS, Azure y Google Cloud ofrecen servicios de auto-escalado que permiten que las aplicaciones aumenten o disminuyan su capacidad en función de métricas específicas, como el uso de CPU, la memoria o la latencia de red. Cuando las métricas alcanzan ciertos umbrales, se agregan automáticamente más instancias, mientras que cuando la carga disminuye, se eliminan las instancias no necesarias, optimizando así el uso de recursos (Newman, 2021; Hohpe & Woolf, 2020).

Una de las características más destacadas del escalamiento automático es la optimización de costos, ya que permite reducir la infraestructura en momentos de baja demanda. Al liberar recursos cuando no son necesarios, las aplicaciones pueden operar de manera más eficiente sin incurrir en costos adicionales innecesarios (Richards & Ford, 2020). Además, este enfoque garantiza escalabilidad en tiempo real, respondiendo de manera dinámica a los cambios en la carga y asegurando que la aplicación siempre disponga de los recursos adecuados según el tráfico o las necesidades operativas (Hohpe & Woolf, 2020).

Estrategias para el Escalamiento Eficiente

Para lograr un escalamiento correcto las principales estrategias son:

1. ***Separación de Servicios (Descomposición en Microservicios):*** Las aplicaciones monolíticas enfrentan desafíos al escalar de manera eficiente, ya que todos sus componentes deben ejecutarse juntos, lo que dificulta la asignación óptima de recursos. En cambio, al adoptar una arquitectura de microservicios, cada servicio puede escalarse de forma independiente según la carga de trabajo específica, lo que mejora la eficiencia en la utilización de recursos. Esta flexibilidad permite que las partes de la aplicación que requieren más recursos se escalen sin afectar a otros componentes, optimizando así el rendimiento y los costos operativos (Newman, 2021; Richards & Ford, 2020). Además, este enfoque facilita la implementación de cambios y actualizaciones en servicios individuales sin

interrumpir el funcionamiento del sistema completo, mejorando la agilidad del desarrollo (Hohpe & Woolf, 2020).

2. **Caching (Cache):** El uso de cachés es clave para mejorar la escalabilidad de las aplicaciones, especialmente aquellas que dependen de bases de datos para recuperar datos. Los sistemas de caché, como Redis o Memcached, almacenan temporalmente los resultados de consultas y procesos frecuentes, lo que reduce significativamente la carga sobre los sistemas de backend y mejora el tiempo de respuesta. Este enfoque es particularmente útil en aplicaciones que enfrentan un alto volumen de solicitudes, ya que disminuye la necesidad de acceder repetidamente a la base de datos, permitiendo un mejor manejo del tráfico y mejorando el rendimiento general del sistema (Hohpe & Woolf, 2020; Richards & Ford, 2020). Además, la integración de un sistema de caché en una arquitectura de microservicios permite que los servicios gestionen su propia memoria caché, mejorando la eficiencia de cada servicio de forma independiente (Newman, 2021).
3. **Balanceo de Carga:** El balanceo de carga es una técnica clave para distribuir de manera equitativa el tráfico entre varias instancias de una aplicación, evitando que una sola instancia se vea sobrecargada. Este proceso es fundamental para el escalado horizontal, ya que permite que las solicitudes se gestionen de manera más eficiente a medida que se agregan más instancias al sistema. Un balanceador de carga actúa como un intermediario entre los clientes y las instancias de la aplicación, enviando las solicitudes de manera inteligente a las instancias más disponibles o menos cargadas, lo que asegura un rendimiento más estable y una experiencia de usuario optimizada (Richards & Ford, 2020; Newman, 2021). Además, esta técnica es esencial en arquitecturas basadas en microservicios, donde cada servicio puede escalar de forma independiente, garantizando que la carga de trabajo se distribuya adecuadamente entre las diferentes partes del sistema (Hohpe & Woolf, 2020).
4. **Escalamiento Basado en Colas:** En sistemas que procesan tareas en segundo plano, el uso de colas de mensajes como RabbitMQ, Kafka o Azure Queue Storage es una estrategia común para gestionar la distribución de trabajos entre los consumidores. Estos sistemas permiten

ajustar dinámicamente el número de instancias consumidoras en función de la carga de trabajo de la cola. Si el volumen de mensajes pendientes es alto, se pueden añadir más consumidores para procesarlos rápidamente. A medida que la cantidad de tareas pendientes disminuye, se pueden eliminar las instancias adicionales, lo que optimiza los recursos y los costos operativos. Esta flexibilidad es esencial para mantener un rendimiento óptimo sin sobrecargar los sistemas ni incurrir en gastos innecesarios (Hohpe & Woolf, 2020; Newman, 2021).

5. ***Serverless Architecture (Arquitectura Sin Servidores)***: En una arquitectura serverless, los recursos se gestionan de manera dinámica, adaptándose automáticamente a la demanda sin que los desarrolladores tengan que gestionar los servidores. En este modelo, se implementan funciones que se ejecutan en respuesta a eventos, y plataformas como AWS Lambda o Azure Functions proporcionan escalado automático. Esto significa que los recursos son asignados por la plataforma según sea necesario, garantizando que las aplicaciones puedan escalar de forma eficiente sin intervención manual, lo que permite a los desarrolladores centrarse solo en el código y la lógica de negocio (Newman, 2021; Hohpe & Woolf, 2020).

Desafíos del Escalamiento

Sin embargo, algunos desafíos al escalar aplicaciones son:

1. ***Gestión de Estado***: La gestión del estado es un desafío clave en el escalado horizontal. Al distribuir las cargas entre múltiples instancias, se complica el manejo de estado si la aplicación depende de datos específicos de cada instancia. Esto puede resolverse de varias maneras: mediante la creación de aplicaciones "stateless", donde no se mantiene estado dentro de las instancias, o utilizando mecanismos externos de almacenamiento de estado, como bases de datos distribuidas, sistemas de caché compartidos (como Redis o Memcached), o almacenamiento persistente que permita a todas las instancias acceder a los mismos datos en tiempo real (Hohpe & Woolf, 2020; Newman, 2021).

2. **Consistencia de Datos:** La consistencia de datos es uno de los principales desafíos cuando se escala horizontalmente, ya que a medida que más instancias se distribuyen, mantener los datos sincronizados entre ellas se vuelve más complejo. En sistemas distribuidos, los problemas de replicación de datos, latencia de red y la necesidad de aplicar consistencia eventual son comunes. Para abordar esto, es fundamental utilizar soluciones como bases de datos distribuidas, sistemas de replicación y patrones arquitectónicos que aseguren una separación eficiente de los comandos y las consultas, como el patrón CQRS (Command Query Responsibility Segregation). Este patrón ayuda a manejar los comandos y consultas de manera independiente, optimizando la consistencia y el rendimiento del sistema (Fowler, 2020; Newman, 2021).
3. **Costo:** A medida que una aplicación escala, los costos asociados con la infraestructura también pueden aumentar significativamente, especialmente cuando se utilizan servicios en la nube. Este aumento de costos puede ser un desafío considerable, ya que se debe encontrar un equilibrio entre escalar la aplicación para satisfacer la demanda y controlar los gastos operativos. Estrategias como el auto-escalado permiten ajustar automáticamente los recursos según el tráfico, lo que ayuda a evitar el desperdicio de recursos durante períodos de baja demanda. Sin embargo, la planificación y optimización de los recursos son fundamentales para evitar gastos innecesarios y lograr una escalabilidad eficiente. El uso de métodos como el análisis de métricas y el ajuste de umbrales de escalamiento puede ayudar a garantizar que los recursos solo se aumenten cuando sea necesario, minimizando los costos a largo plazo (Burns et al., 2016; Hohpe & Woolf, 2020).

I. Kubernetes

Kubernetes es una plataforma de código abierto que facilita la gestión de aplicaciones en contenedores, y se ha establecido como el estándar para la orquestación de contenedores. Desarrollada inicialmente por Google y ahora mantenida por la Cloud Native Computing Foundation (CNCF), Kubernetes permite automatizar tareas clave como el despliegue, escalamiento y administración de aplicaciones. Su arquitectura se basa en una serie de componentes que trabajan juntos para gestionar entornos de producción, facilitando la ejecución de aplicaciones distribuidas de forma escalable y eficiente (Hightower et al., 2017; Burns et al., 2016). A través de su enfoque en contenedores, Kubernetes ayuda a abstraer la infraestructura subyacente, lo que permite a los desarrolladores centrarse más en la creación de aplicaciones que en la gestión de la infraestructura, mientras que los operadores pueden automatizar tareas operativas y gestionar aplicaciones a gran escala de manera eficiente.

Arquitectura de Kubernetes

La arquitectura de la plataforma tiene los siguientes componentes:

1. *Nodos:*

Un clúster de Kubernetes está compuesto por dos tipos de nodos, cada uno con roles específicos en la gestión y ejecución de las aplicaciones:

1.1. Nodos maestros (Master nodes): Son responsables de la administración general del clúster. Gestionan y coordinan los recursos, como la programación de Pods y el mantenimiento del estado deseado del sistema. Los nodos maestros incluyen componentes como el API server, que sirve de interfaz para interactuar con el clúster; el controller manager, que se encarga de mantener el estado del sistema; y el scheduler, que determina en qué nodo de trabajo debe ejecutarse un Pod (Hightower et al., 2017).

1.2. Nodos de trabajo (Worker nodes): Son los nodos en los que se ejecutan las aplicaciones en contenedores. Cada nodo de trabajo ejecuta uno o más Pods. Estos nodos tienen componentes clave como el kubelet, que garantiza que los contenedores estén

funcionando según lo especificado, y el kube-proxy, que maneja la red entre los Pods y permite el acceso desde y hacia el exterior del clúster (Kelsey Hightower et al., 2016).

Esta separación permite una eficiente organización y gestión de las aplicaciones, optimizando tanto el control como la ejecución de los contenedores en un entorno distribuido.

2. *Pods*

Un Pod en Kubernetes es la unidad más pequeña de despliegue y ejecución dentro de un clúster. Este puede contener uno o más contenedores, los cuales comparten el mismo espacio de red y almacenamiento, lo que les permite interactuar de manera eficiente. Los Pods son efímeros, lo que significa que pueden ser creados y destruidos dinámicamente según la demanda de la carga de trabajo (Hightower et al., 2017). Además, cada Pod tiene una dirección IP única dentro del clúster, lo que facilita la comunicación entre Pods. Sin embargo, los Pods son diseñados para ser reemplazados, no modificados, lo que implica que si un Pod falla o es eliminado, otro será creado automáticamente para reemplazarlo (Burns et al., 2016).

Este diseño permite que Kubernetes maneje aplicaciones de manera altamente disponible y resiliente, ajustando los recursos según sea necesario y proporcionando un entorno eficiente para la ejecución de contenedores (Hightower et al., 2017).

3. *Servicios (Services)*

En Kubernetes, un Servicio actúa como una abstracción que permite definir un conjunto de Pods y una política de acceso a ellos. Los servicios son fundamentales para permitir la comunicación entre diferentes partes de una aplicación dentro de un clúster, ya que proporcionan un punto de acceso estable a un conjunto de Pods, incluso cuando estos se escalan o se reinician. Esto es crucial en entornos dinámicos donde las instancias de la aplicación pueden cambiar frecuentemente debido al escalado automático o la actualización de Pods (Hightower et al., 2017).

Kubernetes soporta varios tipos de servicios para gestionar diferentes necesidades de accesibilidad:

3.1.ClusterIP: Este tipo de servicio expone el conjunto de Pods solo dentro del clúster, lo que significa que los Pods solo pueden comunicarse entre sí dentro de ese entorno (Hightower et al., 2017).

3.2.NodePort: Permite acceder a los Pods a través de un puerto específico en cada nodo del clúster, lo que facilita el acceso desde fuera del clúster.

3.3.LoadBalancer: En entornos en la nube, este servicio provisiona automáticamente un balanceador de carga externo, proporcionando acceso equilibrado a los Pods distribuidos en el clúster (Burns et al., 2016).

4. Controladores

En Kubernetes, los controladores son componentes que gestionan el estado de los recursos del clúster, asegurando que los Pods estén en el estado deseado de manera continua. Algunos de los controladores más comunes incluyen:

4.1.ReplicaSet: Este controlador asegura que un número específico de réplicas de un Pod esté corriendo en todo momento. Si un Pod se cae, el ReplicaSet crea uno nuevo para mantener el número de réplicas constante, garantizando alta disponibilidad y resiliencia (Hightower et al., 2017).

4.2.Deployment: A diferencia del ReplicaSet, que maneja la cantidad de Pods, el Deployment se encarga de la actualización y gestión de Pods a través de estrategias declarativas. Permite realizar despliegues progresivos de nuevas versiones de los Pods, facilitando la actualización y la reversión a versiones anteriores si es necesario (Burns et al., 2016).

4.3.StatefulSet: Este controlador es útil para aplicaciones con estado, como bases de datos, donde el orden y la identidad de los Pods son importantes. Asegura que los Pods sean creados, eliminados y gestionados de manera ordenada, y también mantiene la persistencia de los datos entre reinicios de Pods, lo que es esencial

para aplicaciones como bases de datos o sistemas distribuidos (Burns et al., 2016).

Estos controladores permiten la gestión eficiente de las aplicaciones dentro de Kubernetes, manteniendo la infraestructura estable y flexible a medida que las aplicaciones escalan o se actualizan.

5. *Etc*

Etc es una base de datos distribuida clave-valor utilizada por Kubernetes para almacenar todos los datos de configuración del clúster, así como el estado actual de los recursos. Esta base de datos es fundamental para el funcionamiento de Kubernetes, ya que mantiene la consistencia del estado del clúster. Etc asegura que las modificaciones a la configuración o al estado del clúster sean persistentes, distribuidas y accesibles a todos los nodos del clúster. Al ser un sistema altamente disponible y tolerante a fallos, Etc ayuda a garantizar que la infraestructura de Kubernetes funcione de manera confiable incluso en presencia de fallos de red o de nodos (Kelsey et al., 2018; Hightower et al., 2017).

Además, Etc utiliza un modelo de consistencia fuerte basado en el algoritmo Raft, lo que asegura que los datos sean consistentes y estén siempre sincronizados entre los nodos. Esto es crucial para los entornos de Kubernetes, donde el estado del clúster debe ser confiable para coordinar las acciones entre los nodos y los Pods de la infraestructura (Burns et al., 2016).

6. *API Server*

El API Server es un componente esencial dentro de la arquitectura de Kubernetes, ya que expone la API de Kubernetes y actúa como el punto de entrada para todas las interacciones con el clúster. A través de este servidor, los usuarios y otros componentes del sistema pueden realizar operaciones CRUD (crear, leer, actualizar, eliminar) sobre los recursos del clúster, como Pods, Servicios, Deployments, entre otros (Hightower et al., 2017; Burns et al., 2016).

