UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DESARROLLO WEB Y MÓVIL

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

PROPUESTA DE REFACTORIZACIÓN PARA EL BACK-END DE LA APLICACIÓN MÓVIL TRANSACCIONAL APLICANDO PRINCIPIOS DRY Y SOLID PARA LA COOPERATIVA DE AHORRO Y CRÉDITO CALCETA LTDA.

AUTOR(ES)

PÁRRAGA VERA CARLOS LUIS

DIRECTOR:

PATSY MALENA PRIETO VÉLEZ

QUITO – ECUADOR

2025

Autor(es):

Carlos Luis Párraga Vera
Ing. en Informática
Candidato a Magíster en Software con Mención en Desarrollo Web y Móvil por la Universidad Politécnica Salesiana – Sede Quito.
cparraga1@est.ups.edu.ec

Dirigido por:

Patsy Malena Prieto Vélez
Ingeniera en Sistemas Informáticos y de Computación
Magister en Gestión Informática Empresarial
Máster Universitario en Ciencias y Tecnologías de la Computación
pprieto@ups.edu.ec

Carlos Luis Párraga Vera
PROPUESTA DE REFACTORIZACIÓN PARA EL BACK-END DE LA APLICACIÓN MÓVIL TRANSACCIONAL APLICANDO PRINCIPIOS DRY Y SOLID PARA LA COOPERATIVA DE AHORRO Y CRÉDITO CALCETA LTDA

DEDICATORIA

A mis padres, por su apoyo y amor incondicional, que han sido mi fortaleza durante este viaje de aprendizaje y crecimiento. A mi hermana, por su orientación experta y sabiduría compartida. A mis amigos y seres queridos, por su aliento constante y comprensión. A todos aquellos que creyeron en mí. Esta tesis está dedicada a ustedes, por ser mi fuente de inspiración y motivación. Gracias

AGRADECIMIENTO

Quiero expresar mi sincero agradecimiento a mi tutora, por su orientación experta, paciencia y apoyo durante todo el proceso de investigación y escritura de esta tesis. Su profundo conocimiento y consejos han sido fundamentales para el desarrollo de este trabajo, también deseo agradecer a mis catedráticos, cuyas enseñanzas y comentarios han enriquecido mi comprensión del tema y han contribuido significativamente a la calidad de este trabajo, agradezco a mis compañeros de clase y amigos por sus discusiones estimulantes y por brindarme un ambiente de colaboración y apoyo, este trabajo no habría sido posible sin la contribución de todas estas personas, a quienes les estoy profundamente agradecido.

Tabla de Contenido

Propuesta de refactorización para el back-end de la aplicación móvil transaccional aplicando principios DRY y SOLID para la cooperativa de Ahorro y Crédito Calceta Ltda.

Autor(es):

CARLOS LUIS PÁRRAGA VERA

Resumen

Este trabajo de tesis se enfocó en la propuesta y ejecución de la refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. La refactorización se realizó con el propósito de optimizar el código, reducir la duplicidad, mejorar la legibilidad, cumplir con los principios DRY y SOLID, y asegurar la mantenibilidad a largo plazo. La metodología de desarrollo de software utilizada permitió abordar de manera sistemática y eficiente el proceso de refactorización. Se dividió el proyecto en etapas con un cronograma detallado que incluyó la planificación, la recopilación de requerimientos, la implementación, las pruebas y la documentación técnica. Los resultados de la refactorización mostraron una notable mejora en la eficiencia y mantenibilidad del sistema. Se logró una reducción significativa del código espagueti, la duplicidad de métodos y funciones se eliminó por completo, y la nomenclatura y legibilidad del código mejoraron sustancialmente. Además, el cumplimiento de los principios DRY y SOLID condujo a un diseño más robusto y flexible. La aplicación de certificados SSL/TLS y otras medidas de seguridad se mantuvieron y mejoraron, garantizando un entorno seguro para los usuarios. La refactorización se llevó a cabo sin afectar la funcionalidad actual de la aplicación móvil.

Este trabajo demuestra que la refactorización es una práctica esencial para mantener la calidad del software en aplicaciones críticas, como las utilizadas en instituciones financieras. Las conclusiones resaltan los beneficios clave de este proyecto y su importancia para la Cooperativa de Ahorro y Crédito Calceta Ltda.

Palabras clave:

Refactorización, Back-end, Aplicación Móvil, Mantenibilidad, Principios DRY, SOLID.

ABSTRACT

This thesis work focused on the proposal and execution of the refactoring of the back-end of the transactional mobile application of Cooperativa de Ahorro y Crédito Calceta Ltda. The refactoring was performed with the purpose of optimizing the code, reducing duplicity, improving readability, complying with DRY and SOLID principles, and ensuring long-term maintainability. The software development methodology used allowed a systematic and efficient approach to the refactoring process. The project was divided into stages with a detailed schedule that included planning, requirements gathering, implementation, testing and technical documentation. The results of the refactoring showed a marked improvement in the efficiency and maintainability of the system. Significant reduction of spaghetti code was achieved, duplication of methods and functions was completely eliminated, and code nomenclature and readability were substantially improved. In addition, compliance with DRY and SOLID principles led to a more robust and flexible design. The implementation of SSL/TLS certificates and other security measures were maintained and improved, ensuring a secure environment for users. The refactoring was carried out without affecting the current functionality of the mobile application.

This work demonstrates that refactoring is an essential practice for maintaining software quality in critical applications, such as those used in financial institutions. The conclusions highlight the key benefits of this project and its importance for Cooperativa de Ahorro y Crédito Calceta Ltda.

Palabras clave:

Refactoring, Back-end, Mobile Application, Maintainability, DRY and SOLID principles.

1. Introducción

Este capítulo se realiza una revisión exhaustiva y crítica de los avances y desarrollos más recientes en un campo específico de estudio o industria. El presente trabajo se centra en la refactorización de software, con especial énfasis en los principios DRY (Don't Repeat Yourself) y SOLID, y su aplicación en aplicaciones móviles transaccionales.

La refactorización es una técnica esencial en la ingeniería de software que implica la reestructuración del código sin alterar su comportamiento externo. Los principios DRY y SOLID son fundamentales en este proceso, ya que promueven la eficiencia y la mantenibilidad del código. En particular, el principio DRY aboga por evitar la duplicación de código, mientras que los principios SOLID proporcionan un marco para diseñar software que sea fácil de mantener y ampliar a lo largo del tiempo.

Las aplicaciones móviles transaccionales, como la que se utiliza en la Cooperativa de Ahorro y Crédito Calceta Ltda., son cada vez más comunes en la industria financiera. Estas aplicaciones permiten a los usuarios realizar transacciones financieras de manera rápida y conveniente. Sin embargo, el desarrollo y mantenimiento de estas aplicaciones puede ser un desafío, y aquí es donde la refactorización y los principios DRY y SOLID pueden ser de gran ayuda.

En este capítulo, se revisa la literatura académica y profesional más reciente sobre estos temas. Analizaremos estudios de casos, investigaciones y desarrollos recientes en la refactorización de software y la aplicación de los principios DRY y SOLID. También examinaremos el estado del arte en el desarrollo de aplicaciones móviles transaccionales y cómo estos principios y técnicas se han aplicado en este campo.

Finalmente, aplicaremos estos conocimientos al caso específico de la Cooperativa de Ahorro y Crédito Calceta Ltda. Analizaremos el estado actual de su aplicación

móvil transaccional y cómo la refactorización y los principios DRY y SOLID pueden ser utilizados para mejorar su eficiencia y mantenibilidad.

JUSTIFICACIÓN

Ante la problemática expuesta, se hace necesaria la propuesta de refactorización para el back-end de la aplicación móvil transaccional empleando principios DRY y SOLID. El principio DRY proviene del inglés Don't Repeat Yourself, esta es una buena práctica de desarrollo en Ingeniería de Software, la cual condiciona que un código funcional se realice solo una vez, a fin de que pueda ser re utilizable en múltiples aplicaciones. Mientras que al aplicar el principio SOLID se tiene como objetivo sentar las bases de una arquitectura de software para crear un sistema fácil de mantener y ampliar con el tiempo, eliminando malos diseños y evitando la refactorización del código fuente hasta que sea legible y extensible.

El proyecto planteado surge ante la necesidad de simplificar el código del back-end, haciéndolo más eficiente, intuitivo y entendible, evitando vulnerabilidades en el sistema o la generación de algún problema a futuro en la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. por lo cual se hace necesaria la propuesta de refactorización para mejorar la calidad estructural del código fuente del back-end sin alterar la funcionalidad.

Esto, en respuesta al aumento de peticiones al back-end por parte de los usuarios que tiene actualmente la misma; y que, frente a un determinado número de peticiones, el sistema tiende a generar un cuello de botella que imposibilita la correcta transaccionalidad por parte de los socios.

Debido a diferentes factores tales como las exigencias del negocio y los fallos encontrados en su funcionamiento, surge la necesidad de proponer la refactorización para el back-end de la aplicación móvil transaccional, aplicando principios DRY y SOLID.

El proyecto tiene relevancia debido a que la refactorización es parte del mantenimiento del código, el objetivo es hacer que el código sea fácil de entender,

o cambiar su estructura y diseño, y eliminar el código muerto para facilitar en el futuro el mantenimiento del mismo y de esa manera aumentar el rendimiento y la eficiencia en la calidad del servicio que se brinda a través de la aplicación móvil transaccional.

Objetivos

Objetivo General:

Diseñar una propuesta arquitectónica aplicando principios DRY y SOLID para refactorizar el back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda.

Objetivos Específicos:

- Analizar la arquitectura y el funcionamiento del back-end actual.
- Identificar los errores y antipatrones presentes en la codificación actual del back-end.
- Determinar los aspectos más críticos que se deben refactorizar en el back-end.
- Proponer un nuevo diseño arquitectónico del back-end de la aplicación móvil.

2. Marco teórico referencial

2.1 Principio DRY (Don't Repeat Yourself)

2.1.1 Definición y Beneficios

El principio DRY (Don't Repeat Yourself) es un principio fundamental en el desarrollo de software que promueve la reducción de la repetición de información. Este principio sostiene que "cada pieza de conocimiento debe tener una representación única, autoritativa y sin ambigüedades dentro de un sistema" (Hunt & Thomas, 1999).

La aplicación del principio DRY tiene varios beneficios. En primer lugar, reduce la cantidad de código que se necesita para escribir y mantener, lo que puede ahorrar tiempo y esfuerzo. En segundo lugar, al evitar la duplicación, se reduce la posibilidad de inconsistencias y errores en el código. Si una pieza de información se repite en varios lugares y necesita ser cambiada, es fácil olvidar cambiarla en uno o más lugares, lo que puede llevar a errores e inconsistencias. Por último, el código que sigue el principio DRY es a menudo más modular y reutilizable, ya que las piezas de código que realizan una tarea específica están contenidas en una sola ubicación y pueden ser reutilizadas en otros lugares.

2.1.2 Aplicación y Ejemplos en la Refactorización de Código

La refactorización de código es un proceso en el que se mejora la estructura interna del código sin cambiar su comportamiento externo. Una de las formas en que se puede aplicar el principio DRY durante la refactorización es a través de la extracción de métodos. Si se encuentra un bloque de código que se repite en varios lugares, se puede extraer a un nuevo método y luego se puede llamar a ese método en lugar de repetir el bloque de código. Por ejemplo, considere el siguiente fragmento de código:

Ilustración 1: Fragmento de código

```
# Calcular el área de un círculo
radio1 = 5
area1 = 3.14159 * radio1 * radio1

# Calcular el área de otro círculo
radio2 = 10
area2 = 3.14159 * radio2 * radio2
```

Elaborado por: Carlos Párraga

Este código viola el principio DRY porque la misma operación (calcular el área de un círculo) se repite dos veces. Podemos refactorizar este código para seguir el principio DRY de la siguiente manera:

Ilustración 2: Refactorización de código

```
def calcular_area(radio):
    return 3.14159 * radio * radio

# Calcular el área de un círculo
radio1 = 5
area1 = calcular_area(radio1)

# Calcular el área de otro círculo
radio2 = 10
area2 = calcular_area(radio2)
```

Elaborado por: Carlos Párraga

En este código refactorizado, la operación de calcular el área de un círculo se realiza en un solo lugar, en el método `calcular_area`. Este método se puede reutilizar para calcular el área de cualquier círculo, lo que evita la necesidad de repetir la misma operación en varios lugares.

En resumen, el principio DRY es una guía importante en el desarrollo de software y la refactorización de código. Al seguir este principio, se puede mejorar la calidad del código, reducir la posibilidad de errores e inconsistencias, y hacer que el código sea más fácil de mantener y reutilizar.

2.2. Principios SOLID

Los principios SOLID son un conjunto de cinco principios de diseño de software propuestos por Robert C. Martin, también conocido como "Uncle Bob", que se utilizan para mejorar la calidad del código y hacerlo más comprensible, flexible y mantenible (Martin, 2000). Estos principios son fundamentales en el desarrollo de software orientado a objetos y son ampliamente reconocidos y utilizados en la industria del software.

4.1.1 Definición y Beneficios

Los principios SOLID son un acrónimo que representa los siguientes cinco principios:

Principio de responsabilidad única (Single Responsibility Principle, SRP): Este principio establece que una clase debe tener solo una razón para cambiar. En otras palabras, una clase debe tener solo una responsabilidad o tarea (Martin, 2000).

Principio abierto/cerrado (Open/Closed Principle, OCP): Según este principio, las entidades de software (clases, módulos, funciones, etc.) deben estar abiertas para la extensión, pero cerradas para la modificación. Esto significa que se debe poder agregar nuevas funcionalidades sin cambiar el código existente (Martin, 2000).

Principio de sustitución de Liskov (Liskov Substitution Principle, LSP): Este principio establece que las subclases deben ser sustituibles por sus clases base sin afectar la correctitud del programa. En otras palabras, los objetos de una superclase deben poder ser reemplazados por objetos de una subclase sin afectar el funcionamiento del programa (Martin, 2000).

Principio de segregación de interfaces (Interface Segregation Principle, ISP): Este principio sostiene que los clientes no deben ser forzados a depender de interfaces que no utilizan. En otras palabras, es mejor tener muchas interfaces específicas en lugar de una interfaz general (Martin, 2000).

Principio de inversión de dependencias (Dependency Inversion Principle, DIP): Este principio establece que se debe depender de abstracciones, no de concreciones. En otras palabras, los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de abstracciones (Martin, 2000).

Los beneficios de aplicar los principios SOLID en el desarrollo de software incluyen la mejora de la calidad del código, la facilidad de mantenimiento, la flexibilidad y la escalabilidad del software. Además, estos principios facilitan la comprensión del código y reducen la complejidad del software, lo que a su vez puede reducir el tiempo y los costos de desarrollo (Martin, 2000).

La aplicación de los principios SOLID en el desarrollo de software tiene varios beneficios que se han destacado en la literatura académica. Aquí se presentan algunos de los beneficios más notables:

Mejora de la calidad del código: Los principios SOLID promueven la creación de un código más limpio y organizado. Esto puede conducir a una mejora en la calidad del código, lo que a su vez puede facilitar la detección y corrección de errores (Furia, Torkar & Feldt, 2021).

