



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Sistemas

**Migración de la capa de persistencia de microservicios
de Apache Hbase a MongoDB utilizando Spring Boot
en una arquitectura basada en microservicios en Azure**

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Sistemas

AUTOR

Jean Franco MARTINEZ RONCAL

ASESOR

Jorge Luis CHÁVEZ SOTO

Lima, Perú

2026



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

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

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

Referencia bibliográfica

Martinez, J. (2026). *Migración de la capa de persistencia de microservicios de Apache Hbase a MongoDB utilizando Spring Boot en una arquitectura basada en microservicios en Azure*. [Trabajo de suficiencia profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Sistemas e Informática, Escuela Profesional de Ingeniería de Sistemas]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Jean Franco Martinez Roncal
Tipo de documento de identidad	DNI
Número de documento de identidad	70083962
URL de ORCID	https://orcid.org/0009-0001-4434-5647
Datos de asesor	
Nombres y apellidos	Jorge Luis Chávez Soto
Tipo de documento de identidad	DNI
Número de documento de identidad	08675814
URL de ORCID	https://orcid.org/0000-0002-9408-9266
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Javier Elmer Cabrera Díaz
Tipo de documento	DNI
Número de documento de identidad	08692591
Miembro del jurado 1	
Nombres y apellidos	Norberto Ulises Roman Concha
Tipo de documento	DNI
Número de documento de identidad	08510560
Datos de investigación	
Línea de investigación	No aplica
Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento
Ubicación geográfica de la investigación	País: Perú Departamento: Lima Provincia: Lima

	Distrito: Cercado de Lima Jr. Carlos Amezaga No. 375 Universidad Nacional Mayor de San Marcos Latitud: -12.0564232 Longitud: -77.0843327
Año o rango de años en que se realizó la investigación	2025
URL de disciplinas OCDE	2.02.04 -- Ingeniería de sistemas y comunicaciones https://purl.org/pe-repo/ocde/ford#2.02.04



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

Acta Virtual de Sustentación
del Trabajo de Suficiencia Profesional

Siendo las 14:00 horas del día 31 de marzo del año 2026, se reunieron virtualmente los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Mg. Cabrera Díaz Javier Elmer (Presidente), Mg. Román Concha Norberto Ulises (Miembro) y el Lic. Chávez Soto Jorge Luis (Miembro Asesor), usando la plataforma Meet (<http://meet.google.com/hot-tmmg-afm>), para la sustentación virtual del Trabajo de Suficiencia Profesional intitulado: **“MIGRACIÓN DE LA CAPA DE PERSISTENCIA DE MICROSERVICIOS DE APACHE HBASE A MONGODB UTILIZANDO SPRING BOOT EN UNA ARQUITECTURA BASADA EN MICROSERVICIOS EN AZURE”**, por el Bachiller **Martínez Roncal Jean Franco**; para obtener el Título Profesional de Ingeniero de Sistemas.

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

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

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

A continuación el Presidente de Jurados el Mg. Cabrera Díaz Javier Elmer, declara al Bachiller **Ingeniero de Sistemas**.

Siendo las 14:53 horas, se levantó la sesión.

Presidente

Mg. Cabrera Díaz Javier Elmer

Miembro

Mg. Román Concha Norberto Ulises

Miembro Asesor

Lic. Chávez Soto Jorge Luis



UNIVERSIDAD NACIONAL MAYOR DE
SAN MARCOS
Universidad del Perú, Decana de América

CERTIFICADO DE SIMILITUD

Yo, Jorge Luis Chávez Soto en mi condición de asesor acreditado mediante Resolución Decanal N° 001268-2025-D-FISI/UNMSM y OFICIO N° 0011-PTTSP-FISI-2025 del Trabajo de Suficiencia Profesional, cuyo título es Migración de la capa de persistencia de microservicios de Apache Hbase a MongoDB utilizando Spring Boot en una arquitectura basada en microservicios en Azure, presentado por el bachiller Martinez Roncal Jean Franco para optar el Título Profesional de Ingeniero de Sistemas, CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante la plataforma PIDS, el documento evaluado cuenta con el 7% de similitud, nivel PERMITIDO para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional**.

Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del **TITULO PROFESIONAL DE INGENIERO DE SISTEMAS**.

Lima, 04 de junio de 2026



Firmado digitalmente por CHAVEZ
SOTO Jorge Luis FAU 20148092282
soft
Motivo: Soy el autor del documento
Fecha: 04.06.2026 18:09:33 -05:00

Firma del Asesor:

Nombres y Apellidos del asesor: Jorge Luis Chávez Soto

DNI: 08675814

FICHA CATALOGRÁFICA

**MIGRACIÓN DE LA CAPA DE PERSISTENCIA DE MICROSERVICIOS
DE APACHE HBASE A MONGODB UTILIZANDO SPRING BOOT EN
UNA ARQUITECTURA BASADA EN MICROSERVICIOS EN AZURE**

AUTOR: MARTINEZ RONCAL, JEAN FRANCO

ASESOR: CHÁVEZ SOTO, JORGE LUIS

LIMA – PERÚ, 2025

TÍTULO PROFESIONAL: Título Profesional de Ingeniero de Sistemas

**ÁREA / PROGRAMA / LÍNEA DE INVESTIGACIÓN: Ingenierías /
Tecnología de Información y Comunicación / Ingeniería de Software**

**PREGRADO: Universidad Nacional Mayor de San Marcos - Facultad de
Ingeniería de Sistemas e Informática - Escuela Profesional de Ingeniería de
Sistemas**

Formato 21 x 29.7 cm

Páginas: xi, 59

Dedicatoria

Dedico el presente trabajo a mis padres, quienes me enseñaron que la verdadera riqueza está en el esfuerzo, honestidad y el amor. Gracias por su apoyo todo este tiempo, me dieron lo más valioso: su apoyo incondicional y la fuerza para no rendirme. Hoy quiero compartir este logro con ustedes porque cada paso que di estuvo sostenido de sacrificios y su confianza en mí.

A mi hermana, que me vio soñar y luchar por ese momento desde que éramos niños. Hoy, al titularme, quiero que sepas que cada logro que consiga también es una invitación a cumplir tus propios sueños.

A mi pequeño hermano, que llegó a mi vida a mitad de este camino tan grande, este título también es para ti, porque quiero que cuando crezcas veas que todo con esfuerzo y perseverancia se puede lograr.

Agradecimientos

Agradezco a mi mamá, por su amor incondicional, su paciencia infinita y por estar en los momentos más difíciles. Gracias por creer en mí cuando las fuerzas parecían faltar, por tus palabras de aliento y por enseñarme que la perseverancia nace del corazón.

Agradezco a mi papá, por sus enseñanzas de vida, su ejemplo de esfuerzo y responsabilidad, y por inculcarme el valor del trabajo honesto y los valores bien puestos. Gracias por guiarme con firmeza y por mostrarme, con hechos, que los logros verdaderos se construyen con disciplina y compromiso.

Agradezco a mi hermana, quien creció viéndome soñar, luchar y no rendirme. Gracias por estar siempre, por tu compañía silenciosa y por creer en mí incluso en los momentos más difíciles. Espero que este logro sea para ti una señal que los sueños sí se cumplen cuando existe la constancia.

Agradezco a mi hermano, que llegó a mi vida en medio de este camino y transformó mi manera de ver el futuro. Tu presencia me enseñó a valorar lo esencial, a tener esperanza y a recordar cada día por qué vale la pena esforzarse. Este logro también es para ti.

Agradezco a mi novia, gracias por tu apoyo, tu paciencia y tu confianza en mí. En este último tramo me fuiste impulso, compañía y fortaleza. Gracias por animarme a dar este paso y por caminar a mi lado cuando el cansancio y la duda aparecieron.

A la familia Martínez y a la familia Roncal, por acompañarme a lo largo de este camino con su cariño, apoyo y palabras de aliento. Gracias por estar presentes, por creer en mí y por compartir conmigo cada paso de este proceso.

A mis amigos de la universidad, quienes compartieron conmigo este camino académico, las largas jornadas de estudio, el cansancio, las dudas y también las alegrías. Gracias por el compañerismo, el apoyo mutuo y por hacer de este proceso una experiencia más llevadera y significativa. Cada conversación, cada desvelo y cada momento compartido forman parte de este logro.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Migración de la capa de persistencia de microservicios de Apache Hbase a MongoDB utilizando Spring Boot en una arquitectura basada en microservicios en Azure

Autor: Martinez Roncal, Jean Franco

Asesor: Chávez Soto, Jorge Luis

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

Fecha: Septiembre del 2025

RESUMEN

Este proyecto aborda la migración de la capa de persistencia de un conjunto de microservicios de Apache HBase a MongoDB, dentro del marco de una arquitectura basada en microservicios desarrollada con Spring Boot e implementada en la nube de Azure. La investigación fue de tipo aplicada y descriptiva, desarrollada bajo la modalidad de experiencia profesional en un entorno empresarial real. Para la ejecución del proyecto se utilizó la metodología ágil Scrum, organizando el trabajo en sprints, reuniones de seguimiento y entregables incrementales. La motivación principal proviene de las limitaciones detectadas en HBase en cuanto al manejo de datos no estructurados, la flexibilidad de las consultas y la integración con aplicaciones modernas. La propuesta consistió en analizar los requisitos de la organización, diseñar un modelo de persistencia adecuado en MongoDB y adaptar los microservicios para garantizar la compatibilidad, la eficiencia y la escalabilidad sin afectar la lógica existente. Como resultado, se minimizaron los tiempos de desarrollo a futuro y se creó un conjunto de buenas prácticas para la migración de la capa de persistencia, reduciendo la complejidad del acceso a datos y la curva de aprendizaje de la tecnología. Este trabajo demuestra la viabilidad técnica de la migración a bases de datos NoSQL en entornos corporativos y destaca la importancia de adoptar arquitecturas modernas para responder a las exigencias de escalabilidad y transformación digital.

Palabras Clave: Migración, Apache HBase, MongoDB, Scrum, optimización, microservicios.

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

**Migrating the microservices persistence layer from Apache Hbase to MongoDB
using Spring Boot in a microservices-based architecture on Azure**

Author: Martínez Roncal, Jean Franco

Advisor: Chávez Soto, Jorge Luis

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

Date: September 2025

ABSTRACT

This project addresses the migration of the persistence layer of a set of microservices from Apache HBase to MongoDB, within the framework of a microservices-based architecture developed with Spring Boot and implemented in the Azure cloud. The research was applied and descriptive in nature, developed as part of professional experience in a real business environment. The agile Scrum methodology was used to execute the project, organizing the work into sprints, follow-up meetings, and incremental deliverables. The main motivation came from the limitations detected in HBase in terms of handling unstructured data, query flexibility, and integration with modern applications. The proposal consisted of analyzing the organization's requirements, designing an appropriate persistence model in MongoDB, and adapting the microservices to ensure compatibility, efficiency, and scalability without affecting the existing logic. As a result, future development times were minimized and a set of best practices for migrating the persistence layer was created, reducing the complexity of data access and the technology learning curve. This work demonstrates the technical feasibility of migrating to NoSQL databases in corporate environments and highlights the importance of adopting modern architectures to meet the demands of scalability and digital transformation.

Key words: Migration, Apache HBase, MongoDB, Scrum, optimization, microservices.

ÍNDICE GENERAL

ÍNDICE GENERAL	vii
ÍNDICE DE TABLAS	ix
ÍNDICE DE FIGURAS.....	x
ÍNDICE DE ECUACIONES	xi
INTRODUCCIÓN	1
CAPÍTULO I - TRAYECTORIA PROFESIONAL	3
CAPÍTULO II - CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA..	9
2.1. EMPRESA - ACTIVIDAD QUE REALIZA	9
2.2. MISIÓN.....	9
2.3. VISIÓN	9
2.4. ORGANIZACIÓN DE LA EMPRESA	10
2.5. ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS	12
2.6. EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN ...	13
CAPÍTULO III - ACTIVIDADES DESARROLLADAS	15
3.1. SITUACIÓN PROBLEMÁTICA	15
3.1.1. DEFINICIÓN DEL PROBLEMA	16
3.2. SOLUCIÓN.....	16
3.2.1. OBJETIVOS.....	17
3.2.2. ALCANCE	17
3.2.3. ETAPAS Y METODOLOGÍA.....	18
3.2.4. FUNDAMENTOS UTILIZADOS	20
3.2.5. IMPLEMENTACIÓN DE LAS ÁREAS, PROCESOS, SISTEMAS Y BUENAS PRÁCTICAS	31
3.3. EVALUACIÓN.....	43
3.3.1. EVALUACIÓN COSTO - BENEFICIO	43
3.3.2. INTERPRETACIÓN DEL VAN Y DEL TIR.....	45

3.3.3. BENEFICIOS DEL PROYECTO	48
CAPÍTULO IV - REFLEXIÓN CRÍTICA DE LA EXPERIENCIA.....	51
CAPÍTULO V - CONCLUSIONES Y RECOMENDACIONES	53
CONCLUSIONES	53
RECOMENDACIONES	55
FUENTES DE INFORMACIÓN	56
GLOSARIO	58

ÍNDICE DE TABLAS

Tabla 1 <i>Trayectoria profesional</i>	3
Tabla 2 <i>Formación Académica</i>	6
Tabla 3 <i>Cursos Realizados</i>	6
Tabla 4 <i>Certificaciones</i>	7
Tabla 5 <i>Idiomas</i>	8
Tabla 6 <i>Roles del Proyecto</i>	18
Tabla 7 <i>Puntos de las historias</i>	19
Tabla 8 <i>Sprints del Proyecto</i>	31
Tabla 9 <i>Detalle de la Historia de Usuario del ms-campanascliente-neg</i>	32
Tabla 10 <i>Detalle de la Historia de usuario del ms-cartolamt940-neg</i>	32
Tabla 11 <i>Detalle de la Historia de usuario del ms-cartolasempresas-neg</i>	33
Tabla 12 <i>Detalle de la Historia de usuario del ms-productoscliente-neg</i>	33
Tabla 13 <i>Costos del equipo</i>	44
Tabla 14 <i>Flujo de caja anual</i>	46
Tabla 15 <i>Costos anuales por cada motor de base de datos</i>	46
Tabla 16 <i>Flujo de caja anual actualizado para el cálculo de la VAN</i>	47

ÍNDICE DE FIGURAS

Ilustración 1 <i>Organigrama</i>	10
Ilustración 2 <i>Organigrama enfocado en Software Factory</i>	11
Ilustración 3 <i>Modelo de datos orientado a documentos en MongoDB</i>	21
Ilustración 4 <i>Administración de la información dentro de una tabla en HBase</i>	23
Ilustración 5 <i>Arquitectura de Spring Framework</i>	25
Ilustración 6 <i>Arquitectura de microservicios</i>	27
Ilustración 7 <i>Arquitectura de microservicios con aplicaciones de Azure</i>	29
Ilustración 8 <i>Dependencias de Apache Hadoop, Zookeeper y HBase</i>	34
Ilustración 9 <i>Dependencia de Spring Boot MongoDB</i>	35
Ilustración 10 <i>Cadena de Conexión a una sola base de datos</i>	35
Ilustración 11 <i>Método obtener campana antes de la migración</i>	36
Ilustración 12 <i>Método obtener campana después de la migración</i>	36
Ilustración 13 <i>Response de obtener campana</i>	37
Ilustración 14 <i>Cadena de Conexión a dos bases de datos</i>	37
Ilustración 15 <i>Clase de configuración para la primera conexión a la base de datos</i> ..	38
Ilustración 16 <i>Clase de configuración para la segunda conexión a la base de datos</i> ..	38
Ilustración 17 <i>Buscar datos del cliente de cartola antes de la migración</i>	39
Ilustración 18 <i>Buscar datos del cliente de cartola después de la migración</i>	40
Ilustración 19 <i>Response de obtener cartolamt</i>	40
Ilustración 20 <i>Response de obtener cartola</i>	41
Ilustración 21 <i>Response de obtener producto</i>	41

ÍNDICE DE ECUACIONES

Ecuación 1 <i>Cálculo del VAN</i>	47
Ecuación 2 <i>Cálculo de la TIR</i>	47

INTRODUCCIÓN

En los últimos años, la arquitectura de microservicios se ha convertido en una de las tendencias más importantes en el desarrollo de software. Este enfoque permite un mantenimiento más escalable, flexible y sencillo de los sistemas, ya que cada servicio puede desarrollarse, implementarse y actualizarse de forma independiente. Esto permite a las empresas responder con mayor rapidez a las demandas del mercado y adaptarse a los cambios tecnológicos.