El API Server se encarga de recibir las solicitudes, validarlas y autorizarlas, y luego dirigir las a los componentes correspondientes dentro del clúster. Además, es el responsable de gestionar el acceso y la coherencia de los datos en el sistema, funcionando como un intermediario entre el usuario y el clúster (Kelsey et al., 2018). Esto incluye la comunicación con etcd, donde se almacenan los estados de los recursos del clúster (Burns et al., 2016).

7. *Scheduler*

El Scheduler en Kubernetes es un componente clave que se encarga de decidir en qué nodo debe ejecutarse un Pod dentro de un clúster. Este proceso de asignación se basa en la disponibilidad de recursos y en las restricciones definidas para cada Pod. El Scheduler evalúa factores como la capacidad de CPU, memoria y almacenamiento disponibles en los nodos, además de considerar otros criterios como los requerimientos de afinidad, anti-afinidad, políticas de taints y tolerations, y prioridades específicas para asegurar que los Pods se ubiquen de manera óptima.

El Scheduler funciona en estrecha colaboración con otros componentes de Kubernetes, como el API Server, para hacer que la asignación de recursos sea eficiente y acorde a las políticas definidas por el usuario (Burns et al., 2016). Este proceso no solo incluye el análisis de recursos disponibles, sino también la gestión de la distribución de los Pods de manera equilibrada a través de los nodos, lo que permite un escalado adecuado y mejora el rendimiento del clúster (Kelsey et al., 2018).

Ventajas de Kubernetes

Las principales ventajas de usar Kubernetes son:

1. ***Portabilidad:*** Kubernetes permite a las organizaciones ejecutar aplicaciones en una variedad de entornos, desde infraestructuras locales hasta plataformas de nube pública o híbridas. Gracias a su abstracción sobre los recursos subyacentes, las aplicaciones pueden moverse fácilmente entre diferentes entornos de nube sin reconfiguración significativa (Burns et al., 2016). Esto lo convierte en una solución ideal

para empresas que buscan flexibilidad y portabilidad entre diferentes proveedores de servicios en la nube.

2. **Escalabilidad:** Kubernetes facilita el escalado automático y manual de aplicaciones. Aunque no es el foco principal del tema, la capacidad de escalar aplicaciones horizontalmente (añadiendo más Pods) o verticalmente (aumentando los recursos de los Pods existentes) es fundamental para gestionar cargas de trabajo fluctuantes. Además, el auto-escalado se puede configurar para ajustar los recursos de acuerdo con métricas específicas como el uso de CPU o memoria (Hightower et al., 2017).
3. **Resiliencia:** Kubernetes garantiza la alta disponibilidad de las aplicaciones mediante la gestión automática de fallos. Si un contenedor o Pod falla, Kubernetes lo reinicia automáticamente, asegurando que siempre haya el número deseado de instancias en ejecución. Esta capacidad de recuperación ante fallos es clave para garantizar una operación continua y sin interrupciones (Kelsey et al., 2018).
4. **Despliegue Continuo:** Kubernetes permite implementar nuevas versiones de aplicaciones sin tiempo de inactividad, mediante prácticas como canary deployments o rolling updates. Estas técnicas permiten desplegar cambios de manera gradual, reduciendo el riesgo de fallos y permitiendo a los equipos de desarrollo realizar actualizaciones de forma más eficiente (Burns et al., 2016).

J. Escalamiento de aplicaciones en Kubernetes

El escalamiento en Kubernetes es el proceso de ajustar automáticamente la cantidad de recursos disponibles para una aplicación en respuesta a cambios en la demanda, optimizando el uso de los recursos del clúster. Esto se logra mediante la adición o eliminación de réplicas de Pods, lo que permite que las aplicaciones manejen aumentos o disminuciones de tráfico sin intervención manual. Kubernetes facilita el escalamiento tanto de manera vertical (aumentando los recursos de un Pod individual) como horizontal (aumentando o reduciendo el número de Pods en ejecución), lo que ayuda a garantizar un rendimiento constante y eficiente en entornos de producción (Kelsey et al., 2018; Hightower, 2020).

Tipos de Escalamiento en Kubernetes

Kubernetes soporta principalmente tres tipos de escalamiento: escalamiento horizontal, escalamiento vertical, y escalamiento de clúster, usando componentes nativos para cada uno.

1. ***Escalamiento Horizontal (Horizontal Pod Autoscaler - HPA):*** El escalamiento horizontal en Kubernetes implica la capacidad de aumentar o disminuir el número de Pods en función de la demanda de recursos. Este tipo de escalamiento es gestionado principalmente por el Horizontal Pod Autoscaler (HPA), que ajusta el número de réplicas de un Pod basado en métricas como el uso de CPU, memoria o métricas personalizadas.

Algunas características clave del escalamiento horizontal incluyen:

1.1.Escalado automático: El HPA ajusta dinámicamente el número de Pods en función de los requisitos de recursos, como la utilización de CPU o memoria, manteniendo el rendimiento adecuado según la carga de trabajo (Hightower, 2020).

1.2.Soporte para métricas personalizadas: Además de las métricas estándar, el HPA también puede configurarse para escalar basándose en métricas personalizadas como la latencia de respuesta o datos específicos de la aplicación, proporcionando flexibilidad en el control del comportamiento de las aplicaciones (Kelsey et al., 2018).

2. **Escalamiento Vertical (Vertical Pod Autoscaler - VPA):** El Vertical Pod Autoscaler (VPA) ajusta los recursos de CPU y memoria asignados a un Pod individual. Este proceso aumenta o disminuye los recursos necesarios para mantener la eficiencia de la aplicación sin agregar más Pods.

Características:

2.1. Ajuste de recursos: El VPA puede ajustar dinámicamente los recursos (CPU y memoria) de un Pod en función de su carga de trabajo. Este ajuste es realizado para garantizar que el Pod siempre tenga suficientes recursos para su operación eficiente (Hightower, 2020).

2.2. Reinicio de Pods: Debido a los cambios en los recursos, el VPA puede requerir el reinicio del Pod, lo cual podría impactar brevemente su disponibilidad, aunque en la mayoría de los casos, Kubernetes maneja este reinicio de manera que minimiza el impacto en los usuarios (Kelsey et al., 2018).

3. **Escalamiento de Clúster (Cluster Autoscaler):** El Cluster Autoscaler ajusta el número de nodos en el clúster. Si la demanda de recursos excede la capacidad del clúster, se agregan más nodos, y si los recursos son infrautilizados, se eliminan.

Características:

3.1. Escalado automático de nodos: El Cluster Autoscaler agrega o elimina nodos según la carga de trabajo, ayudando a gestionar eficientemente los recursos del clúster (Kelsey et al., 2018).

3.2. Optimización de costos: Reduce el número de nodos cuando no son necesarios, lo que ayuda a reducir los costos operativos, especialmente en entornos de nube (Hightower, 2020).

K. Modelo de arquitectura C4

El modelo C4 (Context, Containers, Components, and Code) es un enfoque estructurado para modelar y documentar la arquitectura de software de manera que sea comprensible para todos los involucrados, desde desarrolladores hasta partes interesadas no técnicas. Fue desarrollado por Simon Brown y tiene como objetivo representar la arquitectura de sistemas complejos a través de diagramas en varios niveles de abstracción. El modelo se enfoca en proporcionar una visión clara de cómo se interrelacionan los componentes del sistema, ayudando a simplificar la documentación y a hacerla accesible y útil tanto para el desarrollo como para la comunicación entre los equipos.

Al estar basado en un enfoque visual y modular, el modelo C4 es especialmente útil para documentar sistemas distribuidos y aplicaciones basadas en microservicios, donde las interacciones y la comprensión de la estructura son clave para garantizar el buen funcionamiento y la escalabilidad del sistema. Además, permite a los equipos trabajar de manera más eficiente, ya que facilita la colaboración y la toma de decisiones informadas sobre el diseño y las modificaciones del sistema (Brown, 2011; Hohensee, 2018).

Niveles del Modelo C4

El modelo C4 se divide en cuatro niveles de representación, cada uno enfocado en un nivel diferente de detalle:

1. **Diagrama de Contexto:** El diagrama de contexto proporciona una visión general del sistema en su entorno. Muestra cómo el sistema interactúa con actores externos, como usuarios y otros sistemas, y destaca las relaciones entre ellos.

Elementos:

- **Sistema:** Representa el sistema en cuestión.
- **Actores:** Personas o sistemas externos que interactúan con el sistema.
- **Relaciones:** Conexiones que indican cómo los actores interactúan con el sistema.

Este tipo de diagrama es fundamental para definir el alcance de un proyecto y establecer las interacciones con los elementos externos de manera sencilla y clara, sin entrar en detalles complejos sobre la implementación o la estructura interna del sistema (Brown, 2011; Hohensee, 2018).

2. **Diagrama de Contenedores:** El diagrama de contenedores descompone el sistema en contenedores individuales que representan aplicaciones o servicios. Cada contenedor se caracteriza por su propósito, tecnología y las interacciones con otros contenedores.

Elementos:

- **Contenedores:** Pueden ser aplicaciones, servicios web, bases de datos, etc.
- **Tecnologías:** Lenguajes de programación, frameworks o plataformas utilizados.
- **Interacciones:** Cómo se comunican los contenedores entre sí.

Este tipo de diagrama es fundamental para definir el alcance de un proyecto y establecer las interacciones con los elementos externos de manera sencilla y clara, sin entrar en detalles complejos sobre la implementación o la estructura interna del sistema (Brown, 2011; Hohensee, 2018).

3. **Diagrama de Componentes:** El diagrama de componentes ofrece un nivel más detallado, mostrando la estructura interna de un contenedor. Este diagrama identifica los componentes principales dentro de un contenedor y cómo se comunican entre sí.

Elementos:

- **Componentes:** Clases, módulos o servicios que constituyen el contenedor.
- **Interfaces:** Cómo los componentes interactúan entre sí.
- **Dependencias:** Relaciones que indican las dependencias entre componentes.

Este diagrama ayuda a los equipos a comprender la estructura del sistema y cómo los diferentes componentes (contenedores) se relacionan y cooperan entre sí. Es especialmente útil en sistemas complejos donde se necesita entender cómo las aplicaciones y servicios se comunican sin entrar en detalles de bajo nivel sobre cada componente (Brown, 2011; Hohensee, 2018).

4. **Diagrama de Código:** El diagrama de código es el nivel más detallado y puede incluir diagramas de clases o de secuencia. Se centra en la implementación técnica de los componentes, mostrando cómo se estructuran las clases y los métodos.

Elementos:

- **Clases:** Representan las entidades en el código.
- **Métodos:** Las funciones dentro de las clases que realizan acciones.
- **Relaciones:** Cómo las clases se relacionan entre sí.

Este diagrama proporciona una visión detallada de cómo la arquitectura conceptual y los componentes funcionales se traducen en la implementación de código, lo que es esencial para garantizar que el diseño sea coherente con las necesidades del sistema y facilitar el mantenimiento y la evolución del software.

Beneficios del Modelo C4

El modelo C4 proporciona varios beneficios para los equipos de desarrollo y las partes interesadas:

1. **Claridad:** Al proporcionar diagramas de diferentes niveles de abstracción, el modelo C4 ayuda a clarificar la estructura del sistema y la interacción entre sus componentes. Esto facilita la comprensión tanto de los desarrolladores como de otros interesados, como gerentes o clientes, mejorando la comunicación entre ellos (Brown, 2011).
2. **Evolutivo:** El enfoque modular del modelo C4 permite que la documentación se mantenga actualizada de manera eficiente a medida que el sistema evoluciona. Los cambios en la arquitectura pueden reflejarse

rápidamente sin requerir grandes reescrituras del material, lo que lo hace útil a lo largo de todo el ciclo de vida del software (Hohensee, 2018).

3. ***Enfoque en el Usuario:*** Al centrarse en los actores externos y sus interacciones con el sistema, el modelo C4 promueve un diseño más enfocado en las necesidades del usuario. Los diagramas de contexto y contenedores, por ejemplo, ayudan a identificar cómo los usuarios y otros sistemas interactúan con la solución, garantizando que el diseño sea útil y accesible para los usuarios finales.

L. Domain Driven Desing

El Domain-Driven Design (DDD) es un enfoque estratégico y técnico para el desarrollo de software que busca alinear las soluciones tecnológicas con las necesidades del negocio mediante la comprensión profunda del dominio en el cual opera la aplicación. Propuesto por Eric Evans en 2003, DDD ha sido adoptado como un marco clave para manejar la complejidad inherente al desarrollo de sistemas orientados a dominios específicos (Evans, 2004; Vernon, 2016).

DDD establece que el núcleo del desarrollo de software debe centrarse en el dominio del problema, que representa el área de conocimiento o actividad principal del negocio. Este enfoque asegura que las decisiones técnicas estén alineadas con los objetivos estratégicos de la organización (Evans, 2004). Según Fowler (2003), comprender el dominio no solo permite crear modelos que reflejen fielmente la realidad del negocio, sino que también facilita la comunicación entre expertos técnicos y no técnicos, promoviendo una visión compartida del sistema.

Lenguaje Ubicuo: Un Lenguaje Común

Uno de los pilares de DDD es el lenguaje ubicuo, un vocabulario compartido entre desarrolladores y expertos del dominio que garantiza claridad y precisión en la comunicación. Este lenguaje elimina ambigüedades y se plasma directamente en el código, reforzando la conexión entre el modelo conceptual y la implementación técnica (Evans, 2004; Millett & Tune, 2015). Beck (2004) enfatiza que el lenguaje ubicuo fomenta una colaboración constante, esencial para mantener la alineación entre el equipo de desarrollo y el negocio.

Modelado Estratégico: Subdominios y Contextos Delimitados

DDD introduce conceptos como subdominios y contextos delimitados para estructurar y manejar la complejidad del software. Los subdominios identifican las diferentes áreas funcionales dentro del dominio, mientras que los contextos delimitados definen los límites de cada modelo, promoviendo modularidad y cohesión en el diseño (Evans, 2004). Según Vernon (2016), esta separación permite a los equipos trabajar en paralelo sin conflictos, facilitando la evolución y el escalamiento del sistema.

Implementación Práctica: Patrones Tácticos

Además de su visión estratégica, DDD ofrece patrones tácticos como entidades, objetos de valor, agregados, repositorios y servicios de dominio para guiar la implementación. Estos patrones estructuran el modelo de dominio y aseguran que las reglas de negocio se representen correctamente en el software (Evans, 2004). Por ejemplo, Richards y Ford (2020) destacan que el uso de agregados mejora la consistencia y reduce la complejidad al gestionar las operaciones críticas dentro de límites claramente definidos.

