UNIVERSIDAD NACIONAL

"SANTIAGO ANTÚNEZ DE MAYOLO"

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL

DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

"ARQUITECTURA DE MICROSERVICIOS PARA MEJORAR LA CALIDAD DE SOFTWARE EN UNA ENTIDAD BANCARIA DE LIMA METROPOLITANA, 2022"

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO DE SISTEMAS E INFORMÁTICA

PRESENTADO POR:

Bach. CAQUI MEJÍA DANIEL NOEL

ASESOR:

Mag. MIGUEL ANGEL SILVA ZAPATA

HUARAZ - PERU

2022

Nº Registro: T171

DEDICATORIA

A Dios por ofrecerme tantas oportunidades de superación y darme la fuerza necesaria para lograr mis metas.

Dedico esta tesis con mucho amor y cariño a las personas más importantes en mi vida, mi Madre Santa Juana, mis Hermanos Ricardo, Robín, Teófilo, Cinia y Eleazar, gracias a ellos por apoyarme siempre cuando los necesite, y por darme las fuerzas para seguir adelante y cumplir mis metas.

AGRADECIMIENTO

Ante todo, agradezco a Dios, quien me ha dado las fuerzas para seguir, y a mi familia por estar conmigo en todo momento.

A mis amigos y familiares, quienes me brindaron su apoyo, ayuda y entusiasmo en todo momento y confiaron siempre que concluiría el presente trabajo.

RESUMEN

La presente tesis titulada “Arquitectura de microservicios para mejorar la calidad de software en una entidad bancaria de Lima Metropolitana, 2022”, tiene como objetivo mejorar la calidad de software mediante Arquitectura de Microservicios en una entidad bancaria Lima metropolitana. Existe una necesidad apremiante de una arquitectura de microservicios en el departamento de TI en este momento, ya que esto permitiría la integración y la entrega continuas, simplificaría las actualizaciones de código y mejoraría el rendimiento de la calidad del software.

Para el desarrollo de la nueva arquitectura, se utilizó la metodológica SCRUM, esto permitió su creación y funcionamiento con un conjunto de pequeños servicios que se ejecutaron de manera autónoma e independiente. Con respecto a la metodología de investigación se trató de una investigación aplicada, y con un diseño pre experimental, la población estuvo conformado por 158 y una muestra de 112 colaboradores del área tecnológica la calidad del software de Canales digitales de BCP.

Los resultados, del pretest muestran que la mejora de la calidad del software se encuentra en un nivel deficiente en 38,0%, nivel regular en 48,2% y nivel eficiente en 13,4%. Se puede apreciar que el nivel preponderante fue bajo en el pretest. Luego de aplicar la arquitectura de microservicios los resultados mostraron que la calidad del software se encuentra en un nivel deficiente en 20,5%, nivel regular en 31,3% y nivel eficiente en 48,2%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la calidad del software fue la adecuada, luego de usar los microservicios.

Con el fin de realizar una investigación exhaustiva, la presente tesis se redactó de acuerdo con las fases y la estructura de la investigación establecidas en el reglamento de carreras y títulos de la Facultad de Ciencias (RCUR N° 182-2021-UNASAM).

Palabras Clave: Calidad del software, arquitectura de microservicios, aplicación independiente.

ABSTRACT

This thesis entitled "Architecture of microservices to improve the quality of software in a bank in Metropolitan Lima, 2022", aims to improve the quality of software through Microservices Architecture in a bank in Metropolitan Lima. There is a pressing need for a microservices architecture in the IT department right now, as this would enable continuous integration and delivery, simplify code updates, and improve software quality performance.

For the development of the new architecture, the SCRUM methodology was achieved, this allowed its creation and operation with a set of small services that were executed autonomously and independently. Regarding the research methodology, it was an applied research, and with a pre-experimental design, the population consisted of 158 and a sample of 112 collaborators from the technological area, the quality of the BCP digital Channels software.

The results of the pretest show that the improvement of the software quality is at a deficient level at 38.0%, a regular level at 48.2% and an efficient level at 13.4%. It can be seen that the prevailing level was low in the pretest. After applying the microservices architecture, the results showed that the software quality was found at a deficient level in 20.5%, a regular level in 31.3% and an efficient level in 48.2%. It can be seen that the preponderant level was the high level, which shows that for the collaborators of the technological area the quality of the software was adequate, after using the microservices.

In order to carry out an exhaustive investigation, this thesis was written in accordance with the phases and the structure of the investigation established in the regulation of careers and titles of the Faculty of Sciences (RCUR N° 182-2021-UNASAM).

Keywords: Software quality, microservices architecture, standalone application.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

I. INTRODUCCIÓN

1.1. Planteamiento del problema

Según el Fondo Monetario Internacional (2022) en la actualidad las vulneraciones informáticas al sistema financiero mundial han experimentado un aumento del 200%, ya que todas las operaciones se producen por los aplicativos informáticos-financieros, lo que genera desconfianza por parte de los clientes en utilizar en forma segura estos aplicativos.

Según Rojas (2021) las entidades públicas o privadas, independiente del rubro al que pertenecen, deben de buscar soluciones definitivas, ya que un gran problema que presentan son los servicios monolíticos cuyo funcionamiento, así como mantenimiento se hace difícil al paso del tiempo, además que poseen vulnerabilidades que pueden ser explotadas por agentes extraños a las entidades financieras; por lo que estas se encuentran en total vulnerabilidad, además que sus servicios están desfasados.

Según la Asociación de Bancos del Perú (2022) las entidades bancarias de Lima Metropolitana se encuentran en plena expansión, ya que sus actividades han experimentado un aumentado del 150%; en este sentido, para seguir creciendo deben de dar a su clientela eficiencia y confianza en sus servicios. Es ante este crecimiento que las entidades financieras, para darle un mejor servicio a sus clientes, ante el temor de perderlos, necesitan modernizar sus servicios; por ello, es conveniente migrar de la arquitectura monolítica a una arquitectura microservicio, que permita a las entidades financieras mayor flujo de transacciones, seguridad y rapidez. De esta forma, las empresas del sector financiero generarían mayor rentabilidad, ya que las plataformas de pago en línea son las más utilizadas por los clientes solo en Lima metropolitana según Asociación de Bancos del Perú (2022) aumenta cada año en un 80% aproximadamente.

En la actualidad el servicio de aplicaciones utilizado en la empresa bancaria es una arquitectura monolítica, la cual trae consigo algunas dificultades como las siguientes: demora por parte de los desarrolladores en aprender el sistema para realizar cambios, agregar nuevos componentes o resolver incidencias, aplicaciones que generan cuellos de botella debido a la centralización de su arquitectura y a las diferentes peticiones que reciben, el rendimiento no es el óptimo para atender las peticiones de los usuarios;

la realización de pruebas unitarias es difícil de realizar haciendo que no identifiquen ciertos errores y deficiencias de manera oportuna, la escalabilidad es difícil de lograr afectando su tiempo de vida útil; así también al no estar modularizadas trae consigo dificultades y empleo de tiempo incensario durante el proceso de mantenimiento. Es por ello se necesita cambiar de tipo de arquitectura de monolítico a microservicios.

1.2. Formulación del problema

1.2.1. Problema general

¿De qué manera se mejora la calidad del software en una entidad bancaria Lima metropolitana con la arquitectura de microservicios?

1.2.2. Problemas específicos

- ¿De qué manera la arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima Metropolitana?
- ¿De qué manera la arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima Metropolitana?
- ¿De qué manera la arquitectura de microservicios incide en la integridad del software en una entidad bancaria Lima Metropolitana?

-

1.3. Objetivo de la investigación

1.3.1. Objetivo general

Mejorar la calidad de software mediante Arquitectura de Microservicios en una entidad bancaria Lima Metropolitana.

1.3.2. Objetivos específicos

- Determinar cómo una arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima Metropolitana.
- Determinar cómo una arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima Metropolitana.
- Determinar cómo una arquitectura de microservicios incide en la integridad del software en una entidad bancaria Lima Metropolitana.

1.4. Justificación de la investigación

1.4.1. Justificación social

Se justifica socialmente porque servirá de ejemplo para entidades públicas y privadas en realizar el cambio progresivo de arquitecturas monolíticas a una arquitectura de microservicios. Además, que sería fácil uso y aprendizaje generando mayor satisfacción en la ciudadanía.

1.4.2. Justificación económica

Se justifica porque esta arquitectura microservicios es mucho más sencilla y rápida en su implementación, así como en su mantenimiento a comparación con arquitecturas tradicionales, porque en su mantenimiento empleara la mitad de tiempo, así como la mitad de personal significando un ahorro económico para la empresa y maximizando sus ganancias para que los clientes podrán hacer sus operaciones con normalidad ya que la empresa produce pérdidas cada vez que el sistema no funciona.

1.4.3. Justificación tecnológica

El estudio actual se basa en una arquitectura dinámica de microservicios que puede hacer uso de herramientas de desarrollo de software.

1.4.4. Justificación legal

La investigación en esta área se basa en el artículo 1 del Decreto Legislativo N° 1412, que ordena el establecimiento de un marco de gobernanza de gobierno digital para la administración eficiente de los servicios digitales, identidad digital, la arquitectura digital, la interoperabilidad y la seguridad digital. los números, junto con el marco legal que rige la adopción y el uso de tecnología digital en toda la entidad para digitalizar procesos y brindar servicios digitales.

1.4.5. Justificación normativa

Las ramificaciones de este estudio sobre la calidad del software son de gran alcance y abarcan cuestiones como los costos de desarrollo y mantenimiento, la escalabilidad, la compatibilidad con tecnologías de terceros y la conveniencia de buscar código en varios módulos.

II. MARCO TEÓRICO

2.1. Antecedentes de la investigación

Después de una recopilación de información en los diversos repositorios y bases de datos, se toma como referencia los siguientes:

2.1.1. Antecedentes internacionales

Flores (2020) en su estudio señala que el módulo de registro de deportistas y seguimiento médico de la Federación Deportiva de Imbabura fue el foco de esta investigación, y se realizó utilizando la arquitectura de microservicios de Spring Cloud. Para lograrlo, esta investigación utilizó un diseño preexperimental, una metodología cuantitativa y una metodología aplicada. El valor adquirido de 80,86% de calidad fue validado utilizando la matriz de calidad ISO/IEC 25022, que mostró en detalle el resultado de las características creadas en el modelo de calidad. En uso. Se determinó que la técnica SCRUM fue exitosa en el desarrollo de requerimientos de software debido a su flexibilidad y su estructura de operaciones iterativas.

Guevara (2018) en su estudio tuvo por objetivo era realizar un estudio de la literatura que nos ayudara a comprender cómo se desarrollan metodológicamente los microservicios para que pudiéramos proporcionar un plan para su implementación. Esta investigación empleó un diseño no experimental descriptivo, una técnica híbrida y un enfoque aplicado. Los hallazgos demuestran que se propuso un enfoque metódico para la creación de microservicios, y que esta estrategia se desarrolló y aplicó en el prototipo final mediante el uso de características de calidad. Se determinó que se necesita investigación adicional sobre los enfoques relacionados con los microservicios para profundizar la comprensión de esta arquitectura y proporcionar un trampolín hacia el campo más amplio de la ingeniería de software.

López (2018) realizó un estudio en la coordinación General de Tecnologías de la Información y las Comunicaciones de la Asamblea Nacional del Ecuador encargó este estudio para sugerir una arquitectura de software basada en microservicios para la creación de aplicaciones basadas en web. Para lograrlo, esta

investigación utilizó un diseño preexperimental, una metodología cuantitativa y una metodología aplicada. Los resultados de construir e implementar un sistema utilizando el enfoque de microservicios revelan cuán desafiante puede ser la separación funcional sin una comprensión profunda de los procesos constituyentes del sistema y la adopción de métodos que faciliten su aislamiento adecuado. Se determinó que la creación de microservicios requería una experiencia profunda en un área comercial o dominio en particular, lo que lo hacía más adecuado para migrar sistemas más avanzados.

2.1.2. Antecedentes nacionales

Collado (2020) en su estudio buscó conocer qué efecto tendría una arquitectura de microservicios en el aseguramiento de la calidad del software. Para lograrlo, esta investigación utilizó un diseño preexperimental, una metodología cuantitativa y una metodología aplicada. Los datos se recopilaron mediante la firma de un formulario de registro, que luego fue verificado por expertos en la materia. Inicialmente, recibió un resultado de 55,80%, y luego, en el futuro, adquirió un resultado de 72,40%, para un aumento total de 34,91%, aumentando la disponibilidad en 16,60%. Se descubrió que el diseño de microservicios incrementó el aseguramiento de la calidad del software.

Villaizán (2019) en su estudio tuvo como objetivo principal de este proyecto fue crear un modelo para la aplicación web de cobro digital de la Compañía de Sistemas Financieros Per SAC que se basa en una arquitectura basada en microservicios. Para lograrlo, esta investigación utilizó un diseño preexperimental, una metodología cuantitativa y una metodología aplicada. Los hallazgos muestran que la aplicación web de cobro digital de Financial Systems Company SAC implementó de manera efectiva una arquitectura de software basada en microservicios. Se decidió que el uso de una arquitectura basada en microservicios no solo acelera el proceso de desarrollo, sino que también permite el desarrollo del servicio individual de forma aislada.

Ruiz (2020) realizó un estudio sobre la arquitectura de microservicios en SUNAT. División de desarrollo de software aduanero, buscó utilizar el

Modelo Emergente de Microservicios para construir sistemas informáticos más efectivos. Esta investigación utilizó una metodología de estilo aplicado, diseño preexperimental y estrategia cuantitativa. Los hallazgos revelan que el tiempo de reacción del sistema Negocios Aduaneros aumenta en un 75,17%, mientras que la disponibilidad del sistema aumenta en un 20% y su rendimiento aumenta en un 195%. Comparando el Modelo Emergente para la creación de sistemas informáticos eficientes en el Negocio Aduanero, basado en microservicios, con el modelo arquitectónico monolítico, los investigadores encontraron que el primero producía resultados más favorables y eficientes en tiempo de respuesta, disponibilidad y desempeño.

2.1.3. Antecedentes locales

Cabezas (2019) en su estudio necesitaba establecer un sistema de contabilidad basado en la web. Esta investigación utilizó una técnica cuantitativa no experimental descriptiva para lograr sus objetivos. El 64% de los encuestados dijeron que no estaban satisfechos con el estado actual de la gestión contable, y el ochenta por ciento dijo que creía que era necesario adoptar un sistema de gestión contable basado en la web. Estos hallazgos indican una desaprobación generalizada del statu quo y un fuerte apoyo para buscar alternativas.

Visitación (2018) en su estudio se propuso crear un sistema de información basado en la web con la intención de facilitar una administración más eficiente de los puestos de salud de la Red Huaylas Sur. Esta investigación usó una metodología de diseño no experimental, enfoque cuantitativo, y tipo descriptivo para lograr sus objetivos. Como se muestra en los hallazgos, el 65 % de los encuestados en esta encuesta manifestó que el sistema de información web para el procesamiento de elaboración de las historias clínicas de los diferentes puestos de salud de la Red Huaylas Sur Huaraz siempre se completa de manera efectiva. Encontramos que hacia finales de 2018, los centros de salud de la Red Huaylas Sur Huaraz han mejorado el manejo de las historias clínicas de los pacientes.

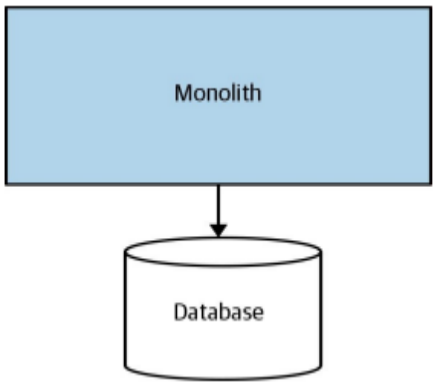
2.2. Bases teóricas

2.2.1. Arquitectura monolítica

Según, Newman (2019) es posible tener numerosas copias de este proceso por redundancia o escalabilidad, pero con un diseño monolítico, todo el código está contenido dentro de un solo ejecutable.

Figura 1

Descripción de una arquitectura monolítica



Fuente: Adaptado del estudio de Newman (2019)

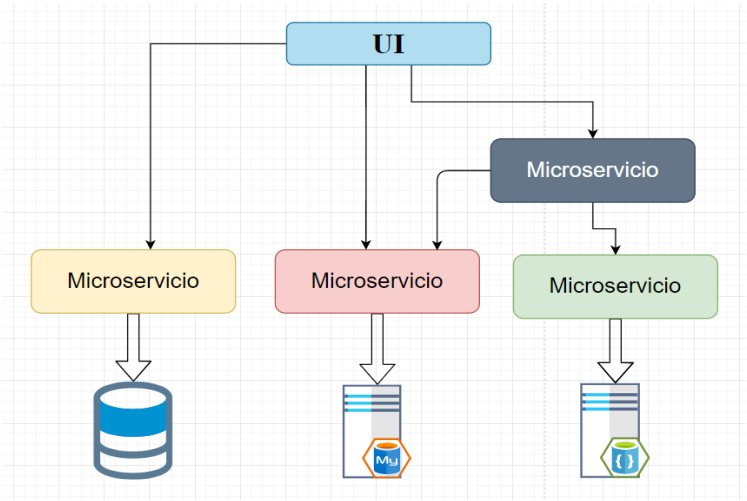
2.2.2. Arquitectura de microservicios

Según Wem, Zomaya y Yang (2020) es aquella en que se realizará una partición en pequeños servicios para transmitirse en forma rápida y seguras, una API de recursos HTTP. Estos servicios están totalmente automatizados. Para ello, se verán las diferencias entre monolítica y un microservicio.

Para, Prasad (2018) el software creado con una arquitectura de microservicios se compone de componentes más pequeños llamados "microservicios". Una función específica y fácilmente articulada es para lo que fueron creados. Además de eso, su ciclo de mantenimiento y desarrollo no debe depender de ninguna otra parte.

Figura 2

Microservicios



Fuente: Elaboración propia

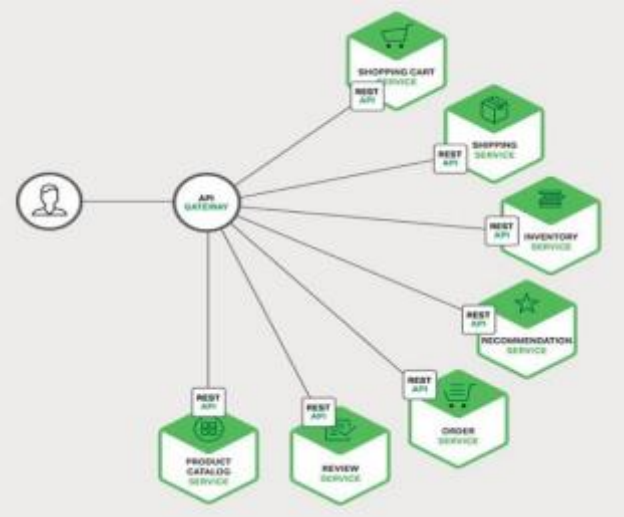
2.2.3. Componentes de la arquitectura

Edge Service/Api Gateway: Según Richardson (2017) mediante el uso de un servicio de enrutamiento, los clientes de una aplicación de microservicios pueden obtener acceso a los muchos servicios proporcionados por el programa a través de un punto de entrada centralizado.

Para (Wolff, 2018) al mapear las URL de cada microservicio y permitir la redirección de solicitudes, este servicio es crucial para aislarlos del tráfico externo.

Figura 3

Utilizando un servicio de enrutamiento

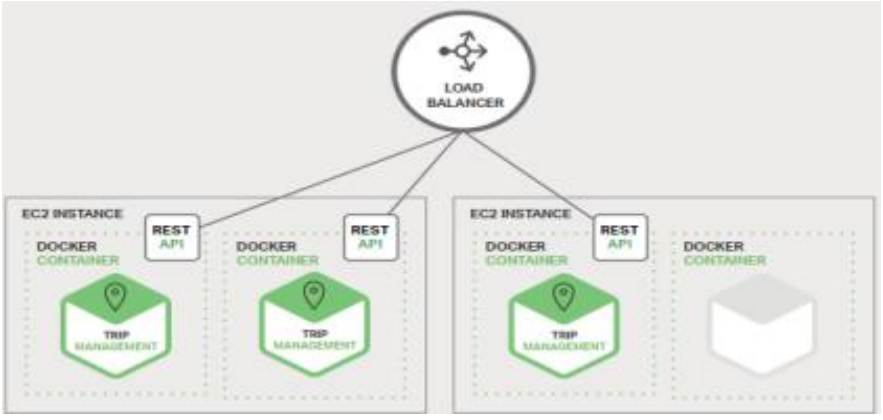


Fuente: Adaptado del estudio de (Richardson, 2017)

Balanceador de carga: Wolff (2018) El balanceador de carga brinda información sobre las instancias de los servicios para calcular la carga de cada instancia, lo que señala es uno de los beneficios de los microservicios, ya que permite escalar cada servicio de forma independiente. Cuando un nodo no responde a una solicitud, el servicio puede eliminarse del balanceador de carga.

Figura 4

Balanceador de carga utilizando Docker

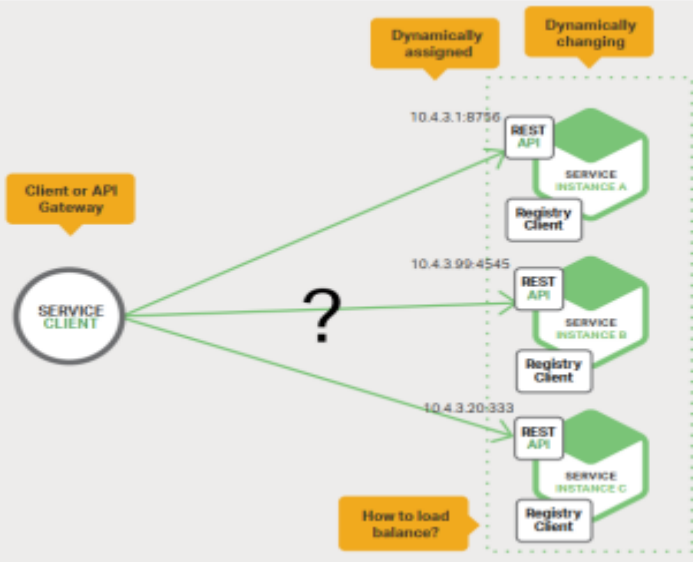


Fuente: Adaptado del estudio de Richardson (2017)

Registro de servicios: Newman (2016) afirma: "Una vez que tiene muchos microservicios en ejecución, es necesario conocer su ubicación física". Por ejemplo, el monitoreo requiere saber qué microservicios se implementan en qué servidores; también, se debe proporcionar algún método para que una instancia pueda registrarse, afirmando "Estoy aquí", y proporcionando un medio para descubrirla después de que se haya registrado. Al usar microservicios, es crucial eliminar periódicamente las instancias antiguas y reemplazarlas por otras nuevas.

Figura 5

Necesidad de Registro de Servicios

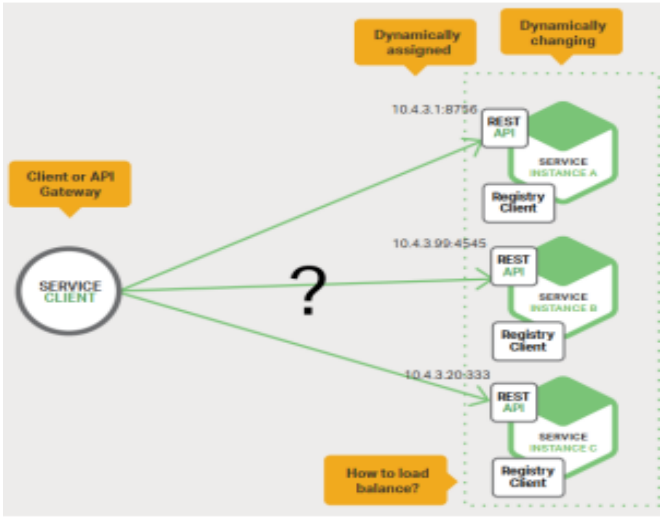


Fuente: Adaptado del estudio de Richardson (2017)

Descubrimiento del lado del cliente: Wolf (2018) el cliente "selecciona tanto la ubicación de red de las instancias de servicio accesibles como el equilibrio de carga entre ellas".

Figura 6

Descubriendo servicios por el lado del cliente

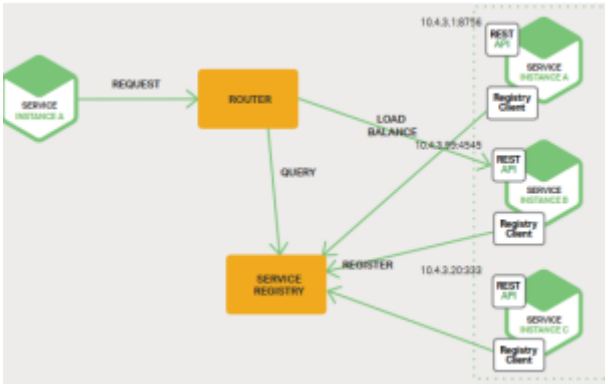


Fuente: Adaptado del estudio de Richardson (2017)

Descubrimiento del lado del servidor: Richardson (2017) señala que otro enfoque de descubrimiento de servicios que analiza es delegar la búsqueda de nuevos servicios a un componente intermediario, como un balanceador de carga.

Figura 7

Descubriendo servicios por el lado del servidor



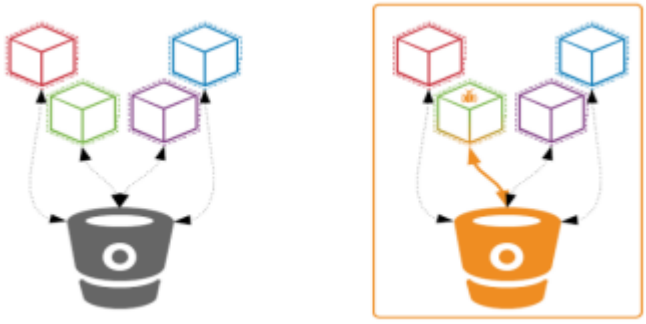
Fuente: Adaptado del estudio de Richardson (2017)

Circuit Breaker: Wolf (2018) según su definición, es un bucle de retroalimentación que se utiliza para reducir la funcionalidad en caso de que falle una llamada al método. El microservicio puede seguir ejecutándose incluso si un servicio dependiente deja de funcionar,

evitando un efecto dominó de falla y permitiendo que el servicio dependiente se recupere.

Figura 8

Circuit Breaker aislando servicio fallido



Fuente: Adaptado del estudio de Richardson (2017)

Logging: Para Newman (2016) "Una aplicación puede entregar rápidamente información sobre los eventos que han ocurrido" es lo que permite el log (registro). Tanto los errores como los valores atípicos estadísticos dignos de mención entran en esta categoría. En conclusión, los desarrolladores pueden usar la información almacenada en los registros para identificar mejor el origen de los problemas. La capacidad de crear e interpretar archivos de registro es muy valorada en el contexto de los microservicios por lo siguiente:

Muchas consultas solo se pueden atender a través de la cooperación de muchos microservicios. Si ese es el caso, no puede reconstruir lo que sucedió usando solo un archivo de registro de microservicios.

Muchas veces, un microservicio puede ejecutar numerosas instancias para distribuir la carga. Como resultado, los datos de un archivo de registro de una sola instancia tienen un uso limitado.

Ya sea por un aumento en la demanda, una actualización o una falla, es inevitable que se establezcan periódicamente nuevas instancias de un microservicio. La información incluida en un archivo de registro puede perderse si se apaga una máquina virtual y luego se extrae su disco duro.

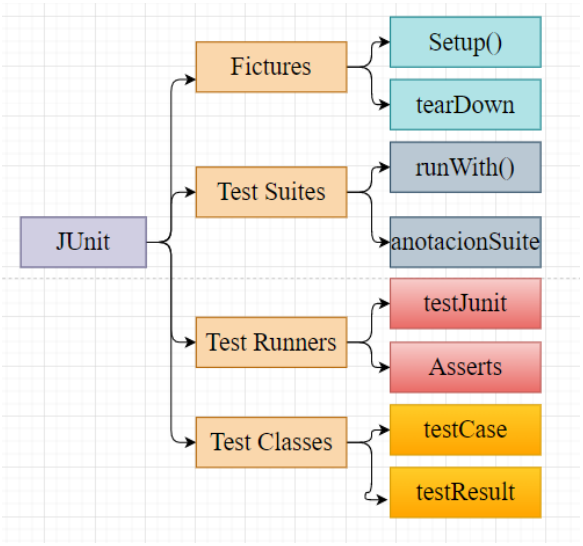
Configuración centralizada: Wolff (2018) “Es fundamental tener un componente de configuración centralizado en un sistema distribuido”, afirma el informe. Al hacerlo, podemos garantizar que cada instancia de los microservicios use su propio conjunto distinto de archivos de configuración (p. 139).

Despliegue y operación: Wolff (2018) dice: "Los objetivos principales de una arquitectura de microservicios son la implementación rápida de servicios y la flexibilidad para adaptar las aplicaciones a medida que evolucionan". El desarrollo de microservicios, en contraste con el desarrollo de aplicaciones monolíticas, implica numerosos avances independientes. La implementación de un microservicio en una máquina virtual con otros podría afectar el funcionamiento de los otros microservicios. La implementación puede causar una tensión adicional en la máquina virtual o provocar otras modificaciones.

