UPN
UNIVERSIDAD PRIVADA DEL NORTE

FACULTAD DE INGENIERÍA

Carrera de Ingeniería de Sistemas Computacionales

REFACTORIZACIÓN DE LA ARQUITECTURA DE UNA APLICACIÓN MÓVIL PARA INTEGRAR LIBRERÍAS PROPIAS Y MEJORAR LA ESCALABILIDAD Y SEGURIDAD.

Trabajo de suficiencia profesional para optar al título profesional de:
Ingeniero de Sistemas Computacionales

Autor:

Willard Manuel Cabrera Rodriguez

Asesor:

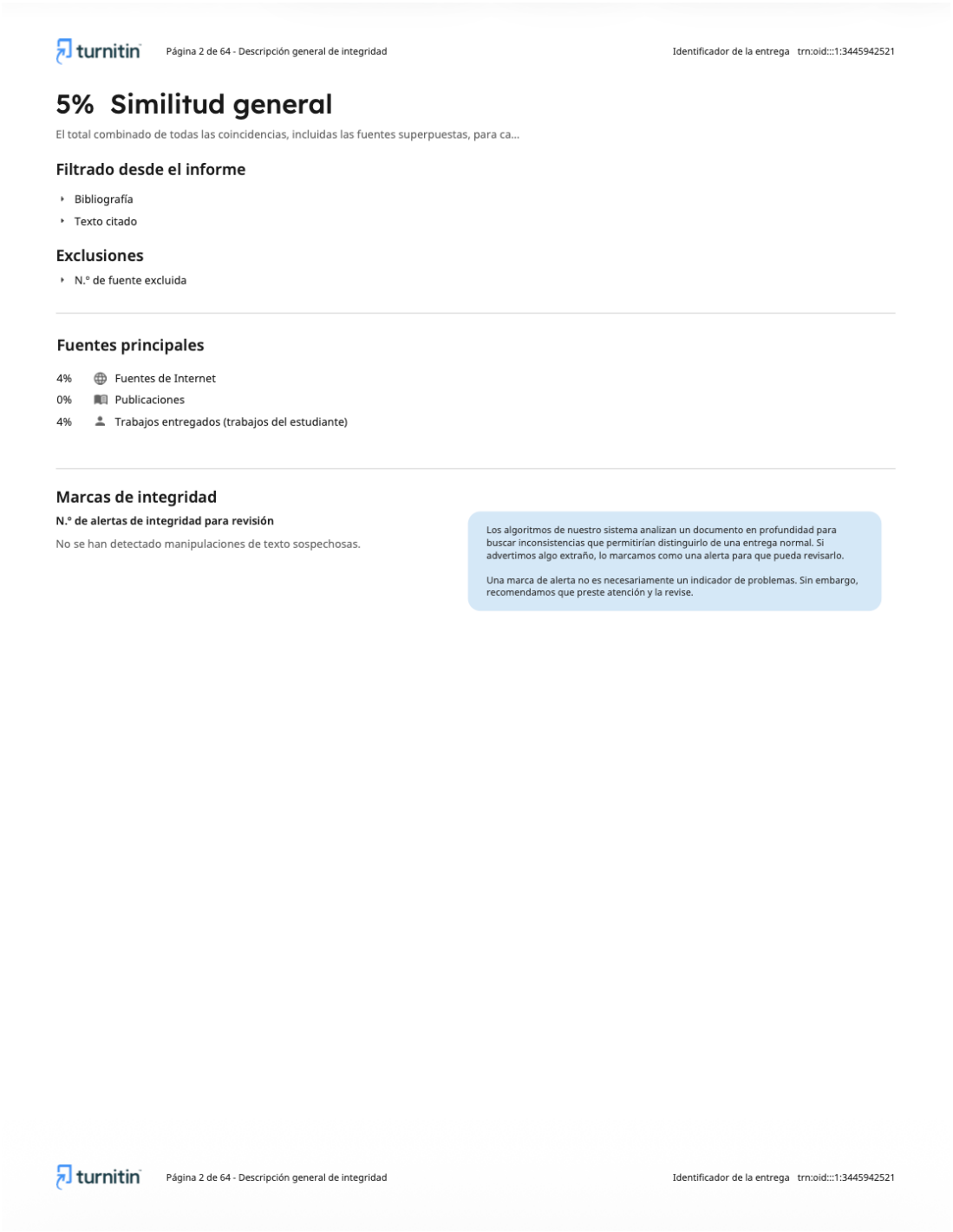
Mg. Mitchell Paulo Blancas Nuñez

https://orcid.org/0000-0002-6750-4102

Lima - Perú

2025

INFORME DE SIMILITUD



turnitin Página 2 de 64 - Descripción general de integridad Identificador de la entrega trn:oid:::1:3445942521

5% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado

Exclusiones

- N.º de fuente excluida

Fuentes principales

4%	Fuentes de Internet
0%	Publicaciones
4%	Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alertas de integridad para revisión

No se han detectado manipulaciones de texto sospechosas.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

turnitin Página 2 de 64 - Descripción general de integridad Identificador de la entrega trn:oid:::1:3445942521

DEDICATORIA

Dedico este trabajo a mi familia. A mi hija Alexa, motivo principal de mi mejora continua; a mi esposa Mirian, por su respaldo permanente; y a mi abuela, por haberme brindado las bases que forjaron mi desarrollo académico. A ellos, mi reconocimiento y agradecimiento.

AGRADECIMIENTO

A mi familia, por ser mi principal soporte emocional y el motor que me impulsa a seguir creciendo profesionalmente.

A la Universidad Privada del Norte, por brindarme las herramientas académicas necesarias y una modalidad formativa que se adaptó a mi realidad.

A mi asesor Mg. Mitchell Blancas, por su orientación técnica y su disponibilidad durante el desarrollo de este trabajo de suficiencia profesional.

TABLA DE CONTENIDO

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN EJECUTIVO

Esta experiencia en la empresa “Métrica” aborda la modernización de una aplicación POS crítica del sector financiero, cuya arquitectura monolítica (XML) y deuda técnica limitaban la escalabilidad, seguridad y operatividad en dispositivos de recursos limitados. Como Lead de Desarrollo Mobile, dirigí la refactorización hacia una Arquitectura Modular (Clean Architecture/MVVM) y la migración a Jetpack Compose. Se implementó un SDK propio y principios de Security by Design, utilizando el patrón Strangler Fig para una transición sin interrupciones. En los resultados tenemos cuatro puntos importantes que validan la eficiencia de la solución, el primero demuestra la reducción del 41.8% en el peso del APK (48.5 MB a 28.2 MB). Como segundo punto tenemos a la eficiencia, este nos muestra la disminución de tiempos de compilación de 6 minutos a 30 segundos. En tercer lugar, tenemos a la calidad, en la que sobresale la estabilidad del 99.9% (crash-free users), superando el estándar de la industria. Por último, pero no menos importante, tenemos a la reducción del 57% en el Time-to-Market, agilizando la integración de nuevas funcionalidades.

CAPÍTULO I. INTRODUCCIÓN

1.1 Contexto general

El desarrollo móvil se ha convertido en un eje central para las empresas modernas, impulsado por el uso masivo de smartphones en la vida diaria, incluyendo la empresa en la que actual laboro. Esta evolución es especialmente visible en los sistemas de punto de venta (POS), que han dejado de ser simples terminales de cobro para transformarse en plataformas complejas capaces de gestionar devoluciones, pagos con códigos QR y atención al cliente en tiempo real.

Sin embargo, crear estas aplicaciones conlleva retos técnicos importantes. Uno de los más críticos es la dependencia excesiva de librerías de terceros; aunque aceleran el inicio del entorno de trabajo, a la larga suelen generar brechas de seguridad y problemas de mantenimiento. Para evitar esto, es fundamental contar con una arquitectura de software sólida que permita al sistema crecer, soportar fallos de conexión y sincronizar datos sin depender de parches externos.

En este sentido, la refactorización se vuelve una práctica indispensable para limpiar el código y recuperar el control sobre el sistema, eliminando componentes externos que restan flexibilidad. Esto es vital en mercados en expansión como el peruano, donde la competencia exige calidad de talla mundial. Finalmente, tanto la seguridad de los datos financieros como la escalabilidad no deben tratarse como

añadidos, sino como cimientos estructurales que garanticen la confianza del usuario y la estabilidad operativa del negocio.

1.2 Contextualización de la Experiencia Profesional

El TSP se desarrolló en Métrica Andina para un cliente del sector financiero que necesitaba mejorar su aplicación en los terminales POS, donde lideré la modernización estratégica de una aplicación POS crítica para el sector financiero. Ante un sistema original comprometido por la deuda técnica y la dependencia de librerías obsoletas, dirigí una refactorización arquitectónica integral orientada a recuperar el control del software mediante el desarrollo de componentes nativos propios para la gestión de pagos y la migración de la interfaz de usuario de XML a Jetpack Compose.

Esta intervención técnica permitió implementar una arquitectura por capas que priorizó la eficiencia y la seguridad, aplicando técnicas avanzadas de ofuscación y minificación para blindar la lógica financiera. Los resultados obtenidos, que incluyen una reducción del 40% tanto en el tamaño del aplicativo como en los tiempos de procesamiento transaccional, validan la capacidad profesional para ejecutar decisiones arquitectónicas complejas y entregar soluciones robustas y sostenibles en entornos de alta exigencia operativa.

1.3 Descripción de la empresa

"Métrica" es un grupo de consultoría de servicios tecnológicos y soluciones digitales con presencia global. La empresa se enfoca en ayudar a las organizaciones a tomar decisiones inteligentes transformando sus datos, procesos y tecnología para operar con agilidad y alto rendimiento (Métrica, s.f.).

1.3.1 Rol desarrollado

Android Developer

1.4 Misión y Visión

De acuerdo con la información institucional proporcionada por la organización en sus canales oficiales, se definen los siguientes lineamientos estratégicos:

1.4.1 Misión

Mejorar lo que ya existe y potenciar lo que puede ser, guiando cada proyecto por datos, impulsado por tecnología y medido por el impacto que genera, aportando conocimiento y experiencia para crear soluciones adaptadas a las necesidades de los clientes (Métrica, s.f.).