Beneficios y Retos del Enfoque DDD

El enfoque DDD aporta beneficios como una mayor alineación con los objetivos del negocio, claridad en el diseño y capacidad de adaptarse a cambios en los requisitos del dominio (Evans, 2004; Vernon, 2016). Sin embargo, también enfrenta retos como su curva de aprendizaje y la dificultad de implementarlo en organizaciones con una cultura que no fomente la colaboración interdisciplinaria (Fowler, 2003; Millett & Tune, 2015).

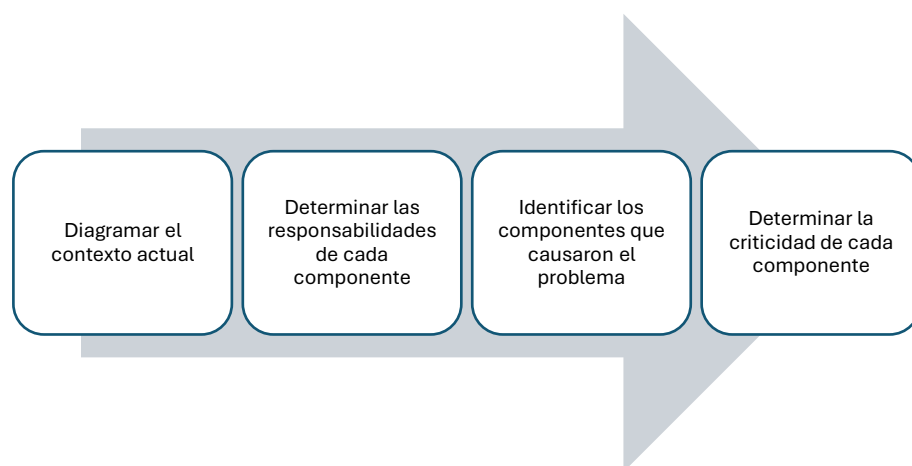
3.2.5. Implementación de las áreas, procesos, sistemas y buenas prácticas

1. Diagnosticar el problema

El diagnóstico del problema fue realizado por el equipo de depósitos en conjunto siguiendo los pasos que se muestran en la **Figura 15** para determinar cómo se encontraba el sistema en el momento del problema y determinar los componentes involucrados y sus responsabilidades para poder elegir cuáles de ellos participarían en la solución.

Figura 15

Pasos para el diagnóstico del problema



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Elaboración propia.

1.1. Paso 1. Diagramar el contexto actual

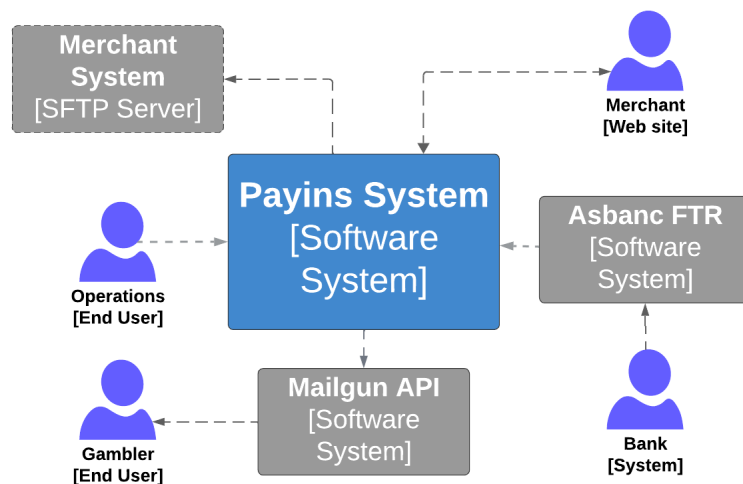
Se diagramó la arquitectura sistema de depósitos como se encontraba durante la ocurrencia de los problemas utilizando el modelo de arquitectura C4 hasta el nivel 2:

A. Nivel 1

En este nivel (ver **Figura 16**) se busca diagramar como es el contexto del sistema de depósitos con respecto a su comunicación con los usuarios y sistemas externos:

Figura 16

Arquitectura C4 As-Is Nivel 1



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Diagramas de “FakeIt”.

A continuación, se hace una descripción de los componentes en el diagrama:

1. **Payins System:** es el Sistema de Depósitos de “Rapiditos”.
2. **Merchant:** son los sitios web de las casas de apuestas que usan “Rapiditos” como medio de pago en línea.
3. **Merchant System:** es el sistema FTP para recibir lotes de depósitos para contabilidad.

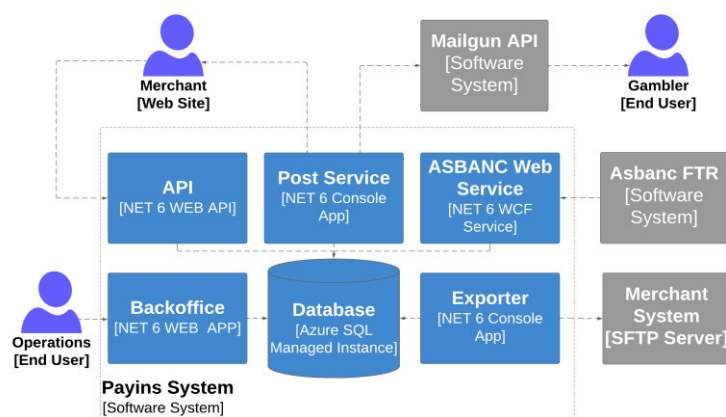
4. **Gambler:** son los clientes finales, que reciben notificaciones y apuestan en los sitios web de las casas de apuestas.
5. **Operations:** es el equipo de operaciones de “Rapiditos” y las casas de apuestas.
6. **Mailgun API:** es el sistema de envío de correos para notificar a los clientes finales.
7. **Asbanc FTR:** sistema de la Asociación de Bancos del Perú que sirve como intermediario entre los bancos y el sistema de depósitos.
8. **Bank:** son los bancos del Perú que están dentro de ASBANC y se integran a su sistema FTR.

B. Nivel 2

En la **Figura 17** se muestra cuáles son los contenedores que existen en el sistema de depósitos, cuáles son las relaciones entre ellos y quiénes se relacionan con los usuarios y sistemas externos:

Figura 17

Arquitectura C4 As-Is Nivel 2



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Diagramas de “FakeIt”.

A continuación, se hace una descripción de los componentes en el diagrama:

1. **API:** es un API de genera los vouchers de pago y muestra la página de instrucciones de pago y es llamada desde el sitio web de las casas de apuestas.
2. **Post Service:** es una tarea que se ejecuta cada 2 minutos y procesa por lotes. Manda notificaciones de los vouchers, expira vouchers vencidos y acredita los vouchers pagados a las cuentas de los clientes finales.
3. **ASBANC Web Service:** es un servicio WCF que es usado por el sistema FTR de ASBANC para hacer consultas y reportar pagos.
4. **Backoffice:** es una aplicación web donde operaciones puede gestionar los depósitos a través de un portal.
5. **Exporter:** es una tarea que envía archivos con lotes de depósitos a las casas de apuestas que usan “Rapiditos” como medio de pago en línea.
6. **Database:** es la base de datos del sistema de depósitos.

1.2. Paso 2. Determinar las responsabilidades de cada componente

Cada componente tiene asociado una lista de responsabilidades, la **Tabla 9** muestra el trabajo que el equipo elaboró una descripción de qué responsabilidades dentro del sistema tenía cada componente:

Tabla 9

Componentes As-Is

Componente	Tipo	Infraestructura		Responsabilidad
API	API	Azure Service	App	Creación de vouchers.
				Consulta de vouchers.
				Generación de página de instrucciones de pago.
				Consulta de vouchers.
Asbanc Service	Web API	Azure Kubernetes Service		Reporte de vouchers como pagados.
				Anulación de vouchers sin pagar.
				Envío de notificación de creación de voucher.
				Acreditación de vouchers pagados.
Post Service	Tarea	Azure Kubernetes Service		Envío de notificación de confirmación de voucher.
				Expiración de vouchers caducados.
				Almacenamiento de datos de los vouchers.
				Almacenamiento de datos de las operaciones.
Database	Base de Datos	Azure Server Managed Instance	SQL	Marcación de vouchers como pagados manualmente.
				Generación de reportes.
				Búsquedas de vouchers.
Backoffice	Web	Azure Kubernetes Service		

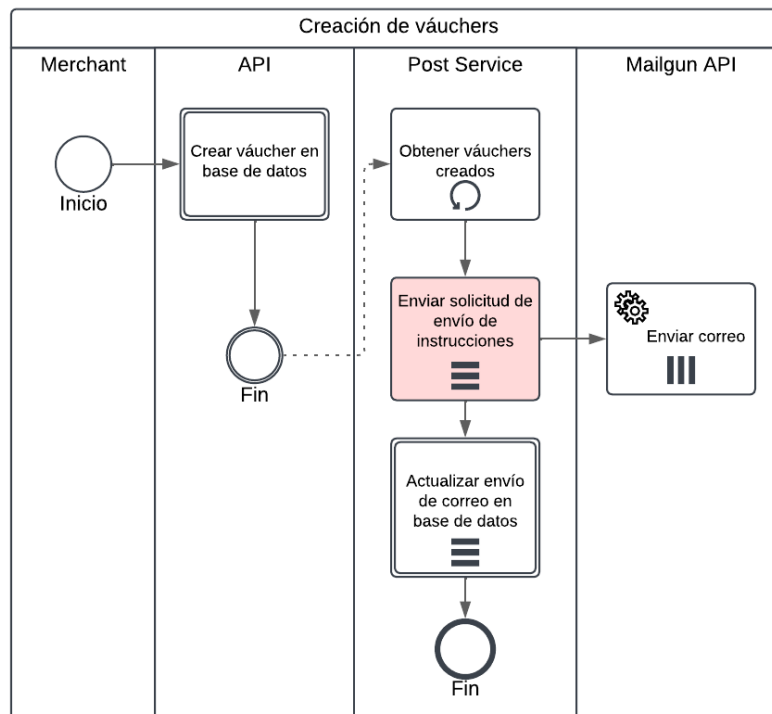
			Generación de reportes de vouchers.
		Azure	
Exporter	Tarea	Kubernetes Service	Envío de reportes por medio de SFTP a las casas de apuestas.

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Elaboración propia.

Luego de analizar las responsabilidades de cada componente podemos elaborar los siguientes diagramas de flujo:

1.2.1. Creación de vouchers

Figura 18
Creación de vouchers AS-IS

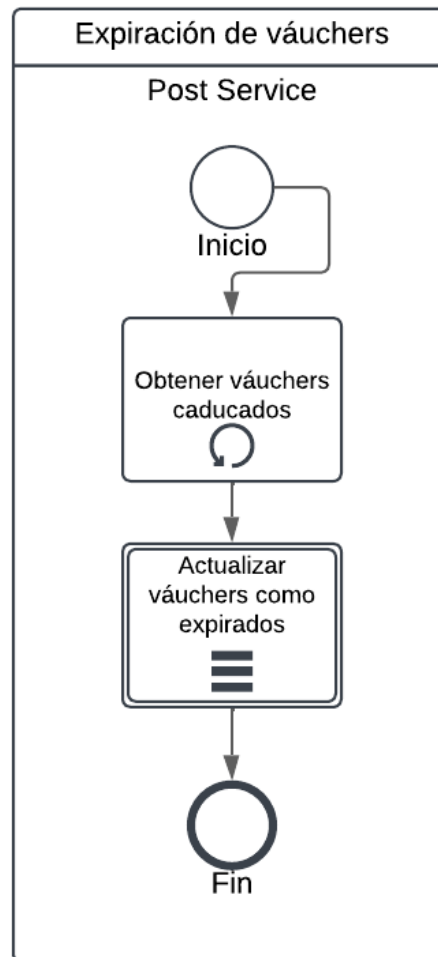


Nota. Datos tomados de “FakeIt” 2024.
Fuente: Diagramas de “FakeIt”.

1.2.2. Expiración de vouchers

Figura 19

Expiración de vouchers AS-IS

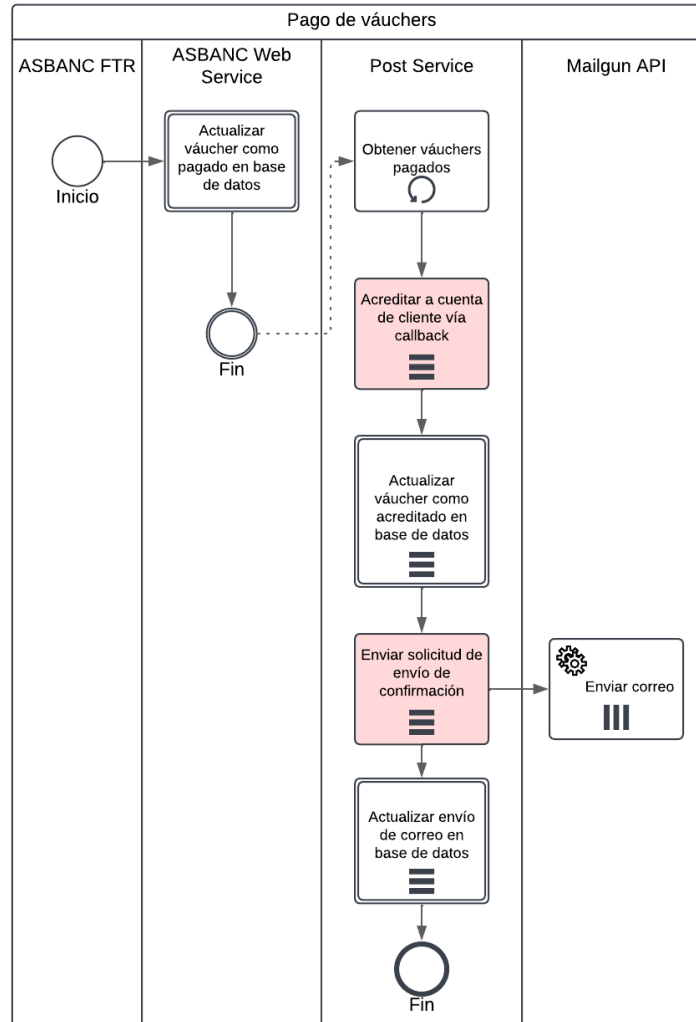


Nota. Datos tomados de “FakeIt” 2024.
Fuente: Diagramas de “FakeIt”.