Frameworks como Spring Boot han contribuido al auge de esta arquitectura y proporcionan herramientas que simplifican el desarrollo y la implementación de microservicios en entornos empresariales. Sin embargo, la eficiencia de un sistema depende no solo de su diseño arquitectónico, sino también del manejo de sus datos. Las bases de datos desempeñan un papel fundamental para garantizar el rendimiento, la fiabilidad y la escalabilidad de las aplicaciones dentro de una organización.

Bases de datos como Apache HBase se han utilizado ampliamente durante mucho tiempo en escenarios que requieren procesamiento distribuido y grandes requisitos de almacenamiento. Sin embargo, con la creciente prevalencia de aplicaciones que procesan datos no estructurados, han surgido limitaciones en cuanto a la flexibilidad de las consultas, la facilidad de integración y la adaptabilidad a nuevas funcionalidades junto con la curva de aprendizaje que requiere.

En este contexto, las bases de datos NoSQL, especialmente MongoDB, se han consolidado como una alternativa sólida para los entornos modernos. Su modelo de documento permite una gestión de información más dinámica, mejora la experiencia de desarrollo y reduce la complejidad del mantenimiento del sistema. Esto facilita la integración de nuevas funcionalidades sin afectar la lógica existente, algo esencial en las arquitecturas de microservicios que es muy actual en estos últimos tiempos.

En este contexto, este trabajo de suficiencia profesional aborda la migración de la capa de persistencia de microservicios de Apache HBase a MongoDB dentro de una arquitectura basada en microservicios desarrollada con Spring Boot e implementada en la nube de Azure. El presente trabajo busca demostrar cómo esta transición optimiza los tiempos de respuesta para operaciones críticas, mejora la mantenibilidad del código y facilita la integración de nuevas funcionalidades.

Asimismo, la adopción de tecnologías modernas orientadas a microservicios permite facilitar procesos de migración rápida y controlada en entornos empresariales altamente dinámicos. El desacoplamiento entre servicios y la separación clara de la capa de persistencia hacen posible realizar transiciones tecnológicas de manera progresiva, reduciendo riesgos operativos y evitando interrupciones significativas en la operación del sistema. En este sentido, el uso de frameworks como Spring Boot, junto con bases de datos flexibles como MongoDB, permite implementar estrategias de migración incremental, pruebas paralelas y despliegues continuos, asegurando la compatibilidad con los servicios existentes y acelerando la adopción de nuevas soluciones sin comprometer la estabilidad del sistema.

El objetivo es proporcionar experiencia práctica que destaque la importancia de seleccionar tecnologías adecuadas para la gestión de datos en entornos distribuidos y enfatice la importancia de migrar a soluciones más flexibles que satisfagan las necesidades actuales de desarrollo de software.

A continuación, se realiza el despliegue de la estructura del presente trabajo:

- En el CAPÍTULO I, se expone una visión global de mi trayectoria profesional, resaltando los aprendizajes adquiridos a lo largo de las diversas experiencias laborales. Se destacan las competencias desarrolladas y los proyectos que han contribuido de manera significativa a mi crecimiento laboral y profesional.
- En el CAPÍTULO II, se aborda la descripción de la empresa donde se llevó a cabo el proyecto, detallando su misión y visión, así como las funciones y responsabilidades asumidas dentro del proyecto. Este apartado comprende la relevancia del proyecto en la organización.
- En el CAPÍTULO III, se analiza la problemática que dio origen al proyecto, identificando las principales dificultades y planteando la solución a las mismas. Se presentan los objetivos, el alcance y la implementación.
- En el CAPÍTULO IV, se presenta una reflexión crítica sobre la experiencia adquirida durante el desarrollo del proyecto. Se presentan desafíos y lecciones aprendidas, destacando el impacto de mi experiencia profesional en la organización.
- En el CAPÍTULO V, se describen las conclusiones y recomendaciones del trabajo en base a los resultados obtenidos a lo largo del desarrollo del proyecto.

CAPÍTULO I - TRAYECTORIA PROFESIONAL

El autor tiene el grado de Bachiller en Ingeniería de Sistemas y cuenta con más de 7 años de experiencia laboral en el diseño y desarrollo de soluciones principalmente basadas en Java bajo el framework Spring y el uso de la aplicación principal Spring Boot, y con el uso de derivados tales como Spring Cloud, Spring Batch, Spring Data, Spring Webflux, etc; haciendo también el uso de otros lenguajes de programación con sus respectivos frameworks de trabajo tales como Python, usando Django y Flash; TypeScript, usando Angular y NestJS; y, C#, usando .NET.

Actualmente se encuentra laborando bajo el rol “Senior Backend Developer Java” en una consultora transnacional encargada de brindar servicios de alta calidad en desarrollo de software con presencia en las regiones de Norteamérica, Latinoamérica y Asia encargándose del proceso End to End con los clientes de cada sede.

Anteriormente el autor trabajó para empresas del sector de Tecnología de Información mayormente asociadas al lenguaje principal de programación. Los detalles se describen en la Tabla 1:

Tabla 1

Trayectoria profesional

Empresa/Periodo		Educación Académica
Entidad	consultora	Cargo: Senior Backend Developer Java
de software		Actividades:
(2023 - actualidad)		• Diseñar y desarrollar soluciones tecnológicas en Java, orientadas a arquitecturas de microservicios. • Ejecutar la migración de microservicios entre frameworks y versiones tecnológicas, asegurando compatibilidad y estabilidad en versiones productivas. • Implementar buenas prácticas de desarrollo como clean code, principios SOLID y control de versiones. • Implementar y optimizar la integración con motores de bases de datos SQL y NoSQL.

-
- Participar en el diseño de arquitectura de soluciones Backend, proponiendo mejoras en rendimiento y escalabilidad.

**Entidad bancaria
(2023 - 2024)**

Cargo: Backend Software Engineer Java

Actividades:

- Diseñar y desarrollar soluciones backend en Java para un producto conocido de préstamo bancario, orientado a la estabilidad, seguridad y alta disponibilidad.
- Gestionar y ejecutar despliegues en ambientes productivos, asegurando ventanas de despliegue controladas y mínima afectación al negocio.
- Coordinar activamente con el equipo de QA, Ciberseguridad y líderes técnicos para la validación, aprobación y verificación de vistos buenos para liberar versiones a producción.
- Participar en la gestión de incidencias productivas, realizando análisis de causa raíz y aplicando correcciones oportunas.

**Entidad consultora de software
(2022 - 2023)**

Cargo: Tech Lead

Actividades:

- Diseñar y desarrollar soluciones backend en Java para un cliente del sector bancario, alineados a requerimientos de negocio y estándares de seguridad.
- Liderar y supervisar un equipo de desarrollo encargado de la gestión de tarjetas, utilizando una arquitectura de microservicios desplegada en Azure.
- Coordinar con los equipos de QA, Ciberseguridad y gestión de cambios para liberar versiones a producción.
- Planificar y conducir sesiones de entrega, asegurando la correcta transferencia de conocimiento y alineación con el cliente.

- Brindar retroalimentación técnica continua al equipo, apoyando su crecimiento y asegurando la calidad del desarrollo.

Entidad consultora Cargo: Software Engineer

de software

(2021 - 2023)

Actividades:

- Diseñar y desarrollar soluciones backend en Java para un cliente del sector bancario, siguiendo lineamientos de seguridad y de negocio.
- Analizar requerimientos funcionales y técnicos del sistema, proponiendo soluciones alineadas a la arquitectura existente.
- Implementar mejoras y correcciones sobre servicios existentes, asegurando estabilidad y calidad del código.
- Atender la demanda de incidencias, realizando análisis, corrección de errores y validación de soluciones.

Entidad consultora Cargo: Analista Programador Java

de software

(2021 - 2021)

Actividades:

- Participar en el desarrollo full stack de soluciones tecnológicas, abarcando capas backend y frontend dentro del ciclo de vida completo del software.
- Implementar funcionalidades en Java, integradas con componentes desarrollados en Angular y con PostgreSQL.
- Analizar requerimientos funcionales y técnicos, proponiendo soluciones y realizando ajustes sobre sistemas existentes.
- Atender y resolver incidencias en los distintos ambientes (desarrollo, QA, producción).

Entidad consultora Cargo: Analista Programador Java

de software

(2020 - 2021)

Actividades:

- Desarrollar y modificar servicios REST en Java para un cliente del sector bancario.

- Diseñar y ejecutar pruebas unitarias usando JUnit y Mockito, asegurando la calidad del código desarrollado.
- Atender correcciones y requerimientos asignados, bajo supervisión de perfiles senior y líderes técnicos.

UNMSM
(2018 - 2020)

Cargo: Desarrollador Backend

Actividades:

- Desarrollar servicios REST y endpoints backend para aplicaciones institucionales, siguiendo lineamientos técnicos establecidos.
- Identificar y solucionar errores en ejecución de aplicaciones backend, realizando ajustes y correcciones.
- Dar mantenimiento a aplicaciones existentes, asegurando su correcto funcionamiento.
- Participar en proceso de CI/CD utilizando contenedores de aplicaciones, bajo supervisión del equipo técnico.

Nota. Fuente: Elaboración propia.

En la tabla 2 se muestra la formación académica:

Tabla 2

Formación Académica

Periodo	Educación Académica
2015 - 2020	Bachiller de la Escuela Profesional de Ingeniería de Sistemas Universidad Nacional Mayor de San Marcos

Nota. Fuente: Elaboración propia.

Los cursos realizados por el autor se encuentran en la tabla 3:

Tabla 3

Cursos Realizados

Periodo	Curso	Institución
2025	Spring AI Developer	Mitocode Network
2025	IA para Developers	Mitocode Network

2025	.NET 9 Full Stack	Mitocode Network
2025	AWS Cloud Practitioner	Mitocode Network
2025	Azure Fundamentals	Mitocode Network
2023	Spring Webflux	Mitocode Network
2023	Microservicios en Spring Cloud	Mitocode Network
2022	Docker Kubernetes Cloud	Mitocode Network
2020	Java Full Stack Developer	Mitocode Network
2020	Java Web Developer	Mitocode Network
2018	Java Enterprise 8 Developer	Mitocode Network
2018	Spring Developer	Mitocode Network
2017	Java Standard Developer	Mitocode Network

Nota. Fuente: Elaboración propia.

Las certificaciones obtenidas por el autor se listan en la Tabla 4:

Tabla 4

Certificaciones

Periodo	Certificado	Institución
2025	Microsoft Azure Fundamentals Certification (AZ-900)	Microsoft Azure
2025	AWS Cloud Practitioner	Amazon Web Services
2025	Java SE 17 (1Z0-829)	Oracle University

Nota. Fuente: Elaboración propia.

Los estudios de idiomas realizados por el autor se listan en la Tabla 5:

Tabla 5

Idiomas

Periodo	Idioma	Institución
2025	Inglés B1	Euroidiomas

Nota. Fuente: Elaboración propia.

CAPÍTULO II - CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA

2.1. EMPRESA - ACTIVIDAD QUE REALIZA

La empresa en la que se llevó a cabo el proyecto en mención es una consultora de software fundada en el año 2000, que brinda soluciones de software a gran escala para distintas empresas distribuidas dentro de Norteamérica, Sudamérica y Asia. Posee tres principales actividades las cuales son: Tecnología y desarrollos; Transformación Digital y Estrategia; y Soporte y Metodologías.

La labor principal se centra en la creación de software personalizado y en la implementación de arquitecturas fundamentadas en microservicios. Su unidad estratégica de negocios más importante es la de servicios de ingeniería de software para clientes corporativos, donde se diseñan, desarrollan y mantienen aplicaciones críticas que soportan la operación de sus clientes.

2.2. MISIÓN

La misión de la consultora de software orienta sus esfuerzos a apoyar a las organizaciones en la modernización de sus procesos mediante el uso de soluciones tecnológicas, haciendo foco en el desarrollo de software y servicios tecnológicos que aporten un valor significativo al negocio y de esta manera aportar al crecimiento sostenible de sus clientes y colaboradores.

2.3. VISIÓN

La visión de la consultora de software proyecta su crecimiento hacia el posicionamiento global como una consultora tecnológica referente, reconocida por su capacidad de innovación, calidad en la entrega de soluciones y compromiso con la generación de impacto tanto en el ámbito empresarial como social.

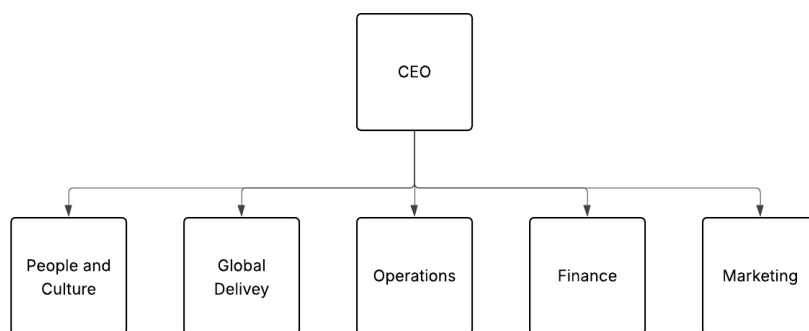
2.4. ORGANIZACIÓN DE LA EMPRESA

En la Ilustración 1, se presenta un organigrama jerárquico que ilustra las principales áreas que conforman la empresa. A continuación, se describirá brevemente las funciones y objetivos de cada una de estas áreas:

- **CEO:** Es responsable de dirigir la estrategia global, tomar decisiones ejecutivas y representa a la empresa frente a clientes.
- **People and Culture:** Área encargada de gestionar el talento humano y fomentar cultura organizacional dentro de la empresa, promoviendo un entorno de bienestar y desarrollo dentro de la organización.
- **Global Delivery:** Área encargada de gestionar y asegurar la correcta ejecución de equipos de desarrollo de los proyectos tecnológicos a nivel internacional. Se encarga de gestionar tiempos, costos, calidad de entregas que cumplan con los estándares de la organización.
- **Operations:** Área que se enfoca en la gestión operativa de la empresa, administra proyectos, gestión del talento asignado a clientes, seguimiento de desempeño operativo y mejora continua de los procesos internos.
- **Finance:** Área responsable de la gestión financiera y económica de la organización. Planifican el presupuesto, controlan gastos, se encargan de la facturación a clientes y cumplimiento de leyes contables y tributarias.
- **Marketing:** Se encargan de posicionar la empresa en el mercado y fortalecer su imagen corporativa a nivel internacional. Se encargan de la gestión de la marca y comunicación institucional. Su función principal es atraer nuevos clientes y reforzar la relación con los clientes actuales.