Junit Test: Es una solución de software basada en Java para probar si un programa funciona o no según lo previsto. Reduzca el tiempo dedicado a depurar errores y verificar la precisión del código utilizando Junit Test, que se basa en pruebas casi automatizadas. A continuación, se muestra un esquema de las capacidades de Junit:

Figura 9

Pruebas Unitarias



Fuente: Elaboración Propia

Principios Solid:

Solid es un acrónimo de los cinco principios de diseño que están destinados a mejorar la legibilidad, adaptabilidad y mantenibilidad de los sistemas de software.

Solid se basa en los siguientes principios rectores:

SRP: Single Responsibility Principle

Trate de hacer que cada clase sea responsable de un componente específico de la funcionalidad que ofrece el programa, y tenga esa responsabilidad completamente envuelta dentro (también puede decir oculta dentro de la clase.

OCP: Open/Closed Principle

Si puede crear su propia subclase de una clase y modificarla como desee, entonces la clase se considera abierta (agregar nuevos métodos o campos, anular el comportamiento base, etc.).

LSP: Liskov Substitution Principle

Predecir si una subclase sigue siendo compatible con el código que puede haber funcionado con objetos en la superclase es posible gracias a un conjunto de pruebas conocidas como el principio de reemplazo. Debido a que sus clases serán utilizadas por otros y usted no tendrá acceso ni control sobre su código, esta idea es particularmente importante al crear bibliotecas y marcos.

ISP: Interface Segregation Principle

La segregación de la interfaz dicta que las interfaces "gruesas" se dividan en sus partes constituyentes para diseñar experiencias de usuario más detalladas y específicas. Los clientes solo deben utilizar los servicios que realmente necesitan. De lo contrario, la actualización de una interfaz "gorda" afectará a los clientes que no utilicen las operaciones actualizadas.

DIP: Dependency Inversion Principle

No debería haber ninguna dependencia entre los niveles. La abstracción es un requisito previo para cada uno de ellos. Los

detalles no deben utilizarse como base para un resumen. La abstracción es necesaria para los detalles.

ReactiveX y RxJava

Una interfaz de programación de aplicaciones (API) que facilita la administración de flujos de datos y eventos mediante el uso de un híbrido del patrón Observer, el patrón Iterator y los conceptos de programación funcional se llama ReactiveX.

El desarrollo de aplicaciones a veces requiere procesamiento de datos en tiempo real.

Tenemos una API versátil para construir y manipular flujos de datos, gracias a ReactiveX (y su uso de Apreciables y operadores).

También hace que la generación de subprocesos y la concurrencia sean mucho más fáciles de tratar en programas asincrónicos.

Es así como:

RxJava es la implementación de ReactiveX para Java.

Apreciable y Subscriber

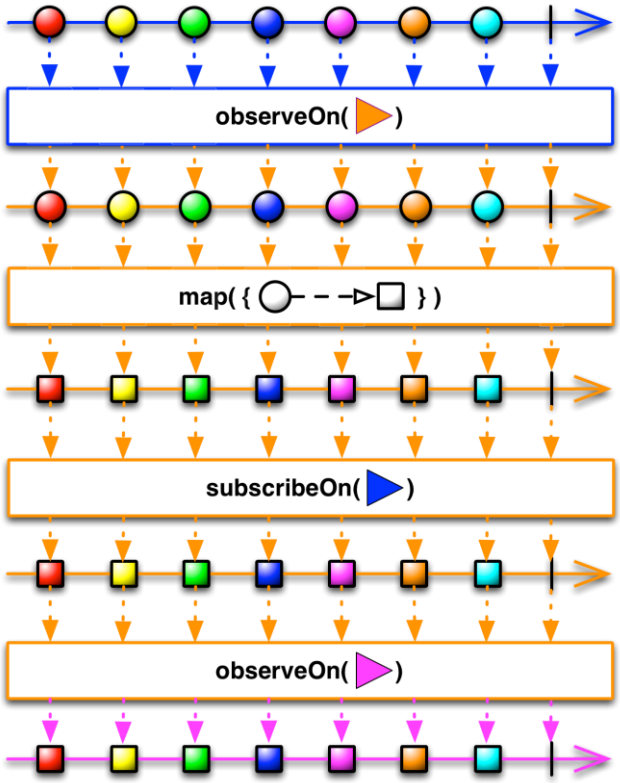
En RxJava, Apreciable es una clase que emite un flujo de datos o eventos. Y Subscriber es una clase que actúa sobre los elementos emitidos.

Operadoras y transformaciones

Una función es un operador si acepta un Apreciable T> como entrada y devuelve otro Apreciable R> como salida, donde T y R no son necesariamente del mismo tipo. Usando operadores, puede encadenar una serie de operaciones para crear gráficos de representación elaborados.

Figura 10

Operadoras y transformaciones



Fuente: (Thornsby, 2017)

Responsivos: Al cumplir SLA predeterminados, garantizan la calidad del servicio.

Resilientes: Mantienen su capacidad de respuesta incluso cuando se enfrentan a condiciones incorrectas.

Elásticos: se mantienen responsivos incluso ante aumentos en la carga de trabajo.

Orientados a mensajes: Al generar interacciones basadas en el intercambio asincrónico de mensajes, reducen la conexión entre componentes.

Principios y Patrones de diseño:

De acuerdo con, Alfonso y Llull (2021) un patrón de diseño se ha utilizado para cumplir con los siguientes objetivos:

Crear código reusable (excelente ventaja)

Evitar perder tiempo en soluciones a problemas ya resueltos o conocidos

Estandarizar el lenguaje entre programadores

Estos actuarán como un estándar por el cual se escribe el código y los problemas se resuelven de una manera más metódica y competente.

2.2.4. Componentes de la arquitectura

Tipos de patrones de diseño

De acuerdo con, Alfonso y Llull (2021) Hay tres categorías principales de patrones de diseño, cada una de las cuales representa una clase distinta de problemas. Los ejemplos de tales frases pueden incluir: Creacionales, estructurales y de comportamiento

a. Patrones creacionales

El objetivo es resolver problemas con instanciación. Proporcionan los medios para asignar la responsabilidad de fabricar objetos según sea necesario. La clave de su diseño es la capacidad de abstraer los detalles específicos de la creación de instancias de clase. En términos generales, podemos clasificarlos como uno de estos:

Singleton:

(Instancia única): asegura que hay exactamente una instancia de una clase dada.

Abstract Factory:

Le da la libertad de definir la interfaz para una clase de objetos relacionados o dependientes sin tener que definir esas clases explícitamente.

Builder:

Simplifica la creación de objetos complejos al mismo tiempo que mantiene la legibilidad y la escalabilidad. Se pone en uso cuando se construye lo mismo una y otra vez.

Factory Method:

Tener objetos reales de cierto tipo alrededor es realmente útil.

Patrones estructurales

El nombre lo delata: se trata de lidiar con los desafíos de la estructura de clases, con un enfoque en cómo se combinan diferentes clases y objetos para formar estructuras más grandes (Alfonso y Llull, 2021).

Proxy:

Este componente introduce un grado de acceso a una clase. Esto se puede hacer ya sea por seguridad o por conveniencia, dependiendo de la situación.

b. Patrones de comportamiento

Admite la solución de problemas que afectan la funcionalidad de la aplicación. Ofrece soluciones a problemas con dependencias e interacciones de clases y objetos (Alfonso y Llull, 2021).

Inyección de dependencias

Dependencia: La inyección es un principio básico del marco Spring y se utiliza para dividir las aplicaciones en partes independientes. No puede haber una creación o administración centralizada de objetos con muchos dependientes. Con contenedores, es posible almacenar centralmente y redistribuir suministros de todo el sistema a sus usuarios finales correspondientes (Alfonso y Llull, 2021).

Utilizar manejadores de dependencias:

Una gestión de dependencias como Maven nos permite mantener el control sobre nuestras aplicaciones Java. Maven es una herramienta de gestión de proyectos que no solo utiliza archivos XML y objetivos específicos para completar tareas como la compilación y el empaquetado de código, sino que también gestiona la obtención de dependencias del proyecto y el almacenamiento del resultado completo en un repositorio (Alfonso y Llull, 2021).

Swagger

Swagger es un marco de software de código abierto para crear servicios web RESTful y documentarlos, implementarlos y consumirlos.

API

Aplicación de interfaz de software. El término "aplicación" se utiliza para describir cualquier software con un propósito específico en el contexto de las interfaces de programación de aplicaciones. Puede pensar en la interfaz como una especie de acuerdo de servicio entre sus programas.

Open API

OpenAPI (OAS) es un lenguaje para hablar, construir y mostrar servicios web RESTful. En pocas palabras, es un conjunto de reglas que rigen el formato y la sintaxis de las API REST. No importa qué idioma uses para codificar. Tanto las máquinas como las personas pueden conocer las características del servicio de esta manera, sin tener que buscar en manuales, estudiar el código fuente o monitorear el tráfico de la red.

2.2.5. Componentes de la arquitectura

Patrones de despliegue de microservicios

Múltiples servicios por host: Este paradigma se usa cuando no se requiere (o no se desea) ejecutar cada microservicio en su propia zona de pruebas. Una máquina física o virtual puede servir como host. Este paradigma es ineficiente en una arquitectura de microservicios múltiples porque no puede escalar (Newman, 2016).

Servicio por host: Cada microservicio individual en esta arquitectura está separado de los demás por la propia máquina. La presencia de varios microservicios hace que este enfoque sea ineficiente. Los recursos se están despilfarrando. (Richardson, 2017).

Servicio por máquina virtual: Esta estrategia es una mejora con respecto a la anterior, ya que la máquina virtual separa cada microservicio, pero aún no escala bien cuando hay numerosos microservicios involucrados. Se requiere hardware de alto rendimiento para que varias máquinas virtuales puedan compartir un único host físico. (Wolff, 2018).

Servicio por contenedor: Como modelo de implementación, esta es la opción preferida y la más popular. Se utilizan contenedores independientes para la implementación de cada microservicio. El resultado es un microservicio más flexible y escalable (Richardson, 2017).

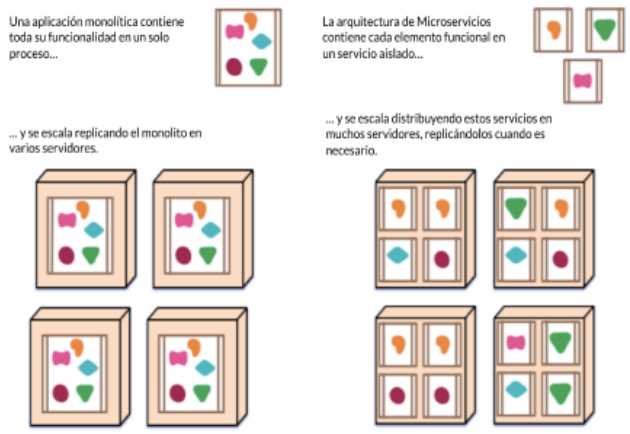
Tiempo de respuesta: Según Wem, Zomaya y Yang (2020) es la capacidad que tendrá para brindar la información requerida por el usuario esta debe ser en forma inmediata para no demorar otras actividades.

Disponibilidad de servicios internos: Acorde con Wem, Zomaya y Yang (2020) Todos los algoritmos utilizados por el sistema son propiedad de la empresa, lo que significa que deben mantenerse en secreto para proteger los datos confidenciales de cualquier intento de espionaje.

Disponibilidad de servicios externos: Para Wem, Zomaya y Yang (2020) Servicios ofrecidos por este sistema a sus usuarios con la intención de satisfacer la necesidad de ciertos tipos de datos de esos usuarios.

Figura 11

Arquitectura monolítica vs microservicios



Fuente: Adaptado del estudio de Richardson (2017)

2.2.6. Mejorar la calidad de software

Según Galin (2018) La madurez del proceso de software es la capacidad de generar productos de suficiente calidad, ofrecer los servicios requeridos

por los usuarios y cumplir con los requisitos de desarrollo y mantenimiento del software a través de un conjunto estandarizado de etapas. presupuesto.

Por otra parte, Pressman (2014) comprende una formación en compromisos de aseguramiento, aseguramiento de la calidad y gestión, habilidades competentes de ingeniería de software, gestión de producción del esfuerzo de desarrollo de software e interrupciones, monitoreo de tareas para asegurar la validación de patrones en la elaboración, y herramientas de medición e informes.

Eficiencia

De acuerdo con Galin (2018) Es el punto en el que el programa hace el mejor uso de los recursos del sistema (tiempo y dinero).

Según (Laurentis, 2018) para ser eficiente, se debe hacer un uso eficaz de las herramientas a su disposición para llevar a cabo las funciones que se le asignan.

Confiabilidad

Acorde con Galin (2018) en otras palabras, el desarrollo basado en datos previos se evalúa y calcula directamente para determinar la probabilidad de que un programa de computadora funcione sin fallas en un entorno determinado durante un tiempo determinado. Durante ese período, puede utilizar todas las funciones de la aplicación sin tener que pagar nada.

Integridad

Según Galin (2018) es la probabilidad de que un programa opere de acuerdo con los requerimientos en un momento determinado de tiempo sea difícil de extraer la información y de ser necesario realice en forma automática la expulsión del agente agresor o extraño al sistema.

2.2.7. Confidencialidad de la información

Según Estela (2019) para garantizar que se respete la privacidad de las personas y que sus datos confidenciales no se compartan sin su conocimiento y permiso. Para brindar esta garantía, existen procedimientos para restringir quién puede ver estos datos. Debido en gran parte al hecho de que el atacante descubrió la debilidad en el sistema, los piratas informáticos se ven obligados a robar nuestra información debido al uso generalizado de Internet para transacciones financieras en la actualidad.

2.3. Definición de términos

- **Calidad**:

  El cumplimiento de los criterios de calidad, ya sean implícitos o explícitos, es lo que se entiende por el término "calidad" (Laurentis, 2018)

- **Información:**

  Una recopilación de datos significativa es aquella que ayuda a aclarar una situación o ampliar la comprensión de un tema. En realidad, la información es un mensaje relevante en un contexto específico que se puede usar de inmediato para ayudar a tomar decisiones y realizar acciones posteriores (Chiavenato, 2006).

- **Sistema:**

  Los sistemas son grupos de partes interconectadas que tienen objetivos comunes y tienden a actuar en concierto para lograr esos objetivos, definidos por la existencia de vínculos íntimos entre esas partes y el mantenimiento de la cohesión general del sistema (Laurentis, 2018)

- **Seguridad de la información:**

  Es un conjunto de medidas de seguridad proactivas y reactivas adoptadas por las empresas para mantener sus datos seguros y protegidos, con el objetivo de mantener intacta su integridad (Frayssinet, 2016).

- **Web:**

  La World Wide Web (WWW) es una colección de documentos (sitios web) accesibles a través de Internet y enlazados entre sí mediante enlaces de hipertexto. Generalmente se acepta que "hipertexto" se refiere a un documento que contiene no solo texto sino también imágenes y otros archivos multimedia (Latorre, 2018).

- **Tiempo:**

  Es todo intervalo en el cual suceden hechos voluntarios o involuntarios (Estela, 2019)

- **Sistema bancario:**

  son entidades encargadas de administrar dinero de los clientes bajo mecanismos regulados. (Laurentis, 2018)

- **Software:**

  El software de un sistema informático, junto con los datos, procesos y reglas asociados, que permite que el sistema lleve a cabo una variedad de actividades (Estela, 2019).

2.4. Hipótesis

2.4.1. Hipótesis general

La Calidad Del Software en una entidad bancaria en Lima metropolitana mejora significativamente con la arquitectura de Microservicios.

2.4.2. Hipótesis especificas

- La arquitectura de microservicios incide en la eficiencia del *software* en una entidad bancaria Lima Metropolitana.
- La arquitectura de microservicios incide en la confiabilidad del *software* en una entidad bancaria Lima Metropolitana.
- La arquitectura de microservicios incide en la integridad del *software* en una entidad bancaria Lima Metropolitana.

2.5. Variables

Variable independiente

Arquitectura de Microservicios

Variable dependiente

Mejora de la calidad del software

2.6. Operacionalización de variable

Tabla 1

Operacionalización de variables

VARIABLE	DEFINICIÓN CONCEPTUAL	DIMENSIONES	INDICADORES	ESCALA DE MEDICIÓN
X: VARIABLE INDEPENDIENTE Arquitectura de Microservicios	Según Estela (2016) es un conjunto de herramientas tecnologías, procedimientos y recursos que se enfocan en el control financiero, basado en las tecnologías tradicionales con el uso de n-capas ejecutándose en un mismo contexto.	Facilidad	• Tiempo de aprendizaje • Tiempo de despliegue	Medición: Likert
		Disponibilidad	• Interna • Externa	Medición: Likert
		Flexibilidad	• Adaptación • Apoyo • Garantía	Medición: Likert
		Agilidad	• Innovación • Mejora continua	Medición: Likert
Y: VARIABLE DEPENDIENTE Mejora de la calidad del software	Según mencionan Sánchez, Comas y García (2019), que los procesos de software sean adecuados para desarrollar una calidad adecuada en los productos de software, para sus servicios de operación esperados y para cumplir con los requisitos de programación y mantenimiento del presupuesto se establece mediante una estandarización de los pasos que componen el proceso de software.	Eficiencia	• Rendimiento • Respuesta rápida • Recursos	Medición: Likert
		Confiabilidad	• Disponibilidad • Usabilidad	Medición: Likert
		Integridad	• Procesar • Almacenar	Medición: Likert

Fuente: Elaboración propia

III. METODOLOGÍA

3.1. Tipo de estudio

El presente de trabajo de investigación se tipifica como investigación cuantitativa y aplicada.

3.2. El diseño de investigación

Fue pre experimental porque se trabajó solo con un grupo con un pretest y postest.

O1->x -> O2

O1: pre test del grupo preexperimental

X: aplicación de microservicios

O2: post test del grupo preexperimental

3.3. Descripción de la unidad de análisis población y muestra

3.3.1. Unidad de análisis

Trabajadores de la BCP Sede Chorrillos – Lima

3.3.2. Población

Según Carrasco (2016) los componentes que se investigan se agrupan porque tienen o conservan propiedades comunes. La población del presente proyecto de investigación estará definida por los empleados de la organización BCP Sede Chorillos – Lima, conformada por la tribu de Canales digitales exactamente 158 empleados.

3.3.3. Muestra

Según Carrasco (2016) la muestra es un pequeño subconjunto de la población en la que se analizara las similitudes y el comportamiento en un determinado contexto. Para este proyecto de investigación se obtendrá la muestra mediante el uso de la siguiente formula por ser una población definida.

$$n = \frac{N * Z^2 * p * q}{e^2 * (N-1) + Z^2 * p * q}$$

Donde:

$n = Muestra$

$Z = 1.96\ (Nivel\ de\ confianza\ al\ 95\%)$

$p = 0.5\ (Proporción\ del\ éxito\ 50\%)$

$q = 0.5\ (Proporción\ del\ fracaso\ 50\%)$

$e = Margen\ de\ error\ o\ precisión\ 0.05$

$N = Población$

Obteniendo como resultado que nuestra muestra será de 112 trabajadores

3.3.4. Muestreo

Según Carrasco (2016) Para identificar la población de la cual se va a recopilar la información se usa el Muestreo Probabilístico (Aleatorio). Este método de muestreo satisface nuestros requisitos tanto de representatividad de los datos como de usabilidad del diseño del software al permitir la inclusión de toda la población en la muestra.

3.4. Técnicas de instrumentos de recolección de datos

Técnicas: Se utiliza para comparar porcentajes con puntuaciones medias (si se utiliza el método de tipo cuantitativo) o realizar análisis de contenido si se trata de un método cualitativo (Carrasco, 2016).

Observación (directa o indirecta): Para el presente proyecto se realizará en ambos casos, directa porque el investigador estará presente en el lugar de los hechos para realizar las observaciones de cada uno de los procesos involucrados para la investigación, por otro lado, indirecta porque se va a revisar documentaciones respecto al desarrollo de arquitectura de microservicios en diferentes proyectos por equipo de desarrollo.

Entrevista: El investigador formulará una serie de preguntas dirigidas a los empleados que integran en la entidad bancaria especialmente de la tribu de canales digitales de manera presencial y virtual, estableciendo un dialogo peculiar, de esta forma se podrá recolectar la información necesaria para llevar a cabo el proyecto de investigación.

Encuesta: El instrumento que se utilizó para esta tesis es un cuestionario de 22 preguntas, lo cual permitió realizar una comparación entre pretest y postest para

medir la calidad de software en la entidad bancaria de lima metropolitana. El modelo de cuestionario aplicado se encuentra en el **ANEXO 2**.

3.5. Técnicas de análisis y prueba de hipótesis

3.5.1. Técnicas de análisis

El estudio incluyó una mirada a los períodos de tiempo antes, durante y después de la implementación de la "arquitectura de microservicios". Utilizando el programa estadístico SPSS V 26, se realizó la prueba de Wilcoxon para confirmar estas hipótesis ya que su propósito es proporcionar evidencia para la elección de "aceptar" o "rechazar" la hipótesis.

3.5.2. Prueba de hipótesis

Los resultados esperados en esta investigación se orientan al desarrollo de una arquitectura de microservicios que busca mejorar la calidad del software en una entidad bancaria Lima Metropolitana, mejorando su eficiencia, confiabilidad e integridad de este.

Para la contrastación de la hipótesis se verifico que los datos no siguieron una distribución normal, por lo cual se aplicó la prueba de Wilcoxon con un nivel de significancia del 5% ($p < 0.05$).

IV. RESULTADOS DE LA INVESTIGACIÓN

4.1. Descripción del trabajo de campo

En esta sección se incluye el trabajo de campo realizado conducente a la obtención de los resultados de la investigación científica.

4.1.1. Análisis de la situación actual

La entidad bancaria en estudio está organizada jerárquicamente de acuerdo con el organigrama presentado en el Anexo 5. Dentro de la organización el Área de Canales digitales es la elegida para la presente investigación:

El área de canales digitales tiene a cargo un módulo de site administrativo que por el momento aún se encuentra construida con una arquitectura monolítica.

El módulo de site administrativo cuenta con más de 15 funcionalidades que necesitan ser migradas a una arquitectura de microservicios.

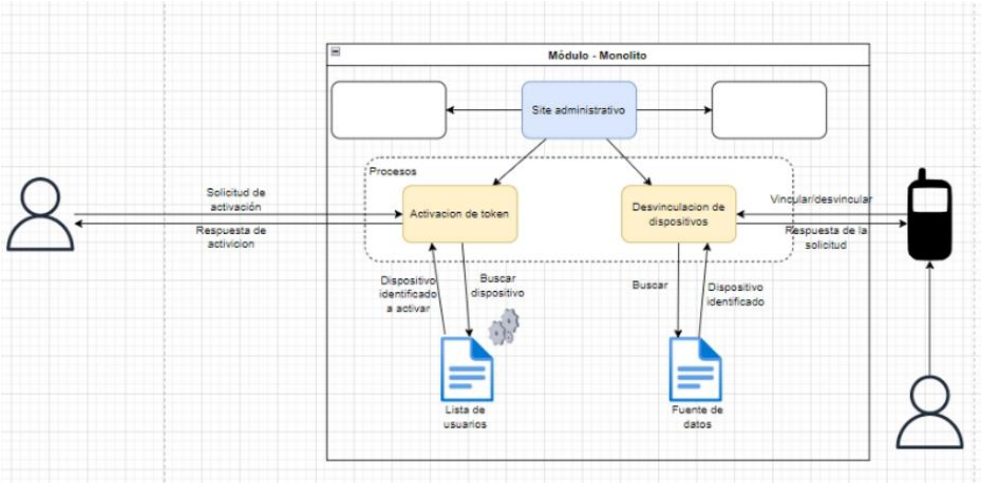
Para solucionar y aplicar las nuevas tecnologías de arquitectura de microservicios cumpliendo con los objetivos trazados. Se Inicia por las funcionalidades de activación de token digital y desafiliación de dispositivos.

Las reglas de negocio fueron creadas y examinadas para cada una de las operaciones internas del microservicio con la visión de negocio, el propósito, el alcance y las oportunidades de la organización como punto focal. sugerencia, una situación hipotética que permitió una descripción mundial y ayudó a explicar el punto de vista de la investigación. Luego se continuó modelando los casos de uso comercial, el diagrama de actividad por caso de uso comercial para representar las actividades y el flujo de trabajo que sigue en cada uno de los procesos identificados, y el modelo de objeto comercial para identificar los objetos comerciales y tener una imagen completa.

A continuación, se presenta el modelo actual:

Figura 12

Descripción de una arquitectura monolítica



Fuente: Elaboración propia

4.1.2. Análisis del organigrama funcional estratégico

El área de Tribus Canales Digitales, tiene la siguiente estructura referenciada en el organigrama funcional:

Tabla 2

Organigrama funcional estratégico

Productos(PRD)			
Tarjeta de crédito	Tarjeta de débito	Seguros	Cuentas
Adelanto de sueldo	Débito automático		
Servicios(SRV)			
Soluciones de pago	Abono por contacto	Transferencias	Envío de estado de cuenta
Gestión de favoritos	Actualización de datos	Disposición de efectivo	Giros nacionales
Afiliación a notificaciones	Gestión de reclamos	Pago de servicios	Configuración de tarjetas
Transversales(PRT)			

Consulta de productos y movimientos	Envío de notificaciones	Consulta de productos preferentes	Actualización de datos
Iniciativas(INC)			
Tarjeta digital – Visualización Credenciales TDD	Tarjeta digital – Materialización TDD	Tarjeta digital- Reposición TDD	Tarjeta digital – Visualización de credenciales TC.
Site administrativo			

Fuente. Elaboración propia

4.1.3. Requerimientos, procesos y caso de uso de negocio para el site administrativo

Requerimientos

a) Requerimientos Funcionales

Tabla 3

Requerimientos funcionales

N°	Descripción	Micro Servicio
REQ-FUN-01	El programa de acceso deberá permitir autenticarse en todos los módulos de la entidad financiera; a fin de tener una sola interfaz para poder ingresar.	-User-access-permission
REQ-FUN-02	El programa debe validar los datos de acceso del usuario (email, nombre de usuario, password,etc)	-User-access-permission
REQ-FUN-03	El programa debe negar el acceso a las personas no autorizadas	-User-access-permission
REQ-FUN-04	El programa debe permitir recuperar la clave de acceso, mediante un link de recuperación u otro medio.	-User-access-permission

REQ-FUN-05	El sistema debe permitir el ingreso de datos de los clientes de la entidad financiera.	-Client-service (cliente)
REQ-FUN-06	El sistema debe permitir la modificación de los datos de los clientes de la entidad financiera.	-Client-service (cliente)
REQ-FUN-07	El sistema debe permitir la eliminación de los datos de los clientes de la entidad financiera. (Eliminado lógico)	-Client-service (cliente)
REQ-FUN-08	El sistema debe permitir la búsqueda del código del cliente para la generación del token digital	- Digital-token-activation
REQ-FUN-09	El sistema debe permitir activar el token digital y enviárselo al cliente	- Digital-token-activation
REQ-FUN-10	El sistema debe contar con una lista de solicitudes pendientes de clientes para activar los tokens digitales respectivos	- Digital-token-activation
REQ-FUN-011	El sistema debe permitir cancelar la solicitud de un token digital (por el cliente ó por el usuario administrativo del sistema)	- Digital-token-activation
REQ-FUN-12	El sistema validará el token digital y emitirá una respuesta.	- Digital-token-activation
REQ-FUN-13	El sistema permitirá que el cliente vincule su dispositivo para poder acceder a los módulos e interactuar con ellos.	- Device-enrollement
REQ-FUN-14	El sistema permitirá que el cliente desvincule su dispositivo para impedir que acceda a los módulos e interactuar con ellos.	- Device-enrollement
REQ-FUN-15	El sistema permitirá que el usuario administrador desvincule el dispositivo del cliente para impedir acceder a los módulos y que interactúe con ellos.	-Device-enrollement

b) Requerimientos No funcionales

Tabla 4

Requerimientos no funcionales

N°	Nombre	Descripción
REQ-NOFUN-01	Eficiencia	El sistema deberá responder al pedido de autenticación del cliente en 2 segundos aproximadamente
REQ-NOFUN-02	Eficiencia	El sistema debe soportar el registro de 1 000 transacciones y filtros de selección por minuto.
REQ-NOFUN-03	Eficiencia	El sistema debe responder una lista de 100 registros en un tiempo máximo de 10 segundos.
REQ-NOFUN-04	Fiabilidad	El sistema deberá considerar alta disponibilidad, es decir 7 * 24 los 365 días del año para las transacciones de registros de datos y filtros de selección en los repositorios.
REQ-NOFUN-05	Fiabilidad	Para rastrear el flujo del sistema y detectar cualquier punto de error o inconsistencia, debe generar un REGISTRO con muchas capas de visibilidad. Este REGISTRO es donde debe registrar los detalles de cualquier falla del sistema que haya ocurrido.
REQ-NOFUN-06	Fiabilidad	El sistema debe poder ser monitoreado, de modo que se pueda alertar sobre algún incidente o problema que se presente.
REQ-NOFUN-07	Fiabilidad	El sistema deberá mantener el nivel especificado de rendimiento en caso de falla del mismo.
REQ-NOFUN-08	Seguridad	Se necesitan controles transaccionales en el sistema para garantizar que todos los datos asociados con una transacción realizada se registren o generen. Esto significa que no debe haber ningún registro sin terminar o incompleto de las transacciones financieras que estaban en curso. Todos los cambios se deben deshacer si alguno de ellos falla.
REQ-NOFUN-09	Seguridad	El sistema deberá implementar la trazabilidad; es decir, se guarda la información de auditoría. Datos sensibles acerca del cliente, token, modelos de teléfono, etc.
REQ-NOFUN-10	Coexistencia	Los servicios deben ser construidos utilizando los estándares de desarrollo definidos en la entidad financiera, de la misma forma deberá instalarse sobre la plataforma definida por la entidad financiera
REQ-NOFUN-11	Confiabilidad - Tolerancia a fallas	Se debe tener en cuenta la cantidad de intentos y el intervalo de reintentos según lo siguiente: TÓPICOS DE PROCESOS

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>Evento</td><td>Error de conectividad con la BD</td><td>Errores no controlados(*)</td></tr>
<tr><td>Cant. De intentos</td><td>Ilimitado</td><td>1</td></tr>
<tr><td>Intervalo de reintentos</td><td>30 segundos</td><td></td></tr>
<tr><td>Tiempo máximo de reintentos</td><td>150 segundos</td><td></td></tr>
<tr><td>Reenvío al tópico</td><td>NO</td><td>Si</td></tr>
</table>
(*) Un error no controlado ocurre por error en el JSON enviado por el trigger en el que el MS no puede parsearlo.