1.2.3. Pago de vouchers

Figura 20

Pago de vouchers AS-IS



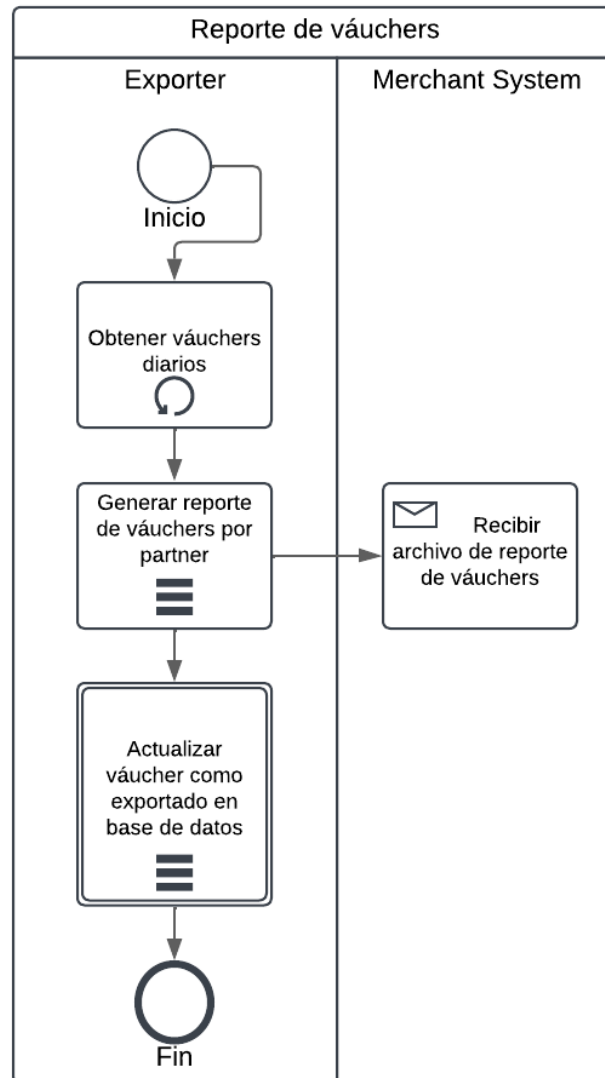
Nota. Datos tomados de “FakeIt” 2024.

Fuente: Diagramas de “FakeIt”.

1.2.4. Reporte de vouchers

Figura 21

Reporte de vouchers AS-IS



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Diagramas de “FakeIt”.

En base al análisis se encontró que los procesos de envío de notificaciones y acreditaciones eran responsabilidad del componente *Post Service* que era una aplicación de consola que se ejecutaba como una tarea programada cada 2 minutos.

1.3. Paso 3. Identificar los componentes que causaron el problema

Una vez se tenía la relación de las responsabilidades de componentes fue más sencillo poder detectar que componente en específico causaba cada problema específico y realizar la **Tabla 10**:

Tabla 10

Problemas por cada componente

Componente		Problema	Responsabilidad
API		No es causante.	Creación de vouchers. Consulta de vouchers. Generación de página de instrucciones de pago.
Asbanc Service	Web	No es causante.	Consulta de vouchers. Reporte de vouchers como pagados. Anulación de vouchers sin pagar.
Post Service		Problema 1 Problema 2	Envío de notificación de creación de voucher. Acreditación de vouchers pagados. Envío de notificación de confirmación de voucher.
Database		No es causante.	Expiración de vouchers caducados. Almacenamiento de datos de los vouchers. Almacenamiento de datos de las operaciones.
Backoffice		No es causante.	Marcación de vouchers como pagados manualmente.

		Generación de reportes. Búsquedas de vouchers. Generación de reportes de vouchers.
Exporter	No es causante.	Envío de reportes por medio de SFTP a las casas de apuestas.

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

Dado el análisis se descubrió que los problemas de retraso en el componente *Post Service* era causado por un procesamiento en lotes de 100 de los vouchers que se realizaba de manera secuencial (ver **Anexo G**) causando que se formaran colas por cada proceso y que no se pueda ejecutar un proceso sin que termine el anterior.

Se realizó un estudio del tiempo promedio que tomaba cada proceso del componente *Post Service* en los logs de la aplicación y se encontró lo siguiente:

Tabla 11

Tiempos de procesos de Post Service

Proceso	Tiempo promedio por lote de 100	Tiempo promedio por unidad
Envío de notificaciones de creación de voucher	50 segundos	0.5 segundos
Acreditación de vouchers pagados	2 minutos	2 segundos
Envío de notificaciones de confirmación de voucher.	50 segundos	0.5 segundos
Expiración de vouchers caducados	50 segundos	0.5 segundos

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

Haciendo una estimación de que haya 10 vouchers creados por segundo durante un minuto, se tendrían 600 vouchers creados en cada minuto, por lo cual se necesitaría que el componente *Post Service* se ejecute 6 veces para terminar de procesar todos los vouchers. Por lo que el último voucher en crearse demoraría 5 minutos en notificarse al cliente en un escenario ideal, sin tomar en cuenta el tiempo de las consultas a la base de datos, el intervalo de 2 minutos entre cada ejecución ni que haya otros procesos que se realicen durante la ejecución de la tarea.

1.4. Paso 4. Determinar la criticidad de cada componente

Ya que sabíamos que el componente responsable del problema era el *Post Service* debíamos también analizar que componentes se vinculaban directamente con sus procesos y determinar su criticidad para el rediseño.

Tabla 12

Criticidad de componentes

Componente	Responsabilidad	Criticidad
Creación de vouchers.		
API	Consulta de vouchers.	Alta
	Generación de página de instrucciones de pago.	
	Consulta de vouchers.	
Asbanc Web Service	Reporte de vouchers como pagados.	Alta
	Anulación de vouchers sin pagar.	
	Envío de notificación de creación de voucher.	
Post Service	Acreditación de vouchers pagados.	Muy alta
	Envío de notificación de confirmación de voucher.	
	Expiración de vouchers caducados.	

Database	Almacenamiento de datos de los vouchers.	Baja
	Almacenamiento de datos de las operaciones.	
Backoffice	Marcación de vouchers como pagados manualmente.	Baja
	Generación de reportes.	
	Búsquedas de vouchers.	
Exporter	Generación de reportes de vouchers.	Baja
	Envío de reportes por medio de SFTP a las casas de apuestas.	

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

En el análisis se determinó que el componente que registraba los vouchers creados era el *API* y el componente que encolaba las acreditaciones era el *Asbanc Web Service*. Por lo cual, se tomaron junto con el *Post Service* para aplicar un rediseño del sistema.

2. Diseñar la solución

Para el diseño de la solución primeramente se hizo una lluvia de ideas donde el equipo de depósitos decidiría que soluciones eran viables:

Figura 22

Lluvia de ideas para solución



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Jamboard de “FakeIt”.

Durante de la lluvia de ideas, el equipo de depósitos también tuvo en cuenta las siguientes restricciones:

- Restricción 1.** Al *API* no se le podía poner más carga de trabajo ya que por el SLA⁵ las solicitudes debían responder en menos de 2 segundos.
- Restricción 2.** Al *Asbanc Web Service* no se le podía poner más carga de trabajo ya que por normativa de ASBANC las solicitudes debían responder en menos de 3 segundos.

⁵ **Service Level Agreement (SLA)** es un contrato entre un proveedor de servicios y un cliente, en el que se establecen los niveles de calidad, disponibilidad y desempeño del servicio que se espera.

Se realizó un análisis comparativo entre las diferentes alternativas para solucionar el problema:

Tabla 13

Alternativas de solución

Alternativa	Ventajas	Desventajas
Dividir el <i>Post Service</i> en diferentes tareas y procesar en paralelo.	Se elimina la cola de espera entre procesos.	No ejecuta los procesos a demanda. No se puede escalar.
Enviar las notificaciones en el <i>API</i> y acreditar los pagos en el <i>Asbanc Web Service</i>	Se elimina la cola de espera entre procesos. Se ejecutan los procesos a demanda.	Las restricciones no lo permiten.
Migrar la arquitectura a una basada en eventos	Se elimina la cola de espera entre procesos. Se ejecutan los procesos a demanda. Se tiene experiencia previa. Es altamente escalable. Mensajes se persisten.	Complejo de implementar. Creación de más componentes.
Migrar la arquitectura a microservicios	Se elimina la cola de espera entre procesos. Se ejecutan los procesos a demanda. Es altamente escalable	Complejo de implementar. Creación de más componentes. Posible pérdida de mensajes.

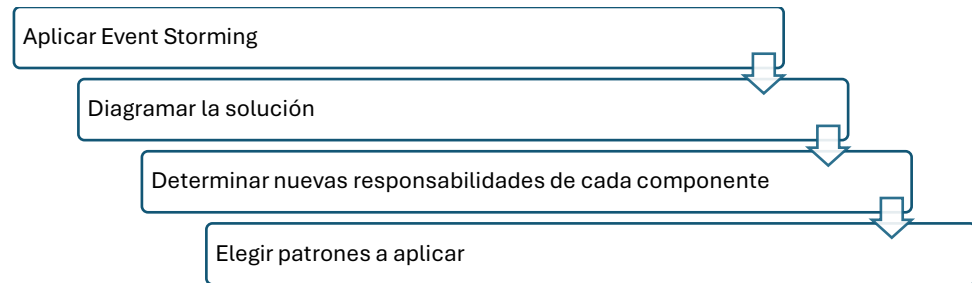
Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

Según las alternativas de solución en la **Tabla 13** el equipo decidió elegir migrar la arquitectura a una basada en eventos, por lo cual se siguieron los siguientes pasos:

Figura 23

Pasos del diseño de solución del problema



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia

2.1. Aplicar Event Storming

Para el rediseño de la arquitectura de software actual a una arquitectura basada en eventos se eligió la metodología de Event Storming, ya que era la más adecuada para tener las diferentes perspectivas de todo el equipo y con eso poder definir mejor los contextos. Además, el equipo de depósitos eligió desarrollar la metodología en el nivel de Diseño para poder tener un resultado más técnico y orientada a construir la solución a detalle.

El desarrollo de la aplicación de la metodología contó con los siguientes pasos:

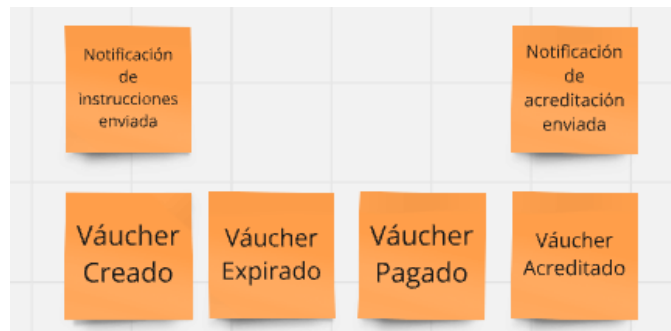
2.1.1. Descubrimiento de eventos claves

Se descubrieron los siguientes eventos:

1. Váucher creado
2. Váucher expirado
3. Váucher pagado
4. Váucher acreditado
5. Notificación de instrucciones enviada
6. Notificación de acreditación enviada

Figura 24

Fase 1 de Event Storming Aplicada



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Tablero de “FakeIt”.

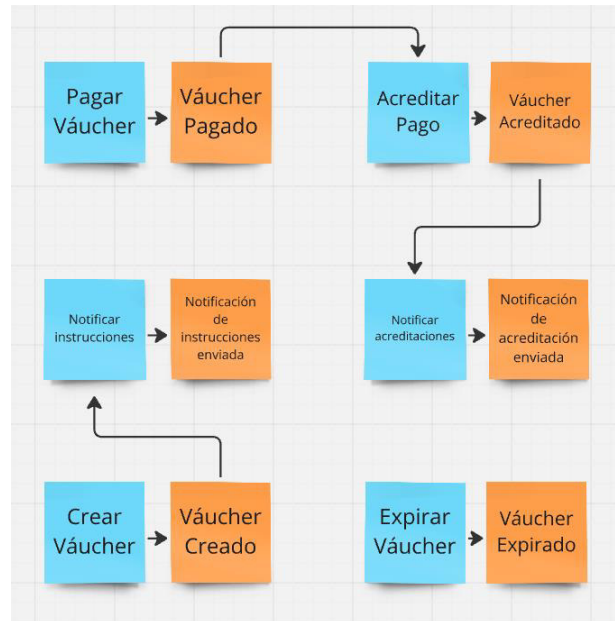
2.1.2. Identificación de comandos

Se identificaron los siguientes comandos:

1. Crear váucher, que genera el evento váucher creado.
2. Expirar váucher, que genera el evento váucher expirado.
3. Pagar váucher, que genera el evento váucher pagado.
4. Acreditar pago, que genera el evento váucher acreditado y debería ejecutarse al haber un váucher pagado.
5. Notificar instrucciones, que genera el evento de notificación de instrucciones enviada y debería ejecutarse al haber un váucher creado.
6. Notificar acreditaciones, que genera el evento de notificación de acreditación enviada y debería ejecutarse al haber un váucher acreditado.

Figura 25

Fase 2 de Event Storming Aplicada



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Tablero de “FakeIt”.

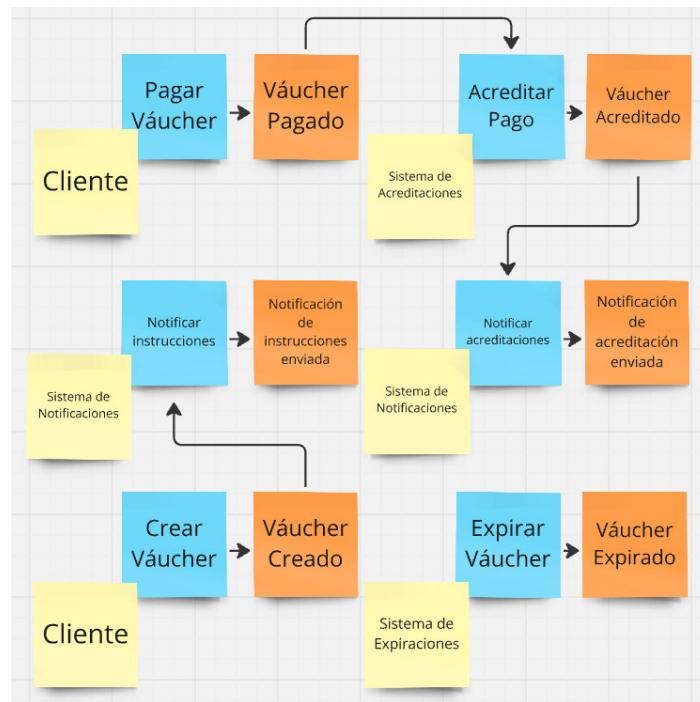
2.1.3. Identificación de actores

Se identificaron los siguientes actores:

1. Cliente, que el apostador en la casa de apuesta que quiere depositar en su cuenta.
2. Sistema de notificaciones, que debería ser el encargado de enviar notificaciones de correo.
3. Sistema de acreditaciones, que debería ser el encargado de acreditar los pagos en las cuentas de los clientes.
4. Sistema de expiraciones, que debería ser el encargado de expirar los vouchers caducados.

Figura 26

Fase 3 de Event Storming Aplicada



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Tablero de “FakeIt”.