Ilustración 1

Organigrama

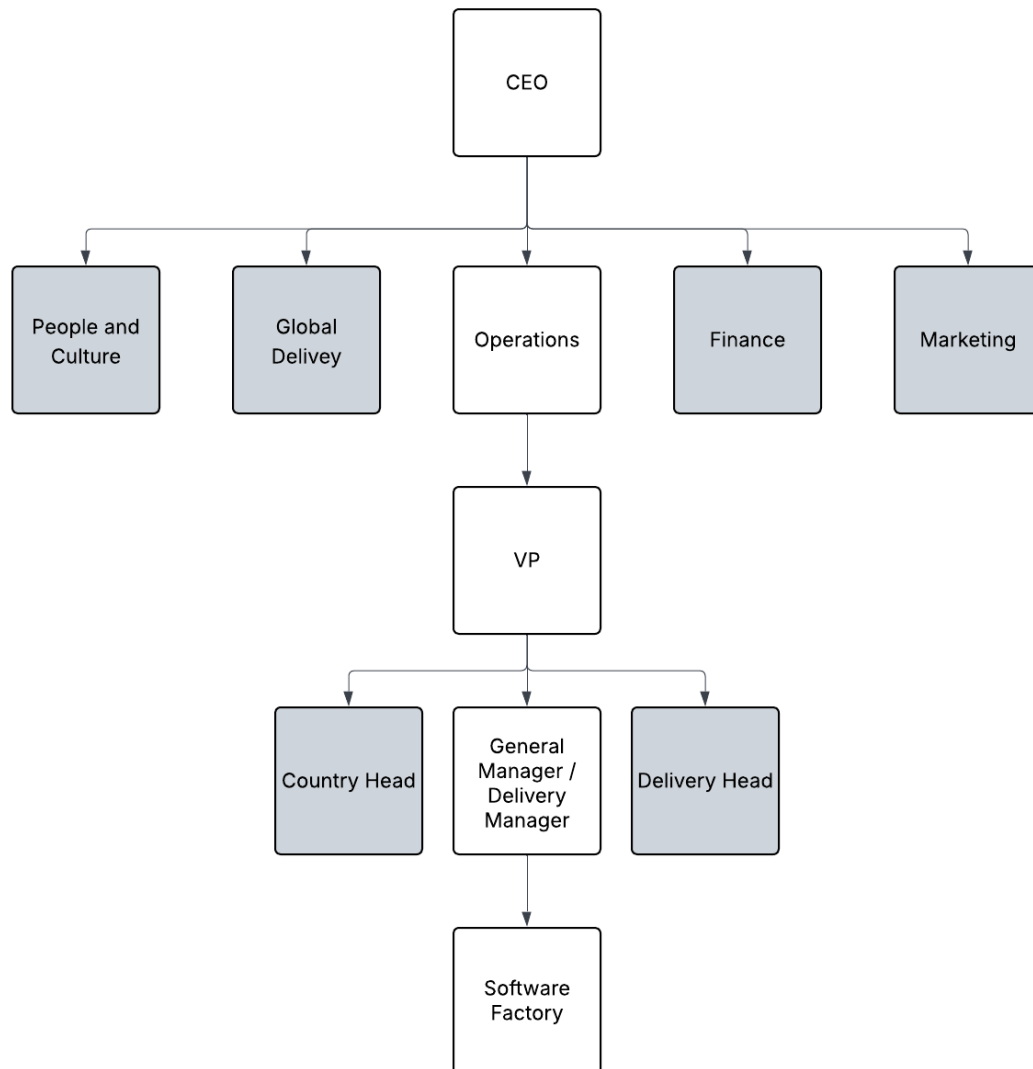


Nota. Fuente: Elaboración propia.

Aunque en el organigrama podemos visualizar las áreas principales, este trabajo se centrará en el despliegue del área de “Operations”, como se muestra en la Ilustración 2, dado que el autor está dentro de la subárea “Software Factory”

Ilustración 2

Organigrama enfocado en Software Factory



Nota. Fuente: Elaboración propia.

2.5. ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS

El autor del presente trabajo desempeñó el puesto de “Senior Backend Developer Java” en el área de Software Factory, durante el desarrollo de sus actividades profesionales, tuvo la oportunidad de asumir diversas responsabilidades que le permitieron fortalecer sus competencias técnicas y contribuir al cumplimiento de los objetivos del proyecto. Entre las más relevantes se encuentran:

- Contribuir en el diseño, desarrollo e implementación de microservicios utilizando el framework Spring Boot, siguiendo principios de arquitectura limpia y buenas prácticas de desarrollo.
- Realizar la migración de la capa de persistencia desde Apache HBase hacia MongoDB, asegurando la continuidad operativa y funcional, la integridad de los datos y la optimización del acceso a la información.
- Participar en el análisis y adaptación de modelos de datos, ajustándolos a las características del nuevo motor de base de datos.
- Colaborar activamente en la integración de servicios dentro de una arquitectura de microservicios desplegada en la nube de Azure, considerando aspectos de escalabilidad y disponibilidad.
- Apoyar en la configuración y consumo de APIs, asegurando una correcta comunicación entre los distintos componentes del sistema.
- Brindar soporte en la revisión de código y promover la adopción de buenas prácticas de desarrollo, orientadas a la calidad, mantenibilidad y seguridad del software.
- Participar en la identificación y resolución de incidencias técnicas, tanto en ambientes de desarrollo, como en QA y producción.
- Fomentar la adopción de nuevas herramientas y soluciones tecnológicas actuales que aporten mejoras en rendimiento y calidad del proyecto.
- Colaborar con el equipo de QA, Control de Cambios, Ciberseguridad y Riesgos para la correcta evaluación y aprobación para liberar las nuevas versiones a producción.

2.6. EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN

En mi trayectoria laboral dentro en una empresa consultora de software, brindando servicios tecnológicos a un cliente de entidad bancaria, tuve la oportunidad de involucrarme activamente en diversos proyectos enfocados en el cambio tecnológico, aportando de esta manera a la transformación digital de sistemas críticos. Durante esta experiencia, participé en iniciativas que buscaban la escalabilidad, mantenibilidad y rendimiento de las aplicaciones existentes.

Entre las principales actividades desarrolladas destacan las siguientes:

- Participé de manera activa en la migración y actualización de un conjunto de microservicios desarrollados con Spring Boot, los cuales fueron modernizados desde la versión 1.5 con Java 8 hacia la versión 2.7 con Java 11, asegurando la compatibilidad funcional y la estabilidad de los servicios durante el proceso de transición.
- Contribuí en la creación y estandarización de un nuevo componente tecnológico, diseñado para ser reutilizado por la entidad bancaria, alineado con la actualización a Java 17 y Spring Boot 3, siguiendo lineamientos de arquitectura moderna y buenas prácticas de desarrollo, este componente iba a ser utilizado para toda la entidad bancaria ya que era una librería padre.
- Colaboré en la integración de los microservicios dentro de una arquitectura distribuida, garantizando una correcta comunicación entre servicios y su despliegue en entornos corporativos de alta disponibilidad.
- Apoyé en la revisión de código y validación técnica de las soluciones implementadas, promoviendo el uso de buenas prácticas, estándares de calidad y principios de desarrollo limpio.
- Participé en la coordinación con diversos equipos, entre ellos el área de QA, Control de cambios, Ciberseguridad y Riesgo, para asegurar la correcta liberación de los desarrollos en los ambientes productivos.
- Contribuí activamente en la documentación técnica de los cambios realizados, facilitando la comprensión, mantenimiento y evolución futura de las soluciones implementadas.

Durante estos proyectos también tuve que enfrentar retos técnicos propios de trabajar con sistemas bancarios, donde cualquier cambio debía evaluarse cuidadosamente antes de pasar a producción. Esto implicaba revisar configuraciones, validar compatibilidades entre versiones y asegurar que los servicios mantuvieran su estabilidad. Esta experiencia me permitió entender la importancia de planificar adecuadamente cada modificación y de trabajar con responsabilidad en entornos de alta criticidad.

Asimismo, el trabajo constante con distintos equipos me ayudó a desarrollar una visión más integral del ciclo de vida del software, comprendiendo no solo la parte de desarrollo, sino también los procesos de validación, seguridad y despliegue. Esto fortaleció mi capacidad para adaptarme a estándares corporativos y para aportar soluciones alineadas con las necesidades reales del negocio.

Este puesto me permitió adquirir y consolidar experiencia práctica en el uso de tecnologías como Spring Boot, Java, MongoDB y arquitectura de microservicios, aplicadas en entornos bancarios de gran escala y alta criticidad. En el ámbito profesional, esta vivencia reforzó mis habilidades técnicas, mi capacidad para colaborar en equipo y mi interés en ofrecer soluciones creativas y factibles que beneficien a la organización.

CAPÍTULO III - ACTIVIDADES DESARROLLADAS

3.1. SITUACIÓN PROBLEMÁTICA

A nivel mundial, la gestión eficiente de grandes volúmenes de información se ha convertido en un desafío estratégico para las organizaciones, especialmente aquellas que dependen de arquitecturas basadas en microservicios y orientadas a la explotación de datos en tiempo real para poder usarlo y sacarle el mayor beneficio. En este contexto, las bases de datos NoSQL han adquirido un rol fundamental, ya que permiten manejar información no estructurados con mayor flexibilidad frente a los sistemas relacionales tradicionales que tienen un enfoque más complejo y con mayor curva de aprendizaje al momento de plantear el enfoque de su arquitectura.

Entre estas tecnologías, Apache HBase ha sido ampliamente utilizada como base de datos distribuida dentro del ecosistema Hadoop. Sin embargo, si bien su arquitectura orientada a columnas ofrece ventajas para determinados escenarios de análisis masivo, en la práctica muchas organizaciones han identificado limitaciones relacionadas con su curva de aprendizaje, el alto consumo de recursos y las dificultades de integración con aplicaciones modernas basadas en microservicios de esta manera aumentando en gastos considerables dentro de la organización.

En el caso del cliente de la consultora de software, diversas organizaciones han apostado por infraestructuras escalables para soportar sus procesos de transformación digital, así que debido a esto se impulsa la transformación digital a nuevas soluciones. No obstante, la dependencia de sistemas como Apache HBase plantea dificultades al momento de responder con agilidad a las demandas del negocio, ya que mantener servidores basados en Hadoop y Zookeeper significa tener costos elevados y requerimientos técnicos que no siempre están disponibles en los equipos locales. Esta situación genera retrasos en los desarrollos, mayores costos operativos y una menor capacidad de innovación lo cual resulta en pérdidas para los clientes a largo plazo, ya que la transformación digital debe ser constante.

En este escenario, MongoDB surge como una alternativa moderna y altamente adoptada a nivel internacional. Su modelo orientado a documentos, su compatibilidad nativa con JSON, su facilidad de integración con microservicios desarrollados en Java y Spring Boot, así como su menor curva de aprendizaje, facilidad de integración y escalabilidad, lo convierten en una opción idónea para reemplazar sistemas tradicionales

como Apache HBase que como bien venimos comentando es una idea buena para sistemas legacy, pero en el caso de hoy también queremos impulsar la transformación digital.

3.1.1. DEFINICIÓN DEL PROBLEMA

3.2.1.1 Problema principal

La organización presenta dificultades en la gestión de la capa de persistencia de sus microservicios debido al uso de Apache HBase, cuya complejidad técnica y limitada integración con Spring Boot generan procesos de desarrollo más lentos, mayor curva de aprendizaje y mayores costos de mantenimiento. Esta situación afecta la eficiencia del equipo de desarrollo y limita la capacidad de adaptación tecnológica dentro de una arquitectura de microservicios desplegada en Azure.

3.2.1.2 Problemas específicos

- La dependencia de microservicios que utilizan Apache HBase, lo cual dificulta la transición hacia una tecnología más flexible sin afectar la lógica de negocio existente.
- La alta complejidad del acceso a datos y la curva de aprendizaje asociada a HBase, lo que impacta en la productividad del equipo de desarrollo.
- La falta de lineamientos y buenas prácticas estandarizadas para facilitar la integración de la capa de persistencia con Spring Boot dentro de la arquitectura actual.

3.2. SOLUCIÓN

La solución planteada consiste en la migración de la capa de persistencia desde Apache HBase hacia MongoDB, con el fin de superar las limitaciones técnicas y operativas asociadas al uso de HBase en entornos de microservicios. Esta migración permitirá optimizar la gestión de datos mediante un modelo orientado a documentos, mejorar la integración con aplicaciones desarrolladas en Spring Boot y reducir la complejidad de la curva de aprendizaje en cuanto a los desarrollos actuales o futuros en los equipos comprometidos. De esta manera, se busca garantizar mayor escalabilidad, flexibilidad en el manejo de estructuras de información mejorada, favoreciendo la entrega continua de valor. Así la organización podrá responder con mayor rapidez a las

necesidades del negocio, reducir costos operativos y asegurar la confiabilidad de los sistemas desarrollados.

3.2.1. OBJETIVOS

3.2.1.1 Objetivo principal

Migrar la capa de persistencia de los microservicios del cliente hacia MongoDB, utilizando Spring Boot en una arquitectura basada en microservicios en Azure, garantizando la mejora en los tiempos de desarrollo y optimización de los recursos de la organización en un periodo de 5 meses.

3.2.1.2 Objetivos específicos

- Migrar exitosamente 17 microservicios que inicialmente trabajan con Apache Hbase hacia MongoDB sin afectar la lógica que se encuentra actualmente para evitar problemas en las distintas áreas del cliente.
- Optimizar la productividad del equipo de desarrollo, reduciendo la complejidad del acceso a datos y la curva de aprendizaje de la tecnología de persistencia.
- Definir un conjunto de buenas prácticas y soluciones efectivas para la migración del motor de base de datos, junto con el uso de la fácil integración de MongoDB con Spring Boot.

3.2.2. ALCANCE

Alcance Funcional

El proyecto contempla que la migración de los microservicios utilizados por el cliente de la consultora de software, seguirán actuando de la misma manera sin afectar la lógica de programación, teniendo en cuenta que sólo se migró el motor de base de datos, de esta manera minimizando los tiempos de desarrollo actuales y a futuro.

Alcance Técnico

Desde el punto de vista técnico, la migración implicó la migración de los repositorios originalmente implementados con clientes de Hadoop y Zookeeper,

adaptándolos a la implementación con Spring Data MongoDB, ya que este cambio hace más fácil la conexión a la base de datos MongoDB.

También se ofrece una mejora tecnológica al fomentar el uso de Spring Data MongoDB, ya que nos brinda un conjunto amplio de métodos útiles para el acceso a los datos. Esto permitirá reemplazar una implementación compleja que ofrecía Apache HBase a una solución mucho más flexible y alineada con el cambio tecnológico actual.

3.2.3. ETAPAS Y METODOLOGÍA

El desarrollo del proyecto se llevó a cabo bajo el marco de trabajo Scrum, el cual se caracteriza por su enfoque ágil, iterativo e incremental, permitiendo adaptarse de manera efectiva a los cambios y garantizando la entrega continua de valor. Dado que la migración de microservicios desde Apache HBase hacia MongoDB requería una planificación gradual, coordinación constante y pruebas frecuentes, Scrum se convirtió en la metodología idónea para gestionar este proceso. En la tabla 6 se pueden mostrar los roles del proyecto.