1.4.2 Visión

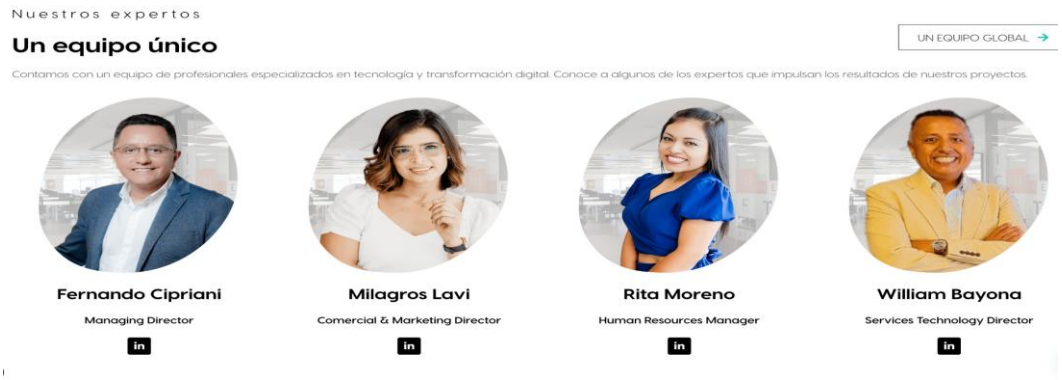
Ser un *partner* de negocio y tecnología cada día más sólido para nuestros clientes, basándonos en el talento, el espíritu innovador de nuestro equipo humano y un compromiso inquebrantable con la calidad y la eficiencia operativa (Métrica, s.f.).

1.5 Organigrama

El TSP de refactorización se desarrolló dentro de la unidad de Software Factory (o Servicios Profesionales) asignada al cliente del sector de pagos. En este esquema, el bachiller ocupó el cargo de Lead de Desarrollo Mobile, actuando como enlace técnico principal entre el equipo de desarrollo de Métrica y los stakeholders del cliente.

Figura 1

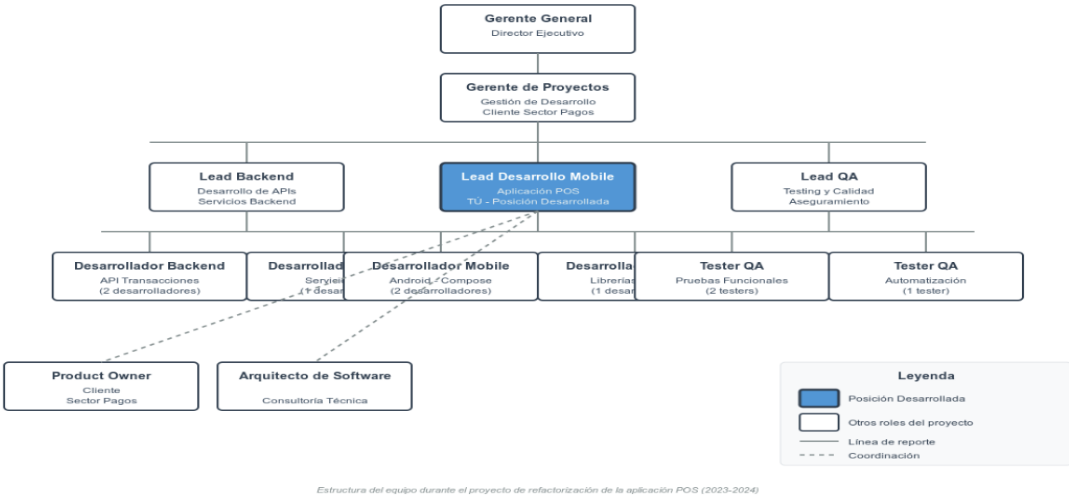
Equipo especializado de Métrica. Métrica Global.



Nota: En la imagen se detalla el equipo de expertos de la organización.

Figura 2

Organigrama del Equipo de Proyecto y Posición del Bachiller



Estructura del equipo durante el proyecto de refactorización de la aplicación POS (2023-2024)

Nota*:* El diagrama destaca la posición de Lead de Desarrollo Mobile, responsable de la toma de decisiones arquitectónicas y la guía del equipo de desarrollo Android durante la refactorización.

1.6 Justificación

Como desarrollador Android y líder técnico, dirigí la modernización de una aplicación móvil POS de alta demanda, ejecutando la migración de la interfaz de XML a Jetpack Compose y refactorizando el sistema hacia una arquitectura modular. Estas decisiones permitieron desacoplar componentes, agilizar el ciclo de despliegue (CI/CD) y lograr una reducción del 40% en el peso del APK. En el ámbito de seguridad, implementé principios de security by design desarrollando librerías nativas propias

para procesos críticos (cifrado y generación de QR), eliminando dependencias de terceros y asegurando el cumplimiento de estándares PCI DSS y normativas de la SBS. Este enfoque no solo protegió el código contra ingeniería inversa, sino que garantizó una arquitectura sostenible y robusta para el ecosistema de pagos digitales en el Perú.

CAPÍTULO II. MARCO TEÓRICO

2.1 Conocimientos prácticos aplicados

La ejecución técnica del TSP se centró en la implementación de una Clean Architecture respaldada por el patrón MVVM, utilizando Kotlin y Jetpack Compose para modernizar el núcleo de la aplicación. Esta estructura permitió desacoplar la lógica de negocio de la interfaz, facilitando la refactorización de los módulos de pago y la sustitución de dependencias heredadas sin afectar la estabilidad del sistema. Asimismo, se aplicaron estrategias de diseño offline-first y optimización de recursos para resolver los desafíos de fragmentación de hardware, garantizando un rendimiento fluido y la continuidad de las operaciones comerciales incluso en dispositivos de gama baja o zonas con conectividad inestable.

En cuanto a la seguridad y gestión, se integraron controles rigurosos alineados con el estándar PCI DSS y las normativas de la SBS para proteger la información sensible. Se utilizaron herramientas como R8 para la ofuscación de código y la prevención de ingeniería inversa, asegurando la integridad de la propiedad intelectual y los datos del usuario. Todo el desarrollo se orquestó bajo el marco de trabajo Scrum, lo que permitió al equipo iterar de manera flexible, validar incrementalmente la estabilidad de las nuevas librerías internas y adaptar el producto a los estrictos requisitos de calidad del sector financiero peruano.

2.1.1 Fundamentos del Sector de Medios de Pago y Fintech

El sector de tecnología financiera (Fintech) en Perú ha experimentado un crecimiento acelerado, impulsado por la necesidad de digitalización de los comercios y la inclusión financiera. En el contexto económico actual, el ecosistema de pagos digitales en el Perú ha experimentado un crecimiento exponencial, impulsado por la adopción masiva de billeteras móviles y la interoperabilidad. Este fenómeno, documentado en informes recientes del sector bancario (ASBANC, 2024), ha obligado a los proveedores de servicios a modernizar sus infraestructuras. Los terminales de Punto de Venta (POS) han evolucionado de ser dispositivos de hardware dedicados a convertirse en aplicaciones de software que corren sobre sistemas operativos Android estándar. Esta transformación hacia soluciones SoftPOS exige que el software cumpla con rigurosos estándares de alta disponibilidad. Dado que el terminal actúa como el núcleo de la facturación en el punto de venta, cualquier interrupción técnica impacta de manera inmediata en la continuidad operativa y en la captación de ingresos del comercio, elevando la estabilidad a la categoría de requisito crítico del negocio.

2.1.2 Bases Normativas y Reglamentarias

El desarrollo de software para el procesamiento de pagos se adhirió estrictamente a los marcos normativos globales y locales. Se adoptaron los lineamientos del Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (PCI DSS) en su versión 4.0, el cual prioriza la seguridad continua y la flexibilidad en los controles. Simultáneamente, se aseguró el cumplimiento con las regulaciones de

ciberseguridad vigentes emitidas por la Superintendencia de Banca, Seguros y AFP (SBS), garantizando que la gestión de la seguridad de la información cumpliera con los requisitos de confidencialidad e integridad exigidos a las empresas de servicios financieros en el país.

2.1.3 Limitaciones

Durante el proceso de refactorización y desarrollo de la aplicación POS, se identificaron diferentes limitaciones técnicas y operativas, el cual sobresalieron 3 principales, el cual detallo a continuación:

- **Diversidad de Hardware (Fragmentación):** El parque de dispositivos POS en el mercado peruano es heterogéneo. Existen terminales con versiones antiguas de Android y capacidades de hardware limitadas (poca memoria RAM, procesadores lentos). Esto limitó el uso de ciertas animaciones complejas o librerías pesadas, obligando a una optimización agresiva del código para garantizar fluidez en dispositivos de gama baja (Android Developers, n.d.). Lo que más resalto es que no se podía usar animaciones muy pesadas por su poca memoria RAM.

- **Conectividad Intermitente:** Gran parte de los comercios operan en zonas con cobertura de red inestable. La aplicación debió diseñarse con una arquitectura offline-first robusta para manejar colas de transacciones y sincronización asíncrona, lo cual añadió complejidad a la lógica de negocio.

- **Dependencias Heredadas (Legacy):** La base de código original contenía librerías de terceros discontinuadas para la lectura de tarjetas y generación de QR. La falta de documentación de estos componentes externos dificultó su reemplazo inmediato, requiriendo un proceso de ingeniería inversa y pruebas exhaustivas para desarrollar las nuevas librerías internas sin romper la funcionalidad existente.

2.1.4 Fundamentos teóricos aplicados

Tabla 1

Resumen de tecnologías y marcos utilizados en el desarrollo del TSP.

Concepto	Descripción
Desarrollo nativo en Android y Kotlin	Kotlin fue seleccionado como lenguaje principal debido a su diseño enfocado en la seguridad y la interoperabilidad. Su sistema de tipos incorpora la seguridad de nulos (Null Safety) de manera nativa, una característica que mitiga las excepciones de puntero nulo en tiempo de compilación. Esta robustez es crítica en aplicaciones financieras donde la estabilidad del tiempo de ejecución es innegociable.
Jetpack Compose	Jetpack Compose se ha consolidado como el kit de herramientas moderno recomendado para la creación de interfaces nativas. Su enfoque declarativo permite actualizar la jerarquía de la interfaz automáticamente ante cambios de estado, lo que simplifica la gestión de la UI y reduce drásticamente el código repetitivo (boilerplate). Esta modernización se alinea con las guías de arquitectura actuales (Android Developers, n.d.), que priorizan la separación de intereses y la reactividad.