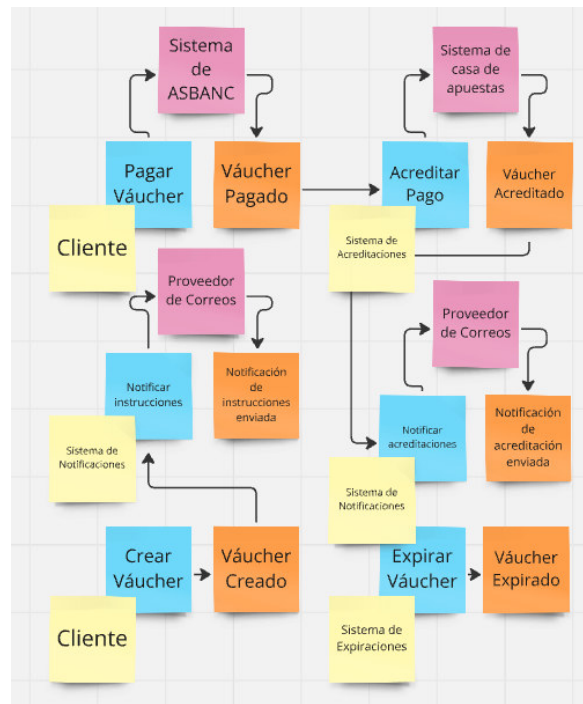
2.1.4. Identificación de sistemas externos

Se identificaron los siguientes sistemas externos:

1. Proveedor de correos, que según la **Figura 16** es *Mailgun Api* y se encarga de la gestión de envío de correos.
2. Sistema de ASBANC, que según la **Figura 16** es *Asbanc FTR* y se encarga de reenviar los pagos de los clientes desde su banco hasta nuestro sistema.
3. Sistema de casa de apuesta, que según la **Figura 16** es *Merchant System* y se encarga de recibir las acreditaciones y abonarlas a las cuentas de los clientes.

Figura 27

Fase 4 de Event Storming Aplicada



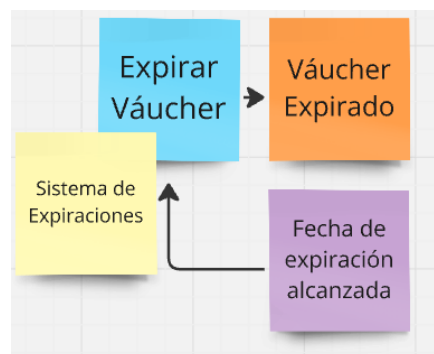
Nota. Datos tomados de “FakeIt” 2024.
Fuente: Tablero de “FakeIt”.

2.1.5. Identificación de políticas

Solo se identificó la política de expiración por no pagar hasta la fecha límite.

Figura 28

Fase 5 de Event Storming Aplicada



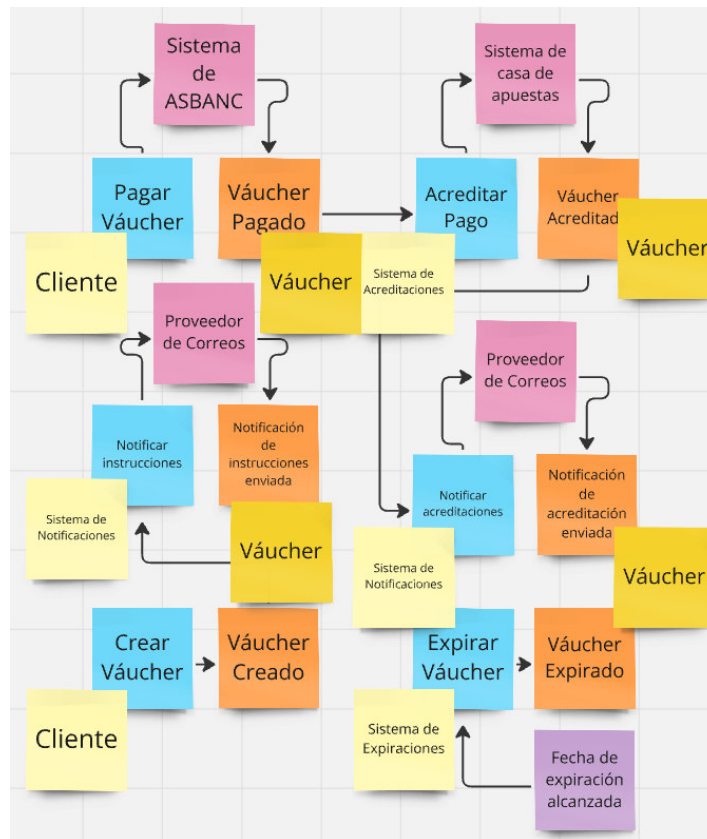
Nota. Datos tomados de “FakeIt” 2024.
Fuente: Tablero de “FakeIt”.

2.1.6. Determinación de agregados y entidades

Solo se determinó el agregado de v  cher.

Figura 29

Fase 6 de Event Storming Aplicada



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Tablero de “FakeIt”.

2.1.7. Determinaci  n de proyecciones

No se determinaron proyecciones para este sistema.

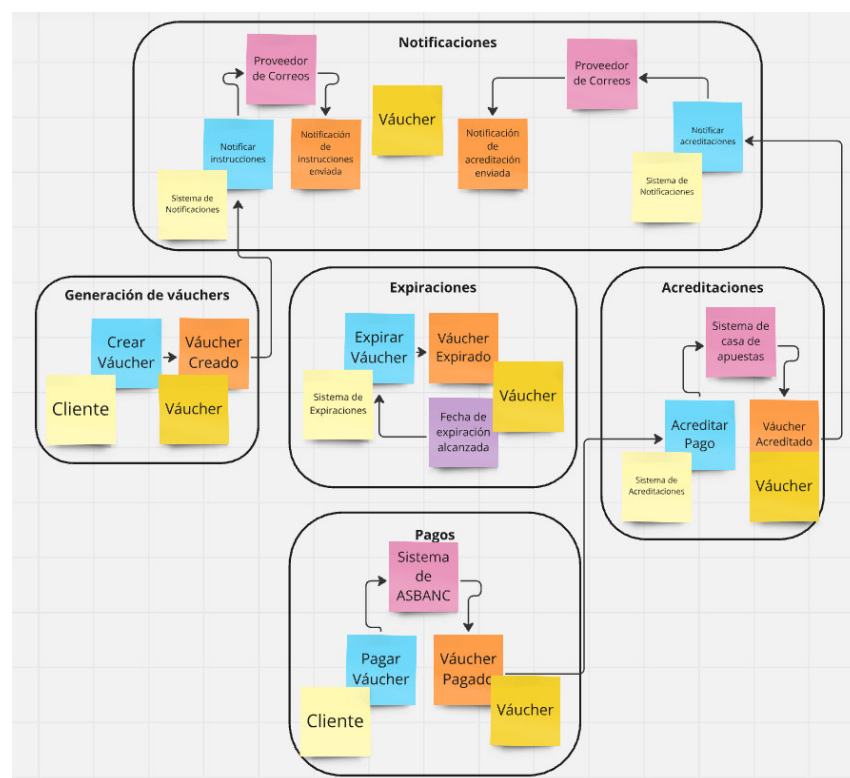
2.1.8. Organizaci  n de contextos

Se organizaron 5 contextos de acuerdo con los diferentes procesos del sistema:

1. **Generación de vouchers:** Este contexto se encarga de registrar los vouchers en el sistema.
2. **Notificaciones:** Este contexto se encarga del envío de correos por medio del proveedor de correos ante eventos de creación y acreditación de vouchers.
3. **Pagos:** Este contexto se encarga de recibir los pagos desde ASBANC y actualizar los vouchers.
4. **Acreditaciones:** Este contexto se encarga de acreditar los vouchers pagados a los sistemas de las casas de apuestas.
5. **Expiraciones:** Este contexto se encarga de expirar los vouchers caducados.

Figura 30

Fase 8 de Event Storming Aplicada



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Tablero de “FakeIt”.

De la aplicación de la metodología de Event Storming se pudieron sacar las siguientes conclusiones:

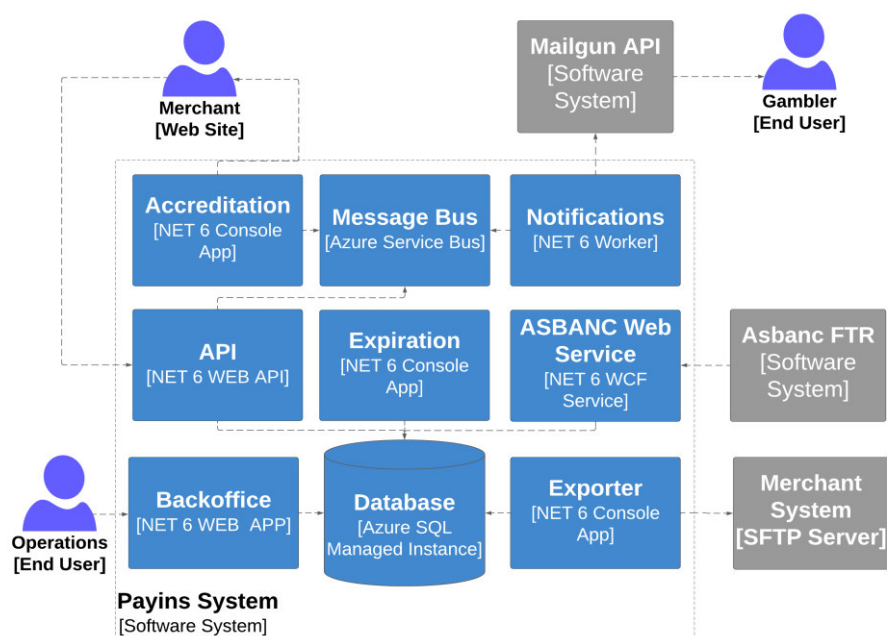
1. Los contextos de generación de vouchers, expiraciones y pagos se desarrollan en diferentes momentos en el tiempo.
2. El contexto de notificaciones debe desarrollarse inmediatamente después de generación de vouchers y acreditaciones.
3. El contexto de acreditaciones debe desarrollarse inmediatamente después de pagos.
4. El contexto de expiraciones está aislado de los demás contextos y se puede programar para que sea una vez al día durante horarios con pocas operaciones.
5. Los demás contextos se comunican entre sí mediante los eventos de dominio, por lo cual se necesitará un bróker de mensajería.
6. El proceso de creación de vouchers se debe separar en la generación de vouchers y notificaciones.
7. El proceso de pago de vouchers se debe separar en pagos, acreditaciones y notificaciones.

1.2.1. Diagramar la solución

Teniendo en cuenta la **Figura 17** y el resultado de aplicar Event Storming el equipo de depósitos diagramó el sistema según el modelo C4 de nivel 2:

Figura 31

Diagrama C4 To-Be Nivel 2



Nota. Datos tomados de “FakeIt” 2024.

Fuente: Diagramas de “FakeIt”.

A continuación, se hace una descripción de los componentes en el diagrama y sus cambios:

1. **API:** se mantiene igual pero adicionalmente deberá publicar eventos de vouchers creados.
2. **ASBANC Web Service:** se mantiene igual pero adicionalmente deberá publicar eventos de vouchers pagados.
3. **Message Bus:** es el bróker de mensajería por colas que manejará los eventos de los vouchers.
4. **Backoffice:** se mantiene igual.
5. **Exporter:** se mantiene igual.
6. **Expiration:** es una tarea que expirará los vouchers caducados.

7. **Accreditation:** es la aplicación que consumirá los eventos de vouchers pagados y los acreditará a las cuentas de los clientes finales.
8. **Notifications:** es una aplicación que consumirá los eventos de vouchers creados y confirmados y los notificará a los clientes finales vía correo electrónico.
9. **Database:** se mantiene igual.

1.2.2. Determinar las nuevas responsabilidades de cada componente

Dada la **Figura 31** se modificaron las responsabilidades de cada componente de acuerdo con la siguiente tabla:

Tabla 14

Componentes To-Be

Componente	Tipo	Infraestructura	Responsabilidad
API	API	Azure Service	App
			Creación de vouchers.
			Consulta de vouchers.
			Generación de página de instrucciones de pago.
Asbanc Service	Web API	Azure Kubernetes Service	Publicación de eventos de vouchers creados.
			Consulta de vouchers.
			Reporte de vouchers como pagados.
			Anulación de vouchers sin pagar.
Message Bus	Bus	Azure Service Bus	Publicación de eventos de vouchers pagados.
			Gestión de eventos de vouchers.
Notifications	Worker	Azure Kubernetes Service	Consumo de eventos de vouchers creados.

				Envío de notificación de creación de vóucher.
				Consumo de eventos de vóuchers confirmados.
				Envío de notificación de confirmación de vóucher.
Accreditation	Worker	Azure Kubernetes Service		Acreditación de vóuchers pagados.
				Publicación de vóuchers confirmados.
Expiration	Tarea	Azure Kubernetes Service		Expiración de vóuchers caducados.
Database	Base de Datos	Azure Server Managed Instance	SQL	Almacenamiento de datos de los vóuchers.
				Almacenamiento de datos de las operaciones.
				Marcación de vóuchers como pagados manualmente.
Backoffice	Web	Azure Kubernetes Service		Generación de reportes.
				Búsquedas de vóuchers.
				Reenvío de notificaciones.
				Generación de reportes de vóuchers.
Exporter	Tarea	Azure Kubernetes Service		Envío de reportes por medio de SFTP a las casas de apuestas.

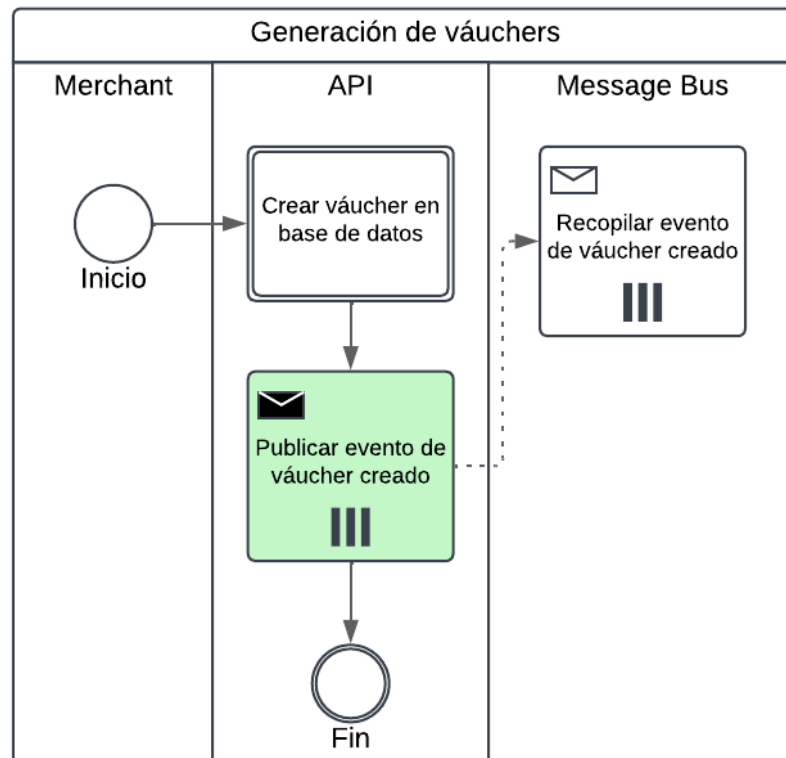
Nota. Datos tomados de “FakeIt” 2024.
Fuente: Elaboración propia.

