



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

## FACULTAD DE INFORMÁTICA

# TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

**TÍTULO:** Medio de pagos digitales, desafíos y futuro  
**AUTOR:** Pablo Yovovich  
**DIRECTOR ACADÉMICO:** Rodolfo Bertone y Luciano Marrero  
**DIRECTOR PROFESIONAL:** César Curcio  
**CARRERA:** Licenciatura en Sistemas

### Resumen

*Las billeteras virtuales se han establecido, en estos últimos años, en el mercado como una de las formas de pago más ágiles y utilizadas y su uso continúa creciendo.*

*En este trabajo se presentan distintas opciones para facilitar pagos en línea o sin contacto, de todas las formas en la que los pagos vienen evolucionando junto a la digitalización y bancarización de los usuarios.*

### Palabras Clave

*Pagos digitales, evolución tecnológica, arquitectura de desarrollo, billeteras virtuales, QR, carritos de compras, botones de pago, transferencias, portales de venta*

### Conclusiones

*Se expone un conjunto de servicios desarrollados que facilitan el pago online para disminuir el uso de efectivo o tarjetas físicas en las transacciones. Se muestra la tecnología utilizada y la manera de integrarla a los ecosistemas existentes. Se considera cómo la innovación tecnológica brinda rapidez y seguridad a dichas transacciones y como evoluciona rápidamente*

### Trabajos Realizados

*Análisis y desarrollo de las soluciones de:*

- *Botón de pago.*
- *Intenciones de pagos para implementar carritos de compras.*
- *Formulario online de pago.*
- *Transferencias 3.0 para el pago QR entre billeteras digitales.*

*Tecnología utilizada para los desarrollos e implementaciones.*

*Monitoreo de transacciones de pago*

### Trabajos Futuros

*Para continuar creciendo, se desplegarán nuevos servicios, como un marketplace orientado a un público joven con compras georreferenciadas dotado de personalización, agilidad y dinamismo*

Fecha de la presentación:



UNIVERSIDAD NACIONAL DE LA PLATA

Facultad de Informática

Medios de pagos digitales, desafíos y futuro.

Alumno: Pablo Yovovich 3945/9

Director: Rodolfo Bertone y Luciano Marrero

## **AGRADECIMIENTOS**

A mis padres, que me dieron la posibilidad de estudiar y formarme.

A mi señora y a mi hija, que me han acompañado en estos tiempos.

A mis amigos, que siempre están al lado cuando se los necesita.

A la facultad por haberme dado la formación que posibilitó mi desarrollo profesional en estos años.

Al equipo de trabajo del que formo parte, por continuar creciendo profesionalmente.

A los directores de tesis que me han acompañado.

# Índice

<b>Índice</b>	<b>4</b>
<b>Capítulo 1</b>	<b>4</b>
1.1 - Objetivo	4
1.2 - Introducción	4
1.3 - Ambiente de trabajo	5
<b>Capítulo 2 - Pautas de los desarrollos</b>	<b>9</b>
2.1 - Microservicios	9
2.2 - Integración continua	14
2.3 - Logueo	15
2.4 - Monitoreos	18
<b>Capítulo 3 - Desarrollos realizados.</b>	<b>22</b>
3.1 - Botón de pago	22
3.1.1 - Descripción funcional	22
3.1.2 - Estados de un botón de pago	24
3.1.3 - Imágenes ilustrativas de la creación de un botón de pago	25
3.1.4 - Descripción técnica de botón de pago	26
3.1.5 - Conclusiones	30
3.2 - Intención de pago	31
3.2.1 - Descripción funcional	31
3.2.2 - Diagrama de flujo	31
3.2.3 - Imágenes ilustrativas de creación de intención de pago	32
3.2.4 - Descripción técnica	33
3.2.5 - Conclusiones	38
3.3 - Formulario de pago	39
3.3.1 - Descripción funcional	39
3.3.2 - Diagrama de flujo del formulario de pago	39
3.3.3 - Imágenes ilustrativas de pago con formulario	40
3.3.4 - Descripción técnica	43
3.3.5 - Conclusiones	46
3.4 - Transferencias 3.0	47
3.4.1 - Descripción funcional	47
3.4.2 - Diagrama de flujo de Transferencias 3.0	47
3.4.3 - Imagen ilustrativa de un QR de Transferencia 3.0	50
3.4.4 - Descripción técnica del funcionamiento de Transferencias 3.0	50
3.4.5 - Conclusiones	52
<b>Capítulo 4 - Conclusiones</b>	<b>52</b>
<b>Capítulo 5 - Trabajos futuros</b>	<b>55</b>
<b>Capítulo 6 - Glosario</b>	<b>55</b>
<b>Capítulo 7 - Referencias bibliográficas.</b>	<b>59</b>

# Capítulo 1

## 1.1 - Objetivo

En los últimos años, las billeteras virtuales [2] (también conocidas como billeteras electrónicas o e-Wallet) se han consolidado en el mercado como el medio de pago más utilizado. Además, la irrupción de una pandemia mundial ha aumentado las transacciones u operaciones a través de internet.

Si bien muchos comercios ya contaban con la posibilidad de realizar ventas online, desde las medidas de aislamiento por la pandemia, comenzó a incrementarse de manera significativa. La incorporación de las billeteras virtuales para el comercio electrónico está presente en la gran mayoría de los comercios del mercado actual.

El propósito de este trabajo es el de presentar diferentes desarrollos que formarán parte de la solución de los pagos digitales en Todo Pago [1], así como la integración de los mismos y las metodologías aplicadas durante el proceso de diseño y desarrollo.

## 1.2 - Introducción

Desde hace algunos años, desde Todo Pago [1], se propuso realizar una transformación digital de sus productos para mejorar sus servicios. Para ello, se formaron equipos de trabajo para analizar, estudiar y evaluar el área encargada de los procesos de cobros con el objetivo de determinar y establecer las mejoras que sean necesarias.

En la área del proceso de cobros, se realizan las siguientes tareas:

- Generación de microservicios encargados de administrar los botones de pagos (aquellos utilizados para brindarles a los clientes un acceso a un formulario de pago).
- Generación de microservicios para resolver las integraciones entre el comercio electrónico y las soluciones brindadas por la empresa.
- Generación de microservicios para permitir el pago con QR's [33] y transferencias inmediatas entre cuentas virtuales (Transferencias 3.0).
- Monitoreo de transacciones.

## 1.3 - Ambiente de trabajo

Se conformaron cuatro equipos diferentes para evaluar y analizar las necesidades del cliente. Estos equipos son:

- Registros: encargado del proceso de registro de nuevos vendedores y billeteras virtuales. También es responsable de administrar los datos de las cuentas.
- Cuenta virtual: encargado de la administración de las cuentas de los clientes.
- Cobros: encargado de administrar los distintos canales para poder realizar el cobro a clientes.
- Postventa: encargado de realizar el cobro de impuestos y/o retenciones de acuerdo al rubro del comercio y el volumen de ventas que realiza.

El presente trabajo se encargará de detallar los desarrollos del equipo de cobros.

Para llevar adelante el proceso de estudio y mejora, los equipos utilizaron la metodología ágil Scrum [10]. Esto se debe a que, a través de Scrum [10], se obtiene un ritmo de trabajo sostenible, adaptable a los cambios y con iteraciones, a fin de obtener el mejor resultado y de esta manera, maximizar el valor del producto. En la figura 1 se presentan los componentes de la metodología Scrum [10].



**Figura 1.** Componentes de la metodología Scrum

En esta metodología, se cuenta con un Product Backlog, que es un listado de todas las tareas que se pretenden realizar durante el desarrollo del proyecto. Esas tareas deben estar visibles para todo el equipo con el objetivo de tener una visión general de lo que se desea realizar. Todas esas tareas se denominan historias de usuario y se gestionan a través de la herramienta Jira [34]. En la figura 2 se presenta una captura de pantalla de Jira [34].

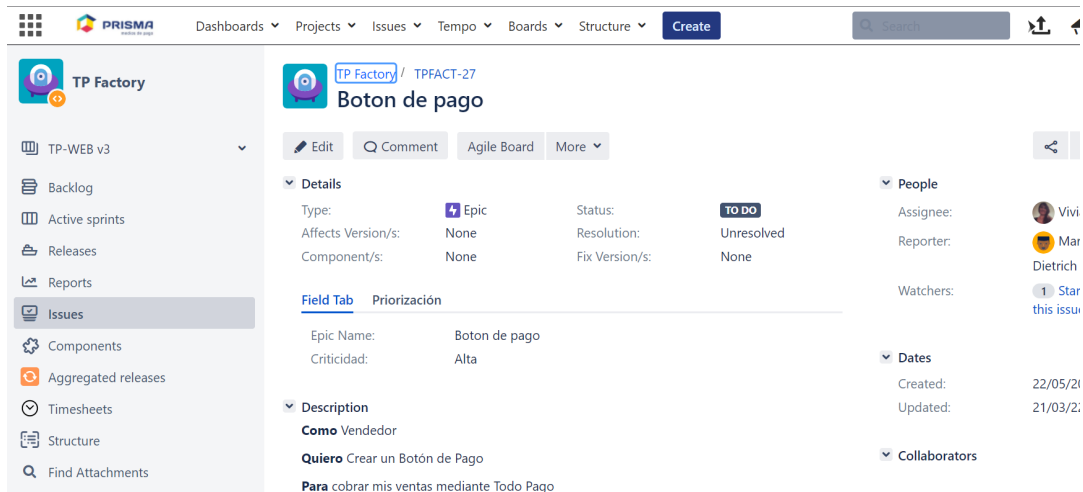


Figura 2. Captura de pantalla de Jira

El Product Owner (propietario del proyecto) posee una visión global de todo lo que hay que construir y tiene la responsabilidad de transmitir esa visión al resto del equipo.

El Scrum Master ayuda al equipo a mantenerse enfocado en los objetivos del proyecto y resuelve los obstáculos que puedan ir surgiendo.

El equipo organiza su trabajo en sprints [9] de dos semanas cada uno. Los sprints son períodos de tiempos fijados por el equipo para completar una serie de tareas del proyecto.

Además, cada equipo se encuentra conformado por desarrolladores backend, desarrolladores frontend y desarrolladores mobile (iOS y Android). Los desarrollos, una vez finalizados de acuerdo a la historia de usuario, se presentan al equipo y luego se pasan a un estado denominado “listo para testear”.

Una vez probados y certificados los desarrollos, la historia asociada se pasa a un estado denominado “listo para implementar”.

Finalmente, las historias probadas se implementan en producción y se cierran.

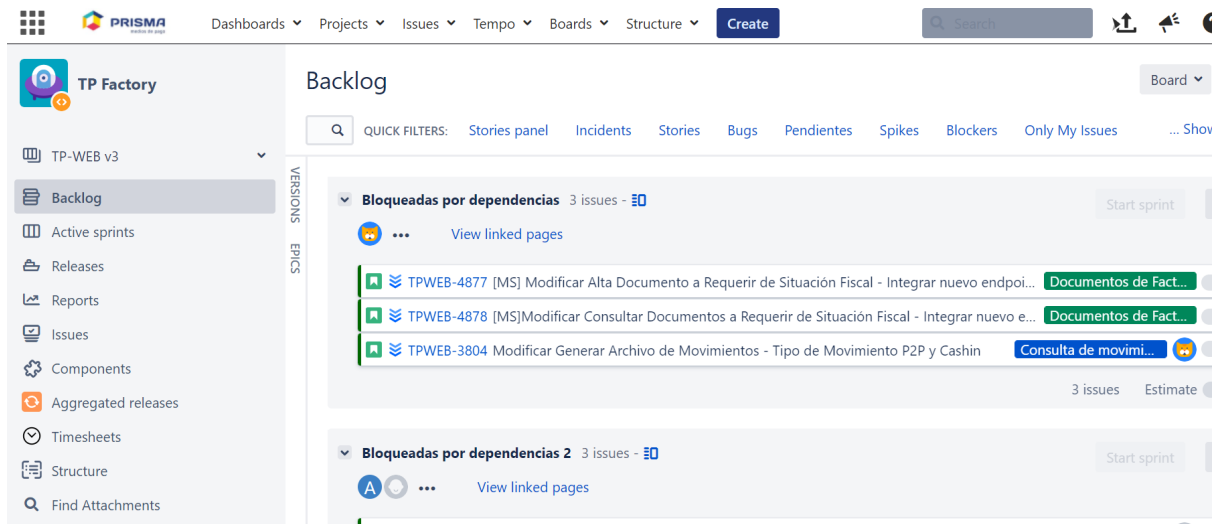
Cada sprint conlleva un conjunto de reuniones que son importantes para mantener la coordinación en las actividades y cumplir en tiempo y forma con las tareas propuestas.



Las reuniones del equipo son las siguientes:

- Planificación previa (pre planning) de historias de usuario: En esta reunión pueden surgir requerimientos de proyectos que no se conocen en profundidad, lo que genera nuevas historias de análisis de esos requerimientos.
- Planificación: En función de lo definido en la pre planning se confirman las historias que se desarrollarán, teniendo en cuenta los desbordes (historias de usuario no finalizadas). Se releva el tablero de historias de usuario y se asignan y definen los puntajes de cada una. Además, se ve cuales entran teniendo en cuenta la capacidad del equipo.
- Se realizan reuniones diarias (Daily Meeting) de 15 minutos de duración, donde cada integrante del equipo expone sus avances, próximas tareas y si posee algún inconveniente que no le permita avanzar.
- Cierre de sprint: El último día del sprint se presentan los avances e historias de usuario completadas, se analiza si quedó alguna sin finalizar y se pasa al siguiente sprint. Además, se dá una visión rápida de lo que va a entrar en el siguiente sprint
- Retrospectiva: El último día del sprint se realiza la reunión de retrospectiva, en el que bajo una consigna propuesta, cada integrante del equipo comenta aspectos positivos, posibles mejoras del sprint y qué acciones tomar sobre las mejoras propuestas. En la siguiente retrospectiva se analizará si dichas acciones pudieron llevarse a cabo.

A continuación, en la figura 3 se presenta una captura de pantalla de Jira [34], en la cual se visualiza un listado de tareas.



**Figura 3:** Vista de Backlog de tareas en Jira

Los desarrollos realizados se basaron en la Arquitectura Orientada a Servicios [11], la solución se desarrolla de forma distribuida, cada componente cumple un rol específico y para comunicarse entre ellos utilizan protocolos de comunicación. La mayoría del software desarrollado fue en Backend, utilizando para ello Spring-boot [4], con Java 11 y Kotlin [4]. Como base de datos relacional se utilizó SQL Server [43]. También se utilizó Redis [5] como base de datos no relacional en memoria y Kafka [6] como cola de mensajes para transacciones asincrónicas. También se utilizaron servicios ya disponibles encargados del manejo de cuentas de usuarios, administración de transacciones y saldos de las cuentas, lo cual disminuyó los tiempos de desarrollo.

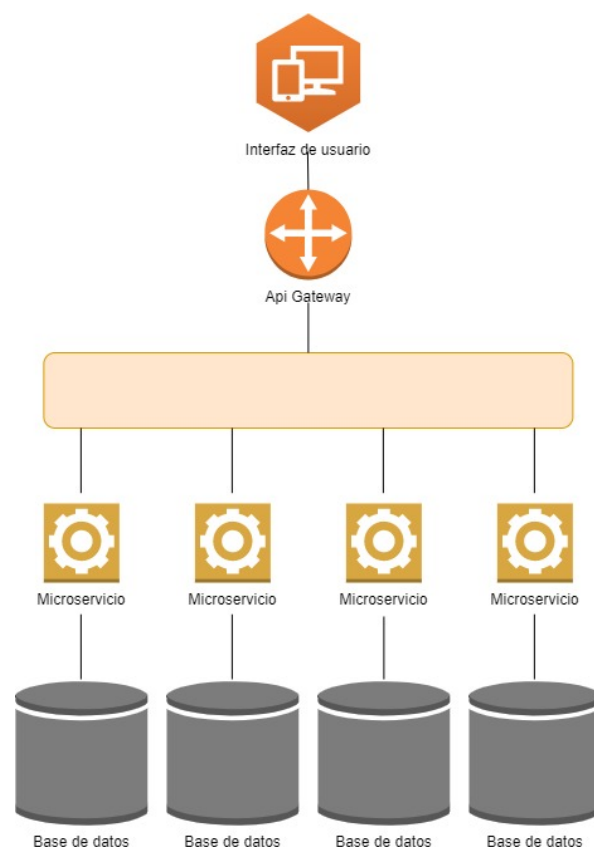
## Capítulo 2 - Pautas de los desarrollos

### 2.1 - Microservicios

El desarrollo de los proyectos que se van a describir se basaron en la arquitectura de microservicios [3]. Dicha arquitectura consta de un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, y que juntos componen una funcionalidad de negocio completa.

