



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

**Evolución tecnológica de microfrontends mediante la
adopción de módulos federados para la optimización
de la arquitectura frontend de una entidad bancaria**

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Raúl SULCA PALACIOS

ASESOR

Dr. Iván Carlo PETRLIK AZABACHE

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

Sulca, R. (2026). *Evolución tecnológica de microfrontends mediante la adopción de módulos federados para la optimización de la arquitectura frontend de una entidad bancaria*. [Trabajo de suficiencia profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Raúl Sulca Palacios
Tipo de documento de identidad	DNI
Número de documento de identidad	70862949
URL de ORCID	https://orcid.org/0009-0005-1706-4515
Datos de asesor	
Nombres y apellidos	Iván Carlo Petrlik Azabache
Tipo de documento de identidad	DNI
Número de documento de identidad	10140461
URL de ORCID	https://orcid.org/0000-0002-1201-2143
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Armando David Espinoza Robles
Tipo de documento	DNI
Número de documento de identidad	08136325
Miembro del jurado 1	
Nombres y apellidos	Jorge Luis Zavaleta Campos
Tipo de documento	DNI
Número de documento de identidad	09696217
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	2024
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 Software

Acta Virtual de Sustentación
del Trabajo de Suficiencia Profesional

Siendo las 18:02 horas del día 01 de abril del año 2026, se reunieron virtualmente los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Lic. Espinoza Robles Armando David (Presidente), Ing. Zavaleta Campos Jorge Luis (Miembro) y el Dr. Petrlik Azabache Iván Carlo (Miembro Asesor), usando la plataforma Meet (<http://meet.google.com/hot-tmmg-afm>), para la sustentación virtual del Trabajo de Suficiencia Profesional intitulado: **“EVOLUCIÓN TECNOLÓGICA DE MICROFRONTENDS MEDIANTE LA ADOPCIÓN DE MÓDULOS FEDERADOS PARA LA OPTIMIZACIÓN DE LA ARQUITECTURA FRONTEND DE UNA ENTIDAD BANCARIA”**, por el Bachiller **Sulca Palacios Raúl**; para obtener el Título Profesional de Ingeniero de Software.

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 **19 (DIECINUEVE)**.

A continuación el Presidente de Jurados el Lic. Espinoza Robles Armando David, declara al Bachiller **Ingeniero de Software**.

Siendo las 18:56 horas, se levantó la sesión.

Presidente

Lic. Espinoza Robles Armando David

Miembro

Ing. Zavaleta Campos Jorge Luis

Miembro Asesor

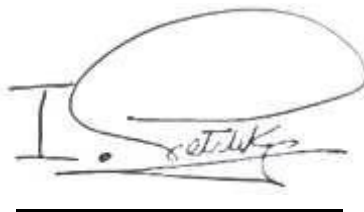
Dr. Petrlik Azabache Iván Carlo

CERTIFICADO DE SIMILITUD

Yo, Iván Carlo Petrlik Azabache en mi condición de asesor acreditado mediante Resolución Decanal N° 001268-2025-D-FISI/UNMSM y OFICIO N° 0027-PTTSP-FISI-2025 del Trabajo de Suficiencia Profesional, cuyo título es Evolución tecnológica de microfrontends mediante la adopción de módulos federados para la optimización de la arquitectura frontend de una entidad bancaria, presentado por el bachiller Sulca Palacios Raúl para optar el Título Profesional de Ingeniero de Software, 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 6% 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 SOFTWARE**.

Lima, 13 de abril de 2026



Firma del Asesor:

Nombres y Apellidos del asesor: Iván Carlo Petrlik Azabache

DNI: 10140461

FICHA CATALOGRÁFICA

EVOLUCIÓN TECNOLÓGICA DE MICROFRONTENDS MEDIANTE LA ADOPCIÓN DE MÓDULOS FEDERADOS PARA LA OPTIMIZACIÓN DE LA ARQUITECTURA FRONTEND DE UNA ENTIDAD BANCARIA

AUTOR: SULCA PALACIOS, RAUL

ASESOR: DR. PETRLIK AZABACHE, IVAN CARLO

LIMA – PERÚ, 2026

TÍTULO PROFESIONAL: Título Profesional de Ingeniero de Software

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 Software

Formato 21 x 29.7 cm

Páginas: xiii, 80

DEDICATORIA

Quiero dedicar el presente trabajo a mi padre Abel, quien siempre ha confiado en mí y me brindó su respaldo incondicional, a mi madre Getrudes, la que siempre me motivó a no rendirme aún en las situaciones imposibles, a mi hermana Rosmery, mi mayor referente de esfuerzo y constancia y a mi hermana Thais, por ser siempre mi mayor defensora aún en los casos indefendibles.

AGREDECIMIENTOS

A mi amigo Anthony, quien siempre tuvo la disposición y el espacio para acompañarme y orientarme a lo largo de mi experiencia profesional.

A Sofía, por la constante motivación y la confianza depositada en mis capacidades.

A mi asesor, Iván, por su permanente predisposición y acompañamiento durante el desarrollo de este trabajo.

Finalmente, a la Universidad Nacional Mayor de San Marcos, que me brindó mucho más de lo que alguna vez pude imaginar.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

**Evolución tecnológica de microfrontends mediante la adopción de módulos
federados para la optimización de la arquitectura frontend de una entidad
bancaria**

Autor: Sulca Palacios, Raúl

Asesor: Ivan Carlo Petrlik Azabache

**Título: Trabajo de Suficiencia Profesional para optar el Título Profesional de
Ingeniero de Software**

Fecha: Enero de 2026

RESUMEN

El presente Trabajo de Suficiencia Profesional tuvo como objetivo optimizar la arquitectura frontend de una entidad bancaria, la cual presentaba limitaciones asociadas al acoplamiento, mantenibilidad y eficiencia en los procesos de despliegue, como parte de un proceso de evolución tecnológica orientado a modernizar sus canales digitales. Para ello, se optimizó y reestructuró la arquitectura microfrontend existente mediante la adopción de módulos federados y la migración tecnológica de Angular 12 a Angular 15. La metodología empleada se fundamentó en la organización del proyecto en etapas consecutivas, tomando como referencia el ciclo de vida del desarrollo de software, lo que permitió abordar de manera estructurada el análisis de la arquitectura existente, el rediseño de la solución frontend, la implementación de aplicaciones tipo shell y remotes, así como la validación posterior a su despliegue. Para el desarrollo de la solución se utilizaron herramientas y tecnologías modernas, entre las que destacan Angular 15 como framework principal, Webpack 5 con Module Federation para la integración dinámica de microfrontends, TypeScript como lenguaje base, Bitbucket Pipelines para la

automatización de los procesos de integración y despliegue continuo, y Microsoft Azure como plataforma de infraestructura y soporte. Como resultados obtenidos, se logró una arquitectura frontend más modular y desacoplada, que permitió el despliegue independiente de los microfrontends, una mejor gestión de dependencias compartidas y una reducción del impacto ante fallos parciales del sistema. La validación técnica post-despliegue evidenció mejoras en métricas relevantes, tales como la disminución de la tasa de errores del frontend, la optimización de los tiempos de carga de las vistas y el incremento en la disponibilidad de los microfrontends, demostrando que la solución implementada cumple con los objetivos planteados y contribuye a una mayor estabilidad, escalabilidad y resiliencia de la plataforma digital.

Palabras claves: evolución tecnológica, microfrontend, módulos federados, arquitectura, entidad bancaria

**MAJOR NATIONAL UNIVERSITY OF SAN MARCOS
FACULTY OF SYSTEM AND INFORMATICS ENGINEERING
PROFESSIONAL SCHOOL OF SOFTWARE ENGINEERING**

**Technological evolution of microfrontends through the adoption of federated
modules for optimizing the frontend architecture of a banking institution**

Author: Sulca Palacios, Raúl

Advisor: Ivan Carlo Petrlik Azabache

**Title: Professional Sufficiency Work to opt for the Professional Title of Software
Engineer**

Date: January de 2026

ABSTRACT

The objective of this Professional Proficiency Project was to optimize the frontend architecture of a banking institution, which had limitations associated with coupling, maintainability, and efficiency in deployment processes, as part of a technological evolution process aimed at modernizing its digital channels. To achieve this, the existing microfrontend architecture was optimized and restructured by adopting federated modules and migrating from Angular 12 to Angular 15. The methodology employed was based on organizing the project into consecutive stages, taking the software development life cycle as a reference, which allowed for a structured approach to analyzing the existing architecture, redesigning the frontend solution, implementing shell and remote applications, and validating them after deployment. Modern tools and technologies were used to develop the solution, including Angular 15 as the main framework, Webpack 5 with Module Federation for dynamic integration of microfrontends, TypeScript as the base language, Bitbucket Pipelines for automation of integration and continuous

deployment processes, and Microsoft Azure as the infrastructure and support platform. The results achieved included a more modular and decoupled frontend architecture, which enabled independent deployment of microfrontends, better management of shared dependencies, and reduced impact in the event of partial system failures. Post-deployment technical validation showed improvements in relevant metrics, such as a decrease in the frontend error rate, optimization of view loading times, and increased availability of microfrontends, demonstrating that the implemented solution meets the objectives set and contributes to greater stability, scalability, and resilience of the digital platform.