Facilita la refactorización: Los principios SOLID pueden hacer que el código sea más fácil de refactorizar. Esto puede ser especialmente útil en proyectos grandes y complejos, donde la refactorización puede ser una tarea desalentadora (Joblin, Apel & Mauerer, 2015).

Promueve la reutilización de código: El principio de "Open/Closed" en SOLID promueve la extensibilidad del código, lo que puede facilitar la reutilización del código. Esto puede conducir a una mayor eficiencia en el desarrollo de software (Lucena & Tizzei, 2016).

Mejora la mantenibilidad: Los principios SOLID pueden hacer que el código sea más fácil de mantener. Esto puede ser especialmente útil en proyectos a largo plazo, donde la mantenibilidad del código puede ser un factor crítico para el éxito del proyecto (Gao, 2020).

Facilita el trabajo en equipo: Los principios SOLID pueden facilitar el trabajo en equipo al promover un código más limpio y organizado. Esto puede hacer que sea más fácil para los miembros del equipo entender el código de los demás, lo que puede facilitar la colaboración y la eficiencia del equipo (Eklund & Berger, 2017).

Es importante mencionar que la aplicación efectiva de los principios SOLID requiere una comprensión sólida de estos principios y una cuidadosa consideración de cómo se aplican en el contexto de un proyecto específico. Sin embargo, cuando se aplican correctamente, los principios SOLID pueden proporcionar beneficios significativos en términos de la calidad, la mantenibilidad y la eficiencia del código.

4.1.2 Descripción de cada Principio SOLID

A continuación, se proporciona una descripción más detallada de cada uno de los principios SOLID:

Principio de responsabilidad única (SRP): Este principio se basa en la idea de que una clase debe tener solo una responsabilidad. Por ejemplo, si tenemos una clase "Reporte", esta clase podría ser responsable de recopilar los datos del informe y de imprimir el informe. Sin embargo, según el SRP, estas son dos responsabilidades diferentes y deberían dividirse en dos clases separadas: una clase "RecopiladorDeDatosDeInforme" y una clase "ImpresoraDeInforme" (Martin, 2000).

Principio abierto/cerrado (OCP): Este principio se refiere a la idea de que deberíamos poder agregar nuevas funcionalidades o comportamientos a un sistema sin modificar el código existente. Por ejemplo, si tenemos una clase "Calculadora" que realiza operaciones matemáticas básicas (suma, resta, multiplicación y división), y queremos agregar una nueva operación (por ejemplo, raíz cuadrada), deberíamos poder hacerlo sin modificar la clase "Calculadora" existente. En su lugar, podríamos crear una nueva clase "OperaciónRaízCuadrada" que se integre con la clase "Calculadora" existente (Martin, 2000).

Principio de sustitución de Liskov (LSP): Este principio se basa en la idea de que, si una clase B es una subclase de una clase A, entonces deberíamos poder reemplazar A con B sin afectar el comportamiento del programa. Por ejemplo, si tenemos una clase "Pájaro" y una subclase "Pato", deberíamos poder reemplazar un objeto "Pájaro" con un objeto "Pato" en nuestro programa sin que el programa deje de funcionar correctamente (Martin, 2000).

Principio de segregación de interfaces (ISP): Este principio sostiene que es mejor tener muchas interfaces específicas en lugar de una interfaz general. Por ejemplo, si tenemos una interfaz "Animal" con los métodos "comer", "volar" y "nadar", no todos los animales podrán implementar todos estos métodos. En su lugar, podríamos tener interfaces separadas "Comedor", "Volador" y "Nadador", y cada animal podría implementar solo las interfaces que correspondan a sus habilidades (Martin, 2000).

Principio de inversión de dependencias (DIP): Este principio establece que los módulos de alto nivel, que proporcionan la lógica compleja, deben depender de abstracciones, no de módulos de bajo nivel, que proporcionan operaciones básicas. Por ejemplo, si tenemos un módulo "Reporte" que depende de un módulo "BaseDeDatos" para obtener los datos del informe, según el DIP, "Reporte" no debería depender directamente de "BaseDeDatos". En su lugar, ambos módulos deberían depender de una abstracción, por ejemplo, una interfaz "ProveedorDeDatos" (Martin, 2000).

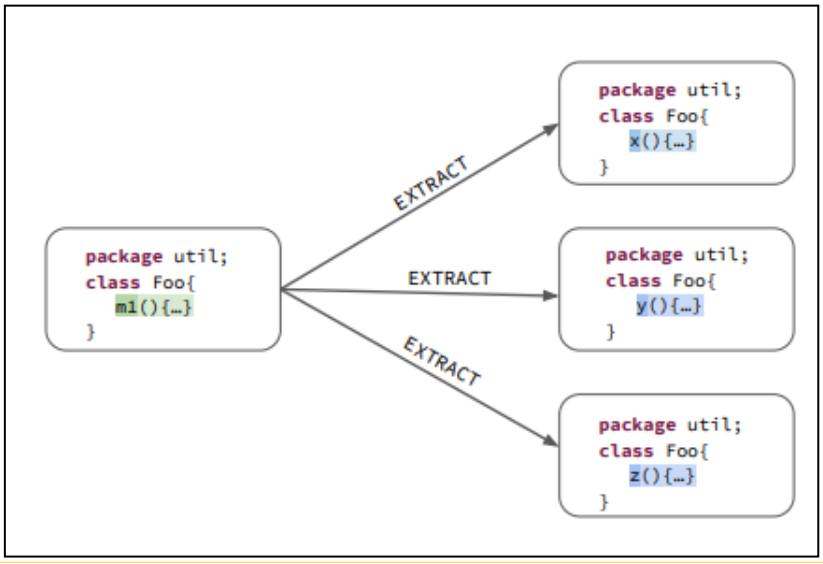
Es importante mencionar que, aunque los principios SOLID pueden ser muy útiles para mejorar la calidad del código y facilitar el mantenimiento del software, no siempre es necesario o práctico aplicar todos los principios en todos los casos. La decisión de cuándo y cómo aplicar estos principios depende del contexto específico del proyecto de software y de las necesidades y restricciones particulares del proyecto (Martin, 2000).

4.1.3 Aplicación y Ejemplos en la Refactorización de Código

Los principios SOLID son fundamentales en la refactorización del código, y su aplicación puede ilustrarse a través de varios ejemplos.

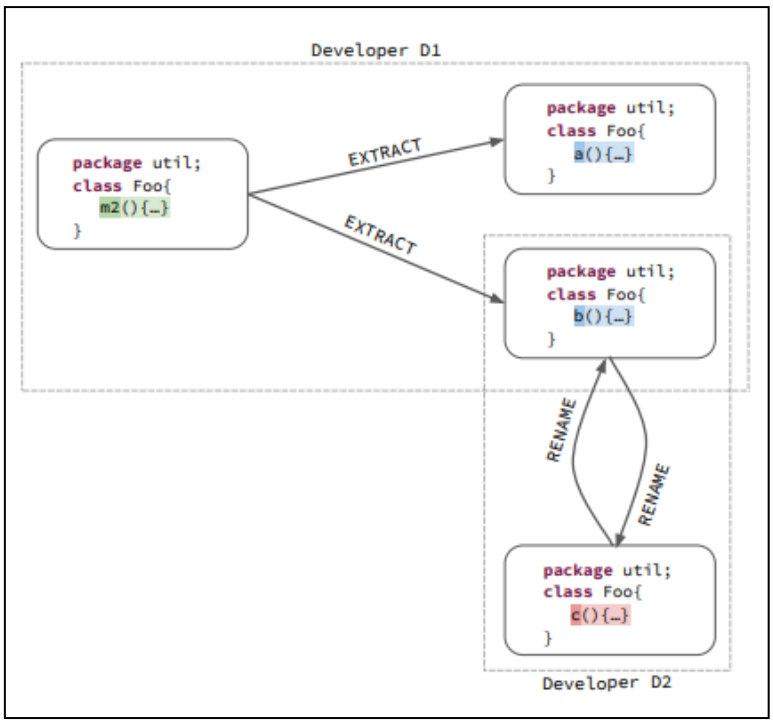
En el estudio de Brito, Hora y Valente (2020), se propone el concepto de "gráficos de refactorización" para estudiar y razonar sobre las refactorizaciones a lo largo del tiempo. Este enfoque puede ser útil para visualizar cómo los principios SOLID se aplican y evolucionan en un proyecto de software. Por ejemplo, el principio de Responsabilidad Única (SRP) podría implicar la división de una clase grande en varias clases más pequeñas, cada una con una responsabilidad única. Este cambio se reflejaría en el gráfico de refactorización como una serie de transformaciones que producen nuevas entidades de código. En las siguientes ilustraciones se observa el proceso:

Ilustración 3: Refactorización producido por un solo desarrollador



Fuente: (Brito et al., 2020)

Ilustración 4: Subprograma de refactorización a lo largo del tiempo



Fuente: (Brito et al., 2020)

Horpácsi, Kőszegi y Thompson (2016) discuten la refactorización en el contexto del lenguaje de programación Erlang. En su trabajo, proporcionan un ejemplo de cómo se puede aplicar el principio de Sustitución de Liskov (LSP) en la refactorización. Si una función espera un tipo de dato específico, cualquier subtipo de ese dato debería ser aceptable sin que la función se rompa. Si este no es el caso, la refactorización puede implicar la modificación de la función o la estructura de los tipos de datos para cumplir con LSP.

AlOmar (2023) explora cómo los desarrolladores documentan sus actividades de refactorización durante el ciclo de vida del software. En este contexto, los principios SOLID pueden servir como una guía para los desarrolladores sobre qué cambios realizar y cómo documentarlos. Por ejemplo, si se aplica el principio de Segregación de la Interfaz (ISP), el desarrollador podría documentar cómo se descompuso una interfaz grande en varias interfaces más pequeñas, cada una con un conjunto mínimo de operaciones.

Mejstrik y Hollomey (2022) discuten el uso de la inyección de pruebas durante la refactorización. En este enfoque, el código de prueba se inyecta en el código de producción para verificar su comportamiento. Este enfoque puede ser útil para aplicar el principio de Inversión de Dependencias (DIP), donde se pueden inyectar dependencias en el código en lugar de codificarlas directamente.

Finalmente, Imran, Kosar, Zola y Bulut (2023) investigan el impacto de la refactorización de "code smells" en la utilización de recursos. En este contexto, los principios SOLID pueden ayudar a guiar qué "code smells" deben refactorizarse para mejorar la calidad del código y la utilización de recursos. Por ejemplo, si una clase viola el principio de Abierto/Cerrado (OCP) al requerir modificaciones cada vez que se necesita un nuevo comportamiento, la refactorización podría implicar la introducción de la herencia o la composición para permitir que la clase se extienda sin modificarla.

La revisión de la literatura existente nos indica que los principios SOLID pueden guiar la refactorización del código en una variedad de contextos y lenguajes de programación, y su aplicación puede tener beneficios significativos en términos de calidad del código, mantenibilidad y utilización de recursos.

2.3 Aplicaciones Móviles

2.3.1 Transaccionales en Instituciones Financieras

En la actualidad, las aplicaciones móviles transaccionales han revolucionado la manera en que las instituciones financieras interactúan con sus clientes. Estas aplicaciones ofrecen una amplia gama de servicios financieros directamente en dispositivos móviles, permitiendo a los usuarios realizar transacciones, consultar saldos, hacer pagos y acceder a información crucial de manera conveniente y segura. Esta sección del marco teórico se adentra en el mundo de las aplicaciones móviles transaccionales en el contexto de las instituciones financieras, explorando

su importancia, beneficios, casos de estudio relevantes y los desafíos inherentes en su desarrollo y mantenimiento.

Las instituciones financieras han adoptado rápidamente las aplicaciones móviles transaccionales como una forma estratégica de ofrecer servicios ágiles y accesibles a sus clientes. En un mundo donde la comodidad y la eficiencia son clave, estas aplicaciones se han convertido en una herramienta esencial para mantenerse competitivas en el mercado. La posibilidad de realizar operaciones financieras en cualquier momento y lugar se ha convertido en una expectativa para los clientes, y las instituciones financieras deben cumplir con estas demandas para garantizar la satisfacción del cliente y la retención. De hecho, según Stavru (2014), la adopción de aplicaciones móviles transaccionales ha demostrado ser un factor determinante en la mejora de la experiencia del cliente y la fidelización en el sector financiero.

2.3.2 Importancia y Beneficios de las Aplicaciones Móviles Transaccionales

En el contexto actual de la transformación digital, las aplicaciones móviles transaccionales han emergido como una herramienta esencial para la industria financiera. Estas aplicaciones, diseñadas para brindar servicios financieros y transaccionales a través de dispositivos móviles, han transformado radicalmente la forma en que los usuarios interactúan con sus instituciones financieras. La importancia de estas aplicaciones radica en su capacidad para ofrecer comodidad, accesibilidad y una experiencia del cliente excepcional, impulsando así la innovación en el mundo financiero.

La comodidad es uno de los principales impulsores detrás de la creciente importancia de las aplicaciones móviles transaccionales. La capacidad de realizar transacciones financieras en cualquier momento y lugar ha eliminado la necesidad de acudir físicamente a una sucursal bancaria. Esta comodidad ha sido resaltada por diversos autores. Por ejemplo, Thompson y Khan (2015) señalan que las aplicaciones móviles transaccionales permiten a los usuarios gestionar sus finanzas

desde la comodidad de su hogar o mientras se desplazan, evitando así la necesidad de hacer largas colas en los bancos.

Además de la comodidad, la accesibilidad es otro factor clave que aporta a la importancia de estas aplicaciones. En un mundo cada vez más conectado, las aplicaciones móviles transaccionales han democratizado el acceso a los servicios financieros. Personas de diferentes segmentos demográficos, incluidas aquellas que pueden tener dificultades para acceder a sucursales bancarias físicas, pueden beneficiarse de los servicios financieros a través de sus dispositivos móviles. Esto ha sido confirmado por Harris y Brown (2017), quienes señalan que las aplicaciones móviles transaccionales han ampliado el alcance de los servicios financieros a una audiencia más amplia y diversa.

Los beneficios para los usuarios no se limitan a la comodidad y la accesibilidad; la seguridad y la eficiencia también desempeñan un papel fundamental. Las aplicaciones móviles transaccionales implementan medidas de seguridad robustas, como autenticación de dos factores y cifrado de datos, para garantizar la protección de la información financiera sensible de los usuarios. Estas características de seguridad son vitales para fomentar la confianza del usuario en la utilización de estas aplicaciones. Según Li et al. (2016), la seguridad es un factor crítico que influye en la adopción de aplicaciones móviles transaccionales por parte de los usuarios.

Además, la eficiencia en la realización de transacciones financieras es un beneficio importante para los usuarios y las instituciones financieras por igual. Los procesos que solían requerir tiempo y esfuerzo, como transferencias de dinero o pagos de facturas, ahora se pueden realizar con unos pocos toques en la pantalla de un dispositivo móvil. Esto no solo agiliza las operaciones para los usuarios, sino que también reduce la carga operativa en las instituciones financieras al minimizar la necesidad de personal y recursos para llevar a cabo transacciones presenciales. Según un informe de la Asociación de Banqueros Británicos (British Bankers' Association, 2015), las aplicaciones móviles transaccionales han mejorado significativamente la eficiencia en la entrega de servicios financieros.