Cada microservicio que compone la solución puede estar escrito en un lenguaje diferente. Los microservicios se comunican entre sí a través de API Rest [20]. Cada uno cuenta con almacenamiento propio evitando una sobrecarga o caída general de la aplicación [21].

A continuación, en la figura 4, se presenta un diagrama ilustrativo de la arquitectura de microservicios y sus componentes.



**Figura 4.** Arquitectura de microservicios

Los microservicios siguen las convenciones de publicaciones de API Rest [20], donde cada operación se describe como HTTP (GET, POST, PUT o DELETE) y su URL. Por ejemplo:

- GET *id\_account/buttons*: Devuelve una lista de botones de una cuenta dada.
- GET *id\_account/buttons/id\_button*: Devuelve un botón específico de una cuenta dada.
- POST *id\_account/buttons* : Crear un nuevo botón en una cuenta dada.
- PUT *id\_account/buttons/id\_button* : Modifica los datos de un botón de una cuenta dada
- DELETE *id\_account/buttons/id\_button*: Elimina un botón de una cuenta dada.

Siendo *id\_account* y *id\_button*, variables que pueden tomar diferentes valores.

Además a las APIS se las invoca enviando las cabeceras o headers HTTP (donde se envía principalmente los datos de autenticación del usuario que las consume) y el cuerpo del mensaje que se encuentra en formato JSON [12].

Como resultado, las APIs deben devolver el código HTTP que indica si se ejecutó correctamente la petición, por ejemplo: 200 si la petición fue correcta, 400 si los datos de entradas no respetan el formato de la petición, 422 si se envió algún dato incorrecto o 500 si ocurrió un error interno del servidor. Además, se retornan los datos en formato JSON.

A continuación, en la figura 5, se presenta un ejemplo de una llamada a una API para la creación de un botón de pago.

The screenshot shows a REST client interface for a service named "Ms-Payment\_Button / 02 Add new Button". The request is a POST to the endpoint `{{host-button}}/public/oauth/v1/ms-payment-button/{{accountIDForButtons}}/button`. The request body is a JSON object:

```

1 {"text": "Pagar",
2  "title": "Boton",
3  "amount": "100.01",
4  "all_payment_methods": "true",
5  "manage_stock": "false"}

```

The response is a 200 OK status with a response time of 733 ms and a body size of 922 B. The response body is a JSON object:

```

1 {}
2  "id_button": 158134,
3  "id_public": "b233f8f9c04e3636ac5d2fdab389e664",
4  "form_url": "https://[redacted]/web-app-form/b233f8f9c04e3636ac5d2fdab389e664"
5 {}

```

Figura 5. Ejemplo de llamada a una API.

Los microservicios desarrollados deberán contar con su propia documentación en Swagger [36], lo cual facilita a que los consumidores de dichas APIs cuenten con la información necesaria para hacer sus desarrollos. A continuación, en la figura 6, se presenta un ejemplo de microservicio documentado con Swagger.

The screenshot shows the Swagger UI for a microservice named "ms-payment-button" (version 0.0.1). The service is described as "Microservice for payment buttons". The API is organized into a "payment-button-controller" (Payment Button Controller). The following endpoints are listed:

- POST** `/public/oauth/v1/ms-payment-button/{accountId}/button` createButton
- GET** `/public/oauth/v1/ms-payment-button/{accountId}/button/{buttonId}` retrieve
- PUT** `/public/oauth/v1/ms-payment-button/{accountId}/button/{buttonId}` UpdateButton
- DELETE** `/public/oauth/v1/ms-payment-button/{accountId}/button/{buttonId}` deleteButton

Figura 6. Ejemplo de la documentación de un microservicio generada por Swagger.

Los microservicios se desarrollaron en Kotlin[4], con el framework de Spring Boot [4].

Internamente, los microservicios fueron divididos en diferentes módulos:

- Controladores (Controllers): Contiene las clases que se encargan de exponer los endpoints [42], recibir las llamadas e invocar con los datos a las respectivas operaciones del servicio.
- Servicio (Service): Contiene las interfaces y clases donde se procesan las operaciones de negocio. El mismo recibe las peticiones, trabaja sobre las entidades del modelo, interactuando, en caso necesario, con el repositorio (repository) o el cliente (client). Luego, genera la respuesta al usuario.
- Modelo (Model): Son las clases que forman el modelo de negocio.
- Repositorio (Repository): Consta de las clases que acceden a los repositorios para guardar o traer los datos hacia el modelo. Dichos repositorios son: Una base de datos SQL Server, Redis [5] o S3.
- Cliente (Client): Contiene las interfaces y las clases necesarias para acceder comunicarse a otros microservicios y conectarlos con el modelo de negocio.
- Error: Contiene las clases que manejan las excepciones y los errores que ocurran (de ejecución, de validación, entre otros) para poder dar una respuesta personalizada ante cada problema.

Los parámetros configurables del microservicio se encuentran en otro microservicio que maneja dichos valores. El mismo provee un mecanismo para mantener una configuración centralizada.

Finalmente, para cada microservicio se escribe un conjunto de pruebas unitarias y de integración. El propósito principal de estas pruebas radica en asegurar una calidad del producto y que ante las modificaciones y/o mejoras que se incorporen al código, se aseguren de que no afecta el funcionamiento de la aplicación.

## 2.2 - Integración continua

El despliegue del código en todos los ambientes se realiza utilizando la metodología de integración continua [23], la cual es una práctica en donde los desarrolladores integran, periódicamente, su código a un repositorio central. Cada vez que un miembro del equipo sube su código modificado (commit) sobre el control de versiones, el proyecto se compila y se ejecutan todas sus pruebas unitarias. Todo esto se realiza de forma automática.

A continuación, en la figura 7, se presenta un diagrama con el flujo de los pasos de la Integración continua.

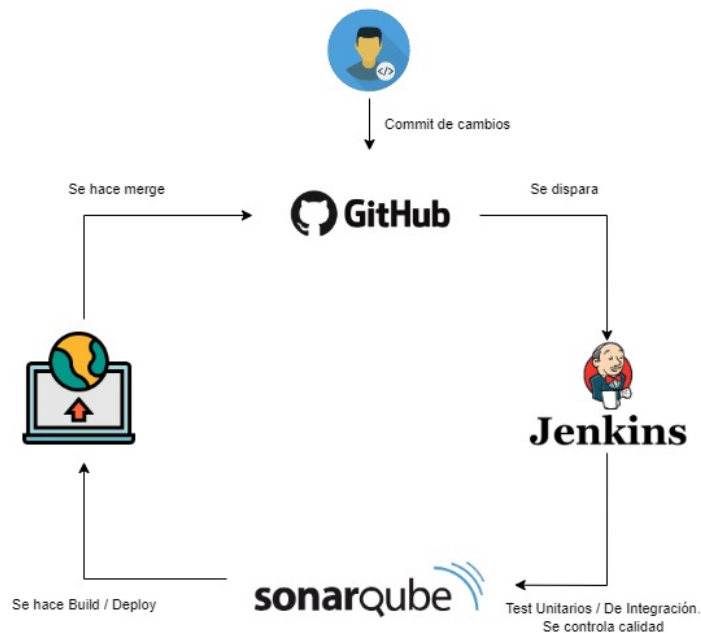


Figura 7. Diagrama de integración continua.

El repositorio del código con control de versiones está en Git [15]. Cada desarrollo o modificación al código se realiza creando una rama (branch) a partir de la principal, sobre la que se trabaja. Una vez realizada la modificación, esos cambios deben impactar sobre la rama principal, previa revisión y validación de dichos cambios por otros desarrolladores.

Luego, la herramienta utilizada para el proceso de implementación automatizado es Jenkins [7], en la cual se ejecutan diferentes procesos o etapas que permiten realizar la validación de los cambios previos al merge (impacto de los

cambios en la rama principal del repositorio). A continuación, en la figura 8, se presenta una captura de pantalla de Jenkins.

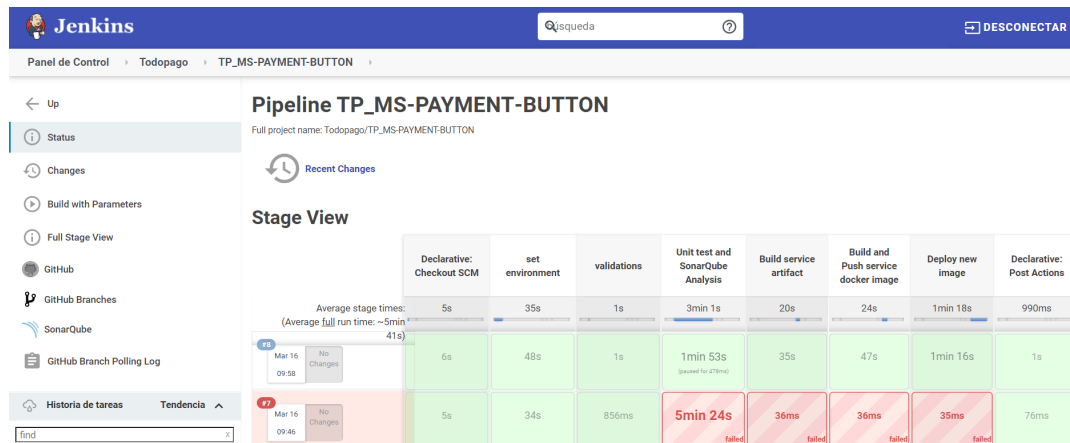


Figura 8. Despliegue de un cambio en Jenkins.

Otra de las herramientas utilizada es SonarQube [24]. Esta es una herramienta validadora de código donde se verifican aspectos como: reglas de escrituras, cobertura de las pruebas y que se cumpla con los estándares, entre otros. A continuación, en la figura 9, se presenta una captura de pantalla de SonarQube [24] aplicado a uno de los microservicios.

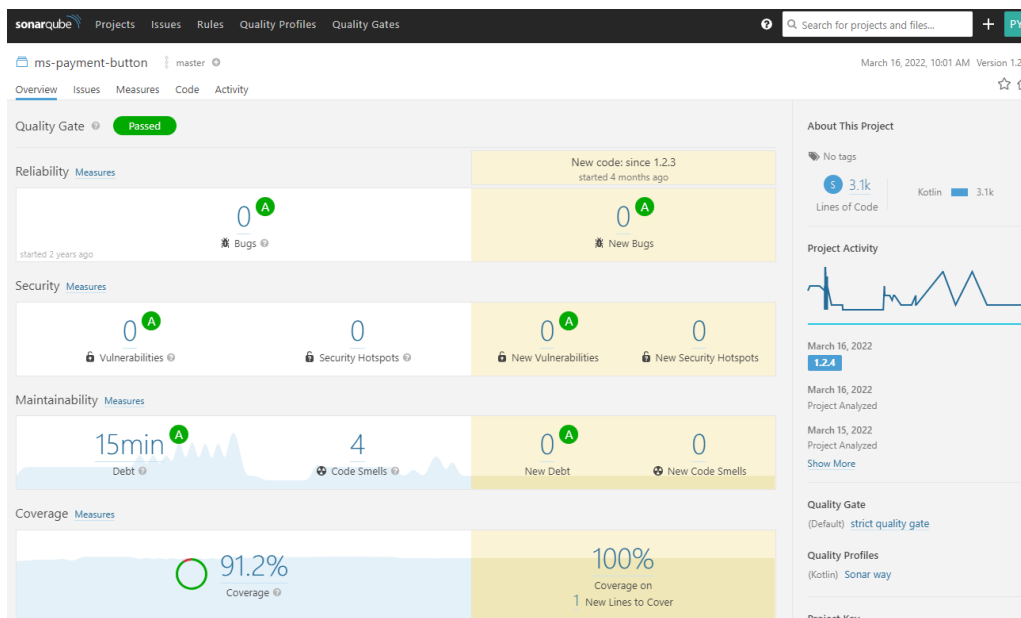


Figura 9. SonarQube con indicadores de código de un microservicio.



## 2.3 - Logueo

Los sistemas transaccionales cada vez manejan volúmenes de datos más grandes. Esto origina la necesidad de contar con herramientas dedicadas a monitorear el funcionamiento de dichos sistemas para poder detectar fallas u obtener la trazabilidad de las operaciones de una manera eficiente.

En este caso, para poder verificar el funcionamiento de los microservicios, se pondrá foco en los logs [17] de operaciones. Los logs son repositorios con información de lo que ocurre en un sistema específico. A continuación, en la figura 10, se presenta la captura de pantalla de un log.

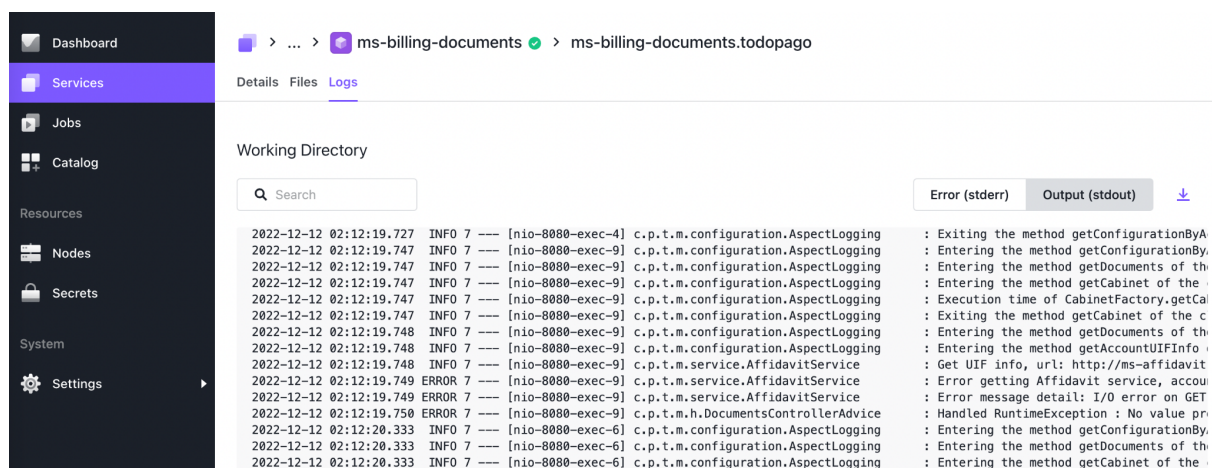


Figura 10. Vista de logs de un servicio.

El formato de un archivo de log se conforma de la siguiente manera:

- Fecha y hora de la ejecución.
- Tipo de información (DEBUG, INFO o ERROR),
- Identificador de actividad, por si hay varias ejecuciones o hilos simultáneos.
- El mensaje o dato que se quiere loguear.

Los tipos de información de los logs que se manejan son: error, info y debug.

El tipo de información error, se utiliza para guardar log de errores, tanto de negocio como errores técnicos (por ejemplo, no se pudo acceder a la base de datos). En caso de un error técnico en la ejecución del microservicio, se muestra todo el detalle de la excepción que arrojó, a fin de poder encontrar el problema y tomar decisiones concretas para corregirlo.

Con el tipo de información info, los datos recolectados son las llamadas a las APIS, desde que se registra las peticiones hasta que el microservicio retorna la respuesta.

En el código de los microservicios se decidió loguear puntos comunes de las llamadas, los cuales son:

- Las peticiones entrantes.
- Llamadas realizadas a otros microservicios (requests y response) o envío de mensajes asincrónicos por medio de Kafka [6].
- La respuesta retornada a los clientes.