Tabla 6

Roles del Proyecto

Roles	Descripción	Cantidad
Product Owner	Responsable de priorizar el Product Backlog, asegurando que el producto cumpla con los requerimientos del cliente.	1
Scrum Master	Encargado de facilitar y hacer respetar la metodología Scrum, mejorar el rendimiento del equipo, eliminar obstáculos y fomentando la mejora continua.	1
Equipo de Desarrollo	Equipo responsable de crear el producto según los requerimientos priorizados por el Product Owner de esta manera cumpliendo los objetivos del Sprint.	5

Nota. Fuente: Elaboración propia.

El proyecto fue organizado en Sprints de 4 semanas, en los que se definieron entregables específicos y se realizaron revisiones periódicas con el equipo. Las ceremonias de Scrum (planificación, dailies, revisión y retrospectiva) aseguraron una comunicación constante, la detección temprana de problemas y la alineación con los objetivos técnicos del proyecto. A continuación, se describirán las ceremonias mencionadas.

Planificación del Sprint

- Durante las reuniones de planificación, el Product Owner presentó las historias de usuario relacionadas con la migración de los microservicios.
- Cada historia de usuario fue evaluada y priorizada en función de su impacto en el sistema y su complejidad técnica.
- Se definieron criterios de aceptación claros para cada historia, asegurando que todos los miembros del equipo tuvieran un entendimiento común de los entregables.
- Se realizó la estimación de esfuerzo en puntos de historia, tomando en cuenta la capacidad del equipo y la complejidad de cada tarea. En la tabla 7 se puede visualizar las estimaciones de tiempo para los puntos de historia por cada sprint.

Tabla 7

Puntos de las historias

Puntos de Historia	Complejidad	Esfuerzo	Duración
2	Bajo	Bajo	10 días
3	Medio Bajo	Medio Bajo	15 días
5	Medio	Medio	20 días
8	Medio Alto	Medio Alto	30 días

Nota. Fuente: Elaboración propia.

Ejecución del Sprint

- El equipo de desarrollo trabajó en la migración progresiva de los microservicios, asegurando la compatibilidad con la lógica existente.
- Se llevaron a cabo pruebas unitarias y de integración para verificar que los servicios respondieran correctamente tras la migración a MongoDB.
- Se utilizaron entornos de prueba en la nube de Azure, lo que permitió validar los despliegues antes de su paso a producción.

Revisión del Sprint

- Al finalizar cada Sprint, el Product Owner hace una revisión del backlog y verifica el estado de las historias en el Sprint, si ha de ser el caso las historias no culminadas siguen para el siguiente Sprint.

Retrospectiva del Sprint

- En las retrospectivas, el equipo evalúa fortalezas y debilidades de cada ciclo, proponiendo mejoras para el siguiente Sprint.
- Se recopila una retroalimentación de los miembros del equipo con la finalidad de minimizar errores recurrentes y ayuda con los bloqueos con otras áreas si ha de ser el caso.

Con esta metodología, se garantizó que el proyecto avanzara de manera controlada, reduciendo riesgos, asegurando la calidad del software entregado y cumpliendo con los objetivos planteados en los plazos establecidos.

3.2.4. FUNDAMENTOS UTILIZADOS

MongoDB

MongoDB es una base de no relacional desarrollado por MongoDB, Inc. Según (MongoDB Inc, 2025) MongoDB es una base de datos que almacena datos de una manera distinta a la tradicional, que se puede interpretar como documentos, que hace mucho más fácil su acceso a consultas ya que tiene un formato amigable como lo es el formato JSON, lo que le permite manejar datos semiestructurados y estructuras dinámicas de manera eficiente, en palabras cortas como cualquier tipo de base de datos NoSQL o no relacional, puedes agregar un campo más sin que los demás registros se vean afectados, de esta manera volviendo más flexible a nuestra base de datos.

En la Ilustración 3, MongoDB emplea un modelo en base a documentos, donde la unidad que veníamos comentando es JSON y se muestra su manera binaria en la que lo representa (BSON). Cada documento tiene pares clave – valor y puedes incluir estructuras anidadas como subdocumentos o arreglos de distintos tipos de datos, lo que permite representar información más compleja de forma natural a lo que se representa en la base de datos relacional y sin imponer una estructura fija para cada documento.

Ilustración 3

Modelo de datos orientado a documentos en MongoDB



Nota. Fuente: (MongoDB Inc, 2025).

El modelo de datos de documentos de MongoDB ofrece de forma natural una asignación de objetos en el código de la aplicación, lo que como tal facilita mucho para la curva de aprendizaje y el uso por parte de los actuales y futuros desarrolladores y que por consecuencia hace mucho más fácil la integración hacia aplicaciones.

Dado que los documentos pueden reunir datos relacionados que a diferencia de un enfoque tradicional relacional “padre-hijo”, MongoDB los puede integrar en un mismo documento ya que soporta ese tipo de datos y de esta manera satisface las necesidades de integración de datos de la mayoría de las aplicaciones. Se puede escribir uno o más campos en una sola operación haciendo más flexible y dinámico el uso de esta tecnología.

Además, MongoDB ofrece escalabilidad horizontal mediante sharding, lo que permite distribuir datos entre múltiples servidores para mejorar el rendimiento y la capacidad de almacenamiento. También incluye mecanismos de replicación, lo que garantiza disponibilidad y recuperación ante fallos. Su integración con frameworks modernos, como Spring Boot, facilita la creación de aplicaciones basadas en microservicios, ya que permite manejar de manera sencilla consultas complejas, índices y operaciones de agregación sobre los documentos. Por estas características, MongoDB se ha consolidado como una opción confiable y flexible para proyectos que requieren rapidez y adaptabilidad en el manejo de datos. (Truică, 2018)

MongoDB también posee compatibilidad con transacciones multidocumento, lo que lo convierte en la única base de datos que combina las garantías ACID de las bases de datos relacionales tradicionales.

La comunidad de desarrolladores prefiere trabajar con MongoDB debido a la flexibilidad y fácil aprendizaje de integración con los diversos lenguajes de programación (6sense, 2025). MongoDB representa a más del 40% de desarrolladores actuales con esta

tecnología, es decir es la base de datos no relacional más usada en los tiempos actuales, debido a su gran carga laboral los desarrolladores actuales y futuros tienden a aprender esta tecnología vital para sobrevivir en el mercado laboral.

A diferencia de HBase, MongoDB no requiere un esquema fijo, lo que facilita la evolución de la aplicación sin necesidad de realizar cambios complejos en la base de datos. Esto es especialmente útil en entornos de desarrollo ágil, donde los requerimientos pueden modificarse con frecuencia.

Apache HBase

Según (Apache Software Foundation, 2024), Apache HBase es una base de datos NoSQL distribuida y orientada a columnas, desarrollada sobre Hadoop, que permite almacenar grandes volúmenes de información de manera escalable y consistente. Se caracteriza por su capacidad de manejar datos estructurados y semiestructurados, lo que la hace ideal para aplicaciones que requieren rapidez en consultas sobre grandes conjuntos de datos. A diferencia de las bases de datos relacionales tradicionales, Apache HBase no depende de esquemas rígidos, lo que brinda mayor flexibilidad a la hora de almacenar información que puede cambiar en el tiempo.

Apache HBase ofrece alta disponibilidad y tolerancia a fallos gracias a su arquitectura distribuida. Cada tabla se divide en regiones que se distribuyen entre múltiples servidores, permitiendo que el sistema continúe operando incluso si uno de los nodos falla. Además, soporta operaciones de lectura y escritura en tiempo real, lo que resulta muy útil para aplicaciones críticas que requieren acceso inmediato a la información. Esta capacidad de escalamiento horizontal y consistencia fuerte hace que Apache HBase haya sido una opción popular en grandes empresas y proyectos de big data.

Apache HBase se integra de forma nativa con el ecosistema de Apache Hadoop, lo que le permite escalar horizontalmente, tolerar fallos y almacenamiento *HDFS*. A diferencia de las bases de datos tradicionales relacionales, HBase utiliza un sistema de organización de la información enfocado en tablas compuestas por filas, familias de columnas y columnas, permitiendo el almacenamiento flexible. Esto lo hace flexible cuando almacena registros de eventos o datos temporales históricos de gran tamaño (Apache Software Foundation, 2023).

La arquitectura de HBase comprende varios componentes clave, entre ellos están HMaster que sirve para la gestión del clúster y balanceo de carga; RegionServer que son

nodos que encargan de almacenar regiones las cuales son particiones horizontales de las tablas; HDFS, que es un sistema de almacenamiento donde se persisten los datos; y Apache ZooKeeper que toma un rol muy importante, el cual se encarga de la sincronización, el estado del clúster y detección de fallos (Apache Software Foundation, 2022). El uso de HBase comprende una terminología muy importante a tener consideración:

- Tabla (Table): Conjunto de filas organizadas en familias de columnas.
- Fila (Row): Identificada por una Row Key única.
- Familia de columnas (Column Family): Agrupación lógica de columnas, definida previamente en el modelado.
- Columna (Column): Par clave – valor almacenado dentro de una familia.
- Región (Region): Fragmento horizontal de una tabla.

En la Ilustración 4, se puede visualizar como se manejan la arquitectura de los datos en HBase teniendo en cuenta la terminología anteriormente vista.

Ilustración 4

Administración de la información dentro de una tabla en HBase

Row Key	Column Family		Column Family	
	Column Qualifier	Column Qualifier	Column Qualifier	Column Qualifier
Rk-1	Cell			•

Nota. Fuente: (Hassan, 2025).

También (Apache Software Foundation, 2024) demuestra que hay ciertas limitaciones que tendrán un impacto significativo dentro del presente trabajo. Las cuales son:

- Alta complejidad operativa y administrativa, esto básicamente es porque el uso de Apache HBase requiere la administración en conjunto de Hadoop, HBase y ZooKeeper, lo que incrementa la dificultad para su despliegue, monitoreo y mantenimiento del clúster, esto implica de por si muchos más gastos operativos.
- Dependencia crítica de Apache ZooKeeper, esto significa que Apache HBase no puede funcionar sin ZooKeeper, que en escenarios complejos si no hay disponibilidad de ZooKeeper puede afectar seriamente al funcionamiento del sistema.

- Curva de aprendizaje elevada, debido a que el uso de familias de columnas y claves de fila significa un tiempo considerable invertido para comprender la tecnología, especialmente en comparación con otras bases de datos NoSQL.
- Costos elevados en la nube, debido a que depende del uso de otras herramientas para su correcto funcionamiento y esto significa mayores costos ya que necesita una infraestructura más especializada.

Spring Boot

Spring Boot es un framework de desarrollo basado en Spring que simplifica la creación de aplicaciones basadas en Java, especialmente orientadas a microservicios. Su principal ventaja es la configuración automática, que reduce significativamente la necesidad de ajustes manuales engorrosos como en sus versiones iniciales, de esta manera permite que los desarrolladores se concentren directamente en la lógica del negocio (International Research Journal of Engineering and Technology, 2024).

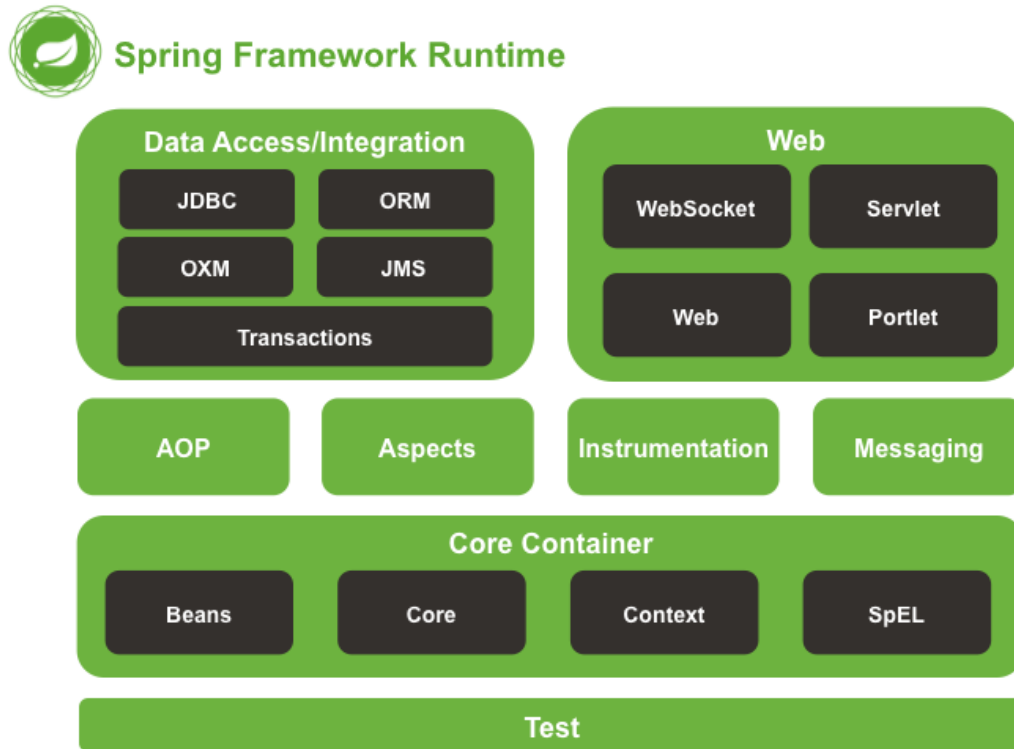
Según (Pivotal Software, Inc, 2024), una de las tantas características principales de Spring Boot es autoconfigurar componentes en función de las dependencias incluidas dentro de cada proyecto. Mediante el uso de *starters* (como *spring-boot-starter-web* o *spring-boot-starter-data-mongodb*), el framework es capaz de configurar servidores embebidos, conectividad a diversas bases de datos, serialización JSON, facilitando de esta manera un rápido despliegue de versiones productivas.

Además, en el contexto de microservicios, Spring Boot facilita la creación de servicios independientes, cada uno con su propio contexto y configuraciones, pero capaz de interactuar con otros servicios mediante API REST. Además, incluye soporte para pruebas automatizadas, gestión de dependencias y herramientas de monitoreo, lo que asegura calidad, mantenibilidad y facilidad de despliegue. Su compatibilidad con MongoDB permite trabajar de manera directa con documentos JSON, simplificando operaciones CRUD y consultas complejas. Esto hace que Spring Boot sea una opción muy utilizada en proyectos de migración y modernización de sistemas basados en microservicios.

En Ilustración 5, podemos visualizar como un enfoque más amplio de lo que es Spring, cabe mencionar que Spring Boot como tal es una manera de simplificar la creación de aplicaciones en Java, pero Spring Framework o simplemente Spring es el centro de todo, ya que junto con sus demás aplicaciones cumplen un rol específico dentro de la aplicación.

Ilustración 5

Arquitectura de Spring Framework



Nota. Estructura principal de Spring Framework Fuente: (Spring Framework Documentation, 2025).

Según (Spring Framework Documentation, 2025), Spring Framework es un ecosistema modular organizado por múltiples capas funcionales, donde cada módulo cumple una responsabilidad única dentro de la aplicación. Entre todas estas capas vemos una que se va a adecuar dentro del presente trabajo. El módulo Data Access/Integration proporciona herramientas para interactuar con diferentes motores de bases de datos, en pocas palabras el módulo enfocado para ello se llama Spring Data, el cuál su labor principal es minimizar la complejidad para el acceso a datos, reduciendo de manera considerable el código engorroso.

Spring Data no es una tecnología inherente para todos los motores de bases de datos, sino más bien es el padre dentro de una familia de subproyectos relacionados para distintos motores de bases de datos. Uno de estos subproyectos es Spring Data MongoDB, el cual permite una conexión directa entre aplicaciones Spring y bases de datos MongoDB la cuáles son NoSQL orientadas a documentos.