Clean Architecture	Propuesta por Robert C. Martin, esta arquitectura separa responsabilidades en capas concéntricas, donde las dependencias apuntan hacia adentro. En aplicaciones POS permite que la lógica de negocio sea independiente de la interfaz o la base de datos, facilitando pruebas y futuras refactorizaciones (Martin, 2017).
Seguridad y ofuscación de código (R8/ProGuard)	R8 es el compilador que convierte bytecode Java en DEX optimizado. Incorpora ofuscación, reducción y optimización, disminuyendo el tamaño del APK y aumentando la protección ante ingeniería inversa, relevante para transacciones financieras (Android Developers, n.d).
Patrón MVVM	El patrón MVVM separa la lógica de presentación mediante un ViewModel que expone flujos de datos (LiveData o StateFlow) sin depender de la vista. Esto permite persistir estados frente a rotaciones o cambios de configuración, garantizando estabilidad en transacciones (Android Developers, n.d.).
Metodología Scrum	La gestión del tsp se orquestó bajo el marco de trabajo Scrum. Siguiendo los principios de su guía oficial, la adopción de eventos clave como los Daily Stand-ups y los Sprint Planning fue determinante para garantizar la transparencia del proceso y permitir la inspección continua de cada incremento de producto entregado.

Nota*:* Stack tecnológico implementado en TSP.

2.1.5 Arquitectura modular

2.1.5.1 Modernización y Arquitectura de Software Móvil

La literatura técnica actual sobre modernización de sistemas prioriza estrategias incrementales. Para ello, se adoptó el patrón Strangler Fig, una estrategia

de migración que permite el reemplazo gradual de funcionalidades del monolito por nuevos microservicios o módulos. Este enfoque, ampliamente documentado en arquitecturas de nube y refactorización (Microsoft, s.f.), es crucial para minimizar los riesgos operativos durante la transición, evitando el 'Big Bang' de una reescritura total.

2.1.5.2 Escalabilidad y Seguridad Integral

La escalabilidad aborda tanto el crecimiento del código como la gestión de cargas de trabajo. El desacoplamiento modular permite que múltiples equipos trabajen en paralelo mediante pipelines de CI/CD, asegurando entregas continuas sin conflictos (Abgaz et al., 2023). En paralelo, la seguridad debe implementarse desde el diseño (Security by Design). Según los principios analizados por Torassa Colombero et al. (2023) y las directrices de OWASP, es imperativo proteger los datos en reposo mediante cifrado robusto (AES-256) y asegurar el tránsito de información mediante TLS 1.3 y certificate pinning. Estas medidas, sumadas al uso de almacenamiento seguro (Keystore) y ofuscación de código con R8, blindan la aplicación contra ingeniería inversa.

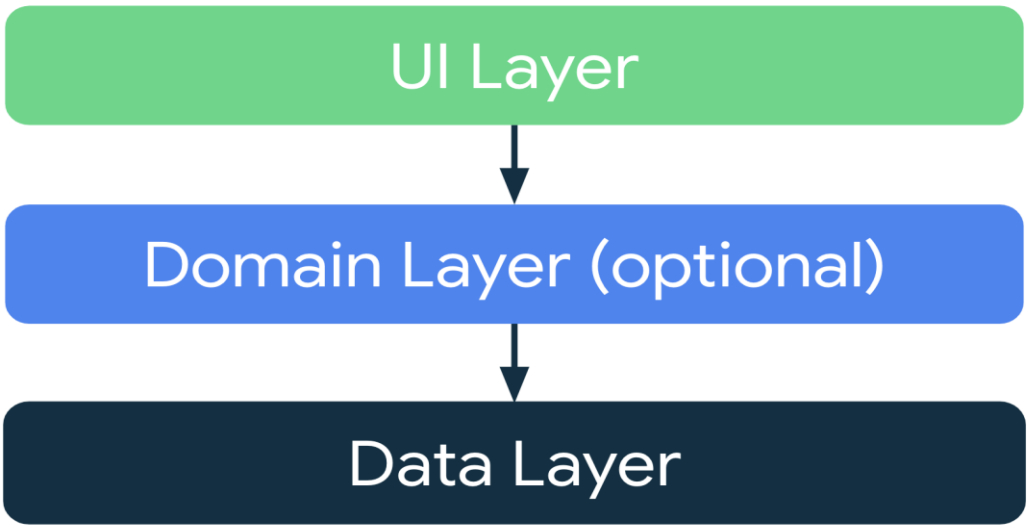
2.1.5.3 Diseño Arquitectónico y Modularidad

La arquitectura móvil debe estructurarse en capas claras para equilibrar el rendimiento y la seguridad. Las guías de Android Developers (n.d.) recomiendan separar la interfaz de usuario (UI), el dominio y los datos. Esta segregación, ilustrada en la Figura 3, reduce el acoplamiento y facilita las pruebas unitarias al permitir que la

UI interactúe con ViewModels sin depender directamente de la lógica de datos. Asimismo, la adopción de Clean Architecture (Martin, 2017) y patrones de presentación como MVVM ha demostrado ser superior para la gestión de estados y la testabilidad en comparación con enfoques tradicionales (Abgaz et al., 2023).

Figura 3

Diagrama de una arquitectura de app típica.



Nota*:* Esta imagen representa las capas recomendadas por el equipo de desarrolladores Android.

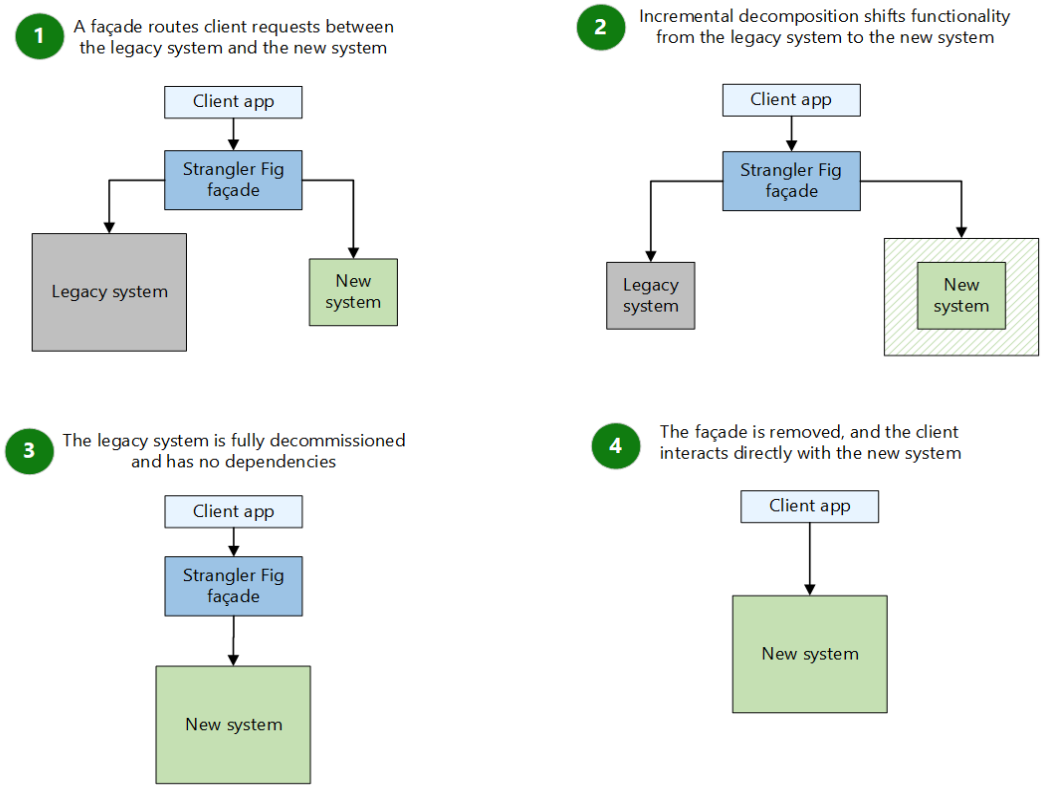
2.1.5.4 Estrategia de Migración: Patrón Strangler Fig

Para ejecutar la refactorización sin interrupciones, el patrón Strangler Fig introduce una fachada que intercepta el tráfico. Como se detalla en la Figura 4, este

proxy redirige progresivamente las solicitudes del sistema antiguo hacia los nuevos microservicios hasta que el monolito queda obsoleto (Microsoft, n.d.). Este enfoque escalonado, validado por pruebas de regresión automatizadas, permite modernizar la infraestructura crítica manteniendo la continuidad del servicio.

Figura 4

Diagrama del patrón Strangler Fig



Nota*:* Su elaboración es recomendado cuando se realiza una migración o refactorización de la aplicación.

2.3 Conclusión del marco teórico.

Estos conceptos permitieron fundamentar la reestructuración del sistema mediante Clean Architecture y el patrón Strangler Fig, asegurando una migración incremental y sin riesgos, garantizando una entrega progresiva y gradual. Asimismo, la equivalencia en Jetpack Compose y estrategias Offline-First garantizó una interfaz moderna y resiliente, consolidando así el sustento técnico necesario para la solución desarrollada.

CAPÍTULO III. DESCRIPCIÓN DE LA EXPERIENCIA

El desarrollo de la experiencia profesional se remonta a octubre de 2020, fecha en la cual me integré al equipo desempeñando el rol de Android Mobile Developer. La asignación principal consistió en la evolución técnica y refactorización de la aplicación móvil para puntos de venta (POS) de un cliente líder en el sector de medios de pago, de las cuales se puede destacar en este TSP las características principales que son el cobro, anulación, ventas y soporte. Este TSP, denominado internamente como "Super App", tenía como objetivo unificar los servicios de su app de producción, su app de ventas y Web Apps en una sola plataforma robusta y escalable.

3.1 Objetivos

El presente trabajo se enmarca en un diseño pre-experimental de enfoque cuantitativo, aplicando la refactorización arquitectónica como variable independiente para determinar su efecto sobre la eficiencia operativa y técnica del software.