Se guardan como debug acciones como la persistencia o lectura de alguna base de datos.

El nivel de log (es decir si se guardan sólo los logs de error, así como también se puedan incluir los de info, e incluso los de debug) es configurable para cada microservicio y puede modificarse según sea necesario.

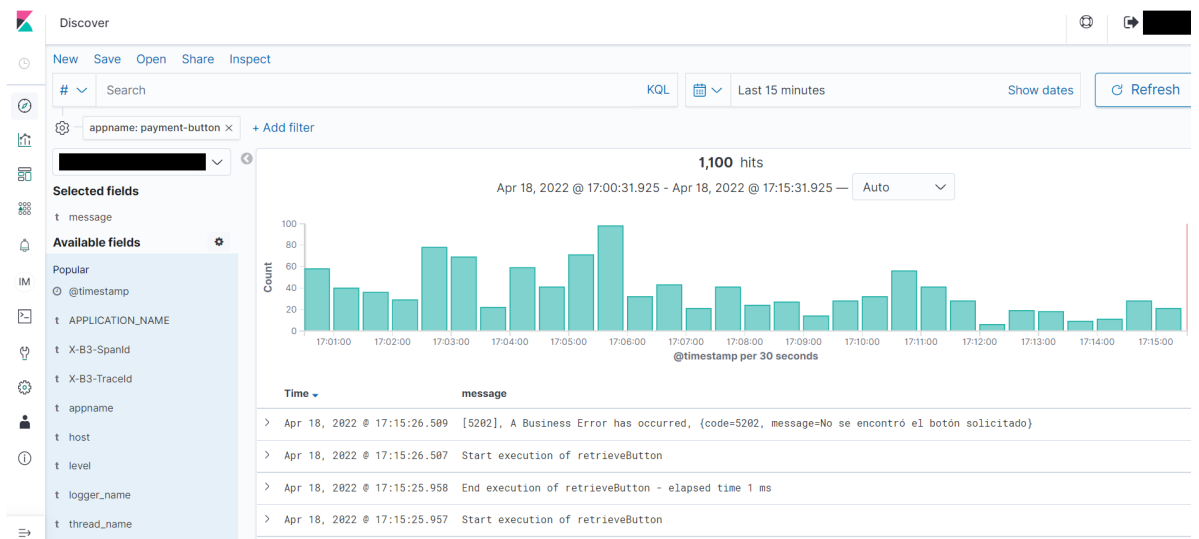
Por ejemplo: el nivel debug se puede activar para obtener información detallada de un incidente, brindando todos los detalles de la operación y una vez solucionado se vuelve a desactivar a fin de no llenar los logs con información ya no relevante.

Debido a que cada microservicio escribe logs en su propia instancia donde se está ejecutando (puede tener más de una), se puede dificultar encontrar la ejecución de una transacción la cual puede involucrar la consulta de logs de diferentes microservicios.

Para poder simplificar la tarea de búsqueda de log en todas las instancias, se utilizó ELK [18]. ELK es un conjunto de herramientas de administración de registros permitiendo la monitorización, consolidación y análisis de logs generados en múltiples servidores. Estas herramientas son: ElasticSearch, Logstash y Kibana [18]. Lo que se consigue es recolectar toda esta información, procesarla y almacenarla de forma distribuida. Así se va a poder visualizar información a gran escala, obtener buenos rendimientos con grandes cantidades de información, transformarla en visualizaciones, búsqueda centralizadas para ver por dónde pasó un número de

transacción, o todas las veces que se ejecutó determinadas APIs en un período de tiempo seleccionado.

A continuación, en la figura 11, se presenta una captura de pantalla para la búsqueda de datos en Kibana.



Figuras 11. Pantalla de Kibana.

## 2.4 - Monitoreos

La herramienta utilizada para monitoreo, análisis y visualizaciones de métricas de los datos transaccionales generados por los microservicios mencionados es Grafana [19]. Los componentes de grafana son:

- Pizarra (Dashboard): Conjunto de paneles que le dan formato a una visualización o presentación.
- Panel: Bloque de construcción básico para la visualización de los datos. Grafana [19] provee diferentes tipos de paneles y cada uno de ellos provee un editor de consulta.
- Intervalo de fechas: Se necesita especificar un intervalo de fechas para mostrar cualquier información.

El principal uso que se le está dando a Grafana [19] es poder acceder a información relacionada al volumen, control y seguimiento de las transacciones de

pago generadas, con visualizaciones generales o específicas a un canal, ya sea por botón de pago, por intenciones de pago (APIS) o Transferencias 3.0, entre otras.

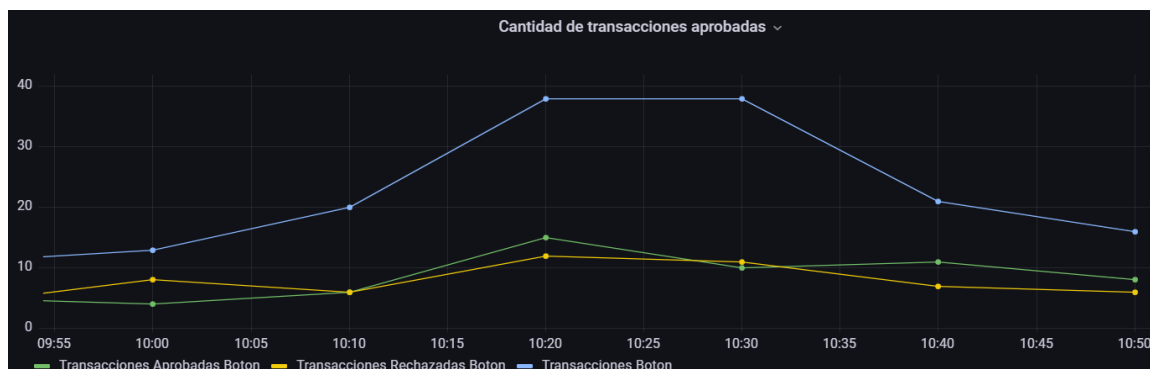
Con estas métricas se podrá observar fácilmente la información sobre algún canal en específico y facilitar la toma de decisiones en base a las mismas.

Las métricas a visualizar son:

- **Generales:** Se muestra un dashboard donde están representadas las transacciones en una visión global. Además, se muestran comparativamente las aprobaciones por cada canal en una línea de tiempo y se brindan datos a nivel global.
- **Por Canal:** Seleccionando un dashboard de un canal específico, por ejemplo: botón de pago. En este caso, se muestran datos y monitoreos específicos de los botones. Además, se muestran las aprobaciones y rechazos exclusivos de ese canal.

En general, los paneles que se muestran son los que representan la siguiente información:

- **Volumen de transacciones aprobadas:** Cantidad y sumatoria de los montos de las transacciones aprobadas en el periodo seleccionado para el canal especificado.
- **Cantidad de transacciones aprobadas:** Cantidad de transacciones aprobadas en el periodo seleccionado para el canal en cuestión. A continuación, en la figura 12 se presenta una captura de pantalla que muestra dicha información.



**Figura 12.** Grafana, transacciones totales, aprobadas y rechazadas de botones en una hora.

- Porcentaje de aprobación: Comparación de transacciones aprobadas o rechazadas (sin reintentos) para el canal en cuestión en el periodo seleccionado. A continuación, la figura 13, presenta un ejemplo de porcentaje de aprobaciones.

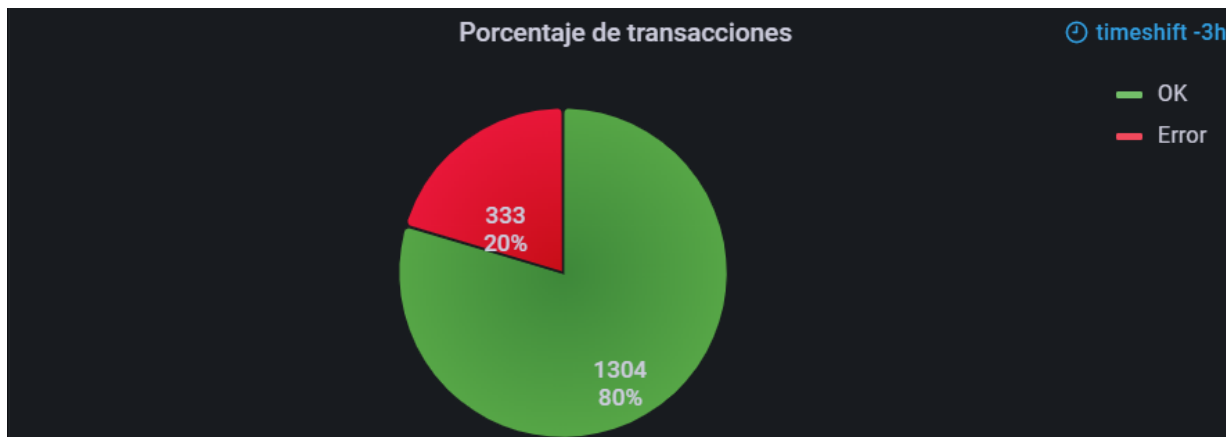


Figura 13. Grafana. Gráfico de torta indicando transacciones aprobadas o rechazadas en un período de tiempo.

A continuación, en la figura 14, se presenta una captura de pantalla que muestra una pizarra completa con los datos generales de las tasas de aprobaciones.

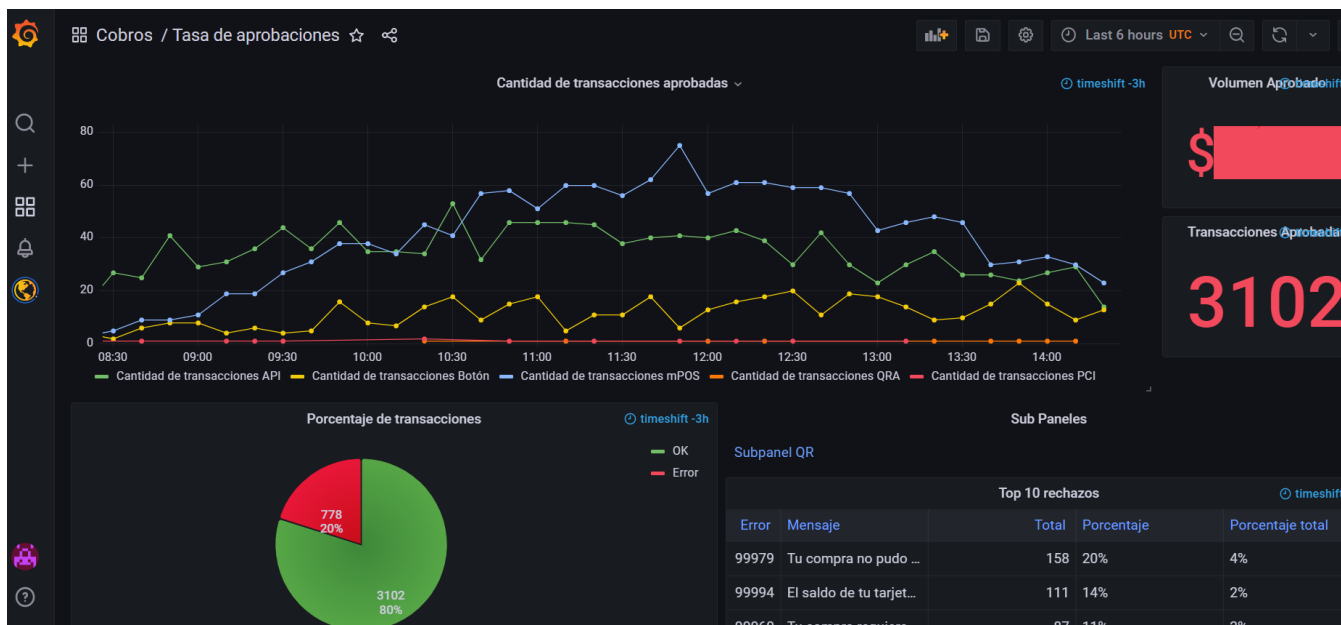


Figura 14. Panel de Grafana mostrando distintos gráficos para la tasa de aprobación general.

La figura 14, presenta en la barra superior el intervalo de consulta por el que se toma la muestra de datos (en este caso las últimas 6 horas) pero puede establecerse diferentes rangos.

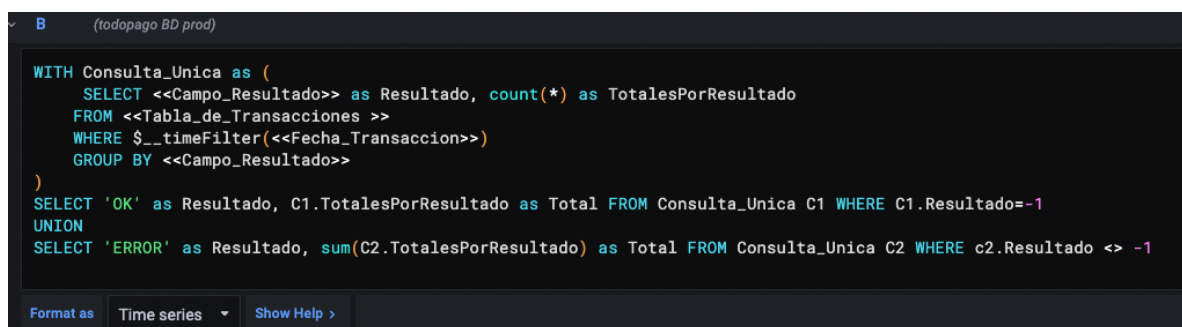
Por debajo aparece un panel con la línea de tiempo y un gráfico de barras con las aprobaciones de los diferentes canales. Además, se cuenta con dos paneles pequeños, ubicados a la derecha, uno con el monto total aprobado de transacciones en ese lapso y el otro con la cantidad de transacciones aprobadas.

Luego, se indica en un gráfico de torta la relación entre la cantidad de transacciones aprobadas y las rechazadas.

Y por último, se muestran los rechazos más frecuentes clasificados por código de error, descripción, el total y los porcentajes respecto a los rechazos como también respecto al total de transacciones.

Cada uno de los paneles mostrados anteriormente consta de una consulta SQL [19] a una tabla. Para poder brindar una consulta eficiente los campos por los que se consultan, se agrupan o se filtran deben estar indexados. El índice en dichos campos permite encontrar, filtrar u ordenar de forma rápida el contenido.

A continuación, las figuras 15 y 16, presentan dos ejemplos de SQL en Grafana [19].



```
WITH Consulta_Unica as (
  SELECT <<Campo_Resultado>> as Resultado, count(*) as TotalesPorResultado
  FROM <<Tabla_de_Transacciones >>
  WHERE $__timeFilter(<<Fecha_Transaccion>>)
  GROUP BY <<Campo_Resultado>>
)
SELECT 'OK' as Resultado, C1.TotalesPorResultado as Total FROM Consulta_Unica C1 WHERE C1.Resultado=-1
UNION
SELECT 'ERROR' as Resultado, sum(C2.TotalesPorResultado) as Total FROM Consulta_Unica C2 WHERE c2.Resultado <> -1
```

Figura 15. Ejemplo de consulta por cantidad de transacciones en Grafana.

La figura 15 presenta una consulta SQL en Grafana [19] que retorna la cantidad de transacciones aprobadas y rechazadas en un intervalo de tiempo específico (dicha información resultante sirve para mostrar un gráfico circular). En esta consulta aparece una función propia de Grafana: “\$\_\_timeFilter”, se trata de

una función que filtra la consulta por medio del campo fecha y el rango que fue indicado en el portal de Grafana [19].



```
SELECT $__timeGroup(<<Fecha_Transaccion>>, '10m') as Momento, count(*) as 'Transacciones Boton'  
FROM <<Tabla de transacciones>>  
WHERE $__timeFilter(CreateTimestamp)  
and canal='Boton'  
GROUP BY $__timeGroup(CreateTimestamp, '10m')
```

Figura 16. Ejemplo de consulta por cantidades en Grafana.