Keywords: technological evolution, microfrontend, module federation, architecture, banking institution

ÍNDICE GENERAL

RESUMEN	v
ABSTRACT	vii
ÍNDICE GENERAL	ix
ÍNDICE DE TABLAS	xi
ÍNDICE DE FIGURAS	xiii
INTRODUCCIÓN	1
CAPÍTULO I – TRAYECTORIA PROFESIONAL	3
1.1. PRESENTACIÓN PROFESIONAL	3
1.2. EXPERIENCIA PROFESIONAL	4
1.3. FORMACIÓN ACADÉMICA	6
CAPÍTULO II – CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA .	9
2.1. EMPRESA – ACTIVIDAD QUE REALIZA	9
2.2. VISIÓN	10
2.3. MISIÓN	10
2.4. ORGANIZACIÓN DE LA EMPRESA	10
2.5. ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS	12
2.5.1. ÁREA	12
2.5.2. CARGO	12
2.5.3. FUNCIONES DESEMPEÑADAS	12
2.6. EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN	13
CAPÍTULO III – ACTIVIDADES DESARROLLADAS	14
3.1. SITUACIÓN PROBLEMÁTICA	14
3.1.1. PROBLEMA GENERAL	15
3.1.2. PROBLEMAS ESPECIFICOS	16
3.2. SOLUCIÓN	16
3.2.1. OBJETIVOS	16
3.2.2. ALCANCE	17
3.2.3. ETAPAS Y METODOLOGÍA	19

3.2.4.	FUNDAMENTOS UTILIZADOS	24
3.2.5.	IMPLEMENTACIÓN DE LAS ÁREAS, PROCESOS, SISTEMAS Y BUENAS PRACTICAS.....	35
3.3.	EVALUACIÓN.....	66
3.3.1.	EVALUACIÓN ECONÓMICA / EVALUACIÓN COSTO-BENEFICIO 66	
3.3.2.	INTERPRETACIÓN DEL VAN Y TIR	70
CAPÍTULO IV – REFLEXIÓN CRÍTICA DE LA EXPERIENCIA		72
CAPÍTULO V – CONCLUSIONES Y RECOMENDACIONES		74
5.1.	CONCLUSIONES.....	74
5.2.	RECOMENDACIONES	75
5.3.	FUENTES DE INFORMACIÓN	76
5.4.	GLOSARIO.....	80

ÍNDICE DE TABLAS

Tabla 1 Experiencias laborales	4
Tabla 2 Educación Académica	6
Tabla 3 Cursos realizados.....	7
Tabla 4 Certificaciones	7
Tabla 5 Idiomas estudiados por el autor	8
Tabla 6 Actividades que comprende el alcance del proyecto	17
Tabla 7 Herramientas tecnológicas.....	18
Tabla 8 Etapas de desarrollo.....	20
Tabla 9 Ceremonias de un sprint	22
Tabla 10 Beneficios de la Evolución Tecnológica	24
Tabla 11 Beneficios del uso de angular	26
Tabla 12 Beneficios de la implementación de microfrontends	27
Tabla 13 Beneficios del uso módulos federados	29
Tabla 14 Características de Webpack 5	30
Tabla 15 Beneficios del uso de pipelines	32
Tabla 16 Beneficios de Bitbucket.....	33
Tabla 17 Beneficios de Microsoft Azure.....	34
Tabla 18 Beneficios del uso de Jira	35
Tabla 19 Squad evolución tecnológica	36
Tabla 20 Recursos registrados	39
Tabla 21 Diferencia entre Angular 12 y 15	45
Tabla 22 Listado de dependencias de Angular actualizados	47
Tabla 23 Listado de dependencias que no son propias de Angular actualizados	47
Tabla 24 Descripción de los ambientes de desarrollo	55
Tabla 25 Casos de pruebas testeados.....	61
Tabla 26 Tasa de errores por integración del frontend	64
Tabla 27 Tiempos de carga de vistas.....	64
Tabla 28 Disponibilidad de microfrontends	65
Tabla 29 Costos estimados de los servicios de Azure	66
Tabla 30 Costos estimados de la implementación de la evolución tecnológica	67

Tabla 31 Estimación de ahorro por menor atención de incidencias	68
Tabla 32 Estimación de ahorro por menor tiempo y esfuerzo por despliegue	68
Tabla 33 Estimación de ahorro por menor retrabajo y mayor estabilidad.....	69
Tabla 34 Beneficios anuales	69
Tabla 35 VAN y TIR	70

ÍNDICE DE FIGURAS

Figura 1 Organigrama de la entidad bancaria.....	11
Figura 2 Organigrama de la Vicepresidencia de Operaciones y Tecnología	11
Figura 3 Flujo Scrum de un Sprint	23
Figura 4 Servicios de Azure	34
Figura 5 Cronograma por etapas	37
Figura 6 Diagrama de relación de microfrontends	40
Figura 7 Diagrama de esquema shell-remotes.....	40
Figura 8 Diagrama de arquitectura frontend no optimizado	41
Figura 9 Diagrama de arquitectura frontend optimizado	43
Figura 10 Sección de importación.....	48
Figura 11 Sección de exportación	49
Figura 12 Sección de configuración de dependencias compartidas en un microfront shell.....	50
Figura 13 Sección de configuración de dependencias compartidas en un microfront remote	51
Figura 14 Manifest json, con rutas locales	52
Figura 15 Manifest json, con rutas remotas.....	52
Figura 16 Integración por módulo	53
Figura 17 Integración por routing.....	54
Figura 18 Ambientes de desarrollo	55
Figura 19 Pasos del pipeline.....	56
Figura 20 Estructura del pipeline	57
Figura 21 Versionamiento independiente.....	58
Figura 22 Despliegue independiente	59
Figura 23 Despliegue del manifest.json actualizado	59

INTRODUCCIÓN

Con el avance sostenido de las tecnologías digitales, Internet ha desarrollado un crecimiento desmedido, transformándose en un medio indispensable para realizar trámites, operaciones y servicios de manera remota. Esta tendencia se origina por la necesidad de los usuarios, en apostar por soluciones digitales que le brinden comodidad, rapidez y accesibilidad (Bank for International Settlements, 2020).

En este contexto, muchas entidades se vieron en la obligación de adaptarse a las necesidades de sus clientes. Siendo las entidades bancarias no ajenas a esta tendencia, dando inicio al proceso de digitalización de la mayor parte de sus servicios, adecuando sus sistemas a las nuevas exigencias del mercado (Osei, Cherkasova & Oware, 2023).

Este proceso de digitalización ha conllevado la evolución constante de las arquitecturas tecnológicas empleadas en el sector bancario. Las cuales responden de manera ágil a los nuevos patrones de consumo de los clientes y estrictas normativas nacionales. Como consecuencia, la adopción de tecnologías modernas suele ser algo indispensable para garantizar seguridad y la eficiencia de sus sistemas (Peltonen, Mezzalira & Taibi, 2020).

En este marco, el presente Trabajo de Suficiencia Profesional plantea la evolución tecnológica de los microfrontends mediante la incorporación de módulos federados, orientados a optimizar la arquitectura frontend de una entidad financiera.

Este enfoque busca plantear una arquitectura más segura, escalable, ágil de cara a despliegues continuos y modernizar la plataforma tecnológica, garantizando sostenibilidad y eficiencia a largo plazo.

Asimismo, la adopción de módulos federados permitirá incrementar la escalabilidad y el despliegue continuo, a través de despliegues independientes, lo que reducirá tiempos de entrega y facilitará la integración de nuevas funcionalidades.

Finalmente, la modernización de la arquitectura frontend, migrando a Angular 15, el cual trae consigo una plataforma actualizada, eficiente y sostenible a largo plazo.

Seguidamente, se presenta la estructura del presente trabajo que consta de cinco capítulos:

- En el CAPÍTULO I, presenta de manera detallada la experiencia profesional del autor, resaltando las funciones, responsabilidades y logros alcanzados a lo largo de su experiencia laboral.
- En el CAPÍTULO II, se expone el contexto en el cual se llevó a cabo la experiencia profesional además de su estructura, misión, visión, el entorno tecnológico y organizacional.
- En el CAPÍTULO III, aborda de manera exhaustiva el desarrollo de la solución en el presente trabajo, las situaciones problemáticas, objetivos, metodología de ciclo de vida y los resultados de la solución propuesta.
- En el CAPÍTULO IV, se presenta una reflexión crítica sobre la experiencia desarrollada, destacando los aportes realizados por el autor, las necesidades atendidas y las competencias profesionales requeridas durante el proyecto.
- En el CAPÍTULO V, reúne las conclusiones y recomendaciones derivadas del Trabajo de Suficiencia Profesional, orientadas a consolidar los aprendizajes obtenidos y proponer mejoras futuras.

CAPÍTULO I – TRAYECTORIA PROFESIONAL

1.1. PRESENTACIÓN PROFESIONAL

El autor del presente trabajo tiene el grado de bachiller en Ingeniería de Software en la Universidad Nacional Mayor de San Marcos, cuenta con más de 5 años de experiencia laboral enfocándose en el desarrollo de soluciones web, usando tecnologías como Spring Boot, Angular, PostgreSQL y Oracle.

Actualmente desempeña sus labores en NTT DATA, consultora internacional, cumpliendo el rol de Lead Engineer en el área de frontend. El autor posee conocimiento en arquitectura de software, patrones de diseño y frameworks modernos tales como Angular, NestJS, Spring Boot, entre otros.

Asimismo, lidera la Comunidad Front Architectures, un espacio conformado por más de 500 ingenieros frontend en NTT DATA. Desde este rol impulsa iniciativas de innovación, promueve el intercambio de conocimiento y fortalece las capacidades técnicas del equipo. Además, organiza actividades formativas y guía la aplicación de estándares y buenas prácticas en el desarrollo frontend.

Además, participa en proyectos de evolución tecnológica, migraciones y propuestas técnicas en el área del frontend. Así como en cursos, conferencias y eventos de la comunidad de Angular y Google. Obteniendo múltiples reconocimientos por parte de NTT DATA y el cliente asignado.

A lo largo de su trayectoria profesional, ha podido obtener certificaciones internacionales en instituciones de renombre como Google Cloud, GitHub, SCRUMstudy

y Apollo.

1.2. EXPERIENCIA PROFESIONAL

A continuación, en la Tabla 1, se describe la experiencia profesional del autor, donde podemos resaltar que se desempeña laboralmente en una consultora internacional desde pregrado, consultora donde pudo realizar línea de carrera ascendiendo hasta la actualidad.

Tabla 1

Experiencias laborales

Empresa/Periodo	Descripción de actividades realizadas
NTT DATA (Diciembre 2021 - Actualidad)	Cargo: Lead Engineer Actividades: - Desarrollo de sistemas bancarios usando el framework Angular - Desarrollo de capas de experiencia usando NestJs - Realizar upgrade tecnológicos - Migrar sistemas legacy - Proponer mejoras tecnológicas anuales - Realizar análisis de impacto de mejoras propuestas - Liderar los pases a producción en el área frontend - Ser referente frontend con el cliente asignado - Ser el referente frontend dentro de la empresa - Liderar comité de evaluación de postulantes al proyecto - Liderar comunidad front architectures - Realizar inducción técnica

-
- Ser evaluador de desempeño de ascensos

Logros:

- Realizar la migración de la arquitectura frontend de la extranet de la entidad financiera
- Realizar el upgrade tecnológico a módulos federados
- Liderar la Comunidad Front Architectures de NTT DATA
- Creación de nuevos microfrontends
- Reconocimiento en NTT DATA en la categoría “Compromio y Curiosidad” por el liderazgo de la comunidad frontend
- Obtención de certificaciones internacionales en Google, GitHub, Apollo
- Recibir un ascenso a Lead Engineer

SUNARP

(Noviembre 2020 - Diciembre 2021)

Cargo: Full Stack Developer

Actividades:

- Mesa de ayuda N3
- Soporte sistemas de escritorio
- Soporte sistemas web
- Capacitación de funcionarios en uso de sistemas web
- Desarrollo de sistemas web

Logros:

- Ganador del concurso de “Mejor proyecto”, que se otorga a los nuevos ingresantes de la empresa

PROTIVITI PERU

(Enero 2020 – Febrero 2020)

Cargo: Practicante

Actividades:

- Brindar soporte a las casuísticas del CRM Salesforce
 - Capacitación del programa de certificación Salesforce Developer credentials – Certification
-

1.3. FORMACIÓN ACADÉMICA

En la Tabla 2, se detalla la trayectoria académica del autor que desempeñó a lo largo de los años.