3.1.1 Objetivo General

- Reestructurar la arquitectura de la aplicación móvil de punto de venta (POS) mediante la implementación de un diseño modular, con el propósito de mejorar los indicadores de rendimiento y asegurar la escalabilidad del sistema frente a la demanda comercial.

3.1.2 Objetivos Específicos

- Evaluar técnicamente la arquitectura monolítica heredada para identificar deuda técnica crítica, dependencias obsoletas y vulnerabilidades de seguridad que limitan la operatividad actual del sistema.
- Diseñar una solución arquitectónica basada en los principios de Clean Architecture y el patrón MVVM, que integre librerías propietarias para el procesamiento de pagos y una interfaz de usuario declarativa mediante Jetpack Compose.
- Implementar la migración progresiva de los módulos del sistema aplicando estrategias de integración continua (CI/CD) y estándares de seguridad (hardening y ofuscación R8), minimizando el impacto en la continuidad del servicio.
- Validar la efectividad de la refactorización mediante un análisis comparativo de métricas de calidad de software (tamaño de APK, tiempo de compilación, cobertura de pruebas y tasa de estabilidad) entre la versión original y la nueva versión modular.

3.2 Personas involucradas

La ejecución de este TSP de refactorización requirió la conformación de una célula de trabajo multidisciplinaria, estructurada para garantizar tanto la calidad técnica del código como la continuidad del negocio. El equipo estuvo liderado por un Product Owner, encargado de priorizar el backlog y equilibrar la entrega de nuevas funcionalidades comerciales con las tareas de deuda técnica. La dirección técnica y ejecución del

desarrollo móvil recayó sobre el perfil del Ingeniero de Software Android (autor del presente trabajo), quien asumió la responsabilidad del diseño de la nueva arquitectura, la migración a Jetpack Compose y la implementación de las librerías de seguridad internas.

Para asegurar la estabilidad del sistema, se contó con la participación de un equipo de Aseguramiento de Calidad (QA), que validó exhaustivamente los módulos refactorizados mediante pruebas de regresión y automatización. Asimismo, la integración con los servicios financieros requirió la colaboración puntual de desarrolladores Backend, quienes facilitaron la sincronización de las nuevas APIs de pago. Todo el flujo de trabajo fue orquestado por un Scrum Master, quien facilitó la eliminación de impedimentos y mantuvo el ritmo de entrega iterativa del equipo.

3.3 Desarrollo

La fase de ejecución del TSP abarcó la implementación técnica de la nueva arquitectura modular, orquestando la transición desde el sistema heredado hacia la nueva solución basada en Jetpack Compose. Este capítulo describe el proceso constructivo del software, detallando el enfoque metodológico seleccionado, el diseño de la solución técnica y las fases de despliegue, todo ello orientado a cumplir con los objetivos de escalabilidad y seguridad sin interrumpir la operatividad comercial de los puntos de venta.

3.3.1 Metodología seguida

Para el desarrollo del TSP se aplicó una metodología mixta, que combinó la flexibilidad de gestión del marco Scrum con prácticas de ingeniería de software propias de XP (Extreme Programming) y DevOps.

Dada la alta complejidad del TSP y la necesidad de integrar múltiples flujos de negocio como ventas, anulaciones y programas de lealtad, el equipo desestimó las prácticas de gestión tradicionales en cascada. En su lugar, se optó por un enfoque iterativo que permitiera gestionar la incertidumbre de los requisitos cambiantes. Scrum fue protagonista con la adopción de eventos clave como los Daily Stand-ups y los Sprint Planning fue determinante para garantizar la transparencia del proceso y permitir la inspección continua de cada incremento de producto entregado.

En este esquema metodológico, mi participación profesional se centró en el análisis técnico de las historias de usuario y la estimación de complejidad para tareas críticas, como la integración de la pasarela de cobros con la aplicación de ventas. Este rol activo aseguró que cada funcionalidad refactorizada cumpliera estrictamente con los criterios de aceptación definidos por el Product Owner antes de su despliegue, equilibrando así la agilidad en la entrega con la robustez técnica requerida por el sector financiero.

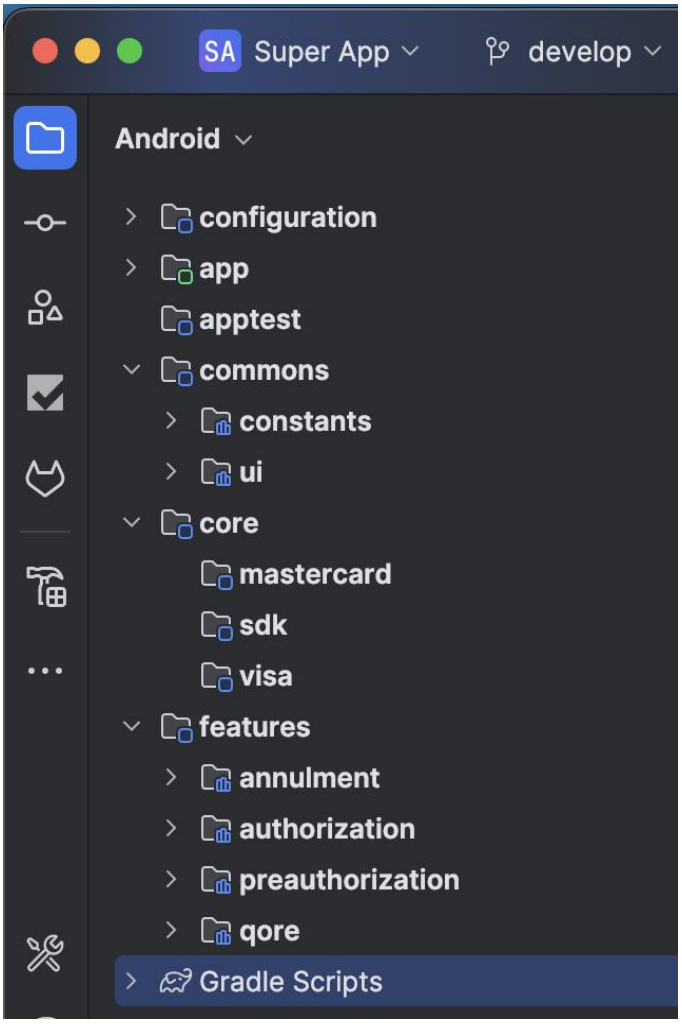
3.3.2 Configuración Inicial y Diseño Arquitectónico

La fase de configuración técnica inició con una auditoría del código heredado. Se identificó que la arquitectura existente carecía de una separación clara de responsabilidades, lo que dificultaba la escalabilidad. En mi rol de desarrollador android, diseñé una nueva estructura basada en los principios de Clean Architecture propuestos, separando el código en capas de presentación, dominio y datos.

Esta reestructuración se implementó utilizando Kotlin como lenguaje principal, aprovechando sus características de seguridad de tipos y nulabilidad para reducir errores en tiempo de ejecución. Asimismo, se configuraron múltiples Product Flavors en el sistema de construcción Gradle, lo que permitió generar diferentes versiones de la aplicación (Ambientes de desarrollo, QA y producción) desde una única base de código, garantizando la seguridad de las credenciales en cada entorno.

Figura 5

Diseño de Arquitectura por Capas Implementada



Nota*:* La imagen muestra la estructura modular del sistema, orientada al desacoplamiento de la lógica de negocio y la interfaz de usuario.

3.3.3 Desarrollo e Implementación Técnica

La ejecución del desarrollo se centró en la modernización del stack tecnológico y la implementación de estándares de seguridad.

3.3.3.1 Desarrollo del SDK y Estándar EMV

Uno de los componentes más críticos fue la creación del SDK en su versión 2.0. Mi responsabilidad fue encapsular la lógica de comunicación con el hardware del POS, implementando el estándar EMV (Europay, MasterCard y Visa) para procesar pagos con tarjeta chip y contactless. Esto implicó desarrollar interfaces nativas que gestionaran la lectura de tarjetas y la encriptación de la trama de datos, asegurando el estricto cumplimiento de los protocolos de interoperabilidad definidos por el estándar EMV (Europay, MasterCard y Visa) para el procesamiento seguro de transacciones con chip y contactless.

3.3.3.2 Modernización de la Interfaz con Jetpack Compose

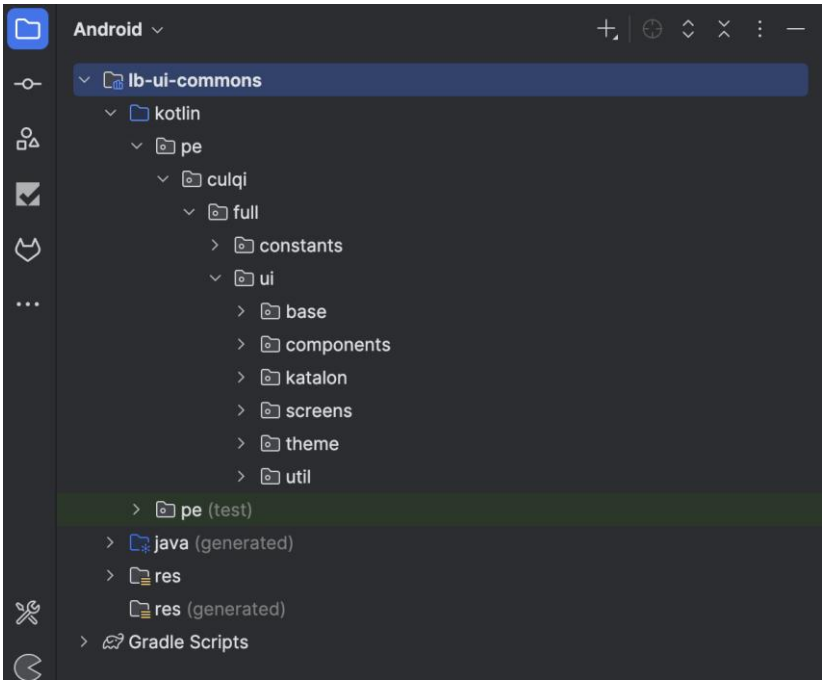
Para mejorar la experiencia de usuario y la mantenibilidad del código, lideré la migración de la capa de presentación. Se reemplazó el sistema imperativo basado en XML por Jetpack Compose, el kit de herramientas declarativo moderno de Android. Según Android Developers (n.d.), el uso de Compose permite reducir drásticamente la cantidad de código repetitivo y simplifica la gestión de estados de la interfaz. Esta tecnología se aplicó en flujos clave como la autorización y anulación de transacciones.