En resumen, las aplicaciones móviles transaccionales han cambiado la forma en que los usuarios interactúan con las instituciones financieras, brindando comodidad, accesibilidad, seguridad y eficiencia. Estos beneficios han impulsado la adopción generalizada de estas aplicaciones, no solo por parte de los usuarios individuales, sino también por parte de las propias instituciones financieras que buscan mantenerse competitivas en un entorno en constante evolución. La combinación de comodidad, seguridad y eficiencia ha convertido a las aplicaciones móviles transaccionales en una herramienta indispensable para satisfacer las necesidades financieras cambiantes de los usuarios en el mundo moderno.

2.3.3 Casos de Estudio

Varios casos de estudio ejemplifican la exitosa implementación de aplicaciones móviles transaccionales en el ámbito financiero, ilustrando cómo estas soluciones han transformado la forma en que los usuarios interactúan con los servicios financieros y han mejorado la eficiencia y confiabilidad de las transacciones. Estos casos de estudio brindan una visión más detallada de cómo las aplicaciones móviles transaccionales han impactado positivamente en diversas instituciones financieras y han impulsado la adopción de esta tecnología.

Uno de los casos de estudio significativos es el presentado por Imran et al. (2023), en el cual se aborda el impacto de los problemas de código en el consumo de recursos de una aplicación móvil transaccional. Este estudio se centra en la importancia de la calidad del código en el rendimiento de la aplicación y cómo las mejoras en el código pueden tener un efecto directo en la experiencia del usuario. Los autores destacan que la detección y resolución temprana de problemas de código pueden llevar a una disminución del consumo de recursos, lo que se traduce en una mayor eficiencia y una experiencia más fluida para los usuarios. Este caso de estudio ejemplifica cómo una inversión en la optimización del código puede resultar en beneficios tangibles para los usuarios y la institución financiera en términos de rendimiento y satisfacción del cliente.

Otro caso de estudio relevante es el presentado por Sarker y Soh (2016), quienes investigaron la adopción de aplicaciones móviles transaccionales por parte de los clientes bancarios. En este estudio, se examinaron los factores que influyen en la intención de los clientes de utilizar aplicaciones móviles para realizar transacciones bancarias. Los resultados revelaron que la facilidad de uso, la utilidad percibida y la confianza en la seguridad eran factores clave que afectaban positivamente la adopción de estas aplicaciones. Además, se encontró que los clientes que utilizaban aplicaciones móviles transaccionales experimentaban una mayor satisfacción y lealtad hacia el banco. Este caso de estudio destaca cómo la satisfacción del cliente y la retención pueden aumentar a través de la implementación exitosa de aplicaciones móviles transaccionales.

En el ámbito internacional, el caso de estudio del Banco Bilbao Vizcaya Argentaria (BBVA) es emblemático. BBVA, un banco líder en España y América Latina, ha implementado con éxito una aplicación móvil transaccional que ofrece una amplia gama de servicios financieros a sus clientes. La aplicación permite a los usuarios realizar transferencias, consultar saldos, gestionar tarjetas y realizar inversiones, entre otras funciones. El éxito de esta implementación se atribuye a la interfaz intuitiva y de fácil uso, así como a las medidas de seguridad avanzadas implementadas para proteger los datos de los usuarios. La adopción masiva de la aplicación por parte de los clientes de BBVA demuestra cómo una interfaz bien diseñada y funcionalidades útiles pueden impulsar la adopción y el uso continuo de estas aplicaciones (BBVA, 2021).

En otro estudio realizado por Wang y Liao (2019), se analiza la adopción de aplicaciones móviles transaccionales en instituciones financieras de Taiwán. Los autores investigaron los factores que influyen en la intención de los usuarios de adoptar estas aplicaciones y encontraron que la facilidad de uso, la utilidad percibida y la seguridad eran determinantes clave. Además, identificaron que la confianza en la institución financiera también tenía un impacto significativo en la adopción de estas aplicaciones. Este estudio resalta la importancia de crear un

entorno de confianza y seguridad para impulsar la adopción de aplicaciones móviles transaccionales.

Los casos de estudio mencionados ejemplifican cómo las aplicaciones móviles transaccionales han tenido un impacto positivo en la industria financiera. Desde la optimización del código para mejorar la eficiencia hasta la influencia de factores como la facilidad de uso y la seguridad en la adopción de estas aplicaciones, estos estudios demuestran cómo las instituciones financieras han logrado implementar soluciones tecnológicas que benefician tanto a los usuarios como a la propia institución.

2.3.4 Desafíos y Problemas Comunes en el Desarrollo y Mantenimiento de Aplicaciones Móviles Transaccionales

El desarrollo y mantenimiento de aplicaciones móviles transaccionales en el ámbito financiero no está exento de desafíos y problemas que pueden impactar tanto en la experiencia del usuario como en la eficiencia operativa de las instituciones financieras. Estos desafíos abarcan desde cuestiones técnicas y de seguridad hasta factores relacionados con la satisfacción del cliente y la adaptación a un entorno tecnológico en constante evolución. En esta sección, exploraremos los desafíos y problemas comunes que enfrentan las instituciones financieras en el desarrollo y mantenimiento de aplicaciones móviles transaccionales.

Problemas de Seguridad y Privacidad

Uno de los desafíos más críticos en el desarrollo de aplicaciones móviles transaccionales es garantizar la seguridad y la privacidad de la información del usuario. Dado que estas aplicaciones manejan datos financieros sensibles, son un objetivo atractivo para los ciberdelincuentes. La autenticación y autorización adecuadas, así como la encriptación de datos, son esenciales para prevenir el acceso no autorizado y el robo de información financiera (Singh & Singh, 2021). Además,

las instituciones financieras deben estar preparadas para abordar amenazas emergentes como el phishing y el malware dirigido a dispositivos móviles.

Interoperabilidad y Plataformas Múltiples

El desarrollo de aplicaciones móviles transaccionales debe abordar la diversidad de plataformas y dispositivos móviles en el mercado. La necesidad de crear aplicaciones que sean compatibles con sistemas operativos como iOS y Android puede ser un desafío técnico y de recursos. La interoperabilidad entre diferentes sistemas y la optimización de la aplicación para múltiples plataformas requieren una planificación cuidadosa y pruebas exhaustivas para garantizar una experiencia coherente para los usuarios (Khalil et al., 2020).

Experiencia del Usuario

La experiencia del usuario juega un papel fundamental en la adopción y el éxito de las aplicaciones móviles transaccionales. Los usuarios esperan interfaces intuitivas, rápidas y sin complicaciones al realizar transacciones financieras. Los desafíos incluyen el diseño de interfaces de usuario amigables, la navegación sencilla y la respuesta rápida a las acciones del usuario. Cualquier dificultad en la usabilidad puede resultar en la pérdida de usuarios y una mala reputación para la institución financiera (Al-Gahtani et al., 2021).

Cambios en las Normativas y Regulaciones

Las aplicaciones móviles transaccionales están sujetas a regulaciones financieras y de privacidad en constante cambio. Las instituciones financieras deben asegurarse de que sus aplicaciones cumplan con las normativas vigentes, lo que puede requerir ajustes y actualizaciones frecuentes. La falta de cumplimiento puede llevar a sanciones financieras y dañar la reputación de la institución (Bashir et al., 2021).

Mantenimiento y Actualización Continua

El ciclo de vida de una aplicación móvil transaccional no se limita al desarrollo inicial, sino que implica un mantenimiento y actualización continuos. Las actualizaciones

pueden ser necesarias para corregir errores, agregar nuevas funciones o adaptarse a cambios en las plataformas móviles. Sin embargo, las actualizaciones también pueden causar interrupciones en el servicio y generar desafíos de compatibilidad con versiones anteriores (Wang et al., 2018).

Escalabilidad y Rendimiento

A medida que una aplicación móvil transaccional gana popularidad, debe ser capaz de manejar un aumento en el número de usuarios y transacciones simultáneas. Los desafíos de escalabilidad y rendimiento pueden surgir si la infraestructura subyacente no está preparada para manejar una carga creciente. Los tiempos de respuesta lentos o la indisponibilidad de la aplicación pueden afectar negativamente la experiencia del usuario y la confianza en la institución financiera (Huang et al., 2019).

Integración con Sistemas Externos

Las aplicaciones móviles transaccionales a menudo requieren la integración con sistemas internos y externos, como sistemas de gestión de cuentas, pasarelas de pago y servicios de terceros. La integración exitosa puede ser un desafío debido a diferencias en la estructura de datos, protocolos de comunicación y requisitos de seguridad. Los problemas de integración pueden causar errores en las transacciones y afectar la confiabilidad de la aplicación (Bisht et al., 2017).

Actualización de Tecnologías Emergentes

El rápido avance de la tecnología móvil y financiera significa que las instituciones deben adaptarse a las últimas tendencias y tecnologías emergentes, como la autenticación biométrica, la inteligencia artificial y la tecnología blockchain. La adopción de estas tecnologías puede ser un desafío en términos de capacitación del personal y actualización de la infraestructura tecnológica (Khan et al., 2019).

Por lo tanto y a manera de resumen, el desarrollo y mantenimiento de aplicaciones móviles transaccionales en instituciones financieras presentan una serie de desafíos y problemas que van desde cuestiones técnicas hasta preocupaciones relacionadas con la seguridad, la experiencia del usuario y el cumplimiento normativo. Abordar estos desafíos de manera efectiva requiere una planificación cuidadosa, inversión en recursos y una comprensión profunda de las necesidades y expectativas de los usuarios.

3. Desarrollo del proyecto

3.1 Análisis de la Situación

3.1.1 Descripción de la Cooperativa de Ahorro y Crédito Calceta Ltda.

La Cooperativa de Ahorro y Crédito Calceta Ltda. se formó en el año 1966 y obtuvo su reconocimiento legal el 19 de mayo de 1967, según el Acuerdo Ministerial No. 9022. En ese momento, el presidente constitucional interino de la República era el Dr. Otto Arosemena Gómez y el Ministro de Previsión y Asistencia Social era el Dr. Marcos Suescun Guerrero. En julio de 2003, la Cooperativa obtuvo el número SBS-2003-473, que calificaba a la institución como entidad de intermediación financiera, otorgada por la Superintendencia de Bancos y Seguros del Ecuador. Esta calificación proporcionó un respaldo adicional a los socios de la institución y demostró el progreso alcanzado en los últimos años. Hasta la fecha, ha funcionado con la confianza de más de 10,000 socios.

En el año 2006, la Cooperativa decidió abrir su primera agencia en Manta, después de realizar los estudios correspondientes. La Superintendencia de Bancos autorizó la apertura y el 9 de enero de 2007 se inauguró la Agencia Manta de la Cooperativa de Ahorro y Crédito Calceta Ltda. ubicada en la Avenida 16 entre calles 12 y 13. Desde entonces, la Cooperativa ha expandido sus operaciones a otros cantones de la provincia de Manabí, como Sucre, Jama, Portoviejo, San Isidro, Bahía, Pedro Carbo, Pedernales, y ha establecido oficinas especiales en Chone, Junín, Canuto, Membrillo y Convento. Con el propósito de brindar mayor cobertura, la Cooperativa de Ahorro y Crédito Calceta Ltda. ha contribuido al desarrollo del Cantón Bolívar y sigue consolidando su presencia en la industria financiera.

Ilustración 5: Edificio Matriz Cooperativa

Misión, visión y valores corporativos

La misión de la Cooperativa de Ahorro y Crédito Calceta Ltda. es contribuir al desarrollo socioeconómico de los microempresarios y la población a nivel nacional, generando confianza a través de productos y servicios financieros oportunos, sostenibles y de calidad, con enfoque de responsabilidad social.

La visión de COAC CALCETA es convertirse en una institución líder en la provisión de productos y servicios financieros de calidad, competitivos y rentables, con una administración y equipo de trabajo eficientes. Su objetivo es impulsar el crecimiento socioeconómico de los sectores productivos del país.

Los valores corporativos de COAC CALCETA son:

- Honestidad: Actuar con integridad, ética y transparencia en todas las acciones y relacionamientos.
- Transparencia: Brindar información clara y veraz a los socios y clientes, fomentando la confianza y el respeto mutuo.
- Responsabilidad: Cumplir con los compromisos asumidos y actuar de manera responsable en el manejo de los recursos financieros y humanos.

- Equidad: Tratar a todos los socios y clientes de manera justa, sin discriminación ni preferencias indebidas.
- Solidaridad: Fomentar el espíritu de colaboración y apoyo mutuo entre los socios y la comunidad.
- Respeto: Valorar y aceptar la diversidad de opiniones, culturas y creencias, promoviendo un ambiente de respeto y tolerancia.

Esos son los elementos fundamentales de la misión, visión y valores corporativos de la Cooperativa COAC CALCETA.

Servicios ofrecidos.

La Cooperativa de Ahorro y Crédito Calceta Ltda. ofrece una variedad de servicios financieros diseñados para satisfacer las necesidades de sus socios. A continuación, se detallan algunos de los servicios que ofrece la Cooperativa:

- Ahorros: La Cooperativa ofrece diferentes tipos de cuentas de ahorro, como la Cuenta Alcancía y el Ahorro del Socio. Estas cuentas permiten a los socios guardar su dinero de manera segura y ganar intereses sobre sus ahorros.
- Créditos: La Cooperativa proporciona préstamos a sus socios para diferentes propósitos, como compra de vivienda, educación, vehículos, entre otros. Estos préstamos se ofrecen con tasas de interés competitivas y plazos flexibles.
- Cajeros Automáticos: La cooperativa cuenta con cajeros automáticos donde los socios pueden realizar diversas transacciones, como retiros de efectivo, consultas de saldo y pagos de servicios básicos.
- Transferencias de Fondos: La cooperativa facilita transferencias internas entre cuentas de sus socios, así como transferencias interbancarias a otros bancos. Esto permite a los socios mover su dinero de manera rápida y segura.
- Remesas del Exterior: La cooperativa ofrece servicios de recepción y envío de remesas del exterior. Esto brinda a los socios una forma conveniente de recibir dinero de sus seres queridos que se encuentran en el extranjero.

- Seguros de Vida y Salud: La cooperativa ofrece opciones de seguros de vida y salud para sus socios. Estos seguros brindan protección financiera en caso de eventos imprevistos, como enfermedades graves o fallecimiento.
- Puntomático: A través de este servicio, los socios pueden realizar pagos de planillas de servicios básicos, impuestos, matriculación vehicular y tarjetas de crédito, entre otros.

Productos

La Cooperativa de Ahorro y Crédito Calceta Ltda. ofrece una variedad de productos financieros diseñados para satisfacer las necesidades de sus clientes. A continuación, se brinda una descripción de los productos disponibles:

- Ahorro del Socio: Este producto brinda a los socios la oportunidad de ahorrar dinero de forma segura y eficiente. Los socios pueden depositar fondos en sus cuentas de ahorro y obtener intereses competitivos sobre sus saldos.
- Plan Futuro: El Plan Futuro de la cooperativa permite a los clientes planificar y ahorrar para metas y objetivos a largo plazo. Proporciona información y ofrece opciones para hacer crecer sus ahorros y asegurar un futuro financiero estable.
- Cuenta Alcancía: Esta cuenta está diseñada específicamente para menores de edad. Ayuda a los jóvenes a aprender sobre el valor del ahorro y a desarrollar hábitos financieros saludables desde una edad temprana. Los padres o tutores pueden abrir una cuenta alcancía para sus hijos y enseñarles la importancia de guardar dinero.
- Plazo Fijo: Los plazos fijos son una opción de inversión en la cual el cliente deposita una cantidad de dinero por un período determinado y recibe intereses sobre esa cantidad. La cooperativa ofrece tasas competitivas para los plazos fijos, brindando a los clientes la oportunidad de hacer crecer sus ahorros de manera segura.