De acuerdo con el análisis, los procesos del sistema se tendrían que actualizar de acuerdo con los siguientes diagramas:

1.2.2.1. Generación de vouchers

Figura 32

Generación de vouchers TO-BE

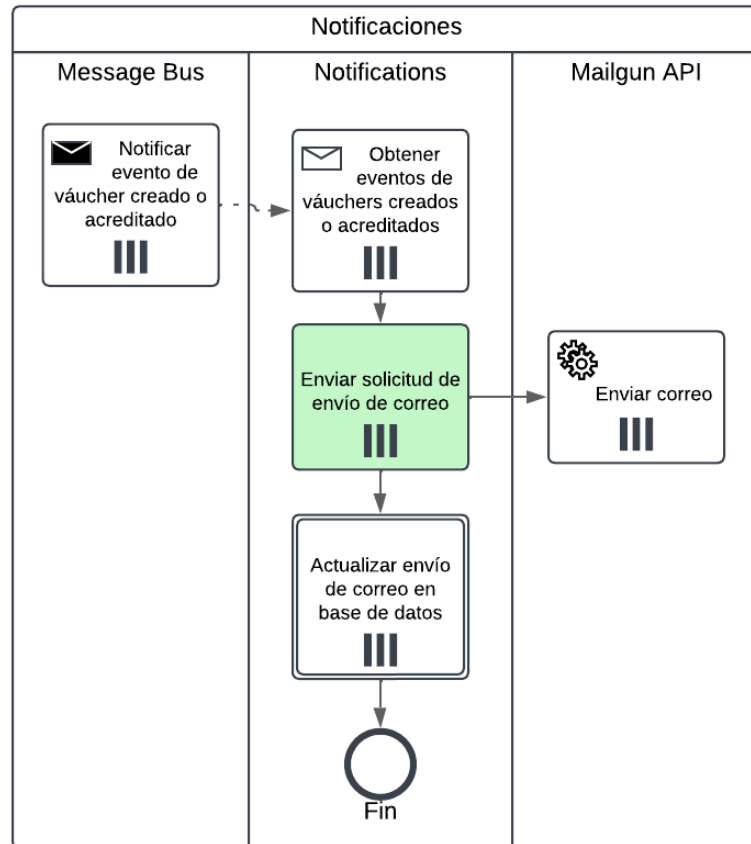


Nota. Datos tomados de “FakeIt” 2024.

Fuente: Diagramas de “FakeIt”.

1.2.2.2. Notificaciones

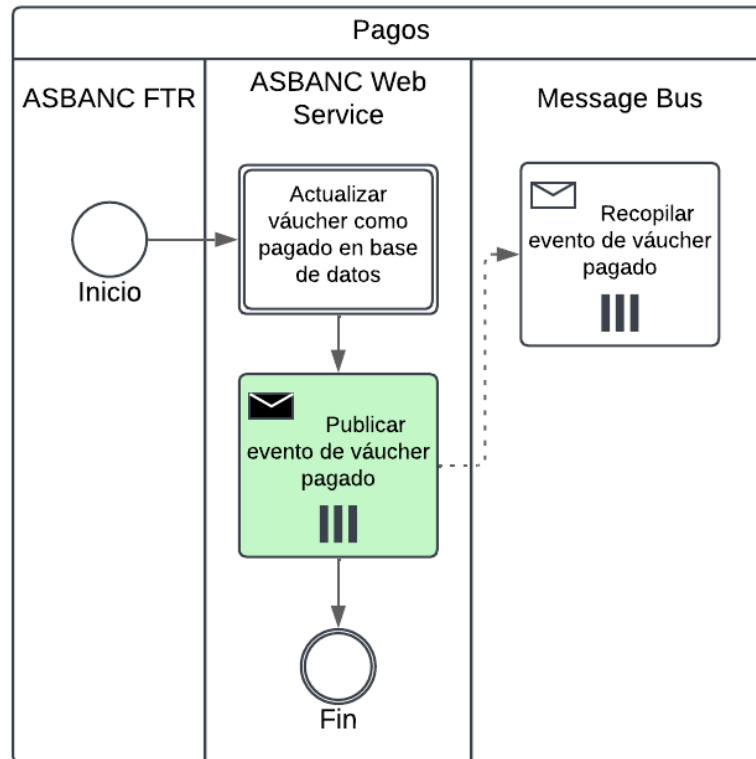
Figura 33
Notificaciones TO-BE



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Diagramas de “FakeIt”.

1.2.2.3.Pagos

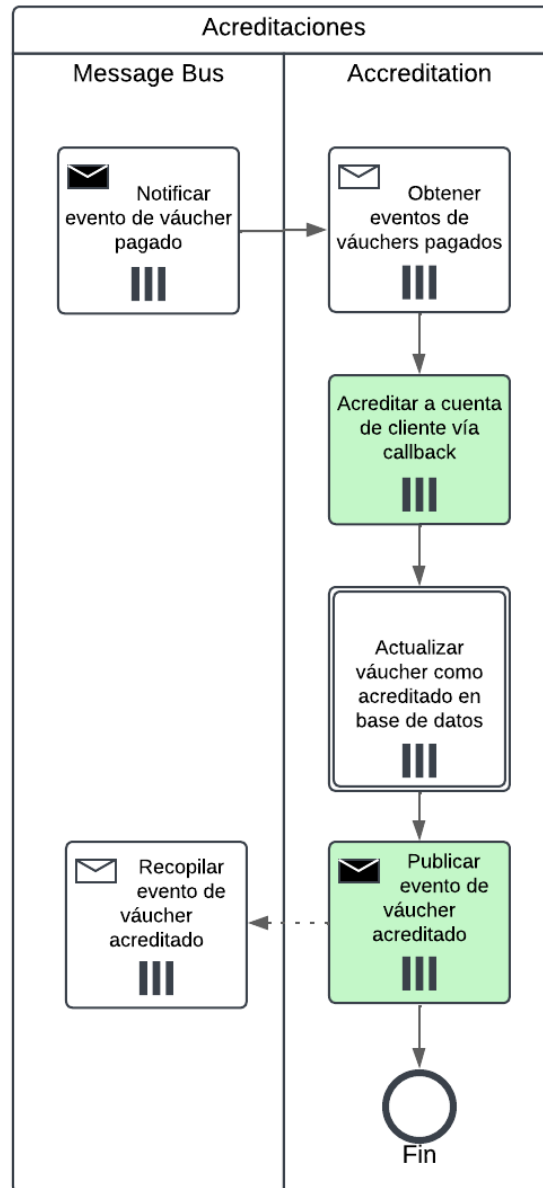
Figura 34
Pagos TO-BE



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Diagramas de “FakeIt”.

1.2.2.4.Accreditaciones

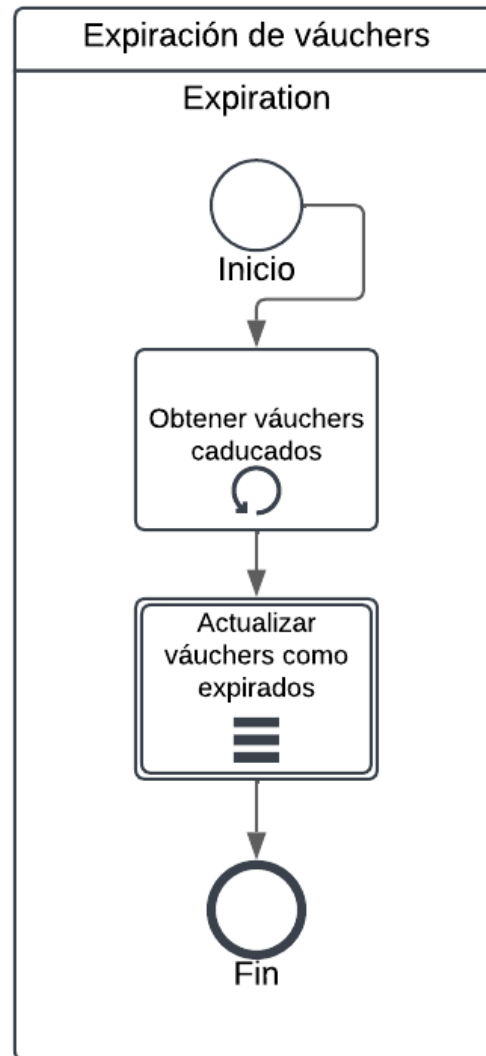
Figura 35
Accreditaciones TO-BE



Nota. Datos tomados de "FakeIt" 2024.
Fuente: Diagramas de "FakeIt".

1.2.2.5.Expiraciones

Figura 36
Expiraciones TO-BE



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Diagramas de “FakeIt”.

1.2.3. Elegir patrones a aplicar

Dados los eventos determinados en la **Figura 24**, se determinaron los componentes que actuarían como publicadores y subscriptores:

Tabla 15

Publicador y subscriptores por evento

Evento	Publicador	Subscriptores
Váucher creado	API	Notification
Correo de instrucciones enviado	Notification	No necesita
Váucher expirado	Expiration	No necesita
Váucher pagado	ASBANC Web Service	Accreditation
Váucher acreditado	Accreditation	Notification
Correo de confirmación de pago enviado	Notification	No necesita

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Elaboración propia.

Según el análisis de la **Tabla 15** se determinó que cada evento solo tiene un solo subscriptor por lo cual el tipo de bróker de mensajería debía ser uno por colas y no por tópicos.

Luego, había que determinar el patrón de eventos que debería utilizarse por lo cual se hizo una comparación:

Tabla 16

Alternativas de patrones de evento

Patrón	Ventajas	Desventajas
Event Notification	Mensaje ligero.	Requiere consulta a base de datos. Alta Latencia de consumo.
Event Carried State	Evita consulta a base de datos. Baja latencia de consumo. Simple.	Mensaje pesado.
Event Sourcing	Permite reconstruir estados. Alta trazabilidad.	Complejo en implementar. Complejo en persistir.

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Elaboración propia.

Después del análisis de las alternativas en la **Tabla 16** el equipo decidió elegir el patrón Event Carried State debido a que tenía baja latencia de consumo ya que evitaba consultas extras a la base de datos.

Finalmente, debido a que la arquitectura basada en eventos tiene la fortaleza de poder escalar sus componentes para manejar concurrentemente los eventos y escalar horizontalmente se determinó que los siguientes componentes necesitaban poder escalar:

1. API
2. Notifications Worker
3. Accreditation Worker
4. ASBANC Web Service

Para cada aplicación se siguió una estrategia de acuerdo con la siguiente tabla:

Tabla 17

Estrategias de escalamiento

Componente	Estrategia	Tipo de Escalado	Métrica de referencia	Límite
API	Escalamiento manual	Horizontal	RPS ⁶	10
Notifications Worker	Escalamiento automático	Horizontal	CPU %	85
Accreditation Worker	Escalamiento automático	Horizontal	CPU %	85
ASBANC Web Service	Escalamiento automático	Horizontal	CPU %	85

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Elaboración propia.

⁶ RPS (Requests per second): Solicitudes por segundo.

3. Implementar la solución

Para la implementación de la solución se actualizaron y crearon los siguientes componentes:

3.1. API

Se agregó la funcionalidad de la publicación del evento v  cher creado:

Figura 37

Publicaci  n de v  cher creado en API

```
= Task.Run(async () =>
{
    try
    {
        await _messageBusSender.SendMessageAsync(objQueue, _config.GetValue<string>("AzureQueue:QueueName"));
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Error - {voucher.payment_id} - {ex.Message}");
        throw new HandlerException(HttpStatusCode.InternalServerError);
    }
});
```

Nota. Datos tomados de “FakeIt” 2024.
Fuente: C  digo fuente de “FakeIt”.

3.2.ASBANC Web Service

Se agregó la funcionalidad de la publicaci  n del evento v  cher pagado:

Figura 38

Publicaci  n de v  cher pagado en Web Service

```
= Task.Run(async () =>
{
    try
    {
        await _messageBusSender.SendMessageAsync(vQueue, _configuration.GetValue<string>("AzureQueue:QueueName"));
    }
    catch (Exception)
    {
        throw;
    }
});
```

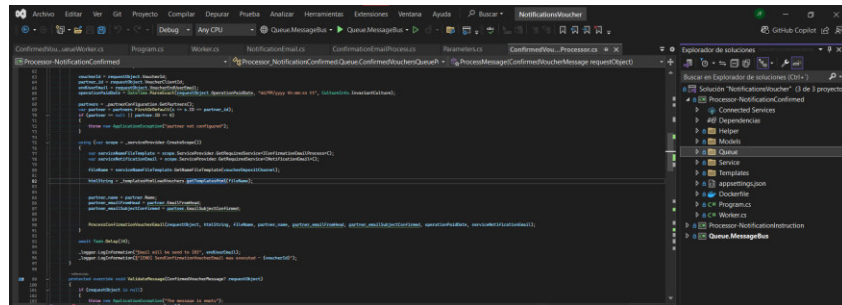
Nota. Datos tomados de “FakeIt” 2024.
Fuente: C  digo fuente de “FakeIt”.

3.3.Notifications

Se desarroll   el componente para cumplir con las responsabilidades detalladas en la **Tabla 14**.

Figura 39

Fragmento de código de Notifications



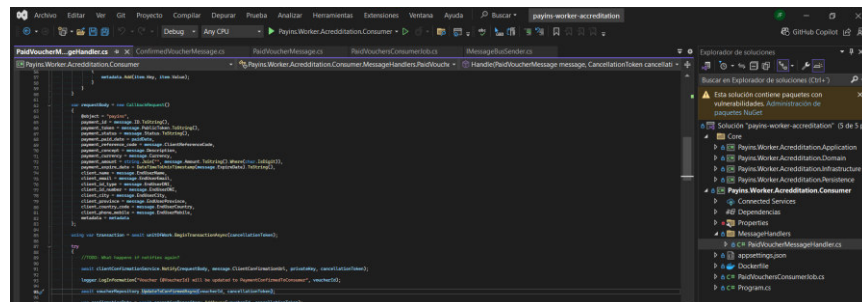
Nota. Datos tomados de “FakeIt” 2024.
Fuente: Código fuente de “FakeIt”.