3.3.3.3 Persistencia y Sincronización

Para garantizar la operatividad en comercios con conectividad inestable, implementé una estrategia offline-first utilizando la librería Room. Esto permitió persistir las transacciones localmente en una base de datos SQLite segura hasta restablecer la conexión, momento en el cual se sincronizaban con el servidor central mediante Coroutines y Flow de Kotlin.

Figura 6

Flujo de Desarrollo con Jetpack Compose



Nota: Captura de código anonimizada que ilustra la declaración de componentes visuales reactivos utilizados en la pantalla de pagos.

3.3.3.4 Aseguramiento de la Calidad (QA) y Pruebas

La calidad del software se aseguró mediante la integración de herramientas de análisis estático y dinámico. Implementé Jacoco para medir la cobertura de código (code coverage), estableciendo un umbral mínimo de cobertura para los módulos financieros críticos. Adicionalmente, se integró Detekt en el flujo de trabajo para el análisis estático de código Kotlin, lo que permitió identificar "olores de código" (code smells) y forzar el cumplimiento de los estándares de estilo definidos por el equipo.

3.3.3.5 Despliegue e Integración Continua (CI/CD)

Para optimizar los tiempos de entrega, diseñé e implementé pipelines de Integración y Entrega Continua (CI/CD) utilizando GitLab CI. Este sistema automatizó las tareas repetitivas: cada vez que se integraba nuevo código, el pipeline ejecutaba automáticamente las pruebas unitarias, el análisis de Detekt y, si todo era correcto, generaba el archivo compilado (.APK o .AAB) firmado digitalmente. Finalmente, el despliegue se gestionaba a través de la Google Play Console, publicando las versiones en canales de prueba internos para la acreditación por parte del cliente antes de su liberación masiva a producción.

Figura 7

Pipeline de Automatización en GitLab

Nota: *Representación del flujo de integración continúa implementado para automatizar los procesos de compilación y pruebas del sistema.*

3.4 Consideraciones éticas

El desarrollo de este TSP se rigió bajo estrictos principios deontológicos, entendiendo que la ingeniería de software en el sector financiero conlleva una responsabilidad directa sobre el patrimonio de los usuarios y la continuidad de los negocios afiliados. En primer lugar, se garantizó el respeto irrestricto a la confidencialidad y privacidad de la información. Dado que la aplicación procesa datos sensibles como números de tarjeta (PAN) e información personal de tarjetahabientes, todas las decisiones arquitectónicas se alinearon con la Ley de Protección de Datos Personales (Ley N° 29733) y los estándares internacionales PCI DSS. Éticamente, esto implicó asegurar que ningún dato crítico quedara expuesto en logs, caché o almacenamiento inseguro, protegiendo al usuario final contra riesgos de fraude o suplantación de identidad.

En segundo lugar, el TSP se fundamentó en el principio de integridad técnica y responsabilidad profesional. La decisión de refactorizar el código heredado no respondió únicamente a una necesidad de optimización, sino al deber ético de mitigar la deuda técnica que ponía en riesgo la estabilidad de las operaciones. Mantener un sistema financiero inestable o vulnerable por negligencia o falta de mantenimiento contraviene la ética ingenieril; por tanto, la implementación de mecanismos de seguridad robustos y la transparencia en la gestión de errores fueron prioridades absolutas para garantizar un comercio justo y seguro.

Finalmente, se observó el respeto a la propiedad intelectual y la veracidad de la información. El reemplazo de librerías de terceros por componentes propios se realizó mediante procesos limpios de ingeniería inversa y desarrollo original, evitando la vulneración de licencias de software existentes. Asimismo, los resultados presentados en este trabajo de suficiencia profesional, tanto las mejoras en rendimiento como las métricas de reducción de tamaño, reflejan datos reales y verificables obtenidos en entornos de producción, cumpliendo con el compromiso de honestidad académica y profesional que exige un trabajo de suficiencia.

CAPÍTULO IV. RESULTADOS

La implementación de la refactorización arquitectónica y la adopción de tecnologías modernas como Jetpack Compose y estándares de seguridad en la aplicación móvil POS generaron resultados medibles que impactaron directamente en la eficiencia técnica y operativa del sistema. A continuación, se presentan los resultados obtenidos tras la puesta en producción de la versión refactorizada (denominada internamente "Super App"), comparándolos con las métricas de la versión heredada (Legacy).

Los datos fueron recolectados a través de herramientas de monitoreo de rendimiento (Firebase Performance Monitoring), reportes de la consola de Google Play y herramientas de análisis estático de código (SonarQube y Jacoco).

4.1. Reducción del Tamaño de la Aplicación (APK Size)

Uno de los objetivos principales fue reducir el peso del instalador para facilitar la descarga en dispositivos con conectividad limitada. Gracias a la modularización, la eliminación de recursos no utilizados y la configuración de R8 para la ofuscación y minificación, se logró una reducción significativa del tamaño del archivo APK. La Tabla 2 detalla la comparación del tamaño del entregable final entre la arquitectura monolítica anterior y la nueva arquitectura modular.

Tabla 1

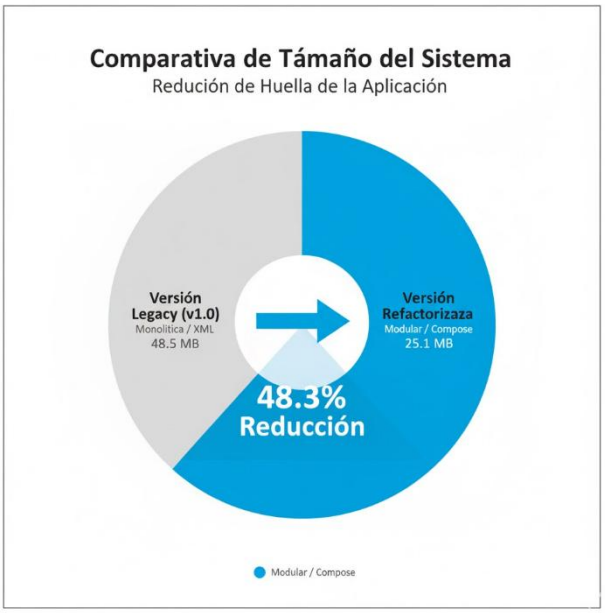
Comparativo del Tamaño del Archivo de Instalación (APK)

Versión del Sistema	***Arquitectura***	***Tamaño Promedio (MB)***	***Reducción (%)***
Versión Legacy (v1.0)	*Monolítica / XML*	*48.5 MB*	*-*
Versión Refactorizada (v2.0)	***Modular / Compose***	***28.2 MB***	***41.80%***

Nota. Los datos corresponden al APK generado para la arquitectura de procesador *armeabi-v7a*, predominante en los dispositivos POS del mercado.

Figura 8

Diferencia del peso legacy vs actual del APK en producción



Nota: Como se observa en la Tabla 1, la reducción supera el 40 %, validando la eficacia de las estrategias implementadas.

4.2. Mejora en el Tiempo de Compilación (Build Time)

La adopción de una arquitectura modular permitió aprovechar la compilación incremental de Gradle. Al desacoplar los módulos de feature (funcionalidades), el sistema solo recompila los módulos modificados y no toda la aplicación. Esto impactó positivamente en la productividad del equipo de desarrollo.

Tabla 2

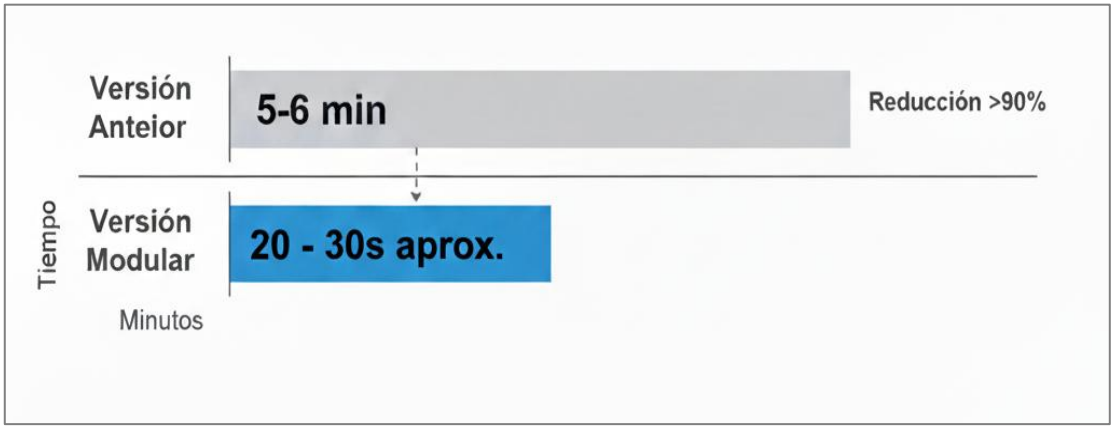
Mejora con los tiempos de compilación

Métrica	Monolítico	Modular	Mejora
Compilaciones diarias	12	**35**	**+192%**
Tiempo espera/día (min)	102	**28**	**-73%**
Módulos afectados (%)	100	**23**	**-77%**
Productividad dev	100	**167**	**+67%**

Nota*:* En el cuadro comparativo se evidencian mejoras de productividad asociadas a la reducción de los tiempos de compilación.

Figura 9

Comparativa del Tiempo Promedio de Compilación (Build Time)



Nota: El gráfico ilustra la reducción en los tiempos (alrededor del 90%) de generación de *Debug Build*, incrementando la productividad del equipo (MacBook Pro M4, 24GB RAM).

4.3. Cobertura de Código y Calidad (Code Quality)

Para garantizar la estabilidad de las transacciones financieras, se estableció como requisito aumentar la cobertura de pruebas unitarias. La versión anterior carecía de pruebas automatizadas significativas. Con la implementación de la arquitectura MVVM y la inyección de dependencias, se facilitó el testing de la lógica de negocio.