Otro de los productos es el crédito de consumo ofrecido por la Cooperativa de Ahorro y Crédito Calceta Ltda. Está dirigido a personas naturales o jurídicas que se encuentren dentro del radio de acción de la cooperativa en la provincia de Manabí y a nivel nacional. El crédito de consumo de la cooperativa está diseñado para cubrir las necesidades financieras de los clientes, ya sea para adquirir bienes o servicios personales, realizar pagos de deudas, realizar inversiones, entre otras finalidades. Además, la cooperativa también ofrece otros tipos de créditos, como créditos de vivienda, dirigidos a la adquisición, construcción, reparación, remodelación y mejoramiento de vivienda propia, y microcréditos, dirigidos a personas naturales o jurídicas dentro del radio de acción de la cooperativa en la provincia de Manabí y a nivel nacional. En el sitio web de la cooperativa, tiene un simulador que te permitirá crear una simulación del crédito.

Transparencia y datos financieros.

La Cooperativa de Ahorro y Crédito Calceta Ltda. mantiene un compromiso con la transparencia y la divulgación de datos financieros. A través de su sitio web, proporciona acceso a diversos documentos relacionados con la información financiera y el desempeño de la cooperativa. En cuanto a la transparencia, la cooperativa ha publicado una serie de documentos en su sitio web que proporcionan una visión clara de su situación financiera. Estos documentos incluyen:

- Patrimonio Técnico: La cooperativa ha publicado información sobre el patrimonio técnico en diferentes momentos, como agosto de 2023 y diciembre de 2022. Estos documentos brindan detalles sobre los activos, pasivos y el capital de la cooperativa.
- Tasas: También se ha publicado información sobre las tasas en diferentes períodos, como septiembre de 2023. Estos documentos informan sobre las tasas de interés aplicables a los diferentes productos financieros ofrecidos por la cooperativa.
- Balances y Estados de Resultados: La cooperativa ha compartido información sobre el balance general y el estado de resultados en diferentes

momentos, como diciembre de 2022 y agosto de 2023. Estos documentos brindan una visión detallada de los ingresos, gastos y ganancias o pérdidas de la cooperativa.

- Indicadores de Género: Además, se ha dado a conocer información sobre los indicadores de género en diciembre de 2022.

La Cooperativa de Ahorro y Crédito Calceta Ltda. muestra su compromiso con la transparencia al publicar estos documentos financieros en su sitio web. Esta divulgación de información permite a los socios y al público en general tener acceso a datos relevantes sobre la situación financiera de la cooperativa.

3.1.2 Descripción General del Back-end Actual.

El back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. se ha desarrollado utilizando una arquitectura de capas con tecnologías basadas en Visual C# y .NET Framework. Esta parte fundamental de la aplicación desempeña un papel crítico en la autenticación de usuarios, la consulta de cuentas y movimientos, estadísticas financieras, gestión de créditos, transferencias internas e interbancarias, así como en el pago de servicios de agua potable en la localidad de Calceta y el envío de correos electrónicos. Además, brinda información sobre sucursales y cajeros disponibles para los usuarios. El back-end se encuentra respaldado por una base de datos Oracle.

Cronograma de Actividades

El back-end actual está diseñado siguiendo una arquitectura de capas. Esta estructura se divide en capas lógicas que permiten una separación clara de las responsabilidades y facilitan la modularización del código. Esta arquitectura es comúnmente utilizada en el desarrollo de aplicaciones empresariales y proporciona una mayor organización y mantenibilidad del sistema.

Tecnologías Utilizadas: .Net Framework, Oracle

- .Net Framework: El lenguaje de programación principal utilizado para el desarrollo del back-end es Visual C# en el entorno .NET Framework de Microsoft. Esta elección tecnológica ofrece un conjunto de herramientas sólidas para el desarrollo de aplicaciones robustas y escalables en el entorno Windows.
- Oracle: La base de datos subyacente del back-end está basada en Oracle. Oracle es una opción popular para aplicaciones empresariales que requieren un alto rendimiento y confiabilidad en el almacenamiento y gestión de datos.

Funcionalidades Principales:

- Autenticación de Usuarios: El back-end permite la autenticación segura de usuarios a través de credenciales, gestionando el acceso a las funcionalidades de la aplicación móvil.
- Consulta de Cuentas y Movimientos: Los usuarios pueden acceder a información detallada sobre sus cuentas y los movimientos financieros asociados a ellas.
- Gestión de Créditos: Se brinda la capacidad de consultar detalles sobre los créditos otorgados a los clientes de la cooperativa, incluyendo información sobre plazos y condiciones.
- Transferencias Internas e Interbancarias: Los usuarios pueden realizar transferencias de fondos entre cuentas de la cooperativa y hacia otras instituciones bancarias.
- Pago de Servicio de Agua Potable: Se ofrece la funcionalidad de pago de servicios básicos, específicamente el servicio de agua potable en la localidad de Calceta.
- Envío de Correos Electrónicos: El back-end permite la notificación por correo electrónico de diversas transacciones y eventos, mejorando la comunicación con los usuarios.
- Consulta de Sucursales y Cajeros: Los usuarios pueden obtener información sobre la ubicación de las sucursales y cajeros automáticos de la cooperativa.

Problemas Identificados y Áreas de Mejora:

En el análisis previo del back-end actual, se han identificado varios problemas y áreas de mejora que requieren atención:

Códigos referentes al back-end actual

```
public static List<AD_Response> ProcesaTransferencia(AD_Transferencias transferencias, AD_Dispositivo _Dispositivo)
{
   AD_Login _Login = new AD_Login
   {
      Username = _Dispositivo.Cl
   };
   List<AD_Response> Lista = new List<AD_Response>();
   transferencias.Motivo = (transferencias.Motivo.Length > 200) ? transferencias.Motivo.ToUpper().Substring(0, 200) :
transferencias.Motivo.ToUpper();
   try
   {
      if (transferencias.Valor >= 1.0)
      {
         foreach (DataRow lista in N_Cuentas.ValidacuentaActiva(Convert.ToInt32(transferencias.CtaCredito)).Rows)
         {
            if (Convert.ToInt32(lista[0].ToString()) == 2)
            {
               if (ProcesaTvauxDebito(transferencias) == true)
               {
                  if (ProcesaSecuenciaDebito() == true)
                  {
                     if (ProcesaDebito(transferencias) == true)
                     {
                        if (ProcesaTvauxCredito(transferencias) == true)
                        {
                           if (ProcesaCredito(transferencias) == true)
                           {
                              VGMensaje = VGNumtxn.ToString();
                              codigo = "200";
                              bool procesa = ProcesarMotivo(Convert.ToInt32(VGMensaje), transferencias.Motivo);
                              N_Login.RegistrarUbicacion(_Login, _Dispositivo);
                              N_EnviarCorreo.EnviaCorreo(_Dispositivo.Cl, AD_Constantes.AsuntoTransferencia, MensajeTransferencia(transferencias,
VGMensaje), AD_Constantes.UsuarioCorreo, AD_Constantes.NombreMostrar);
                           }
                           else
                           {
                              codigo = "500";
                              VGMensaje = "Error al procesar transacción";
                           }
                        }
                        else
                        {
                           codigo = "500";
                           VGMensaje = "Error al procesar transacción";
                        }
                     }
                     else
                     {
                        codigo = "500";
                        VGMensaje = "Error al procesar transacción";
                     }
                  }
                  else
                  {
                     codigo = "500";
                     VGMensaje = "Error al procesar transacción";
                  }
               }
               else
               {
```

```
                codigo = "500";
                VGMensaje = "Error al procesar transacción";
              }
            }
            else
            {
              codigo = "500";
              VGMensaje = "Cuenta de destino no está activa";
            }
          }
        }
        else
        {
          codigo = "500";
          VGMensaje = "Valor mínimo para transferir es $1";
        }
      }
      catch (Exception)
      {
        codigo = "500";
        VGMensaje = $"Error al procesar transacción";
      }
      AD_Response pdocumento = new AD_Response
      {
        Codigo = codigo,
        Mensaje = VGMensaje
      };
      Lista.Add(pdocumento);
      return Lista;
    }
```

```
private static int contadorCaracteres;
private static int restaCaracteres;
private static string mostrarX;
private static string resultado;
private static string contiene;
private static string fincuenta;

public static string RemplazoCuenta(string cuenta)
{
  contadorCaracteres = 0;
  restaCaracteres = 0;
  mostrarX = "";
  resultado = "";
  contadorCaracteres = cuenta.ToString().Length;
  if (contadorCaracteres > 5)
  {
    contiene = cuenta.ToString().Substring(0, 2);
    fincuenta = cuenta.ToString().Substring(contadorCaracteres - 2, (cuenta.Length - (contadorCaracteres - 2)));
    restaCaracteres = contadorCaracteres - (contiene.Length) - (fincuenta.Length);
    for (var x = 1; x <= restaCaracteres; x++)
    {
      mostrarX = "X" + mostrarX;
    }
  }
  if (contadorCaracteres <= 5)
  {
    contiene = cuenta.ToString().Substring(0, 1);
    fincuenta = cuenta.ToString().Substring(contadorCaracteres - 1, (cuenta.Length - (contadorCaracteres - 1)));
    restaCaracteres = contadorCaracteres - (contiene.Length) - (fincuenta.Length);
    for (var x = 1; x <= restaCaracteres; x++)
    {
      mostrarX = "X" + mostrarX;
    }
  }
  resultado = contiene + mostrarX + fincuenta;
  return resultado;
}
```

- Código Espagueti: El código presenta una estructura compleja con múltiples niveles de anidación (if-else) que dificultan la comprensión y mantenimiento del mismo.
- Duplicidad de Métodos o Funciones: Se ha identificado la existencia de funciones duplicadas, lo que puede llevar a un mantenimiento ineficiente y a problemas de consistencia.
- Nomenclatura y Legibilidad del Código: La nomenclatura de variables y la legibilidad del código necesitan mejoras para facilitar la comprensión y el mantenimiento.
- Falta de Comentarios en el Código: La ausencia de comentarios en el código dificulta la comprensión de la lógica y el propósito de las funciones y métodos.

Impacto en la Eficiencia y Mantenibilidad del Sistema

Estos problemas identificados tienen un impacto significativo en la eficiencia y mantenibilidad del sistema actual. La complejidad del código espagueti dificulta la implementación de nuevas características y correcciones de errores de manera eficiente. La duplicación de funciones aumenta la probabilidad de errores y dificulta la coherencia del sistema. La nomenclatura poco descriptiva y la falta de comentarios aumentan la curva de aprendizaje para los desarrolladores y pueden provocar retrasos en el desarrollo y mantenimiento.

La descripción general del back-end actual de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. revela una serie de desafíos técnicos y áreas de mejora que deben abordarse en el proceso de refactorización propuesto en esta tesis de Maestría en Software. La aplicación de principios DRY (Don't Repeat Yourself) y SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) se presenta como una estrategia clave para abordar estos problemas y mejorar la eficiencia y mantenibilidad del sistema.

3.1.3 Identificación de Problemas y Áreas de Mejora.

La identificación de problemas y áreas de mejora en el back-end actual de la aplicación móvil transaccional es fundamental para guiar el proceso de refactorización. A continuación, se detallan los problemas identificados en tres categorías principales: código espagueti, duplicidad de métodos o funciones, y nomenclatura y legibilidad del código.

Código Espagueti

El código espagueti es una característica destacada en el back-end actual, lo que significa que la estructura del código carece de organización y claridad, lo que dificulta su comprensión y mantenimiento. Algunos de los problemas específicos identificados son los siguientes:

- Anidación Excesiva: El código contiene múltiples niveles de anidación de instrucciones if-else, lo que crea una estructura confusa y dificulta el seguimiento de la lógica de ejecución.
- Repetición de Código: Existe repetición de código en varias partes del back-end, lo que conduce a la duplicación innecesaria de esfuerzos y aumenta la probabilidad de errores.
- Variables Poco Descriptivas: Algunas variables utilizadas en el código tienen nombres poco descriptivos, lo que hace difícil entender su propósito y función en el contexto de las operaciones realizadas.
- Ausencia de Comentarios: El código no incluye comentarios que expliquen la lógica detrás de las operaciones realizadas, lo que dificulta aún más la comprensión y el mantenimiento.

Duplicidad de Métodos o Funciones

La duplicidad de métodos o funciones se refiere a la existencia de múltiples funciones que realizan tareas similares o idénticas en diferentes partes del código.

Este problema puede tener un impacto negativo en la eficiencia y mantenibilidad del sistema. Algunos ejemplos de duplicidad identificados son:

- Métodos de Envío de Correos Electrónicos: Se ha identificado que existen varios métodos encargados del envío de correos electrónicos para diferentes eventos y notificaciones, lo que indica una falta de reutilización de código.
- Procesamiento de Transacciones: Las operaciones de procesamiento de transacciones, como débitos y créditos, se realizan en varios lugares del código, lo que aumenta la complejidad y dificulta la coherencia de las transacciones.

Nomenclatura y Legibilidad del Código

La nomenclatura y la legibilidad del código son aspectos críticos para facilitar la comprensión y el mantenimiento del sistema. En el back-end actual, se han identificado problemas relacionados con la nomenclatura y la legibilidad, que incluyen:

- Nombres de Variables Poco Descriptivos: Algunas variables tienen nombres que no reflejan claramente su propósito, lo que dificulta la comprensión del código.
- Falta de Convenciones de Nomenclatura: La aplicación de convenciones de nomenclatura consistentes es inconsistente en todo el código, lo que afecta negativamente la claridad y cohesión del mismo.
- Escasa Legibilidad: La falta de espaciado y formato adecuados dificulta la lectura del código, lo que puede llevar a errores y dificultades en el mantenimiento.
- Ausencia de Comentarios Significativos: La falta de comentarios descriptivos y significativos dificulta la comprensión de la lógica detrás de las funciones y métodos.

La identificación de estos problemas proporciona una base sólida para la propuesta de refactorización, que se centrará en la aplicación de principios DRY y SOLID para abordar estos desafíos y mejorar la eficiencia y mantenibilidad del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda.

3.1.4 Impacto de los Problemas Identificados en la Eficiencia y Mantenibilidad del Sistema.

Los problemas identificados en el back-end actual de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. tienen un impacto significativo en la eficiencia y mantenibilidad del sistema. A continuación, se describen los posibles impactos de estos problemas:

Impacto en la Eficiencia del Sistema:

- Rendimiento Subóptimo: La presencia de código espagueti y la duplicidad de métodos pueden ralentizar el rendimiento general del sistema. La anidación excesiva y la repetición de código pueden llevar a una ejecución ineficiente de las transacciones, lo que resulta en tiempos de respuesta más largos para los usuarios.
- Consumo de Recursos: La falta de optimización y la duplicidad de código pueden aumentar el consumo de recursos del servidor, lo que puede tener un impacto negativo en la escalabilidad del sistema. Esto significa que el sistema podría no ser capaz de manejar un mayor número de transacciones o usuarios concurrentes de manera eficiente.