La figura 16 presenta una consulta que proyecta las cantidades de transacciones de botones entre dos fechas y agrupadas cada 10 minutos. Se utiliza otra función de Grafana[19], “timeGroup”, dicha función permite dividir los resultados en intervalos de tiempos especificados.

Los paneles (dashboard) permiten mayor control junto al seguimiento de transacciones generadas desde los diferentes canales en tiempo real, así como también la automatización de procesos que antes se realizaban manualmente.

La implementación de los diferentes dashboard en Grafana [19], permitieron que los informes transaccionales pudieran presentarse en tiempo real. Previamente, la consulta y el informe respectivo eran manuales, tarea que demoraba dos o tres días.

## Capítulo 3 - Desarrollos realizados.

### 3.1 - Botón de pago

#### 3.1.1 - Descripción funcional

El avance de la tecnología y el avance de los medios de pagos electrónicos que fueron sustituyendo a las operaciones comerciales en billete papel, fueron apareciendo nuevas herramientas para facilitar al usuario el manejo de esas transacciones, por ejemplo, el pago electrónico de forma presencial entre el vendedor y el cliente, a través de una tarjeta de débito y/o crédito. Frente a esta nueva realidad, surge la necesidad de brindar diferentes opciones para las compras on-line.

Así como grandes marcas pueden ofrecer tiendas para vender sus productos, también pequeños vendedores publican sus productos en Internet, principalmente mediante el uso de redes sociales ofreciendo la opción de poder pagarlo online.

Ante este contexto, se vió la necesidad de contar con una herramienta que es el “botón de pago”. Un botón de pago es un link de pago directo que permite realizar cobros de forma rápida y segura. Contiene la siguiente información: un identificador del botón, un concepto o una descripción del producto o servicio que se ofrece y el monto del mismo.

El ciclo de vida de un botón de pago, consiste en su creación por el lado del vendedor, compartirlo por distintos medios, como ser, copiar la URL de pago (cada botón generado se identifica por un código único), enviar dicha URL por mail a uno o más destinatarios o compartir por redes sociales como Facebook o Whatsapp la URL del botón de pago junto a la descripción e imagen del mismo. De esa forma el comprador accede a través del enlace al formulario de pago, donde ingresa datos de pago como sus datos personales y los datos de su tarjeta para concretar la operación. A continuación, en la figura 17 se presenta gráficamente el ciclo de vida de un botón de pago.

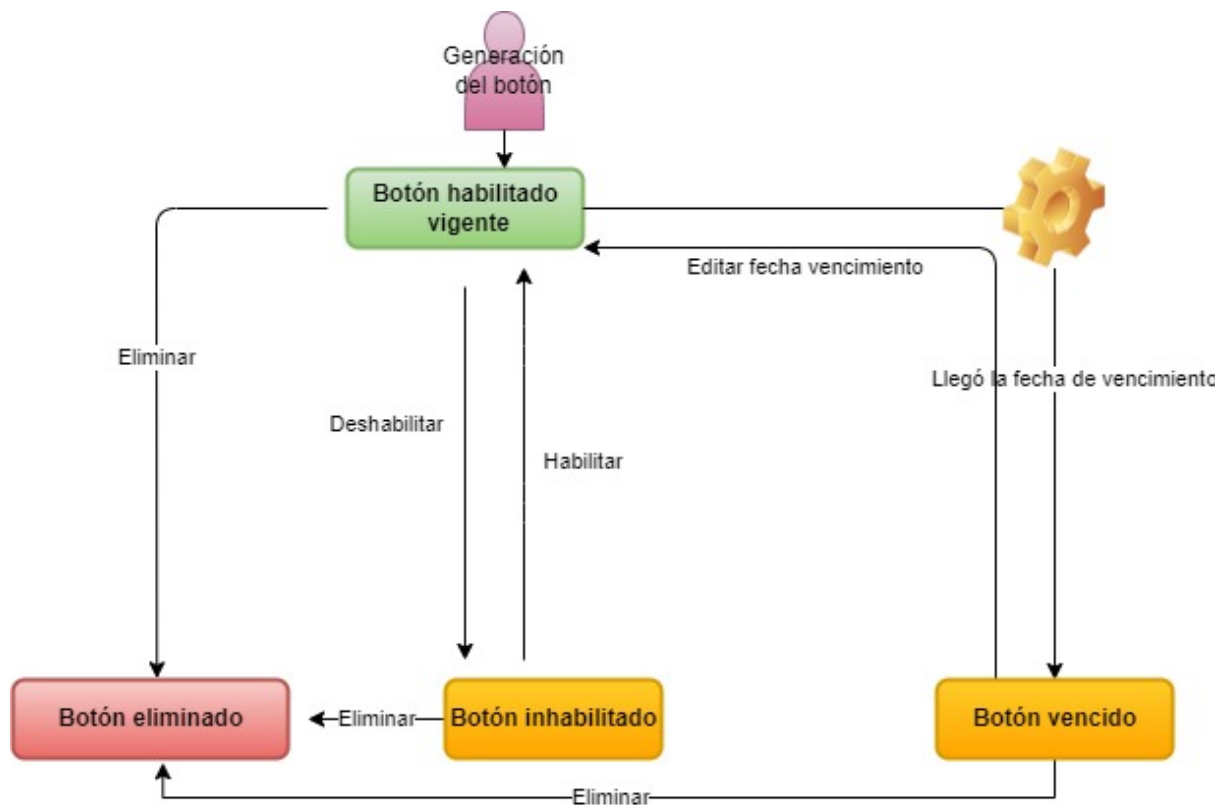




**Figura 17.** Ciclo de vida de un botón de pago.

### 3.1.2 - Estados de un botón de pago

Los botones de pago, contienen propiedades y/o características, como ser: los medios de pagos que acepta (tarjetas de débito, crédito, PagoFacil, Rapipago, etc), stock e imagen descriptiva de lo que se va a cobrar. Opcionalmente, tienen una fecha en la cual expiran y después ya no se pueden utilizar para habilitar un pago (al menos que se establezca otra fecha). Pueden inhabilitarse para pagarse temporalmente (solo pueden consultarse y editarse). Finalmente, se pueden dar de baja cuando ya no se deba utilizar (por ejemplo, ante el vencimiento o la venta del producto). A continuación, la figura 18 presenta los distintos estados por los que puede pasar un botón de pago.



**Figura 18.** Diagrama de estados de un botón de pago.

Un botón está asociado a la cuenta que lo generó (App Comercios o Portal de Vendedores).

En el proceso de creación se solicitan datos básicos como el título, la descripción y el monto, los cuales son obligatorios. Adicionalmente, posee valores por defecto que luego son configurables, como las formas de pago que acepta (débito, crédito o cupón de pago en efectivo), si tiene vencimiento y si posee una cantidad ilimitada de pagos. También puede contener una imagen que se adjunta a la descripción.

### 3.1.3 - Imágenes ilustrativas de la creación de un botón de pago

A continuación, se consideran los distintos pasos para crear un nuevo botón, y la información que le será solicitada al usuario durante dicho proceso.

La figura 19 muestra las opciones adicionales para crear el botón: formas de pago, características del botón o imagen a compartir.

Nuevo botón de pago

\$ 3.000,00

Ingresá un monto mayor o igual a \$ 3.00

Título  
Televisor Philips  
Nombre del producto/servicio que van a ver tus clientes 17 / 80

Descripción  
21 pulgadas, 3 años de uso. Impecable  
37 / 140

**Formas de Pago**  
Tarjeta, Efectivo

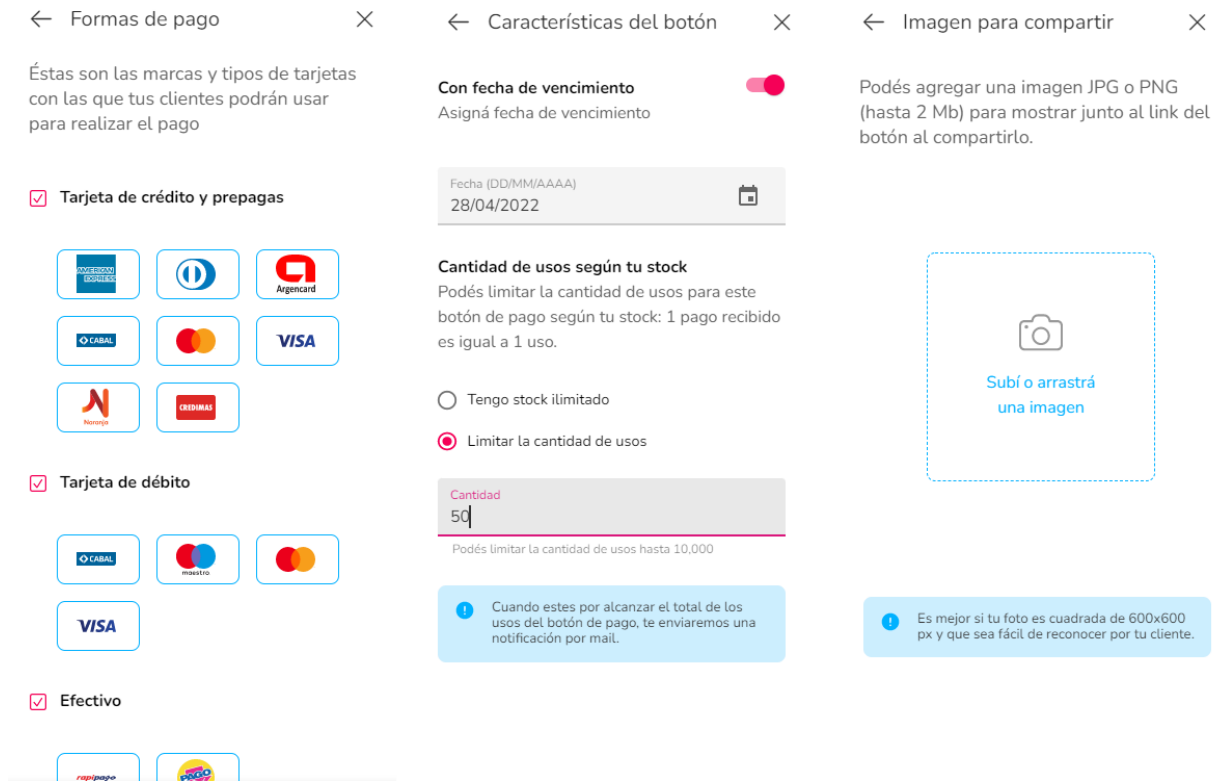
**Características del botón**  
Sin vencimiento, Stock ilimitado

**Imagen para compartir**  
JPG o PNG (Hasta 2 Mb)

CREAR BOTÓN

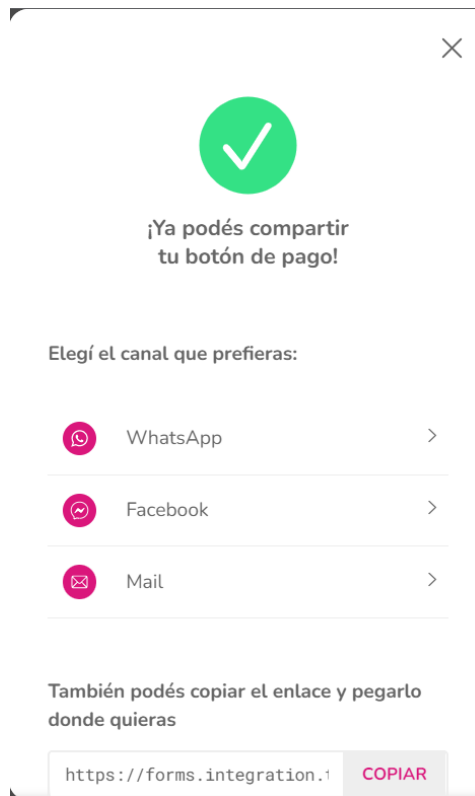
Figura 19. Pantalla general de creación de botón.

A continuación, en la figura 20, se presentan 3 diferentes capturas de pantallas durante el proceso de creación del botón.



**Figura 20.** Opciones extras para la creación del botón de pago (medios de pago, stock y vencimiento, e imagen)

Una vez creado el botón, se brinda las opciones para compartirlo. A continuación, en la figura 21, se pueden observar tales opciones.



**Figura 21.** Botón creado listo para compartir por diferentes canales

A continuación, en la figura 22 se presenta el listado de los botones creados.

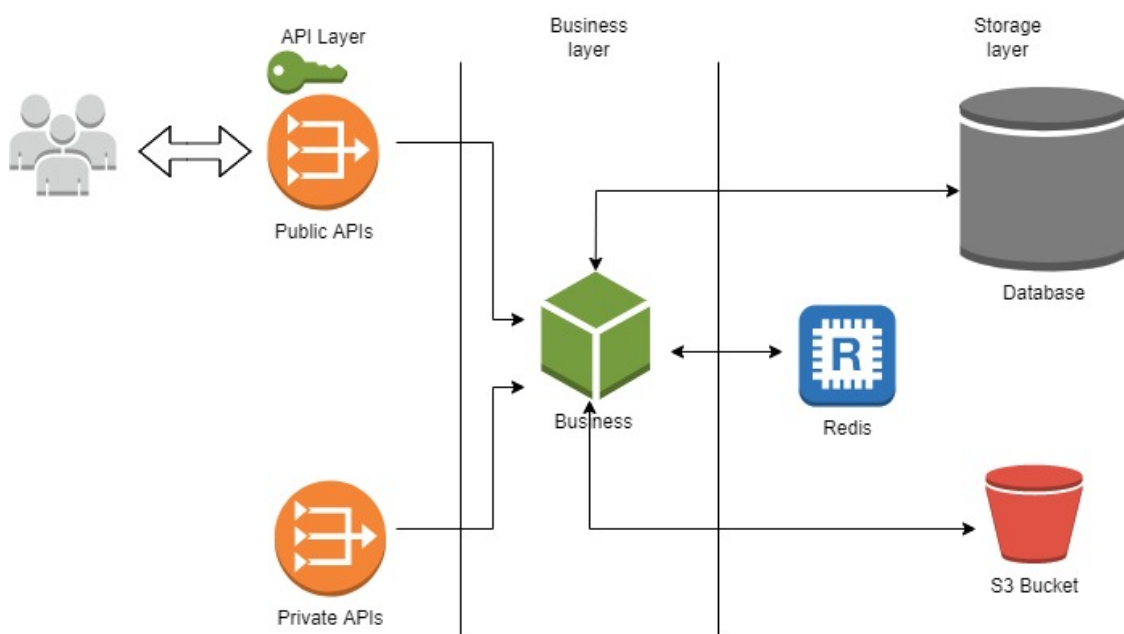
TÍTULO	MONTO	ACCIONES
Heladera Philco	\$ 60,00	
Bicicleta Cross	\$ 50,00	
BOTON FULL	\$ 3,00	
con imagen	\$ 3,02	

**Figura 22.** Listado de botones de pago creados.

### 3.1.4 - Descripción técnica de botón de pago

El microservicio de botón de pago, expone URL que son públicas y privadas. Los endpoints [42] públicos son aquellos que pueden ser consumidos desde un frontend (por ejemplo el portal de vendedores desarrollado por otro equipo). Dichos endpoints [42] son invocados y visibles en Internet, por lo que se necesita un mecanismo de seguridad. Son aquellas operaciones en las que el usuario vendedor interactúa directamente con los botones de pago (crearlos, modificarlos, listar los botones, compartirlos por email, inhabilitarlos, rehabilitarlos, darlos de baja).

Los endpoints [42] privados sólo son visibles a otros microservicios que se encuentren en el mismo ambiente que el de botón de pago. Estos sólo son visibles internamente a los otros microservicios, no se requiere autenticación, y generalmente son invocados para obtener información general del proceso de pago. A continuación, en la figura 23 se presenta gráficamente el diagrama del microservicio de botón de pago.



**Figura 23.** Diagrama del microservicio de botón

El microservicio usa una base de datos donde se almacena la información de los botones de pago que se generan. En el caso de las imágenes se utilizó el servicio de almacenamiento en la nube S3 de Amazon Web Services (AWS) [13].