Tabla 2

Educación Académica

Periodo	Educación Académica
2016 - 2022	Grado Académico de Bachiller en Ingeniería de Software Escuela Académico Profesional de Ingeniería de Software Facultad de Ingeniería de Sistemas e Informática Universidad Nacional Mayor de San Marcos
2025 - Actualidad	Programa Escuela para Líderes Universidad ESAN

Los cursos realizados por el autor se muestran en la Tabla 3, donde algunos de ellos se llevaron en sus centros de labores, así como en instituciones externas.

Tabla 3*Cursos realizados*

Institución	Curso
NTT DATA	GenAI Academy
SUNARP	Sistema de Gestión de Seguridad de la Información ISO 27001
Fundación telefónica España	Gestión de Proyectos con Metodologías Ágiles y Enfoques Lean

Las certificaciones logradas por el autor se reseñan en la Tabla 4. Donde se destacan certificados emitidos por instituciones de renombre, como Google Cloud, Microsoft, Apollo Graph, entre otros.

Tabla 4*Certificaciones*

Periodo	Certificación	Institución
2025	API Design and Fundamentals of Google Cloud's Apigee API Platform	Google Cloud
2025	Deploy and Manage Apigee X Skill Badge	Google Cloud
2025	GitHub Actions	Microsoft
2024	GitHub Foundations	Microsoft
2023	Graph Developer - Professional	Apollo Graph
2023	Graph Developer - Associate	Apollo Graph
2020	Scrum Fundamentals Certified	SCRUMstudy

Los estudios de idiomas realizados por el autor se listan en la Tabla 5, las cuales fueron desarrollado en la misma institución donde desarrolló sus estudios de pregrado.

Tabla 5

Idiomas estudiados por el autor

Periodo	Idioma	Nivel	Institución
2016-2017	Inglés	A1 A2	Centro de Idiomas UNMSM
2018-2019	Inglés	B1 B2	Centro de Idiomas UNMSM

CAPÍTULO II – CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA

2.1. EMPRESA – ACTIVIDAD QUE REALIZA

Es una entidad bancaria peruana, que ofrece una amplia gama de productos y servicios financieros, tanto a personas naturales como a empresas. A fin de atender las diversas necesidades económicas de sus clientes.

Con el compromiso de adecuarse en un mercado financiero cada vez más dinámico, la entidad apuesta fuertemente en la digitalización de sus procesos y canales de atención. Con el propósito de proporcionar servicios más eficientes, ágiles, seguros y accesibles, permitiendo que los usuarios tengan acceso desde cualquier lugar y en todo momento.

Además, la entidad bancaria promueve activamente la educación financiera entre todos sus usuarios y población en general, con el objetivo de fomentar la inclusión y el uso responsable de los productos y servicio bancarios. Este enfoque no solo busca fortalecer la relación con sus clientes, sino también aportar al crecimiento económico y al bienestar social del país.

A continuación, se lista los principales servicios que ofrece la entidad bancaria, de los cuales resaltan:

- Cuentas de ahorro
- Cuentas corrientes
- Tarjetas de débito
- Tarjetas de crédito

- Préstamos personales
- Créditos vehiculares
- Créditos hipotecarios
- Financiamiento para pequeñas y medianas empresas
- Seguros
- Canales digitales de banca por internet y aplicativos móviles

La entidad bancaria representa un espacio altamente dinámico y competitivo, donde la innovación tecnológica y la digitalización de procesos son pilares fundamentales para atender los requerimientos del mercado.

2.2. VISIÓN

Ser la mejor entidad bancaria peruana en base a las mejores personas, con compromiso en la excelencia a través de los mejores talentos humanos y una cultura organizacional que fomenta la diversidad, inclusión y desarrollo profesional para generar un impacto positivo en la calidad de vida de los clientes y respaldarlos en cumplir sus sueños.

2.3. MISIÓN

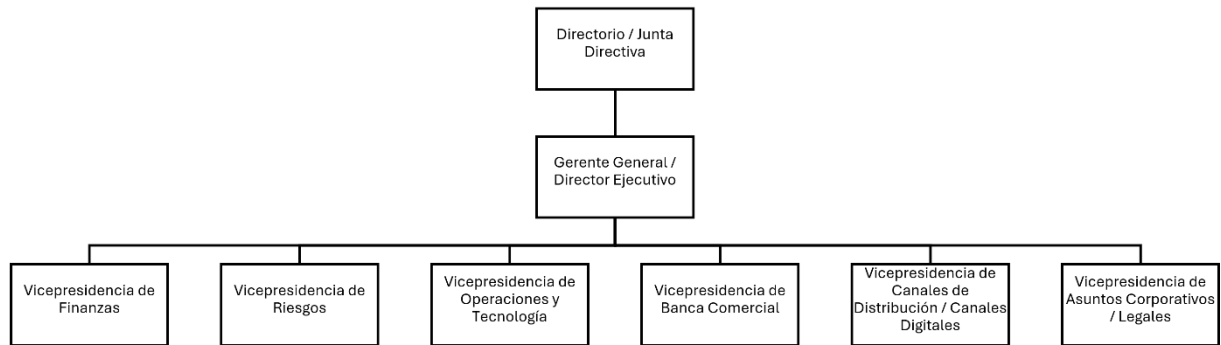
Contribuir con el bienestar de los peruanos, brindando accesos a productos financieros y asesoría profesional mediante servicio ágiles y accesibles que permitan realizar los sueños de sus clientes.

2.4. ORGANIZACIÓN DE LA EMPRESA

En la Figura 1 se muestra el organigrama actual de la entidad bancaria, donde se visualiza las 6 principales vicepresidencias.

Figura 1

Organigrama de la entidad bancaria

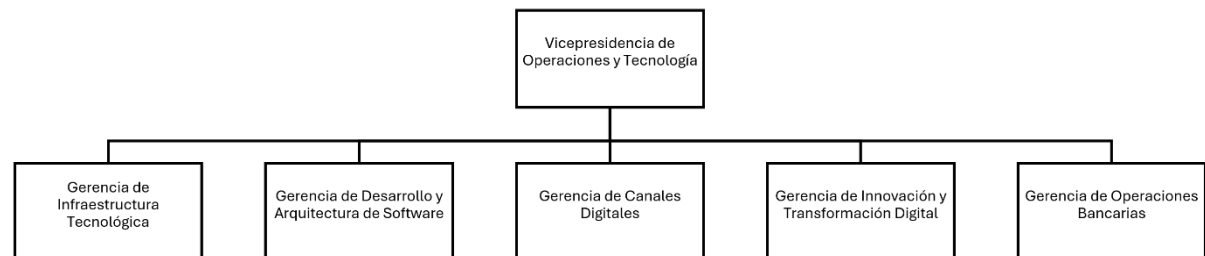


Nota. Fuente: Adaptado de la entidad financiera

La Vicepresidencia de Operaciones y Tecnología, es el área que se encargada de implementar las soluciones tecnológicas aprobadas. Dicha área a su vez consta de un organigrama propio, donde se detalla las 5 gerencias especializadas en la Figura 2.

Figura 2

Organigrama de la Vicepresidencia de Operaciones y Tecnología



Nota. Fuente: Adaptado de la entidad financiera

La Gerencia de Desarrollo y Arquitectura de Software tiene un rol estratégico y técnico debido a que tiene por objetivo asegurar que los sistemas y aplicaciones estén bien diseñados, sean escalables y respondan a las necesidades del negocio. Tiene como principales responsabilidades:

- Definición de la arquitectura tecnológica
- Desarrollo de software bancario
- Innovación y modernización tecnológica
- Gestión de calidad de software

- Soporte al negocio y alineamiento estratégico
- Gestión de equipos y talento

2.5. ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS

2.5.1. ÁREA

Venta Digital, área que engloba canales y servicios que permite comercializar productos financieros a través de medios electrónicos. Brindando comodidad, rapidez y seguridad al cliente. Optimizando procesos y accesos a servicios bancarios.

2.5.2. CARGO

El autor se desempeña como Lead Engineer en el área frontend, asumiendo diversos roles a lo largo del proyecto. Entre ellos destacan los roles de analista, referente frontend, y desarrollador frontend senior, siendo este último rol, el cual ejecutó la evolución tecnológica de la arquitectura frontend.

2.5.3. FUNCIONES DESEMPEÑADAS

El autor ha desempeñado múltiples funciones en los años de experiencia que lleva en la entidad bancarias, de las cuales resaltan:

- Coordinación técnica de todas las actividades del equipo de frontend
- Intermediación entre la entidad bancaria y el equipo proveedor
- Planificación y seguimiento de las tareas programadas
- Definición e implementación de estándares de buenas prácticas y lineamientos técnicos
- Resolución de bloqueos técnicos y soporte
- Participación en la migración tecnológica de la arquitectura frontend
- Desarrollo de nuevos microfrontends

- Capacitación y mentoría a los ingenieros del equipo
- Ser revisor de código
- Liderar los pases a producción del lado frontend
- Participación de los workshops
- Upgrade de módulos legacy
- Presentación de mejorar técnicas anuales

2.6. EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN

Durante su experiencia laboral en el área de Venta Digital, el autor participó en diversas iniciativas asignadas por la entidad bancaria, las cuales se listan a continuación.

- Desarrollo de un sistema que lista las deudas de los clientes y evidencia el riesgo que trae consigo.
- Desarrollo de web components nativos, evitando el uso de librerías externas a las entidades bancarias
- Desarrollo de un sistema extranet, que permite realizar cambio de condiciones a las cotizaciones
- Migración de sistemas legacy
- Mejora de auditoria en los sistemas de la entidad bancaria
- Evolución tecnológica en la intranet
- Definición de lineamientos de estándares

CAPÍTULO III – ACTIVIDADES DESARROLLADAS

3.1. SITUACIÓN PROBLEMÁTICA

A nivel global, la banca digital dejó de ser el futuro, para convertirse en una realidad consolidada. El desafío actual no es iniciar la digitalización de los productos y servicios, sino de mantenerla vigente y en constante evolución. Las exigencias del mercado demandan no solo la accesibilidad en línea, también que estas estén actualizadas, seguras y con un óptimo rendimiento (World Bank, 2024).

Informes internacionales señalan que el uso de servicios digitales ha registrado un crecimiento constante, consolidándose como uno de los principales medios para realizar operaciones bancarias. Este crecimiento ha generado una mayor demanda por servicios accesibles, rápidos y disponibles en entorno digitales. En este sentido, las entidades bancarias se ven forzadas a modernizar de manera recurrente sus sistemas tecnológicos para garantizar la confiabilidad de sus plataformas (Organisation for Economic Co-operation and Development, 2022).