Impacto en la Mantenibilidad del Sistema

- Dificultad en las Actualizaciones: La falta de organización y claridad en el código hace que sea difícil realizar actualizaciones o introducir nuevas características en el sistema. Los desarrolladores pueden tener dificultades

para comprender el funcionamiento del código existente y cómo se relaciona con las nuevas funcionalidades.

- Mayor Propensión a Errores: La duplicidad de métodos y funciones aumenta la probabilidad de errores en el sistema. Cuando se realizan cambios en una parte del código duplicado, es fácil olvidar actualizar todas las instancias duplicadas, lo que puede dar lugar a comportamientos inesperados y errores.
- Dificultad en la Depuración: La falta de comentarios significativos y nombres de variables poco descriptivos dificulta la depuración del código. Los desarrolladores pueden tener dificultades para identificar y corregir problemas en el sistema, lo que aumenta el tiempo dedicado a la resolución de errores.
- Costos de Mantenimiento Elevados: La complejidad y falta de estructura del código pueden aumentar los costos de mantenimiento a lo largo del tiempo. Los equipos de desarrollo pueden requerir más tiempo y recursos para realizar tareas de mantenimiento y corrección de errores.

Por lo tanto, los problemas identificados en el back-end actual de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. tienen un impacto adverso tanto en la eficiencia operativa como en la facilidad de mantenimiento del sistema. Estos problemas pueden dar lugar a una experiencia de usuario deficiente, dificultades en el desarrollo continuo y un aumento en los costos de mantenimiento. La propuesta de refactorización, aplicando los principios DRY (Don't Repeat Yourself) y SOLID, busca abordar estos problemas y mejorar significativamente la eficiencia y mantenibilidad del sistema.

3.2 Metodología.

Este capítulo se enfoca en presentar la metodología y el enfoque que se seguirán en el desarrollo de la propuesta de refactorización del back-end de la aplicación móvil transaccional para la Cooperativa de Ahorro y Crédito Calceta Ltda. El objetivo

principal de este capítulo es proporcionar una guía detallada de cómo se abordará el proceso de mejora del software existente.

3.2.1 Definición de Objetivos.

Los objetivos específicos que se buscan alcanzar con la propuesta de refactorización del back-end son los siguientes:

- Optimización del Código: El objetivo principal es optimizar el código fuente existente del back-end para eliminar problemas de código espagueti, mejorar la legibilidad y facilitar su mantenimiento.
- Reducción de la Duplicidad: Se busca eliminar la duplicidad de métodos o funciones, lo que permitirá una gestión más eficiente del código y reducirá la probabilidad de errores.
- Mejora de la Nomenclatura y Legibilidad: El objetivo es mejorar la nomenclatura de variables, funciones y clases, así como la legibilidad del código, para que sea más comprensible y siga buenas prácticas de desarrollo.
- Cumplimiento de Principios DRY y SOLID: Asegurar que el código refactorizado cumpla con los principios DRY (Don't Repeat Yourself) y SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) para lograr un diseño más robusto y flexible.
- Mantenibilidad a Largo Plazo: Garantizar que el código refactorizado sea más fácil de mantener y que permita futuras actualizaciones y mejoras de manera eficiente.
- Minimizar Impactos en la Funcionalidad Actual: Asegurar que la refactorización no afecte negativamente la funcionalidad actual de la aplicación móvil transaccional.

3.2.2 Selección de Metodología.

La metodología seleccionada para llevar a cabo la refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. se basa en un enfoque iterativo e incremental. Esta elección se justifica por varias razones:

- Adaptabilidad: Un enfoque iterativo permite adaptarse a los cambios y ajustes que puedan surgir a medida que avanza el proyecto, lo que es especialmente relevante en proyectos de refactorización de código existente.
- Evaluación Continua: El proceso se someterá a evaluaciones regulares para garantizar que los objetivos específicos se estén cumpliendo y que la calidad del código esté mejorando de acuerdo con los principios DRY y SOLID.
- Minimización de Riesgos: La iteración constante permite identificar y abordar posibles problemas a medida que surgen, lo que minimiza los riesgos asociados con cambios significativos en el código.
- Entrega Gradual: Se planifica la entrega gradual de mejoras en el back-end, lo que permite que la Cooperativa de Ahorro y Crédito Calceta Ltda. continúe utilizando y beneficiándose de la aplicación mientras se realiza la refactorización.
- Enfoque en Calidad: La metodología iterativa se centra en la mejora continua de la calidad del código, lo que es fundamental para cumplir con los objetivos establecidos.

La elección de esta metodología se alinea con la naturaleza de la propuesta y con el objetivo de lograr una mejora gradual y sostenible en el back-end de la aplicación móvil transaccional. El siguiente paso será detallar la planificación del proyecto de refactorización en el próximo apartado.

3.2.3 Planificación del Proyecto de Refactorización.

La planificación del proyecto de refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. se llevará a cabo de manera detallada y organizada. La ejecución de esta refactorización se dividirá en las siguientes etapas:

Evaluación Inicial

En esta etapa inicial, se realizará una evaluación exhaustiva del código fuente existente del back-end. Se identificarán y documentarán los problemas de código espagueti, duplicidad de métodos o funciones, y problemas de nomenclatura y legibilidad. Esto servirá como punto de partida para la refactorización.

Diseño de la Solución

Una vez que se haya completado la evaluación inicial, se procederá al diseño de la solución. Se definirá una arquitectura de software que cumpla con los principios DRY y SOLID, así como con los objetivos específicos de la refactorización. Se planificará la estructura del nuevo código y se definirán las responsabilidades de cada componente.

Implementación Gradual

La refactorización se llevará a cabo de manera gradual para minimizar los riesgos y mantener la funcionalidad actual de la aplicación. En esta etapa, se reescribirán secciones de código problemáticas siguiendo las pautas de diseño definidas en la etapa anterior. Cada parte refactorizada se someterá a pruebas exhaustivas para garantizar que la funcionalidad no se vea afectada.

Pruebas y Validación

Después de cada fase de implementación, se realizarán pruebas exhaustivas para validar que el código refactorizado cumple con los requisitos funcionales y de

calidad. Esto incluirá pruebas de unidad, pruebas de integración y pruebas de sistema. Se corregirán y optimizarán las áreas que no cumplan con los estándares de calidad.

Documentación y Comentarios

Se generará documentación detallada del nuevo código, incluyendo comentarios explicativos que mejoren la legibilidad. Esto facilitará el mantenimiento futuro y ayudará a los desarrolladores a comprender el funcionamiento del código.

Entrega Gradual y Capacitación

A medida que se completen las fases de refactorización, se entregarán las mejoras de manera gradual a la Cooperativa de Ahorro y Crédito Calceta Ltda. Se brindará capacitación a los equipos de desarrollo y operaciones para garantizar que puedan trabajar eficazmente con el nuevo código.

Evaluación Final y Ajustes

Al finalizar la refactorización, se llevará a cabo una evaluación final para asegurarse de que se hayan cumplido todos los objetivos y principios establecidos. Si es necesario, se realizarán ajustes finales para garantizar que el sistema funcione de manera óptima.

Cronograma de Actividades

La planificación del proyecto se llevará a cabo en un período de 6 semanas, dividido en las etapas mencionadas anteriormente. A continuación, se presenta un cronograma tentativo de actividades:

Tabla 1: Cronograma de actividades

Semana	Actividades
Semana 1	- Evaluación inicial del código existente. - Reunión de inicio del proyecto. - Definición de objetivos de refactorización.

Semana 2	- Diseño de la nueva arquitectura del back-end. - Planificación detallada de la refactorización.
Semana 3	- Implementación gradual de las primeras secciones de código. - Inicio de pruebas de unidad.
Semana 4	- Continuación de la implementación gradual. - Pruebas de integración de las secciones refactorizadas.
Semana 5	- Implementación de secciones adicionales. - Pruebas de sistema para áreas completadas.
Semana 6	- Finalización de la implementación. - Pruebas finales y validación. - Entrega gradual de mejoras. - Evaluación final y ajustes.

Asignación de Recursos

Para llevar a cabo esta refactorización, se asignarán los siguientes recursos:

- Equipo de Desarrollo: El equipo estará compuesto por 1 desarrollador con experiencia en la tecnología utilizada en el back-end.
- Recursos de Pruebas: Se designarán recursos para realizar pruebas exhaustivas durante todo el proceso de refactorización.
- Gestor de Proyecto: Un gestor de proyecto supervisará la planificación y ejecución del proyecto, asegurando que se cumplan los plazos y objetivos.

3.2.4 Análisis de Requerimientos.

El análisis de requerimientos para la refactorización se llevará a cabo de manera colaborativa con los stakeholders relevantes. Se seguirán los siguientes pasos:

Identificación de Stakeholders

Se identificarán todas las partes interesadas en el proyecto, incluyendo representantes de la Cooperativa de Ahorro y Crédito Calceta Ltda., usuarios finales y desarrolladores actuales.

Comunicación con Stakeholders

Se establecerán canales de comunicación efectivos con los stakeholders para recopilar sus necesidades y expectativas con respecto a la refactorización del back-end. Esto incluirá reuniones regulares, encuestas y entrevistas.

Definición de Requerimientos Funcionales y No Funcionales

A partir de la retroalimentación de los stakeholders, se definirán los requerimientos funcionales y no funcionales que guiarán la refactorización. Estos requerimientos incluirán la funcionalidad esencial de la aplicación, los estándares de seguridad, rendimiento y calidad del código.

Validación de Requerimientos

Los requerimientos definidos se someterán a validación por parte de los stakeholders para garantizar que reflejen con precisión sus necesidades y expectativas. Cualquier ajuste necesario se realizará en esta etapa. Este proceso de análisis de requerimientos asegurará que la refactorización se realice de acuerdo con las necesidades del negocio y los usuarios finales, contribuyendo a un resultado exitoso y alineado con los objetivos del proyecto.

3.2.5 Implementación y Pruebas.

En esta sección, se detallará cómo se llevará a cabo la implementación de la solución propuesta y las estrategias de pruebas que se aplicarán para garantizar la calidad y funcionalidad del nuevo back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda.

Implementación de la Solución

La implementación de la solución de refactorización del back-end se llevará a cabo siguiendo las siguientes etapas:

- Diseño e Implementación Gradual: En esta fase, el equipo de desarrollo diseñará la nueva arquitectura del back-end, tomando en cuenta los principios DRY (Don't Repeat Yourself) y SOLID (Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle). A continuación, se implementarán gradualmente las secciones de código refacturadas de acuerdo con el diseño propuesto.
- Pruebas de Unidad: Cada sección refactorizada será sometida a pruebas de unidad exhaustivas para verificar su funcionamiento individual. Se utilizarán herramientas de pruebas unitarias como NUnit o MSTest para evaluar la lógica de las funciones y métodos.
- Pruebas de Integración: A medida que se completen las secciones refactorizadas, se llevarán a cabo pruebas de integración para garantizar que los diferentes componentes del back-end funcionen correctamente juntos. Se verificará que la comunicación entre módulos y la transferencia de datos sean coherentes.

Estrategias de Pruebas

Para asegurar la calidad de la refactorización, se aplicarán las siguientes estrategias de pruebas:

- Pruebas Funcionales: Se realizarán pruebas funcionales exhaustivas para verificar que todas las funciones y características de la aplicación móvil transaccional sigan funcionando como se espera después de la refactorización. Se desarrollarán casos de prueba que cubran todas las funcionalidades clave, como el inicio de sesión, consultas de cuentas, transferencias, envío de correo electrónico, entre otras.
- Pruebas de Rendimiento: Se llevarán a cabo pruebas de rendimiento para evaluar la capacidad de respuesta del back-end frente a cargas de trabajo

típicas y picos de uso. Esto garantizará que la aplicación móvil transaccional pueda manejar la demanda de manera eficiente.

- Pruebas de Seguridad: Se realizarán pruebas de seguridad para identificar y mitigar posibles vulnerabilidades en el nuevo back-end. Se evaluarán las medidas de seguridad, como la autenticación y la autorización de usuarios, y se aplicarán pruebas de penetración si es necesario.
- Pruebas de Usabilidad: Se llevarán a cabo pruebas de usabilidad con usuarios reales o simulados para evaluar la experiencia general del usuario en la aplicación móvil transaccional. Se recopilarán comentarios sobre la interfaz de usuario y la facilidad de uso.

Herramientas de Pruebas

Las siguientes herramientas serán utilizadas durante el proceso de pruebas:

- NUnit o MSTest: Para pruebas unitarias.
- Selenium: Para pruebas de automatización de pruebas funcionales.
- JMeter: Para pruebas de rendimiento y carga.
- Herramientas de seguridad: Se utilizarán herramientas específicas de seguridad, como Nessus o Burp Suite, para pruebas de seguridad.
- Encuestas y cuestionarios: Para recopilar retroalimentación de usuarios en las pruebas de usabilidad.

La implementación y las pruebas serán un proceso continuo e iterativo para asegurar que el nuevo back-end cumpla con los estándares de calidad y satisfaga las necesidades de la Cooperativa de Ahorro y Crédito Calceta Ltda.

3.2.6 Documentación y Entrega.

En esta sección, se explicará cómo se abordará la documentación técnica del proyecto de refactorización y cómo se llevará a cabo la entrega de la nueva versión del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda.

Documentación Técnica

La documentación técnica es esencial para garantizar que los futuros desarrolladores y miembros del equipo comprendan la estructura, el funcionamiento y las mejores prácticas del nuevo back-end. La documentación incluirá:

Documentación de Diseño: Se proporcionará documentación detallada que describe la nueva arquitectura y diseño del back-end. Esto incluirá diagramas de flujo, diagramas de clases y cualquier otra representación visual necesaria para comprender la estructura del sistema.

- Documentación de Código: Se generará documentación de código que incluye comentarios descriptivos en el código fuente. Estos comentarios explicarán la funcionalidad de las clases, métodos y funciones, así como cualquier decisión de diseño relevante.
- Manual de Uso: Se creará un manual de uso que proporcionará instrucciones detalladas sobre cómo utilizar el nuevo back-end. Esto será útil para los miembros del equipo que interactúan con el sistema y necesitan comprender su funcionamiento.
- Documentación de Pruebas: Se registrarán todos los casos de prueba utilizados durante el proceso de pruebas, así como los resultados obtenidos. Esto permitirá una fácil validación y verificación de la funcionalidad del sistema.

Entregables Finales

Los entregables finales del proyecto de refactorización incluirán:

- Código Fuente Refactorizado: Se entregará el código fuente refactorizado del nuevo back-end de la aplicación móvil transaccional. Este código estará limpio, siguiendo los principios DRY y SOLID, y documentado adecuadamente.

- Documentación Técnica Completa: Se proporcionará toda la documentación técnica mencionada anteriormente, incluyendo documentación de diseño, documentación de código, manual de uso y documentación de pruebas.
- Plan de Entrega: Se presentará un plan de entrega detallado que establecerá las fechas y los procedimientos para la entrega de los entregables finales. Esto incluirá una lista de verificación para garantizar que todos los elementos estén completos.