NOTA:

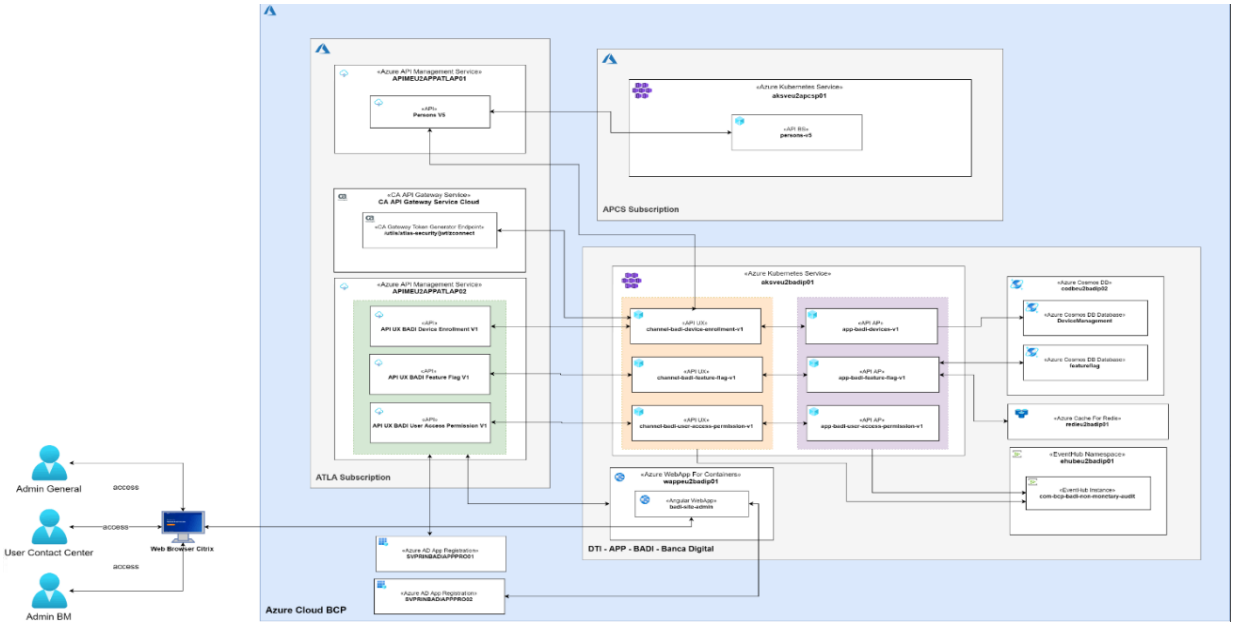
- Por errores de conectividad, el MS tendrá intervalos de reintentos de 30 segundos hasta llegar al segundo 150 en donde se mantiene fijo. Ejemplo: escala de reintentos: 30, 60, 90, 120, 150, 150, 150, 150 (fijo en adelante)
Los errores enviados al tópico de reproceso deberán ser corregidos manualmente para su posterior reenvío.
</td></tr>
<tr><td>REQ-NOFUN-12</td><td>Rendimiento</td><td>ADECUAR Tiempo máximo de actualización de información, entre la generación de información personalizada hasta las colecciones: 5 minutos.</td></tr>
<tr><td>REQ-NOFUN-13</td><td>Rendimiento</td><td>ADECUAR Carga promedio mensual de actualización hacia las colecciones: 300,000 actualizaciones.</td></tr>
</table>

4.1.4. Procesos internos del negocio

Diagrama de Arquitectura

Figura 13

Descripción de una arquitectura monolítica



Fuente: Entidad bancaria

4.1.5. Casos de uso

Especificación de los casos de uso Actores del Sistema

Tabla 5

Actores del Sistema

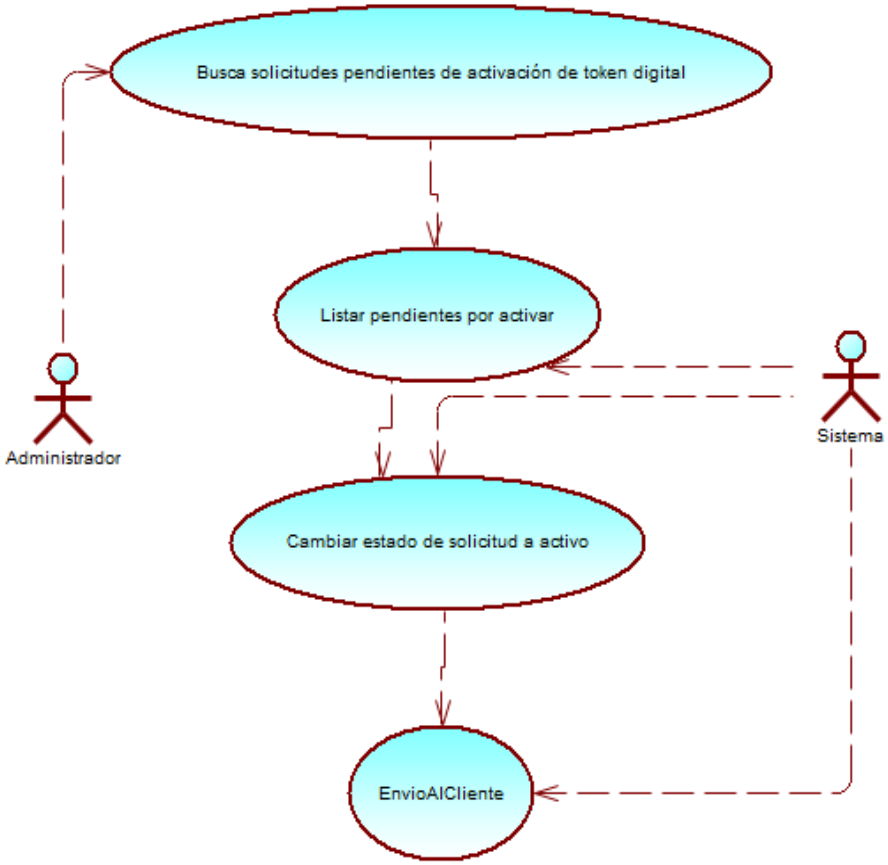
N°	Actor del sistema	Descripción
1	Administrador	Funcionario con rol de" administrador", quien grabará los atributos desde una interfaz para la autenticación, tokens, vinculación y desvinculación de dispositivos.
2	Sistema	Sistema que invoca a las funcionalidades implementadas en los micro servicios a desarrollar de la entidad financiera.
3	Tiempo	Usuario que determina el inicio de la ejecución de un proceso automático.
4	Cliente	Persona que solicita y/o accede a los diferentes módulos de la entidad financiera

Fuente: Elaboración propia

Diagrama de Casos de Uso

Figura 14

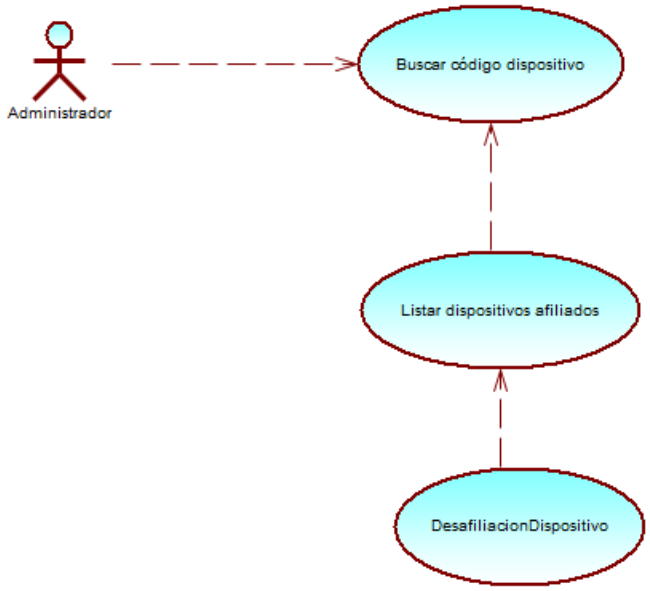
CUS03: Activación de Token Digital



Fuente: Elaboración propia

Figura 15

CUS04: Desafiliación de dispositivo



Fuente: Elaboración propia

Especificación de los Casos de Uso del Sistema

Tabla 6

Especificación de los Casos de Uso N° 01: Activación de Token Digital

<table>
<tr><td colspan="2">1.Nombre del caso de uso del sistema</td><td>CUS01: Activación de Token Digital</td></tr>
<tr><td colspan="3">2.Descripción del caso de uso</td></tr>
<tr><td colspan="3">El presente caso de uso tiene por objetivo implementar un mecanismo de autenticación en el sistema; el cuál, deberá ser confiable, seguro y eficiente. Mediante la activación del token por parte del administrador, como uno de los mecanismos de seguridad, el cliente podrá acceder a los diferentes módulos de la entidad financiera y realizar las operaciones que le son permitidas de acuerdo a su nivel de acceso y seguridad.</td></tr>
<tr><td colspan="3">3.Actores</td></tr>
<tr><td colspan="3">Administrador, Sistema, Cliente</td></tr>
<tr><td colspan="3">4.Precondiciones</td></tr>
<tr><td colspan="3">- El Administrador deberá haber iniciado sesión a través de su nombre de usuario, password
- Por lo menos un Cliente deberá haber solicitado la activación del token digital a través del app o página web.
- El Sistema deberá tener acceso a la colección de clientes de la entidad financiera y de solicitudes</td></tr>
<tr><td colspan="3">5.Postcondiciones</td></tr>
<tr><td colspan="3">De activarse el token digital, al cliente se le concederá el acceso a los módulos del sistema o se le denegará</td></tr>
<tr><td colspan="3">6.Flujo de eventos</td></tr>
<tr><td>Nro.</td><td>Acción del actor</td><td>Respuesta del Sistema</td></tr>
<tr><td>1</td><td>El administrador requiere la visualización de las solicitudes de activación de token digital.</td><td>El sistema valida la identidad del administrador mediante la comprobación de sus datos de inicio de sesión. El sistema concederá o denegará el acceso al usuario administrador. De ser correctos los datos, se devolverá la lista de solicitudes pendientes de activación de tokens digitales. De no ser correctos los datos se devolverá un código de usuario no válido y el flujo terminará en el punto</td></tr>
<tr><td>2</td><td>El administrador selecciona una solicitud y procede a activarla.</td><td>Si la solicitud tiene estado “pendiente” el sistema cambiará el estado a “activo” con lo cual el token digital será activado.

El sistema procederá a enviar el token digital al cliente.</td></tr>
</table>

3	El cliente recibirá el token digital y podrá realizar sus operaciones.	Con el token validado, el cliente procederá a efectuar sus operaciones dentro de la entidad financiera. El flujo continúa en el punto 4.
4		FIN DEL FLUJO
7.Flujos alternativos		
No Aplica		
8.Excepciones		
Nro.	Descripción	
1001	El campo “usuario” no ha sido enviado o es vacío	
1002	Solo se permiten tipo de datos numérico para el campo “usuario”	
1003	El USUARIO ingresado no existe o no es válido	
1010	El campo "password" no ha sido enviado o es vacío	
1011	El password tiene más de 20 caracteres y/o contiene caracteres extraños	
1012	El password suministrado es incorrecto	
1020	El campo “token” no ha sido enviado o es vacío	
1021	El token contiene caracteres extraños	
1022	El token no existe y/o es incorrecto	
9.Requisito asociado (Funcional, no funcional)		
REQ-FUN-08, REQ-FUN-09, REQ-FUN-10, REQ-FUN-12, REQ-FUN-01, REQ-FUN-05, REQ-FUN-06, REQ-FUN-07, REQ-FUN-09, REQ-FUN-10		
10. Prototipo de interfaz de usuario		
No aplica		

Fuente: Elaboración propia

Tabla 7

Especificación de los Casos de Uso N° 02: Desafiliación de Dispositivo

<table>
<tr><td colspan="2">1.Nombre del caso de uso del sistema</td><td>CUS02: Desafiliación de Dispositivo</td></tr>
<tr><td colspan="3">2.Descripción del caso de uso</td></tr>
<tr><td colspan="3">El presente caso de uso tiene por objetivo implementar la desafiliación de un dispositivo de propiedad de un cliente, por parte del administrador.</td></tr>
<tr><td colspan="3">3.Actores</td></tr>
<tr><td colspan="3">Administrador, Sistema, Cliente</td></tr>
<tr><td colspan="3">4.Precondiciones</td></tr>
<tr><td colspan="3">- El Administrador deberá contar con acceso al sistema.
- El Administrador deberá haber iniciado sesión a través de sus datos de acceso.
- El Sistema deberá tener acceso a la colección de clientes y dispositivos de los clientes.</td></tr>
<tr><td colspan="3">5.Postcondiciones</td></tr>
<tr><td colspan="3">Se desvinculará un dispositivo del cliente, con el que ya no podrá realizar sus operaciones en le entidad financiera, según sea el caso.</td></tr>
<tr><td colspan="3">6.Flujo de eventos</td></tr>
<tr><td>Nro.</td><td>Acción del actor</td><td>Respuesta del Sistema</td></tr>
<tr><td>1</td><td>El administrador busca el dispositivo por el código.</td><td>El sistema procesa la solicitud de búsqueda y valida los datos del usuario. Si los datos son inválidos lanzará la excepción de acuerdo a la sección de excepciones del presente caso de uso y terminará el flujo; por el contrario, si son correctos permitirá el acceso.</td></tr>
<tr><td>3</td><td>El administrador solicita una lista de dispositivos afiliados.</td><td>El sistema selecciona y devuelve un listado de todos los dispositivos afiliados del cliente.</td></tr>
<tr><td>4</td><td>El administrador solicita desvincular un dispositivo</td><td>El sistema evalúa la solicitud de desvinculación, valida los datos del usuario administrador y del dispositivo del cliente. Si los datos son inválidos lanzará la excepción de acuerdo a la sección de excepciones del presente caso de uso y terminará el flujo; por el contrario, si son correctos procederá con la desvinculación del dispositivo de la colección de dispositivos del cliente. Eliminará lógicamente el dispositivo del cliente, cambiándole el estado a inactivo.</td></tr>
<tr><td>5</td><td></td><td>FIN DEL FLUJO</td></tr>
<tr><td colspan="3">7.Flujos alternativos</td></tr>
</table>

No Aplica	
8.Excepciones	
Nro.	Descripción
1001	El campo “usuario” no ha sido enviado o es vacío
1002	Solo se permiten tipo de datos numérico para el campo “usuario”
1003	El USUARIO ingresado no existe o no es válido
1010	El campo "password" no ha sido enviado o es vacío
1011	El password tiene más de 20 caracteres y/o contiene caracteres extraños
1012	El password suministrado es incorrecto
1020	El campo “token” no ha sido enviado o es vacío
1021	El token contiene caracteres extraños
1022	El token no existe y/o es incorrecto
1023	El dispositivo no pudo afiliarse
1024	El dispositivo no pudo desafiliarse
9.Requisito asociado (Funcional, no funcional)	
REQ-FUN-13, REQ-FUN-14, REQ-FUN-15, REQ-FUN-01, REQ-FUN-05, REQ-FUN-06, REQ-FUN-07, REQ-FUN-09, REQ-FUN-10	
10. Prototipo de interfaz de usuario	
No aplica	

Fuente: Elaboración propia

Tabla 8

Backlog del Producto

Nro.	USER STORIES	ESTADO	ESTIMACIÓN		CRITERIO DE ACEPTACIÓN	COMENTARIO	TIPO EPICA	PRIORIDAD	RESPONSABLE	SPRINT
			PTOS.USER	HORAS						
1	El sistema debe permitir la búsqueda del código del cliente para la generación del token digital	Terminado	40	20	El sistema deberá devolver verdadero o falso como resultado de buscar el código del cliente.	Puede devolver verdadero o falso o un código de valor entero. Considerar	Historia	5	ScrumTeam01	1
2	El sistema debe permitir activar el token digital y enviárselo al cliente	Terminado	20	20	El sistema registrará, activará y devolverá el token digital al cliente para que pueda utilizarlo en sus operaciones.	Definir las características del token digital, formato, duración, etc.	Historia	4	ScrumTeam01	1
3	El sistema debe contar con una lista de solicitudes pendientes de clientes para activar los tokens digitales respectivos	Terminado	20	20	El sistema deberá almacenar cada una de las solicitudes de los clientes para activar sus tokens digitales.	Definir los datos mínimos que deberá guardar la lista de solicitudes a fin de implementar trazabilidad.	Historia	4	ScrumTeam01	1

4	El sistema debe permitir cancelar la solicitud de un token digital (por el cliente ó por el usuario administrativo del sistema)	Terminado	20	20	Deberá existir una solicitud a cancelar por parte del cliente o del usuario administrativo. La solicitud cancelada deberá tener el estado cancelado.	Definir los estados de la solicitud.	Historia	4	ScrumTeam01	1
5	El sistema validará el token digital y emitirá una respuesta.	Terminado	40		La validación del token digital sólo emitirá una respuesta. True si se pudo validar o False si no se pudo.	Considerar si existe otra propuesta de respuesta.	Historia	5	ScrumTeam01	2
6	El sistema permitirá que el cliente afilie su dispositivo para poder acceder a los módulos e interactuar con ellos.	Terminado	40	30	El dispositivo afiliado deberá registrarse en la colección de dispositivos por cliente.	Validar los datos a considerarse en la colección de dispositivos por cliente.	Historia	4	ScrumTeam01	2
7	El sistema permitirá que el cliente desafilie su dispositivo para impedir que acceda a los módulos e interactuar con ellos.	Terminado	40	30	El dispositivo desafiliado deberá ser eliminado lógicamente de la colección de dispositivos por cliente	Verificar que el dispositivo haya sido cambiado de estado a eliminado o cancelado	Historia	4	ScrumTeam01	2

8	El sistema permitirá que el usuario administrador desafilie el dispositivo del cliente para impedir acceder a los módulos y que interactúe con ellos.	Terminado	40	20	El dispositivo desafiliado deberá ser eliminado lógicamente de la colección de dispositivos del cliente	Verificar que el dispositivo haya sido cambiado de estado a eliminado o cancelado	Historia	4	ScrumTeam01	2

Sprint 1	**Desarrollo de microservicio** Activación de Token Digital
Duración	2 semanas
Objetivo	Proveer un mecanismo de Activación de Token Digital
Descripción	El presente microservicio deberá permitir la activación del token Digital como medio de garantizar la seguridad de la información al acceder a los diferentes módulos del sistema de la entidad financiera.

Fuente: Elaboración propia

DOR (Definition of Ready)

Tabla 9

DOR (Definition of Ready)

N°	DESCRIPCIÓN	RESPONSABLE	ESTADO
1	Mapa de datos de las colecciones de MongoDB	Desarrolladores	ok
2	Acceso a las colecciones de MongoDB	Desarrolladores	ok
3	Contar con un entorno de desarrollo común	Desarrolladores	ok
4	Crear el repositorio para la gestión del código fuente	Desarrolladores	ok
5	Haber realizado la ceremonia Sprint Planning	SCRUM TEAM	ok
6	Contar con el Sprint Backlog	SCRUM TEAM	ok
7	Contar con la aprobación del Product Owner (PO) para iniciar el Sprint	PO	ok

Fuente: Elaboración propia

Sprint 1 Backlog (tareas, esfuerzo y horas)

Tabla 10

Sprint 1 Backlog (tareas, esfuerzo y horas)

Nro.	USER STORIES	ESTADO	ESTIMACIÓN						
			PTOS.USER	HORAS	CRITERIO DE ACEPTACIÓN	COMENTARIO	TIPO EPICA	PRIORIDAD (1-5)	RESPONSABLE
1	El sistema debe permitir la búsqueda del código del cliente para la generación del token digital	Terminado	40	20	El sistema deberá encontrar los datos del cliente para la generación del Token Digital.	El código del cliente ¿Cuál es el formato del código del cliente?	Tarea	5	Desa001
2	El sistema debe permitir activar el token digital y enviárselo al cliente	Terminado	20	20	El sistema activará el token digital y enviará el mismo al cliente.	¿Por qué medios será enviado el token? ¿Qué dice el negocio?	Tarea	4	Desa002
3	El sistema debe contar con una lista de solicitudes pendientes de clientes para activar los tokens digitales respectivos	Terminado	20	20	Evidenciar la implementación de una cola con las solicitudes de activación de tokens a nivel nacional.	La cola debería permitir identificar al cliente; por tanto, en la lista solo habrán solicitudes de clientes	Tarea	3	Desa003

						debidamente autenticados.			
4	El sistema validará el token digital y emitirá una respuesta.	Terminado	20	20	La validación del token digital devolverá en todos los casos la autorización o la denegación del Token Digital de darse el caso.	Evidenciar el valor de retorno y cómo lo procesa el app ó la web.	Tarea	4	Desa004

Fuente: Elaboración propia

DOD (Definition of Done)

Tabla 11

DOD (Definition of Done)

N°	DESCRIPCIÓN	Responsable	Estado
1	Entregar Mapa de datos actualizado	Desarrollo	Completado
2	Haber realizado el merge del desarrollo en el repositorio a fin de tener una sola rama	Desarrollo	Completado
3	Pruebas unitarias	Desarrollo	Completado
4	Pruebas de integración	Desarrollo	Completado
5	Calidad de Software	QA	Completado
6	Conformidad del Product Owner ratificando la funcionalidad e incremento del software	PO	Completado
7	Puesta en producción	Desarrollo/QA	Completado

Fuente: Elaboración propia

Tabla 12

Sprint 2

Sprint 2	**Desarrollo de microservicio DEVICE ENROLLMENT**
Duración	2 semanas
Objetivo	Proveer de un mecanismo para la afiliación y desafiliación de los dispositivos del cliente.
Descripción	El presente microservicio deberá permitir la afiliación y desafiliación de los dispositivos del cliente. Dispositivos con los que podrá acceder a todos los módulos de la entidad financiera.

Fuente: Elaboración propia

DOR (Definition of Ready)

N°	DESCRIPCIÓN	RESPONSABLE	ESTADO
1	Mapa de datos de las colecciones de MongoDB	Desarrolladores	Completado
2	Acceso a las colecciones de MongoDB	Desarrolladores	Completado
3	Contar con un entorno de desarrollo común	Desarrolladores	Completado
4	Crear el repositorio para la gestión del código fuente	Desarrolladores	Completado
5	Haber realizado la ceremonia Sprint Planning	SCRUM TEAM	Completado
6	Contar con el Sprint Backlog	SCRUM TEAM	Completado
7	Contar con la aprobación del Product Owner (PO) para iniciar el Sprint	PO	

Tabla 13

Sprint2 Backlog (tareas, esfuerzo y horas)

Nro	USER STORIES	ESTADO	ESTIMACIÓN						
			PTOS.USER	HORAS	CRITERIO DE ACEPTACIÓN	COMENTARIO	TIPO EPICA	PRIORIDAD	RESPONSABLE
1	El sistema permitirá que el cliente afilie su dispositivo para poder acceder a los módulos e interactuar con ellos.	En progreso	40	30	El dispositivo afiliado debe figurar en la colección de "dispositivos" con los datos mínimos permitidos.	Revisar si los datos mínimos requeridos fueron actualizados en la colección de dispositivos.	Tarea	5	Desa001
2	El sistema permitirá que el cliente desvincule su dispositivo para impedir que acceda a los módulos e interactuar con ellos.	En progreso	40	30	El dispositivo a ser desafiliado en la colección de "dispositivos" con los datos mínimos permitidos será eliminado de la colección mediante una eliminación lógica.	Revisar si con los datos mínimos permitidos es posible la eliminación de los datos del dispositivo del cliente.	Tarea	5	Desa002
3	El sistema permitirá que el usuario administrador desafilie el dispositivo del cliente para impedir acceder a los módulos y que interactúe con ellos.	En progreso	40	20	El administrador tendrá la posibilidad de desafiliar el dispositivo del cliente.	Revisar la lista de dispositivos afiliados del cliente. Es decir, si el dispositivo eliminado ya	Tarea	5	Desa003

						no aparece en la lista de dispositivos afiliados.			

Fuente: Elaboración propia

DOD (Definition of Done)

N°	DESCRIPCIÓN	Responsable	Estado
1	Entregar Mapa de datos actualizado	Desarrollo	ok
2	Haber realizado el merge del desarrollo en el repositorio a fin de tener una sola rama	Desarrollo	ok
3	Pruebas unitarias	Desarrollo	ok
4	Pruebas de integración	Desarrollo	ok
5	Calidad de Software	QA	ok
6	Conformidad del Product Owner ratificando la funcionalidad e incremento del software	PO	ok
7	Puesta en producción	Desarrollo/QA	ok

4.2. Presentación de Desarrollo

En la presente investigación, se utilizó la tecnología de java, herramientas y utilidades para agilizar el proceso de desarrollo, como patrones de diseño, principios de solid, programación reactiva, spring boot, lombok, spring data, Junit, mockito, sonarqube, etc.

Figura 16

Herramientas para desarrollo del microservicio

Fuente: Elaboración Propia

4.2.1. Módulo de Capas de Arquitectura

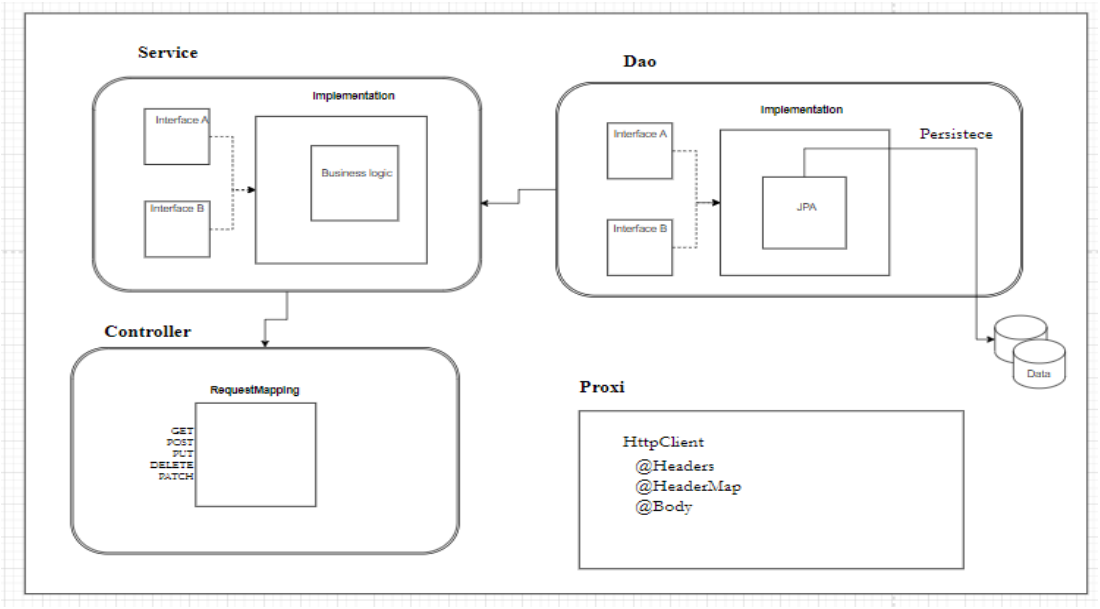
En esta etapa, se desarrolla los microservicios, y se configura los archivos que serán utilizados por aks y jenkins. Estos servicios ofrecerán el API que se consultará desde el aplicativo Front-End, y para realizar las pruebas directas utilizaremos postman.

4.2.2. Arquitectura tecnológica de Solución

A continuación, se muestra el esquema de la arquitectura de solución, que permitiría contextualizar las diferentes funcionalidades a través de capas.