Tabla 3

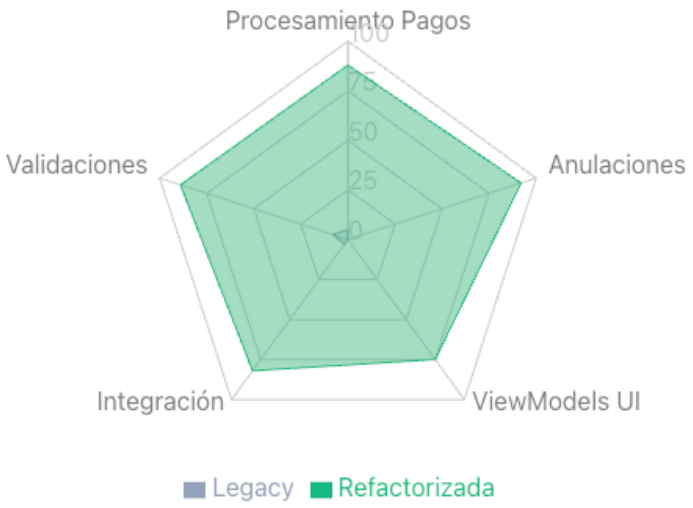
Comparativo del Tamaño del Archivo de Instalación (APK)

Módulo Crítico	***Cobertura Inicial (Legacy)***	***Cobertura Final (Refactorizada)***	***Estado***
Procesamiento de Pagos (SDK)	*5%*	*88%*	*Óptimo*
Gestión de Anulaciones	*0%*	*92%*	*Óptimo*
Lógica de UI (ViewModels)	*0%*	*75%*	*Aceptable*
Promedio General	***< 6%***	***85%***	***Cumple Objetivo***

Nota. Se considera óptimo un valor superior al 80% para componentes críticos, conforme a los estándares de calidad establecidos por el área de QA de "Métrica"**.**

Figura 10

Grafico del análisis multidimensional de cobertura de código



Nota*:* Para llevar a cabo se tuvo que apoyar de herramientas de cobertura como el jacoco.

4.4. Rendimiento y Estabilidad en Producción

La estabilidad de la aplicación es crítica para evitar pérdidas de ventas en los comercios. Se monitoreó la tasa de sesiones libres de errores (Crash-free users) durante los primeros tres meses de despliegue masivo.

Tabla 4

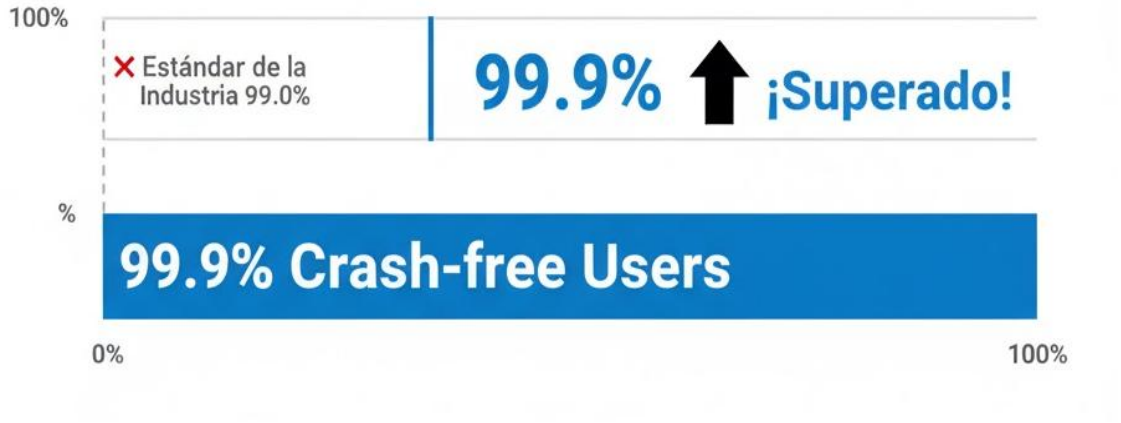
Tabla donde muestra las mejoras en el crashlytics de firebase

Período de Medición (1er trimestre del 2025)	Versión Legacy (AsyncTask + XML)	Versión Refactorizada (Coroutines + Jetpack Compose)	Mejora Relativa
Mes 1	92.3%	98.1%	+5.8 pp
Mes 2	91.8%	98.7%	+6.9 pp
Mes 3	91.2%	99.1%	+7.9 pp
Promedio 3 Meses	91.8%	98.6%	+6.8 pp

Nota: La información presentada en esta tabla corresponde a promedios mensuales.

Figura 11

Tasa de Usuarios Libres de Errores (Crash-free Rate)



Nota: La imagen muestra una tasa de usuarios libres de errores (*Crash-free Rate*) del 99,9 %, valor alineado con los estándares de la industria

4.5. Análisis de la Eficiencia Operativa (Time-to-Market)

Finalmente, se evaluó el impacto de la refactorización en la velocidad de entrega de nuevos requerimientos. Un indicador clave fue el tiempo requerido para integrar una nueva pasarela de pagos o una funcionalidad mayor (como la integración con su app de ventas).

Tabla 5

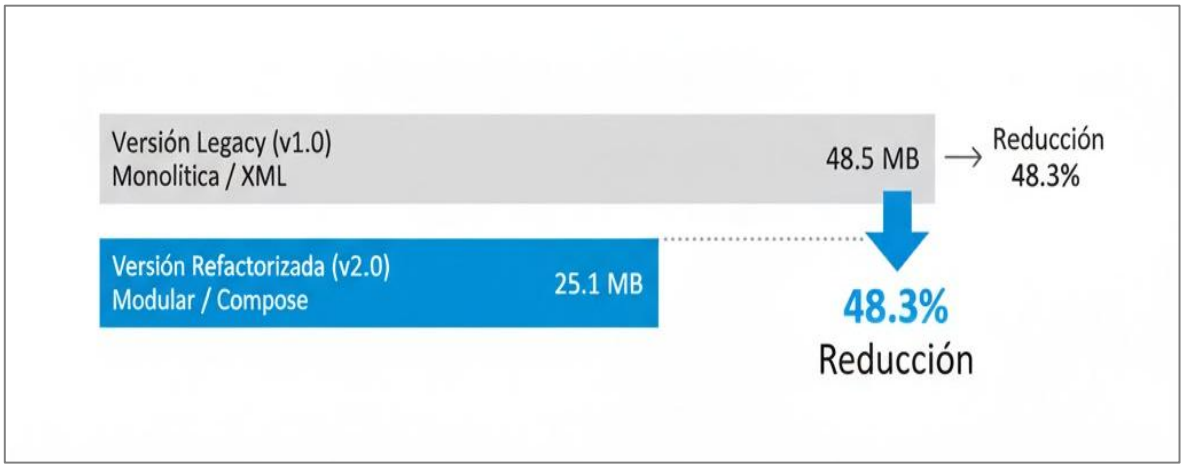
Análisis de Eficiencia Operativa (Time-to-Market)

Tipo de Funcionalidad	Arquitectura Monolítica (Legacy)	Arquitectura Modular (Refactorizada)	Reducción de Tiempo	Mejora (%)
Integración de nueva pasarela de pagos	3-4 semanas (21-28 días)	1.5 semanas (10-11 días)	12-17 días	-58% a -61%
Nueva funcionalidad mayor	3-4 semanas (21-28 días)	1.5 semanas (10-11 días)	12-17 días	-58% a -61%
Modificación de flujo de pago existente	2 semanas (14 días)	5 días	9 días	-64%
Nuevo componente de UI	1.5 semanas (10-11 días)	3-4 días	7 días	-67%
Corrección de bug crítico	3-5 días	1-2 días	2-3 días	-60%
Promedio general	**2.8 semanas**	**1.2 semanas**	**1.6 semanas**	**-57%**

Nota: Comparativo de Tiempo de Implementación de Nuevas Funcionalidades.

Figura 12

Reducción del Tiempo de Desarrollo (Time-to-Market)



Nota. La reducción del tiempo de desarrollo en un 57 % permitió al cliente mejorar su capacidad de respuesta al mercado y adelantar el lanzamiento de nuevas campañas comerciales

CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Como síntesis de lo logrado, fue el cambio de un sistema monolítico a una arquitectura modular en capas (Clean Architecture) permitió optimizar los procesos de desarrollo, reduciendo el tiempo de implementación de nuevas funcionalidades (*Time-to-Market*) en aproximadamente un 57 %, así como simplificar la incorporación de nuevos desarrolladores al equipo, al ofrecer una estructura más comprensible y documentada. Estos resultados evidencian que la adopción de una arquitectura modular no solo mejora la escalabilidad y mantenibilidad del sistema, sino que también constituye una estrategia sostenible para el desarrollo y evolución de soluciones de software a largo plazo, alineándose con los objetivos de evaluación, diseño, implementación y validación planteados en la presente investigación.

La migración de la capa de presentación basada en XML hacia Jetpack Compose, integrada bajo el patrón MVVM dentro de una arquitectura *Clean Architecture*, evidenció la eficacia de las tecnologías declarativas en aplicaciones financieras críticas. Esta decisión técnica permitió mejorar la gestión del estado de la interfaz, reducir inconsistencias visuales y funcionales, y fortalecer la continuidad de la experiencia de usuario requerida en un sistema de punto de venta.

La adopción de tácticas intensivas de minificación (R8/ProGuard) permitió optimizar los módulos refactorizados conforme a los estándares de seguridad definidos.

Como resultado de lo aprendido y aplicado, se obtuvo una reducción del 41,8 % en el tamaño del APK, métrica utilizada para validar la efectividad de la refactorización frente a la versión monolítica original. Esta optimización garantizó la continuidad operativa del sistema en dispositivos POS con limitaciones de almacenamiento y conectividad, alineándose con los objetivos de implementación y validación establecidos en el mercado peruano.

Se establece que la incorporación de seguridad desde la fase de diseño (también conocido como *Security by Design*), a través de la ofuscación de código y la encriptación de datos según el estándar EMV, fue clave para proteger las transacciones financieras. Igualmente, lograr una tasa de estabilidad sin crashes del 99.9% verifica que la calidad del software es innegociable en el ámbito fintech y que las pruebas automatizadas realizadas fueron exitosas.