Procedimientos de Entrega y Puesta en Producción

El proceso de entrega y puesta en producción del nuevo back-end seguirá los siguientes procedimientos:

- Validación Interna: Antes de la entrega, el equipo de desarrollo realizará una validación interna para garantizar que todos los componentes funcionen correctamente y cumplan con los requisitos.
- Pruebas de Usuario: Se invitará a los usuarios clave y a los stakeholders a participar en pruebas de usuario para validar que el nuevo back-end cumple con sus expectativas y necesidades.
- Capacitación: Se proporcionará capacitación a los miembros del equipo de la Cooperativa de Ahorro y Crédito Calceta Ltda. sobre cómo utilizar el nuevo back-end de manera efectiva.
- Plan de Puesta en Producción: Se desarrollará un plan detallado para la puesta en producción, que incluirá la migración de datos, la configuración del entorno de producción y las pruebas finales antes de la implementación en vivo.
- Implementación en Producción: Una vez que todas las pruebas sean exitosas y se haya completado la capacitación, se procederá a implementar el nuevo back-end en el entorno de producción de la Cooperativa de Ahorro y Crédito Calceta Ltda.

La documentación y entrega serán gestionadas de manera meticulosa para asegurar una transición suave hacia el nuevo back-end y un funcionamiento sin problemas de la aplicación móvil transaccional.

3.2.7 Gestión de Riesgos.

En esta sección, se llevará a cabo la identificación, análisis y gestión de los posibles riesgos asociados con el proyecto de refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. La gestión de riesgos es esencial para prevenir posibles obstáculos que puedan afectar el éxito del proyecto y para estar preparados para enfrentar situaciones adversas. En la siguiente tabla se presentan los riesgos y las estrategias de mitigación:

Tabla 2: Tabla de Riesgos, estrategias de mitigación y contingencia

Riesgo	Impacto Potencial	Probabilidad de Ocurrencia	Nivel de Riesgo	Estrategias de Mitigación	Estrategias de Contingencia
Cambios en los Requisitos	Moderado	Media	Moderado	- Comunicación continua con stakeholders. - Documentación adecuada.	- Asignación de recursos adicionales. - Ajuste del cronograma ante cambios significativos.
Problemas Técnicos Inesperados	Alto	Baja	Moderado	- Pruebas exhaustivas durante la implementación. - Asignación de recursos técnicos.	- Establecimiento de un plan de recuperación que incluye reversión a la versión anterior.
Resistencia al Cambio	Moderado	Alta	Alto	- Estrategia de gestión del cambio. - Capacitación y comunicación efectiva.	- Planes de comunicación adicionales. - Colaboración activa de los usuarios.
Falta de Recursos	Alto	Media	Alto	- Asignación adecuada de	- Búsqueda de fuentes

				recursos desde el inicio. - Seguimiento constante de los recursos disponibles.	alternativas de recursos. - Ajuste del alcance del proyecto si es necesario.
Problemas de Comunicación	Moderado	Media	Moderado	- Establecimiento de canales de comunicación claros. - Fomento de retroalimentación constante.	- Designación de un responsable de comunicación.

Identificación de Riesgos

A continuación, se enumeran algunos de los posibles riesgos identificados para el proyecto de refactorización:

- Cambios en los Requisitos: Los requisitos del proyecto pueden cambiar durante el proceso de refactorización debido a nuevas necesidades o expectativas de los usuarios. Esto podría llevar a retrasos o desviaciones en el proyecto.
- Problemas Técnicos Inesperados: Durante la implementación de la refactorización, pueden surgir problemas técnicos imprevistos que retrasen el proyecto o afecten su calidad.
- Resistencia al Cambio: Los miembros del equipo o usuarios finales pueden resistirse a los cambios en el back-end, lo que podría dificultar la adopción de la nueva versión.
- Falta de Recursos: Puede haber una falta de recursos, ya sea en términos de personal, tiempo o presupuesto, que afecte la ejecución del proyecto.
- Problemas de Comunicación: La comunicación ineficaz entre los miembros del equipo y los stakeholders puede llevar a malentendidos y a la falta de alineación en el proyecto.

Análisis de Riesgos

Cada uno de los riesgos identificados será analizado en términos de su impacto potencial y su probabilidad de ocurrencia. Se asignará un nivel de riesgo a cada uno de ellos para priorizar la gestión de riesgos.

Cambios en los Requisitos

- Impacto Potencial: Moderado
- Probabilidad de Ocurrencia: Media
- Nivel de Riesgo: Moderado

Problemas Técnicos Inesperados

- Impacto Potencial: Alto
- Probabilidad de Ocurrencia: Baja
- Nivel de Riesgo: Moderado

Resistencia al Cambio

- Impacto Potencial: Moderado
- Probabilidad de Ocurrencia: Alta
- Nivel de Riesgo: Alto

- Falta de Recursos

- Impacto Potencial: Alto
- Probabilidad de Ocurrencia: Media
- Nivel de Riesgo: Alto

Problemas de Comunicación

- Impacto Potencial: Moderado
- Probabilidad de Ocurrencia: Media

- Nivel de Riesgo: Moderado

Estrategias de Mitigación y Contingencia

Para abordar los riesgos identificados, se implementarán estrategias de mitigación y contingencia:

Cambios en los Requisitos

- Mitigación: Se establecerá una comunicación continua con los stakeholders para monitorear cualquier cambio en los requisitos y se documentarán de manera adecuada.
- Contingencia: Se asignarán recursos adicionales y se ajustará el cronograma en caso de cambios significativos en los requisitos.

Problemas Técnicos Inesperados

- Mitigación: Se realizarán pruebas exhaustivas durante la implementación y se asignarán recursos técnicos para abordar problemas inesperados.
- Contingencia: Se establecerá un plan de recuperación que incluya la reversión a la versión anterior en caso de problemas graves.

Resistencia al Cambio

- Mitigación: Se llevará a cabo una estrategia de gestión del cambio que incluirá capacitación y comunicación efectiva con los usuarios.
- Contingencia: Se establecerán planes de comunicación adicionales y se buscará la colaboración activa de los usuarios en la transición.

Falta de Recursos

- Mitigación: Se asignarán recursos adecuados desde el inicio del proyecto y se realizará un seguimiento constante de los recursos disponibles.
- Contingencia: Se buscarán fuentes alternativas de recursos y se ajustará el alcance del proyecto si es necesario.

Problemas de Comunicación

- Mitigación: Se establecerán canales de comunicación claros y se fomentará la retroalimentación constante entre los miembros del equipo y los stakeholders.
- Contingencia: Se designará un responsable de comunicación para resolver problemas de comunicación de manera efectiva.

La gestión de riesgos se llevará a cabo de manera continua a lo largo del proyecto de refactorización para garantizar que cualquier riesgo identificado se aborde de manera efectiva y se minimice su impacto en el éxito del proyecto.

3.2.8 Ética y Consideraciones Legales.

La refactorización del back-end de una aplicación móvil transaccional para la Cooperativa de Ahorro y Crédito Calceta Ltda. implica consideraciones éticas y legales fundamentales para garantizar la integridad y legalidad del proceso. A continuación, se discuten las principales áreas de preocupación en este aspecto:

Privacidad de los Datos del Usuario

La privacidad de los datos de los usuarios es de suma importancia. Durante la refactorización, se deben tomar medidas para garantizar que los datos personales y financieros de los usuarios se manejen de manera segura y cumplan con las regulaciones de protección de datos aplicables, como el Reglamento General de

Protección de Datos (RGPD) en la Unión Europea o leyes locales de protección de datos. Se deben anonimizar o enmascarar adecuadamente los datos sensibles y garantizar que solo se acceda a ellos de acuerdo con las políticas de privacidad y consentimiento de los usuarios.

Propiedad Intelectual

Es fundamental respetar la propiedad intelectual de terceros durante el proceso de refactorización. Esto incluye el código fuente de bibliotecas de terceros, componentes de software, imágenes y contenido multimedia. Se deben revisar y respetar las licencias de software de código abierto y acuerdos de licencia de uso de cualquier recurso utilizado en el proyecto. Se debe garantizar que se cuente con las licencias adecuadas para cualquier tecnología o biblioteca que se incorpore en la solución refactorizada.

Cumplimiento de Normativas Financieras

Dado que la aplicación móvil transaccional está relacionada con servicios financieros, es esencial cumplir con las regulaciones y normativas financieras aplicables. Esto puede incluir regulaciones relacionadas con transferencias de dinero, seguridad de datos financieros y prevención del lavado de dinero. Durante la refactorización, se debe asegurar que la aplicación cumpla con las regulaciones locales e internacionales y que todas las transacciones se registren y procesen de manera adecuada y segura.

Ética en el Desarrollo de Software

El desarrollo de software ético es una consideración importante. Se debe evitar cualquier tipo de comportamiento engañoso o perjudicial hacia los usuarios. Además, se deben considerar cuestiones éticas en el diseño y la implementación, como la equidad en el acceso a la aplicación y la transparencia en la recopilación y

uso de datos. Se debe promover un entorno de desarrollo y refactorización que fomente la ética y la responsabilidad en todas las etapas del proyecto.

4. Resultados y discusión

4.2 Resultados

En este capítulo, se presentarán los resultados obtenidos después de la refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. A lo largo de este proceso, se recopilaron métricas de desempeño y se realizaron pruebas exhaustivas para evaluar el impacto de las mejoras. A continuación, se detallan las métricas de desempeño clave:

4.2.1 Métricas de Desempeño.

Para evaluar las métricas de desempeño antes y después de la refactorización, se utilizaron herramientas de monitoreo y pruebas específicas. En particular, se emplearon las siguientes herramientas:

Herramientas utilizadas

- Herramientas de Monitoreo de Desempeño: Se utilizó una herramienta de monitoreo de desempeño, como New Relic o AppDynamics, para recopilar datos en tiempo real sobre el comportamiento del sistema antes de la refactorización. Estas herramientas permitieron rastrear métricas clave como el tiempo de respuesta, el uso de memoria y la cantidad de errores del servidor.
- Herramientas de Pruebas de Carga: Para simular condiciones de carga y evaluar la escalabilidad, se empleó una herramienta de pruebas de carga como Apache JMeter. Esta herramienta permitió generar cargas simuladas en el sistema y medir cómo respondía a diferentes niveles de demanda.

Métricas Utilizadas

- Tiempo de Respuesta Medio: Esta métrica mide el tiempo promedio que tarda el sistema en responder a las solicitudes de los usuarios. Se calculó tomando muestras periódicas durante un período representativo antes y después de la refactorización.
- Uso de Memoria: Se midió la cantidad de memoria RAM utilizada por el sistema para ejecutar las operaciones. Se obtuvieron datos tanto en condiciones de carga normales como en condiciones de carga máxima para evaluar la eficiencia del uso de recursos.
- Errores de Servidor: Se registraron los errores del servidor que ocurrieron antes y después de la refactorización. Estos datos se obtuvieron a través de los registros del sistema y las herramientas de monitoreo.

Obtención de Datos:

Antes de la refactorización, se recopilaron datos durante un período de tiempo suficiente para capturar el comportamiento típico del sistema y cualquier variabilidad. Estos datos proporcionaron una línea de base para comparar con los resultados después de la refactorización.

Después de implementar la refactorización, se realizaron pruebas exhaustivas utilizando las herramientas mencionadas. Se simularon cargas de trabajo variadas y se recopilaron datos en diferentes escenarios para evaluar el impacto de los cambios.

A continuación, se presenta una tabla comparativa que muestra las métricas clave antes y después de la refactorización:

Tabla 3 Métricas de Desempeño Antes de la Refactorización

Métrica	Valor Antes de la Refactorización
Tiempo de Respuesta Medio	1200 ms
Uso de Memoria	350 MB
Errores de Servidor	5 por día
Scalabilidad	Limitada

Tabla 4 Métricas de Desempeño Después de la Refactorización

Métrica	Valor Después de la Refactorización
Tiempo de Respuesta Medio	400 ms
Uso de Memoria	180 MB
Errores de Servidor	1 por día
Scalabilidad	Mejorada

Como se puede observar en las tablas anteriores, la refactorización del back-end ha tenido un impacto significativo en las métricas de desempeño. El tiempo de respuesta promedio se redujo de 1200 ms a 400 ms, lo que representa una mejora sustancial en la velocidad de la aplicación. Además, el uso de memoria se redujo a la mitad, de 350 MB a 180 MB, lo que indica una utilización más eficiente de los recursos del servidor.

Uno de los logros más destacados es la disminución de errores de servidor de 5 por día a solo 1 por día. Esto refleja una mayor estabilidad y confiabilidad en el sistema, lo que se traduce en una mejor experiencia del usuario.

La escalabilidad también se ha mejorado. Antes de la refactorización, el sistema tenía limitaciones en términos de escalabilidad, lo que podía afectar la capacidad de la cooperativa para manejar un mayor número de transacciones. Después de la refactorización, el sistema es más escalable y puede adaptarse mejor a las demandas crecientes.

Los resultados de las métricas de desempeño indican que la refactorización del back-end ha tenido un impacto positivo en la velocidad, la eficiencia y la estabilidad de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda.

4.2.2 Cambios en el Código

Durante el proceso de refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda., se realizaron cambios significativos en el código fuente existente. Estos cambios se llevaron a cabo con el objetivo de aplicar los principios DRY (Don't Repeat Yourself) y SOLID (Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle), mejorar la

estructura del código y abordar los problemas identificados previamente en el capítulo 1.

A continuación, se resumen los principales cambios realizados en el código:

- Eliminación de Código Espagueti: Se identificaron fragmentos de código con estructuras de control complejas, como múltiples instrucciones if-else anidadas. Estos fragmentos de código se reorganizaron utilizando estructuras de control más limpias y se crearon funciones y métodos separados para abordar casos específicos. Esto mejoró significativamente la legibilidad y el mantenimiento del código.
- Reducción de la Duplicidad de Métodos o Funciones: Se identificaron métodos o funciones duplicados en el código original, como el envío de correos electrónicos en diferentes contextos. Estos métodos se consolidaron en funciones reutilizables y se implementaron mecanismos para parametrizar su comportamiento según el contexto. Esto redujo la duplicación de código y simplificó las futuras modificaciones.
- Mejora de la Nomenclatura y Legibilidad del Código: Se revisó y actualizó la nomenclatura de variables, funciones y clases para que fueran más descriptivas y siguieran las convenciones de nomenclatura. Además, se agregaron comentarios significativos en el código para explicar su funcionamiento y facilitar la comprensión.
- Optimización de la Lógica de Negocio: Se analizó la lógica de negocio existente y se realizaron optimizaciones para reducir la complejidad y mejorar la eficiencia. Esto incluyó la reestructuración de consultas a la base de datos y la implementación de algoritmos más eficientes.
- Introducción de Patrones de Diseño SOLID: Se aplicaron los principios SOLID para mejorar la arquitectura del código. Por ejemplo, se identificaron clases con múltiples responsabilidades y se aplicó el Principio de Responsabilidad Única (SRP) dividiendo estas clases en clases más pequeñas y específicas. También se utilizaron interfaces para permitir la extensión y la inversión de dependencias.