3.4. Accreditation

Se desarrolló el componente para cumplir con las responsabilidades detalladas en la **Tabla 14**.

Figura 40

Fragmento de código de Accreditation



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Código fuente de “FakeIt”.

Para el caso de los componentes que tuvieron la estrategia de escalamiento automático y que estuvieran desplegados en Kubernetes se utilizó el recurso Horizontal Pod Autoscaler:

Figura 41

Recurso de escalado automático en Kubernetes para Accreditation

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  namespace: payins
  name: payins-worker-accreditation-hpa
  labels:
    app: payins-worker-accreditation
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: payins-worker-accreditation-deployment
  minReplicas: 1
  maxReplicas: 4
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 85
```

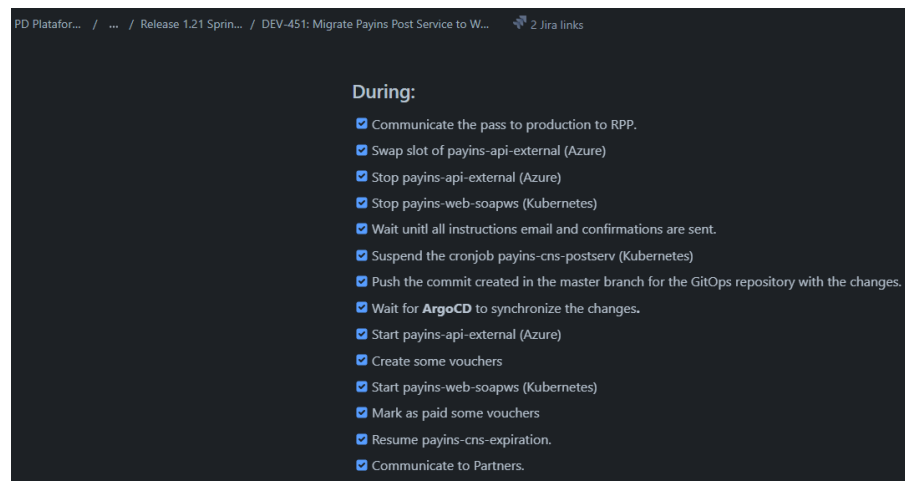
Nota. Datos tomados de “FakeIt” 2024.
Fuente: Repositorio Gitops de “FakeIt”.

4. Desplegar en Producción

Para el despliegue en el ambiente de Producción se realizó un plan con todos los pasos a seguir para la publicación y se planificó para el día 28 de junio del 2024:

Figura 42

Fragmento del plan de pase a producción



Nota. Datos tomados de “FakeIt” 2024.
Fuente: Confluence de “FakeIt”.

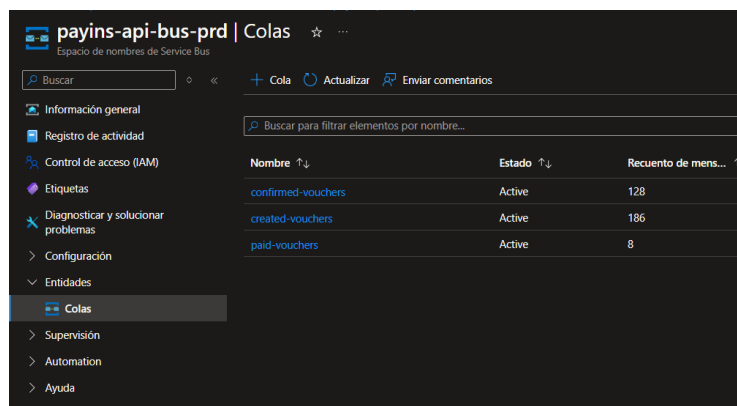
Para el día de la publicación en el ambiente de Producción se desplegaron los componentes:

1. Message Bus

Para la cola de eventos se desplegó un recurso de Azure Service Bus⁷ ya que este soporta el tipo de bróker por colas, concurrencia para consumir mensajes y es de fácil integración con los otros recursos en Azure:

Figura 43

Fragmento de recurso Azure Service Bus en Producción



payins-api-bus-prd Colas		
Espacio de nombres de Service Bus		
Buscar para filtrar elementos por nombre...		
Nombre ↑↓	Estado ↑↓	Recuento de mens... ↑↓
confirmed-vouchers	Active	128
created-vouchers	Active	186
paid-vouchers	Active	8

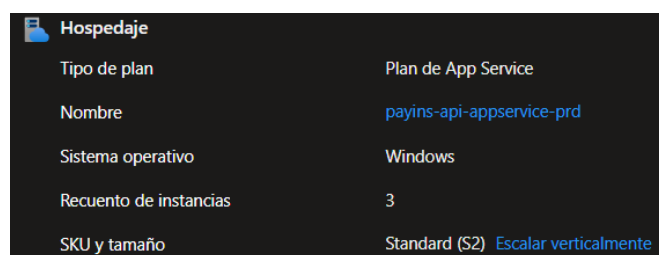
Nota. Datos tomados de “FakeIt” 2024.
Fuente: Suscripción de Azure de “Rapiditos”.

2. API

El despliegue de la aplicación se realizó mediante la publicación en recurso de Azure App Service⁸:

Figura 44

Fragmento de recurso Azure de API en Producción



Hospedaje	
Tipo de plan	Plan de App Service
Nombre	payins-api-appservice-prd
Sistema operativo	Windows
Recuento de instancias	3
SKU y tamaño	Standard (S2) Escalar verticalmente

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Suscripción de Azure de “Rapiditos”.

⁷ **Azure Service Bus**: un sistema de mensajería en la nube de Azure para conectar aplicaciones y dispositivos en nubes públicas y privadas.

⁸ **Azure App Service**: es un servicio basado en HTTP para hospedar aplicaciones web, API REST y back-ends para dispositivos móviles en la nube de Azure.

3. ASBANC Web Service

El despliegue de la aplicación se realizó mediante el repositorio GitOps⁹ para el despliegue automático de los recursos en Kubernetes:

Figura 45

Fragmento de manifiesto de despliegue de Web Service

```
imagePullSecrets:
- name: docker-secret
containers:
- name: payins-web-soapws-container
  image: rpppdcontainerregistry.azurecr.io/payins/payins-web-soapws:1b7dd03
  imagePullPolicy: IfNotPresent
  resources:
    limits:
      memory: "300Mi"
      cpu: "250m"
    requests:
      memory: "100Mi"
      cpu: "125m"
  ports:
    - containerPort: 80
    - containerPort: 443
  env:
    - name: ASPNETCORE_ENVIRONMENT
      value: Production
    - name: ASPNETCORE_URLS
      value: http://+:80;https://+:443
    - name: TZ
      value: America/Lima
```

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Repositorio GitOps de “FakeIt”.

4. Notifications

El despliegue de la aplicación se realizó mediante el repositorio GitOps para el despliegue automático de los recursos en Kubernetes:

Figura 46

Fragmento de manifiesto de despliegue de Notifications

```
imagePullSecrets:
- name: docker-secret
containers:
- name: payins-worker-notifications-confirmation-container
  image: rpppdcontainerregistry.azurecr.io/payins/payins-worker-notifications-confirmation:d09071b
  imagePullPolicy: IfNotPresent
  resources:
    limits:
      memory: "200Mi"
      cpu: "125m"
    requests:
      memory: "100Mi"
      cpu: "50m"
  env:
    - name: ASPNETCORE_ENVIRONMENT
      value: Production
```

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Repositorio GitOps de “FakeIt”.

⁹ **Gitops**: usa repositorios de código de Git para gestionar las implementaciones de código de infraestructura y aplicaciones.

5. Accreditations

El despliegue de la aplicación se realizó mediante el repositorio GitOps para el despliegue automático de los recursos en Kubernetes:

Figura 47

Fragmento de manifiesto de despliegue de Accreditation

```
containers:
- name: payins-worker-accreditation-container
  image: rpppdcontainerregistry.azurecr.io/payins/payins-worker-accreditation:485a09a
  imagePullPolicy: IfNotPresent
  resources:
    limits:
      memory: "500Mi"
      cpu: "125m"
    requests:
      memory: "200Mi"
      cpu: "50m"
  env:
    - name: ASPNETCORE_ENVIRONMENT
      value: Production
    - name: TZ
      value: America/Lima
```

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Repositorio GitOps de “FakeIt”.

6. Expiration

El despliegue de la aplicación se realizó mediante el repositorio GitOps para el despliegue automático de los recursos en Kubernetes:

Figura 48

Fragmento de manifiesto de despliegue de Expiration