Según (VMware, Inc, 2024), Spring Data MongoDB forma parte del ecosistema Spring Data, como ya veníamos mencionando, esto quiere decir básicamente que el propósito y razón de ser de Spring Data MongoDB es simplificar el acceso a datos para las aplicaciones Spring Boot que usan un motor de base datos MongoDB. Este módulo permite mapear directamente documentos en MongoDB hacia objetos en Java, los llamados *POJO* mediante anotaciones, eliminando la necesidad de escribir código de acceso a datos. También trae una interfaz capaz de ejercer operaciones *CRUD* dentro de la aplicación de esta manera reduciendo el código repetitivo entre diversas clases, incluso configuraciones de estas, así ganando tiempo significativo entre los desarrollos, además también trae objetos adicionales que funcionan para hacer consultas complejas y elaboradas, como lo es *Criteria* y *Query*. A diferencia de Apache HBase, donde las consultas suelen depender de los filtros a nivel de filas y familias de columnas, Spring Data MongoDB permite definir consultas mediante los últimos objetos ya mencionados, facilitando la implementación de una lógica de búsqueda más elaborada.

Arquitectura de microservicios

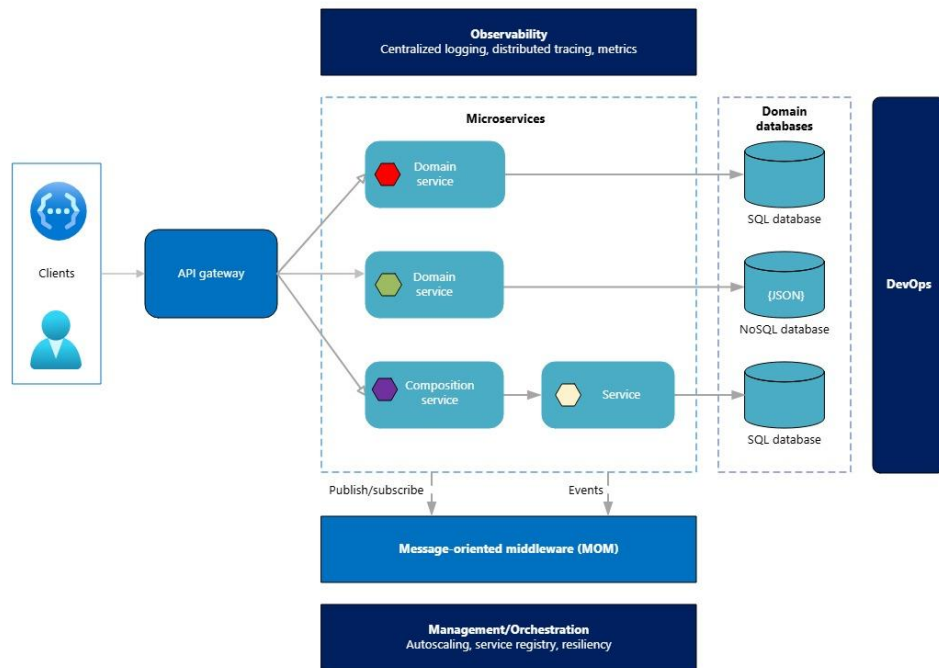
La arquitectura de microservicios consiste en dividir una aplicación en múltiples servicios independientes, cada uno encargado de una funcionalidad específica. A diferencia de las aplicaciones monolíticas, donde todo el código se encuentra en un solo bloque, los microservicios permiten que cada componente sea desarrollado, desplegado y escalado de manera autónoma. Esto ofrece varias ventajas, como mayor flexibilidad, facilidad para mantener el sistema y la posibilidad de integrar nuevas tecnologías de manera progresiva. (Akinpalli, 2024)

Cada microservicio se comunica con otros mediante APIs, generalmente a través de protocolos ligeros como REST, lo que asegura independencia y modularidad. Además, esta arquitectura facilita la adopción de metodologías ágiles, como Scrum, porque los equipos pueden trabajar en servicios específicos sin generar conflictos con otras áreas del sistema. La implementación de microservicios requiere planificación en aspectos como la gestión de datos, la seguridad, el monitoreo y la tolerancia a fallos, pero cuando se aplica correctamente, permite desarrollar sistemas altamente escalables y resilientes, capaces de adaptarse a la demanda y al crecimiento de la organización. (Vashisht, 2025).

En la Ilustración 6, vemos una arquitectura de microservicios típica donde diversos servicios con diferentes roles, exponen APIs para comunicarse entre sí o con clientes externos.

Ilustración 6

Arquitectura de microservicios



Nota. Arquitectura de microservicios. Fuente: (Microsoft Azure, Inc, 2025).

Según (Atlassian, Inc, 2025), las ventajas claves para el uso de microservicios en un enfoque actual y moderno son:

- Escalabilidad independiente, esto quiere decir que cada microservicio se puede escalar según sus necesidades de negocio, reduciendo costos, optimizando recursos y su fácil mantenimiento.
- Desarrollo y despliegues sencillos, esto quiere decir una mayor facilidad de despliegue de cada microservicio sin tener la necesidad de apagar todo el sistema, ya que este enfoque permite un desarrollo más independiente.
- Tolerancia a fallos, esto quiere decir que aunque existan fallos de algunos microservicios el sistema completa no dejará de funcionar, comparado con el enfoque antiguo esto es una ganancia significativo de tiempo de respuesta de solución ante los problemas que puedan suceder.

- Tecnología heterogénea, esto quiere decir que distintas tecnologías o bases de datos pueden convivir dentro de un mismo sistema ya que no limita el uso de las mismas.

Microsoft Azure

Según (Microsoft Azure, 2025), Microsoft Azure es una plataforma de computación en la nube desarrollada por Microsoft que ofrece un amplio conjunto de servicios para el desarrollo, despliegue y administración de aplicaciones mediante una red global de centros de datos. Azure se fundamenta en el paradigma de cloud computing, permitiendo a las organizaciones consumir recursos tecnológicos bajo demanda a través de modelos como Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS). A continuación, se dará más detalle sobre estos 3 modelos:

- Infraestructura como Servicio (IaaS), este modelo proporciona recursos de computación, almacenamiento y redes, accesibles mediante el portal, eliminando la necesidad de gestionar hardware físico y facilitando la escalabilidad de este, de esta manera minimiza de manera significativa los costos operativos.
- Plataforma como servicio (PaaS), este modelo ofrece un entorno completo para desarrollar, implementar y administrar aplicaciones, de esta manera lograr independizar a los desarrolladores de todo lo relacionado a la infraestructura, es decir, el sistema operativo, middleware, redes, etc.).
- Software como servicio (SaaS), este modelo entre aplicaciones directamente para usar desde tu navegador, esto significa que no necesitas instalar nada localmente, mantener una infraestructura, como ocurre con sistemas tradicionales.

Según (Microsoft Azure, 2025), usar Microsoft Azure trae consigo diversas ventajas para los negocios. A continuación, veremos algunas:

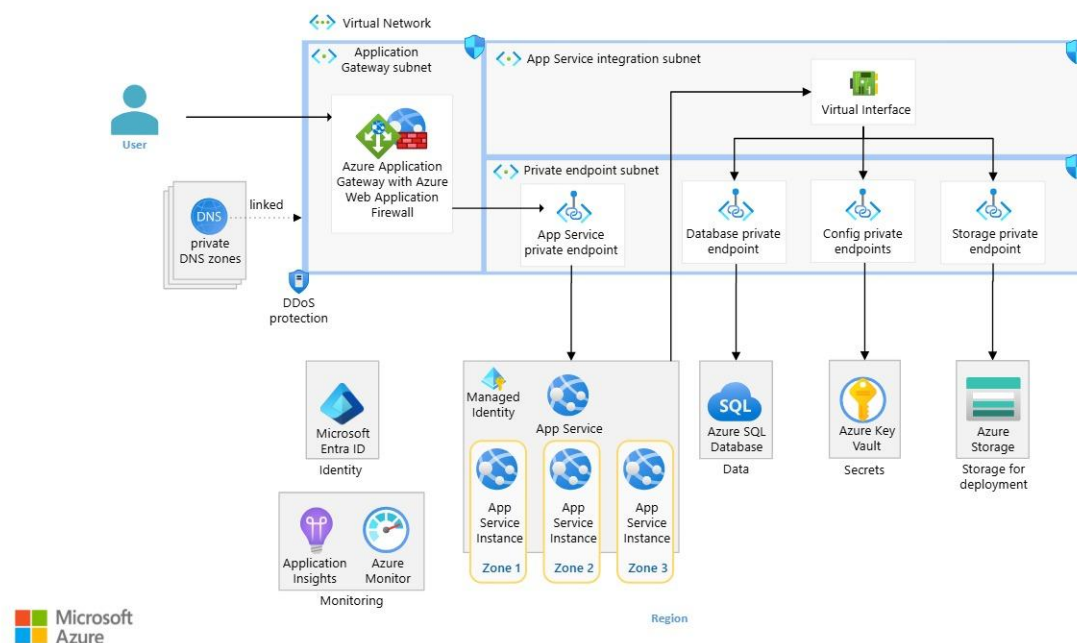
- Escalabilidad y elasticidad, esto quiere decir que Azure internamente adapta dinámicamente los recursos en función de demanda, es decir que automáticamente agiliza los tiempos de respuesta cuando existen picos de uso o crecimiento de la aplicación.
- Optimización de costos, esta ventaja va asociada directamente a las aplicaciones que uno quiera usar, es decir solo uno paga por lo que consume y es directamente proporcional al ajuste del mercado, comparado con el enfoque de hardware físico tradicional, su ganancia es mucho mayor.

- Amplia cartera de servicios, esta ventaja aborda a sus diversas aplicaciones para distintas necesidades de negocio que puedan requerir las organizaciones.
- Alta disponibilidad y resiliencia, Azure cuenta con diversas regiones y zonas de disponibilidad alrededor del mundo, esto garantiza disponibilidad continua y tolerancia a fallos.

En la Ilustración podemos visualizar una arquitectura común de Azure y cuáles son sus principales herramientas, entre las cuales destacan *Azure Key Vault*, *Microsoft Entra ID*, *APIM*, *Azure SQL Database*, etc.

Ilustración 7

Arquitectura de microservicios con aplicaciones de Azure



Nota. Ejemplo de una arquitectura de microservicios con aplicaciones de Azure. Fuente: (Microsoft Azure, Inc, 2025)

Entonces toda esta información va de acuerdo con lo que plantea el presente trabajo, trabajar con Azure bajo este enfoque de microservicios en la nube, genera una rápida adaptabilidad frente a cambios tecnológicos, es decir si mantenemos el uso de Azure, reduce la complejidad del proceso de migración.

Scrum

Scrum es una metodología ágil que permite gestionar proyectos de manera organizada y flexible, especialmente cuando se trabaja en entornos donde los

requerimientos pueden cambiar rápidamente. Su enfoque se basa en dividir el trabajo en ciclos cortos llamados Sprint, los cuales tienen una duración limitada y al final de cada uno se entrega un resultado funcional y evaluable. Esto permite que el equipo pueda ajustar sus objetivos constantemente y responder con rapidez a nuevas necesidades, asegurando que el proyecto avance de manera ordenada. (Scrum Organization, 2017)

Según (Scrum Organization, 2020), uno de los puntos fuertes de Scrum es la definición clara de roles dentro del equipo. A continuación, se describirán brevemente los roles:

- Product Owner: encargado de priorizar las funcionalidades y de representar las necesidades del negocio, asegurando que lo que se desarrolle tenga valor real para los usuarios finales.
- Scrum Master: actúa como facilitador del proceso, ayudando a eliminar obstáculos y promoviendo la adopción de prácticas ágiles.
- Equipo de desarrollo: responsable de implementar las tareas, asegurando la calidad técnica y funcional de cada entrega.

Además, Scrum establece reuniones periódicas, como las Daily, donde el equipo revisa avances y bloqueos; las Sprint Review, donde se presentan los resultados a los interesados; y las Sprint Retrospective, para analizar procesos y mejorar continuamente.

En el contexto de proyectos de migración tecnológica y desarrollo de sistemas basados en microservicios, este marco teórico resulta relevante, ya que facilita la adaptación progresiva a cambios técnicos y organizacionales.

3.2.5. IMPLEMENTACIÓN DE LAS ÁREAS, PROCESOS, SISTEMAS Y BUENAS PRÁCTICAS

La implementación del proyecto se dividió en 4 Sprint, cada uno con 4 semanas de duración. En la tabla 8, se detallan las historias de usuario que componen cada uno de ellos.

Tabla 8

Sprints del Proyecto

Sprint	Sprint Backlog
Sprint 1	- Migración del microservicio ms-campanascliente-neg - Migración del microservicio ms-cartolamt940-neg - Migración del microservicio ms-cartolasempresas-neg - Migración del microservicio ms-productoscliente-neg
Sprint 2	- Migración del microservicio ms-mesadedineroclientes-neg - Migración del microservicio ms-miactividadfinanciera-neg - Migración del microservicio ms-mismetasmovimiento-neg - Migración del microservicio ms-perfilcliente-neg
Sprint 3	- Migración del microservicio ms-transferenciaempresa-neg - Migración del microservicio ms-pagosmasivosconsulta-neg - Migración del microservicio ms-contenidofeedapp-neg - Migración del microservicio ms-planvinculacionpyme-neg
Sprint 4	- Migración del microservicio ms-saludfinanciera-neg - Migración del microservicio ms-transaccionesphoenix-util - Migración del microservicio ms-valevistacliente-neg - Migración del microservicio ms-listasriesgo-neg - Migración del microservicio ms-movimientosenriquecidoscuentas-neg

Nota. Fuente: Elaboración propia.

Planificación del Sprint 1

En la planificación del Sprint 1, se realizó la estimación de puntos y criterios de aceptación por cada historia de usuario. En las tablas 9, 10, 11 y 12 se detalla la evaluación de las historias de usuario del Sprint 1.

Tabla 9

Detalle de la Historia de Usuario del ms-campanascliente-neg

Migración del microservicio ms-campanascliente-neg	
Historia de usuario	- Como desarrollador del microservicio, quiero migrar la base de datos Apache Hbase a MongoDB para reducir el tiempo de desarrollo y mantenimiento.
Criterios de Aceptación	- Consultar campañas de cliente no debe haber afectado su funcionalidad. - Consultar campañas de cliente con filtros de cantidad de registros no debe haber afectado su funcionalidad. - Guardar interacciones sobre una campaña no debe haber afectado su funcionalidad.
Puntos de Historia	- 8

Nota. Fuente: Elaboración propia.

Tabla 10

Detalle de la Historia de usuario del ms-cartolamt940-neg

Migración del microservicio ms-cartolamt940-neg	
Historia de usuario	- Como desarrollador del microservicio, quiero migrar las dos bases de datos Apache Hbase a MongoDB para reducir el tiempo de desarrollo y mantenimiento.
Criterios de Aceptación	- Generar cartola no debe haber afectado su funcionalidad. - Gestión de cuenta cartola no debe haber afectado su funcionalidad.

Puntos de Historia - 8

Nota. Fuente: Elaboración propia.

Tabla 11

Detalle de la Historia de usuario del ms-cartolasempresas-neg

Migración del microservicio ms-cartolasempresas-neg	
Historia de usuario	- Como desarrollador del microservicio, quiero migrar las dos bases de datos Apache Hbase a MongoDB para reducir el tiempo de desarrollo y mantenimiento.
Criterios de Aceptación	- Generar cartola con moneda extranjera no debe haber afectado su funcionalidad. - Generar cartola con moneda nacional no debe haber afectado su funcionalidad.
Puntos de Historia	- 8

Nota. Fuente: Elaboración propia.