Dado que en los distintos procesos en los que se interactúa con los botones de pago hay muchos accesos a la base de datos, se utiliza Redis [5] como base de datos caché de almacenamiento temporal en memoria principal, con el objetivo de agilizar los accesos.

### **3.1.5 - Conclusiones**

Debido a que fue uno de los primeros microservicios que se estaba diseñando, el desarrollo del botón de pago resultó complejo. Conforme se fue avanzando fueron apareciendo nuevas funcionalidades y requerimientos que no estaban contemplados en el diseño original. A su vez, los nuevos roles y el trabajo de equipo, se tuvo que adaptar al ritmo y flujo de trabajo propuesto y eso llevó un tiempo de adaptación.

Inicialmente, no estaba previsto que un botón de pago fuera compartido en diferentes plataformas sociales como son Facebook [27] y Whatsapp [28], lo que obligó a analizar cuál era la mejor forma de compartirlo en dichas plataformas. Se incorporó a un desarrollador con experiencia en las APIs integradoras ya que el equipo no contaba con recursos y tiempos para poder agregar esa funcionalidad.

## 3.2 - Intención de pago

### 3.2.1 - Descripción funcional

El avance del comercio electrónico vino acompañado de la aparición de distintos sitios o portales en los que las diferentes marcas ofrecen sus productos. Dichos portales están centrados en permitir búsquedas, selección de productos que ofrecen, generar un carrito de compras y su abono correspondiente.

Los carritos de compras que se generan desde una tienda son únicos tanto para el comprador como para el vendedor, éstos carritos se generan con el único fin de permitir el pago de la transacción y después se eliminan.

Estos carritos son generados por herramientas de integración para ofrecerlos en sus plataformas de e-commerce.

La información necesaria para la generación de un carrito de compra se denomina intención de pago, contiene los datos del vendedor y del comprador, y puede utilizarse únicamente para un pago.

### 3.2.2 - Diagrama de flujo

A continuación, la figura 24 muestra los diferentes estados que posee una Intención de pago.



**Figura 24.** Ciclo de vida de una intención de pago.



En el flujo de compras presentado en la figura 24, un cliente abre su portal de compras (por ejemplo una tienda de electrodomésticos). Selecciona los productos que desea adquirir, y los va añadiendo a un carrito de compras. Por medio de herramientas de Integraciones, el carrito de compra generado es enviado a la API de Intención de pago.

El microservicio de intención de pago genera una intención de pago, con los datos de la cuenta vendedora (asociada al comercio) y el importe a pagar. Cuando genera la intención de compra, la misma contiene también la página de confirmación de compra una vez que el pago ya se encuentre aprobado o rechazado, a fin de confirmar la compra y poder continuar en el sitio en el que se encontraba.

Una vez generada la intención de pago, el microservicio retorna la URL del formulario de pago a fin de ingresar los datos del pago (los datos de la tarjeta o las tarjetas que tenga el comprador si ya se encuentra registrado). Posteriormente, se redirige a una página de confirmación.

### 3.2.3 - Imágenes ilustrativas de creación de intención de pago

En esta sección se presenta el proceso de creación de la intención de pago desde una tienda online a través de un carrito de compras, y su posterior pago.

A continuación, se presentan capturas de pantallas referentes al proceso de compra utilizando un carrito de compras. La figura 25 presenta la oferta de productos de la tienda.



Figura 25. Pantalla de selección de producto de una tienda integrada.

A continuación, la figura 26, presenta el carrito de compras con los productos seleccionados.

**ital**

**CARRITO DE COMPRAS**

PRODUCTO	SUBTOTAL
 Protector de celular ENVÍO GRATIS \$15,00 - 2 +	\$30,00 
 Heladera ENVÍO GRATIS \$69.999,00 - 1 +	\$69.999,00 

Subtotal (sin envío) : **\$70.029,00**

**Total: \$70.029,00**  
O hasta 12 cuotas sin interés de \$5.835,75

[VER MÁS PRODUCTOS](#) [INICIAR COMPRA](#)

Figura 26. Carrito de compras con productos seleccionados

Una vez confirmado el carrito de compra, se solicita el ingreso de los datos del comprador, tal como lo muestra la figura 27.

## ENTREGA

 Nos comunicaremos para coordinar la entrega del producto

## DATOS DEL DESTINATARIO

Nombre	Juan
Apellido	Perez

## DOMICILIO DEL DESTINATARIO

Calle	Olleros		
Número	3412	<input type="checkbox"/> Sin número	Departamento (Opcional)
Barrio (Opcional)			
Ciudad	Capital Federal		
Código Postal	1427	Provincia	Capital Federal

[Cambiar CP](#)

## DATOS DE FACTURACIÓN

Mis datos de facturación y entrega son los mismos

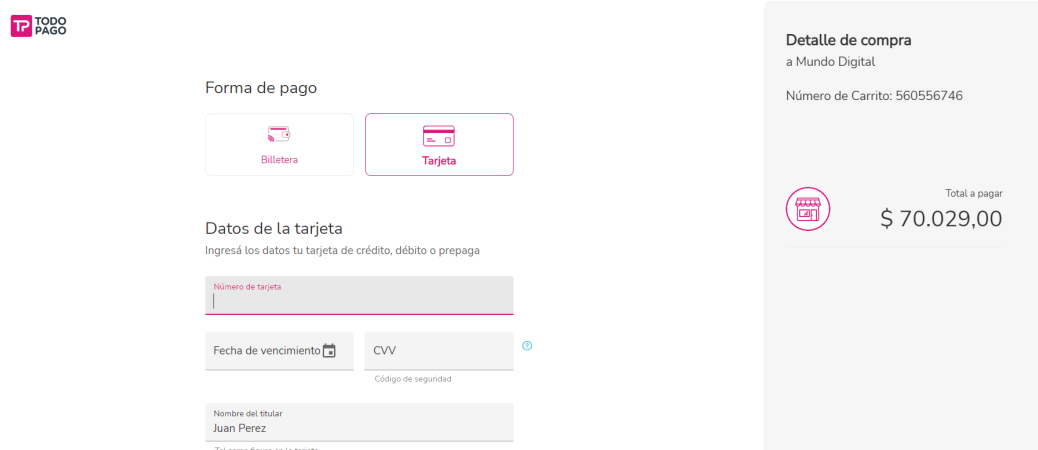


CONTINUAR

	Protector de celular x 2	\$30,00
	Heladera x 1	\$69.999,00
Subtotal		\$70.029,00
Costo de envío		A convenir
<b>Total</b>		<b>\$70.029,00</b>

Figura 27. Ingreso de datos del comprador

A continuación, se llama desde el portal a la SDK [25], la cual invoca a la API de Intención de pago para crearla con los datos ingresados y el carrito de compra correspondiente. Posteriormente, se ingresa al formulario de pago, tal como se presenta en la figura 28.



**TP TODO PAGO**

Forma de pago

Billetera  Tarjeta

Datos de la tarjeta

Ingresá los datos tu tarjeta de crédito, débito o prepaga

Número de tarjeta

Fecha de vencimiento  CVV

Código de seguridad

Nombre del titular  
Juan Perez

Tal como figura en la tarjeta

Detalle de compra  
a Mundo Digital

Número de Carrito: 560556746

Total a pagar  
**\$ 70.029,00**

Figura 28. Formulario de pago esperando datos de la tarjeta para pagar el pedido.

En la figura 29, se presenta la confirmación del pago.

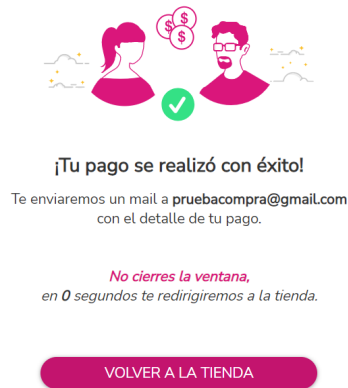


Figura 29. Finalización del pago y redirección a la tienda

Finalmente, se redirige a la tienda desde donde se creó la intención de pago y se informa el resultado correspondiente a dicho pago. A continuación, la figura 30, presenta el resumen de la compra resultante.



Figura 30. Cierre del pago en la tienda.

### 3.2.4 - Descripción técnica

El microservicio de intenciones de pago brinda endpoints [42] públicos, los cuales cuentan con un mecanismo de seguridad por Oauth [35] y endpoints [42] privados para ser utilizados internamente.

Los principales endpoints [42] públicos que brinda a las SDK [25] son:

- Crear intención de pago: Se envían los datos de la tienda vendedora (validados por el token bearer de autenticación [14]), los datos del carrito de compras, con el monto total y los datos del comprador (sin los datos de la tarjeta, los cuales se completan en el formulario). Además, se ingresa la URL de destino ante un pago exitoso o un pago erróneo. Posteriormente, se retorna la dirección del formulario de pago con la intención recién creada. La misma expira luego de 15 minutos de haber sido generada.

Una vez que se confirma el pago, desde el formulario se redirige a la página de confirmación, a la cual se le añade el número de transacción aprobada.

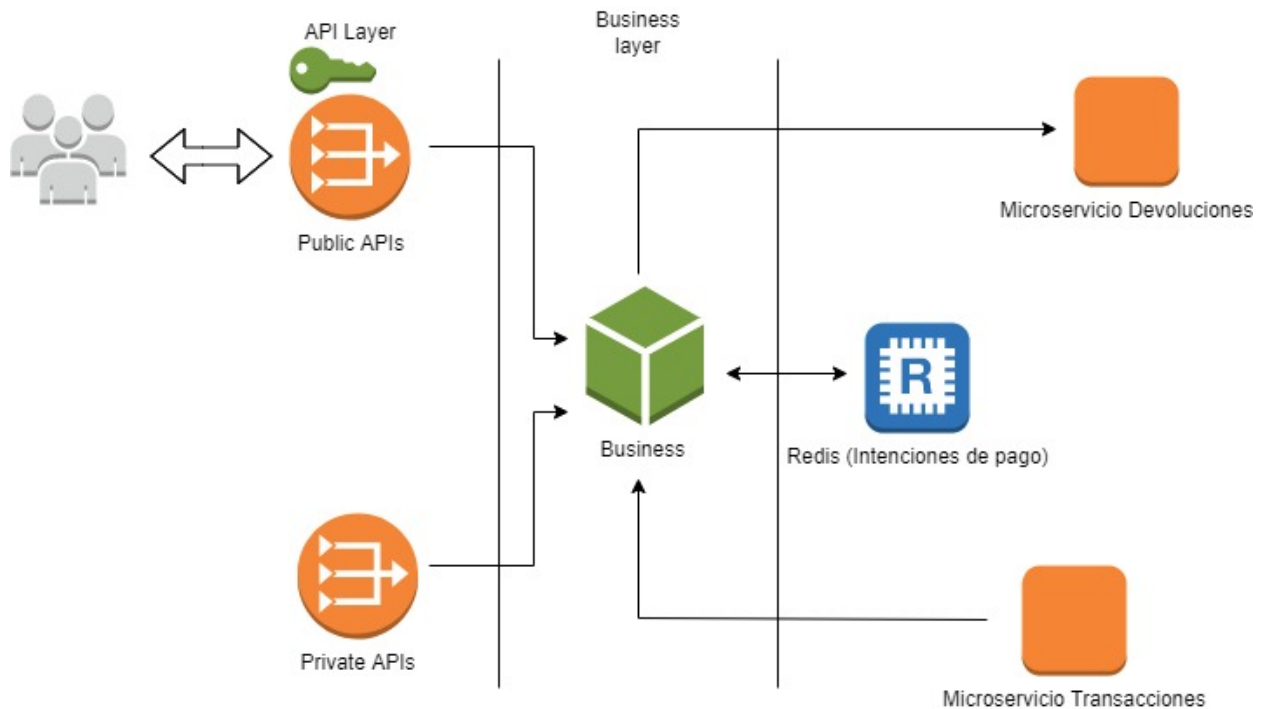
- Obtener resultado de operación: Retorna los datos de una transacción por medio de su número de transacción, los datos sensibles se ocultan.
- Realizar devolución de la compra: Para el administrador de una tienda se brinda la posibilidad de poder realizar la devolución (total o parcial) de una compra.

Los endpoints privados que se exponen son:

- Obtener datos de la intención de pago: Dado un número de intención de pago, se recuperan los datos de la misma. En este caso, es utilizado principalmente por el microservicio de formulario de pago (que será analizado posteriormente), para poder mostrar los datos de la intención generada en la página del formulario.

- Eliminar intención de pago: Dado un número de intención de pago, se elimina la misma. Esta operación es usada principalmente luego del pago de la intención, haya sido exitoso o no.

A continuación, la figura 31 presenta el diagrama del microservicio de intención de pago.



**Figura 31.** Diagrama del Microservicio de intención de pago

El microservicio utiliza Redis [5] para las intenciones de pago, que tienen un tiempo de vida máximo y luego expiran.

Los endpoints [42] públicos funcionan con un token Bearer [14] como mecanismo de seguridad, desde el portal de vendedores se puede generar para la tienda un Client y un Secret [41] para que puedan configurar en su SDK [25] y puedan generar el token que se usará para consumir los endpoints [42].

El endpoint [42] que genera una intención de pago retornará:

- Un identificador público de la intención de pago (el que se utiliza para consultarlo o pagar en el formulario)

- Un identificador privado de la intención de pago. Se retorna a la SDK [25] que utiliza el portal de ventas y servirá para poder consultar el estado de una transacción.

En el momento que se abona una intención de pago, se genera un identificador de respuesta, el cual se envía a la tienda al finalizar el pago. Este proceso es el que se encarga de persistir la transacción en la base de datos.

Si la tienda necesita información del pago, con el código de respuesta se llama al endpoint [42] de obtener resultado de operación, y luego busca en la base de datos de transacciones.

El tercer endpoint [42] de devoluciones, además del código de respuesta, necesita el identificador privado que posee la intención de pago. Por ello, la importancia de la SDK [25] que está integrada a los portales de compra, que almacena estos valores conforme se va operando. Desde aquí, se llama a otro microservicio de devoluciones encargado de procesarla.

### **3.2.5 - Conclusiones**

La generación de las APIs de intención de pago, fue una forma novedosa de integrar las tiendas online con Todo Pago sin perder la seguridad que se otorga en cada una de sus transacciones.

Este tipo de integración obligó a la contratación de roles específicos en diferentes lenguajes para cubrir el desarrollo y mantenimiento del SDK [25], en lenguajes de programación como .NET [29], Python [30], PHP [31], Ruby [32], y en especial de PHP [31], en donde se buscó la integración de plataformas open source [40] y gratuitas, que aporten a la utilización masiva, ya que son servicios de simple instalación y/o configuración.

A su vez, la búsqueda de especialistas reforzó al equipo con nuevos puntos de vista hacia el cliente (emprendedores y pequeños clientes). Esta nueva visión permitió encarar los desarrollos buscando una nueva integración para los servicios brindados.

## 3.3 - Formulario de pago

### 3.3.1 - Descripción funcional

El formulario de pago se utiliza para abonar tanto intenciones de pago como botones de pago. Para el formulario de pago se realizó un sitio web para interactuar con el comprador.