```
timezone: 'America/Lima'
schedule: '* * * * *'
concurrencyPolicy: Forbid
successfulJobHistoryLimit: 3
failedJobHistoryLimit: 3
jobTemplate:
  spec:
    template:
      spec:
        volumes:
        - name: payins-cns-expiration-configuration
          configMap:
            name: payins-cns-expiration-configuration
        - name: payins-cns-expiration-logs
          persistentVolumeClaim:
            claimName: payins-logs-pvc
        - name: payins-cns-expiration-files
          persistentVolumeClaim:
            claimName: payins-files-pvc
        imagePullSecrets:
        - name: docker-secret
        containers:
        - name: payins-cns-expiration-container
          image: rpppdcontainerregistry.azurecr.io/payins/payins-cns-expiration:a7fcd52
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              memory: "500Mi"
              cpu: "250m"
            requests:
              memory: "200Mi"
              cpu: "125m"
          env:
            - name: ASPNETCORE_ENVIRONMENT
              value: Production
```

Nota. Datos tomados de “FakeIt” 2024.
Fuente: Repositorio GitOps de “FakeIt”.

1.5. Evaluación

1.5.1. *Evaluación costo – beneficio*

El rediseño de la arquitectura del sistema se orientó hacia la resolución de problemas de rendimiento y capacidad, con un enfoque principal en la reducción de tiempos y en la preparación del sistema para altas cargas durante la Copa América 2024 y también a futuros eventos deportivos. A continuación, se presenta el análisis costo-beneficio basado en métricas de tiempo y beneficios intangibles:

Costos

1. Infraestructura

Al migrar el diseño de la arquitectura de software se necesitó desplegar un bróker de mensajería en la nube de Azure que cuesta 0.05 USD por millón de operaciones.

2. Interrupción de servicios

Durante la migración de la arquitectura de software tuvieron que parar las operaciones de “Rapidito” durante aproximadamente 30 minutos para desplegar los cambios y hacer pruebas internas.

3. Uso de recursos

Debido a que los componentes son escalables se estima que el uso de recursos de CPU y memoria se puede multiplicar hasta por 4 para notificaciones y hasta por 4 para acreditaciones. Al estar desplegados en Kubernetes esto puede derivar en el escalamiento de clúster a un nodo más que cuesta aproximadamente 80 USD.

Beneficios

1. Reducción de tiempos de procesamiento

1.1. Notificaciones

Para evaluar la reducción de tiempos de procesamiento en el envío de notificaciones se tomó como muestra el envío de 100 notificaciones (que toma 0.5 segundos en procesar) según los registros de las aplicaciones, debido a que no se guardan estos tiempos en la base de datos:

Tabla 18

Evaluación de tiempos de procesamiento de notificaciones

Arquitectura	Capacidad concurrente	Tiempo total en segundos	Reducción de tiempo (%)
Anterior	1	50	No aplica
Nueva (1 réplica)	10	5	90%
Nueva (2 réplicas)	20	2.5	95%
Nueva (3 réplicas)	30	2	96%
Nueva (4 réplicas)	40	1.5	97%

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Registros de *Post Service* y *Notifications*.

Según el análisis de la **Tabla 18** se observó que hubo una reducción de entre el 90% al 97% del tiempo total en procesar 100 envíos de notificaciones.

1.2.Acreditaciones

Para la evaluación de la reducción de tiempos de las acreditaciones se comparó el tiempo promedio en acreditar un pago durante la etapa final de La Liga de Campeones 2023-2024 (que ocurrió mientras estaba la antigua arquitectura) y durante la Copa América 2024 (que ocurrió mientras estaba la nueva arquitectura) ya que se contaba con los tiempos en la base de datos.

Tabla 19

Evaluación de tiempos de procesamiento de acreditaciones

Evento deportivo	Cantidad total de vouchers pagados	Tiempo promedio de acreditación en segundos	Reducción de tiempo (%)
Liga de Campeones	373269	77.24	No aplica
Copa América	181066	0.90	98.83%

Nota. Datos tomados de “FakeIt” 2024.

Fuente: Base de datos de Depósitos.

Según la **Tabla 19**, el tiempo promedio para la acreditación de un pago durante eventos deportivos con alta demanda disminuyó drásticamente de aproximadamente 1 minutos y 20 segundos a aproximadamente un segundo que es una reducción de un 98.83% del tiempo.

2. Procesamiento concurrente

2.1. Notificaciones

Según la **Tabla 18**, la cantidad de notificaciones a procesar concurrentemente aumentó de 1 a 10 con una sola réplica y a 40 con escalamiento de 4 réplicas.

2.2. Acreditaciones

Según la **Figura 41**, el componente *Accreditation* puede escalarse hasta 4 réplicas, por lo cual la capacidad de procesar concurrentemente aumentó de 1 a 10 con una sola réplica y a 40 con escalamiento de 4 réplicas.

3. Mantenimiento

El mantenimiento de los nuevos componentes se ha vuelto más sencilla ya que cada componente tiene una responsabilidad única y está en su propio repositorio.

4. Capacidad de crecimiento

El rediseño del sistema permite que el sistema crezca para soportar aún cargas más altas debido a sus políticas de escalamiento, pudiendo ajustar la cantidad máxima de réplicas.

5. Posicionamiento en el mercado

El rediseño de la arquitectura de software logró posicionar a “Rapiditos” como la empresas que acredita más rápido los pagos en las cuentas de los clientes, superando a sus competidores.

6. Satisfacción del cliente

Al tener en tiempo real las notificaciones y acreditaciones el cliente ha aumentado su confianza en “Rapiditos” ya que ve sus operaciones más rápidas, posibilitándolo seguir jugando en la casa de apuestas.

CAPÍTULO IV - REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

El desarrollo del proyecto realizado representó un gran desafío técnico y profesional que puso a prueba el conocimiento adquirido durante la formación académica en la universidad y el adquirido durante los años de experiencia profesional.

Durante la formación académica puedo resaltar que se formaron bases sólidas en los fundamentos de arquitectura de software, patrones de diseño y principios de desacoplamiento, concurrencia y paralelismo.

Además, dado que en la formación académica se realizaron proyectos en equipo, se potenciaron las habilidades blandas como el liderazgo, el trabajo en equipo, el compañerismo, la comunicación efectiva y la coordinación.

A pesar de que la base de conocimientos en la formación académica fue clave, hubo un proceso de adaptación, algunos aspectos técnicos se tuvieron que adquirir mediante el continuo aprendizaje y capacitación con las tecnologías emergentes en el mercado.

Aunque se tenía conocimiento de la arquitectura orientada a eventos, no se tuvo la oportunidad de ponerlo en práctica ni de observarlo en un ejemplo, por lo que fue un gran desafío ponerlo en práctica y aplicarlo al sector empresarial.

Conceptos clave como consistencia eventual, procesamiento asíncrono, observabilidad y contenedores me parecieron que hicieron falta en la formación académica y que son importantes aprender para estar a la vanguardia con lo requerido por empresas que cada vez llegan a más clientes.

También podría comentar que sería importante reconciliar la formación académica con el aprendizaje de computación en la nube y aprender acerca de sistemas de orquestación de contenedores como Kubernetes.

Finalmente, el liderazgo también fue crucial durante el desarrollo del proyecto ya que se tuvo que delegar eficazmente las tareas de implementación entre el equipo demostrando la capacidad para resolver inquietudes y trazar un horizonte gestionando las controversias y la resistencia al cambio. Eso también ayudó a poder establecer mejor comunicación y relación con los interesados en el proyecto para que pudieran dar su apoyo al desarrollo de este.

CAPÍTULO V - CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

El rediseño de la arquitectura de software del sistema de depósitos cumplió el objetivo general reduciendo los tiempos de los procesos durante La Copa América 2024 atendiendo eficientemente la alta carga de solicitudes, sin embargo, por la complejidad de la solución se retrasó el despliegue en producción una semana.

La implementación de esta solución permitió una respuesta rápida y escalable a las demandas del sistema, proporcionando un diseño más robusto y preparado para futuras necesidades.

Evaluación de objetivos específicos:

1. Optimizar el envío de notificaciones para soportar altas cargas

1.1.Resultados obtenidos

Durante La Copa América 2024 se observó que el envío de notificaciones de instrucciones de pago y de confirmaciones de pago se realizaban en un promedio de medio segundo, independientemente de si la carga era alta y no se presenciaron cuellos de botella.

1.2.Métricas clave

Incremento en la capacidad del procesamiento de notificaciones concurrentes.

Reducción de tiempo total para procesar gran cantidad de notificaciones.

1.3.Conclusión

El objetivo fue cumplido logrando que el proceso de envío de notificaciones esté preparado para atender altas cargas con baja latencia.

2. Optimizar las acreditaciones para soportar altas cargas

2.3.Resultados obtenidos

Durante La Copa América 2024 se observó que las acreditaciones de pagos en las cuentas de los clientes se realizaban en un promedio de un segundo, independientemente de si la carga era alta y no se presenciaron cuellos de botella.

2.4.Métricas clave

Incremento en la capacidad del procesamiento de acreditaciones concurrentes.

Reducción de tiempo total para procesar gran cantidad de acreditaciones.

2.5.Conclusión

El objetivo fue cumplido logrando que las acreditaciones se vean reflejadas en las cuentas de los clientes con tiempos significativamente reducidos incluso con altas cargas.

5.2. Recomendaciones

Dentro del desarrollo del proyecto de rediseño se presenciaron algunas dificultades que pudieron haber sido gestionadas de mejor manera:

1. **Adopción de nuevas tecnologías:** Debido a que el rediseño del sistema de depósitos conllevó a aprender conceptos modernos, se dificultó al equipo de desarrollo aprender estos nuevos conceptos en poco tiempo. Recomendamos realizar capacitaciones continuas y permanentes acerca de las tecnologías emergentes y realizar talleres prácticos para que el equipo de desarrollo se familiarice con ellas.
2. **Observabilidad:** Al tener componentes separados fue difícil poder mantener la trazabilidad cuando se realizaban las operaciones, ya que no se contaba con una plataforma centralizada, sino que cada componente mantenía sus registros. La recomendación es disponer de una plataforma que cumpla con los estándares de trazabilidad, y de observabilidad, como refiere Stephen Watts¹⁰, la capacidad de medir los estados internos de un sistema examinando sus resultados donde se pueda observar y agrupar las trazas de los diferentes componentes en una sola vista y crear alertas.
3. **Políticas de fallo:** Los componentes que consumen eventos pueden presentar fallas, por lo cual el bróker de mensajería tiene una política de reintento en la cual se puede reintentar el consumo del mismo mensaje más de una vez. cuando estos reintentos se agoten, ya que el mensaje quedaría colgado sin tener ningún procesamiento. Nosotros recomendamos definir una política de fallos para cada evento, con el propósito de minimizar el impacto de los incidentes y eliminar los incidentes recurrentes.

¹⁰ **Stephen Watts** : trabaja en marketing de crecimiento en Splunk. Stephen tiene un título en Filosofía de la Universidad de Auburn y es candidato a MSIS en UC Denver. Colabora con una variedad de publicaciones, incluidas CIO.com

4. **Conexiones a base datos:** El sistema al ser altamente escalable creaba muchas conexiones a la base de datos, las cuales a veces se cerraban debido a que la base de datos se encontraba en la nube y había errores de intermitencia. En caso de eventos o desastres inesperados, el sistema debe ser capaz de mantener o recuperar rápidamente la disponibilidad, la accesibilidad y la integridad de los datos, esto implica diseñar una estrategia que contemple la capacidad de recuperarse o adaptarse fácilmente a una interrupción. Nuestra recomendación es manejar la resiliencia de conexiones a base de datos.
5. **Uso de servicios externos:** Debido a que las notificaciones se realizaban mediante un proveedor de correos externo mediante consultas HTTP, muchas veces por la alta concurrencia el servidor del proveedor de correos arrojaba errores de indisponibilidad. Nuestra sugerencia es manejar la resiliencia de solicitudes HTTP.
6. **Seguridad:** Debido a que los componentes se conectan a un bróker de mensajería en la nube por simplicidad se utilizó la cadena de conexión. Sin embargo, sugerimos que haya una autenticación más segura para conectarse al bróker ya que no se desea que las colas se vean comprometidas.
7. **Patrones de arquitectura de aplicaciones:** Debido a que diferentes desarrolladores trabajaron en la construcción del sistema no hubo un estándar de qué arquitectura de aplicación usar. Recomendamos usar un patrón definido como Domain Driven Design ya que está orientado a la construcción de soluciones que usan eventos de dominio.

Fuentes de información

Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., & Stafford, J. (2010). *Documenting software architectures: Views and beyond* (2nd ed.). Addison-Wesley.

Kazman, R., Klein, M., & Clements, P. (2020). *ATAM: Method for architecture evaluation*. Addison-Wesley.

Newman, S. (2021). *Monolith to microservices: Evolutionary patterns to transform your monolith*. O'Reilly Media.

Richards, M., & Ford, N. (2020). *Fundamentals of software architecture: An engineering approach*. O'Reilly Media.

Rozanski, N., & Woods, E. (2020). *Software systems architecture: Working with stakeholders using viewpoints and perspectives* (2nd ed.). Addison-Wesley.

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2020). *Software architecture: Foundations, theory, and practice*. Wiley.

Montgomery, R. (2020). *Event-driven microservices: Leveraging event-driven architecture for modern applications*. Apress.

Poniatowski, S. (2021). *Event-driven architecture in the cloud: The definitive guide*. O'Reilly Media.

Hohpe, G., & Woolf, B. (2020). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional.

Fowler, M. (2020). *Patterns of enterprise application architecture*. Addison-Wesley Professional.

Burns, B., Beda, J., & Hightower, K. (2020). *Kubernetes: Up & Running* (3rd ed.). O'Reilly Media.

Cloud Native Computing Foundation (CNCF). (2023). *Kubernetes: The Journey to Modern Application Management*. CNCF. <https://cncf.io>

Google Cloud. (2022). *Horizontal Pod Autoscaler* en Kubernetes Engine. Google Cloud Documentation. <https://cloud.google.com/kubernetes-engine/docs/how-to/horizontal-pod-autoscaling>

Kubernetes.io. (2023). *Scaling your application*. <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#scaling-your-deployment>

Red Hat. (2022). *Vertical Pod Autoscaler* on OpenShift. Red Hat Documentation. <https://access.redhat.com/documentation/en-us>

Liebig, T., & Diel, A. (2021). *Kubernetes in Action*. Manning Publications.

Brown, S. (2020). *Software Architecture for Developers* (2nd ed.). Leanpub. <https://leanpub.com/software-architecture-for-developers>

C4 Model. (2024). *C4 Model for visualising software architecture*. <https://c4model.com/>

Humble, J., & Farley, D. (2020). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* (2nd ed.). Addison-Wesley.

Nygaard, M. (2021). *Release It!: Design and Deploy Production-Ready Software* (2nd ed.). Pragmatic Bookshelf.

Hewitt, C., & Ruhl, S. (2021). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media.

Kreps, J., Narkhede, N., & Rao, J. (2011). *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media.

Liu, Y., & Liu, J. (2020). *Event-Driven Architecture: How to Use Event-Driven Architecture for Business Processes*. Springer.

Martin, F. (2020). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Schmidt, D. C., & Stal, M. (2020). *Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing*. Wiley.

- Cito, A. (2022). *Performance Testing Guidance for Web Applications*. Microsoft Press.
- González, A. (2021). *Performance Testing: A Comprehensive Guide*. Apress.
- Mayer, M. (2020). *Load Testing for Web Applications*. O'Reilly Media.
- Pace, D. (2023). *The Complete Guide to Performance Testing with JMeter*. Leanpub.
- Rao, S., & Kumar, A. (2021). *Performance Engineering of Software Systems*. Springer.
- Lutz, D. W. (2021). *Understanding UUIDs: A Brief Overview of the Universal Unique Identifier*. *Journal of Software Engineering and Applications*, 14(2), 34-45.
- Peterson, J. (2023). *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design*. Addison-Wesley.
- Tennant, J., & He, J. (2022). *UUIDs as Primary Keys in Database Design*. *International Journal of Computer Science and Information Technology*, 14(1), 57-66.
- Winston, H. (2020). *Database Management Systems: A Practical Approach to Design, Implementation, and Management*. Cengage Learning.
- Joshi, K. (2016). *Message Queues and Their Role in Fault Tolerant Systems*.
- Muthukrishnan, M., et al. (2018). *Reliability and Persistence in Message Brokers*.
- Tarkoma, S., Rothenberg, C. E., & Lagerspetz, E. (2011). "Theory and Practice of Publish/Subscribe Systems." *Computer Science Review*, 5(3), 157-167
- Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. M. (2003). "The Many Faces of Publish/Subscribe." *ACM Computing Surveys (CSUR)*, 35(2), 114-131
- Chockler, G., Keidar, I., & Vitenberg, R. (2001). "Group Communication Specifications: A Comprehensive Study." *ACM Computing Surveys (CSUR)*, 33(4), 427-469

- Balazinska, M., Balakrishnan, H., Stonebraker, M., & Cetintemel, U. (2005). "Fault-tolerance in the Borealis Distributed Stream Processing System." *ACM Transactions on Database Systems (TODS)*, 30(2), 97-141
- Kelsey, M., et al. (2018). *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*. O'Reilly Media
- Beck, K. (2004). *Extreme Programming Explained: Embrace Change (2nd ed.)*. Addison-Wesley.
- Evans, E. (2004). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.
- Millett, S., & Tune, N. (2015). *Patterns, Principles, and Practices of Domain-Driven Design*. Wiley.

ANEXOS

Anexo A: Formulario de solicitud de depósito

BCP

S/ 15 - S/ 35000

Interbank

S/ 15 - S/ 35000

BBVA

S/ 15 - S/ 35000

Scotiabank

PagoEfectivo

Scotiabank

S/ 15 - S/ 7000

Yape Servicios

500

Yape Servicios

S/ 15 - S/ 2000

Yape-Plin

via Yape PagoEfectivo

Yape-Plin

S/ 15 - S/ 500

PagoDirecto

S/ 15 - S/ 35000

Depositar

X

Escoge una opción de pago

Este depósito será procesado por

Solo las cuentas en las que seas la persona titular se pueden utilizar para realizar depósitos.

Al hacer clic en 'Depositar' se generará un código de pago (CIP) en una ventana emergente y se enviará a tu correo electrónico.

Monto

100

300

500

1000

50

PEN

Puedes depositar desde S/ 15 hasta S/ 50

Depositar

El monto elegido es de S/ 50.00

PCI DSS

Chat de ayuda aquí

125

Anexo B: Váucher de pago



Pago de servicios

Empresa: **ALUMINUM S.A.**

Servicio: **RENTA DE EQUIPOS**

Código de pago (CIP)

7310607

 Copiar CIP

Monto a pagar

S/.50.00

Anexo C: Correo de instrucciones de pago

Instrucciones de Pago via

7310607

Recibidos x

para mi

Anular suscripción

8:30 a.m. (hace 3 minutos)

BCP

Pago de servicios

Empresa:

Servicio:

Código de pago (CIP)

7310607

Monto a pagar

S/.50.00

Anexo D: Pago en aplicación del banco



Anexo E: Correo de confirmación de pago

Confirmación de pago 7310607

Recibidos x

para mí

Anular suscripción

8:48 a.m. (hace 6 minutos)

☆ 😊 ↩ ⋮

CONFIRMACION DE PAGO

¡Hola!

Tu pago hacia a sido recibido exitosamente (*) (**)

Código de pago: 7310607

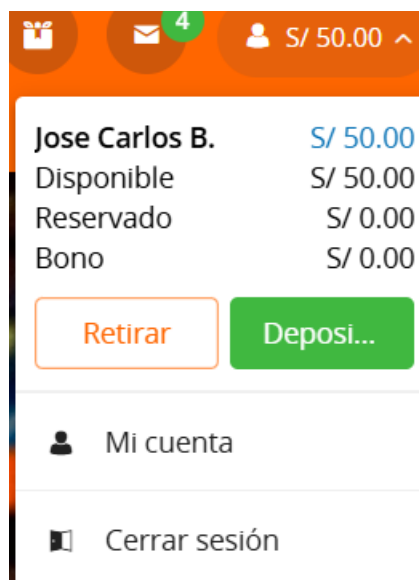
Monto: S/.50.00

Pago recibido el 28/11/24 a las 08:48 AM.

(*) Tu importe será agregado en tu cuenta rea le inmediato si tu operación es en línea, en caso contrario puede tomar hasta 10 minutos.

(**) Este email es solo para informar que el pago ha sido realizado, no puede ser usado como comprobante o voucher de pago, ya que esta constancia es emitida por la entidad bancaria o financiera donde se realizó el pago.

Anexo F: Acreditación de pago en cuenta virtual



Anexo G: Diagrama de flujo de Post Service