Tabla 12

Detalle de la Historia de usuario del ms-productoscliente-neg

Migración del microservicio ms-productoscliente-neg	
Historia de usuario	- Como desarrollador del microservicio, quiero migrar la base de datos Apache Hbase a MongoDB para reducir el tiempo de desarrollo y mantenimiento.
Criterios de Aceptación	- Consultar productos de cliente no debe haber afectado su funcionalidad. - Consultar productos externos del cliente no debe haber afectado su funcionalidad.
Puntos de Historia	- 8

Nota. Fuente: Elaboración propia.

Desarrollo del Sprint 1

En esta sección se describe el desarrollo de las principales historias de usuario al Sprint 1.

Migración del microservicio ms-campanascliente-neg

La migración de cada microservicio empieza por verificar el bloque de dependencias existentes, ya que todos los microservicios utilizan como gestor de dependencias a *Gradle*, buscamos dentro del *build.gradle* que es donde se almacenan todas las dependencias que tiene el proyecto. A continuación, en la Ilustración 8 son dependencias que debemos eliminar de nuestro proyecto y la que debemos agregar es en la Ilustración 9 para que se pueda integrar correctamente con MongoDB.

Ilustración 8

Dependencias de Apache Hadoop, Zookeeper y HBase

```
dependencies {  
  
    implementation 'org.apache.hadoop:hadoop-auth'  
    implementation 'org.apache.hadoop:hadoop-common'  
  
    implementation 'org.apache.zookeeper:zookeeper'  
  
    implementation 'org.apache.hbase:hbase-client'  
    implementation 'org.apache.hbase:hbase-common'  
    implementation 'org.apache.hbase:hbase-protocol'  
  
}
```

Nota. Fuente: Elaboración propia.

Las dependencias mostradas representan a Apache Hadoop, Zookeeper y HBase, serán descritas a continuación:

- **Apache Hadoop:** Son las dependencias *org.apache.hadoop:hadoop-auth* y *org.apache.hadoop:hadoop-common*.
- **Apache Zookeeper:** Es la dependencia *org.apache.zookeeper:zookeeper*.
- **Apache HBase:** Son las dependencias *org.apache.hbase:hbase-client*, *org.apache.hbase:hbase-common* y *org.apache.hbase:hbase-protocol*.

Ilustración 9

Dependencia de Spring Boot MongoDB

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'  
}
```

Nota. Fuente: Elaboración propia.

Después de eliminar las dependencias asociadas junto con las clases que involucren la conexión a Apache Hbase, Zookeeper o Hadoop se compila el proyecto para poder seguir con el desarrollo. Posteriormente a ello se empieza a configurar el yaml de configuración del microservicio para una conexión directa hacia MongoDB. A continuación, en la Ilustración 10, veremos como poder hacerlo.

Ilustración 10

Cadena de Conexión a una sola base de datos

```
spring:  
  data:  
    mongodb:  
      uri: mongodb://${atlas.mongodb.username}:${atlas.mongodb.password}@{atlas.mongodb.cluster}/{atlas.mongodb.database}
```

Nota. Fuente: Elaboración propia.

Las variables expuestas para hacer posible la conexión a la base de datos en la Ilustración 10 significan lo siguiente:

- **atlas.mongodb.username:** En esta variable va ubicado el nombre del usuario que se conecta a la base de datos mongodb.
- **atlas.mongodb.password:** En esta variable va ubicado la contraseña del usuario que se conecta a la base de datos mongodb.
- **atlas.mongodb.cluster:** En esta variable va ubicado el nombre del cluster donde se encuentra ubicado la base de datos mongodb.
- **atlas.mongodb.database:** En esta variable va ubicado el nombre de la base de datos que se conecta a la base de datos mongodb.

Una vez se configuró la conexión con éxito se procede a reestructurar el proyecto junto con la configuración de MongoDB para su conexión se empieza a desarrollar los

métodos sin afectar la lógica anterior. A continuación, en la Ilustración 11, podemos ver como estaba uno de los métodos antes del cambio.

Ilustración 11

Método obtener campana antes de la migración

```
public RegistroCampanaFeedHBaseDto getCampanaFeed(String key) {  
  
    Connection connection = hbaseConnectionConfiguration.getConnectionInstance();  
    Table table = connection.getTable(TableName.valueOf(tablaCampanas));  
  
    Get get = new Get(Bytes.toBytes(key));  
    get.setCacheBlocks(true);  
  
    Result result = table.get(get);  
  
    RegistroCampanaFeedHBaseDto registroCampanaFeedHBaseDto = RegistroCampanaFeedHBaseDto.builder().build();  
    if (result != null) {  
        registroCampanaFeedHBaseDto = resultToRegistroCampanaFeedHBaseDto(result);  
    }  
    return registroCampanaFeedHBaseDto;  
}
```

Nota. Fuente: Elaboración propia.

Como podemos ver en la Ilustración 11, visualizamos una conexión en primera instancia al llamar al objeto *Connection*, luego una llamada a la clase *Table* para poder manipular las tablas, posteriormente el llamado al registro con la clase *Get*, si nos damos cuenta todo este proceso puede ser muy engorroso para hacer un simple llamado a un registro o grupo de registros. Como pueden observar en la Ilustración 12 esto disminuye.

Ilustración 12

Método obtener campana después de la migración

```
public RegistroCampanaFeedMongoDto getCampanaFeed(String key) {  
  
    Query query = new Query();  
    query.addCriteria(Criteria.where(Constants.MONGO_ID).is(key));  
    RegistroCampanaFeedMongoDto registroCampanaFeedMongoDto = mongoTemplate.findOne(query, RegistroCampanaFeedMongoDto.class, collectionCampanas);  
    if (registroCampanaFeedMongoDto == null) {  
        registroCampanaFeedMongoDto = RegistroCampanaFeedMongoDto.builder().build();  
    }  
    return registroCampanaFeedMongoDto;  
}
```

Nota. Fuente: Elaboración propia.

Como se puede visualizar dentro del método la conexión a la tabla ya está predefinido solo configurando la cadena de conexión vista en la Ilustración 10, entonces no habría necesidad de hacer el llamado a la misma. Vemos que se usa *Criteria* para añadirle un criterio de búsqueda a nuestra *Query* y después se usa el método *findOne* para traer el registro que se está buscando mediante un filtro especificado dentro de *Criteria*.

Posteriormente a esto se empieza a migrar los métodos faltantes, para posteriormente compilar y hacer pruebas exitosas sobre la ejecución de las consultas. A

continuación, en la Ilustración 13, mostramos un ejemplo del response con su respectivos tipos de datos de la respuesta para verificar que sigue respondiendo lo mismo post migración, quedando así cerrada nuestra historia.

Ilustración 13

Response de obtener campana

```
{
  "CampanaId": "{String}",
  "User": "{String}",
  "Amount": "Long"
}
```

Nota. Fuente: Elaboración propia.

Migración del microservicio ms-cartolamt940-neg

Se empieza el cambio de la dependencia dentro del build.gradle como se pude ver en la Ilustración 8 y 9, anteriormente vista. Después de eliminar se compila el proyecto para poder seguir con el desarrollo. Posteriormente a ello se empieza a configurar el yaml de configuración del microservicio para una conexión directa hacia MongoDB. A continuación, en la Ilustración 14 se puede visualizar como poner una cadena de conexión con dos bases de datos.

Ilustración 14

Cadena de Conexión a dos bases de datos

```
spring:
  data:
    primary:
      mongodb:
        uri: mongodb://${atlas.mongodb.username}:${atlas.mongodb.password}@{atlas.mongodb.cluster}/{atlas.mongodb.database}
    secondary:
      mongodb:
        uri: mongodb://${atlas.mongodb.username}:${atlas.mongodb.password}@{atlas.mongodb.cluster}/{atlas.mongodb.database}
```

Nota. Fuente: Elaboración propia.

En este caso la configuración es distinta ya que se conecta a dos bases de datos diferentes lo que hace necesario hacer una configuración adicional, como primer paso podemos visualizar que cada conexión dentro del yaml está dividida entre primary y secondary.

La configuración se haría directamente en una clase por cada base de datos a la cual se conecta, dentro del paquete de configuración dentro del proyecto. A continuación, en Ilustración 15 y 16 se podrán visualizar los contenidos para cada clase de configuración.

Ilustración 15

Clase de configuración para la primera conexión a la base de datos

```
@Configuration
public class PrimaryMongoConfiguration {

    @Value("${spring.data.primary.mongodb.uri}")
    private String atlasMongodbDatabase;

    @Primary
    @Bean(name = "primaryMongoDatabaseFactory")
    public MongoDatabaseFactory primaryMongoDatabaseFactory() {
        return new SimpleMongoClientDatabaseFactory(atlasMongodbDatabase);
    }

    @Primary
    @Bean(name = "primaryMongoTemplate")
    public MongoTemplate primaryMongoTemplate() {
        return new MongoTemplate(primaryMongoDatabaseFactory());
    }
}
```

Nota. Fuente: Elaboración propia.

Ilustración 16

Clase de configuración para la segunda conexión a la base de datos

```
@Configuration
public class SecondaryMongoConfiguration {

    @Value("${spring.data.secondary.mongodb.uri}")
    private String atlasMongodbDatabase;

    @Bean(name = "secondaryMongoDatabaseFactory")
    public MongoDatabaseFactory secondaryMongoDatabaseFactory() {
        return new SimpleMongoClientDatabaseFactory(atlasMongodbDatabase);
    }

    @Bean(name = "secondaryMongoTemplate")
    public MongoTemplate secondaryMongoTemplate() {
        return new MongoTemplate(secondaryMongoDatabaseFactory());
    }
}
```

Nota. Fuente: Elaboración propia.

En esta primera clase vemos ciertas configuraciones que no existían en el caso de la primera historia, es decir tenemos que señalar ciertos puntos importantes y de buenas prácticas para su desarrollo y mantenimiento a futuro. En la segunda clase, también

podemos apreciar una estructura similar a la primera contando con una pequeña pero significativa diferencia. A continuación, vamos a describir los hallazgos encontrados dentro de esta clase de configuración:

- **Configuration:** La anotación Configuration sirve para especificar que la clase es un *Bean* de configuración, es decir Spring Boot tomará estas configuraciones al momento de levantar la aplicación.
- **Value:** La anotación Value sirve para asignar un valor a una variable, es decir para este caso lo estamos usando para obtener la cadena de conexión.
- **Primary:** La anotación Primary es usada para que tome una configuración como prioritaria antes de levantar el proyecto, si no se coloca genera una duplicidad entre los *Bean* lo que trae consigo errores en tiempo de ejecución.
- **Bean:** La anotación Bean sirve para transformar en un *Bean* el método o la referencia.
- **MongoTemplate:** Esta clase sirve para poder integrar la conexión contra MongoDB.
- **MongoDatabaseFactory:** Esta clase sirve para crear un cliente para la conexión a MongoDB.
- **SimpleMongoClientDatabaseFactory:** Esta es una implementación de MongoDatabaseFactory, sirve para crear un conexión mediante una cadena de conexión como parámetro.

Al igual que se cambió la configuración de la conexión, se cambia también los métodos para ser consumidos por MongoDB y no por Apache Hbase.

Ilustración 17

Buscar datos del cliente de cartola antes de la migración

```
public Optional<HbaseDto> consultarDatoClienteHbase(String datoCliente, String key) {  
    Connection connection = obtenerConexionHbase();  
    Table table = connection.getTable(TableName.valueOf(tablaListaRiesgoConfiguration.getTablaQueryHbase()));  
    String sha1 = obtenerSha1(datoCliente);  
    Get get = new Get(Bytes.toBytes(key + sha1));  
    return obtenerListaRiesgo(table.get(get));  
}
```

Nota. Fuente: Elaboración propia.

En la Ilustración 17, podemos visualizar una conexión compleja y extensa para conectarse a la base de datos de Apache HBase, muy parecida a la Ilustración 11, entonces queda claro que en los demás microservicios esas clases se repiten y por ende la implementación de la mismas puede ser repetitiva.

Ilustración 18

Buscar datos del cliente de cartola después de la migración

```
@Qualifier("primaryMongoTemplate")
private MongoTemplate primaryMongoTemplate;

public Optional<OrigenMongo> consultarDatoClienteMongo(String datoCliente, String key) {
    String sha1 = obtenerSha1(datoCliente);
    return obtenerListaRiesgo(primaryMongoTemplate.findById(key + sha1, ListaRiesgo.class));
}
```

Nota. Fuente: Elaboración propia.

Vamos a hacer énfasis en las nuevas implementaciones o los métodos nuevos que estamos viendo en la Ilustración 18. A continuación, explicamos cómo funcionan:

- **Qualifier:** Esta anotación sirve para especificar que *Bean* se está inyectando dentro de esta clase, esto como se ve en la imagen trae la configuración que se vio en la primera configuración de la conexión a la base de datos como la Ilustración 15.
- **findById:** Este es un método para hacer una búsqueda por un Id, en este caso es invocado por MongoTemplate, que ya tiene la configuración de la base de datos previamente.

Igualmente, que en el otro microservicio probamos al final con los datos necesarios y verificamos que todo es correcto para poder cerrar la historia asociada a este microservicio. A continuación, en la Ilustración 19 se puede visualizar la respuesta, mostramos un ejemplo del response con su respectivos tipos de datos de la respuesta para verificar que sigue respondiendo lo mismo post migración, quedando así cerrada nuestra historia.

Ilustración 19

Response de obtener cartolamt

```
{
  "CartolaId": "{String}",
  "User": "{String}",
  "Name": "{String}"
}
```

Nota. Fuente: Elaboración propia.

Migración del microservicio ms-cartolasempresas-neg

Se empieza el cambio de la dependencia dentro del build.gradle como se puede ver en la Ilustración 8 y 9, anteriormente presentada. Este microservicio cuenta con dos bases de datos, por lo que el proceso de migración se ejecutó siguiendo el mismo patrón aplicado en el microservicio previamente migrado con las características similares.

Al finalizar la migración, se verificó que la funcionalidad del microservicio post migración se mantienen los datos como anteriormente se trabajaba como podemos visualizar en la Ilustración 20, de esta manera también cerramos la historia asociada a este microservicio.

Ilustración 20

Response de obtener cartola

```
{
  "CartolaId": "{String}",
  "Name": "{String}"
}
```

Nota. Fuente: Elaboración propia.

Migración del microservicio ms-productoscliente-neg

Se empieza el cambio de la dependencia dentro del build.gradle como se puede ver en la Ilustración 8 y 9, pero de esta manera se encuentra un patrón en la migración del microservicio. A diferencia del último microservicio migrado, este cuenta con una base de datos, por lo tanto se siguió el mismo enfoque aplicado al primer microservicio migrado.

Posteriormente, se validó que, la funcionalidad del microservicio se mantiene post migración como podemos visualizar en la Ilustración 21, de esta manera también cerramos la historia y a la vez el Sprint.

Ilustración 21

Response de obtener producto