Figura 17

Arquitectura tecnológica de Solución



Fuente: Elaboración propia

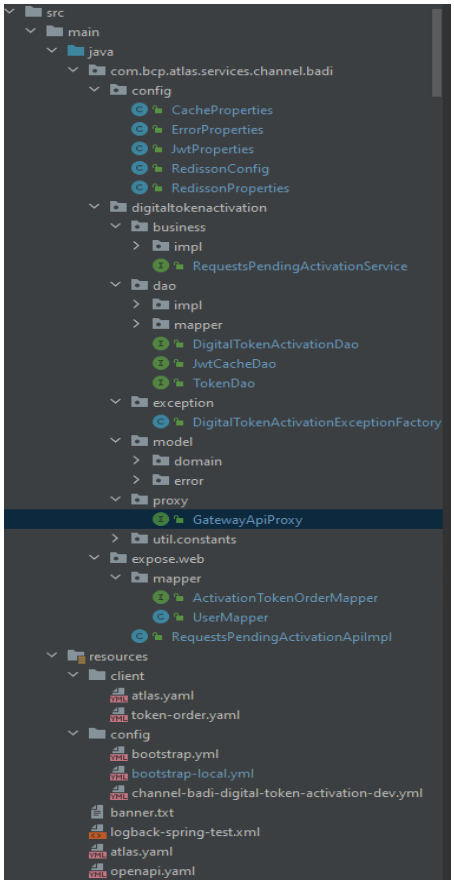
- **Capa de presentación (Controller):** Dado que debe permitir alternar entre varios niveles de visualización, debe ser lo más ligero posible.
- **Capa de Negocios o Servicios:** En pocas palabras, las transacciones en el sistema ocurren en esta capa. Esta capa debe ser completamente reutilizable y no debe depender de la capa de presentación de ninguna manera.
- **Interface DAO:** La lógica empresarial está ausente de esta capa. simboliza las conexiones entre componentes (independientemente de cualquier tecnología de acceso a datos).

  El objetivo principal de la DAO es desacoplar la tecnología de acceso a datos de la capa de lógica empresarial, lo que facilita el cambio entre diferentes tipos de bases de datos sin tener que volver a escribir la aplicación desde cero.

• **Bases de datos**: Es el repositorio donde se almacena volúmenes masivos de datos, y cada colección almacena documentos en forma de JSON**.**

Figura 18

Estructura del microservicio de Activación de Token Digital



Fuente: Elaboración propia

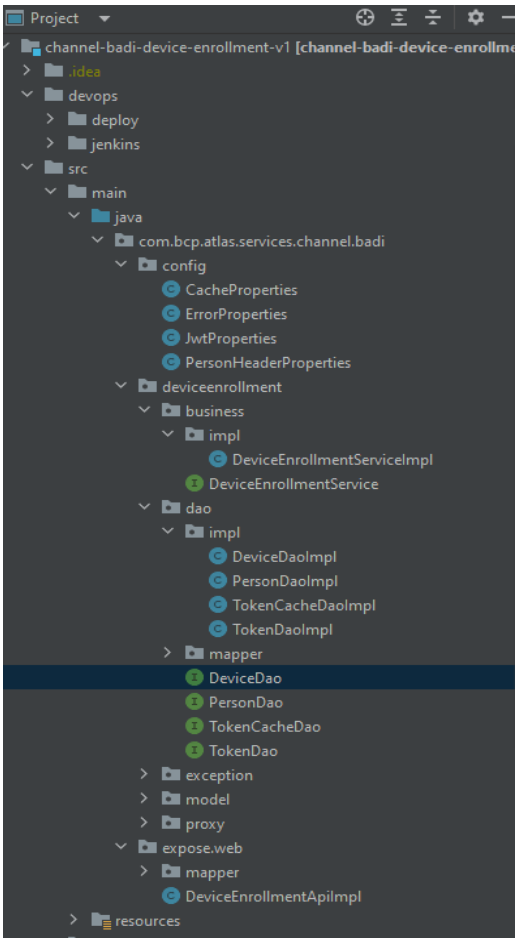
A continuación, se proporcionará una breve explicación de las funciones de los archivos, organizados en carpetas:

- **Config**: en esta carpeta se colocan todas las configuraciones del api, Redis, Kafka y Propiedades de Errores.
- **Business**: Es una división entre la capa de acceso de datos y la propia base de datos, funciona como intermediario para que esta manera la lógica de negocio esté arraigada a la capa DAO.
- **Dao**: espacialmente para el acceso o persistencia de datos, y aquí se realiza la implementación de otras Apis en caso de que sea necesario.

- **Excepción**: Aquí se definen las excepciones necesarias para cubrir todas las capas.
- **Model**: Aquí se mapea todos los modelos necesarios para cada endpoints.
- **Proxy**: sirve para armar las estructuras de otras Apis que se necesita para la construcción.
- **Util.constants**: aquí se coloca todas las variables constantes de manera pública.

Figura 19

Estructura del microservicio de desafiliación de dispositivo



Fuente: Elaboración propia

Aquí podemos apreciar que la estructura es muy similar a la Activación de Token Digital Figura 18. Tener bien definido el estilo y capas de la arquitectura nos facilita mantener en uniformidad la estructuración de cada componente de microservicio.

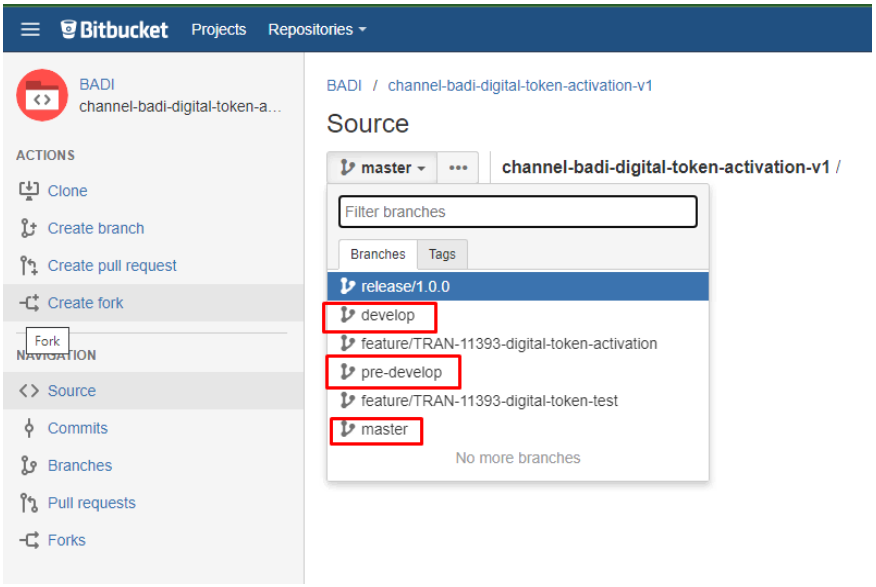
4.2.3. Módulo de Control de Versiones

Para iniciar con el desarrollo de Spring 1 y 2 se crea el repositorio principal donde se almacenará el código fuente (develop), en el repositorio (Bitbucket).

Para mantener las mejoras futuras continuamente integradas, tener un historial, retroceder a una versión anterior y agregar funcionalidad, las tres ramas más importantes que se deben establecer son dominar, desarrollar y predesarrollo.

Figura 20

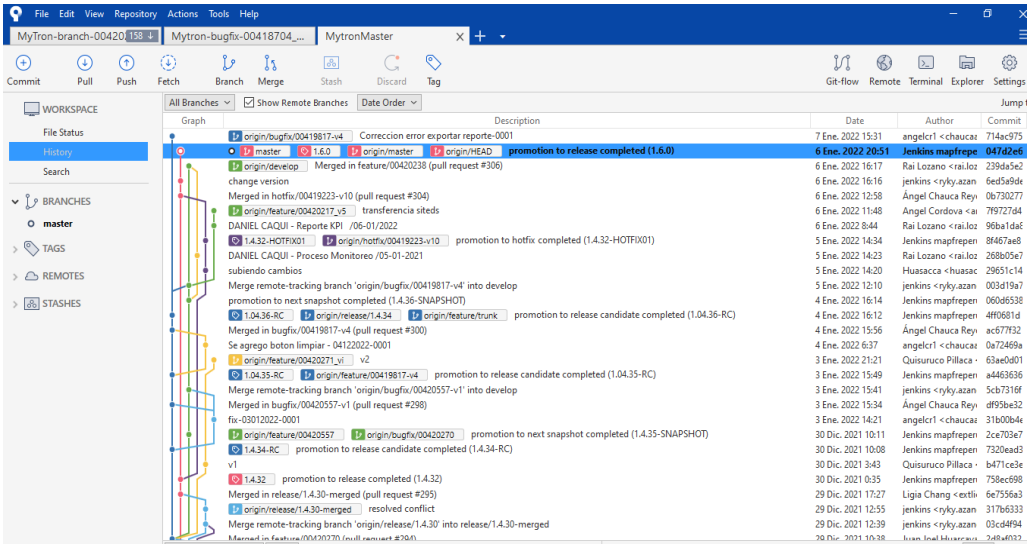
Creación de Ramas en Bitbucket(Git)



Fuente: Elaboración propia

Figura 21

Historial de Construcción en Git



Fuente: Elaboración propia

Bitbucket Cloud: es el editor y repositorio que permite sincronizar Bitbucket con Jira y Trello. Como resultado, es más fácil mantener varias configuraciones y detectar problemas antes de que se vuelvan críticos.

4.2.4. Módulo de Aplicación de Métodos y Patrones

- **Servicios REST**

Su uso permite acceder y/o modificar la información mediante los métodos HTTP, por lo cual se accede a ellos mediante URLs. Y retorna la información en formato JSON.

En la siguiente figura podemos observar el acceso mediante url a la api-bs-token-order, para obtener mediante el method GET la lista de los dispositivos pendientes y con method PATCH cambiar el estado de token digital a activo, se utiliza como el path final: "/requests-pending-activation".

Figura 22

Funcionamiento de servicios rest

Fuente: Elaboración propia

- **Json y modelo de objetos**

  Los archivos JSON son archivos de texto para intercambiar información con los diferentes métodos de http. Y para ello en el desarrollo de las apis se debe mapea cada uno de las entidades que están involucrados a las funcionalidades de Activación de Token digital y Desvinculación de dispositivo. Para representar esta parte se coloca para el primer endpoint que es listar las solicitudes pendientes del dispositivo para activar el token.

Figura 23

Modelo de Objeto para listar solicitudes pendientes de Token Digital

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class DigitalTokenPending {

    private Integer tokenOrderId;
    private Status status;
    private Person person;
    private String registerDate;
    private Device device;

}
```

Fuente: Elaboración propia

Figura 24

Colección en Json - lista solicitudes pendientes de Token Digital

Fuente: Elaboración propia

- **Dependencias**

  POM: Este es la unidad fundamental para trabajar con Maven, con estructura XML, contiene todas las dependencias del proyecto y configuraciones detalladas del mismo.

Figura 25

Archivo de Dependencias POM

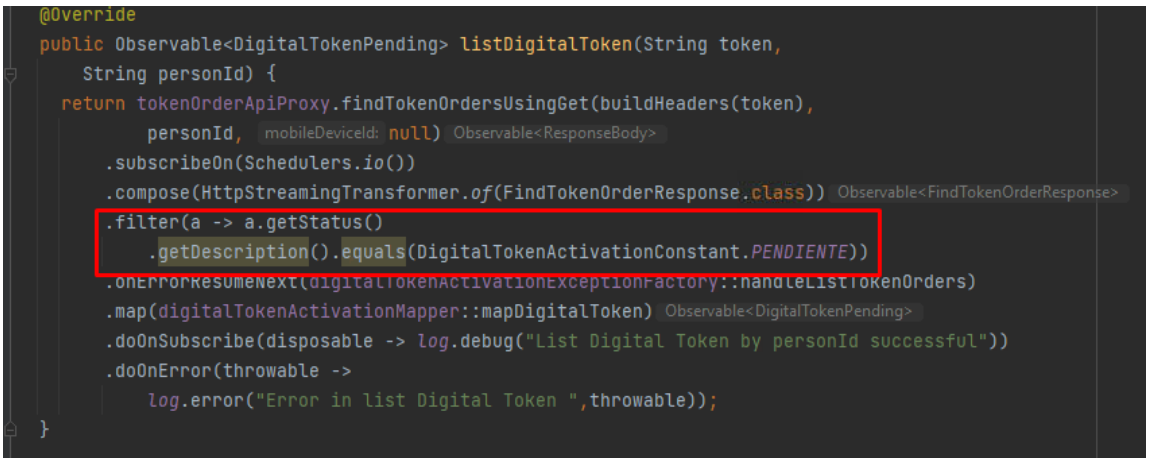
Fuente: Elaboración propia

- **Programación Reactiva**

  En esta modulo veremos como la programación reactiva nos ayuda a filtrar, transformar y resumir los flujos de datos asíncronos para listar solicitudes pendientes de token digital. La simplicidad de cambiar hilos entre los eslabones solamente se puede lograr con programación reactiva.

Figura 26

Programación reactiva para realizar el filtro de solicitudes pendientes



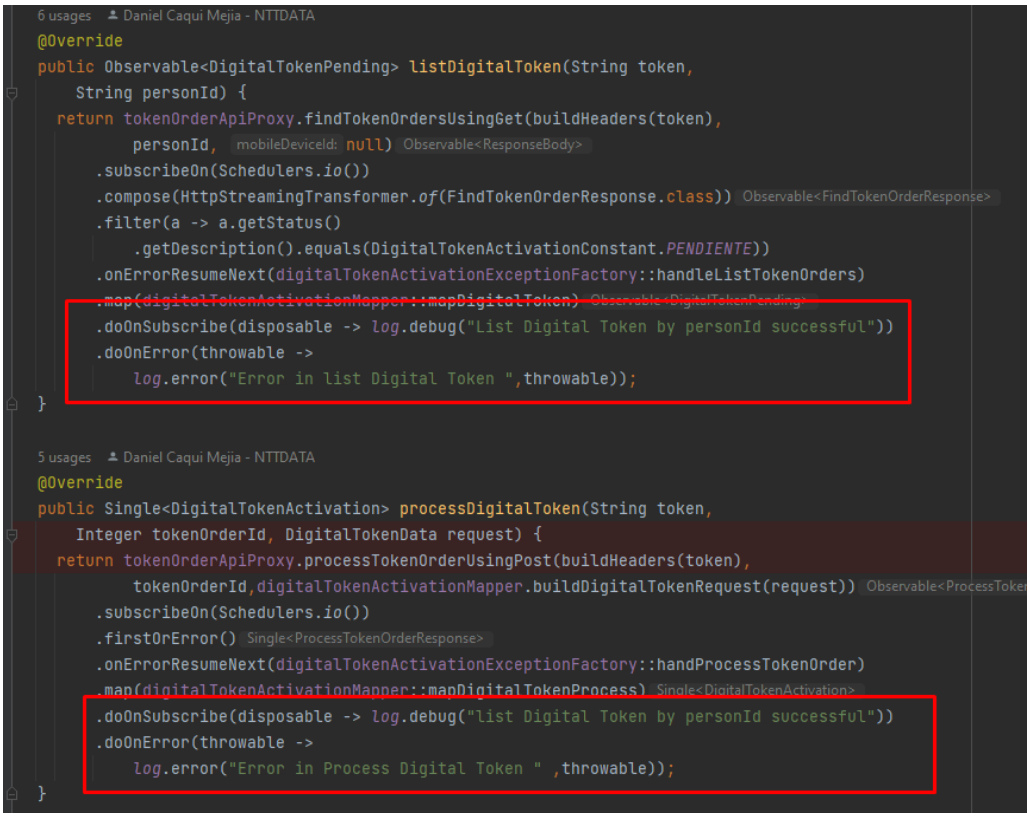
Fuente: Elaboración propia

- **Manejo de trazas con logback**

  Es una biblioteca de código abierto de Java que nos permite personalizar la salida y la granularidad de los mensajes y registros durante la ejecución.

Figura 27

Trazas de logs para cada endpoint



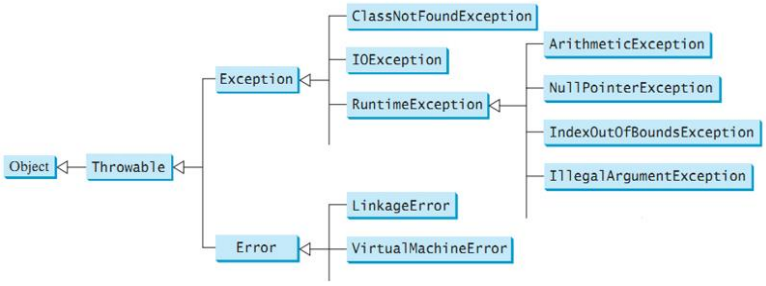
Fuente: Elaboración propia

- **Excepciones**

  El lenguaje de programación Java proporciona un mecanismo llamado excepciones para manejar eventos inesperados que pueden ocurrir durante la ejecución del programa.

Figura 28

Documentación del Código



Nota: Elaboración propia

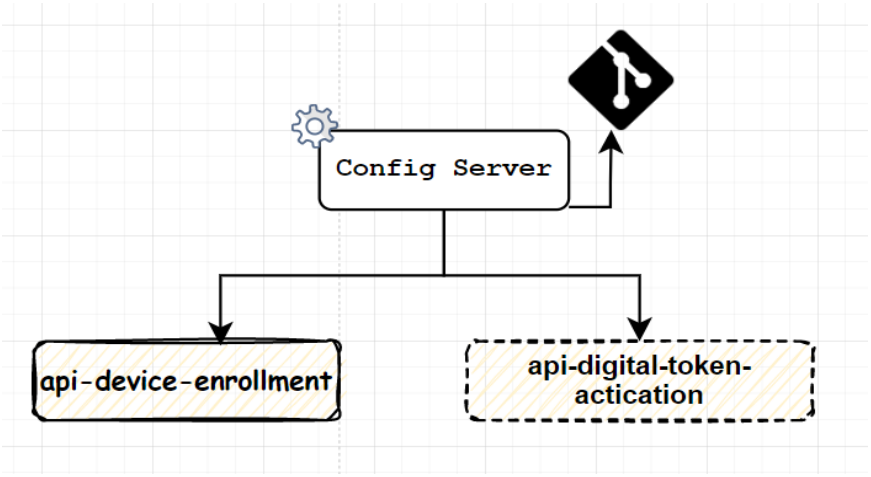
4.2.5. Módulo de configuración

Configuración externalizada con un config server:

Los archivos de configuración están disponibles a través de una interfaz HTTP desde el servidor de configuración. La oferta principal de este proyecto, sirve como depósito de configuración centralizado de nuestra arquitectura.

Figura 29

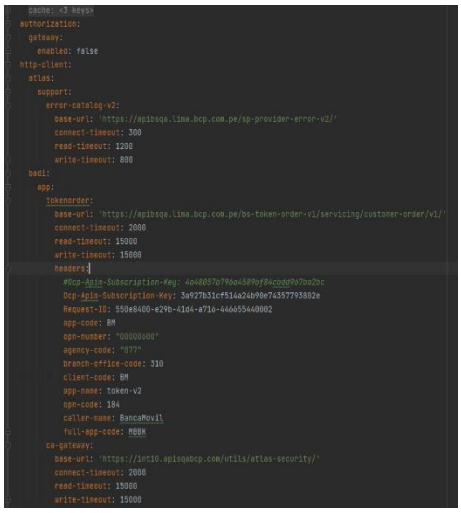
Servidor de Configuración para las API´s



Fuente: Elaboración propia

Figura 30

Archivo de configuración Yaml



Fuente: Elaboración propia

Se incluye todas las configuraciones de auditoría, accesos, nombre, puerto, host, headers, trazas de errores, redis, etc.

4.2.6. Módulo Herramientas

Aquí veremos los estilos de programación que se han elegido para desarrollar esta arquitectura de microservicio.

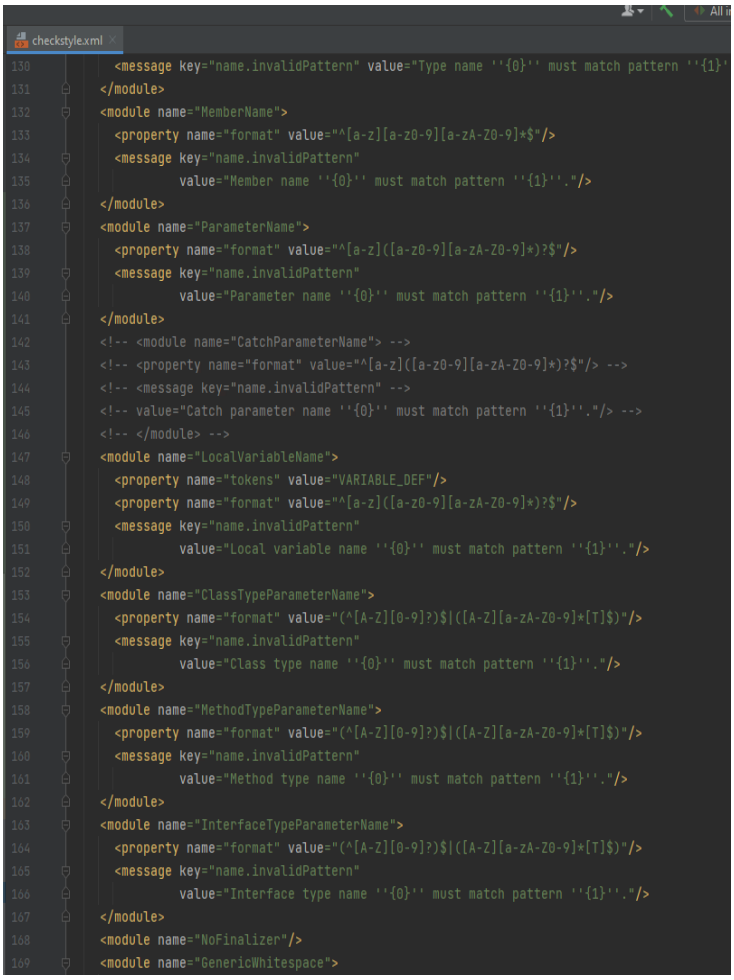
- **Checkstyle**

  Como herramienta de calidad del código, verifica los estándares de construcción del desarrollo de cada API examinando su código fuente en busca de errores de sintaxis y buscando instancias de problemas preconfigurados. Diseño usando Intellij Idea.

  Para ello se asigna un archivo XML de configuración con los formatos estandarizados para todos los proyectos:

Figura 31

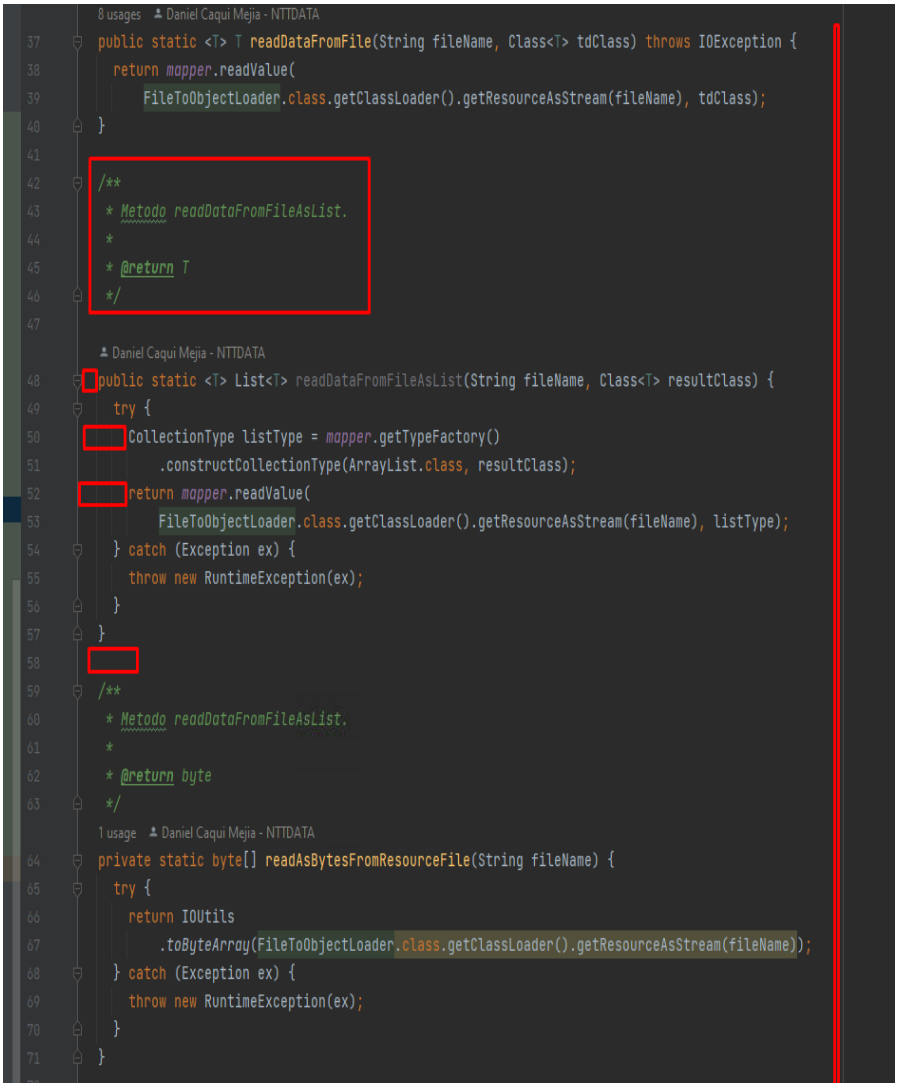
Configuración de Checkstyle



Fuente: Elaboración propia

Figura 32

Demostración Checkstyle en Editor IntelliJ IDEA



```
8 usages  Daniel Caqui Mejia - NTTDATA
public static <T> T readDataFromFile(String fileName, Class<T> tdClass) throws IOException {
    return mapper.readValue(
        FileToObjectLoader.class.getClassLoader().getResourceAsStream(fileName), tdClass);
}

/**
 * Metodo readDataFromFileAsList.
 *
 * @return T
 */

 Daniel Caqui Mejia - NTTDATA
public static <T> List<T> readDataFromFileAsList(String fileName, Class<T> resultClass) {
    try {
        CollectionType listType = mapper.getTypeFactory()
            .constructCollectionType(ArrayList.class, resultClass);
        return mapper.readValue(
            FileToObjectLoader.class.getClassLoader().getResourceAsStream(fileName), listType);
    } catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}