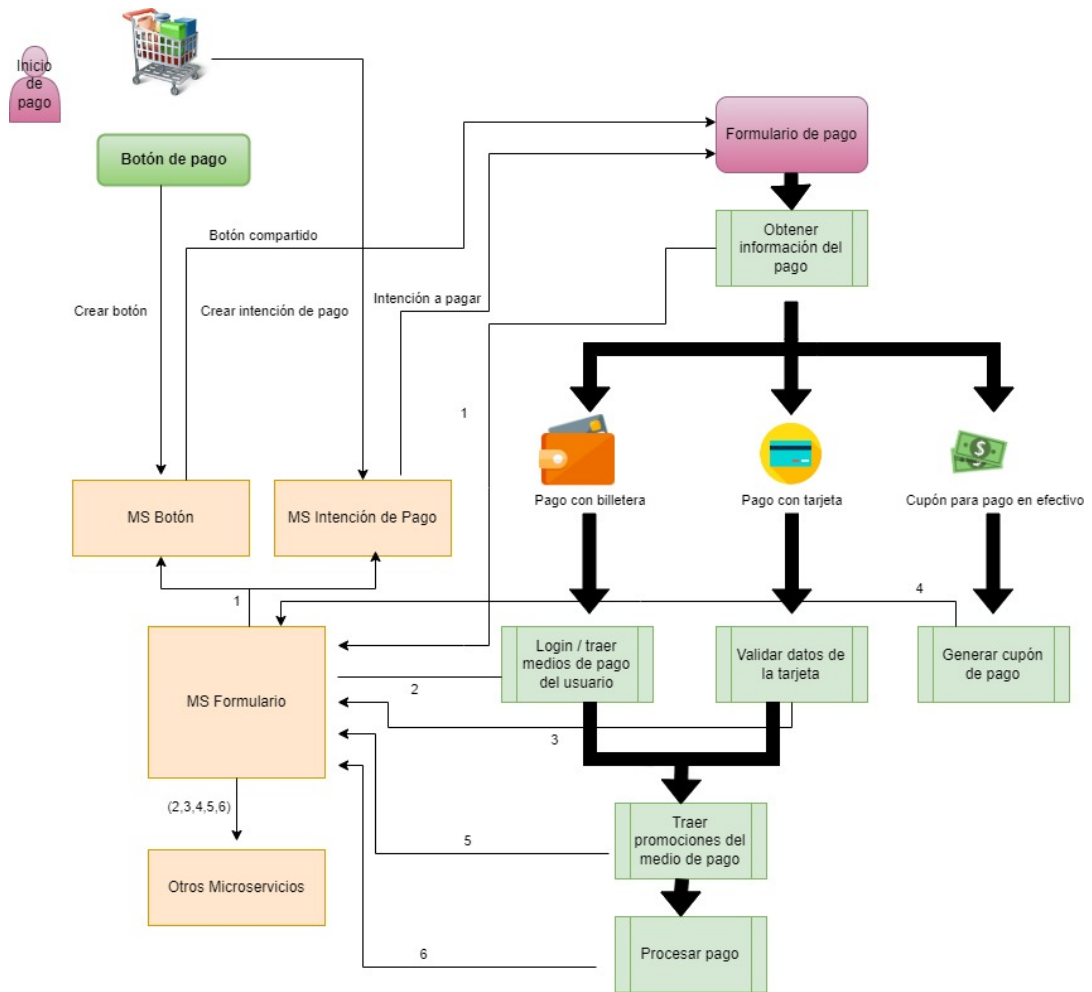
El formulario de pago consta también de un microservicio que brinda las siguientes operaciones:

- Recuperar los datos que se muestran al usuario.
- Determinar si el botón de pago o la intención de pago permiten utilizar el medio de pago seleccionado.
- Retornar el listado de promociones de pago en cuotas que ofrece el vendedor asociado al botón de pago o intención de pago y al medio de pago seleccionado.
- Permitir abonar un botón de pago, con las cuotas elegidas y los datos ingresados en el formulario de pago, donde serán procesados por otro microservicio.

### 3.3.2 - Diagrama de flujo del formulario de pago

A continuación, la figura 32 presenta un diagrama de flujo con las funcionalidades del formulario de pago.





**Figura 32.** Diagrama de flujo del formulario de pago

La figura 32 describe los diferentes flujos de acciones que el usuario realiza cuando interactúa con el formulario de pago. Dichas acciones son:

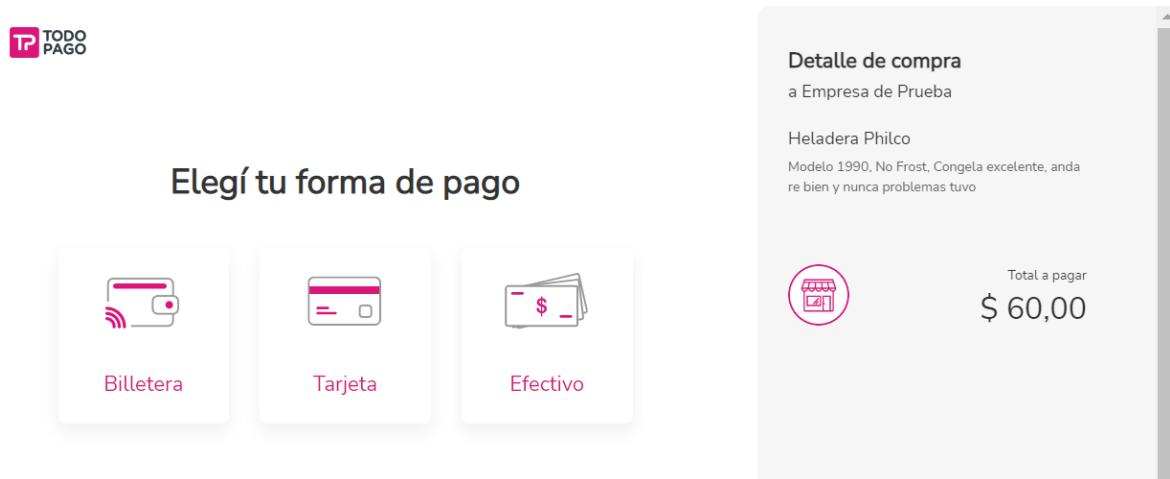
- 1) Obtener información del botón de pago o intención a pagar.
- 2) Obtener las tarjetas de pago guardadas para el usuario registrado
- 3) Validar los datos de las tarjetas ingresadas.
- 4) Generar un cupón de pago.
- 5) Traer las diferentes opciones de pago (cuotas y promociones) según el medio de pago seleccionado.
- 6) Procesar el pago.

En este caso, se explicará el flujo 1, donde el microservicio interactúa con los microservicios de botón de pago e intención de pago. Respecto de las validaciones e interacciones que se efectúan en los flujos 2 a 6, se realiza con microservicios que

fueron desarrollados por otros equipos, los cuales no son objeto de consideración en el presente trabajo.

### 3.3.3 - Imágenes ilustrativas de pago con formulario

A continuación, la figura 33, presenta la pantalla inicial del formulario para seleccionar la forma de pago.



**Figura 33.** Pantalla inicial del formulario de pago

En caso de seleccionar la opción de billetera, el comprador debe estar registrado en el sistema. Una vez autenticado puede realizar el pago utilizando alguna de sus tarjetas. A continuación, la figura 34 presenta el listado de tarjetas pertenecientes a un comprador.

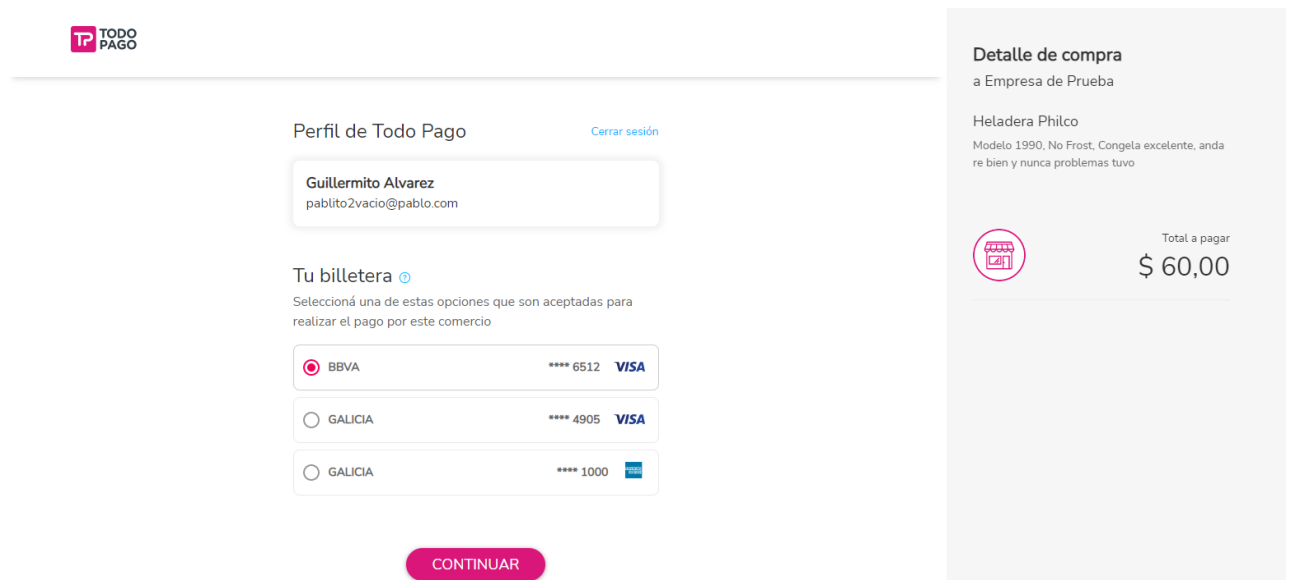


Figura 34. Selección de tarjetas. Opción de pago con Billetera.

En el caso de haber seleccionado la opción de tarjeta en la pantalla inicial, el comprador debe ingresar los datos de su tarjeta, no hace falta que tenga cuenta registrada. A continuación, la figura 35 presenta una captura de pantalla que contiene el formulario para cargar los datos de la tarjeta.

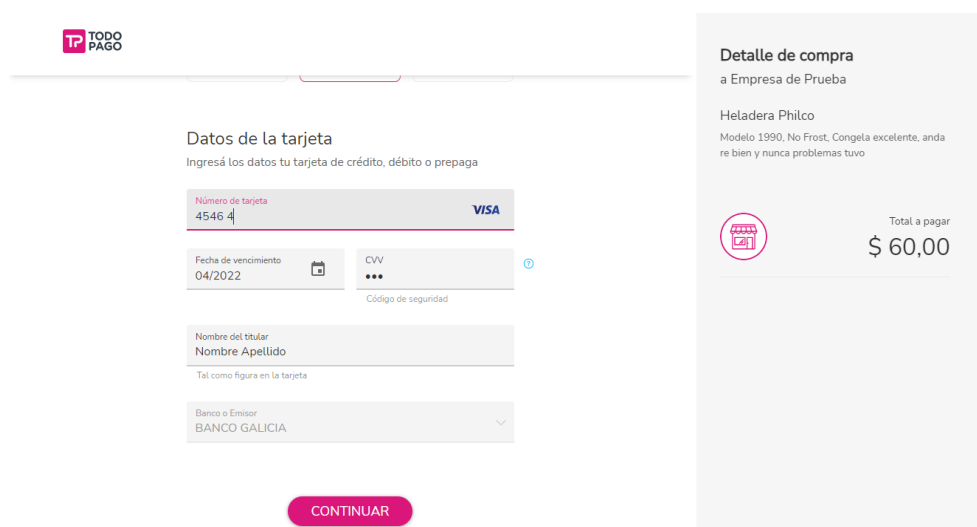


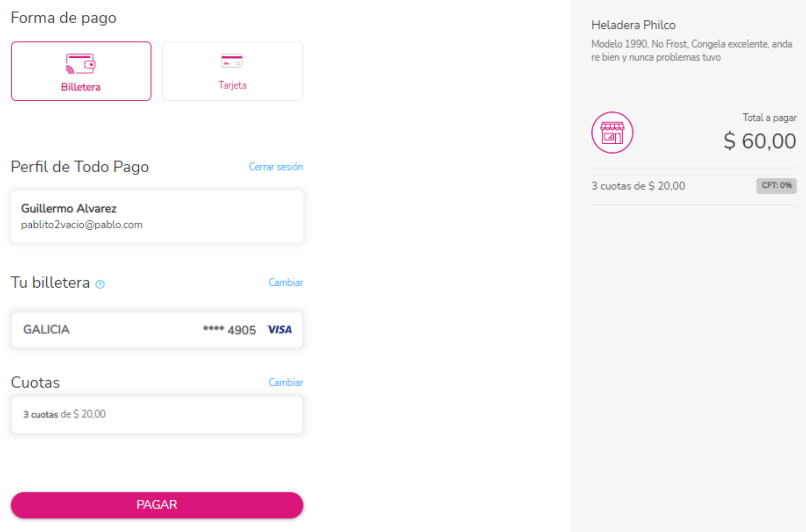
Figura 35. Ingreso de datos de la tarjeta.

Luego de seleccionar una tarjeta de la billetera, o bien ingresar la tarjeta manualmente, se procede a seleccionar las cuotas con las que se puede pagar. Sobre la parte derecha de la pantalla se irá mostrando el interés y el costo financiero si aplicaran a la cuota. A continuación, en la figura 36 se presenta una captura de pantalla en donde se puede observar el detalle de cuotas e interés.

The screenshot displays the 'TODO PAGO' app interface. On the left, the 'Forma de pago' section offers 'Billetera' (highlighted) and 'Tarjeta'. Below, the user profile for 'Guillermo Alvarez' is shown. The 'Tu billetera' section displays a 'GALICIA' card with a 'VISA' logo. The 'Cuotas' section allows selection of 1, 2, or 3 payment installments, with the 3-installment option selected. A 'CONTINUAR' button is at the bottom. On the right, a 'Detalle de compra' summary shows a total of \$60,00 and 3 installments of \$20,00 each, with a 0% interest rate (CFT: 0%).

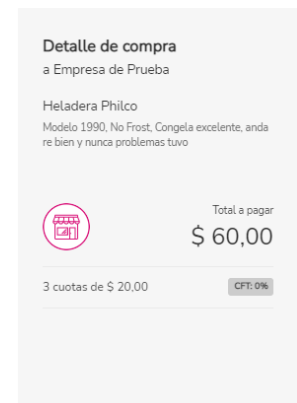
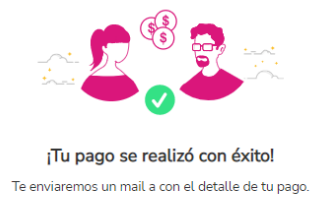
**Figura 36.** Selección de las cuotas para la compra.

Una vez seleccionado el plan de la cuota, se muestra un resumen con los datos de la compra para confirmar el pago. A continuación, en la figura 37, se presenta una captura de pantalla con el resumen de compra previo a confirmar el pago.



**Figura 37.** Pantalla previa de confirmación del pago

Finalmente, se muestra la confirmación del pago realizado. Además, se envía un correo electrónico tanto al vendedor como al comprador. A continuación, la figura 38, presenta la pantalla de confirmación de la transacción.



**Figura 38.** Pago exitoso.

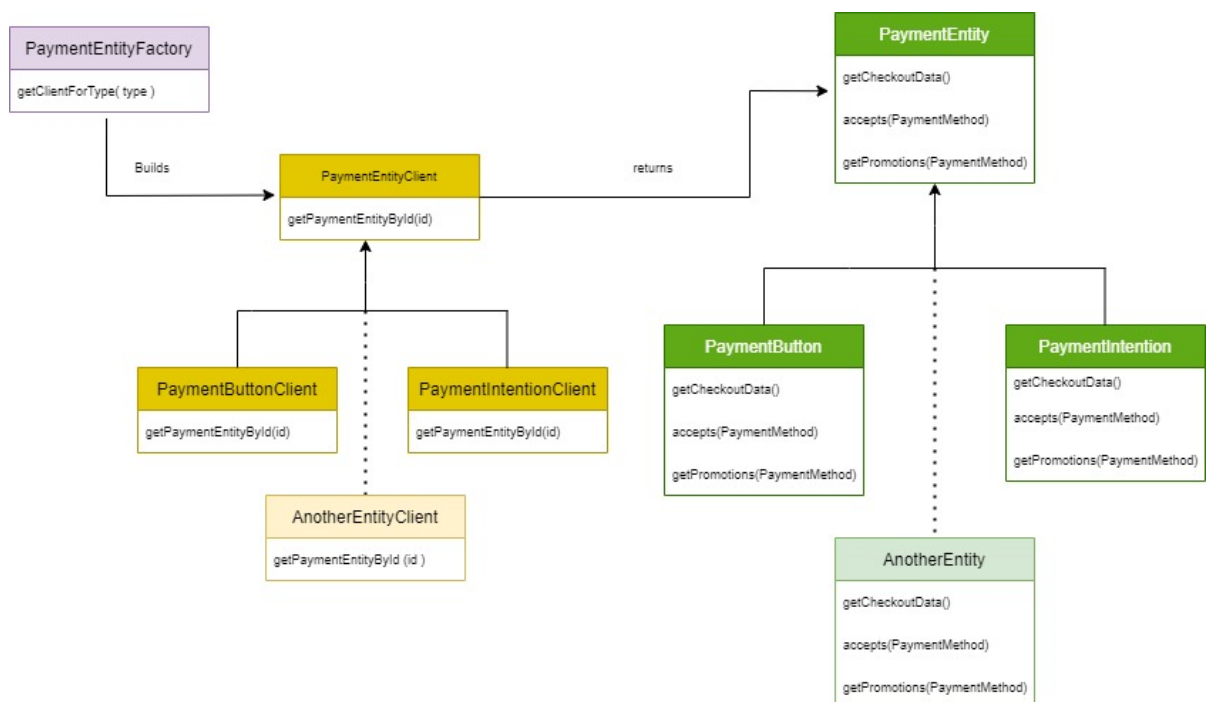
En el caso de tratarse del abono de una intención de pago, se mostrará la pantalla de confirmación unos segundos y luego se redirige al sitio de la tienda que generó la intención, para confirmar allí el pago.