En el Perú, las entidades bancarias han sabido adaptarse a los servicios digitales notablemente. Según el Banco Central de Reserva del Perú (BCRP), los pagos digitales per cápita pasaron de 256 en 2023 a 334 en junio de 2024, reflejando no solo la preferencia de los usuarios por el uso de canales digitales, sino también de la confianza que estos tienen en el uso de dichos canales (Banco Central de Reserva del Perú, 2024).

No obstante, el rápido crecimiento y evolución de los hábitos de consumo digital, exige que los sistemas no solo sean funcionales, sino que garanticen alta disponibilidad, mantenibilidad, resiliencia, interoperabilidad y una experiencia de usuario consistente y

confiable.

En consecuencia, muchas entidades bancarias afrontan el reto de modernizar sus arquitecturas, debido a que estas empiezan a mostrar rezagos tecnológicos frente a estándares internacionales.

Bajo este contexto, la entidad bancaria objeto de estudio, ya contaba con una arquitectura frontend implementando una arquitectura de microfrontends con Angular 12. Siendo esta arquitectura suficiente para el alcance que previamente había sido definido. Con el transcurso del tiempo, se fueron incorporando nuevos y más sofisticados módulos, lo que amplió significativamente su alcance inicial. Sin embargo, el modelo no contaba con una federación real ni con una estrategia eficiente de compartición de dependencias, lo que limitaba los beneficios esperados del enfoque microfrontend.

Con la incorporación progresiva de nuevas funcionalidades, el deterioro arquitectónico se evidenció también en métricas concretas: el tiempo de carga inicial se situaba entre 3000 y 3500 ms, mientras que el build en el pipeline CI/CD aumentó de aproximadamente 6 a 11–12 minutos. El bundle inicial superaba los 7 MB, y se registraban en promedio 20 errores diarios por integración, junto con incidencias asociadas a incompatibilidades entre microfrontends. Asimismo, se identificaron dependencias en versiones obsoletas o fuera de soporte oficial, incrementando riesgos de seguridad y dificultades de mantenimiento.

3.1.1. PROBLEMA GENERAL

La arquitectura frontend de la entidad bancaria, desarrollada en Angular 12, comenzó a mostrar limitaciones que afectaban el rendimiento y aumentaba el tiempo de carga del sistema completo. Los despliegues resultaron ser poco eficientes volviéndose dependientes, reduciendo agilidad del equipo de desarrollo. Además, el uso de una versión fuera del soporte oficial incrementaba los riesgos de seguridad y el rezago tecnológico.

3.1.2. PROBLEMAS ESPECIFICOS

- **Limitaciones de rendimiento en la arquitectura:** Las compilaciones de los microfrontends, cada vez generaban aplicaciones más pesadas generando un mayor tiempo de carga. Afectando indicadores de rapidez y eficiencia.
- **Baja eficiencia en los procesos de despliegues e integración:** Se requería la recompilación integral del sistema ante cambios en un solo microfrontend, debido a una integración estática y a la ausencia de federación real. Esto generaba despliegues acoplados y versionamiento centralizado, limitando la independencia operativa de cada componente.
- **Riesgos de seguridad y rezago tecnológico:** La arquitectura frontend fue desarrollada usando Angular 12, versión que está fuera del soporte oficial, lo cual expone a la arquitectura a distintos tipos de vulnerabilidades y a los parches de seguridad actualizados.

3.2. SOLUCIÓN

3.2.1. OBJETIVOS

3.2.1.1. OBJETIVO GENERAL

Optimizar la arquitectura de microfrontends existente de la entidad bancaria mediante la incorporación de módulos federados y la migración tecnológica de Angular, con el fin de reducir la carga y complejidad de la arquitectura frontend, incrementar la eficiencia de los procesos de despliegue y fortalecer la seguridad y actualizar la base tecnológica.

3.2.1.2. OBJETIVOS ESPECÍFICOS

- **Reducir la carga y complejidad de la arquitectura frontend:** Mediante la adopción de módulos federados que permitirá generar aplicaciones más livianas suprimiendo duplicidad de dependencias y uso de la carga dinámica.

- **Incrementar la eficiencia de los procesos de despliegue:** Habilitando la publicación independiente de microfrontends sin afectar el funcionamiento total del sistema.
- **Fortalecer la seguridad y actualizar la base tecnológica:** Realizando una migración tecnológica de Angular 12 a Angular 15, siendo esta, una versión que cuenta con soporte oficial y parches de seguridad además de gozar funcionalidades más recientes.

3.2.2. ALCANCE

3.2.2.1. ALCANCE DEL PROYECTO

El alcance del presente estudio se limita a comprender la optimización de la arquitectura frontend de una entidad bancaria mediante la adopción de módulos federados y la migración tecnológica de Angular 12 a Angular 15.

A continuación, en la Tabla 6, listamos una tabla de actividades que comprenden el alcance del proyecto:

Tabla 6

Actividades que comprende el alcance del proyecto

N°	Actividad
1	Registrar la totalidad de microfrontends participantes del upgrade
2	Realizar el upgrade de Angular 12 a Angular 15
3	Adoptar la configuración de módulos federados en todos los microfrontends
4	Configurar los pipelines de despliegue de cada microfront
5	Rediseñar la arquitectura frontend

3.2.2.2. *ALCANCE TÉCNICO*

El proyecto se desarrolló utilizando Angular 15 como base del frontend, incorporando módulos federados a través del Module Federation Plugin de Webpack 5 bajo una arquitectura de tipo shell y remote. El desarrollo se apoyó en TypeScript como lenguaje principal y en RxJS para la gestión de flujos reactivos. Además, la automatización de la integración y el despliegue se realizó mediante Bitbucket Pipelines, mientras que la infraestructura se ejecutó sobre la plataforma de computación en la nube Microsoft Azure

A continuación, en la Tabla 7, se listan las herramientas tecnológicas que se emplearon para el desarrollo de la solución propuesta.

Tabla 7

Herramientas tecnológicas

Herramientas	Tecnología
Framework	Angular 15
Módulos federados	Module Federation Plugin de Webpack 5
Arquitectura frontend	Aplicación shell y remote
Lenguajes y librerías de soporte	TypeScript como lenguaje principal RxJS para programación reactiva
Integración y despliegue continuo CI/CD	Bitbucket Pipelines
Infraestructura en la nube	Microsoft Azure

3.2.2.3. *ALCANCE ORGANIZACIONAL*

La evolución tecnológica de la arquitectura de la entidad bancaria está a cargo de la gerencia de Desarrollo y Arquitectura de Software, quienes se encargan de la definición arquitectónica y del desarrollo de soluciones de software que soportan los canales digitales de la entidad bancaria. Esta gerencia se organiza en áreas especializadas que permiten una adecuada coordinación entre arquitectura, desarrollo y aseguramiento de la calidad.

3.2.2.4. *ALCANCE GEOGRÁFICO*

El alcance geográfico del proyecto se limita al ámbito nacional, dado que los desarrollos realizados están orientados exclusivamente a atender las necesidades del mercado peruano. Las funcionalidades implementadas, así como los servicios digitales de la intranet y extranet, fueron diseñados para operar dentro del contexto normativo, operativo y comercial del país.

3.2.3. **ETAPAS Y METODOLOGÍA**

3.2.3.1. *Etapas*

El desarrollo de la evolución tecnológica se organizó en etapas consecutivas, tomando como referencia el ciclo de vida del desarrollo de software (SDLC) y los modelos iterativos e incrementales utilizados en la ingeniería de software. Este enfoque permitió estructurar el proyecto desde la planificación hasta la implementación, pruebas, despliegue y validación de la solución, de acuerdo con el contexto y alcance del trabajo (Sommerville, 2016).

Bajo este enfoque se dividió en 7 etapas, permitiendo garantizar la correcta transición tecnológica en la arquitectura frontend. Las cuales describimos en la Tabla 8.

Tabla 8*Etapas de desarrollo*

Etapa	Aplicación
Levantamiento y planificación de los microfrontends involucrados	- Registro y análisis de los microfrontends que participarían en la nueva arquitectura - Definición del alcance de cada microfront, dependencias y relación con la aplicación host o remota
Adecuación de la arquitectura existente para la migración	- Definir el esquema shell-remote bajo Modulos federados - Actualización de librerías y dependencias necesarias - Ajuste de la arquitectura previa considerando Angular 12 como framework principal
Actualización tecnológica y configuración de microfrontends	- Migración de Angular 12 a Angular 15 - Revisión de compatibilidad de dependencias - Configuración de Webpack 5 y Module Federation Plugin
Desarrollo e integración de microfrontends	- Definición de módulos compartidos - Implementación progresiva de cada microfrontend bajo el enfoque federado - Integración de microfronts remotes con microfront shell - Routing de microfront
Actualización de pipelines y despliegue en la nube	- Actualización de los pipelines incluyendo el nuevo esquema del build y versionamiento - Automatización de releases en el entorno de Microsoft Azure
Pruebas funcionales e integración continua	- Verificación de la carga dinámica en tiempo de ejecución - Revisión del correcto funcionamiento de la

arquitectura propuesta

Monitoreo y validación post-
despliegue

- Seguimiento del comportamiento de la aplicación en entornos reales
 - Revisión de métricas de rendimiento
-

3.2.3.2. Metodología

Para el desarrollo de la propuesta, se utilizó un marco de trabajo ágil, siguiendo las directrices y políticas establecidas por la entidad bancaria, el cual se basa en emplear SCRUM, un marco de trabajo ágil ampliamente reconocido por su capacidad de adaptación, entregas iterativas y mejora continua.

Scrum

Corresponde a un marco de trabajo ágil que guía los procesos de desarrollo de software, se basa en la ejecución de iteraciones cortas denominadas sprints, con el objetivo de entregar incrementos funcionales de manera frecuente y adaptarse a los cambios en los requerimientos, lo que ha sido señalado como un factor clave para la flexibilidad y efectividad (Agh, García, Piattini & Ramsin, 2020).

Sprint

De acuerdo con Schwaber y Sutherland (2020), “Los *sprints* constituyen el eje central de Scrum, ya que representan el periodo en el cual las ideas se transforman en valor. Se trata de eventos de duración fija, de un mes o menos, diseñados para mantener la consistencia en el proceso de desarrollo. Cada *sprint* se inicia de manera inmediata tras la finalización del anterior, garantizando la continuidad del trabajo.”

Historia de Usuario

Representa de manera simple y clara un requerimiento desde la perspectiva del usuario. Cada HU describe qué quiere lograr el usuario, para qué lo necesita y cuál es el valor que aporta al producto (Bass, Clements & Kazman, 2021).