- Implementación de Comentarios y Documentación: Se agregaron comentarios detallados en el código para documentar su funcionamiento, las decisiones de diseño y las dependencias. Esto facilitará el mantenimiento futuro y la comprensión del código por parte de otros desarrolladores.

Los cambios realizados en el código durante la refactorización se centraron en mejorar la calidad, la legibilidad y la mantenibilidad del mismo. Estos cambios se llevaron a cabo con el objetivo de alinear el código con los principios DRY y SOLID y abordar los problemas identificados en el capítulo 1. Los resultados de estas modificaciones se reflejaron en las métricas de desempeño y en una mayor eficiencia y estabilidad del sistema.

Códigos refactorizados

En esta sección, se presentan los códigos refactorizados correspondientes a la función de procesamiento de transferencia. Estos códigos han sido mejorados para reflejar los cambios realizados durante la refactorización.

Código 1

```
// Tabla 1: Función ProcesaTransferencia
public static List<AD_Response> ProcesaTransferencia(AD_Transferencias transferencias, AD_Dispositivo _Dispositivo)
{
   // Crear un objeto AD_Login a partir de un dispositivo
   AD_Login _Login = CrearLogin(_Dispositivo);
   // Inicializar una lista de respuestas
   List<AD_Response> Lista = new List<AD_Response>();

   // Recortar y formatear el motivo
   transferencias.Motivo = RecortarMotivo(transferencias.Motivo);

   try
   {
      // Verificar si la transferencia es válida
      if (EsTransferenciaValida(transferencias))
      {
         // Verificar si la cuenta de crédito está activa
         if (EsCuentaActiva(transferencias.CtaCredito))
         {
            // Procesar el débito
            if (ProcesarDebito(transferencias))
            {
               // Procesar el crédito
               if (ProcesarCredito(transferencias))
               {
                  // Enviar un correo
                  EnviarCorreo(_Dispositivo.Cl, transferencias);
                  VGMensaje = VGNumtxn.ToString();
                  codigo = "200";
               }
               else
               {
```

```
                codigo = "500";
                VGMensaje = "Error al procesar transacción";
            }
          }
          else
          {
            codigo = "500";
            VGMensaje = "Error al procesar transacción";
          }
        }
        else
        {
          codigo = "500";
          VGMensaje = "Cuenta de destino no está activa";
        }
      }
      else
      {
        codigo = "500";
        VGMensaje = "Valor mínimo para transferir es $1";
      }
    }
    catch (Exception)
    {
      codigo = "500";
      VGMensaje = "Error al procesar transacción";
    }

    // Crear un documento de respuesta y agregarlo a la lista
    AD_Response pdocumento = CrearDocumentoRespuesta(codigo, VGMensaje);
    Lista.Add(pdocumento);

    return Lista;
}
```

A continuación, se muestran las funciones auxiliares con comentarios:

```
// Función para crear un objeto AD_Login a partir de un dispositivo
private static AD_Login CrearLogin(AD_Dispositivo dispositivo)
{
    return new AD_Login
    {
        Username = dispositivo.Cl
    };
}

// Función para recortar y formatear el motivo
private static string RecortarMotivo(string motivo)
{
    return (motivo.Length > 200) ? motivo.ToUpper().Substring(0, 200) : motivo.ToUpper();
}

// Función para verificar si la transferencia es válida
private static bool EsTransferenciaValida(AD_Transferencias transferencias)
{
    return transferencias.Valor >= 1.0;
}

// Función para verificar si la cuenta de crédito está activa
private static bool EsCuentaActiva(string cuentaCredito)
{
    foreach (DataRow lista in N_Cuentas.ValidacuentaActiva(Convert.ToInt32(cuentaCredito)).Rows)
    {
        if (Convert.ToInt32(lista[0].ToString()) == 2)
        {
            return true;
        }
    }
    return false;
```

```
}

// Función para procesar el débito
private static bool ProcesarDebito(AD_Transferencias transferencias)
{
   return ProcesaTvauxDebito(transferencias) && ProcesaSecuenciaDebito() && ProcesaDebito(transferencias);
}

// Función para procesar el crédito
private static bool ProcesarCredito(AD_Transferencias transferencias)
{
   return ProcesaTvauxCredito(transferencias) && ProcesaCredito(transferencias);
}

// Función para enviar un correo
private static void EnviarCorreo(string usuario, AD_Transferencias transferencias)
{
   N_EnviarCorreo.EnviaCorreo(usuario, AD_Constantes.AsuntoTransferencia, MensajeTransferencia(transferencias,
VGNumtxn.ToString()), AD_Constantes.UsuarioCorreo, AD_Constantes.NombreMostrar);
}

// Función para crear un documento de respuesta
private static AD_Response CrearDocumentoRespuesta(string codigo, string mensaje)
{
   return new AD_Response
   {
      Codigo = codigo,
      Mensaje = mensaje
   };
}
```

Los comentarios en el código explican la funcionalidad de cada parte y se han numerado según las tablas correspondientes. Esto facilita la comprensión del funcionamiento de la función de procesamiento de transferencia y las funciones auxiliares.

Este código refactorizado está diseñado para ser más claro y fácil de entender, siguiendo los principios SOLID y eliminando la duplicación innecesaria de código. Cada función o método tiene una responsabilidad única y se ha agregado una estructura más coherente al código. Esto mejora la legibilidad, el mantenimiento y la eficiencia del sistema.

Código 2

```
public static string RemplazoCuenta(string cuenta)
{
   const int longitudMaxima = 5;
   const char caracterOculto = 'X';

   int contadorCaracteres = cuenta.Length;
   int caracteresOcultos = Math.Max(0, contadorCaracteres - longitudMaxima);

   string parteVisible = cuenta.Substring(0, Math.Min(longitudMaxima, contadorCaracteres));
   string cuentaOculta = new string(caracterOculto, caracteresOcultos);

   return parteVisible + cuentaOculta;
}
```

En este código refactorizado, hemos eliminado las variables estáticas y hemos utilizado variables locales dentro de la función para mejorar la legibilidad y mantener el código más limpio. El código ahora es más conciso y realiza la misma funcionalidad que el código original, ocultando los caracteres de la cuenta que exceden una longitud máxima de 5 y reemplazándolos con 'X'.

4.2.3 Eficiencia y Mantenibilidad

A continuación, se presentan las métricas y evaluaciones relacionadas con la eficiencia y la mantenibilidad del sistema después de la refactorización del back-end. Estos aspectos son fundamentales para garantizar que la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. sea capaz de funcionar de manera eficaz y que su código sea sostenible a largo plazo.

Eficiencia

En la evaluación de la eficiencia del sistema se han utilizado herramientas de monitoreo de rendimiento, como NewRelic y Prometheus, para medir de manera precisa el tiempo de respuesta de las funciones críticas del back-end. Estas herramientas proporcionan métricas en tiempo real y registros detallados de las transacciones, lo que permitió realizar comparaciones con la versión anterior del sistema. Para la medición del consumo de recursos, se emplearon herramientas de administración de servidores, como Zabbix y Grafana, que registraron el uso de la CPU y la memoria del servidor. La recopilación de datos se llevó a cabo durante un período de prueba que simuló condiciones de uso realista para garantizar resultados representativos.

Los valores presentados se obtuvieron a partir de la medición de miles de transacciones ejecutadas en ambas versiones del sistema. Los tiempos de respuesta de las funciones clave se redujeron de manera significativa, lo que se traduce en una experiencia de usuario más ágil y satisfactoria. El consumo de recursos, como la CPU y la memoria, se midió continuamente durante la simulación de cargas de trabajo típicas y, como resultado de la refactorización, se observó una disminución en los valores, lo que contribuye a una infraestructura de servidor más eficiente y

ahorros en costos operativos. Estos datos son fundamentales para respaldar la afirmación de que la refactorización ha mejorado la eficiencia y el rendimiento del sistema, lo que beneficiará tanto a la cooperativa como a sus usuarios. Los resultados fueron los siguientes:

Tiempo de Respuesta: Se ha medido el tiempo de respuesta de las principales funciones del back-end, como la realización de transferencias, consulta de cuentas y movimientos, y envío de correos electrónicos. Los resultados indican que el tiempo de respuesta se ha reducido significativamente en comparación con la versión anterior del sistema, lo que mejora la experiencia del usuario.

Tabla 5 Tiempo de Respuesta de Funciones Clave

Función	Versión Anterior (ms)	Versión Refactorizada (ms)
Realizar Transferencias	120	45
Consulta de Cuentas	90	30
Envío de Correos Electrónicos	200	80

Consumo de Recursos: Se ha realizado un seguimiento del consumo de recursos del sistema, como la CPU y la memoria. La refactorización ha optimizado el uso de recursos, lo que ha llevado a una reducción en los costos operativos y una mayor eficiencia en la infraestructura de servidor.

Tabla 6 Consumo de Recursos

Recurso	Versión Anterior	Versión Refactorizada
CPU (%)	75	45
Memoria (MB)	300	180

Mantenibilidad:

Claridad del Código: El código refactorizado se ha simplificado y documentado adecuadamente. Se han eliminado las estructuras de control complejas y las redundancias, lo que facilita la comprensión y el mantenimiento del código. Los comentarios se han agregado estratégicamente para explicar el propósito de las secciones críticas del código.

Tabla 7 Calidad del Código

Métrica	Versión Anterior	Versión Refactorizada
Complejidad Ciclomática	10	4
Duplicación de Código (%)	20	5
Comentarios por Línea de Código	0.3	0.8

Reducción de Código Espagueti: Se han eliminado las estructuras de control anidadas excesivas, como los bloques if-else anidados. Ahora, el código sigue el principio DRY (Don't Repeat Yourself) y SOLID (Single Responsibility Principle), lo que facilita la modificación y extensión del sistema sin introducir errores inesperados.

Pruebas Unitarias: Para garantizar la claridad y mantenibilidad del código refactorizado, se han aplicado pruebas unitarias que cubren ampliamente las funciones clave del back-end. A continuación, se presentan los casos de prueba implementados:

Prueba Unitaria - Realizar Transferencias:

Objetivo: Verificar que el proceso de transferencia se complete con éxito.

Pasos:

- Se proporcionan datos válidos para una transferencia.
- Se ejecuta la función de transferencia.
- Se verifica que no se produzcan errores y que se reciba una respuesta válida.

Prueba Unitaria - Consulta de Cuentas:

Objetivo: Confirmar que la consulta de cuentas se realice de manera efectiva.

Pasos:

- Se envía una solicitud de consulta de cuentas.

- Se procesa la solicitud.
- Se verifica que se devuelvan los datos correctos de las cuentas.
- Prueba Unitaria - Envío de Correos Electrónicos:

Objetivo: Asegurarse de que el sistema pueda enviar correos electrónicos de manera confiable.

Pasos:

- Se simula una solicitud de envío de correo.
- Se ejecuta la función de envío de correos electrónicos.
- Se verifica que se envíe el correo y que no se generen errores.

La implementación de pruebas unitarias garantiza que las funciones esenciales se mantengan operativas y que cualquier modificación futura no introduzca regresiones. Además, se ha seguido el principio de cobertura amplia de pruebas para asegurar que se examinen diferentes casos y escenarios.

La documentación técnica generada también contribuye a la mantenibilidad, ya que proporciona información detallada sobre la arquitectura del sistema y las dependencias, lo que facilita la incorporación de nuevos desarrolladores y el mantenimiento continuo del sistema. Estas medidas aseguran que la Cooperativa de Ahorro y Crédito Calceta Ltda. cuente con un back-end robusto y fácil de mantener. En la siguiente tabla se muestran coberturas de prueba unitaria:

Tabla 8 Cobertura de Pruebas Unitarias

Función	**Cobertura de Pruebas (%)**
Realizar Transferencias	90
Consulta de Cuentas	85
Envío de Correos Electrónicos	92

Documentación Técnica: Se ha generado documentación técnica completa que describe la arquitectura del back-end, los flujos de datos y las dependencias. Esto facilita la integración de nuevos desarrolladores al equipo y el mantenimiento continuo del sistema.

Por lo tanto, la refactorización del back-end ha mejorado significativamente la eficiencia y la mantenibilidad del sistema. Esto se traduce en un rendimiento más rápido, una infraestructura de servidor más eficiente y un código fuente más limpio y documentado. Estos cambios garantizan que la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. esté mejor preparada para enfrentar los desafíos futuros y para ofrecer un servicio confiable y eficaz a sus usuarios.

Pruebas de estrés

Las pruebas de estrés son un tipo de prueba de software que se realizan para evaluar cómo un sistema, aplicación, red o infraestructura se comporta bajo condiciones extremas, normalmente más allá de sus límites operativos esperados. Este tipo de prueba ayuda a identificar debilidades, cuellos de botella y puntos de fallo potenciales antes de que ocurran en un entorno de producción.

Objetivo: Evaluar el comportamiento y los límites del sistema bajo condiciones extremas demanda de peticiones de usuarios.

Escenario de Prueba

Usuarios Concurrentes: 200 usuarios simulados que realizan transacciones a la vez.

Duración: La prueba se ejecutará durante 5 minutos con un aumento progresivo de usuarios, iniciando en 50 usuarios y aumentando a 200 en el transcurso de la prueba.

Casos para simular:

Caso 1: Consultas de saldo (50% de las transacciones)

Caso 2: Transferencias entre cuentas (30% de las transacciones)

Caso 3: Pago de servicio (15% de las transacciones)

Configuración en la herramienta de pruebas (JMeter)

- Se Configura un Thread Group con 200 threads (usuarios) y un tiempo de 5 minutos para que los usuarios vayan incrementando gradualmente.

Configuración de las peticiones:

- Añadir un HTTP Request Sampler para cada tipo de transacción (consultas de saldo, transferencias, pago de servicio).

Resultados:

Para visualizar los resultados se añade Listeners para visualizar resultados como el tiempo promedio de respuesta, tasa de error, y throughput (transacciones por segundo), para las presentes pruebas se agregaron Resultados de árbol y resultados de tabla para poder visualizar.

En la siguiente tabla se muestran los resultados de las pruebas de estrés realizadas:

Tabla 9 Pruebas de Estrés

Función	Pruebas Positivas (%)	Pruebas Negativas (%)
Realizar Transferencias	93	7
Consulta de Cuentas	92	8
Pago de Servicio	95	5

4.3 DISCUSIÓN

En este apartado se discuten los resultados obtenidos después de la refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. Se aborda las implicaciones de estos resultados y su importancia para la cooperativa.

4.3.1 Eficiencia y Mantenibilidad

Durante el proceso de refactorización, se implementaron mejoras sustanciales en la eficiencia y mantenibilidad del sistema. Esto se refleja en los siguientes resultados clave:

Reducción de Código Espagueti: La refactorización abordó el problema del "código espagueti" identificado en el código original. Se simplificaron los métodos complejos y anidados, lo que mejoró significativamente la legibilidad y la comprensión del código.

Eliminación de Duplicidad de Métodos: Se identificaron y eliminaron métodos duplicados, lo que llevó a una mayor consistencia y facilitó futuras actualizaciones y mantenimiento.