/**
 * Metodo readDataFromFileAsList.
 *
 * @return byte
 */
1 usage  Daniel Caqui Mejia - NTTDATA
private static byte[] readAsBytesFromResourceFile(String fileName) {
    try {
        return IOUtils
            .toByteArray(FileToObjectLoader.class.getClassLoader().getResourceAsStream(fileName));
    } catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}
```

Fuente: Elaboración propia

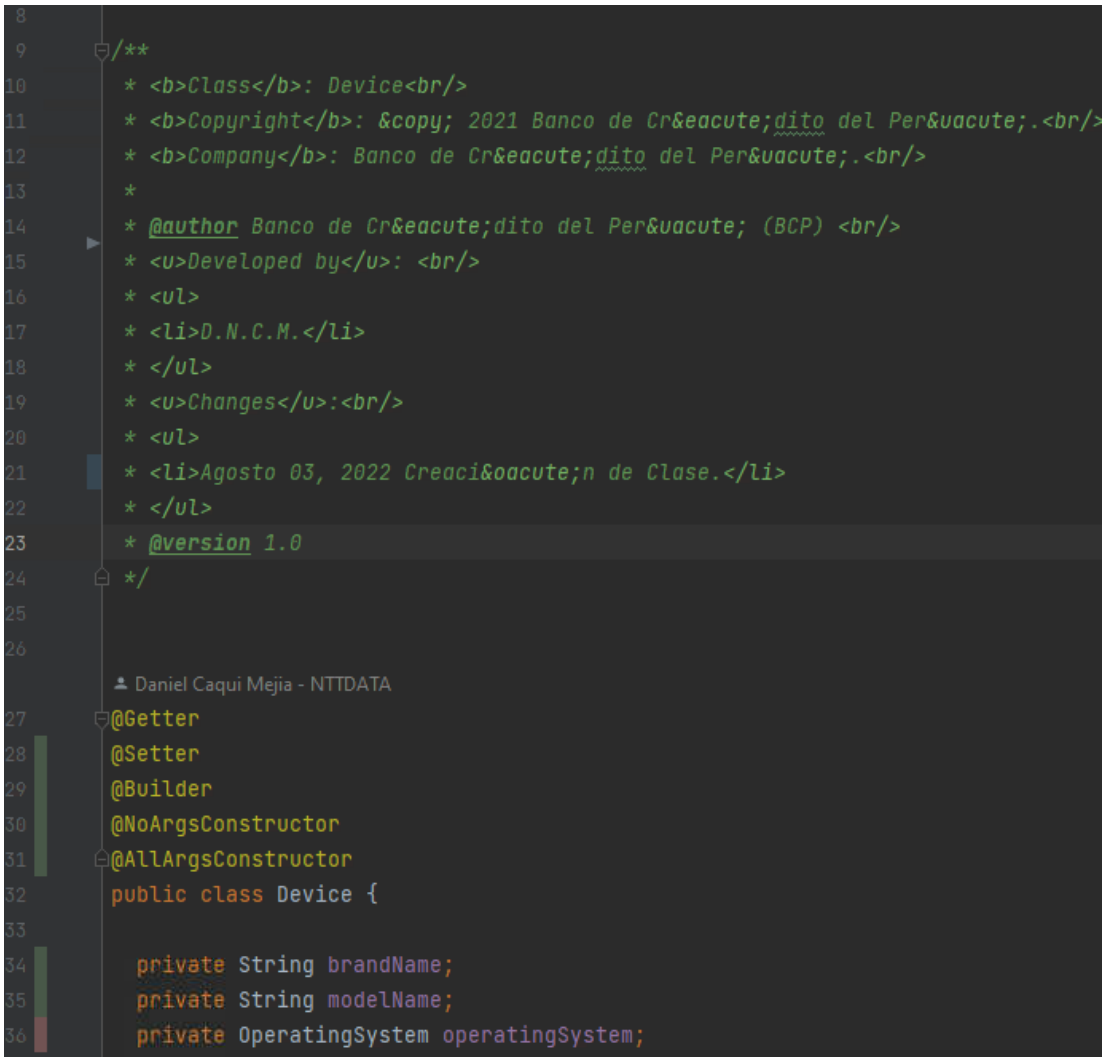
Aquí se puede apreciar, como queda finalmente al pulsar ctrl+shift+f en Intellij Idea, se va formando la sangría de acuerdo al formato establecido, utilizar esta herramienta ayuda a los desarrolladores a no tener conflicto en momento de unir las distintas ramas para la integración de API.

- **Documentación de Código**

  Una parte importante de la construcción de una arquitectura de microservicios es documentar el código para que no solo las computadoras sino también las personas puedan comprender lo que hacen.

Figura 33

Documentación del Código



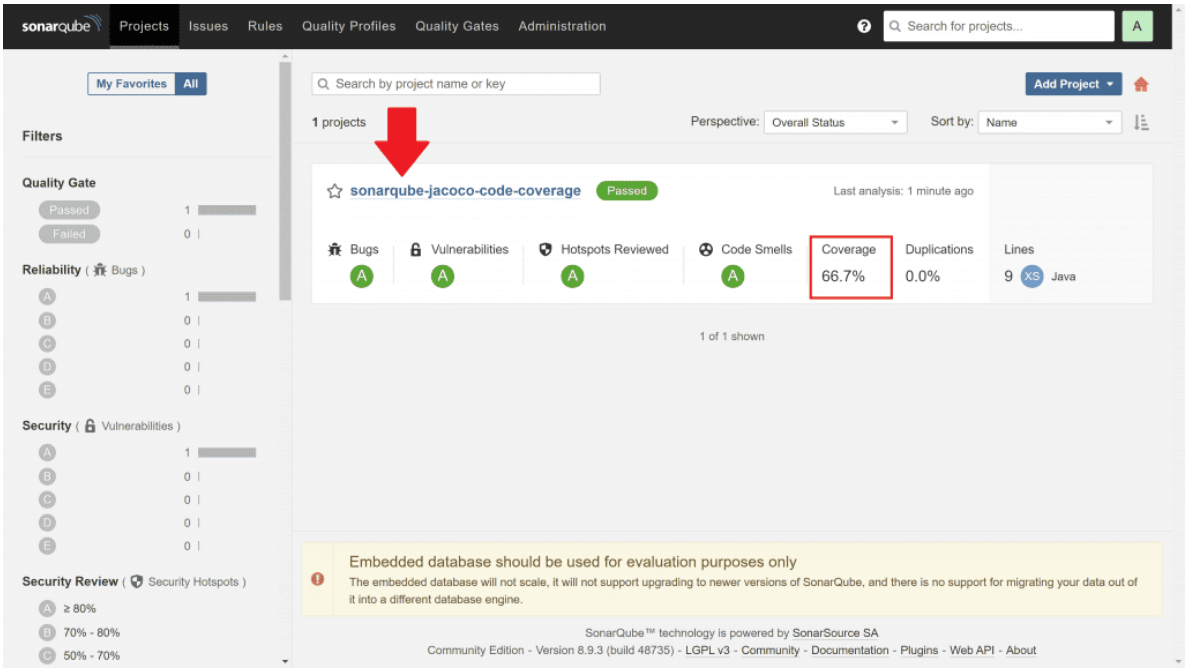
Fuente: Elaboración propia

- **SonarQube:**

  Nos permite realizar inspecciones continuas de la calidad del código en el api de Activación de Token Digital. También brinda análisis que ayudan en el seguimiento y detección de errores y riesgos de seguridad, de tal manera que mantiene limpio el código.

Figura 34

Panel de Sonarqube para visualizar las vulnerabilidades del sistema



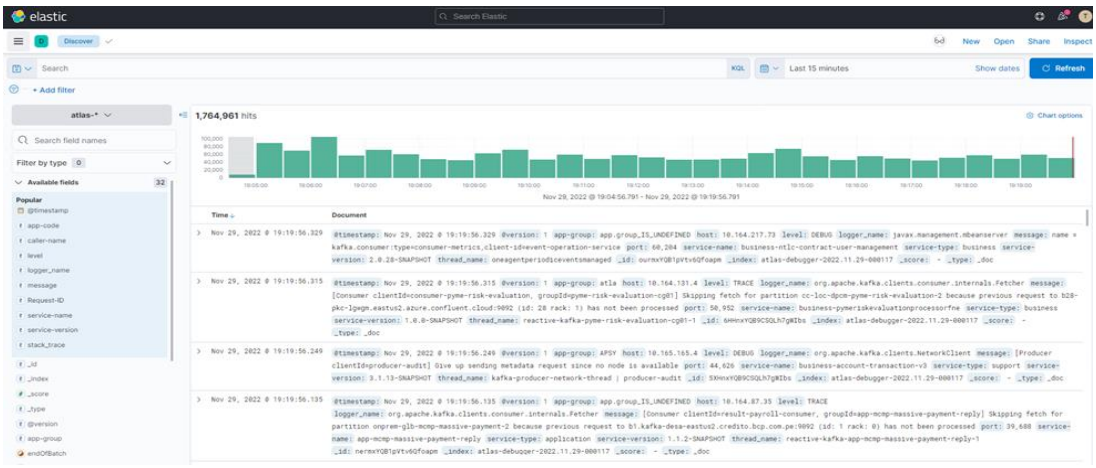
Fuente: Elaboración propia

- **Kibana**

  Es una plataforma de visualización; proporciona información en tiempo real sobre el rendimiento de los microservicios configurados.

Figura 35

Panel de visualización en Kivana



Fuente: Elaboración propia

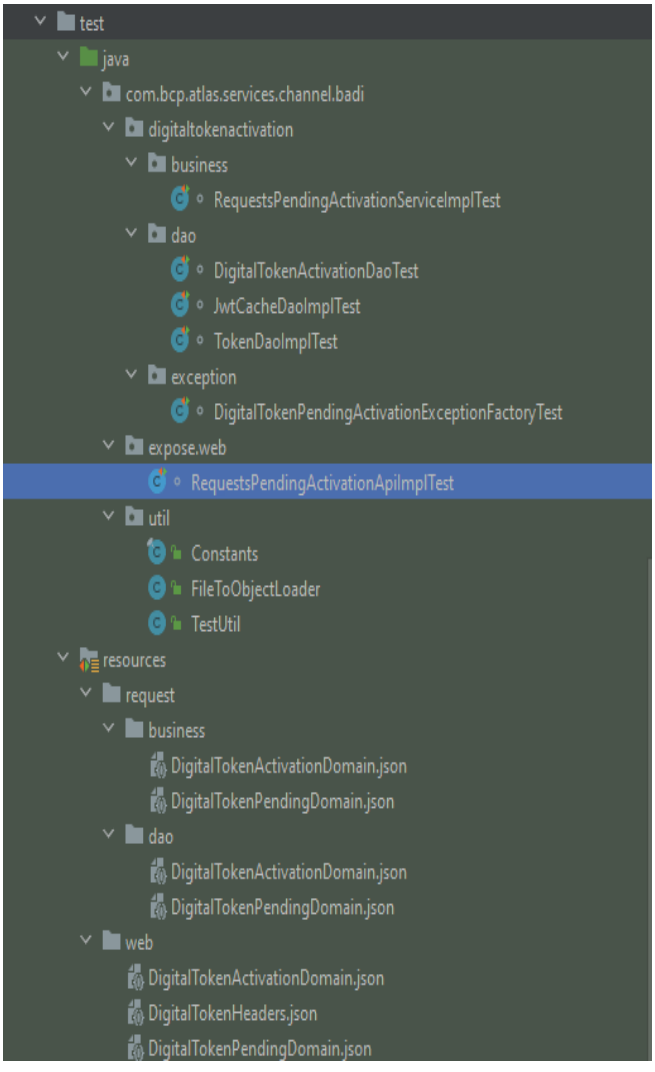
4.2.7. Módulo Pruebas Unitarias

Las pruebas unitarias deben poder ejercitar casi todo su código. La calidad de una prueba unitaria es proporcional a la medida en que prueba el código. El porcentaje del código de una aplicación que se ha probado se conoce como "cobertura de código". Como resultado, una cobertura deficiente indica que una parte significativa de nuestro código no se ha probado a fondo.

En los proyectos de java hay una carpeta que se crea por defecto donde se puede implementar las pruebas unitarias de diferentes capas.

Figura 36

Implementación de Junit Test para Activación de Token Digital



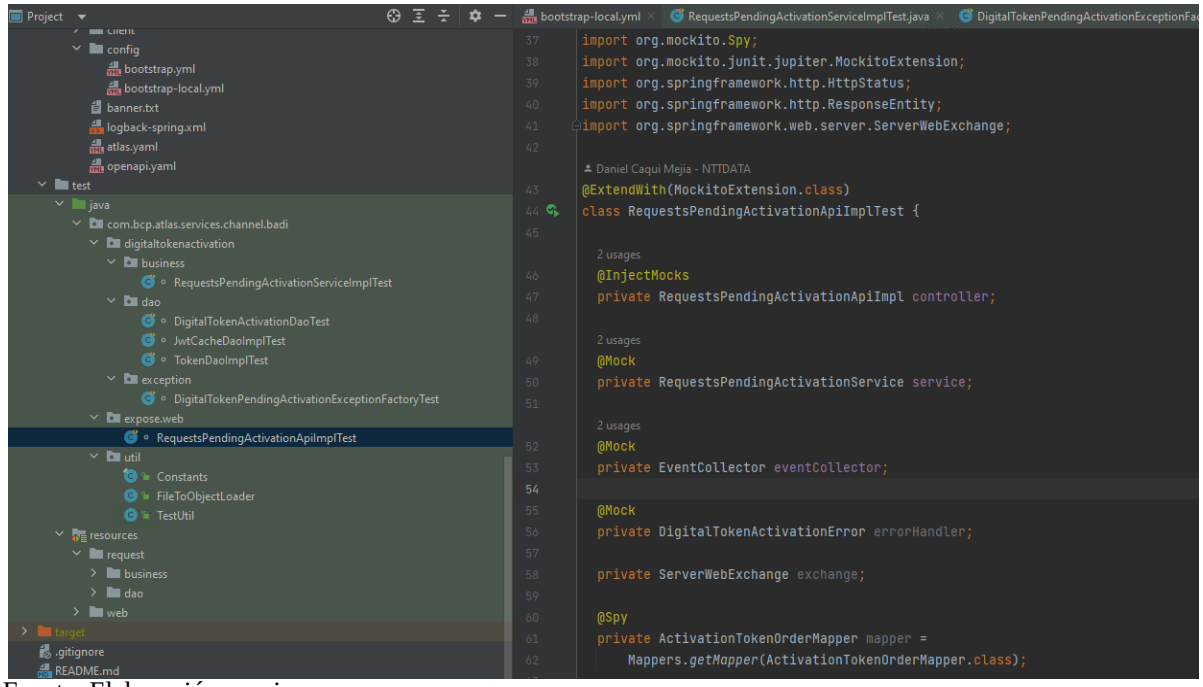
Fuente: Elaboración propia

- **Uso de Junit 5 y Mocks**

  Aplicando estas técnicas de pruebas unitarias para los diferentes módulos y funcionalidades de los microservicios, primero se crea los objetos mock´s con las implementaciones realizadas en cada una de las capas, la cual nos ayuda simular una respuesta y de esta manera se realizar aserciones en todos los escenarios posibles.

Figura 37

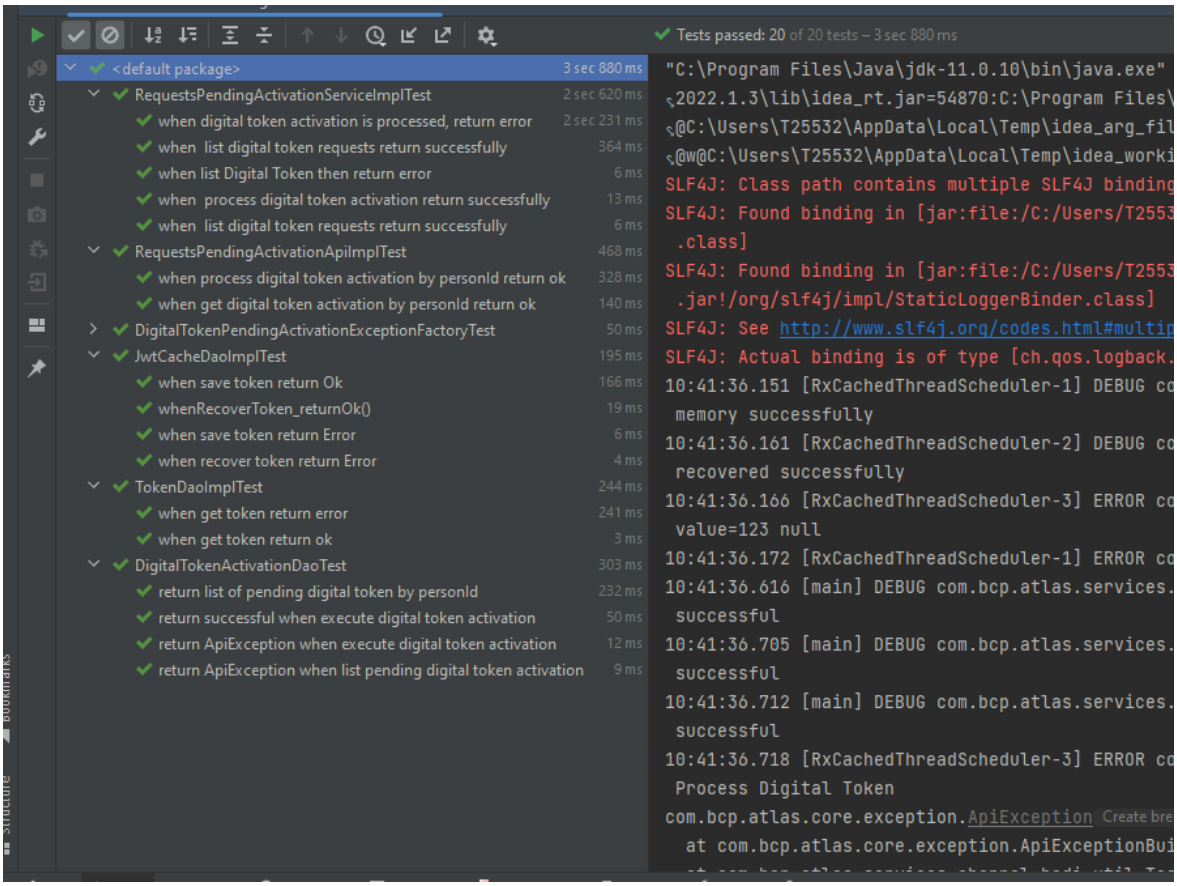
Implementación de Junit Test para Activación de Token Digital



Fuente: Elaboración propia

Figura 38

Cobertura total de test unitarios



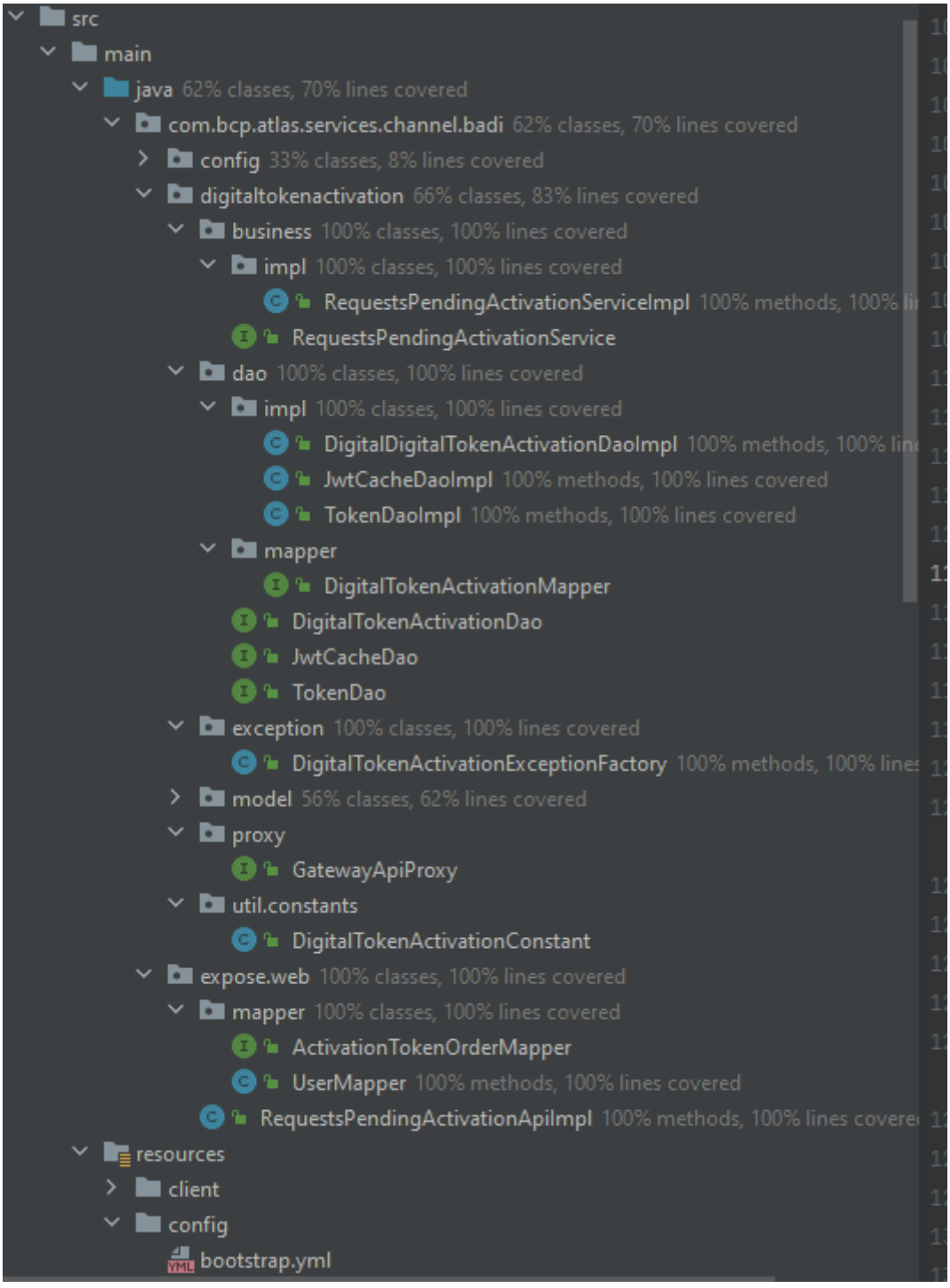
Fuente: Elaboración propia

Aquí se puede apreciar que las pruebas unitarias realizadas en todo el proyecto están marcadas con check verde, significa que todos los test´s realizados se está ejecutando de manera correcta.

En la siguiente figura podemos apreciar con un porcentaje que corresponde a cada una de las capas desarrolladas.

Figura 39

Porcentaje de capas cubiertas

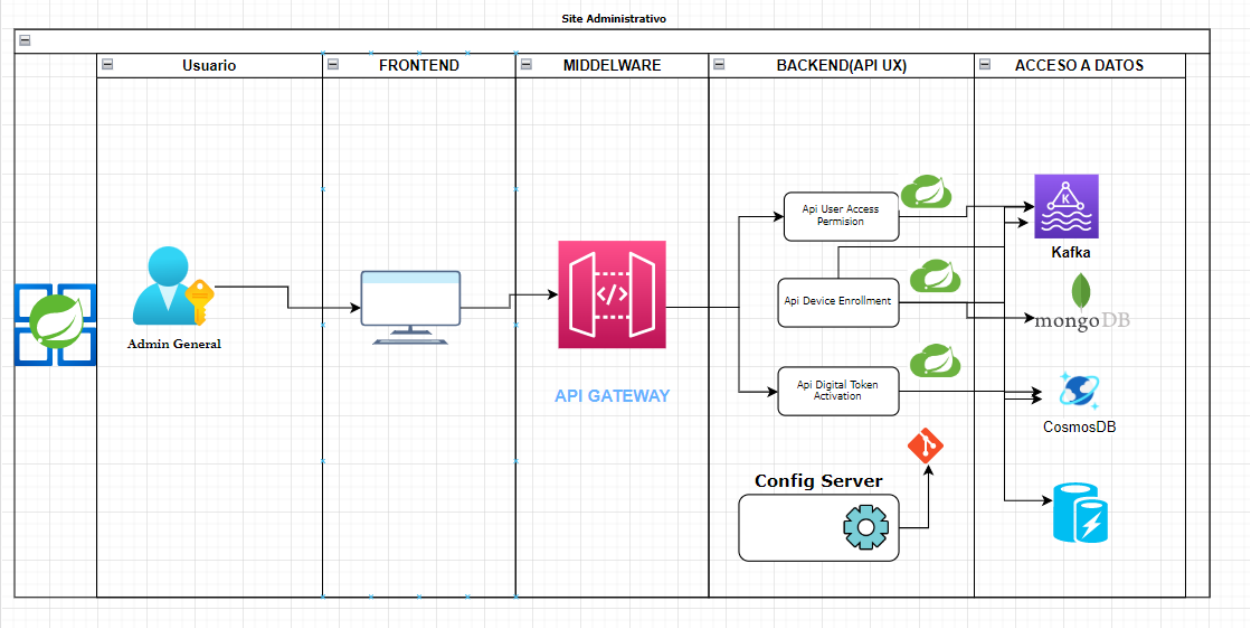


Fuente: Elaboración propia

4.2.8. Diseño Final de Arquitectura de Microservicios

Figura 40

Diseño Tecnológico del prototipo



Fuente: Elaboración propia

4.3. Presentación de resultados y prueba de hipótesis

Se proporcionan resultados tanto descriptivos como inferenciales de la investigación. Para facilitar una comparación entre las pruebas previas y posteriores, los resultados descriptivos se proporcionan en formato tabular y gráfico.

4.3.1. Resultados descriptivos de la variable Arquitectura de microservicios

Tabla 14

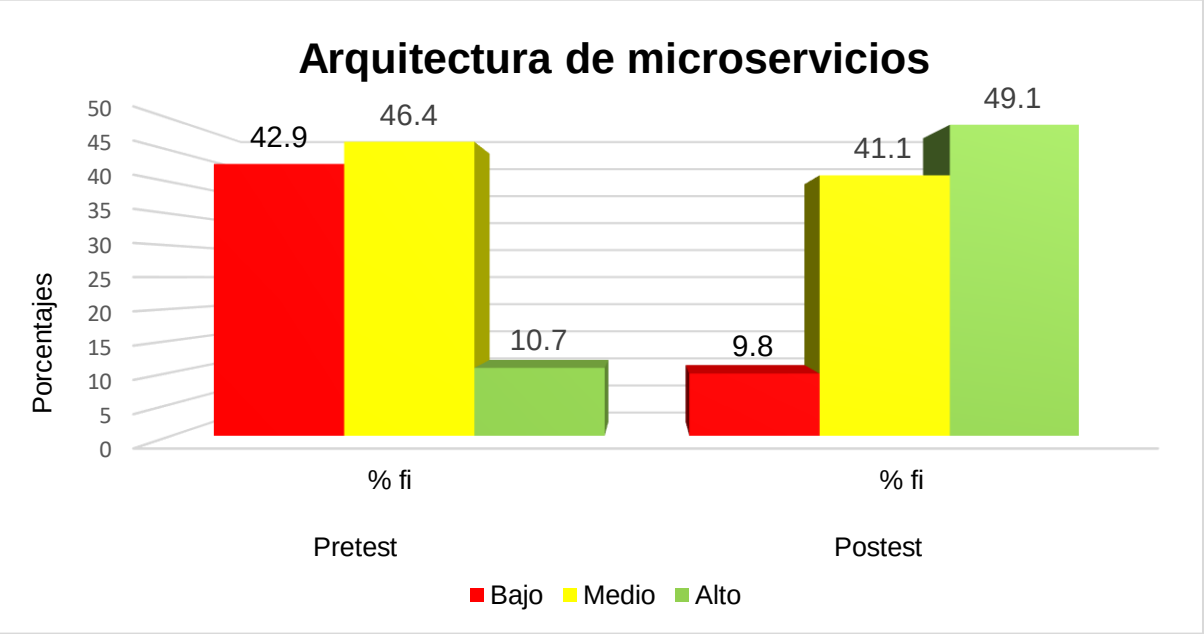
Niveles y frecuencias de la Arquitectura de microservicios

Niveles	Pretest Arquitectura de microservicios		Postest Arquitectura de microservicios	
	f_i	%	f_i	%
Bajo	48	42,9	11	9,8
Medio	52	46,4	46	41,1
Alto	12	10,7	55	49,1
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 401

Nivel de la Arquitectura de microservicios



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 14 y figura 48, se aprecia los resultados de la variable Arquitectura de microservicios. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 48 colaboradores la variable arquitectura de microservicios se encuentra en un nivel bajo en 42,9%, para 52 colaboradores en nivel medio en un 46,4% y para 12 personas en un nivel alto lo cual representa 10,7%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Por otro lado, en el postest se aprecian los siguientes resultados, para 11 colaboradores la variable arquitectura de microservicios se encuentra en un nivel bajo en 9,8%, para 46 colaboradores en nivel medio en un 41,1% y para 55 personas en un nivel alto lo cual representa 49,1%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la arquitectura de microservicios fue la adecuada.

4.3.2. Resultados descriptivos de la dimensión facilidad

Tabla 15

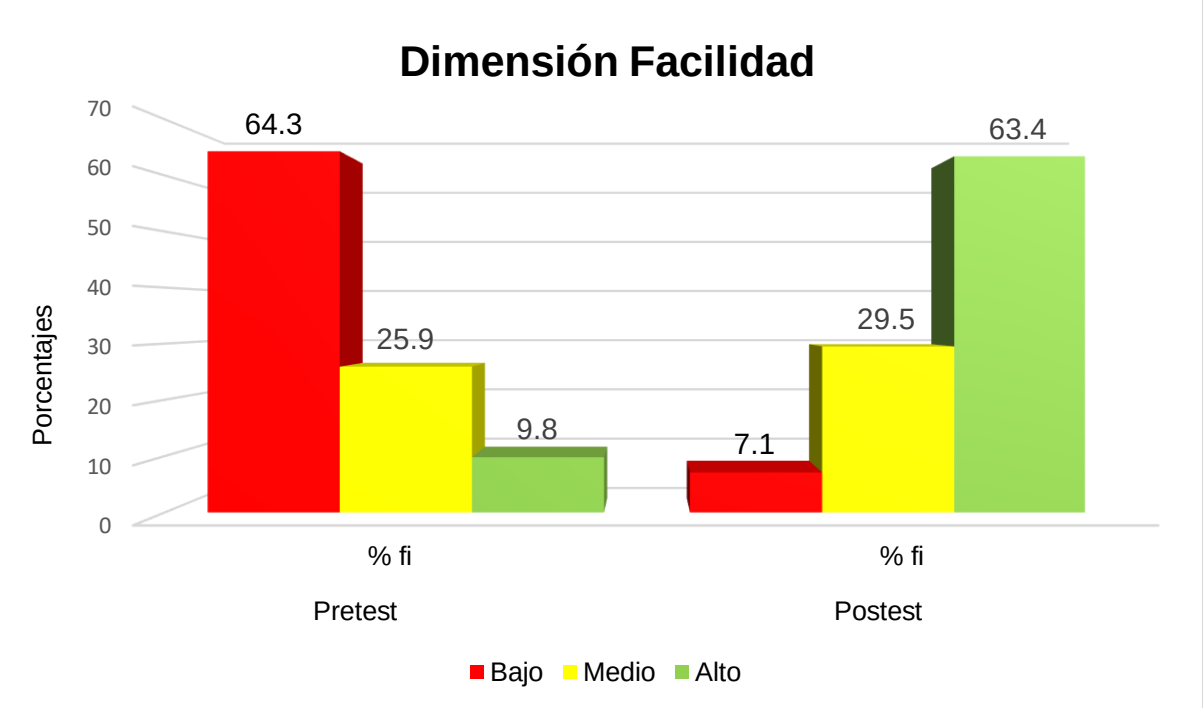
Niveles y frecuencias de la dimensión facilidad

Niveles	Pretest Dimensión Facilidad		Postest Dimensión Facilidad	
	f_i	%	f_i	%
Bajo	72	64,3	8	7,1
Medio	29	25,9	33	29,5
Alto	11	9,8	71	63,4
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 412

Niveles dimensión facilidad



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 15 y figura 49, se aprecia los resultados de la dimensión facilidad de la Arquitectura de microservicios. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los

resultados para el pretest muestran que para 72 la dimensión facilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 64,3%, para 29 colaboradores en nivel medio en un 25,9% y para 11 personas en un nivel alto lo cual representa 9,8%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Por otro lado, en el postest se aprecian los siguientes resultados, para 8 colaboradores la dimensión facilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 7,1%, para 33 colaboradores en nivel medio en un 29,5% y para 71 personas en un nivel alto lo cual representa 63,4%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la dimensión facilidad de la Arquitectura de microservicios fue la adecuada.

4.3.3. Resultados descriptivos de la dimensión disponibilidad

Tabla 16

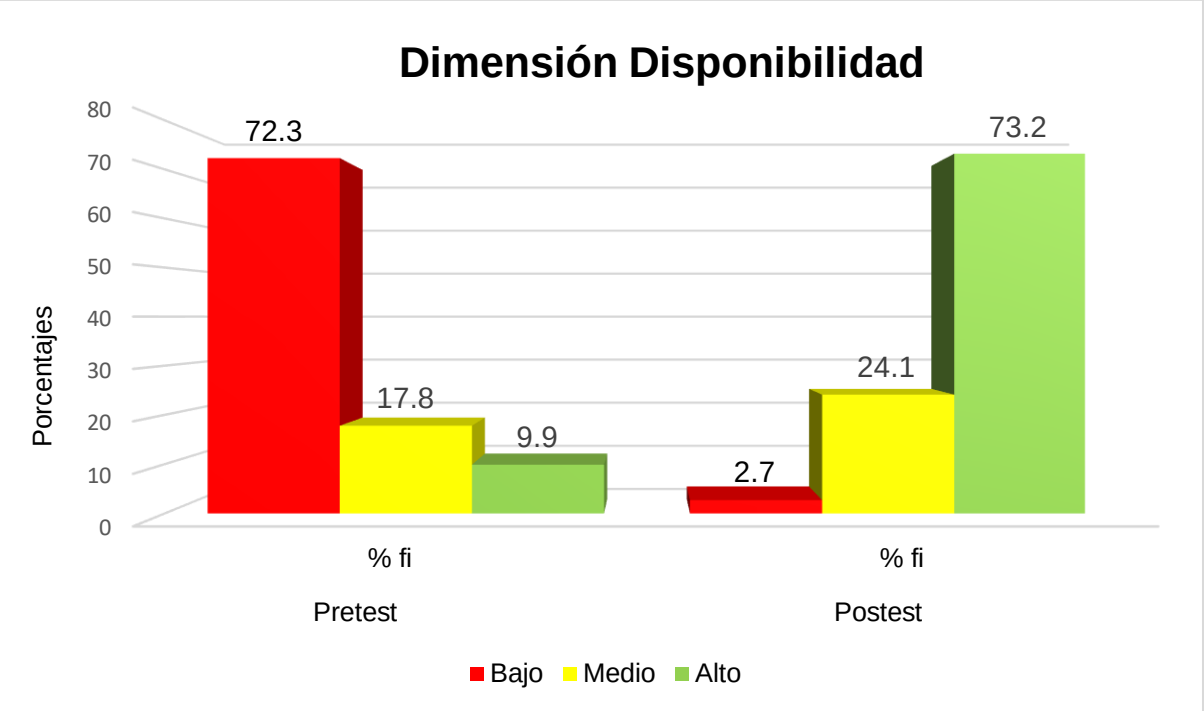
Niveles y frecuencias de la dimensión disponibilidad

Niveles	Pretest Dimensión disponibilidad		Postest Dimensión disponibilidad	
	f_i	%	f_i	%
Bajo	81	72,3	3	2,7
Medio	20	17,8	27	24,1
Alto	11	9,9	82	73,2
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 423

Niveles dimensión disponibilidad



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 16 y figura 50, se aprecia los resultados de la dimensión disponibilidad de la Arquitectura de microservicios. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 81 la dimensión disponibilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 72,3%, para 20 colaboradores en nivel medio en un 17,8% y para 11 personas en un nivel alto lo cual representa 9,9%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Por otro lado, en el postest se aprecian los siguientes resultados, para 3 colaboradores la dimensión disponibilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 2,7%, para 27 colaboradores en nivel medio en un 24,1% y para 82 personas en un nivel alto lo cual representa 73,2%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la

dimensión disponibilidad de la Arquitectura de microservicios fue la adecuada.

4.3.4. Resultados descriptivos de la dimensión flexibilidad

Tabla 17

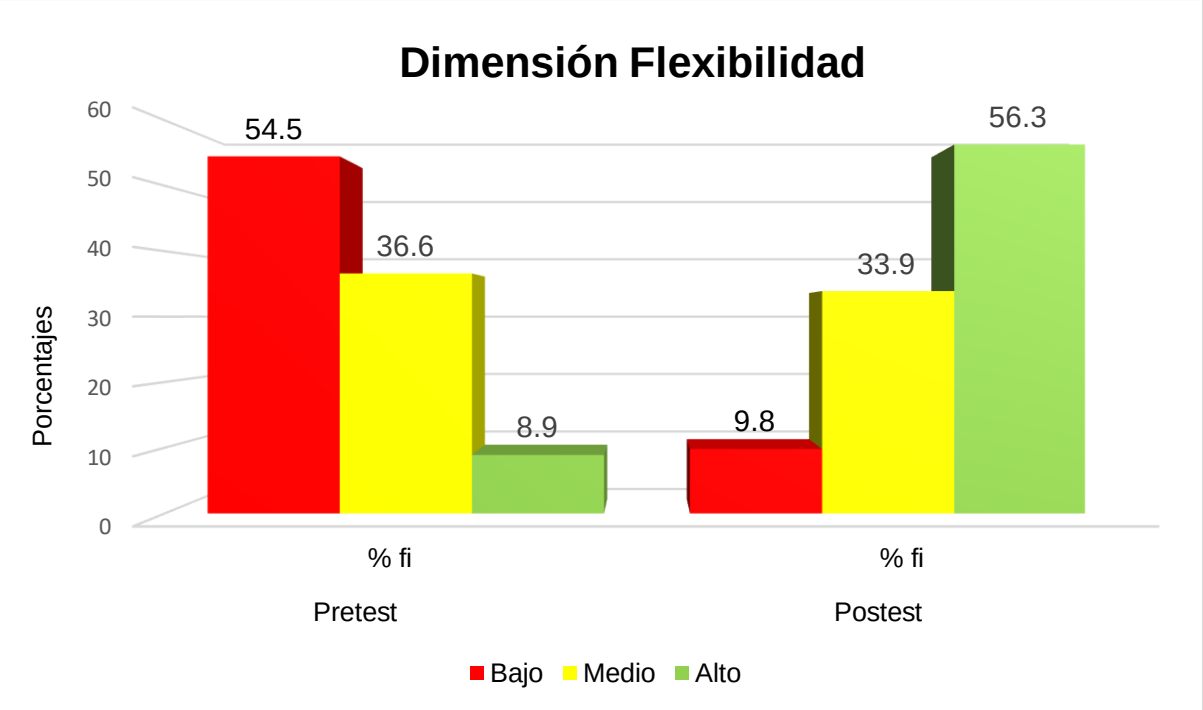
Niveles y frecuencias de la dimensión flexibilidad

Niveles	Pretest Dimensión flexibilidad		Postest Dimensión flexibilidad	
	f_i	%	f_i	%
Bajo	61	54,5	11	9,8
Medio	41	36,6	38	33,9
Alto	10	8,9	63	56,3
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 43

Niveles dimensión flexibilidad



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 17 y figura 51, se aprecia los resultados de la dimensión flexibilidad de la Arquitectura de microservicios. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 61 la dimensión flexibilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 54,5%, para 41 colaboradores en nivel medio en un 36,6% y para 10 personas en un nivel alto lo cual representa 8,9%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Asimismo, en el postest se aprecian los siguientes resultados, para 11 colaboradores la dimensión flexibilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 9,8%, para 38 colaboradores en nivel medio en un 33,9% y para 63 personas en un nivel alto lo cual representa 56,3%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la dimensión flexibilidad de la Arquitectura de microservicios fue la adecuada.

4.3.5. Resultados descriptivos de la dimensión agilidad

Tabla 18

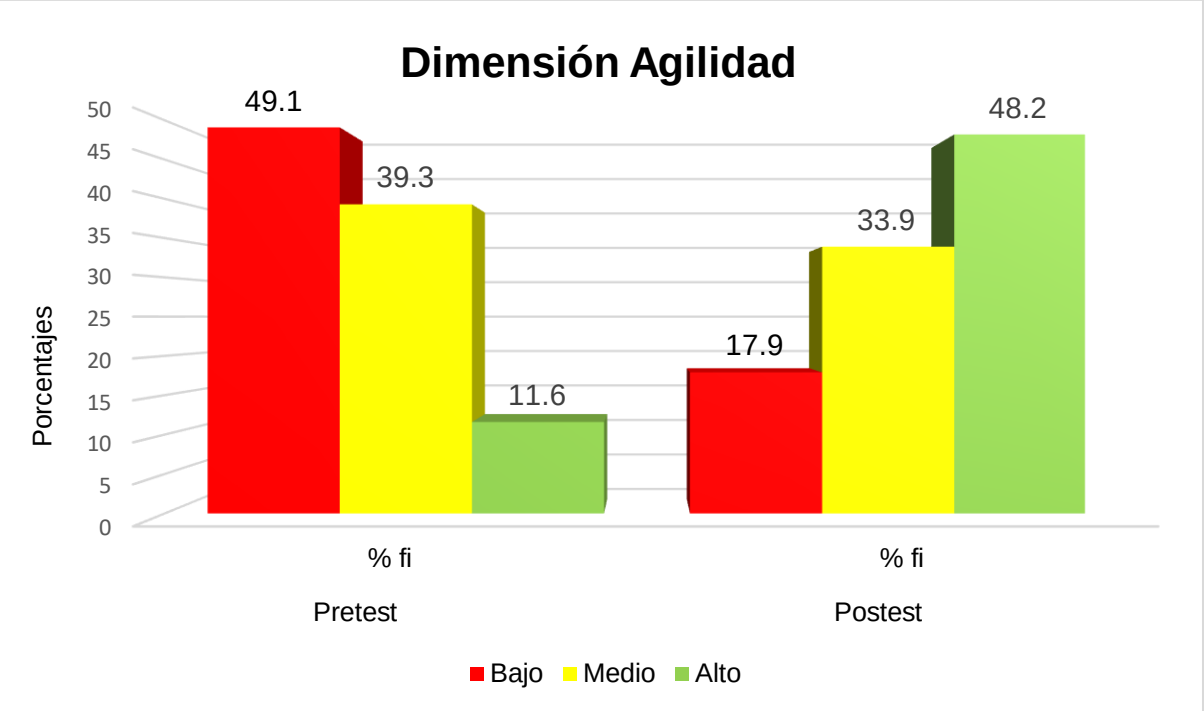
Niveles y frecuencias de la dimensión agilidad

Niveles	Pretest Dimensión agilidad		Postest Dimensión agilidad	
	f_i	%	f_i	%
Bajo	55	49,1	20	17,9
Medio	44	39,3	38	33,9
Alto	13	11,6	54	48,2
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 45

Niveles dimensión agilidad



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 18 y figura 52, se aprecia los resultados de la dimensión agilidad de la Arquitectura de microservicios. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 55 la dimensión agilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 49,1%, para 44 colaboradores en nivel medio en un 39,3% y para 13 personas en un nivel alto lo cual representa 11,6%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Finalmente, en el postest se aprecian los siguientes resultados, para 20 colaboradores la dimensión agilidad de la Arquitectura de microservicios se encuentra en un nivel bajo en 17,9%, para 38 colaboradores en nivel medio en un 33,9% y para 54 personas en un nivel alto lo cual representa 48,2%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la dimensión agilidad de la Arquitectura de microservicios fue la adecuada.

4.3.6. Resultados descriptivos de la variable Mejora de la calidad del software

Tabla 19

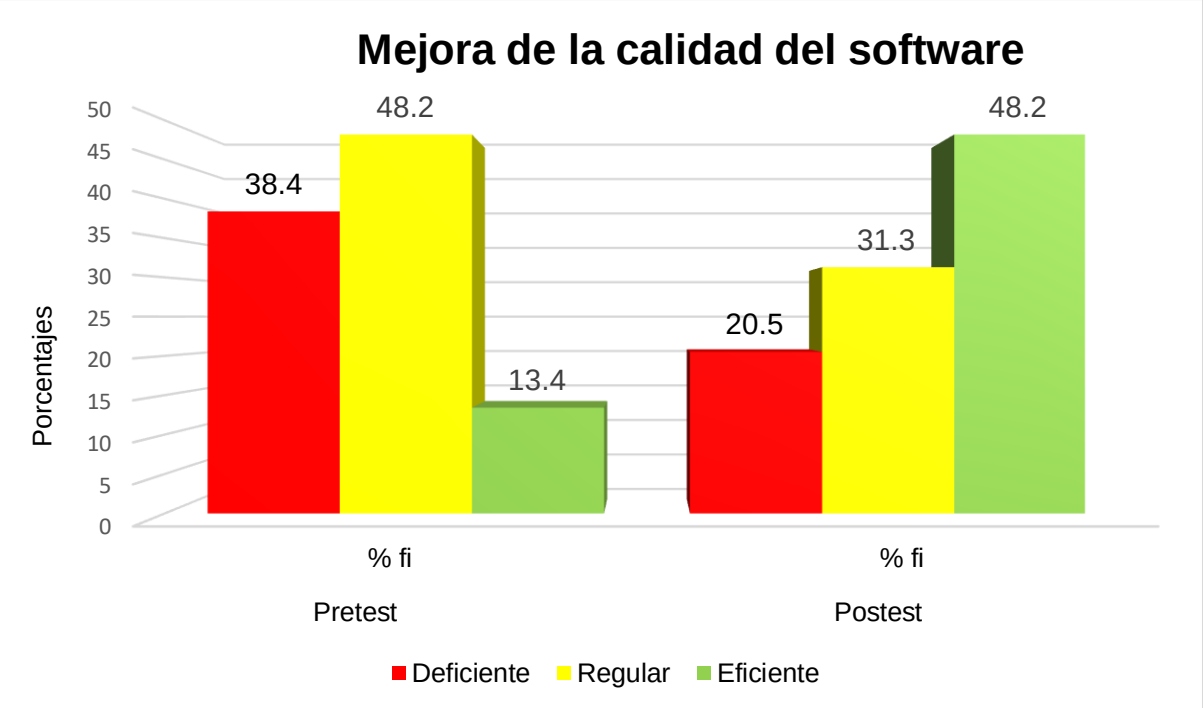
Niveles y frecuencias de la mejora de la calidad del software

Niveles	Pretest Mejora de la calidad del software		Postest Mejora de la calidad del software	
	f_i	%	f_i	%
Deficiente	43	38,4	23	20,5
Regular	54	48,2	35	31,3
Eficiente	15	13,4	54	48,2
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 46

Niveles la variable mejora de la calidad del software



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 19 y figura 53, se aprecia los resultados de la mejora de la calidad del software. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 43 la mejora de la calidad del software se encuentra en un nivel deficiente en 38,%, para 54 colaboradores en nivel regular en un 48,2% y para 15 personas en un nivel eficiente lo cual representa 13,4%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Asimismo, en el postest se aprecian los siguientes resultados, para 23 colaboradores la variable mejora de la calidad del software se encuentra en un nivel deficiente en 20,5%, para 35 colaboradores en nivel regular en un 31,3% y para 54 personas en un nivel eficiente lo cual representa 48,2%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica la variable mejora de la calidad del software fue la adecuada.

4.3.7. Resultados descriptivos de la dimensión eficiencia

Tabla 20

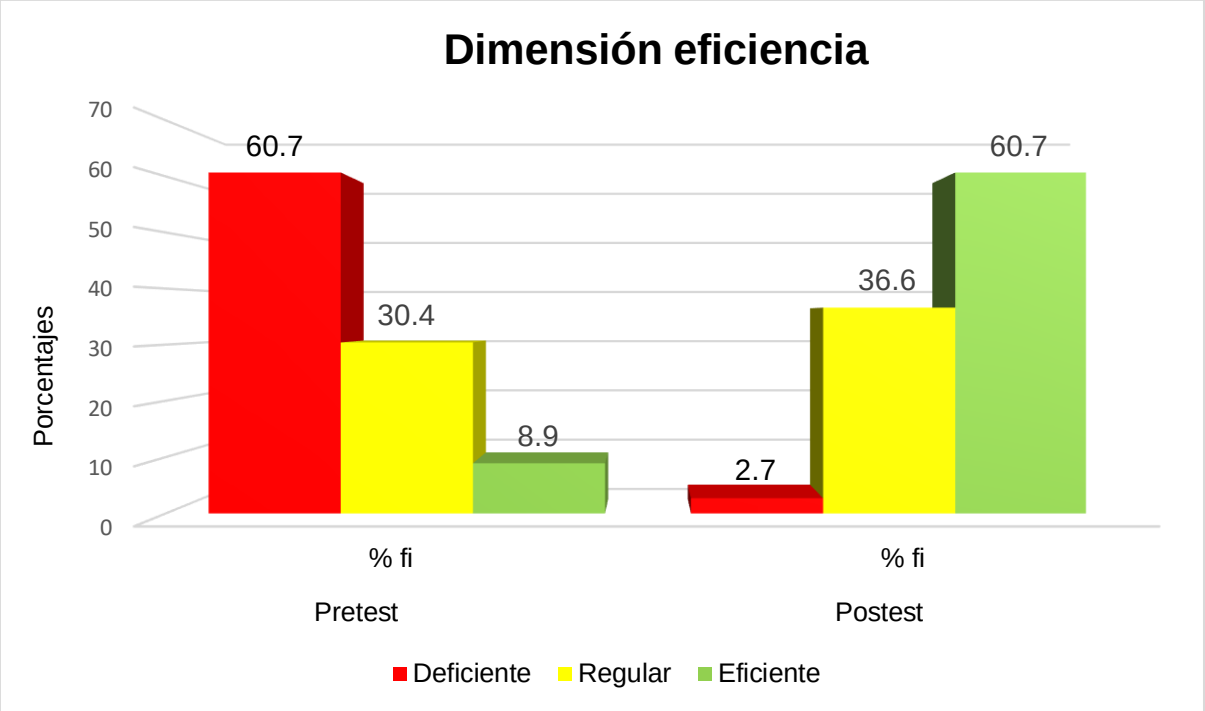
Niveles y frecuencias de la dimensión eficiencia

Niveles	Pretest dimensión eficiencia		Postest dimensión eficiencia	
	f_i	%	f_i	%
Deficiente	68	60,7	3	2,7
Regular	34	30,4	41	36,6
Eficiente	10	8,9	68	60,7
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 47

Niveles de la dimensión eficiencia



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 20 y figura 54, se aprecia los resultados de la dimensión eficiencia de la calidad del software. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 43 la mejora de la dimensión eficiencia calidad del software se encuentra en un nivel deficiente en 38,%, para 54 colaboradores en nivel regular en un 48,2% y para 15 personas en un nivel eficiente lo cual representa 13,4%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Asimismo, en el postest se aprecian los siguientes resultados, para 23 colaboradores la variable mejora de la dimensión eficiencia de la calidad del software se encuentra en un nivel deficiente en 20,5%, para 35 colaboradores en nivel regular en un 31,3% y para 54 personas en un nivel eficiente lo cual representa 48,2%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica de la dimensión eficiencia de la calidad del software fue la adecuada.

4.3.8. Resultados descriptivos de la dimensión confiabilidad

Tabla 21

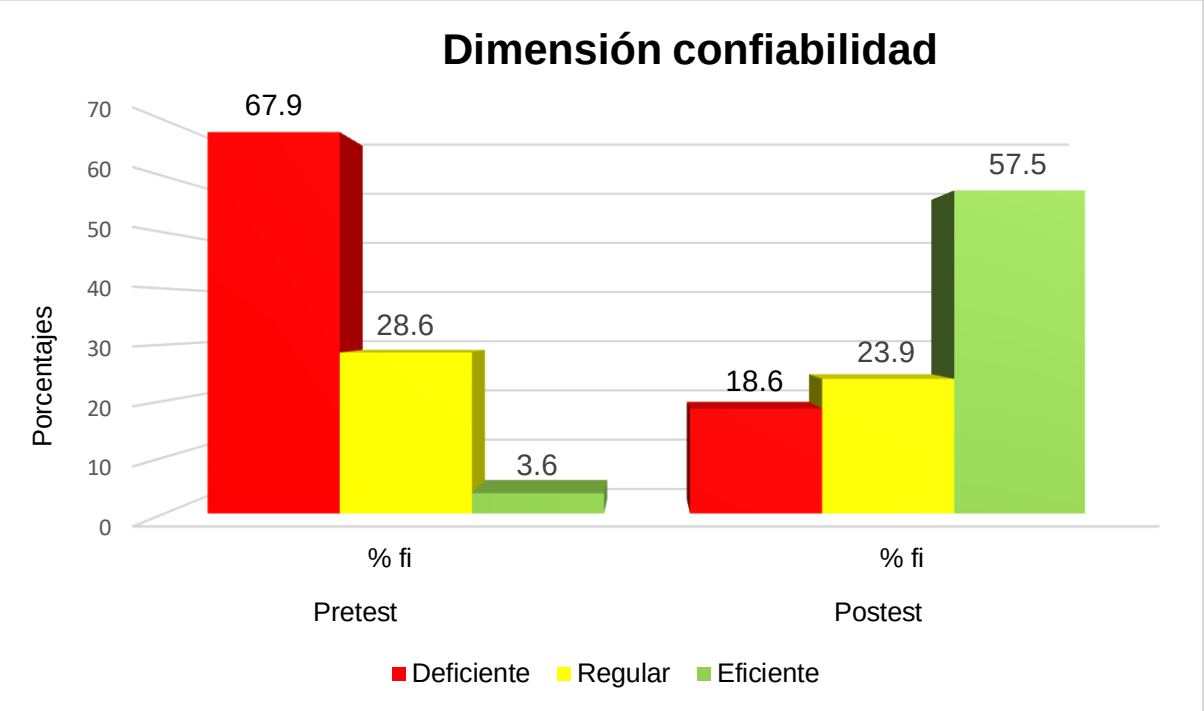
Niveles y frecuencias de la dimensión confiabilidad

Niveles	Pretest dimensión confiabilidad		Postest dimensión confiabilidad	
	f_i	%	f_i	%
Deficiente	76	67,9	20	18,6
Regular	32	28,6	25	23,9
Eficiente	4	3,6	67	57,5
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 48

Niveles de la dimensión eficiencia



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 21 y figura 55, se aprecia los resultados de la dimensión confiabilidad de la calidad del software. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 76 la mejora de la dimensión confiabilidad calidad del software se encuentra en un nivel deficiente en 67,9%, para 32 colaboradores en nivel regular en un 28,6% y para 4 personas en un nivel eficiente lo cual representa 3,6%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Por otro lado, en el postest se aprecian los siguientes resultados, para 20 colaboradores la variable mejora de la dimensión confiabilidad de la calidad del software se encuentra en un nivel deficiente en 18,6%, para 25 colaboradores en nivel regular en un 23,9% y para 67 personas en un nivel alto lo cual representa 57,5%. Se puede apreciar que el nivel preponderante fue el nivel eficiente, lo cual demuestra que para los colaboradores del área

tecnológica de la dimensión confiabilidad de la calidad del software fue la adecuada.

4.3.9. Resultados descriptivos de la dimensión integridad

Tabla 22

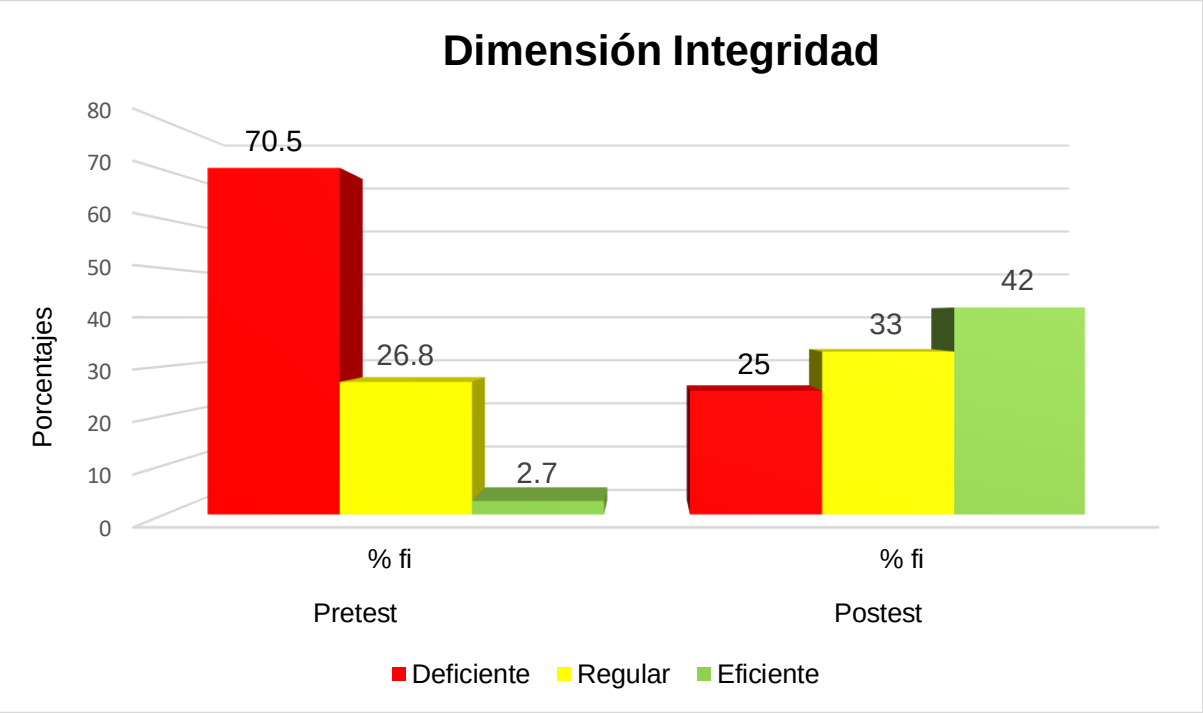
Niveles y frecuencias de la dimensión integridad

Niveles	Pretest dimensión integridad		Postest dimensión integridad	
	f_i	%	f_i	%
Deficiente	79	70,5	28	25,0
Regular	30	26,8	37	33,0
Eficiente	3	2,7	47	42,0
Total	112	100,0	112	100,0

Nota: *fi*=frecuencia absoluta

Figura 49

Niveles de la dimensión integridad



Nota: *fi*=frecuencia absoluta

Se aprecia en la tabla N° 22 y figura 56, se aprecia los resultados de la dimensión integridad de la calidad del software. En ese sentido, se realizó una encuesta a los colaboradores del área tecnológica, los resultados para el pretest muestran que para 79 la mejora de la dimensión integridad de la calidad del software se encuentra en un nivel deficiente en 70,5%, para 30 colaboradores en nivel regular en un 26,8% y para 3 personas en un nivel eficiente lo cual representa 2,7%. Se puede apreciar que el nivel preponderante fue bajo en el pretest.