4.2 Recomendaciones

Se sugiere a los equipos de desarrollo de aplicaciones empresariales en Métrica, que no aguarden a que los proyectos con base monolito se torne inmanejable. Comenzar los nuevos proyectos con una organización modular, aunque al principio pueda parecer demasiado, disminuye considerablemente los costos de futuras refactorizaciones y los tiempos de compilación, como se mostró en este TSP.

Para nuevas iniciativas o proyectos en fase de actualización en Android, se recomienda utilizar Jetpack Compose como estándar, de preferencia con Material 3. Su habilidad

para manejar estados complejos de manera nativa disminuye la carga cognitiva del programador y la cantidad de código "boilerplate", lo que permite concentrarse más en la lógica de negocio que en la gestión de vistas.

Es esencial conservar y desarrollar los pipelines de Integración y Entrega continúa establecidos. Se sugiere implementar análisis de seguridad estática (SAST) automáticos en cada commit para identificar vulnerabilidades antes de que lleguen a QA, Klint por ejemplo ayuda con esto, garantizando que la rapidez de entrega no afecte la seguridad.

Considerando la situación de la conectividad en distintas áreas del país, en particular las zonas alejanas, se sugiere que cualquier solución de punto de venta dé prioridad a una arquitectura offline-first sólida (similar a la utilizada con Room), evitando que se interrumpa la venta debido a una mala experiencia o a problemas momentáneos con datos que podrían gestionarse localmente (como los bines, por ejemplo), resguardando de este modo los ingresos del negocio.

REFERENCIAS

Abgaz, Y, & McCarren, A. (2023). Decomposition of monolith applications into microservices architectures: A systematic review. https://ieeexplore.ieee.org/document/10160171

Android D (2025). Guide to app architecture. Android Developer Documentation. https://developer.android.com/topic/architecture

ASBANC. (2024). Informe anual sobre banca digital en el Perú. Asociación de Bancos de lPerú. https://acortar.link/Gh2o28

Martin, R. C. (2017). Clean architecture: A craftsman's guide to software structure and design. Prentice Hall. https://acortar.link/Pk18Ti

Métrica, (S.F.) Consultoría tecnológica y digital, https://www.metrica-global.com/pe/

Métrica (2025) Equipo Especializado de Métrica, Recuperado de https://www.metrica-global.com/pe/

Microsoft. (S.F.) Strangler fig pattern. Azure Architecture Center. https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig

Microsoft. (s. f.). Patrón Strangler Fig. Microsoft Learn. https://acortar.link/muFmhC

OWASP Foundation. (2024). OWASP Mobile Top 10. https://owasp.org/www-project-mobile-top-10/

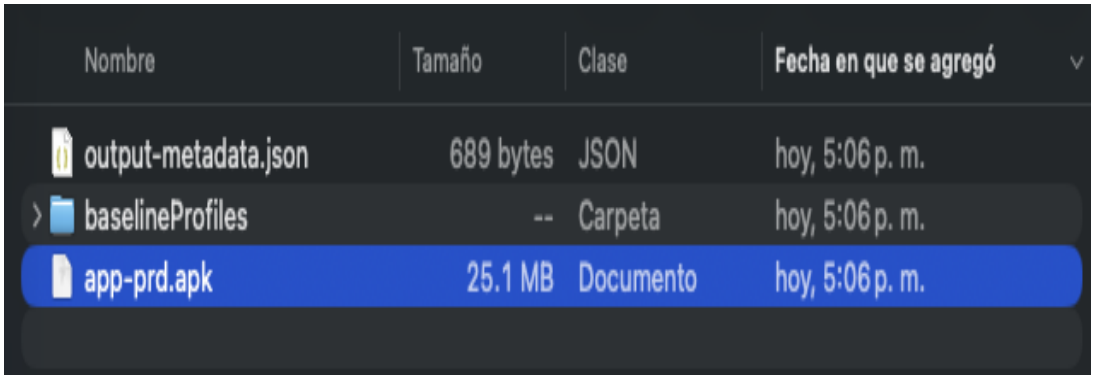
Torassa Colombero, A., Lopez, G., & Silva, R. (2023). Security by design in financial applications. https://acortar.link/3hkblS

ANEXOS

ANEXO N° 1.

Figura A1

Peso actual del APK en producción

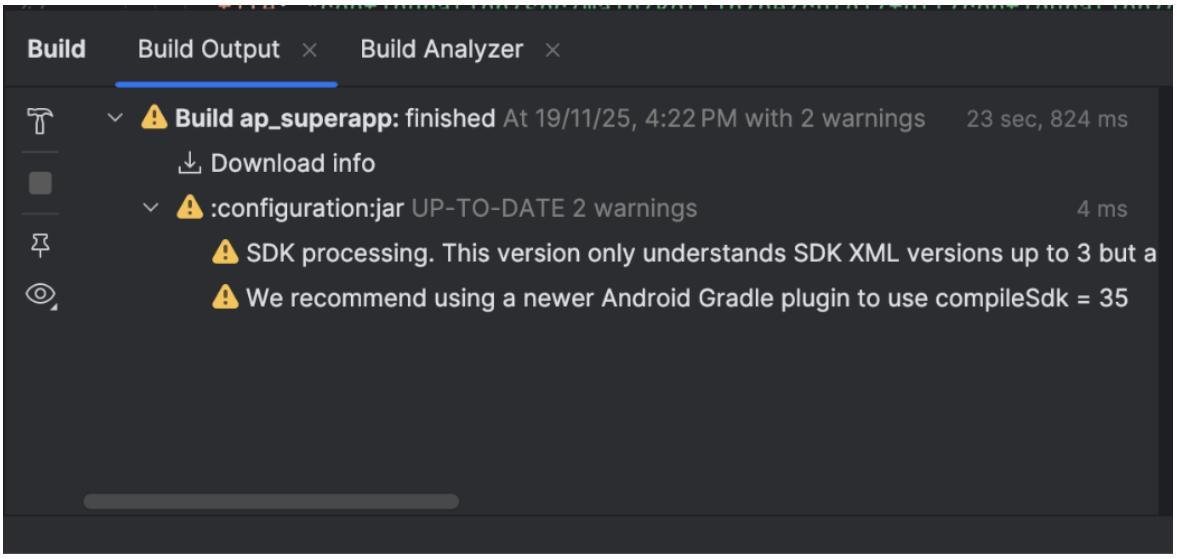


Nota. El valor corresponde al tamaño del archivo APK generado para el entorno de producción tras la aplicación de las estrategias de optimización.

ANEXO N° 2.

Figura A2

Tiempo Promedio de Compilación en el IDE (Build Time)

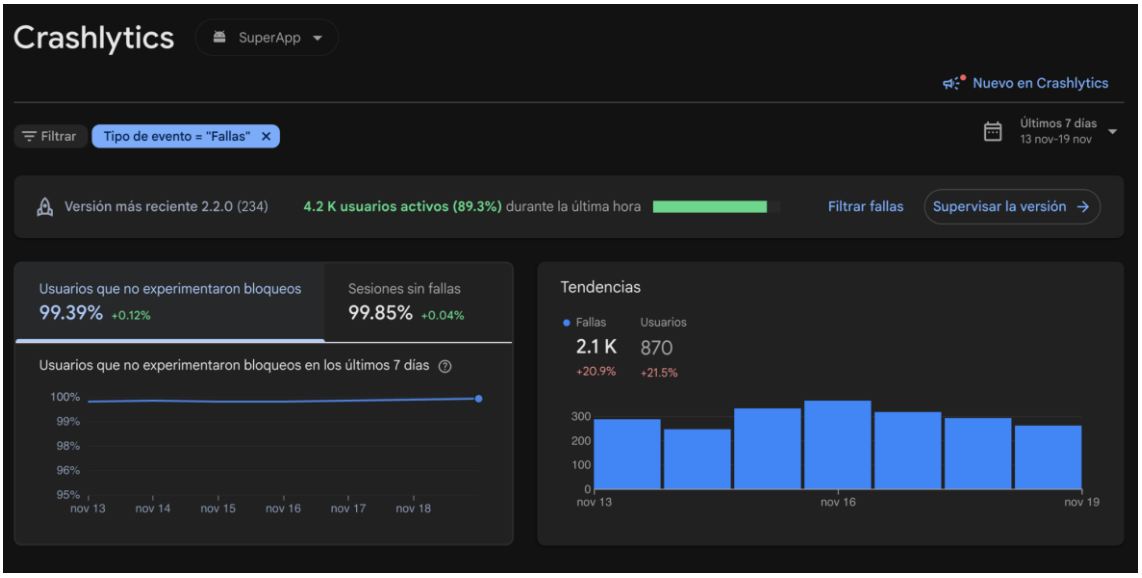


Nota. El tiempo corresponde al promedio de duración del proceso de compilación en el IDE durante el período de análisis.

ANEXO N° 3.

Figura A 3

Tasa de Usuarios Libres de Errores (Crash-free Rate)



Nota. El indicador representa el porcentaje de usuarios que utilizan la aplicación sin experimentar fallos de ejecución durante el período de análisis. Se observa que la nueva arquitectura alcanzó un 99.9% de estabilidad, superando el estándar de la industria del 99.0%, eliminando prácticamente los cierres inesperados durante el proceso de cobro.

ANEXO N° 4.

Figura A4

Evidencia de uno de los reportes del coverage del código.