Artefactos de Scrum

Representan los componentes clave que aseguran la transparencia y facilitan la inspección y adaptación durante el desarrollo de un proyecto. Según Schwaber y Sutherland (2020), los artefactos de Scrum tienen como finalidad “garantizar la transparencia de la información clave, de modo que todos los involucrados tengan el mismo entendimiento del producto y del trabajo a realizar”

Ceremonia de un Sprint

De acuerdo con Verheyen (2024), las ceremonias que se desarrollan durante un Sprint se describen en la Tabla 9.

Tabla 9
Ceremonias de un sprint

Ceremonia	Descripción
Sprint planning	Reunión inicial donde el equipo establece el objetivo de este y selecciona los elementos del <i>Product Backlog</i> que serán abordados en el ciclo de trabajo.
Daily meeting	Reunión diaria de corta duración (máximo 15 minutos), donde los integrantes revisan el progreso, identifican impedimentos y ajustan el plan de trabajo para el día.
Sprint Review	Reunión donde, el equipo presenta el incremento de producto a los <i>stakeholders</i> , quienes brindan retroalimentación para asegurar que el resultado esté alineado con las necesidades del cliente.
Sprint Retrospective	Orientada a la mejora continua. El equipo analiza qué funcionó bien, qué aspectos deben corregirse y establece acciones de mejora. para los siguientes sprints.

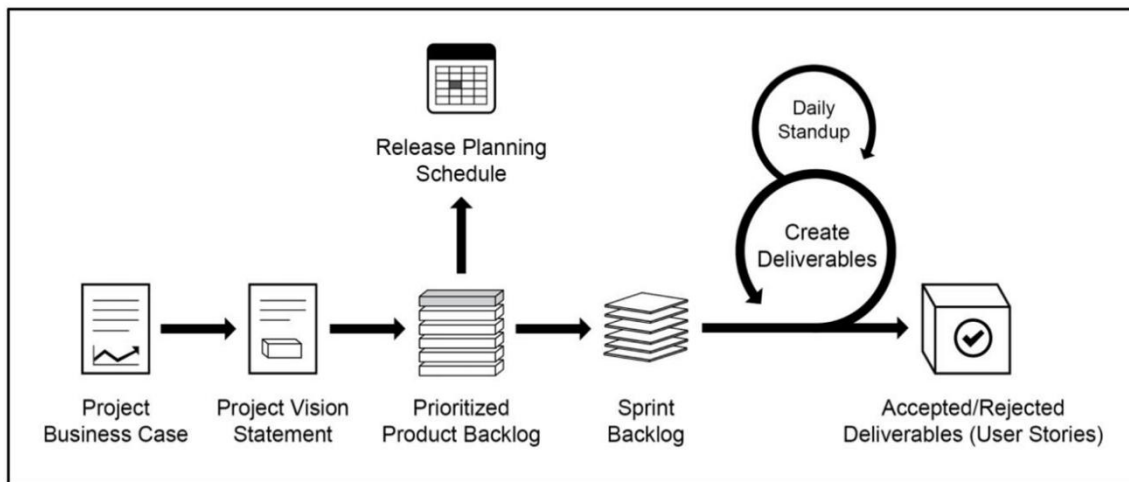
El proceso inicia con el análisis de los casos de negocio, posteriormente se elaboran las Historias de Usuario que conforman el Product Backlog.

Posteriormente, en la etapa de planificación, se seleccionan las tareas que se desarrollarán en el sprint, lo que da origen al Sprint Backlog. Durante la ejecución del sprint no solo se lleva a cabo el desarrollo de dichas tareas, sino que también se realizan eventos clave como las reuniones diarias, que permiten el seguimiento continuo.

Finalmente, al concluir el ciclo, se entrega el incremento correspondiente a las historias de usuario previamente definidas. A continuación, en la Figura 3 se muestra gráficamente el Flujo Scrum de un Sprint.

Figura 3

Flujo Scrum de un Sprint



Nota. Fuente: SCRUMstudy™ (2022)

Para la integración y el despliegue continuo, se usó el enfoque DEVOPS, con la finalidad de automatizar y optimizar el ciclo de vida del software. Para así enfocarse en el desarrollo y aprovechar esfuerzos.

Según Bass, Weber y Zhu (2021), el enfoque DevOps integra desarrollo y operaciones para optimizar el ciclo de vida del software, que comprende desde el

desarrollo hasta el despliegue y la operación en producción. Permitiendo despliegues continuos, confiables y controladas.

3.2.4. FUNDAMENTOS UTILIZADOS

Evolución tecnológica

La evolución tecnológica responde a la necesidad de adaptarse a entornos dinámicos, caracterizados por constantes innovaciones y nuevas demandas de los usuarios. Según Li, Liang, Soliman y Avgeriou (2021), el avance tecnológico es indispensable para evitar la obsolescencia de los sistemas y la erosión arquitectónica, es decir, la pérdida de calidad estructural en el software cuando no se actualiza oportunamente.

A continuación, en la Tabla 10 se listará algunos beneficios que trae consigo la implementación de Evolución tecnológica en nuestros sistemas:

Tabla 10

Beneficios de la Evolución Tecnológica

Nº	Beneficio
1	Proceso continuo de cambio y modernización en herramientas, arquitecturas, metodologías y sistemas
2	Favorece la capacidad de adaptación de las organizaciones frente a entornos dinámicos y exigencias del mercado
3	Reduce la obsolescencia tecnológica y evita que los sistemas queden desfasados
4	Favorece la adopción de nuevas arquitecturas más modulares, escalables y mantenibles
5	Impulsa mejoras en seguridad, rendimiento y experiencia de usuario
6	Asegura la competitividad de empresas y sectores frente al avance constante de la tecnología

Migración tecnológica

La migración tecnológica implica trasladar sistemas, frameworks o componentes de software desde versiones antiguas o enfoques legados a tecnologías más modernas, con el fin de mejorar rendimiento, compatibilidad, seguridad y mantenibilidad, garantizando la evolución y sostenibilidad a largo plazo (Razavian, Tamburri, & Lago, 2020).

En el contexto de Angular, por ejemplo, se han desarrollado schematics que automatizan muchos de los pasos necesarios para actualizar a versiones recientes, lo que facilita adoptar nuevas capacidades del framework sin comprometer la estabilidad del código previo (Modernize Your Angular App with Migration Schematics, 2022).

Frontend

Se refiere a la parte de la aplicación con la que interactúan directamente los usuarios: su interfaz, presentación visual y comportamiento en el cliente. Engloba tecnologías, frameworks, librerías, estilos y componentes UI que permiten renderizar vistas, gestionar la interacción del usuario y comunicarse con la capa de backend (Thallapally, 2023).

Según Jørgensen (2024), el frontend se entiende como la parte del sistema orientada a representar los requisitos del usuario en soluciones visibles y funcionales. Además, indica que la fase frontend no solo implica la interfaz visual de un sistema, sino también actividades de análisis y planificación que son pilares que garantizan el éxito del producto final (Jørgensen, 2024).

Angular

Según Emmanni (2024), indica que Angular se distingue como un framework de desarrollo frontend completo que permite la construcción de aplicaciones empresariales escalables y mantenibles. Posee una arquitectura modular, con gestión de estado y soporta patrones recomendados de diseño que ayudan a estructurar proyectos extensos que requieren alta fiabilidad y consistencia (Emmanni, 2024).

Angular nos permite desarrollar aplicaciones web escalables, mantenibles y seguras, reduciendo la complejidad en sistemas de gran tamaño además de permitir el trabajo colaborativo entre los equipos de desarrollo (Freeman y Shreve 2023).

Asimismo, en la Tabla 11 se listará algunos de sus principales beneficios en su uso.

Tabla 11

Beneficios del uso de angular

Nº	Beneficio
1	Framework integral que incluye enrutamiento, formularios, servicios, comunicación HTTP y pruebas de manera nativa.
2	Arquitectura basada en componentes que facilita el modularidad, reutilización de código y mantenimiento.
3	Uso de TypeScript, que permite tipado estático, orientación a objetos y detección temprana de errores.
4	Sistema de inyección de dependencias que promueve la separación de responsabilidades y escalabilidad.
5	Actualizaciones frecuentes que incluyen mejoras de rendimiento, seguridad y compatibilidad con estándares modernos.
6	Amplia comunidad y soporte con abundante documentación, tutoriales y librerías disponibles.

Microfrontend

El concepto de microfrontends surge como una extensión de los principios de la arquitectura de microservicios aplicados a la capa frontend, con el propósito de dividir aplicaciones web complejas en unidades más pequeñas y autónomas. Diversos estudios señalan que esta aproximación responde a la necesidad de mejorar la escalabilidad, mantenibilidad y organización de aplicaciones frontend de gran tamaño (Peltonen, Mezzalira & Taibi, 2020).

Los microfrontends son una aproximación arquitectónica que representa una extensión de los principios de los microservicios al desarrollo frontend, dividiendo una aplicación en módulos independientes que pueden ser desarrollados, probados y desplegados por separado. Esto nos permite la escalabilidad organizacional al facilitar que diversos equipos trabajen de forma simultánea sin interferir en otras partes de la interfaz (Kurian & Kulkarni, 2024).

Cada parte, denominada microfrontend, puede ser desarrollada, desplegada y mantenida de forma autónoma, lo que permite que distintos equipos trabajen en paralelo sin generar dependencias demasiado fuertes. Esta estrategia se inspira en los principios de los microservicios aplicados al backend para mejorar modularidad y escalabilidad (Antunes, Lima, Pereira, Taibi & Kalinowski, 2024). Adicionalmente los respectivos autores complementan los microfronts en las arquitecturas, tal como se ve en la Tabla 12 mostrando los siguientes beneficios:

Tabla 12
Beneficios de la implementación de microfrontends

Atributo de calidad	Descripción
Modularidad	División del frontend en partes pequeñas e independientes que facilitan la mantenibilidad.
Escalabilidad organizacional	Distintos equipos pueden trabajar en paralelo en diferentes módulos sin bloquearse entre sí.
Despliegue independiente	Cada microfrontend puede actualizarse o publicarse sin necesidad de recompilar todo el sistema.
Reutilización de componentes	Los módulos expuestos pueden integrarse en otros sistemas con facilidad.
Flexibilidad tecnológica	Diferentes microfrontends pueden usar versiones distintas del mismo framework o incluso frameworks diferentes.
Reducción de riesgos	Los cambios se aíslan en un microfrontend, disminuyendo el impacto de errores en toda la aplicación.
Entrega continua más ágil	Facilita la incorporación rápida de nuevas funcionalidades sin afectar al sistema completo.