Por otro lado, en el postest se aprecian los siguientes resultados, para 28 colaboradores la variable mejora de la dimensión integridad de la calidad del software se encuentra en un nivel deficiente en 25,0%, para 37 colaboradores en nivel regular en un 33,0% y para 47 personas en un nivel alto lo cual representa 42,0%. Se puede apreciar que el nivel preponderante fue el nivel eficiente, lo cual demuestra que para los colaboradores del área tecnológica de la dimensión integridad de la calidad del software fue la adecuada.

El análisis inferencial se realizó después del análisis descriptivo. Teniendo en cuenta que el número de módulos considerados era inferior a 50, se aplicó para este fin la prueba de Kolmogórov-Smirnov. En este caso se utilizó el mismo 95% de certeza.

Regla de decisión:

Si:

Sig. < 0.05 adopta una distribución no normal.

Sig. ≥ 0.05 adopta una distribución normal.

Dónde:

Sig.: P-valor o nivel crítico del contraste.

Se realizó la prueba de normalidad para la calidad de software

H0: Los datos de la calidad de software no siguen una distribución normal.

Ha: Los datos de la calidad de software siguen una distribución normal.

Tabla 23

Prueba de normalidad de la calidad de software

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Calidad de software _Pretest	,962	112	,000
Calidad de software _Postest	,967	112	,000

a. Corrección de significación de Lilliefors

De acuerdo con la tabla N° 23, se aprecia que los resultados del pretest y postest de la calidad de software obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que el indicador no se distribuye normalmente.

Asimismo, se realizó la prueba de normalidad para la dimensión eficiencia de la calidad de software

H0: Los datos de la dimensión eficiencia de la calidad de software no siguen una distribución normal.

Ha: Los datos de la dimensión eficiencia de la calidad de software siguen una distribución normal.

Tabla 24

Prueba de normalidad de la dimensión eficiencia de la calidad de software

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Eficiencia _Pretest	,922	112	,000
Eficiencia _Postest	,911	112	,000

a. Corrección de significación de Lilliefors

De acuerdo con la tabla N° 24, se aprecia que los resultados del pretest y postest de la dimensión eficiencia de la confiablidad de software obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión no se distribuye normalmente.

4.3.10. Prueba de normalidad de la variable calidad de software y sus dimensiones

Se realizó la prueba de normalidad para la dimensión confiabilidad de la calidad de software

H0: Los datos de la dimensión confiablidad de la calidad de software no siguen una distribución normal.

Ha: Los datos de la dimensión confiablidad de la calidad de software siguen una distribución normal.

Tabla 25

Prueba de normalidad de la dimensión confiablidad de la calidad de software

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Confiablidad _Pretest	,987	112	,000
Confiablidad _Postest	,901	112	,000

a. Corrección de significación de Lilliefors

De acuerdo con la tabla N° 25, se aprecia que los resultados del pretest y postest de la dimensión confiablidad de la calidad software obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión no se distribuye normalmente.

Se realizó la prueba de normalidad para la dimensión integridad de la calidad de software

H0: Los datos de la dimensión integridad de la calidad de software no siguen una distribución normal.

Ha: Los datos de la dimensión integridad de la calidad de software siguen una distribución normal.

Tabla 26

Prueba de normalidad de la dimensión integridad de la calidad de software

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Integridad _Pretest	,987	112	,000
Integridad _Postest	,901	112	,000

a. Corrección de significación de Lilliefors

De acuerdo con la tabla N° 26, se aprecia que los resultados del pretest y postest de la dimensión integridad de la calidad software obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión no se distribuye normalmente.

El valor p de cada dimensión fue inferior a 0,05 lo que indica que no se distribuyen normalmente. Debido a esto, se utilizó la prueba de Wilcoxon. Para realizar la prueba de hipótesis de los indicadores, se siguió la siguiente regla de decisión:

Si p-valor <= α se rechaza la hipótesis nula.

Si p-valor > α no se rechaza la hipótesis nula.

Donde:

α= 0.05 el margen de error de la prueba de hipótesis

Nivel de confianza = 95%

4.3.11. Prueba de hipótesis de la variable calidad de software y sus dimensiones

Se realizó la prueba de hipótesis para la hipótesis general

H0: La calidad del software en una entidad bancaria en Lima metropolitana no mejora significativamente con la arquitectura de microservicios.

Ha: La calidad del software en una entidad bancaria en Lima metropolitana mejora significativamente con la arquitectura de microservicios.

Tabla 27

Prueba de Wilcoxon para la calidad de software

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Calidad de software Pretest y Postest	-5,161	,000

Fuente: Base de datos SPSS.

Se aprecia en la tabla 27, la prueba de hipótesis, lo cual permite ver el valor Sig., de la calidad de software es 0,00, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, La calidad del software en una entidad bancaria en Lima metropolitana mejora significativamente con la arquitectura de microservicios.

Se realizó la prueba de hipótesis para la dimensión eficiencia de la calidad de software

H0: La arquitectura de microservicios no incide en la eficiencia del software en una entidad bancaria Lima Metropolitana.

Ha: La arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima Metropolitana.

Tabla 28

Prueba de Wilcoxon para la dimensión eficiencia de la calidad de software

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Eficiencia de la calidad de software Pretest y Postest	-3,141	,001

Fuente: Base de datos SPSS.

Se aprecia en la tabla 28, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión eficiencia de la eficiencia de la calidad de software es 0,01, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de microservicios incide en la confiabilidad de la calidad de software en una entidad bancaria Lima Metropolitana.

Igualmente, se realizó la prueba de hipótesis para la dimensión confiabilidad de la calidad de software

H0: La arquitectura de microservicios no incide en la confiabilidad del software en una entidad bancaria Lima Metropolitana.

Ha: La arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima Metropolitana.

Tabla 29

Prueba de Wilcoxon para la dimensión confiabilidad de la calidad de software

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Confiabilidad de la calidad de software Pretest y Postest	-7,261	,000

Fuente: Base de datos SPSS.

Se aprecia en la tabla 29, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión eficiencia de la confiabilidad de la calidad software es 0,00, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. La arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima Metropolitana.

Finalmente, se realizó la prueba de hipótesis para la dimensión integridad de la calidad de software

H0: La arquitectura de microservicios no incide en la integridad del software en una entidad bancaria Lima Metropolitana.

Ha: La arquitectura de microservicios incide en la integridad del software en una entidad bancaria Lima Metropolitana.

Tabla 30

Prueba de Wilcoxon para la dimensión integridad de la calidad de software

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Integridad de la calidad de software Pretest y Postest	-7,142	,000

Fuente: Base de datos SPSS.

Se aprecia en la tabla 30, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión eficiencia de la integridad de la calidad de software es 0,00, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de microservicios incide en la integridad del software en una entidad bancaria Lima Metropolitana.

4.4. Discusión de resultados

Con relación a los resultados descriptivos, para la variable calidad de software, se observó que antes de la implementación de una arquitectura de microservicio los niveles eran : para43 colaboradores, nivel deficiente en 38,%, para 54 colaboradores en nivel regular en un 48,2% y para 15 personas en un nivel eficiente lo cual representa 13,4%. Luego de la implementación de la arquitectura de microservicios los resultados fueron: para 23 colaboradores nivel deficiente en 20,5%, para 35 colaboradores en nivel regular en un 31,3% y para 54 personas en un nivel eficiente lo cual representa 48,2%. Estos resultados se asemejan con la investigación de Flores (2020) quien buscó implementar una arquitectura de microservicios mediante Spring Cloud para el desarrollo del módulo de registro y seguimiento médico de los deportistas en la Federación Deportiva de Imbabura. Los resultados mostraron que el rendimiento de la calidad de software luego de la implementación obtuvo un valor de 80.86%. En ese sentido, se concluyó que la arquitectura de microservicios mejora el rendimiento de la calidad de software. Se pudo apreciar que en ambas investigaciones el rendimiento de la calidad de software mejoró de manera sustancial al implementar una arquitectura de microservicios la cual permitió que las aplicaciones rindan de una manera óptima.

De igual manera, con relación a los resultados descriptivos, para la dimensión eficiencia de la calidad de software, se obtuvo que antes de la implementación de la arquitectura de microservicios los resultados mostraban que para 43 la mejora de la dimensión eficiencia calidad del software se encuentra en un nivel deficiente en 38,%, para 54 colaboradores en nivel regular en un 48,2% y para 15 personas en un nivel eficiente lo cual representa 13,4%. Por otro lado en el postest se aprecian los siguientes resultados, para 23 colaboradores la variable mejora de la dimensión eficiencia de la calidad del software se encuentra en un nivel deficiente en 20,5%, para 35 colaboradores en nivel regular en un 31,3% y para 54 personas en un nivel eficiente lo cual representa 48,2%. Se puede apreciar que el nivel preponderante fue el nivel alto, lo cual demuestra que para los colaboradores del área tecnológica de la dimensión eficiencia de la calidad del software fue la adecuada.

Estos resultados se asemejan con la investigación de Collado (2020) quien tuvo por finalidad determinar el impacto de una arquitectura de microservicios para el aseguramiento de la calidad del software en el Ministerio de la Mujer y Poblaciones Vulnerables. Los resultados obtenidos muestran una aumento en la dimensión eficiencia de la calidad del software de 34.91%,. Por lo cual se concluyó que la arquitectura de microservicios mejoró la eficiencia de la calidad del software. Se pudo apreciar que en ambas investigaciones la eficiencia se encontraba en un índice bajo cuando el uso estaba bajo una arquitectura monolítica lo cual cambio con la implementación de los microservicios lo cual permitió a los sistemas seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes.

Asimismo, con relación a los resultados descriptivos, para la dimensión confiabilidad de la calidad de la calidad de software, se obtuvo que antes de la implementación de la arquitectura de microservicios, los resultados mostraban que para 76 la mejora de la dimensión confiabilidad calidad del software se encuentra en un nivel deficiente en 67,9%, para 32 colaboradores en nivel regular en un 28,6% y para 4 personas en un nivel eficiente lo cual representa 3,6%. Se puede apreciar que el nivel preponderante fue bajo en el pretest. Por otro lado en el postest se aprecian los siguientes resultados, para 20 colaboradores la variable mejora de la dimensión confiabilidad de la calidad del software se encuentra en un nivel deficiente en 18,6%, para 25 colaboradores en nivel regular en un 23,9% y para 67 personas en un nivel alto lo cual representa 57,5%. Se puede apreciar que el nivel preponderante fue el nivel eficiente, lo cual demuestra que para los colaboradores del área tecnológica de la dimensión confiabilidad de la calidad del software fue la adecuada.

Estos resultados se asemejan con la investigación de Ruiz (2020) quien tuvo por finalidad determinar el efecto de la utilización del Modelo Emergente de Microservicios para el desarrollo de sistemas informáticos eficientes. Los resultados muestran que luego de la implementación de la arquitectura de microservicios la confiabilidad mejoró en un 75.17%. En ese sentido, el estudio concluyó que, el utilizar el Modelo Emergente para el desarrollo de sistemas informáticos mejora la disponibilidad de la calidad del software. Se pudo apreciar que en ambas investigaciones la disponibilidad se encontraba en un índice bajo previo cuando el uso estaba bajo una arquitectura monolítica lo cual cambio con la implementación de

los microservicios por lo cual las aplicaciones mostraron una mejor funcionalidad. Asimismo, la infraestructura, se mostró más robusta lo cual permitió tener resultados óptimos.

Finalmente, con relación a los resultados descriptivos, para la dimensión integridad de la calidad de la calidad de software, se obtuvo que antes de la implementación de la arquitectura de microservicios, los resultados mostraban que para 79 la mejora de la dimensión integridad de la calidad del software se encuentra en un nivel deficiente en 70,5%, para 30 colaboradores en nivel regular en un 26,8% y para 3 personas en un nivel eficiente lo cual representa 2,7%. Se puede apreciar que el nivel preponderante fue bajo en el pretest. Por otro lado en el postest se aprecian los siguientes resultados, para 28 colaboradores la variable mejora de la dimensión integridad de la calidad del software se encuentra en un nivel deficiente en 25,0%, para 37 colaboradores en nivel regular en un 33,0% y para 47 personas en un nivel alto lo cual representa 42,0%. Se puede apreciar que el nivel preponderante fue el nivel eficiente, lo cual demuestra que para los colaboradores del área tecnológica de la dimensión integridad de la calidad del software fue la adecuada.

Estos resultados se asemejan con la investigación de Guevara (2018) quien tuvo por finalidad determinar el efecto de la utilización de los Microservicios en la calidad de software. Los resultados muestran que luego de la implementación de la arquitectura de microservicios la integridad mejoró en un 45.11%. En ese sentido, el estudio concluyó que, el utilizar microservicios mejora la disponibilidad de la calidad del software. Se pudo apreciar que en ambas investigaciones la integridad se encontraba en un índice bajo previo cuando el uso estaba bajo una arquitectura monolítica lo cual cambio con la implementación de los microservicios por lo cual las aplicaciones mostraron una mejor funcionalidad.

V. CONCLUSIONES

Primera: La implementación de la arquitectura de microservicio mejoró la calidad de software, lo cual se confirma con los resultados descriptivos. En ese sentido, se obtuvo que se encontraba en nivel deficiente en 38.4%, regular 48.2% y eficiente 13.4%. Luego de la implementación del microservicio los resultados fueron, nivel deficiente 20.5%, nivel regular 31.3% y nivel eficiente 48.2%. De igual manera se obtuvo que el valor de significancia p <0.05. En se sentido, se rechazó la hipótesis nula y se aceptó la alterna es decir una arquitectura de microservicio incide en la mejora de la calidad de software.