```
{
  "ProductoId": "{String}",
  "Subproductos": "{Array}",
  "Name": "{String}"
}
```

Nota. Fuente: Elaboración propia.

Retrospectiva del Sprint 1

En la retrospectiva del Sprint 1, se destacó el buen desempeño del equipo, que logró la migración de 4 microservicios. Se reconoció el compromiso de todos los miembros, así como la rápida atención ante la necesidad de la comunicación entre los miembros. También se llegó a un hito muy importante como lo fue que se minimizó la curva de aprendizaje incluso durante este primer Sprint, lo que permitió estandarizar el proceso de migración y sentar las bases para una ejecución más eficiente en los Sprints posteriores.

Justificación para evitar la repetición de las plannings de los Sprints posteriores

A partir del Sprint 2, no se detallan nuevamente las planificaciones ni las historias de usuario de cada microservicio, debido a que el proceso de migración definido en el Sprint 1 se mantiene constante para el resto de los microservicios. Si bien cada microservicio presenta particularidades funcionales, el patrón de migración de la capa de persistencia, los criterios de aceptación y las actividades técnicas realizadas son equivalentes.

Por este motivo, la descripción detallada de plannings y de historias de usuario para los siguientes sprints resultaría redundante y no aportaría nuevo valor técnico al presente trabajo. En su lugar, se opta por presentar un enfoque resumido, centrado en la cantidad de microservicios migrados, los resultados obtenidos y las mejoras observadas en términos de tiempo, eficiencia y curva de aprendizaje del equipo, manteniendo así la claridad y coherencia del documento.

Desarrollo del Sprint 2

Durante el Sprint 2 se continuó con la migración de microservicios aplicando el proceso técnico definido y validado en el Sprint 1. En esta iteración se migraron 4 microservicios adicionales, los cuales presentaban estructuras de datos similares y variaciones mínimas en el acceso a la información.

Gracias a la estandarización del proceso, el tiempo de implementación se redujo considerablemente y la curva de aprendizaje del equipo fue mínima. Al finalizar el Sprint, se validó que los microservicios mantuvieran su funcionalidad original, cumpliendo con los criterios de aceptación establecidos.

Retrospectiva del Sprint 2

En la retrospectiva del Sprint 2 se destacó la mejora en los tiempos de desarrollo y la mayor confianza del equipo en el proceso de migración. Asimismo, se evidenció una

mejor organización del trabajo y una reducción de incidencias técnicas durante la implementación.

Desarrollo del Sprint 3

En el Sprint 3, el proceso de migración se encontraba completamente consolidado, permitiendo migrar 4 microservicios de manera eficiente. Las actividades se enfocaron principalmente en ajustes menores de configuración y validaciones funcionales, asegurando la correcta integración con MongoDB.

El equipo logró mantener un ritmo constante de trabajo, confirmando la estabilidad del enfoque técnico adoptado.

Retrospectiva del Sprint 3

La retrospectiva del Sprint 3 evidenció una alta eficiencia en la ejecución de las tareas, así como una correcta aplicación de las buenas prácticas definidas en iteraciones anteriores.

Desarrollo del Sprint 4

Durante el Sprint 4 se finalizó la migración de los microservicios restantes, completando así el proceso de transición desde Apache HBase hacia MongoDB. En esta etapa se realizó una validación integral del sistema, verificando la correcta operación de todos los microservicios en conjunto.

Con ello, se dio por concluida la fase de desarrollo del proyecto, asegurando el cumplimiento de los objetivos planteados.

Retrospectiva del Sprint 4

En la retrospectiva final se concluyó que la planificación progresiva y el enfoque incremental permitieron alcanzar los objetivos del proyecto de manera ordenada y eficiente, destacando la correcta adopción de la metodología Scrum.

3.3. EVALUACIÓN

3.3.1. EVALUACIÓN COSTO - BENEFICIO

La migración de HBase a MongoDB, junto con la implementación de microservicios y CI/CD, permitió reducir significativamente los tiempos tanto en el desarrollo como en procesos internos. Antes de la migración, las operaciones sobre Apache HBase requerían soporte especializado y ajustes manuales complejos, lo que

generaba retrasos y afectaba la eficiencia del equipo en cuanto al tiempo de desarrollo se refiere.

Después de la migración, los tiempos de desarrollo se redujeron, lo que se traduce en ahorro de horas de trabajo y en mayor productividad. Por ejemplo, operaciones que antes tardaban varios minutos ahora se completan en segundos, y el personal dedica menos tiempo a mantenimiento y soporte, concentrándose en actividades de valor agregado. Este impacto se puede reflejar en tablas comparativas con estimaciones de ahorro anual en costos operativos y tiempos de respuesta, mostrando una reducción de aproximadamente 20% en horas-hombre dedicadas a tareas de soporte y optimización. Este enfoque también requiere establecer la base para determinar su rentabilidad. A continuación, en la Tabla 13, se presenta un desglose de los costos mensuales del equipo a cargo del proyecto.

Tabla 13

Costos del equipo

Rol	Costo Mensual	Costo Total
Product Owner	3,500 USD	14,000 USD
Scrum Master	1,500 USD	6,000 USD
Equipo de Desarrollo	20,000 USD	80,000 USD

Nota. Costos mensuales del equipo a cargo del proyecto por 4 meses. Fuente:

Elaboración propia.

La implementación de MongoDB como motor de persistencia permitió un beneficio importante en el tiempo de desarrollo y ejecución de tareas, debido a la amplia gama de métodos y funcionalidades que ofrece en comparación con Apache Hbase. MongoDB proporciona una API más flexible y con menor curva de aprendizaje, debido a su manejo de consultas complejas y manejo de documentos JSON; además posee una fácil integración con Spring Boot. Por otro lado, HBase requiere configuraciones más complejas y de alto nivel para lograr funcionalidades iguales o parecidas, incrementando la curva de aprendizaje de este y con ello el tiempo de desarrollo. Esta clara diferencia de tecnologías se traduce en reducción considerable en tiempos de desarrollo e implementación, depuración de código y aprendizaje de la herramienta, impactando de manera positiva en la eficiencia del equipo de desarrollo.

En el contexto del costo-beneficio, la ganancia de tiempo obtenida con MongoDB toma un rol importante en el impacto de nuevos desarrollos a futuro, tomando como pilar fundamental el uso de este nuevo motor de base de datos. Como resultado, el equipo de desarrollo puede destinar menos horas a tareas complejas o de mantenimientos largos, por tareas mucho más sencillas y con mantenimientos cortos; esto impacta en el valor agregado del equipo de desarrollo y por consiguiente un valor fundamental en la entidad bancaria, lo que refuerza la viabilidad económica del proyecto y justifica la inversión realizada en la migración tecnológica.

3.3.2. INTERPRETACIÓN DEL VAN Y DEL TIR

Los indicadores financieros también reflejan el impacto de la reducción de tiempos de respuesta. Al acortar los ciclos de ejecución de consultas y despliegues, se aumenta la productividad y se optimiza el flujo de trabajo, lo que genera un retorno más rápido de la inversión.

Se simularon escenarios optimista, pesimista y realista considerando la mejora en tiempos de respuesta: en el escenario optimista, los microservicios responden casi instantáneamente a todas las consultas; en el pesimista, se contemplan posibles retrasos por ajustes iniciales; y en el realista, se estima una mejora sustancial, pero con ciertos márgenes de tolerancia. Los gráficos de flujo de caja proyectado muestran que la inversión se recupera más rápido gracias a la eficiencia operativa, reflejada en la reducción de tiempos de respuesta y la disminución de costos asociados a soporte y correcciones.

El análisis económico del proyecto considera como beneficio principal la mejora de productividad obtenida tras la migración de la capa de persistencia desde Apache HBase hacia MongoDB. La inversión inicial, equivalente a USD 100,000 correspondiente a cuatro meses de ejecución del proyecto, permitió que las tareas posteriores se realicen aproximadamente un 20% más rápido. Esta mejora fue monetizada considerando el costo anual del equipo de desarrollo, generando flujos de ahorro equivalentes a USD 60,000 anuales. A continuación, en la Tabla 14 podemos visualizar el flujo de caja anual donde se ven los resultados de ganancias en términos monetarios.

Tabla 14*Flujo de caja anual*

Año	Flujo de Caja
0	- 100,000 USD
1	60,000 USD
2	60,000 USD
3	60,000 USD
4	60,000 USD
5	60,000 USD

Nota. Flujos de caja anuales. Fuente: Elaboración propia.

En la Tabla 13 se da un detalle del flujo monetario por cada año, en el primer año se puede visualizar claramente que en el año 0 sale como egreso 100,000 USD debido a la implementación de la migración del motor de base de datos, los demás años son ganancia neta, el cálculo se hizo directamente tomando en cuenta el valor de 25,000 USD como un equipo base de desarrollo que anualmente siempre genera 300,000 USD, es decir, la cuota mensual (25,000 USD) multiplicado por la cantidad de meses (12), entonces teniendo como base las ganancias habituales, cuando se obtiene una productividad del 20% se obtiene 60,000 USD por año, que es lo que está expuesto en la Tabla 14.

En la Tabla 15 podemos visualizar los costos operativos en esta migración de motor de base de datos, que vemos que es favorable para nosotros en unos 20,000 USD que posteriormente se irá adicionando al flujo de caja explicado en la Tabla 14.

Tabla 15*Costos anuales por cada motor de base de datos*

Motor de base de datos	Gastos anuales
Apache Hbase	35,000 USD
MongoDB	15,000 USD

Nota. Flujos de caja anuales. Fuente: Elaboración propia.

En la Tabla 16 visualizamos como calculamos el flujo real para posteriormente sacar el resultado real del VAN, ahora teniendo en cuenta que el flujo de caja anual que teníamos previamente en la Tabla 14 le sumamos la ganancia que existe por usar MongoDB en vez de Apache HBase visualizado en la Tabla 15.

Tabla 16

Flujo de caja anual actualizado para el cálculo de la VAN

Año	Flujo de Caja	Factor de descuento	Flujo actualizado
0	- 100,000 USD	1.000	- 100,000 USD
1	80,000 USD	0.893	71,440 USD
2	80,000 USD	0.797	63,760 USD
3	80,000 USD	0.712	56,960 USD
4	80,000 USD	0.636	50,880 USD
5	80,000 USD	0.567	45,360 USD

Nota. Flujos de caja anuales. Fuente: Elaboración propia.

Ecuación 1

Cálculo del VAN

$$VAN = \sum_{t=0}^n \frac{F_t}{(1 + r)^t}$$

Nota. Fuente: Elaboración propia.

Ecuación 2

Cálculo de la TIR

$$0 = \sum_{t=0}^n \frac{F_t}{(1 + TIR)^t}$$

Nota. Fuente: Elaboración propia.

El Valor Actual Neto fue determinado aplicando la Ecuación 1, considerando los flujos de caja proyectados y una tasa de descuento del 12%. Asimismo, la Tasa Interna de Retorno fue obtenida resolviendo la Ecuación 2, la cual iguala a cero el valor de los flujos de caja del proyecto.

Los resultados del análisis financiero muestran un VAN positivo de USD 188,400 y una TIR aproximada de 75%, valores significativamente superiores a la tasa de descuento establecida del 12%. Estos indicadores demuestran que el proyecto es económicamente viable y que la optimización del tiempo y la productividad justifican plenamente la inversión realizada.

3.3.3. BENEFICIOS DEL PROYECTO

Los beneficios del proyecto se clasifican en tangibles e intangibles. Entre los beneficios tangibles se destacan la reducción de tiempos de respuesta en consultas, el ahorro en licencias y soporte, y la disminución de errores operativos. Esto impacta directamente en la eficiencia del sistema y en la satisfacción del usuario interno, al permitir que los microservicios funcionen de manera más rápida y confiable.

En cuanto a los beneficios intangibles, se incluye la mayor satisfacción del equipo de trabajo, que adquirió nuevas competencias en MongoDB, Spring Boot y arquitecturas de microservicios. Asimismo, se fortaleció la capacidad de respuesta de la organización frente a cambios y nuevas demandas, y se promovió una cultura de buenas prácticas en desarrollo, despliegue y mantenimiento de sistemas. Estos factores contribuyen a la sostenibilidad del proyecto y al fortalecimiento del capital humano dentro de la empresa.

La adopción de MongoDB como tecnología de persistencia, en conjunto con el uso del framework Spring Data MongoDB, permitió simplificar de manera significativa la curva de aprendizaje para los desarrolladores. Spring Data MongoDB ofrece una abstracción clara y estandarizada para el acceso a datos, mediante repositorios, consultas derivadas y mapeo automático de documentos, lo que reduce la necesidad de implementar lógica de persistencia compleja. Esta integración elimina gran parte de la configuración manual y del código de bajo nivel que era necesario al trabajar con Apache HBase, facilitando que nuevos desarrolladores se adapten rápidamente a la capa de persistencia y comprendan el funcionamiento del sistema en menor tiempo.

Desde una perspectiva operativa y económica, la reducción o eliminación de la curva de aprendizaje se traduce en una disminución de los tiempos improductivos

asociados a la capacitación y al acompañamiento técnico. Al integrarse rápidamente al flujo de trabajo, los nuevos desarrolladores pueden aportar valor desde etapas tempranas del proyecto, reduciendo la necesidad de formación prolongada y minimizando riesgos de retraso en la entrega de funcionalidades. Este beneficio, aunque de naturaleza intangible, tiene un impacto directo en los costos del proyecto y en la sostenibilidad de la arquitectura de microservicios, ya que facilita la escalabilidad del equipo, optimiza los tiempos de desarrollo futuro y refuerza la viabilidad de la inversión tecnológica realizada.

En términos de adopción en el mercado tecnológico actual, MongoDB es una de las bases de datos NoSQL más utilizadas y demandadas por la comunidad de desarrolladores, lo que tiene implicaciones directas sobre los costos asociados al talento y la productividad. Según encuestas globales de desarrolladores, más del 25 % de los profesionales han trabajado con MongoDB recientemente, posicionándolo como la base de datos NoSQL más popular entre desarrolladores generales, mientras que tecnologías como HBase no aparecen con una adopción comparable en estos listados de uso general. (Mobile World Capital, 2024)

Además, evaluaciones de participación de mercado estiman que MongoDB posee aproximadamente 45.84 % de cuota en el segmento de bases de datos NoSQL, mientras que HBase representa cerca de 3.98 %, indicando una diferencia sustancial en presencia y uso en entornos productivos. Esta diferencia se traduce en una mayor oferta de desarrolladores con experiencia en MongoDB y una comunidad técnica más sólida, lo que reduce costos de capacitación, mejora la disponibilidad de perfiles técnicos y minimiza los riesgos de dependencia de especialistas escasos, aspectos que impactan positivamente el análisis costo-beneficio del proyecto. (6sense, 2025)

La mayor adopción de MongoDB en el mercado también influye en la disminución de costos operativos a largo plazo y en la sostenibilidad de la arquitectura implementada. Dado que MongoDB es más frecuente en proyectos y tiene mayor presencia en encuestas de popularidad, es más probable encontrar talento disponible sin necesidad de inversiones significativas en entrenamiento especializado, lo que se traduce en un retorno sobre la inversión más rápido y mayor eficiencia en la ejecución de tareas relacionadas con la persistencia de datos. Este escenario contrasta con HBase, cuya menor visibilidad y cuota de uso sugieren una comunidad más reducida y menor disponibilidad de recursos capacitados, lo que puede incrementar tiempos de incorporación y gastos adicionales. Incluir esta diferencia de mercado en el análisis costo-beneficio fortalece la

justificación económica de la migración, ya que los beneficios no solo se encuentran en mejoras técnicas y de productividad, sino también en la optimización de los costos asociados a talento humano y continuidad operativa.

CAPÍTULO IV - REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