Mejora en la experiencia de desarrollo	Equipos más autónomos, con control sobre su propio ciclo de vida y decisiones técnicas.
Optimización del mantenimiento	Actualizaciones localizadas que reducen la complejidad de la gestión del frontend.

Arquitectura frontend

La arquitectura frontend, hace referencia a la organización estructural de componentes, flujos de datos y patrones de diseño en aplicaciones web modernas, con el objetivo de asegurar escalabilidad, mantenibilidad y seguridad durante el ciclo de vida del software. Este enfoque tiene en cuenta principios de diseño, gestión de estado, modularidad y separación de responsabilidades para lograr aplicaciones robustas que puedan crecer sin comprometer la calidad ni la experiencia del usuario (Tkachenko et al., 2025).

Además, la arquitectura frontend integra buenas prácticas como la adecuada separación de responsabilidades, así como la reutilización de componentes y la incorporación de patrones de diseño modernos. De acuerdo con Hangai, Corte Real, Ferreira, Rosa y Goldman (2025), estos principios no solo reducen la complejidad técnica, sino que también permiten mejorar la escalabilidad y facilitar los despliegues continuos, aspectos esenciales en la construcción de sistemas digitales sostenibles en el tiempo.

Módulos federados

Los módulos federados permiten que distintas partes de una aplicación frontend se construyan y desplieguen de forma independiente, compartiendo código y dependencias en tiempo de ejecución sin necesidad de recompilar todo el sistema. Mediante Webpack Module Federation, las aplicaciones pueden cargar dinámicamente módulos remotos, reduciendo el tamaño de los paquetes iniciales y mejorando el rendimiento, brindando autonomía a los equipos de desarrollo (Mane et al., 2024).

Asimismo, algunos de los beneficios más resaltantes, respecto al uso de los módulos federados se listan en la Tabla 13.

Tabla 13*Beneficios del uso módulos federados*

Nº	Beneficio
1	Permiten cargar módulos en tiempo de ejecución, sin requerir la recompilación completa de la aplicación
2	Habilitan la exposición de componentes, librerías o aplicaciones completas para ser consumidas por otros proyectos
3	Permiten que equipos trabajen en despliegues independientes sin afectar a toda la aplicación
4	Reducen la duplicación de código y tamaño de los paquetes gracias al uso compartido de dependencias
5	Mejoran la escalabilidad y mantenibilidad de las aplicaciones frontend

Esquema shell-remote

Según Peltonen, Mezzalira y Taibi (2020), este tipo de arquitectura permite dividir responsabilidades de forma clara, facilitando que equipos diferentes trabajen de manera simultánea sin generar dependencias críticas entre sí. El Shell cumple la función de host y es responsable de orquestar la integración de los distintos remotos, asegurando la cohesión del sistema y el control de aspectos globales como la navegación, autenticación y manejo del estado compartido (Peltonen, Mezzalira y Taibi, 2020).

Al implementar el patrón Shell-Remote, las aplicaciones frontend modernas, logran una reducción del acoplamiento al dividir funcionalidades de gran tamaño en unidades autocontenidas que se cargan bajo demanda. Esta estructura contribuye a disminuir los tiempos de carga inicial al evitar que todo el sistema se descargue de una sola vez, y refuerza la modularidad y mantenibilidad (Alekseev, 2024).

Webpack 5

Webpack 5 representa una evolución significativa en las herramientas de empaquetado frontend mejorando el rendimiento, manejo de caché y optimización de

recursos para aplicaciones web modernas. Esta versión incorpora un sistema de build más eficiente y una mejor gestión de dependencias compartidas, lo cual contribuye a la reducción del tamaño de los paquetes y optimiza los tiempos de carga en aplicaciones de gran escala (Bhasin, 2021).

Asimismo, esta herramienta ha evolucionado para convertirse en un componente esencial de los ecosistemas frontend, ya que permite gestionar dependencias complejas y mejorar el rendimiento en entornos de producción.

En la Tabla 14, describimos sus principales características, donde destacan los siguientes.

Tabla 14

Características de Webpack 5

Elemento	Descripción
Module Federation	Proceso continuo de cambio y modernización en herramientas, arquitecturas, metodologías y sistemas
Caché persistente mejorada	Permite que las organizaciones se adapten a entornos dinámicos y exigencias del mercado
Optimización del tamaño de los bundles	Permite que las organizaciones se adapten a entornos dinámicos y exigencias del mercado
Soporte mejorado para WebAssembly y recursos estáticos	Reduce la obsolescencia tecnológica y evita que los sistemas queden desfasados

Carga dinámica

La carga dinámica en aplicaciones frontend consiste en la importación de módulos o recursos, a demanda del usuario en tiempo de ejecución del sistema, en lugar de cargarlos todos al inicio. Esta técnica permite reducir el tamaño del bundle inicial, mejorar los tiempos de carga y optimizar el consumo de recursos del cliente (Banks et al., 2021).

Con respecto al rendimiento, la carga dinámica ayuda a mejorar métricas clave

como el Time to Interactive y el First Contentful Paint, evitando la descarga innecesaria de código no utilizado en la primera renderización. Esto termina siendo demasiado beneficiosa en aplicaciones de gran escala, donde múltiples funcionalidades no son requeridas simultáneamente por el usuario. Obteniendo como resultado, interfaces más rápidas y una experiencia de usuario más fluida (Jones, A., 2025).

CI/CD

La Integración Continua (CI) se refiere a integrar de manera frecuente los cambios de código de los desarrolladores en un repositorio compartido, mientras que la Entrega Continua (CD) extiende estos procesos automatizados hasta el punto de tener disponible el software para despliegues en entornos de prueba o producción (Mylsamy & Mehrotra, 2025).

El uso de CI/CD mejora la colaboración entre desarrollo y operaciones al estandarizar y automatizar tareas repetitivas, lo que nos permite liberar a los equipos para enfocarse en innovación y calidad del producto. Esta automatización facilita la retroalimentación temprana y constante, permitiéndonos acelerar el ciclo de entregas de funcionalidades (Chava, A., 2024).

En proyectos complejos, el CI/CD no solo acelera la velocidad de entrega, sino que también unifica los procesos de automatización bajo pipelines bien diseñados que reducen errores humanos y aseguran consistencia entre entornos de desarrollo, prueba y producción. Esto resulta ser crucial para mantener la calidad de aplicaciones en fase de integración y despliegue (Jeeri, K., 2024).

Pipelines

Los pipelines automatizan tareas repetitivas del ciclo de vida del software, integrando compilación, pruebas y despliegue en una secuencia automatizada que reduce errores humanos y acelera las entregas. Esta automatización forma parte del ecosistema de DevOps, ya que permite integrar cambios de forma continua y confiable, facilitando el desarrollo colaborativo y la entrega rápida de código funcional (Thallapally, 2021).

El uso de pipelines en proyectos ágiles permite que los equipos integren con mayor frecuencia cambios y mantengan un flujo continuo de entrega, reduciendo tiempos de entrega y aumentando la eficiencia. Esto se logra por medio de la integración y la entrega continuas en cada iteración de desarrollo, con pruebas automatizadas que mejoran la calidad del software (Donca et al., 2022).

De acuerdo con el autor, en la Tabla 15 listamos los beneficios de la adopción de pipelines.

Tabla 15

Beneficios del uso de pipelines

Nº	Beneficio
1	Automatización de compilación, pruebas y despliegue
2	Reducción de errores humanos en tareas repetitivas
3	Retroalimentación rápida y continua sobre cambios en el código
4	Integración con múltiples herramientas de control de versiones y despliegue
5	Escalabilidad y adaptación a entornos en la nube

Bitbucket

Bitbucket es una plataforma de gestión de repositorios de código fuente, que permite a los equipos almacenar, versionar y colaborar en código fuente de forma segura. Nos ofrece herramientas integradas para trabajar con ramas, pull requests, revisiones de código y control de versiones, pudiendo así gestionar los cambios entre desarrolladores independientemente de su ubicación (Orange Editorial Board, 2025).

Acorde a lo dicho por el autor, en la Tabla 16 listamos los beneficios y características principales de Bitbucket.

Tabla 16*Beneficios de Bitbucket*

Nº	Beneficio
1	Hospedaje de repositorios Git privados y públicos
2	Integración nativa con Jira, Trello y Confluence
3	Control de acceso mediante autenticación y permisos
4	Revisión de código con pull requests y comentarios en línea
5	Integración con pipelines de CI/CD

Microsoft Azure

Corresponde a la plataforma de servicios en la nube de Microsoft que ofrece alojamiento de aplicaciones, manejo de bases de datos y soluciones de inteligencia artificial, configuración de redes sin necesidad de tener servidores propios. Con el objetivo de tener un espacio flexible y escalable en los centros de datos que Microsoft tiene distribuidos a nivel mundial (Sullivan y Scott, 2020).

A continuación, en la Tabla 17 se presentan sus principales características, las cuales resumen los aspectos más relevantes de su propuesta tecnológica.

Tabla 17

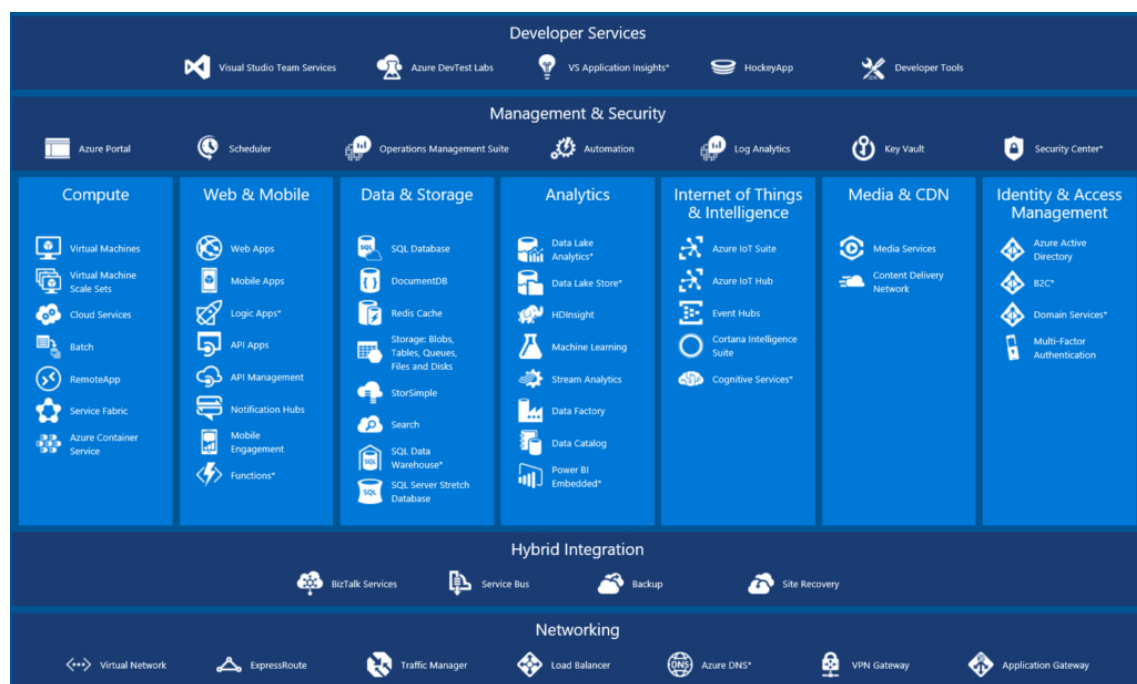
Beneficios de Microsoft Azure

Nº	Beneficio
1	Servicios en la nube en modalidad IaaS, PaaS y SaaS
2	Escalabilidad global con centros de datos en distintas regiones
3	Integración nativa con herramientas como Visual Studio y GitHub para múltiples lenguajes y frameworks
4	Opciones avanzadas de seguridad y cumplimiento normativo
5	Herramientas de análisis de datos y soluciones de inteligencia artificial
6	Modelos de pago flexibles según consumo

Azure ofrece diversos servicios en la nube, de los cuales se resaltan las más usadas en la Figura 4.