### 3.3.4 - Descripción técnica

El microservicio de formulario de pago brinda endpoints [42] públicos, cuentan con un mecanismo de seguridad con comunicación HTTPS [26] y también por Oauth [35], en especial los que manejan información sensible, como por ejemplo, los que se utilizan para iniciar sesión y pagar con la billetera (tarjetas ya cargadas).

La totalidad de los endpoints [42] son consumidos por el sitio web del formulario de pago y exponen las operaciones detalladas previamente en el flujo de ejecución de dicho formulario.

La operación que se va a describir es la que obtiene información del botón de pago o intención de pago. A continuación, en la figura 39 se presenta un diagrama UML [37] que representa una parte del modelo de objetos que se utilizó para llevar a cabo la operación de “obtener información”.



**Figura 39.** Diagrama UML representando las clases de botón e intención de pago, utilizadas en el microservicio de formulario.

El botón o intención de pago, se implementaron utilizando herencia sobre una interfaz que se denominada “PaymentEntityClient”, tal como se observa en la figura

39. Esta interfaz permite recuperar los datos de las respuestas de los microservicios. La respuesta se representa como un objeto que se ha denominado “PaymentEntity”, como muestra en la figura 39, el cual representa una intención o bien un botón de pago. Este objeto brinda las siguientes operaciones:

- Retornar un objeto con la información de la entidad de pago que se va a mostrar en el formulario de pago.
- Determinar medios de pagos aceptados.
- Listar las promociones disponibles para una entidad de pago y medio de pago determinado.

La Clase “PaymentEntityFactory” presentada en la figura 39, sigue el patrón de diseño Factory [38], de acuerdo a los parámetros de entrada, se puede generar un “PaymentEntityClient” de botón o de intención de pago, siendo este un modelo perfectamente adaptable y escalable a las modificaciones que puedan surgir.

### **3.3.5 - Conclusiones**

Previamente, se contaba con una versión monolítica del formulario de pago, en el cual se tenían pocas opciones, no era adaptable a dispositivos móviles y tenía problemas de usabilidad.

En este nuevo formulario, se desarrolló un “front end” en React [39] y se focalizó en los diseños de las pantallas. El equipo de experiencia de usuario tuvo un rol muy importante en definir los flujos para darle a los compradores una mejor experiencia.

Se brindó prioridad a la experiencia del usuario simplificando los flujos de carga de datos, por ejemplo, se eliminó el requerimiento de la doble carga de datos por parte del comprador.

## 3.4 - Transferencias 3.0

### 3.4.1 - Descripción funcional

En el año 2020, se implementó la iniciativa Transferencias 3.0, un sistema innovador que impulsa los pagos con transferencia mediante códigos QR [33] estandarizados. Esta iniciativa introdujo el concepto de QR Interoperable, lo que significa que dicho QR [33] es válido para ser procesado el pago con billeteras de diferentes marcas. Esto permite una mejor flexibilidad a la hora de los pagos permitiendo realizar las operaciones sin tener la marca de billetera como limitación. También se brinda una mayor seguridad en las operaciones comerciales, siendo el principal desafío reemplazar el uso de dinero en efectivo.

### 3.4.2 - Diagrama de flujo de Transferencias 3.0

A continuación, la figura 40 representa el diagrama general del funcionamiento de Transferencias 3.0.

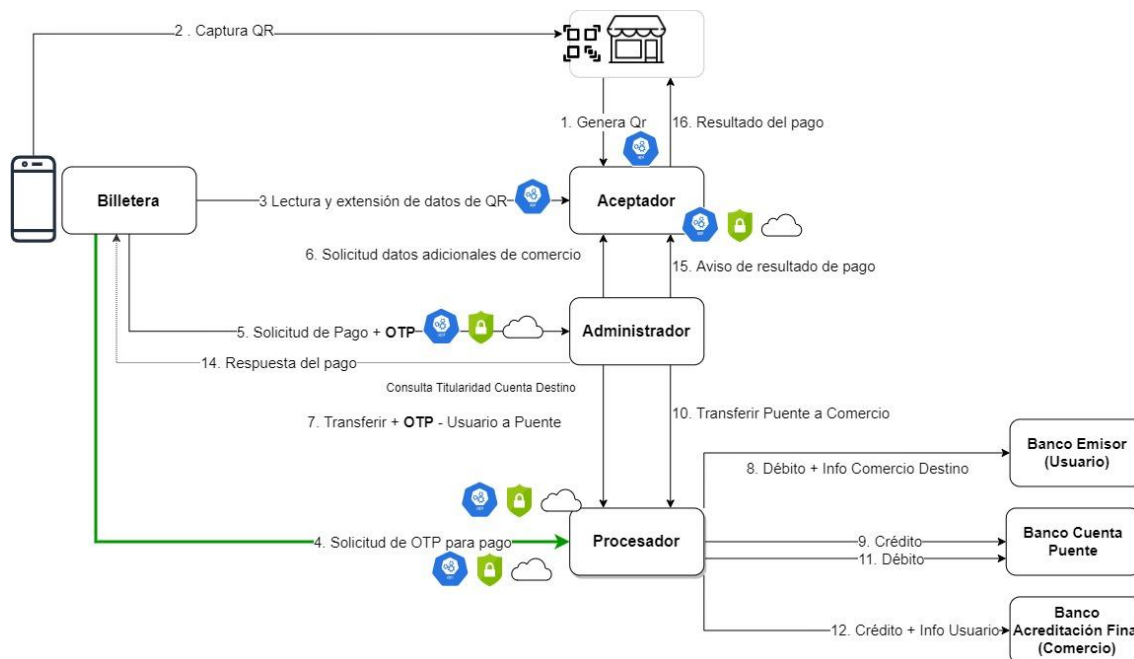


Figura 40. Diagrama de flujo de Transferencias 3.0

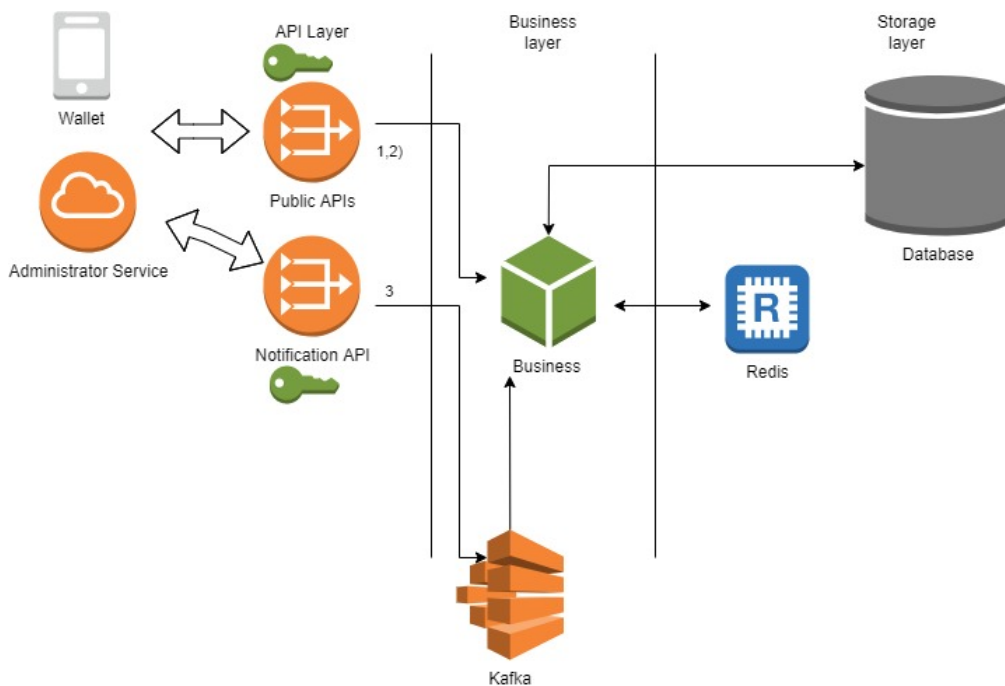
Los principales componentes de esta integración son:

- Billetera: Dispara el evento de lectura de QR [33] y pago de la transacción.
- Administrador: Recibe la orden de pago y genera las transferencias.



- Procesador: Realiza las operaciones de débito y crédito de las cuentas.
- Aceptor: Genera la intención de pago por QR, y persiste los resultados de la transacción en la cuenta vendedora.

Las funciones del lado del aceptor, mencionado previamente, es generar intenciones de pago por QR [33] y persistir los resultados de las transferencias de dichos pagos. A continuación, en la figura 41, se presenta el diagrama del microservicio del aceptor de Transferencias 3.0.



**Figura 41.** Diagrama de flujo del microservicio aceptor, de Transferencias 3.0

En la figura 41 se pueden diferenciar tres operaciones numeradas que forman parte del proceso o flujo de pago. Estas operaciones son:

- 1 - Crear una intención de pago.
- 2 - Obtener datos del comercio.
- 3 - Aviso del resultado de pago.

A continuación se detalla cada una de estas operaciones.

1) Crear una intención de pago: Se especifican datos de la intención que se va a crear como ser el identificador del comercio, el canal por el que se crea la intención, y finalmente el monto (valor y moneda). Se retorna un identificador único de transacción que se utilizará en el resto del flujo.

2) Obtener datos del comercio: operación que es invocada desde el administrador al inicio del flujo de pago. En la misma, se retornan datos de la cuenta vendedora, como ser el CVU (Clave Virtual Uniforme) o CBU (Clave Bancaria Uniforme) donde se debe acreditar la transferencia, el rubro de la cuenta, para temas impositivos y si se le debe cobrar comisión de venta, en el caso de que la cuenta haya superado un cierto monto de venta.

3) Aviso del resultado de pago: Una vez que se realizó el pago (o falló la transacción) se recibe desde el administrador todos los datos de la misma, datos del comprador, monto original de la compra, monto que se acredita, descuento de las comisiones, retenciones sobre la venta y el código de autorización. Con estos datos, se guardan los detalles de la transacción, y se notifica mediante correo electrónico la acreditación del pago.

Un flujo de contingencia ocurre en caso que, durante una transacción el administrador le solicite más datos del comercio y luego no realice la notificación del resultado de la operación. En caso que no se reciba una notificación dentro de una determinada cantidad de tiempo, el aceptador invocará al administrador un número de veces en un intervalo de tiempo (ambos configurables) para consultarle el estado de la transacción, hasta que se reciba una respuesta (si fue rechazada o aprobada) y en caso de haber superado el número máximo de intentos se persistirá como pendiente de conciliar.

### **3.4.3 - Imagen ilustrativa de un QR de Transferencia 3.0**

Una de las formas de generar un QR [33], es a través del portal vendedor, o desde un smartphone donde se cuente con la app de comercios. Se especifica el monto a cobrar y se muestra QR [33] para que desde un smartphone se lea el QR y se pueda pagar con la billetera correspondiente. A continuación, en la figura 42 se

presenta una captura de pantalla que muestra un código QR [33] generado por Transferencias 3.0.



**Figura 42.** Generación de un QR Interoperable desde el Portal para poder pagarse con Transferencias 3.0

#### **3.4.4 - Descripción técnica del funcionamiento de Transferencias 3.0**

Para almacenamiento de las Intenciones de pago por QR [33] se utiliza Redis [5], debido a que son creadas temporalmente en memoria principal para su rápido acceso. Las intenciones de pago QR [33] en Redis [5] tienen un tiempo de vida prefijado. Una vez que llega una notificación del pago de esa intención, la misma automáticamente se elimina, y se persisten todos los datos en la transacción. En caso que haya vencido la intención de pago, puede tener dos comportamientos:

- Si se ejecutó previamente un pedido de más información del comercio, significa que la billetera disparó el pago del QR [33] y por ese motivo el administrador ya invocó al aceptador, y en el caso de que no llegue la notificación se inicia un proceso de consulta al administrador para conocer el estado de la transacción.

- Si no se ejecutó el pedido de más información al aceptador, quiere decir que el QR [33] leído, nunca se intentó pagar, y la intención caduca sin iniciar ningún proceso adicional.

Finalmente, cuando se notifica un pago desde el lado de administrador, se realiza la persistencia de manera asíncrona, es decir, ni bien llega la notificación se da una respuesta inmediata de recibido y simultáneamente se manda el mensaje a Kafka [6].

El mismo Aceptador tiene el proceso que consume el tópico de Kafka [6], toma los datos de la intención que se está pagando, los junta con los datos del pago (datos del comprador, estado de la transacción, código de autorización y mensaje de respuesta) y persiste los datos en la base de datos transaccional. Luego, se notifica al vendedor de la acreditación del pago.

En caso que el dato de la transacción en Redis [5] alcance su tiempo de expiración, por no haber llegado una notificación de pago, se ejecuta un evento para consultar al administrador el estado de la transacción en curso.

Luego, se persiste la aprobación o rechazo según la respuesta del administrador y se cierra el proceso de pago. En caso que el administrador no esté disponible y la respuesta sea errónea, se vuelve a repetir el proceso, poniendo nuevamente un tiempo de expiración, e incrementando la cantidad de intentos, para asegurar que esos nuevos intentos se ejecuten un número configurable de veces. Si no se incluye una respuesta satisfactoria se graban los datos de la transacción con estado pendiente.

### **3.4.5 - Conclusiones**

El desarrollo de Transferencias 3.0 tuvo que afrontar distintos obstáculos.

Previo a iniciar el desarrollo, se llevaron a cabo reuniones entre distintos equipos para definir los contratos de mensajería entre componentes. A eso se sumaron cambios y definiciones en la documentación que fueron surgiendo conforme avanzaba el desarrollo. También hubo que considerar la definición de casos de pruebas entre todas las marcas de billeteras.

La salida del producto al mercado marcó un hito importante después del esfuerzo invertido por los equipos que participaron en el proyecto. La solución propuesta tuvo un impacto positivo que se vió reflejado en el volumen transaccional obtenido.

## Capítulo 4 - Conclusiones

En estos últimos años, el comercio electrónico ha experimentado un crecimiento notorio.

En este contexto, se presentaron nuevos desafíos que dieron lugar a la búsqueda de nuevas metodologías que permitan la implementación de soluciones para un mercado altamente competitivo. Se han definido prioridades y se han agregado nuevos desarrollos para poder lograr soluciones innovadoras con el objetivo de satisfacer las necesidades del cliente.

Se utilizó la metodología Scrum [10] ya que al ser muy flexible y centrada en el cliente permitió una adaptación a los cambios que fueron surgiendo en los requerimientos del usuario.

Ante ciertos inconvenientes, fue necesario brindar soporte, para ello, se contó con distintas herramientas de registro y monitoreo de las transacciones, a fin de prevenir posibles fallos, o bien dar respuesta rápida a los reclamos que llegasen.

Además, para cada uno de los desarrollos, se tuvo en cuenta la eficiencia en cuanto a disponibilidad y tiempos de respuestas. Para lo cual se realizaron pruebas de rendimiento y se realizaron los ajustes correspondientes.