La migración de la capa de persistencia de microservicios de Apache HBase a MongoDB dentro de una arquitectura basada en Azure no solo implicó un cambio tecnológico, sino también una oportunidad para evaluar de manera crítica el impacto de esta transición en el equipo de desarrollo, los procesos y la operación del sistema. Esta reflexión permite identificar los aprendizajes obtenidos, los retos enfrentados, así como las mejoras implementadas y las oportunidades para optimizar la eficiencia y sostenibilidad del proyecto. Asimismo, proporciona una visión integral de cómo la elección de tecnologías y la adopción de buenas prácticas influyen directamente en la productividad, el costo y la escalabilidad del sistema, permitiendo extraer conclusiones útiles para proyectos futuros.

A continuación, se abordan los cuatro pilares fundamentales de la experiencia: limitantes de la tecnología, posibilidades de mejora, crecimiento y adaptabilidad del proceso, los cuales reflejan tanto los desafíos como los logros obtenidos durante la implementación de la migración.

- Durante la migración de la capa de persistencia de Apache HBase a MongoDB en Azure, se identificaron varias limitantes inherentes a la tecnología y a la infraestructura utilizada. Uno de los principales riesgos proviene de la dependencia de la plataforma en la nube, ya que posibles caídas o interrupciones de los servicios de Azure pueden afectar la disponibilidad de los microservicios y la ejecución de procesos críticos. Aunque Azure ofrece mecanismos de alta disponibilidad y redundancia, la experiencia mostró que incidentes imprevistos pueden generar retrasos temporales y requerir procedimientos de recuperación rápida. Otra limitante se relaciona con la gestión de índices y optimización de consultas; si bien MongoDB facilita la creación de índices, un diseño inadecuado puede impactar negativamente en el rendimiento cuando se manejan grandes volúmenes de datos. Finalmente, se identificó que la integración de microservicios en entornos distribuidos implica complejidad adicional en la gestión de dependencias, monitoreo y orquestación, especialmente durante fases iniciales de adopción.
- La experiencia permitió también visualizar áreas claras de mejora, especialmente en el uso de índices en MongoDB para optimizar el rendimiento de consultas y operaciones de agregación. La correcta definición de índices compuestos y la

elección de campos estratégicos puede reducir significativamente el tiempo de respuesta de las consultas, disminuir la carga sobre los microservicios y mejorar la eficiencia general del sistema. Además, la implementación de estrategias de monitoreo más robustas y alertas tempranas para posibles caídas de servicios en Azure puede minimizar el impacto de incidentes y reforzar la resiliencia de la arquitectura. Otra oportunidad de mejora se encuentra en la estandarización de patrones de consulta y buenas prácticas de modelado de datos, que facilitarán la escalabilidad futura y el mantenimiento del sistema a medida que crece el volumen de información.

- El proyecto contribuyó al crecimiento del equipo y del proceso de desarrollo, tanto desde el punto de vista técnico como organizativo. La migración a MongoDB y el uso de Spring Data MongoDB simplificaron la curva de aprendizaje, lo que permitió integrar nuevos desarrolladores de manera más rápida y eficiente, reduciendo los tiempos improductivos y facilitando la transferencia de conocimiento dentro del equipo. A nivel de proceso, la experiencia fortaleció la capacidad del equipo para trabajar con arquitecturas de microservicios, promoviendo un enfoque de desarrollo modular, colaborativo y escalable. Este crecimiento se refleja también en la adopción de buenas prácticas de desarrollo, pruebas automatizadas y monitoreo continuo, que mejoran la calidad del software y aumentan la confianza en la operación de los servicios.
- Finalmente, la experiencia evidenció que la adaptabilidad del proceso es un factor crítico para la sostenibilidad del proyecto. La arquitectura basada en microservicios, combinada con pipelines de CI/CD, permitió realizar iteraciones rápidas, implementar mejoras continuas y responder a cambios en los requisitos o en la carga de trabajo sin afectar la estabilidad del sistema. La flexibilidad de MongoDB para manejar distintos tipos de datos y consultas, junto con la integración con herramientas de orquestación y monitoreo, favoreció la capacidad de adaptación ante escenarios imprevistos y optimizó la eficiencia operativa. En conjunto, estos factores refuerzan la importancia de seleccionar tecnologías alineadas con la estrategia de negocio y con el perfil del equipo, garantizando que el proceso de desarrollo y mantenimiento sea escalable, resiliente y sostenible a largo plazo.

CAPÍTULO V - CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

En base a los objetivos definidos en el presente proyecto, se describirán sus conclusiones correspondientes:

- En primer lugar, el resultado del Sprint 1 fue crucial para la migración de los demás microservicios, ya que en éste se definen las estrategias para realizar una correcta migración en cuanto a la tecnología se refiere, a partir del Sprint 2 en adelante fue mucho más sencillo los siguientes pasos. Se concluye que la migración de los 17 microservicios desde Apache HBase hacia MongoDB se realizó de manera exitosa, cumpliendo el objetivo de preservar la lógica de negocio existente y evitando impactos negativos en otras áreas del sistema. Esto demuestra que es posible modernizar la capa de persistencia dentro de una arquitectura basada en microservicios sin comprometer la estabilidad funcional ni la continuidad operativa como podemos visualizar al finalizar cada sprint dentro de la sección 3.2.5 IMPLEMENTACIÓN DE LAS ÁREAS, PROCESOS, SISTEMAS Y BUENAS PRÁCTICAS.
- En relación con la productividad del equipo de desarrollo, se concluye que la adopción de MongoDB permitió reducir significativamente la complejidad del acceso a datos, optimizando la productividad del equipo de desarrollo. Dentro del Sprint 1 y en la sección 3.2.5 IMPLEMENTACIÓN DE LAS ÁREAS, PROCESOS, SISTEMAS Y BUENAS PRÁCTICAS, se demostró que el uso de Spring Boot y Spring Data MongoDB simplificó la implementación de repositorios, consultas y operaciones sobre la base de datos, disminuyendo la necesidad de configuraciones complejas y conocimientos altamente especializados. Como resultado, la curva de aprendizaje de la tecnología de persistencia se redujo considerablemente, permitiendo que los desarrolladores dediquen más tiempo a tareas de valor agregado y menos a actividades de soporte o mantenimiento.
- Asimismo, se concluye que la definición de un conjunto de buenas prácticas y soluciones efectivas para la migración del motor de base de datos fue un factor determinante para el éxito del proyecto como lo podemos visualizar al comienzo del Sprint 1 en la Sección 3.2.5 IMPLEMENTACIÓN DE LAS ÁREAS,

PROCESOS, SISTEMAS Y BUENAS PRÁCTICAS. Estas prácticas permitieron estandarizar el proceso de migración en los demás Sprint, facilitando la integración de MongoDB con Spring Boot y asegurando la consistencia en el desarrollo de los microservicios. La documentación generada y las lecciones aprendidas dentro del Sprint 1 constituyen una base sólida para futuras migraciones o ampliaciones del sistema, reduciendo la incertidumbre técnica o de gestión, junto con los costos asociados a nuevos cambios tecnológicos.

RECOMENDACIONES

Finalmente, como parte del análisis posterior a la migración de la capa de persistencia de microservicios, se identificaron algunas áreas de oportunidad que podrían ser abordadas en futuras versiones para mejorar su funcionalidad y adaptabilidad:

- Visibilidad de documentación:
 - Se recomienda implementar un proceso formal de transferencia de conocimiento y documentación técnica, donde las buenas prácticas, decisiones arquitectónicas y lecciones aprendidas de esta migración queden registradas y sean accesibles para equipos futuros. Esto reducirá la dependencia de conocimiento individual y garantizará la continuidad técnica del proyecto ante cambios de personal.
- Unificar el uso de bases de datos NoSQL:
 - Se recomienda establecer como norma técnica que MongoDB sea la tecnología de persistencia por defecto para nuevos microservicios, siempre que los casos de uso involucren datos no estructurados o semiestructurados. Esta decisión permitirá mantener coherencia tecnológica, reducir la fragmentación del conocimiento dentro del equipo y facilitar la incorporación de nuevos desarrolladores, aprovechando la amplia disponibilidad de talento y documentación existente en el mercado.
- Metodología de trabajo alternativo:
 - Durante el proyecto se identificaron algunos aspectos de la metodología del trabajo Scrum que podrían cambiarse para mejorar la efectividad del equipo y así eliminar los tiempos de reunión innecesario y enfocarlos al desarrollo. Es decir, en vez de algunas ceremonias como “dailies” que durante el proyecto se hace muy repetitivo e innecesario, se podría cambiar por una metodología Kanban. Kanban permite una mayor flexibilidad para gestionar prioridades en tiempo real y adaptándose a los objetivos del equipo y el proyecto.

FUENTES DE INFORMACIÓN

- 6sense. (2025). *MongoDB Vs Hbase : In-Depth Comparison*. From <https://6sense.com/tech/nosql-databases/mongodb-vs-hbase>
- Akinepalli, K. (2024). *Microservices with Spring Boot: Simplifying Distributed Systems*. From <https://doi.org/10.36948/ijfmr.2024.v06i05.28930>
- Apache Software Foundation. (2022). *Apache ZooKeeper Documentation*. From <https://zookeeper.apache.org/doc/current/>
- Apache Software Foundation. (2023). *Apache Hadoop Architecture Overview*. From <https://hadoop.apache.org/docs/current/>
- Apache Software Foundation. (2024). *Apache HBase Reference Guide*. From <https://hbase.apache.org/book.html>
- Atlassian, Inc. (2025). *Arquitectura de microservicios*. From <https://www.atlassian.com/es/microservices/microservices-architecture>
- Hassan, M. U. (2025). *A Comprehensive Study of HBase Storage Architecture—A Systematic Literature Review*. From <https://www.mdpi.com/2073-8994/13/1/109>
- International Research Journal of Engineering and Technology. (2024). *Microservices Architectures Using Spring Boot: Embracing Containerization*. From <https://www.irjet.net/archives/V11/i7/IRJET-V11I754.pdf>
- Microsoft Azure. (2025). *Base de datos NoSQL: ¿qué es NoSQL?* From <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-nosql-database>
- Microsoft Azure. (2025). *What is Azure?* From <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure>
- Microsoft Azure, Inc. (2025). *Browse Azure Architectures*. From <https://learn.microsoft.com/en-us/azure/architecture/browse/>
- Microsoft Azure, Inc. (2025). *Estilo de arquitectura de microservicios*. From <https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>
- Mobile World Capital. (2024). *Digital Talent Overview*. From https://mobileworldcapital.com/app/uploads/2024/07/Digital_Talent_Overview_2024_ENG.pdf
- MongoDB Inc. (2025). *JSON and BSON*. From <https://www.mongodb.com/resources/basics/json-and-bson>

- MongoDB Inc. (2025). *MongoDB vs HBase*. From <https://www.mongodb.com/resources/compare/mongodb-hbase>
- Pivotal Software, Inc. (2024). *Spring Boot Reference Documentation*. From <https://docs.spring.io/spring-boot/docs/current/reference/html/>
- Scrum Organization. (2017). *¿Qué es Scrum?* From <https://www.scrum.org/resources/blog/que-es-scrum>
- Scrum Organization. (2020). *The 2020 Scrum Guide Purpose of the Scrum Guide*. From <https://scrumguides.org/scrum-guide.html>
- Spring Framework Documentation. (2025). *Spring Framework official reference*. From <https://docs.spring.io/spring-framework/reference/html/>
- Truică, C.-O. R. (2018). *Performance evaluation for CRUD operations in asynchronously replicated document oriented database*<https://doi.org/10.48550/arXiv.1806.04761>. From <https://doi.org/10.48550/arXiv.1806.04761>
- Vashisht. (2025). *Microservices and Real-Time Processing in Retail IT: A Review...* From <https://doi.org/10.48550/arXiv.2506.09938>
- VMware, Inc. (2024). *Spring Data MongoDB Reference Documentation*. From <https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/>

GLOSARIO

- *APIM*: Servicio de Azure que permite publicar, proteger, monitorear y administrar APIs, actuando como una capa de seguridad y control entre los clientes y los microservicios.
- *Azure Key Vault*: Servicio de Azure que permite almacenar y gestionar de forma segura secretos, claves y certificados, evitando el uso de credenciales sensibles en el código fuente.
- *Azure SQL Database*: Servicio de base de datos relacional administrada en la nube de Azure, basado en SQL Server, que ofrece alta disponibilidad, escalabilidad y mantenimiento automatizado.
- *Bean*: Objeto gestionado por el contenedor de Spring, cuya creación, configuración y ciclo de vida son controlados automáticamente por el framework.
- *Connection*: Interfaz utilizada para establecer una conexión con una base de datos, ya sea HBase o MongoDB, permitiendo ejecutar operaciones sobre los datos.
- *Criteria*: Clase de Spring Data MongoDB utilizada para definir condiciones de filtrado en consultas, permitiendo construir consultas dinámicas de forma programática.
- *Get*: Operación de HBase utilizada para recuperar un registro específico a partir de su clave primaria (row key).
- *Gradle*: Herramienta de automatización de construcción y gestión de dependencias utilizada en proyectos Java, que permite definir configuraciones de compilación de forma flexible mediante scripts.
- *HDFS*: Sistema de almacenamiento distribuido del ecosistema Hadoop, diseñado para almacenar grandes volúmenes de datos de forma escalable y tolerante a fallos. En HBase, HDFS se utiliza como capa subyacente para la persistencia física de los datos.
- *Microsoft Entra ID*: Servicio de gestión de identidades y accesos de Microsoft (antes Azure Active Directory), utilizado para la autenticación y autorización de usuarios y aplicaciones en Azure.
- *org.apache.hadoop:hadoop-auth*: Dependencia del ecosistema Hadoop que proporciona mecanismos de autenticación y seguridad para servicios distribuidos.

- *org.apache.hadoop:hadoop-common*: Biblioteca base de Hadoop que contiene utilidades, configuraciones y componentes comunes utilizados por otros módulos del ecosistema Hadoop.
- *org.apache.hbase:hbase-client*: Dependencia que proporciona las APIs necesarias para que las aplicaciones Java interactúen con HBase, permitiendo operaciones de lectura y escritura.
- *org.apache.hbase:hbase-common*: Módulo que contiene clases y utilidades compartidas utilizadas por los diferentes componentes de HBase.
- *org.apache.hbase:hbase-protocol*: Biblioteca que define los protocolos de comunicación utilizados internamente por HBase para la interacción entre clientes y servidores.
- *org.apache.zookeeper:zookeeper*: Servicio de coordinación distribuida utilizado por HBase para la gestión de configuraciones, sincronización y disponibilidad de los nodos del clúster.
- *POJO*: Objeto Java simple que no depende de frameworks específicos, utilizado para representar modelos de datos de manera ligera y flexible dentro de la aplicación.
- *Query*: Clase de Spring Data MongoDB que representa una consulta hacia la base de datos, construida a partir de criterios y utilizada para recuperar o manipular documentos.
- *spring-boot-starter-data-mongodb*: Starter de Spring Boot que facilita la integración con MongoDB mediante Spring Data, proporcionando repositorios, plantillas y configuraciones automáticas para el acceso a datos.
- *spring-boot-starter-web*: Starter de Spring Boot que proporciona las dependencias necesarias para desarrollar aplicaciones web y servicios REST, incluyendo Spring MVC y un servidor embebido.
- *Starters*: Conjuntos de dependencias preconfiguradas que simplifican la configuración de aplicaciones Spring Boot, permitiendo incorporar funcionalidades específicas sin configuraciones complejas.
- *Table*: Representación lógica de una tabla en HBase que permite realizar operaciones de acceso a los datos almacenados en una familia de columnas.