Figura 4

Servicios de Azure



Nota. Fuente: Microsoft. (2023)

Jira

Jira corresponde a una herramienta orientada a la gestión de proyectos creada por Atlassian, empleado por muchos equipos que trabajan con metodologías ágiles. Sirve para planear, organizar y dar seguimiento a tareas, errores o historias de usuario. Su flexibilidad hace que pueda usarse tanto en proyectos pequeños como en iniciativas más grandes y complejas (Li, 2022).

A continuación, en la Tabla 18 se muestran los principales beneficios del uso de Jira en nuestros proyectos.

Tabla 18

Beneficios del uso de Jira

Nº	Beneficio
1	Tableros ágiles con soporte para Scrum y Kanban
2	Creación y seguimiento de issues (tareas, errores, historias de usuario)
3	Integración con herramientas como Bitbucket, GitHub y Confluence
4	Informes y métricas en tiempo real (burndown charts, velocity, etc.)
5	Personalización de flujos de trabajo según el equipo
6	Soporte para automatizaciones de procesos repetitivos
7	Disponible en la nube o en servidores propios

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

Esta sección está organizada acorde con el punto 3.2.3.1, dónde se describen las etapas del proyecto.

Antes de exponer la solución propuesta, vale presentar el equipo de desarrollo que

participó en el proyecto, así como el cronograma que organizó la distribución de las etapas.

Equipo de desarrollo

Para llevar a cabo el desarrollo de la evolución tecnológica, se formó un squad de desarrollo, donde el autor formó parte con el rol de desarrollador frontend senior, el squad se describe en la Tabla 19.

Tabla 19

Squad evolución tecnológica

Rol	Descripción	Cantidad de miembros
Arquitecto Front	Encargado de liderar el desarrollo de la capa visual del sistema, asegurando buenas prácticas, orientando al equipo y participando en decisiones técnicas relevantes.	1
Desarrollador Frontend Senior	Define lineamientos generales de la arquitectura y vela por la coherencia técnica de los diferentes componentes del proyecto.	2
Technical Lead	Coordina tareas y supervisa la correcta integración de los entregables desarrollados.	1
QA	Responsable de planificar y ejecutar pruebas para verificar la calidad, estabilidad y correcto funcionamiento del sistema.	1

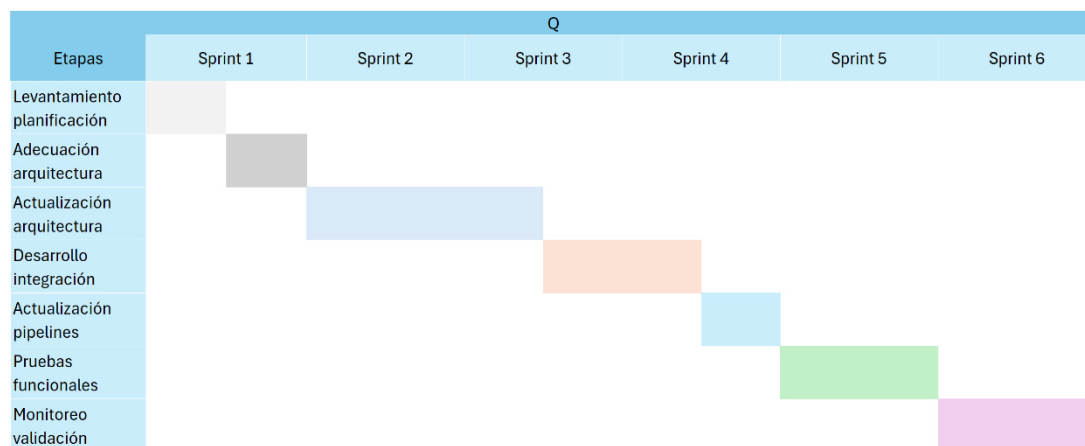
Cronograma del proyecto

Asimismo, se comparte el roadmap que se tuvo para el desarrollo de la evolución tecnológica, recordemos que se está usando el marco de trabajo ágil SCRUM, el cual trabaja en base a sprints, en esta ocasión el sprint consta de 2 semanas.

Se constituye un Q, que equivale a 6 sprints, dando un tiempo aproximado de 12 semanas. En la Figura 5 vemos el cronograma de actividades por etapas con relación a los sprints del Q.

Figura 5

Cronograma por etapas



Nota. Fuente: Adaptado al cronograma del proyecto

En la Figura 5 notamos que la columna de Etapas, contamos con las 7 etapas que comprenden el desarrollo de la solución. Estas se relacionan con el tiempo de desarrollo mediante sprints, que se traducen a 2 semanas de trabajo.

La etapa de *Levantamiento y planificación* tiene una duración de 1 semana, al igual que la etapa de *Adecuación de la arquitectura*. Puesto que son etapas de registro y levantamiento de información.

En cambio, la etapa de *Actualización de la arquitectura* es la etapa mas larga de los 7, con una duración de 3 semanas. Puesto que involucra la migración a Angular 15 de todos los microfronts seguido de la configuración de webpack para el uso de módulos federados.

La etapa de *Desarrollo e integración* comprende la duración de 2 semanas, puesto que en esta etapa se va a unir las piezas, pero desde el enfoque de módulos federados.

Seguido de la etapa de *Actualización de pipelines* que duró 1 semana, considerando que ya se contaba con pipelines previos y se pasó a actualizarlas por el nuevo enfoque.

Las etapas finales son *Pruebas funcionales* y *Monitoreo y validación*, ambos con una duración de 2 semanas. Puesto que estas etapas se encargarán de verificar, validar y controlar el correcto funcionamiento de la solución desarrollada. Así como también, poder dar solución frente a algunas inconsistencias identificadas en la fase de pruebas.

3.2.5.1. *Levantamiento y planificación de los microfrontends involucrados*

En esta etapa, se hizo el registro de todos los microfrontends involucrados, así como de las librerías que el sistema utiliza. Cabe resaltar que acorde a las políticas de la entidad bancaria, se prioriza el uso de las librerías desarrolladas por el mismo, a fin de dejar de depender de librerías de terceros y garantizar la seguridad de sus sistemas.

Durante el desarrollo de esta etapa, se tuvo que recabar toda la documentación posible, para conocer el estado real de la arquitectura antes de iniciar con la evolución tecnológica, además de identificar que recursos se consumen, que librerías se usan, el total de microfrontends entre otros.

Uno de los desafíos que se tuvo que afrontar, fue la poca documentación que existía respecto a la arquitectura. Además, que estos se encontraban muy desactualizados. Por ende, se tuvo que investigar mediante la revisión de código, revisión de repositorios de Bitbucket e ingresando al portal de Azure para ver los recursos.

Finalmente, pudimos identificar los siguientes registros, las cuales se detallan en la Tabla 20.

Tabla 20*Recursos registrados*

Tipo	Descripción	Cantidad
Microfront	Microfront con una funcionalidad en particular que forma parte del todo en el sistema	9
Librería	Biblioteca compartida utilizada por todos los microfronts del sistema	2
Recurso estático compartido	Artefacto estático desplegado de uso común entre los microfront	2

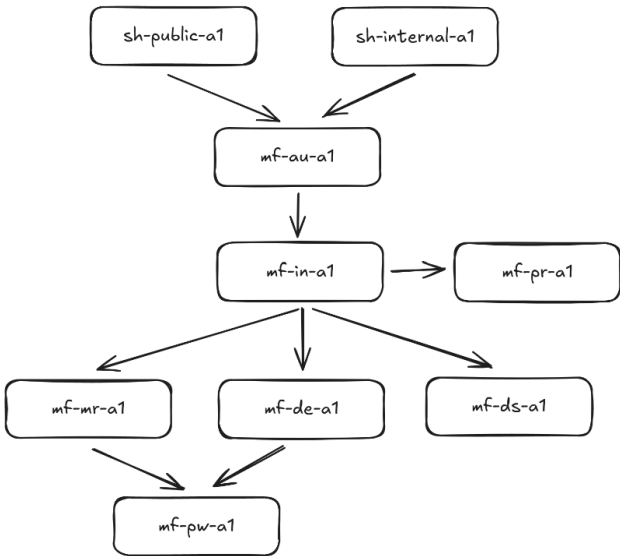
Luego del registro de recursos, se hizo un análisis de cada microfront, donde se pudo identificar la relación que tiene uno con otro, sus dependencias y el alcance que tiene cada uno de ellos.

Para fines de protección de los componentes que conforman la arquitectura, se decidió aplicar un renombramiento por nomenclatura estandarizado que nos facilitará el entendimiento de los recursos. En la Figura 6, vemos la relación que guarda los microfrontends del sistema.

Donde contamos con 2 microfronts por el cual podemos acceder al sistema `sh_public_a1` y `sh_internal_a1`. Estos conducen al `mf-au-a1` que a su vez nos dirige al `mf-in-a1` el cual nos permite acceder al resto de microfronts. Considerando que la relación entre un microfront y otro microfront se rige en base al modelo de negocio que la entidad posee.