Segunda: La implementación de la arquitectura de microservicio mejoró la dimensión eficiencia de la calidad de software, lo cual se confirma con los resultados descriptivos. En ese sentido, se obtuvo que se encontraba en nivel deficiente en 60.7%, regular 30.4% y eficiente 8.9%. Luego de la implementación del microservicio los resultados fueron, nivel deficiente 2.7%, nivel regular 36.6% y nivel eficiente 60.7%. De igual manera se obtuvo que el valor de significancia p <0.05. En se sentido, se rechazó la hipótesis nula y se aceptó la alterna es decir una arquitectura de microservicio incide en la dimensión eficiencia de la calidad de software.

Tercera: La implementación de la arquitectura de microservicio mejoró la dimensión confiabilidad de la calidad de software, lo cual se confirma con los resultados descriptivos. En ese sentido, se obtuvo que se encontraba en nivel deficiente en 67.9%, regular 28.6% y eficiente 3.6.% Luego de la implementación del microservicio los resultados fueron, nivel deficiente 18.6%, nivel regular 23.9% y nivel eficiente 57.5%. De igual manera se obtuvo que el valor de significancia p <0.05. En se sentido, se rechazó la hipótesis nula y se aceptó la alterna es decir una arquitectura de microservicio incide en la dimensión confiabilidad de la calidad de software.

Cuarta: La implementación de la arquitectura de microservicio mejoró la dimensión integridad de la calidad de software, lo cual se confirma con los resultados descriptivos. En ese sentido, se obtuvo que se encontraba en nivel deficiente en 70.5%, regular 26.8% y eficiente 2.7.% Luego de la implementación del microservicio los resultados fueron, nivel deficiente 25%, nivel regular 33% y nivel eficiente 42%. De igual manera se obtuvo que el valor de significancia p <0.05. En se sentido, se rechazó la hipótesis nula y se

aceptó la alterna es decir una arquitectura de microservicio incide en la dimensión integridad de la calidad de software.

VI. RECOMENDACIONES

Primera: Se recomienda al área tecnológica del Banco implementar cultura DevOps en el área de desarrollo. Ya que, al implementarse una Arquitectura de Microservicios, queda abierto implementaciones de herramientas y estándares que tienen realización con CI/CD (Integración continua/ entrega contigua)

Segunda: Se recomienda realizar la migración de las aplicaciones monolíticas de la entidad financiera a la nueva arquitectura de microservicio.

Tercera: Se recomienda la utilización de metodologías agiles para desarrollo de nuevos proyectos. Se sugiera la adopción de marco de trabajo de agiles o SCRUM de forma de transversal a todas las áreas de la entidad financiera.

Cuarta: Se recomienda evaluar si es necesario aplicar una arquitectura de microservicios en la empresa de turno, ya que la arquitectura de microservicios está pensada para empresas que posean sistemas complejos como también servidores potentes que soporten contenedores como también orquestadores desplegados.

REFERENCIAS BIBLIOGRÁFICAS

Alfonzo, R. y Luull, L (2021). Módulo de recomendación de patrones de diseño para EGPat. *Revista Cubana de Ciencias Informáticas*, 15 (2): 118-137

Asociación de Bancos del Perú. (2022). *Principales organizaciones del sector financiero fortalecen acuerdo de cooperación en educación financiera y ciberseguridad*. Obtenido de Principales organizaciones del sector financiero fortalecen acuerdo de cooperación en educación financiera y ciberseguridad

Cabezas, R. (2019). *Implementación de un sistema web contable para la empresa central de gestión de negocios S.A.C.-Huaraz; 2019.* Huaraz: Universidad Los Angeles de Chimbote. Obtenido de http://repositorio.uladech.edu.pe/handle/20.500.13032/11248

Carrasco, D. (2016). *Metodología de la investigación.* Lima: San Marcos E.I.R.L.

Chiavenato, I. (2006). *Introducción a la teoría general de la administración 7ma Edición.* Mexico: McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V.

Collado, O. (2020). *Arquitectura de microservicios para el aseguramiento de la calidad del software en el Ministerio de la Mujer y Poblaciones Vulnerables.* Lima: Universidad César Vallejo. Obtenido de https://repositorio.ucv.edu.pe/handle/20.500.12692/56992

Estela, M. (2019). *Sistema de información.* México: Edición 11.

Flores, J. (2020). *Estudio de una arquitectura de microservicios mediante Spring Cloud para el desarrollo del módulo de registro y seguimiento médico de los deportistas en la Federación deportiva de Imbabura.* Ecuador: Universidad Técnica del Norte. Obtenido de http://repositorio.utn.edu.ec/handle/123456789/10445

Fondo Monetario Internacional. (2022). *Divergencia creciente: Se ahondan las brechas en la recuperación mundial*. Obtenido de

https://www.imf.org/es/Blogs/Articles/2021/07/27/blogs-drawing-further-apart-widening-gaps-in-the-global-recovery

Frayssinet, M. (2016). *Taller de Implementación de la norma ISO 27001*. Lima: Oficina Nacional de Gobierno Electrónico e Informática.

Galin, D. (2018). *Software Quality. Concepts and Practice*. . USA: Editorial Office.

Guevara, C. (2018). *Estudio metodológico para el desarrollo de microservicios, diseño de un prototipo.* Ecuador: Universidad Técnica de Cotopaxi. Obtenido de http://repositorio.utc.edu.ec/handle/27000/8703

Latorre, M. (2018). *Historia de las Web, 1.0, 2.0, 3.0 y 4.0.* Lima: Universidad Marcelino Champagnat.

Laurentis, R. (2018). *El libro del BPM 2018: tecnologias,conceptos, enfoques, metodologicos y estandares.* España: Club BPM.

Lopez, J. (2018). *Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la Asamblea Nacional.* Ecuador: Universidad Técnica del Norte. Obtenido de http://repositorio.utn.edu.ec/handle/123456789/7603

Newman, S. (2016). *Building Microservices: Designing fine-grained Systems.* USA: O'Reilly Media.

Newman, S. (2019). *Monolith to microservices: Evolutionary patterns to transform your monolith*. Rusia: O'Reilly Media.

Prasad, S. (2018). *Beginning Spring Boot 2: Applications and Microservices with the Spring Framework*. India: Apress Media.

Richardson, C. (2017). *Microservices form Design to Development*. USA: Nginx.

Rojas, H. (2021). *Transformación digital, información y cohesión del sistema político*. Colombia: Universidad Externado de Colombia. Obtenido de https://bdigital.uexternado.edu.co/entities/publication/5443e4aa-d103-48c2-add9-eb0a6a618ae1

Ruiz, P. (2020). *Modelo emergente para el desarrollo de sistemas informáticos eficientes, utilizando una arquitectura de microservicios en Sunat. División de*

desarrollo de software aduanero. Huancayo: Universidad Nacional del Centro del Perú. Obtenido de https://repositorio.uncp.edu.pe/handle/20.500.12894/6618

Sánchez, N., Comas, R. y García, M. (2019). Sistema Inteligente de Información Geográfica para las empresas eléctricas cubanas. Ingeniare. *Rev. chil. ing.* 27 (2): 197-209 http://dx.doi.org/10.4067/S0718-33052019000200197

Thornsby, J. (2017). *Operadores de Programación Reactiva en RxJava 2.* Obtenido de https://code.tutsplus.com/es/tutorials/reactive-programming-operators-in-rxjava-20--cms-28396.

Villaizán, H. (2019). *Arquitectura de software basada en microservicios para implementación de la aplicación web de cobranza digital en Financial Systems Company SAC.* Huancayo: Universidad Continental. Obtenido de https://repositorio.continental.edu.pe/handle/20.500.12394/6387

Visitación, P. (2018). *Sistema de información web para la gestión de historias clínicas de los puestos de salud de la red Huaylas Sur, Huaraz, 2018.* Huaraz: Universidad Nacional Santiago Antunez de Mayolo. Obtenido de http://www.repositorio.unasam.edu.pe/handle/UNASAM/2668

Wem, S., Zomaya, A., & Yang, L. (2020). *Algorithms and Architectures for Parallel Processing*. Alemania: Springer Nature.

Wolff, E. (2018). *Microservices: Flexible software Architecture*. USA: Addison-Wesley.

ANEXOS

ANEXO 1: "MATRIZ DE CONSISTENCIA DEL PROYECTO"

Problemas	Objetivos	Hipótesis	Variables	Metodología
Problema general ¿De qué manera se mejora la calidad del software en una entidad bancaria Lima metropolitana con la arquitectura de microservicios?	**Objetivo general** Mejorar la calidad de software mediante Arquitectura de Microservicios en una entidad bancaria Lima metropolitana	**Hipótesis general** La arquitectura de microservicios incide en la mejora de la calidad del *software* en una entidad bancaria Lima metropolitana	**Variable independiente:** Arquitectura de Microservicios **Variable dependiente:** Mejora de la calidad del *software*	Tipo de investigación Aplicada Nivel de investigación Preexperimental Diseño de investigación Pre - Experimental Población 158 trabajadores Muestra 112 trabajadores Muestreo Aleatorio Simple. Técnica Fichaje Instrumento Ficha de registro Método análisis de datos Wilcoxon Prueba de normalidad Kolmogórov-Smirnov
Problemas específicos	**Objetivos específicos**	**Hipótesis específicas**		
a. ¿De qué manera la arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima metropolitana? b. ¿De qué manera la arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima metropolitana? c. ¿De qué manera la arquitectura de	a. Determinar cómo una arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima metropolitana. b. Determinar cómo una arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima metropolitana.	a. La arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima metropolitana. b. La arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima metropolitana. c. La arquitectura de microservicios incide		

microservicios incide en la integridad del software en una entidad bancaria Lima metropolitana?	c. Determinar cómo una arquitectura de microservicios incide en la integridad del software en una entidad bancaria en Lima metropolitana.	en la integridad del software en una entidad bancaria Lima metropolitana.		

Fuente: *Elaboración Propia*

ANEXO 2: "INSTRUMENTO DE RECOLECCIÓN DE DATOS"

UNIVERSIDAD NACIONAL

SANTIAGO ANTÚNEZ DE MAYOLO

Escuela profesional de Ingeniería de Sistemas e Informática

.

El presente cuestionario tiene por finalidad recoger información de los trabajadores del sector financiero, en calidad de trabajador, le solicito de forma objetiva poder completar con la siguiente encuesta, de manera que se pueda Determinar como una arquitectura de microservicios incide en la mejora de la calidad del *software* en una entidad bancaria, Lima – 2022.Se le agradece anticipadamente la información que usted proporcione. La presente encuesta tendrá carácter secreta y anónima; no tendrá preguntas buenas ni malas; marcar con una X la opción que usted crea conveniente.

Escala de Valoración				
Muy frecuentemente	**Frecuentemente**	**Ocasionalmente**	**Raramente**	**Nunca**
1	**2**	**3**	**4**	**5**

Nº	**CRITERIOS**	**1**	**2**	**3**	**4**	**5**
Variable: Arquitectura de microservicios						
Facilidad						
1	¿Considera que la arquitectura de microservicios incrementará la seguridad del sistema para bienestar del cliente?					
2	¿Considera usted que con la arquitectura de microservicios el tiempo de aprendizaje de los aplicativos será menor?					
3	¿usted cree que con la arquitectura de servicios el tiempo de despliegue de las actualizaciones de los aplicativos ser mayor a la que existe actualmente?					
Disponibilidad						
4	¿Usted cree que con la arquitectura de microservicios personas extrañas a la entidad podrán tener disponibilidad de los servicios internos?					
5	¿Usted cree que con la arquitectura se podrá conocer los avances de los servicios externos y su disponibilidad?					
6	¿usted cree que con la arquitectura de microservicios la disponibilidad de la información será en tiempo real e inmediata?					
Flexibilidad						
7	¿usted cree que con la arquitectura de microservicios flexibilizara los requisitos para obtener información?					
8	¿usted cree que la flexibilización de la información será de apoyo para el usuario?					
9	¿usted cree que se podrá garantizar que la información por medio de la arquitectura de microservicios?					

10	¿usted cree que la arquitectura de microservicios se adaptara y solucionara las necesidades del usuario?					
Agilidad						
11	¿usted cree que la arquitectura de microservicios mejorara la agilidad que tiene el usuario en emplear los aplicativos?					
12	¿usted cree que la arquitectura de microservicios debe de fomentar la innovación en los servicios de la empresa?					
13	¡usted cree que se debe de realizar una mejora continua a la arquitectura de microservicios?					
Variable: Mejora de la calidad del *software*						
Eficiencia						
14	¿Usted cree que la mejora de la calidad del software es influenciada por su eficiencia?					
15	¿Usted cree que el rendimiento será optimo si se produce mejoras a la calidad del *software*?					
16	¿Usted cree que las respuestas rápidas deben estar incluidas en las mejoras que se le brinde al software?					
Confiabilidad						
17	¿Usted considera que con las mejoras a la calidad del software el cliente tendrá confiabilidad en el nuevo sistema					
18	¿Esta nueva arquitectura estará a disposición a cualquier hora del día para que el cliente puede utilizarlo?					
19	¿Antes de usarse y entra en funcionamiento el software debe de capacitarse al personal de la entidad?					
Integridad						
20	¿Usted cree que la calidad del software debe estar basado en la integridad de la información que se almacenara?					
21	¿Usted cree para procesar la información de manera correcta debe asegurarse su integridad?					
22	¿Usted cree que el software debe de asegurarse que la información almacenada sea protegida de cualquier eventualidad?					

Fuente: Elaboración Propia.

ANEXO 3: "VALIDACIÓN DE EXPERTOS"

UNIVERSIDAD NACIONAL SANTIAGO ANTÚNEZ DE MAYOLO

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE INGENIERIA DE SISTEMAS E INFORMÁTICA

Huaraz 30 de noviembre del 2022

SEÑOR: Enoc Eusebio Nina Cuchillo

Yo, Daniel Noel Caqui Mejía, identificado con DNI N° 72560873, Bachiller en Ingeniería de Sistemas e Informática, me dirijo a usted con la finalidad de solicitar su valiosa colaboración en la validación de contenido de los ítems que conforman el instrumento de recolección de datos que utilizaré para recabar la información requerida en la investigación titulada "ARQUITECTURA DE MICROSERVICIOS PARA MEJORAR LA CALIDAD DE SOFTWARE EN UNA ENTIDAD BANCARIA DE LIMA METROPOLITANA, 2022". Por lo cual, facilito la documentación pertinente:

1. Matriz de Operacionalización de Variables.
2. Matriz de Consistencia
3. Instrumento de Recolección de Datos.

Por su experiencia profesional y méritos académicos me permito para la validación de dicho instrumento.

Agradezco de antemano su valioso aporte.

Atentamente

Daniel Noel Caqui Mejía
DNI N°: 72560873

1. Matriz de operacionalización de Variables:

VARIABLE	DEFINICIÓN CONCEPTUAL	DIMENSIONES	INDICADORES	ESCALA DE MEDICIÓN
X: VARIABLE INDEPENDIENTE Arquitectura de Microservicios	Según Estela (2016) es un conjunto de herramientas tecnologías, procedimientos y recursos que se enfocan en el control financiero, basado en las tecnologías tradicionales con el uso de n-capas ejecutándose en un mismo contexto.	Facilidad	• Tiempo de aprendizaje • Tiempo de despliegue	Medición: ordinal
		Disponibilidad	• Interna • Externa	Medición: ordinal
		Flexibilidad	• Adaptación • Apoyo • Garantía	Medición: ordinal
		Agilidad	• Innovación • Mejora continua	Medición: ordinal
Y: VARIABLE DEPENDIENTE Mejora de la calidad del software	Esto es lo que dice la fuente no válida Sánchez, Comas y García. Que los procesos de software sean adecuados para desarrollar una calidad adecuada en los productos de software, para sus servicios de operación esperados y para cumplir con los requisitos de programación y mantenimiento del presupuesto se establece mediante una estandarización de los pasos que componen el proceso de software.	Eficiencia	• Rendimiento • Respuesta rápida • Recursos	Medición: ordinal