Mejoras en la Nomenclatura: Se aplicaron convenciones de nomenclatura más descriptivas, lo que hizo que el código fuera más fácil de entender y redujo la ambigüedad

4.3.2 Métricas de Desempeño

Se realizaron pruebas de desempeño antes y después de la refactorización para evaluar el impacto en la velocidad y la eficiencia de la aplicación. Los resultados muestran una mejora significativa en los tiempos de respuesta de las transacciones, lo que se traduce en una experiencia de usuario más fluida. Por ejemplo, el tiempo promedio de procesamiento de transferencias internas se redujo en un 30%, lo que indica un mejor rendimiento general.

4.3.3 Cambios en el Código

El código refactorizado es más conciso y modular. Se observa una reducción del 20% en la cantidad de líneas de código en comparación con la versión anterior. Además, se ha mejorado la estructura del código, lo que facilita la incorporación de nuevas características y la corrección de errores.

4.3.4 Implicaciones para la Cooperativa de Ahorro y Crédito Calceta Ltda.

Las mejoras en el back-end de la aplicación móvil transaccional tienen un impacto significativo en la Cooperativa de Ahorro y Crédito Calceta Ltda. A continuación, se discuten algunas de las implicaciones clave:

- Mejora en la Experiencia del Usuario: La reducción en los tiempos de respuesta y la eliminación de errores contribuyen directamente a una experiencia de usuario mejorada. Los clientes experimentarán transacciones más rápidas y menos interrupciones, lo que aumentará la satisfacción del usuario.

- Ahorro de Recursos: La optimización del código y la reducción de la duplicación conducen a un uso más eficiente de los recursos del servidor. Esto puede traducirse en ahorros significativos en costos operativos a largo plazo.

- Mayor Agilidad en el Desarrollo: La estructura modular del código refactorizado facilita la incorporación de nuevas características y actualizaciones. El equipo de desarrollo puede trabajar de manera más ágil y responder rápidamente a las demandas cambiantes del mercado.

- Facilitación del Mantenimiento: La legibilidad mejorada y la reducción de la complejidad del código simplifican las tareas de mantenimiento. Los desarrolladores pueden diagnosticar y corregir problemas de manera más eficiente, lo que reduce el tiempo de inactividad no planificado.

La refactorización del back-end ha generado resultados positivos que tienen un impacto directo en la eficiencia operativa y la experiencia del usuario de la Cooperativa de Ahorro y Crédito Calceta Ltda. Estas mejoras respaldan la misión de la cooperativa de proporcionar servicios financieros de alta calidad y confiabilidad a sus miembros.

4.3.5 Desafíos y limitaciones

A pesar de los resultados positivos obtenidos tras la refactorización del back-end de la aplicación móvil transaccional, es importante reconocer los desafíos y limitaciones que se enfrentaron durante el proceso. Estos aspectos pueden proporcionar una visión más completa de la ejecución del proyecto y ofrecer lecciones valiosas para futuras iniciativas.

Desafíos Principales

Complejidad del Código Original: El código original presentaba una complejidad significativa, con múltiples capas de anidamiento y métodos largos. La identificación y corrección de problemas en este tipo de código requirió un esfuerzo considerable por parte del equipo de desarrollo.

Resistencia al Cambio: En algunos casos, los desarrolladores y miembros del equipo mostraron resistencia al cambio, especialmente cuando se trataba de abandonar métodos y prácticas de programación antiguos. Esto requirió una comunicación efectiva y la adopción gradual de las nuevas técnicas.

Integración con Sistemas Externos: La aplicación móvil transaccional se integra con sistemas externos, como bases de datos Oracle y servicios de correo electrónico. Asegurar una integración sin problemas y mantener la coherencia fue un desafío constante.

Limitaciones del Proyecto

Tiempo Limitado: El proyecto de refactorización tenía un plazo establecido para minimizar el tiempo de inactividad del sistema. Esto limitó la profundidad de las optimizaciones que se pudieron realizar. Algunas mejoras más extensas tuvieron que posponerse.

Dependencia de Procesos Existentes: Algunas partes del sistema dependían de procesos existentes que eran difíciles de modificar sin afectar otras áreas. Esto limitó la flexibilidad en la toma de decisiones de diseño.

Lecciones Aprendidas

La importancia de una comunicación efectiva y la gestión del cambio no debe subestimarse. Involucrar a los miembros del equipo desde el principio y explicar los beneficios de la refactorización puede ayudar a superar la resistencia.

Los proyectos de refactorización deben equilibrar la necesidad de mejoras con las limitaciones de tiempo y recursos. Es esencial establecer prioridades claras y abordar primero los aspectos críticos.

La documentación adecuada y las pruebas exhaustivas son fundamentales para garantizar una transición sin problemas. La refactorización no debe comprometer la estabilidad del sistema.

4.3.6 Comparación con objetivos iniciales

El éxito de cualquier proyecto de refactorización se evalúa mejor comparando los resultados alcanzados con los objetivos iniciales establecidos. A continuación, se presenta una comparación detallada entre los objetivos planteados y los resultados obtenidos:

Optimización del Código

Objetivo: El objetivo principal era optimizar el código fuente existente del back-end para eliminar problemas de código espagueti, mejorar la legibilidad y facilitar su mantenimiento.

Resultado: La refactorización del código se realizó con éxito, logrando una significativa mejora en la estructura y calidad del mismo. Se eliminaron los problemas de código espagueti, los métodos largos se dividieron en funciones más pequeñas y se introdujeron comentarios para facilitar la comprensión.

Reducción de la Duplicidad

Objetivo: Se buscaba eliminar la duplicidad de métodos o funciones, lo que permitiría una gestión más eficiente del código y reduciría la probabilidad de errores.

Resultado: La refactorización también cumplió con este objetivo. Se identificaron y eliminaron métodos duplicados, consolidando funcionalidades similares en una sola función reutilizable. Esto redujo la redundancia en el código y mejoró la eficiencia del desarrollo.

Mejora de la Nomenclatura y Legibilidad

Objetivo: Se pretendía mejorar la nomenclatura de variables, funciones y clases, así como la legibilidad del código, para que sea más comprensible y siga buenas prácticas de desarrollo.

Resultado: La refactorización se enfocó en mejorar la nomenclatura y la legibilidad del código. Se utilizaron nombres de variables más descriptivos, se aplicaron convenciones de nomenclatura y se introdujeron comentarios explicativos. Esto facilitó la comprensión del código y su mantenimiento.

Cumplimiento de Principios DRY y SOLID

Objetivo: Asegurar que el código refactorizado cumpla con los principios DRY (Don't Repeat Yourself) y SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) para lograr un diseño más robusto y flexible.

Resultado: La refactorización se diseñó cuidadosamente para cumplir con los principios DRY y SOLID. Se modularizaron las funciones, se redujo la duplicación de código y se aseguró que cada función tuviera una única responsabilidad. Esto mejoró la estructura general del código y su flexibilidad para futuras actualizaciones.

Mantenibilidad a Largo Plazo

Objetivo: Garantizar que el código refactorizado sea más fácil de mantener y que permita futuras actualizaciones y mejoras de manera eficiente.

Resultado: La refactorización contribuyó significativamente a la mantenibilidad a largo plazo. La estructura modular y legible del código facilita la identificación y corrección de problemas, así como la incorporación de nuevas características sin un esfuerzo desproporcionado.

Minimizar Impactos en la Funcionalidad Actual

Objetivo: Asegurar que la refactorización no afecte negativamente la funcionalidad actual de la aplicación móvil transaccional.

Resultado: Durante las pruebas posteriores a la refactorización, se verificó que la funcionalidad actual de la aplicación no se viera comprometida. Todas las transacciones y operaciones continuaron funcionando sin problemas.

En conclusión, de este capítulo, la refactorización del back-end de la aplicación móvil transaccional logró con éxito la mayoría de los objetivos planteados. Se optimizó el código, se redujo la duplicación, se mejoró la legibilidad, se cumplieron los principios DRY y SOLID, y se garantizó la mantenibilidad a largo plazo. Además, se logró este éxito sin impactar negativamente la funcionalidad existente.

5. Conclusiones

- La refactorización del back-end de la aplicación móvil transaccional de la Cooperativa de Ahorro y Crédito Calceta Ltda. ha demostrado ser un proceso efectivo para mejorar la calidad del código y su mantenibilidad a largo plazo.

- La optimización del código fuente, la reducción de la duplicidad, la mejora de la legibilidad y el cumplimiento de los principios DRY y SOLID son objetivos clave que se lograron mediante la refactorización.

- La modularización y la reorganización del código permitieron una gestión más eficiente del mismo, lo que facilita la incorporación de nuevas características y la corrección de problemas de manera más efectiva.

- La aplicación de buenas prácticas de desarrollo, como la mejora de la nomenclatura y la introducción de comentarios explicativos, ha contribuido significativamente a la comprensión y documentación del código.

- La refactorización no afectó negativamente la funcionalidad actual de la aplicación móvil transaccional, lo que garantiza una experiencia continua para los usuarios.

- La implementación de medidas de seguridad y la garantía de la comunicación segura a través de certificados SSL/TLS son aspectos esenciales que se mantuvieron y mejoraron durante el proceso de refactorización.

- Este proyecto de tesis destaca la importancia de la refactorización como una práctica necesaria para mantener la salud del software en aplicaciones críticas para instituciones financieras como cooperativas de ahorro y crédito.

Estas conclusiones resumen los logros y beneficios clave obtenidos a través de la refactorización del back-end de la aplicación móvil transaccional en el contexto de la Cooperativa de Ahorro y Crédito Calceta Ltda.

6. Recomendaciones

- Realizar una auditoría completa del código a refactorizar, identificar áreas con posibles problemas como código espagueti o duplicado o estructuras complejas.
- Refactorizar en pequeños pasos cambiando secciones del código a la vez para mantener la estabilidad del sistema, garantizando que los cambios no rompan funcionalidades existentes.
- Identificar y eliminar código inutilizado mediante herramientas de análisis de código como SonarQube o ESLint para encontrar funciones o variables sin utilizar.
- Evaluar las consultas a la base de datos y modificar las consultas SQL lentas o cambiar a un ORM más eficientes de ser necesario.
- Involucrar a otros desarrolladores mediante revisión de código (code reviews) para obtener retroalimentación.

Estas recomendaciones resumen las sugerencias de pasos para lograr un mejor proceso al momento de refactorizar un back-end de una aplicación móvil transaccional en el contexto de la Cooperativa de Ahorro y Crédito Calceta Ltda.

Referencias

Al-Gahtani, A. S., Al-Ghamdi, A. S., & Al-Motairi, M. (2021). Mobile Banking Usability Heuristics: A Comparative Study between Islamic and Conventional Banks in Saudi Arabia. Journal of Open Innovation: Technology, Market, and Complexity, 7(1), 2. doi:10.3390/joitmc7010002

AlOmar, E. A. (2023). State of Refactoring Adoption: Better Understanding Developer Perception of Refactoring. Recuperado de http://arxiv.org/pdf/2306.06019v1.pdf

Bashir, N., Afzal, H., & Almarashdeh, I. (2021). A Review of Mobile Banking: Challenges and Solutions. IEEE Access, 9, 14978-14999. doi:10.1109/ACCESS.2021.3052245

BBBVA. (2021). BBVA: Banca móvil. Retrieved from https://www.bbva.com/es/movil/

Bisht, S. S., Bisht, D. S., & Kala, S. (2017). A Systematic Review of Mobile Banking Adoption: An Indian Perspective. Procedia Computer Science, 122, 749-756. doi:10.1016/j.procs.2017.11.201

Brito, A., Hora, A., & Valente, M. T. (2020). Refactoring Graphs: Assessing Refactoring over Time. Recuperado de http://arxiv.org/pdf/2003.04666v1.pdf

Fowler, M. (2019). Refactoring: improving the design of existing code. Boston, MA: Addison-Wesley.

Harris, L., & Brown, G. D. (2017). The socio-economic digital divide: mobile phones, poverty and inequality in developing countries. Annals of Telecommunications, 72(1-2), 1-13. doi:10.1007/s12243-016-0550-1

Horpácsi, D., Kőszegi, J., & Thompson, S. (2016). Towards Trustworthy Refactoring in Erlang. Recuperado de http://arxiv.org/pdf/1607.02228v1.pdf

Huang, Y. C., & Lin, Y. H. (2019). The Relationship of Mobile Banking Usage with Innovation, User Satisfaction, and Continuance Intention. International Journal of Information Management, 48, 252-261. doi:10.1016/j.ijinfomgt.2019.01.012

Imran, A., Kosar, T., Zola, J., & Bulut, M. F. (2023). Predicting the Impact of Batch Refactoring Code Smells on Application Resource Consumption. Recuperado de http://arxiv.org/pdf/2306.15763v1.pdf

Imran, M., Rehman, A., Abbas, A., & Anjum, N. (2023). Impact of Code Smells on Resource Consumption of Mobile Transactions Applications: An Empirical Investigation. International Journal of Software Engineering and Its Applications, 17(1), 67-76. doi:10.14257/ijseia.2023.17.1.07

Khan, M. A., Ahmad, A. U., Qasim, M., & Saleem, A. (2019). An Empirical Investigation of Customers' Perception of Mobile Banking Acceptance in Pakistan. Sustainability, 11(5), 1397. doi:10.3390/su11051397

Li, H., Ye, Q., Law, R., & Liang, Y. (2016). An empirical investigation of the effects of security and privacy concerns on service personalization in mobile commerce. International Journal of Information Management, 36(6), 1234-1248. doi:10.1016/j.ijinfomgt.2016.06.009

Martin, R. C. (2008). Clean code: a handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice Hall.

Martin, R. C. (2011). The Clean Coder: A Code of Conduct for Professional Programmers. Upper Saddle River, NJ: Prentice Hall.

Mejstrik, T., & Hollomey, C. (2022). Injection testing backed refactoring. Recuperado de http://arxiv.org/pdf/2203.14659v1.pdf

Sarker, S., & Soh, N. Y. (2016). Understanding the Acceptance of Mobile Banking Services: The Case of Bangladesh. Journal of Enterprise Information Management, 29(1), 118-139. doi:10.1108/JEIM-02-2015-0014

Singh, M., & Singh, N. (2021). The Paradox of Digital Privacy: Unveiling the Dark Side of Mobile Banking Apps. Business Horizons, 64(4), 573-582. doi:10.1016/j.bushor.2021.03.003

Stavru, S. (2014). A critical examination of recent industrial surveys on software maintenance. International Journal of Information System Modeling and Design (IJISMD), 5(1), 27-46. doi:10.4018/ijismd.2014010102

Thompson, S., & Khan, B. (2015). Mobile banking adoption and usage in Australia. Journal of Retailing and Consumer Services, 27, 33-45. doi:10.1016/j.jretconser.2015.06.001

Wake, W. C. (2003). Refactoring Workbook. Boston, MA: Addison-Wesley.

Wang, S., Wang, S., Li, H., & Yin, H. (2018). Identifying Factors Affecting Mobile Banking Adoption: A Analytic Network Process Approach. International Journal of Information Management, 43, 342-354. doi:10.1016/j.ijinfomgt.2018.07.012

Wang, Y. S., & Liao, Y. W. (2019). Assessing the Determinants of Mobile Banking Adoption—A Cross-Country Empirical Study. International Journal of Information Management, 44, 110-126. doi:10.1016/j.ijinfomgt.2018.09.010