Al tratarse de un producto de gestión de transacciones monetarias e información sensible en las cuentas de los usuarios, cada uno de los desarrollos tuvo que adaptarse a las normativas financieras vigentes, y en todos los casos se debió considerar todos los protocolos de seguridad informática necesarios para evitar fraudes o accesos no autorizados.

En el caso de los botones de pago y carritos de compras se propusieron mejoras tanto en el microservicio como en las distintas SDK [25], a fin de poder mejorar la personalización por parte de la empresa, pero sin que esto le quite seguridad al uso de la herramienta, también se ha considerado utilizar herramientas analíticas con el fin de detectar nuevas necesidades (nuevas para el cliente y/o para la plataforma) y con eso brindar mejor servicio.

En cuanto al formulario de pago, se consideraron realizar reuniones con el equipo de experiencia de usuario para poder mejorar la forma de interacción con los

clientes al momento de realizar el pago; por ejemplo, mostrar la información de la tarjeta mientras se está ingresando su número, presentando una ilustración con la entidad y el banco a la que pertenece. También se analizó, para los carritos de compra, mejorar la presentación en el formulario, mostrando en detalle cada ítem de compra. Por último se tuvo en cuenta permitir utilizar el saldo virtual existente en la billetera virtual y también, incorporar la posibilidad de combinarlo con las tarjetas que se tengan cargadas en dicha billetera.

En el caso de Transferencia 3.0 para los pagos con QR [33], se planificó la opción para que el vendedor pueda realizar devoluciones de los pagos recibidos y entregar los QR [33] de manera impresa, con el logo de marca de Todo Pago.

El mercado de los medios de pagos digitales se encuentra en constante innovación, incorporando otros modelos de negocio, interoperabilidad entre billeteras y herramientas que faciliten el manejo del dinero. Sin embargo pocos actores pueden llevar el volumen y muchos luchan por sobrevivir, segmentando su mercado o con la idea de convertirse en un importante competidor a corto plazo

Frente a tales avances, el desafío del mercado es cada vez más atomizado y dinámico, que obliga a ir un paso adelante de la competencia, con el objetivo de brindar siempre un servicio de excelencia para los vendedores y compradores. Para ello se busca un cambio de paradigma en donde no solo hay que lograr el contacto permanente con los usuarios, sino también utilizar los datos generados por estos para estudiar su comportamiento y adelantarse a sus necesidades.

El aumento del costo de financiación, golpeó a muchas pequeñas empresas que financiaban su crecimiento con deuda. La caída de las criptomonedas [46] afectó a las billeteras que llevaban un negocio donde estos medios de pagos alternativos le brindaban una ventaja al no estar regulados. Las empresas de medios de pago deben adaptarse a estos cambios continuos de contexto para poder sostenerse.

Los desarrollos aquí presentados son solo herramientas para facilitarles a los usuarios el uso de estos medios, pero ninguno se presenta como disruptivo sino como diferente. Y es en eso que deberían enfocarse las empresas que quieran sobrevivir.

## Capítulo 5 - Trabajos futuros

Con un enfoque hacia el crecimiento vertical de la compañía y la capacidad de desplegar nuevos servicios se decidió la creación de un marketplace [47] multitienda propio, donde por medio de soluciones innovadoras y reducción de costos se puede brindar acceso a servicios propios, para los pequeños comercios de igual manera que MercadoPago [45].

Esta solución va a estar orientada a las compras con el uso desde el celular, haciendo uso y provecho de virtudes como geolocalización, personalización y agilidad, orientado a un público joven y de uso intensivo y casi exclusivo del celular.

El desarrollo va a ser realizado en React [39] y acompañado con un backend en Kotlin [4] para brindar seguridad y robustez a la solución. La misma aprovecha los desarrollos antes mostrados (botones de pago, intenciones de pago, etc) permitiendo un despliegue más ágil.



## Capítulo 6 - Glosario

**API:** Es un conjunto de protocolos y definiciones que se utilizan para integrar los distintos componentes de una aplicación, brindando una interfaz de uso y abstrayendo el comportamiento interno de la aplicación.

**App Billetera:** Se trata de la aplicación utilizada para realizar compras por QR por los usuarios. En la misma se agregan las tarjetas que el usuario usará para hacer los pagos. Brinda opciones extras como pagar servicios.

**App Comercios:** Se trata de la aplicación utilizada por los vendedores. Posibilita realizar cobros por lectores de tarjetas (mPos), códigos QR y generar botones de pago. Permite también la opción de cambiar las preferencias de venta, por ejemplo cantidad de cuotas, si son con o sin interés, etc.

**Arquitectura de microservicios:** La arquitectura de microservicios es un método de desarrollo de aplicaciones software, que funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa. En ella, cada microservicio es un código que puede estar en un lenguaje de programación diferente, y que desempeña una función específica. Los microservicios se comunican entre sí a través de APIs, y cuentan con sistemas de almacenamiento propios, lo que evita la sobrecarga y caída de la aplicación.

**Elasticsearch:** es una base de datos distribuida. Distribuye toda la información en todos los nodos, por tanto es tolerante a fallos y tiene alta disponibilidad. Cuando se realiza una consulta o búsqueda y esa información se encuentra distribuida, será cada nodo el que procese dicha información y devuelva los resultados. Por tanto, podemos obtener mejores rendimientos.

**Endpoint:** Los endpoints son las URLs de un API o un backend que responden a una petición.

**Herencia:** Es el mecanismo por el cual una clase permite heredar las características (atributos y métodos) de otra clase.

La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación.

**Jenkins:** Es un servidor automatizado mediante el cual se puede programar una serie de acciones que ayudan a lograr los procesos de integración continua de forma automática.

**Jira:** Es una herramienta de planificación donde los equipos pueden gestionar los requerimientos y el roadmap de los sprints.

**Kafka:** Es un servicio de mensajería. Kafka tiene tres componentes fundamentales: Los productores, los consumidores y los brokers. Los productores son los encargados de escribir mensajes en Kafka, los consumidores los pueden leer y procesar, y los brokers son los nodos que forman parte del cluster de Kafka y almacenan y distribuyen los datos.

**Kibana:** es la herramienta visual, donde se realizan las búsquedas solicitadas, y consultando a Elasticsearch, retornará los datos solicitados.

**Kotlin:** es un lenguaje de programación expresivo y conciso que reduce errores comunes. Se integra fácilmente a apps existentes y es 100% interoperable con el lenguaje Java.

**Logstash:** Es la parte que toma como inputs los logs de cada servidor, los pre procesa y alimenta la base de datos de Elasticsearch previamente definida.

**Oauth:** Se trata de un protocolo para pasar la autorización de un servicio a otro sin compartir las credenciales de usuario reales, como un nombre de usuario y contraseña. Con esta herramienta, un usuario puede iniciar sesión en una plataforma y luego estar autorizado para realizar acciones y ver datos en otra plataforma.

**Patrones de diseño:** Son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos. Soluciones a problemas específicos y comunes del diseño orientado a objetos.

**Patrón de diseño factory:** Describe un diseño que sirve para crear objetos sin tener que especificar su clase exacta. Esto quiere decir que el objeto creado puede intercambiarse con flexibilidad y facilidad.

**Portal de Vendedores:** Es la aplicación Web utilizada para administrar las cuentas vendedoras, configurando sus datos, preferencias de ventas, credenciales de integraciones por SDK, visualizar los movimientos, y poder generar Botones de pago y QRs para cobrarlos.

**Redis:** proviene de las iniciales de Remote Dictionary Server, un tipo de servidor apto como memoria rápida para datos. Los datos no se guardan en disco, sino en la memoria principal. Esto permite que Redis funcione como memoria caché y como unidad de memoria principal, con independencia de si los datos permanecen en la base por mucho o poco tiempo.

**REST:** Representational State Transfer: Es un mecanismo utilizado para comunicarse con las APIs. Lo que caracteriza a un servicio REST es que no manejan estado, y cualquier llamada depende exclusivamente de los datos que se envíen en cada petición.

**SDK:** Kit de desarrollo de software: Es un conjunto de herramientas que se provee a los desarrolladores de otras plataformas para que puedan integrarse al desarrollo que se está realizando.

**SonarQube:** Es una plataforma web que se utiliza para analizar y cuantificar la calidad del código fuente.

**Spring boot:** es una tecnología que nos permite crear aplicaciones autocontenidas, con esto nos podemos olvidar de la arquitectura y enfocarnos únicamente en

desarrollo, delegando a Spring Boot labores como configuración de dependencias, por ejemplo Maven o Gradle, desplegar nuestro servicio o aplicación a un servidor de aplicaciones y enfocarnos únicamente en crear nuestro código.

Para esto Spring Boot utiliza internamente un servidor de aplicaciones embebido, por defecto utiliza Tomcat.

**Swagger:** Es una aplicación para documentación de interfaces API Rest, generando de manera automática la documentación, en base al código de la API.

**Test unitarios:** son un método de pruebas de software que se realizan escribiendo fragmentos de código que probará unidades de código fuente. El objetivo es asegurar que cada unidad funciona como debería de forma independiente.

**UML:** El Lenguaje Unificado de Modelado (UML) representa lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento.

## Capítulo 7 - Referencias bibliográficas.

1. Todo Pago: Pagar y cobrar (<https://portal.todopago.com.ar>) Marzo 2022
2. Wikipedia - Billetera digital ([https://es.wikipedia.org/wiki/Billetera\\_digital](https://es.wikipedia.org/wiki/Billetera_digital)) Marzo 2022
3. Microservices a definition of this new architectural term  
(<https://martinfowler.com/articles/microservices.html>), Abril 2022
4. Spring Boot in Action 1st Edición, Craig Walls, Andrew Glover, 2016, Manning Publications  
  
Microservices with Spring Boot - Building Microservices Application Using Spring Boot (<https://www.javadevjournal.com/spring-boot/microservices-with-spring-boot/>), Marzo 2022
5. Redis: una base de datos diferente | ¿Cómo funciona el DBMS? - IONOS.  
(<https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/que-es-redis/>), Abril 2022
6. What is Apache Kafka? (<https://www.redhat.com/es/topics/integration/what-is-apache-kafka>), Marzo 2022.
7. ¿Qué es Jenkins? Herramienta de Integración Continua.  
(<https://ciberninjas.com/jenkins/>), Abril 2022
8. Everything You Need to Know about Grafana  
(<https://www.skedler.com/blog/everything-you-need-to-know-about-grafana/>), Marzo 2022  
  
Learn Grafana 7.0, Eric Salituro, 2020, Packt Publishing
9. Scrum Sprints: Everything You Need to Know | Atlassian  
(<https://www.atlassian.com/agile/scrum/sprints>), Abril 2022
10. Scrum, qué es, cómo funciona y por qué es excelente  
(<https://www.atlassian.com/es/agile/scrum>), Abril 2022 <sup>10</sup>

## 11. ¿Qué es SOA o Arquitectura Orientada a Servicios?

(<https://www.ecityclic.com/es/noticias/que-es-soa-o-arquitectura-orientada-a-servicios>), Abril 2022 <sup>11</sup>

## 12. Introducción a JSON (<https://www.json.org/json-es.html>), Abril 2022 <sup>12</sup>

## 13. Introducción a Amazon S3 - Amazon Simple Storage Service

([https://docs.aws.amazon.com/es\\_es/AmazonS3/latest/userguide/GetStartedWithS3.html](https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/GetStartedWithS3.html)), Abril 2022 <sup>13</sup>

## 14. What is Bearer token and How it works? - DevOpsSchool.com

(<https://www.devopsschool.com/blog/what-is-bearer-token-and-how-it-works/>), Abril 2022

## 15. What is Git? A Beginner's Guide to Git Version Control

(<https://www.freecodecamp.org/news/what-is-git-learn-git-version-control/>), Abril 2022

## 16. Version Control Systems - GeeksforGeeks

(<https://www.geeksforgeeks.org/version-control-systems/>), Abril 2022

## 17. Microservices logging best practices every team should know | TechTarget

(<https://www.techtarget.com/searcharchitecture/tip/5-essential-tips-for-logging-microservices>), Abril 2022

## 18. ¿Qué es Stack ELK? | KeepCoding Tech School

(<https://keepcoding.io/blog/que-es-stack-elk/>), Marzo 2022

## 19. Microsoft SQL Server query editor | Grafana documentation

(<https://grafana.com/docs/grafana/latest/datasources/mssql/query-editor/>), Marzo 2022

## 20. Buenas prácticas para el Diseño de una API RESTful Pragmática

(<https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-restful-pragmatica/>), Abril 2022

## 21. Arquitectura de microservicios: qué es, ventajas y desventajas

(<https://decidesoluciones.es/arquitectura-de-microservicios/>), Febrero 2022

## 22. ¿Qué es un MVP en desarrollo app o web? - Owius

(<https://owius.com/que-es-un-mvp-en-desarrollo-app-o-web/>), Abril 2022

## 23. La integración continua actual pasa por pipelines | SDOS

(<https://sdos.es/blog/la-integracion-continua-actual-pasa-por-pipelines>), Marzo 2022

24. Improve your Code Quality with SonarLint and SonarQube - DEV Community

(<https://dev.to/leading-edge/improve-your-code-quality-with-sonarqube-3kk6>), Marzo 2022

25. SDK: ¿qué es el software development kit?

(<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/software-development-kit/>), Abril 2022

26. ¿Qué es SSL, TLS y HTTPS? | DigiCert

(<https://www.websecurity.digicert.com/es/es/security-topics/what-is-ssl-tls-https>), Marzo 2022

27. Facebook (<https://www.facebook.com>), Marzo 2022

28. Whatsapp (<https://whatsapp.com>), Marzo 2022

29. .NET (<https://desarrolloweb.com/home/net>), Marzo 2022

30. ¿Qué es Python? (<https://datademia.es/blog/que-es-python>), Marzo 2022

31. Lenguaje PHP (<https://www.um.es/docencia/barzana/DAWEB/2017-18/daweb-tema-14-php-1.html>),  
Marzo 2022

32. Ruby on Rails (<https://kodigo.org/hablemos-de-ruby-on-rails-ruby-on-rails-que-es-y-para-que-sirve/>),  
Marzo 2022

33. ¿Qué es el Código QR? (<https://biblioguias.cepal.org/QR>), Marzo 2022

34. What is Jira Software used for?

(<https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for>), Marzo 2022

35. Oauth 2.0 (<https://oauth.net/2/>), Abril 2022

36. What is Swagger (<https://swagger.io/docs/specification/2-0/what-is-swagger/>), Abril 2022

37. ¿Qué es un diagrama UML y cómo crear uno?

(<https://geekflare.com/es/about-uml-diagram-and-tools/>), Abril 2022

38. Factory Method Design Pattern - Javatpoint

(<https://www.javatpoint.com/factory-method-design-pattern>), Abril 2022

39. React (<https://es.reactjs.org/>), Abril 2022

40. ¿Qué es open source o código abierto? Definición y más | Proofpoint ES

(<https://www.proofpoint.com/es/threat-reference/open-source-software>), Mayo 2022

41. The Client ID and Secret - OAuth 2.0 Simplified

(<https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/>), Abril 2022

42. What is an API Endpoint? | SmartBear Software Resources

(<https://smartbear.com/learn/performance-monitoring/api-endpoints/>), Abril 2022

43. ¿Qué es Microsoft SQL Server y para qué sirve?

(<https://intelequia.com/blog/post/2948/qu%C3%A9-es-microsoft-sql-server-y-para-qué-sirve>), Abril 2022

44. Pruebas de rendimiento con JMeter. Ejemplos básicos | SDOS

(<https://sdos.es/blog/pruebas-de-rendimiento-con-jmeter-ejemplos-basicos>), Abril 2022

45. Mercado Pago (<https://www.mercadopago.com.ar/>), Abril 2022

46. Criptomonedas, ¿Que son?

(<https://www.xataka.com/basics/criptomonedas-que-como-funcionan-que-otras-existen-bitcoin>), Enero 2023

47. Marketplace en Argentina: qué es y cómo funciona

(<https://www.tiendanube.com/blog/marketplace-argentina/>), Marzo 2023