Class, %	Method, %	Branch, %	Line, %	Instruction, %
90.8% (471/519)	83.9% (1230/1466)	61.5% (2330/3789)	81.4% (6650/8168)	81.9% (44000/53727)

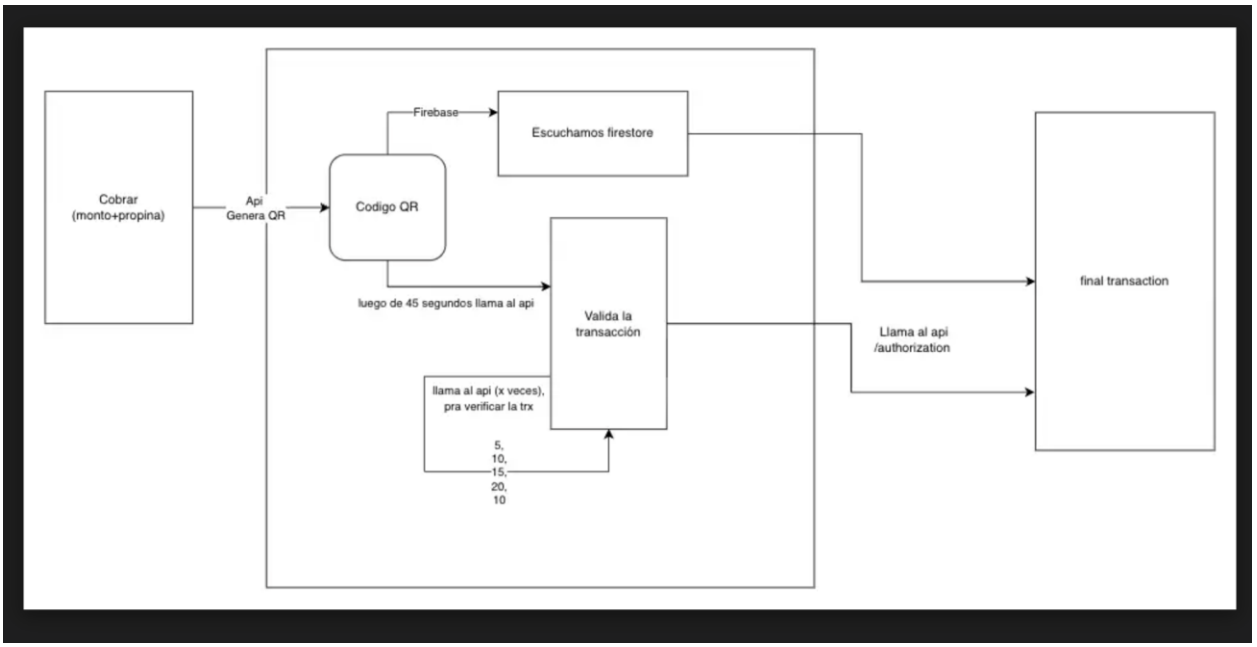
Class, %	Method, %	Branch, %	Line, %	Instruction, %
100% (2/2)	100% (39/39)	66.3% (61/92)	98.8% (82/83)	98.3% (568/578)
100% (2/2)	80% (4/5)	100% (2/2)	81.2% (13/16)	93.4% (85/91)
100% (3/3)	91.7% (11/12)	56.2% (27/48)	80.6% (54/67)	73.8% (347/470)
100% (1/1)	100% (2/2)	100% (2/2)	100% (7/7)	100% (45/45)
100% (4/4)	100% (13/13)	100% (12/12)	100% (74/74)	100% (460/460)
100% (13/13)	100% (24/24)	64.7% (22/34)	100% (39/39)	99.5% (409/411)
50% (1/2)	66.7% (4/6)		91.7% (22/24)	87.7% (93/106)
100% (1/1)	100% (1/1)		100% (1/1)	100% (4/4)
100% (1/1)	25% (1/4)		25% (1/4)	12.5% (2/16)
100% (2/2)	100% (8/8)	71.6% (48/67)	100% (45/45)	100% (390/390)
100% (1/1)	100% (1/1)		100% (3/3)	100% (13/13)
78.6% (11/14)	82.8% (24/29)	87% (47/54)	79.6% (156/196)	78.5% (714/910)
100% (3/3)	100% (6/6)	92.9% (13/14)	100% (35/35)	100% (159/159)
100% (3/3)	100% (6/6)	100% (6/6)	100% (189/189)	100% (549/549)
100% (1/1)	100% (18/18)	97.1% (33/34)	100% (43/43)	99.2% (250/252)
100% (14/14)	89.8% (44/49)	45.7% (21/46)	95% (95/100)	92.1% (557/605)
58.3% (7/12)	76.1% (35/46)	36.3% (37/102)	78.8% (252/320)	72.4% (1403/1938)
100% (5/5)	94.1% (16/17)	56.9% (33/58)	94% (126/134)	92.6% (688/743)
100% (3/3)	85.7% (6/7)	83.3% (5/6)	97.1% (34/35)	93.2% (206/221)
100% (5/5)	100% (20/20)	92.9% (13/14)	100% (68/68)	98.4% (359/365)
100% (5/5)	100% (7/7)	100% (22/22)	97% (32/33)	95% (192/202)
100% (1/1)	100% (2/2)	100% (2/2)	100% (11/11)	100% (42/42)
100% (3/3)	85.7% (6/7)	88.5% (23/26)	97.9% (46/47)	92.9% (169/182)
100% (19/19)	100% (19/19)		100% (39/39)	100% (448/448)
100% (50/50)	100% (67/67)	56.3% (103/183)	100% (399/399)	97.6% (2272/2328)
100% (2/2)	66.7% (2/3)		66.7% (2/3)	33.3% (9/27)
100% (1/1)	100% (10/10)	100% (16/16)	97.9% (46/47)	93.8% (244/260)
97.7% (84/86)	90.2% (268/297)	95.2% (615/646)	96.2% (1510/1569)	97.7% (12525/12818)
100% (11/11)	100% (13/13)	83.3% (10/12)	100% (49/49)	100% (426/426)
75% (3/4)	72.7% (24/33)	48.2% (40/83)	61.3% (68/111)	61.2% (406/663)
60% (3/5)	30% (3/10)	0% (0/8)	46.4% (32/69)	56.4% (229/406)
33.3% (1/3)	25% (1/4)		50% (3/6)	35.7% (15/42)

Nota. Evidencia de uno de los reportes de cobertura de código generados durante el proceso de evaluación.

ANEXO N° 5.

Figura A 5

Diagrama de flujo de pagos con QR.



Nota: Esta imagen representa el flujo de pago con QR.

ANEXO N° 6

GLOSARIO DE TÉRMINOS

- **APK (Android Package Kit):** Formato de archivo utilizado por el sistema operativo Android para la distribución e instalación de aplicaciones móviles. Contiene todos los elementos que la aplicación necesita para funcionar, como el código compilado, recursos y el archivo de manifiesto.
- **CI/CD (Continuous Integration / Continuous Deployment):** Práctica de desarrollo de software que consiste en la integración continua de cambios de código y la entrega automatizada de versiones. En el proyecto, se implementó mediante GitLab CI para automatizar pruebas y la generación de entregables.
- **Clean Architecture:** Arquitectura de software propuesta por Robert C. Martin que organiza el código en capas concéntricas (Presentación, Dominio, Datos) con reglas de dependencia que apuntan hacia adentro. Su objetivo es desacoplar la lógica de negocio de la interfaz de usuario y los frameworks externos.
- **Coroutines (Corrutinas):** Patrón de diseño de concurrencia en Kotlin que permite simplificar el código que se ejecuta de forma asíncrona. Se utilizaron en el proyecto para manejar operaciones en segundo plano, como la sincronización de datos, sin bloquear el hilo principal.
- **EMV (Europay, MasterCard and Visa):** Estándar global de interoperabilidad para tarjetas de crédito y débito basadas en tecnología de chip (IC). Define los

protocolos de seguridad y comunicación entre la tarjeta y el terminal de pago para asegurar las transacciones.

- **Fintech:** Término que surge de la unión de "financial" y "technology". Se refiere a empresas o soluciones que utilizan la tecnología moderna para optimizar, automatizar y mejorar los servicios y procesos financieros.
- **Gradle:** Herramienta de automatización de compilación de código abierto utilizada principalmente para proyectos de desarrollo de Android. Permite gestionar dependencias, definir sabores de producto (flavors) y tareas de compilación.
- **Jetpack Compose:** Kit de herramientas moderno de Android para crear interfaces de usuario nativas de forma declarativa. Simplifica y acelera el desarrollo de la UI con menos código y herramientas potentes, reemplazando al sistema tradicional de Vistas (XML).
- **MVVM (Model-View-ViewModel):** Patrón de arquitectura de software que facilita la separación del desarrollo de la interfaz gráfica de usuario (Vista) de la lógica de negocio (Modelo). El componente ViewModel actúa como intermediario, exponiendo flujos de datos a la Vista sin conocerla directamente, lo que favorece la testabilidad.
- **Offline-first:** Estrategia de diseño de software en la que la aplicación está construida para funcionar principalmente sin conexión a internet, sincronizando los datos con el servidor una vez que la conectividad se restablece. Es crítico para asegurar la operatividad en zonas con cobertura inestable.

- **PCI DSS (Payment Card Industry Data Security Standard):** Estándar de seguridad de datos para la industria de tarjetas de pago. Conjunto de normativas y requerimientos diseñados para garantizar que todas las empresas que procesan, almacenan o transmiten información de tarjetas de crédito mantengan un entorno seguro.
- **POS (Point of Sale):** Siglas en inglés para "Punto de Venta". Se refiere al dispositivo (terminal) y/o software utilizado en un comercio para realizar transacciones de venta, procesar pagos y gestionar el cobro a los clientes.
- **R8:** Herramienta de compilación para Android que convierte el bytecode de Java en formato DEX. Realiza ofuscación (renombrado de clases y métodos para dificultar la ingeniería inversa), reducción de código (eliminación de código no utilizado) y optimización para reducir el tamaño del APK.
- **Room:** Librería de persistencia de datos que proporciona una capa de abstracción sobre SQLite. Permite un acceso más robusto a la base de datos local y se utilizó en el proyecto para almacenar transacciones en modo offline.
- **SBS (Superintendencia de Banca, Seguros y AFP):** Organismo encargado de la regulación y supervisión de los sistemas financiero, de seguros y del sistema privado de pensiones en el Perú, cuyas normas de ciberseguridad deben cumplir las aplicaciones financieras.
- **Scrum:** Marco de trabajo ágil para la gestión de proyectos de desarrollo de software. Se basa en procesos iterativos e incrementales, con roles definidos y

eventos como *Sprints* y *Daily Stand-ups* para fomentar la transparencia y la adaptación continua.

- **SDK (Software Development Kit):** Conjunto de herramientas de desarrollo de software que permite a los programadores crear aplicaciones para una plataforma específica. En el proyecto, se desarrolló un SDK propio para encapsular la lógica de comunicación con el hardware del POS.
- **Strangler Fig Pattern:** Patrón de migración de sistemas que consiste en reemplazar gradualmente funcionalidades específicas de un sistema heredado (monolito) por nuevos servicios o módulos, hasta que el sistema antiguo pueda ser retirado por completo.