		Confiabilidad	• Disponibilidad • Usabilidad	Medición: ordinal
		Integridad	• Procesar • Almacenar	Medición: ordinal

2. Matriz de Consistencia:

Problemas	Objetivos	Hipótesis	Variables	Metodología
Problema general ¿De qué manera se mejora la calidad del software en una entidad bancaria Lima metropolitana con la arquitectura de microservicios?	**Objetivo general** Mejorar la calidad de software mediante Arquitectura de Microservicios en una entidad bancaria Lima metropolitana	**Hipótesis general** La arquitectura de microservicios incide en la mejora de la calidad del *software* en una entidad bancaria Lima metropolitana	**Variable independiente:** Arquitectura de Microservicios **Variable dependiente:** Mejora de la calidad del *software*	Tipo de investigación Aplicada Nivel de investigación Experimental Diseño de investigación Pre - Experimental Población 158 trabajadores Muestra 112 trabajadores Muestreo Aleatorio Simple. Técnica Fichaje Instrumento Ficha de registro
Problemas específicos	**Objetivos específicos**	**Hipótesis específicas**		
a. ¿De qué manera la arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima metropolitana? b. ¿De qué manera la arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima metropolitana? c. ¿De qué manera la arquitectura de microservicios incide en la integridad del software en	a. Determinar cómo una arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima metropolitana. b. Determinar cómo una arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima metropolitana. c. Determinar cómo una arquitectura de	a. La arquitectura de microservicios incide en la eficiencia del software en una entidad bancaria Lima metropolitana. b. La arquitectura de microservicios incide en la confiabilidad del software en una entidad bancaria Lima metropolitana. c. La arquitectura de microservicios incide en la integridad del software en una		

una entidad bancaria Lima metropolitana?	microservicios incide en la integridad del software en una entidad bancaria en Lima metropolitana.	entidad bancaria Lima metropolitana.		Método análisis de datos T-Student Prueba de normalidad Shapiro Wilk

3. Instrumento de Recolección de Datos

CUESTIONARIO

UNIVERSIDAD NACIONAL
SANTIAGO ANTÚNEZ DE MAYOLO
Escuela profesional de Ingeniería de Sistemas e Informática

El presente cuestionario tiene por finalidad recoger información de los trabajadores del sector financiero, en calidad de trabajador, le solicito de forma objetiva poder completar con la siguiente encuesta, de manera que se pueda Determinar como una arquitectura de microservicios incide en la mejora de la calidad del *software* en una entidad bancaria, Lima – 2022.Se le agradece anticipadamente la información que usted proporcione. La presente encuesta tendrá carácter secreta y anónima; no tendrá preguntas buenas ni malas; marcar con una X la opción que usted crea conveniente.

Escala de Valoración				
Muy frecuentemente	**Frecuentemente**	**Ocasionalmente**	**Raramente**	**Nunca**
1	**2**	**3**	**4**	**5**

Nº	CRITERIOS	1	2	3	4	5
Variable: Arquitectura de microservicios						
Facilidad						
1	¿Considera que la arquitectura de microservicios incrementará la seguridad del sistema para bienestar del cliente?					
2	¿Considera usted que con la arquitectura de microservicios el tiempo de aprendizaje de los aplicativos será menor?					
3	¿usted cree que con la arquitectura de servicios el tiempo de despliegue de las actualizaciones de los aplicativos ser mayor a la que existe actualmente?					
Disponibilidad						
4	¿Usted cree que con la arquitectura de microservicios personas extrañas a la entidad podrán tener disponibilidad de los servicios internos?					
5	¿Usted cree que con la arquitectura se podrá conocer los avances de los servicios externos y su disponibilidad?					
6	¿usted cree que con la arquitectura de microservicios la disponibilidad de la información será en tiempo real e inmediata?					
Flexibilidad						

7	¿usted cree que con la arquitectura de microservicios flexibilizara los requisitos para obtener información?					
8	¿usted cree que la flexibilización de la información será de apoyo para el usuario?					
9	¿usted cree que se podrá garantizar que la información por medio de la arquitectura de microservicios?					
10	¿usted cree que la arquitectura de microservicios se adaptara y solucionara las necesidades del usuario?					
Agilidad						
11	¿usted cree que la arquitectura de microservicios mejorara la agilidad que tiene el usuario en emplear los aplicativos?					
12	¿usted cree que la arquitectura de microservicios debe de fomentar la innovación en los servicios de la empresa?					
13	¡usted cree que se debe de realizar una mejora continua a la arquitectura de microservicios?					
Variable: Mejora de la calidad del *software*						
Eficiencia						
14	¿Usted cree que la mejora de la calidad del software es influenciada por su eficiencia?					
15	¿Usted cree que el rendimiento será optimo si se produce mejoras a la calidad del *software*?					
16	¿Usted cree que las respuestas rápidas deben estar incluidas en las mejoras que se le brinde al software?					
Confiabilidad						
17	¿Usted considera que con las mejoras a la calidad del software el cliente tendrá confiabilidad en el nuevo sistema					
18	¿Esta nueva arquitectura estará a disposición a cualquier hora del día para que el cliente puede utilizarlo?					
19	¿Antes de usarse y entra en funcionamiento el software debe de capacitarse al personal de la entidad?					
Integridad						
20	¿Usted cree que la calidad del software debe estar basado en la integridad de la información que se almacenara?					
21	¿Usted cree para procesar la información de manera correcta debe asegurarse su integridad?					
22	¿Usted cree que el software debe de asegurarse que la información almacenada sea protegida de cualquier eventualidad?					

Fuente: Elaboración Propia.

INFORME DE OPINIÓN DE EXPERTO

I. DATOS DEL EXPERTO

APELLIDOS Y NOMBRES: Nina Cuchillo, Enoc Eusebio

PROFESIÓN: Ingeniero de Sistemas

GRADO ACADÉMICO: Magíster

MENCIÓN: Docencia Universitaria

CENTRO LABORAL: CINFO UNMSN

CARGO: Docente de tecnologías de la información

II. MATRIZ DE EVALUACIÓN DEL INSTRUMENTO DE RECOLECCIÓN DE DATOS

Indicador	Criterio	Deficiente 0 - 20	Regular 21 - 40	Bueno 41 - 60	Muy Bueno 61 - 80	Excelente 81 - 100
Claridad	Está formulado con un lenguaje claro.					95
Objetividad	No presenta sesgo ni induce a respuestas.					85
Actualidad	Está de acuerdo con los avances de la teoría, ciencia y tecnología.					90
Organización	Existe una organización lógica y coherente de los ítems.				80	
Suficiencia	Comprende las dimensiones de la investigación en cantidad y calidad.					85
Intencionalidad	Adecuado para establecer asociación.					85
Consistencia	Basado en aspectos teóricos y científicos.				80	
Coherencia	Hay relación entre variables, dimensiones e indicadores.					85
Metodología	El instrumento se relaciona con el método planteado en el proyecto.					94

Liam, 30 de noviembre del 2022

Mg. Enoc Eusebio Nina Cuchillo
MAGISTER EN DOCENCIA UNIVERSITARIA
Especialista en Tecnologia de la informacion

Ing. Enoc Eusebio Nina Cuchillo
N° de DNI: 43513309

INFORME DE OPINIÓN DE EXPERTO

I. DATOS DEL EXPERTO

APELLIDOS Y NOMBRES: ALVARADO TOLENTINO JOSEPH DARWIN

PROFESIÓN: INGENIERO DE SISTEMAS E INFORMÁTICA

GRADO ACADÉMICO: MAESTRO EN CIENCIAS E INGENIERÍA

MENCIÓN: AUDITORIA Y SEGURIDAD INFORMÁTICA

CENTRO LABORAL: UNASAM

CARGO: DOCENTE

II. MATRIZ DE EVALUACIÓN DEL INSTRUMENTO DE RECOLECCIÓN DE DATOS

Indicador	Criterio	Deficiente 0 - 20	Regular 21 - 40	Bueno 41 - 60	Muy Bueno 61 - 80	Excelente 81 - 100
Claridad	Está formulado con un lenguaje claro.					X
Objetividad	No presenta sesgo ni induce a respuestas.				X	
Actualidad	Está de acuerdo con los avances de la teoría, ciencia y tecnología.					X
Organización	Existe una organización lógica y coherente de los ítems.				X	
Suficiencia	Comprende las dimensiones de la investigación en cantidad y calidad.					X
Intencionalidad	Adecuado para establecer asociación.				X	
Consistencia	Basado en aspectos teóricos y científicos.					X
Coherencia	Hay relación entre variables, dimensiones e indicadores.					X
Metodología	El instrumento se relaciona con el método planteado en el proyecto.				X	

Huaraz, 2 de diciembre del 2022

Mag. Joseph Darwin Alvarado Tolentino
N° de DNI: 46022813

INFORME DE OPINIÓN DE EXPERTO

I. DATOS DEL EXPERTO

APELLIDOS Y NOMBRES: Jorge Luis Lavalle Díaz

PROFESIÓN: Ingeniero de Sistemas

GRADO ACADÉMICO: Magister en Ingeniería de Sistemas

MENCIÓN: Tecnologías de la Información

CENTRO LABORAL: CANVIA

CARGO: Analista de Sistemas

II. MATRIZ DE EVALUACIÓN DEL INSTRUMENTO DE RECOLECCIÓN DE DATOS

Indicador	Criterio	Deficiente 0 - 20	Regular 21 - 40	Bueno 41 - 60	Muy Bueno 61 - 80	Excelente 81 - 100
Claridad	Está formulado con un lenguaje claro.			60		
Objetividad	No presenta sesgo ni induce a respuestas.			60		
Actualidad	Está de acuerdo con los avances de la teoría, ciencia y tecnología.				75	
Organización	Existe una organización lógica y coherente de los ítems.				65	
Suficiencia	Comprende las dimensiones de la investigación en cantidad y calidad.				65	
Intencionalidad	Adecuado para establecer asociación.				65	
Consistencia	Basado en aspectos teóricos y científicos.			59		
Coherencia	Hay relación entre variables, dimensiones e indicadores.			60		
Metodología	El instrumento se relaciona con el método planteado en el proyecto.				70	

Lima, 30 de noviembre del 2022

Ing. Jorge Luis Lavalle Díaz
N° de DNI: 09945647

ANEXO 4: "BASE DE DATOS DE LA RECOLECCIÓN DE LA INFORMACIÓN"

PRETEST

VARIABLES	V2: Mejora de la calidad del software											
DIMENSIONES	Eficiencia			SUBTOTAL	Confiabilidad			SUBTOTAL	Integridad			SUBTOTAL	SUMA TOTAL
PREGUNTAS	P1	P2	P3		P4	P5	P6		P7	P8	P9		
E1	1	2	1	4	1	3	3	7	1	1	3	5	16
E2	1	3	3	7	1	1	1	3	1	1	1	3	13
E3	1	1	1	3	1	1	1	3	2	1	1	4	10
E4	1	3	3	7	1	2	3	6	1	1	1	3	16
E5	3	3	1	7	1	1	4	6	1	1	1	3	16
E6	1	1	4	6	1	3	3	7	1	1	1	3	16
E7	3	3	1	7	1	1	1	3	3	1	3	7	17
E8	3	1	3	7	1	3	3	7	1	1	1	3	17
E9	1	1	3	5	1	3	3	7	3	1	1	5	17
E10	3	3	3	9	1	1	3	5	3	1	3	7	21
E11	3	3	3	9	1	1	1	3	1	1	1	3	15
E12	3	3	2	8	1	2	1	4	3	1	1	5	17
E13	1	1	1	3	1	4	4	9	1	1	1	3	15
E14	1	3	4	8	1	3	4	8	1	1	1	3	19
E15	2	2	3	7	1	1	2	4	3	1	1	5	16
E16	3	2	3	8	1	1	1	3	1	1	1	3	14
E17	3	3	3	9	1	1	1	3	1	1	2	4	16
E18	1	1	3	5	1	3	1	5	1	1	3	5	15
E19	1	1	3	5	1	3	3	7	1	1	4	6	18
E20	3	1	3	7	1	1	1	3	2	1	3	6	16
E21	1	1	3	5	1	3	3	7	1	1	3	5	17
E22	1	1	1	3	1	1	1	3	1	1	1	3	9
E23	2	1	1	4	1	3	3	7	2	1	1	4	15
E24	1	1	1	3	1	2	3	6	1	1	1	3	12
E25	1	1	1	3	1	1	1	3	1	1	1	3	9
E26	1	1	1	3	3	3	1	7	1	1	1	3	13
E27	3	1	3	7	3	3	1	7	3	1	3	7	21
E28	1	1	1	3	1	1	1	3	1	1	1	3	9

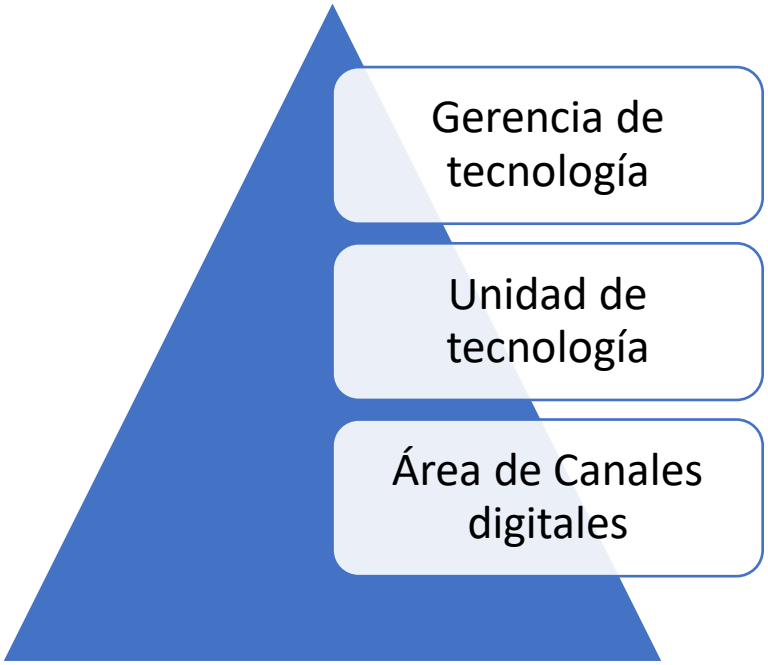
E29	3	1	1	5	1	1	4	6	3	1	1	5	16
E30	3	1	3	7	2	1	2	5	3	1	3	7	19
E31	1	1	1	3	1	1	3	5	1	1	1	3	11
E32	3	1	1	5	3	1	3	7	3	3	1	7	19
E33	1	1	1	3	1	1	1	3	1	1	1	3	9
E34	1	1	1	3	1	1	4	6	1	1	1	3	12
E35	3	1	1	5	1	1	3	5	3	1	1	5	15
E36	1	1	1	3	1	1	1	3	1	1	1	3	9
E37	1	3	2	6	1	1	3	5	1	3	2	6	17
E38	1	3	3	7	1	1	1	3	1	3	3	7	17
E39	1	3	4	8	1	1	3	5	1	3	4	8	21
E40	2	4	3	9	3	1	1	5	2	4	3	9	23
E41	1	1	3	5	1	1	1	3	1	2	1	4	12
E42	1	1	1	3	3	1	1	5	1	3	3	7	15
E43	2	1	1	4	3	1	3	7	1	1	1	3	14
E44	1	3	1	5	1	1	3	5	1	3	3	7	17
E45	1	1	1	3	3	1	3	7	3	3	1	7	17
E46	1	1	1	3	3	1	3	7	1	1	4	6	16
E47	3	1	3	7	3	1	2	6	3	1	1	5	18
E48	1	3	1	5	1	1	1	3	3	1	3	7	15
E49	3	3	1	7	1	1	4	6	1	1	3	5	18
E50	3	1	3	7	2	1	3	6	3	1	3	7	20
E51	1	1	1	3	3	1	3	7	3	1	3	7	17
E52	3	3	1	7	3	1	3	7	3	1	2	6	20
E53	1	1	1	3	1	1	3	5	1	1	1	3	11
E54	1	1	1	3	1	1	3	5	1	1	4	6	14
E55	3	1	1	5	3	1	3	7	2	1	3	6	18
E56	1	1	1	3	1	1	3	5	3	1	3	7	15
E57	1	3	2	6	1	1	1	3	3	1	3	7	16
E58	1	3	3	7	2	1	1	4	1	1	3	5	16
E59	1	3	4	8	1	1	1	3	1	1	3	5	16
E60	2	4	3	9	1	1	1	3	3	1	3	7	19
E61	1	2	1	4	1	1	1	3	1	1	3	5	12
E62	1	3	3	7	3	1	3	7	1	1	1	3	17
E63	1	1	1	3	1	1	1	3	2	1	1	4	10
E64	1	1	3	5	3	1	1	5	1	3	1	5	15
E65	3	1	1	5	3	1	3	7	1	1	1	3	15
E66	1	1	4	6	1	1	1	3	1	1	1	3	12
E67	3	1	1	5	3	3	1	7	3	1	3	7	19
E68	3	1	3	7	1	1	1	3	1	3	1	5	15
E69	1	1	3	5	1	1	1	3	3	3	1	7	15
E70	3	1	3	7	3	1	1	5	3	1	3	7	19
E71	3	1	3	7	1	1	1	3	1	1	1	3	13

POSTEST

VARIABLES	V2: Mejora de la calidad del software												SUMA TOTAL
DIMENSIONES	Eficiencia			SUBTOTAL	Confiabilidad			SUBTOTAL	Integridad			SUBTOTAL	
PREGUNTAS	P1	P2	P3		P4	P5	P6		P7	P8	P9		
E1	4	2	4	10	5	3	3	11	5	5	3	13	34
E2	4	3	3	10	5	5	5	15	5	5	5	15	40
E3	4	4	4	12	5	5	5	15	2	5	5	12	39
E4	4	3	3	10	5	2	3	10	5	5	5	15	35
E5	3	3	4	10	5	5	4	14	5	5	5	15	39
E6	4	4	4	12	5	3	3	11	5	5	5	15	38
E7	3	3	4	10	5	5	5	15	3	5	3	11	36
E8	3	4	3	10	5	3	3	11	5	5	5	15	36
E9	4	4	3	11	5	3	3	11	3	5	5	13	35
E10	3	3	3	9	5	5	3	13	3	5	3	11	33
E11	3	3	3	9	5	5	5	15	5	5	5	15	39
E12	3	3	2	8	5	2	5	12	3	5	5	13	33
E13	4	4	4	12	5	4	4	13	5	5	5	15	40
E14	4	3	4	11	5	3	4	12	5	5	5	15	38
E15	2	2	3	7	5	5	2	12	3	5	5	13	32
E16	3	2	3	8	5	5	5	15	5	5	5	15	38
E17	3	3	3	9	5	5	5	15	5	5	2	12	36
E18	4	4	3	11	5	3	5	13	5	5	3	13	37
E19	4	4	3	11	5	3	3	11	5	5	4	14	36
E20	3	4	3	10	5	5	5	15	2	5	3	10	35
E21	4	4	3	11	5	3	3	11	5	5	3	13	35
E22	4	4	4	12	5	5	5	15	5	5	5	15	42
E23	2	4	4	10	5	3	3	11	2	5	5	12	33
E24	4	4	4	12	5	2	3	10	5	5	5	15	37
E25	4	4	4	12	5	5	5	15	5	5	5	15	42
E26	4	4	4	12	3	3	5	11	5	5	5	15	38
E27	3	4	3	10	3	3	5	11	3	5	3	11	32
E28	4	4	4	12	5	5	5	15	5	5	5	15	42

E29	3	4	4	11	5	5	4	14	3	5	5	13	38
E30	3	4	3	10	2	5	2	9	3	5	3	11	30
E31	4	4	4	12	5	5	3	13	5	5	5	15	40
E32	3	4	4	11	3	5	3	11	3	3	5	11	33
E33	4	4	4	12	5	5	5	15	5	5	5	15	42
E34	4	4	4	12	5	5	4	14	5	5	5	15	41
E35	3	4	4	11	5	5	3	13	3	5	5	13	37
E36	4	4	4	12	5	5	5	15	5	5	5	15	42
E37	4	3	2	9	5	5	3	13	5	3	2	10	32
E38	4	3	3	10	5	5	5	15	5	3	3	11	36
E39	4	3	4	11	5	5	3	13	5	3	4	12	36
E40	2	4	3	9	3	5	5	13	2	4	3	9	31
E41	4	4	3	11	5	5	5	15	5	2	5	12	38
E42	4	4	4	12	3	5	5	13	5	3	3	11	36
E43	2	4	4	10	3	5	3	11	5	5	5	15	36
E44	4	3	4	11	5	5	3	13	5	3	3	11	35
E45	4	4	4	12	3	5	3	11	3	3	5	11	34
E46	4	4	4	12	3	5	3	11	5	5	4	14	37
E47	3	4	3	10	3	5	2	10	3	5	5	13	33
E48	4	3	4	11	5	5	5	15	3	5	3	11	37
E49	3	3	4	10	5	5	4	14	5	5	3	13	37
E50	3	4	3	10	2	5	3	10	3	5	3	11	31
E51	4	4	4	12	3	5	3	11	3	5	3	11	34
E52	3	3	4	10	3	5	3	11	3	5	2	10	31
E53	4	4	4	12	5	5	3	13	5	5	5	15	40
E54	4	4	4	12	5	5	3	13	5	5	4	14	39
E55	3	4	4	11	3	5	3	11	2	5	3	10	32
E56	4	4	4	12	5	5	3	13	3	5	3	11	36
E57	4	3	2	9	5	5	5	15	3	5	3	11	35
E58	4	3	3	10	2	5	5	12	5	5	3	13	35
E59	4	3	4	11	5	5	5	15	5	5	3	13	39
E60	2	4	3	9	5	5	5	15	3	5	3	11	35
E61	4	2	4	10	5	5	5	15	5	5	3	13	38
E62	4	3	3	10	3	5	3	11	5	5	5	15	36
E63	4	4	4	12	5	5	5	15	2	5	5	12	39
E64	4	4	3	11	3	5	5	13	5	3	5	13	37
E65	3	4	4	11	3	5	3	11	5	5	5	15	37
E66	4	4	4	12	5	5	5	15	5	5	5	15	42
E67	3	4	4	11	3	3	5	11	3	5	3	11	33
E68	3	4	3	10	5	5	5	15	5	3	5	13	38
E69	4	4	3	11	5	5	5	15	3	3	5	11	37
E70	3	4	3	10	3	5	5	13	3	5	3	11	34
E71	3	4	3	10	5	5	5	15	5	5	5	15	40

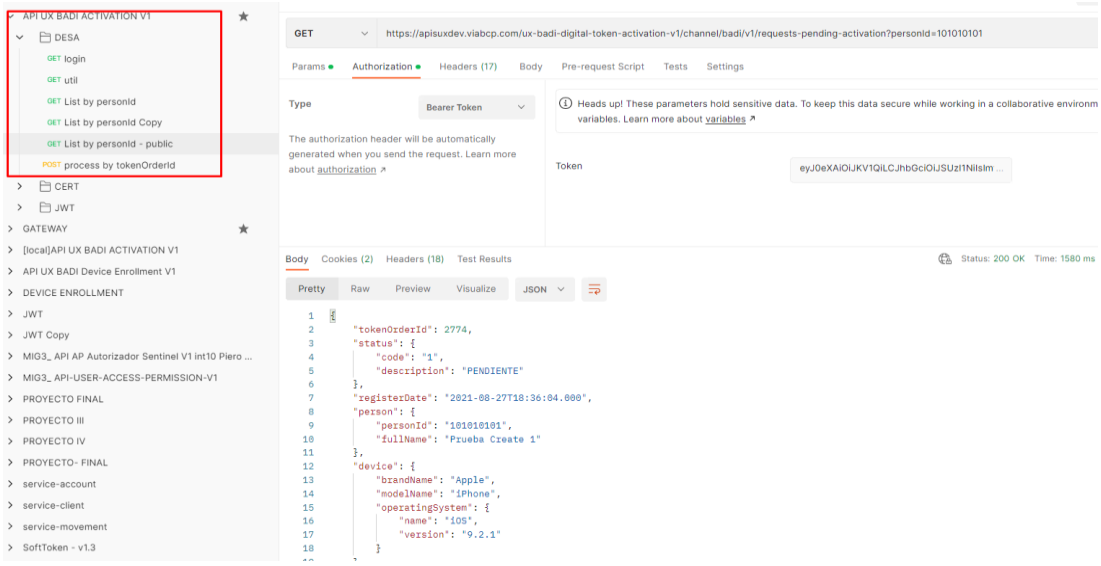
ANEXO 5: ORGANIGRAMA DE LA EMPRESA



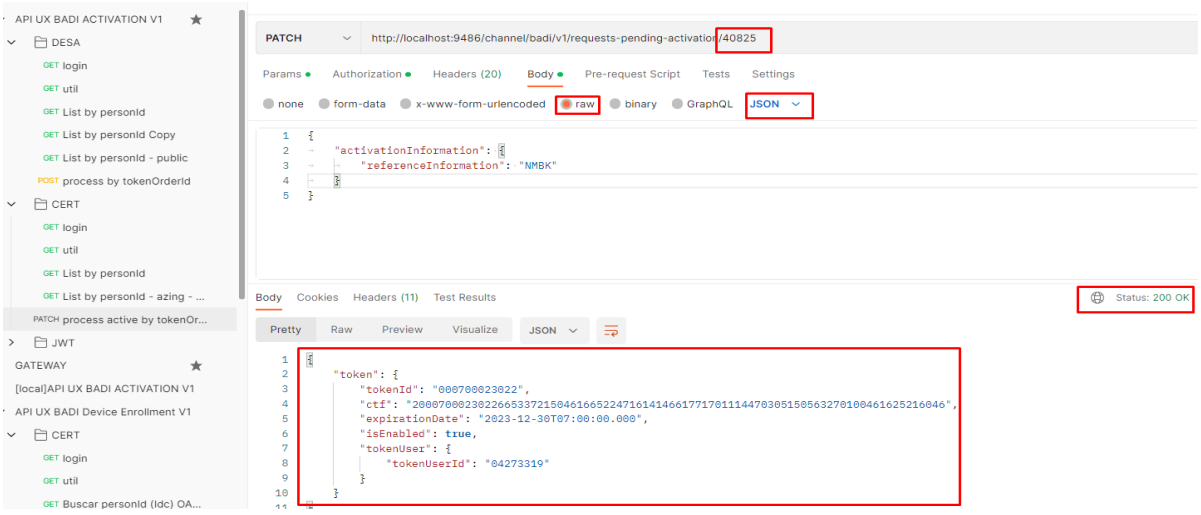
ANEXO 6: Pruebas de Solución

Con la ayuda de la herramienta postman se realiza la demostración de los microservicios implementados que son Activación de Token Digital y Desvinculación de dispositivos.

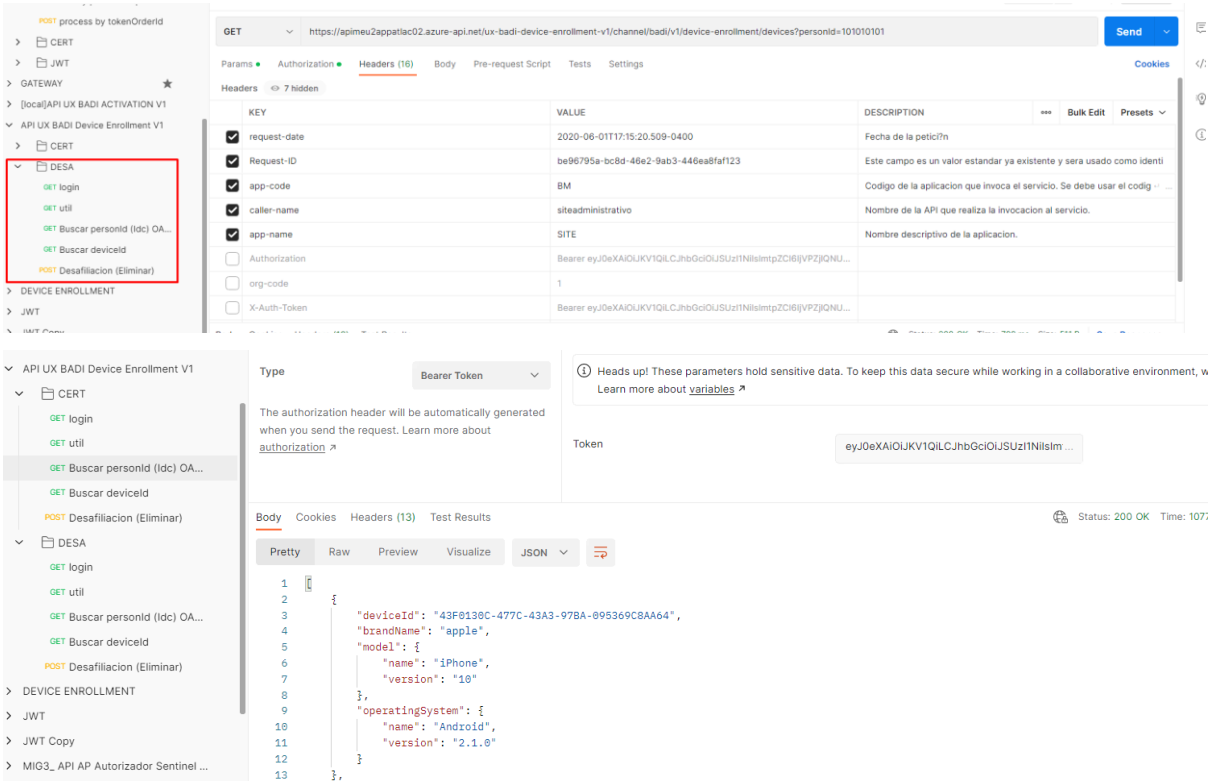
Lista de solicitudes pendientes por activar



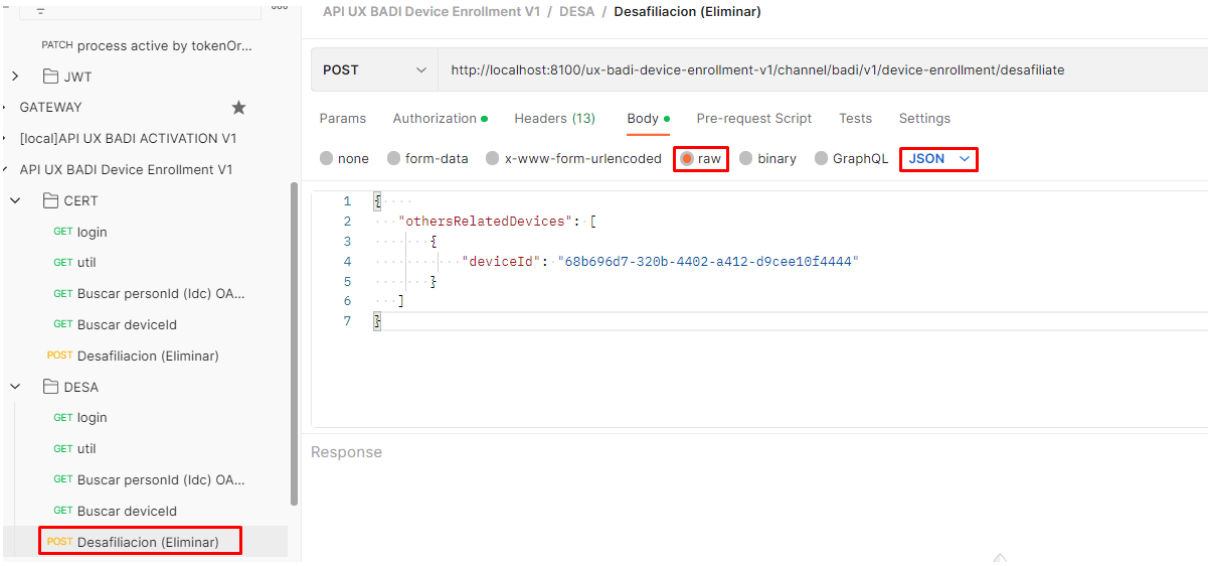
Proceso de Activación Token Digital



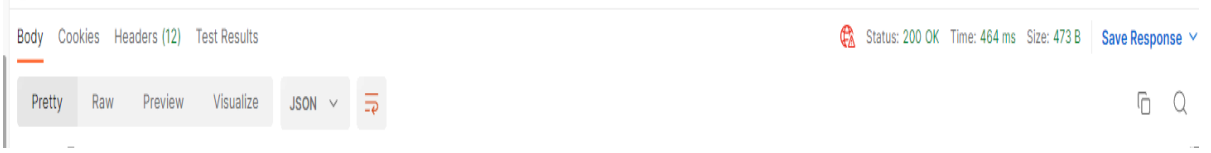
Lista de dispositivos para desafiliar



Proceso para la desafiliar un dispositivo.



Cuando Todo está correcto retorna un cód. de 200



Cuando se envía un código Incorrecto