Figura 6

Diagrama de relación de microfrontends

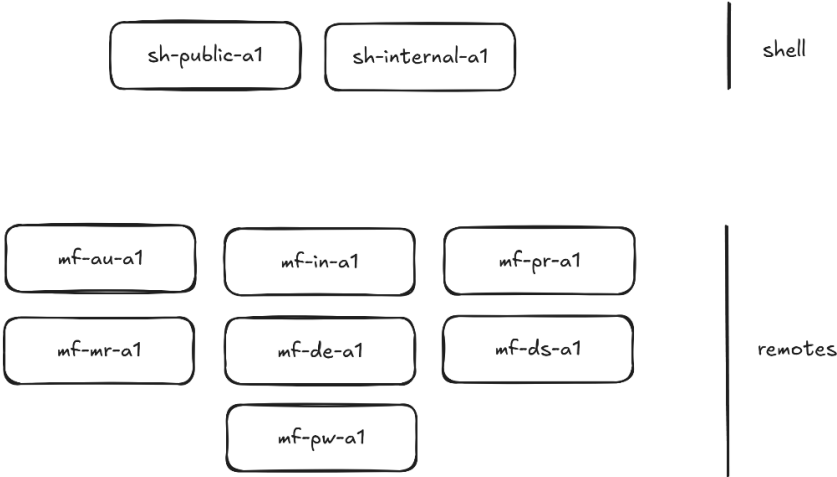


3.2.5.2. *Adecuación de la arquitectura existente para la migración*

En esta etapa definimos el esquema shell-remote como paso previo a la integración de los módulos federados. En la Figura 7, se evidencia que se cuenta con 2 microfronts de tipos shell, debido a que ambos son puntos de accesos de la intranet y extranet del sistema que van a orquestar y proveer al resto de los microfronts. En consecuencia, se identifican los 7 microfronts restantes como microfronts remotos.

Figura 7

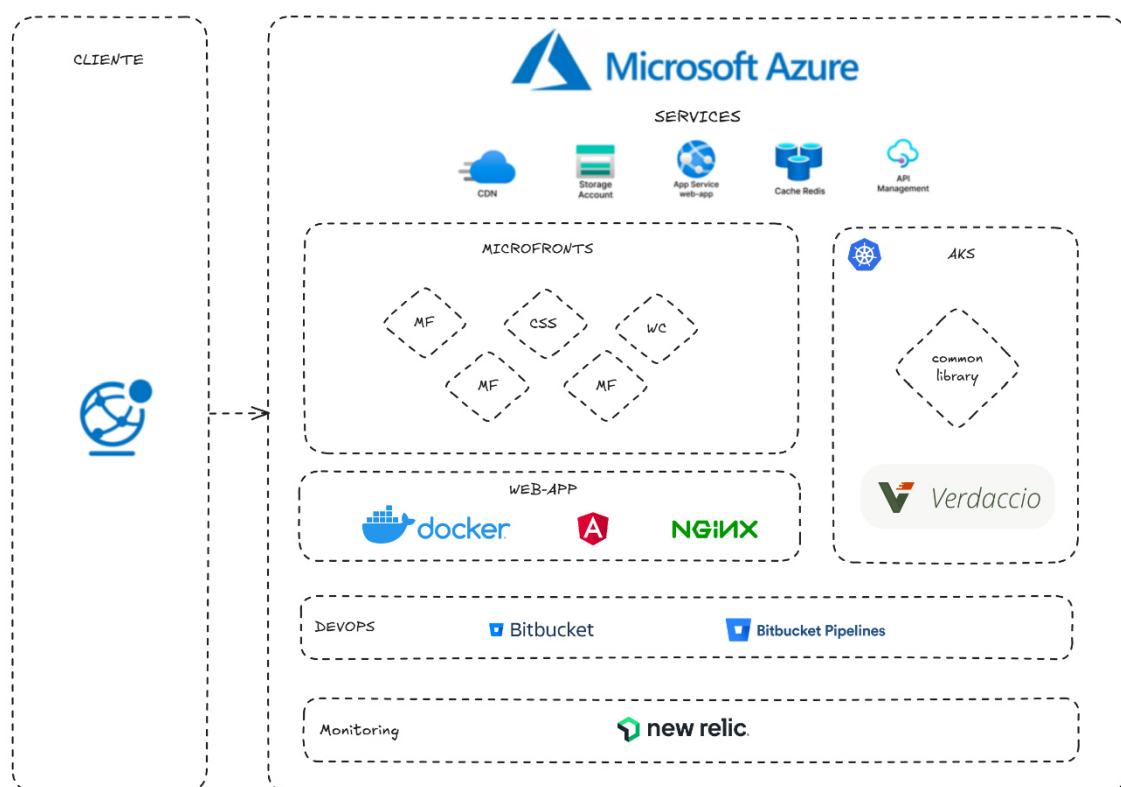
Diagrama de esquema shell-remotes



Continuando con esta etapa, se procedió con el ajuste de la arquitectura actual del proyecto, que si bien es cierto estaba definida bajo una arquitectura de microfrontends, no se contaba con un diagrama de arquitectura clara donde se pueda evidenciar todos los microfronts, servicios, recursos compartidos, etc, que nos facilitara un mayor entendimiento del sistema a alto nivel.

Figura 8

Diagrama de arquitectura frontend no optimizado



Nota. Fuente: Arquitectura no optimizada adaptado de la entidad financiera

A continuación, en la Figura 8 podemos ver el diagrama de arquitectura que se contaba justo antes de iniciar la optimización, si bien es cierto el diagrama cuenta con algunos servicio y tecnologías que se utilizan. No era lo suficientemente descriptivo como para iniciar la solución. Además de ser un diagrama desfasado en el tiempo que no reflejaba la actualidad del sistema.

Los microfronts se encontraban muy acoplados, las integraciones estáticas dentro

del sistema, así como los despliegues. Respecto a las dependencias, se representaban de manera implícita, además se evidencia una complejidad creciente en la mantenibilidad. Por ello, era indispensable graficar la nueva arquitectura considerando las bondades que los módulos federados ofrecen.

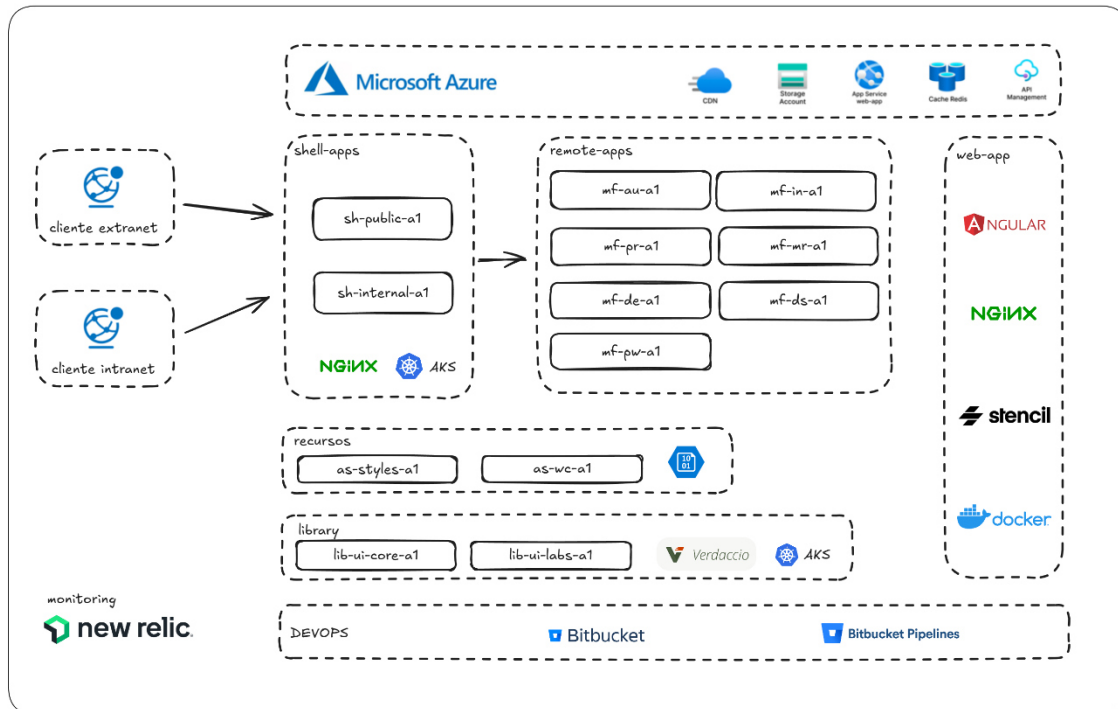
Luego de la etapa 1 en el punto 3.2.5.2, se pudo levantar un registro de todos los recursos que cuenta el proyecto, así como la relación que tiene uno con otro. Esto con la finalidad de poder generar el diagrama de arquitectura frontend.

Desde el punto de vista arquitectónico, el rediseño priorizó atributos de calidad como disponibilidad, mantenibilidad, escalabilidad organizacional y desacoplamiento. La adopción de un esquema federado permitió aislar fallos por microfrontend, reducir el impacto de despliegues parciales y facilitar la evolución independiente de cada componente. Asimismo, se evaluaron alternativas de integración como enfoques monolíticos o integraciones estáticas, descartándolas por su alto nivel de acoplamiento y menor flexibilidad evolutiva frente a los requerimientos de un entorno bancario.

A continuación, en la Figura 9 podemos ver el diagrama de arquitectura optimizado del proyecto. Dónde evidenciamos que se puede acceder al sistema por dos frentes, por el navegador extranet, que corresponde a los clientes en general y por el cliente intranet, donde solo tiene acceso personal administrativo de la entidad bancaria.

Figura 9

Diagrama de arquitectura frontend optimizado



Nota. Fuente: Arquitectura optimizada adaptado de la entidad financiera

Seguidamente contamos con la capa shell-apps que contiene a sh_public_a1 y sh_internal_a1, donde alojamos a nuestros microfronts que contendrán a los orquestadores de los microfronts remotos. Dicha capa estará desplegada en Azure Kubernetes Service (AKS) y expuesta mediante NGINX, que actuará como hosts para la orquestación de los microfronts.

La capa remote-apps contiene los 7 microfronts que engloba el total de funcionalidades del sistema, cada uno posee una lógica y funciones en particular. Así como independencia en su desarrollo y despliegue, integrándose dinámicamente por los módulos federados.

La capa de recursos compartidos cuenta con 2 recursos, el as-styles-a1, que contiene los estilos, clases, constantes, etc. Que serán usados por todos los microfronts que contiene el sistema y el as-wc-a1, que contiene todos los webcomponents que el sistema va a utilizar en todos sus microfronts.

La capa de librerías que cuenta con 2 recursos, el `lib-ui-core-a1` que nos provee de enums, interfaces, operations, queries, etc, que serán usados de manera repetitiva por los microfronts y `lib-ui-labs-a1`, que nos proveerá de funciones que nos facilitará el tratamiento de tipos de datos primitivos, validaciones, pipes etc. Las librerías serán deployadas en un registro privado llamado Verdaccio.

La capa web-app contiene las tecnologías que se van a utilizar en el sistema, el framework Angular en su versión 15.2.2, posteriormente se va a explicar a detalle el versionamiento. Angular será el framework principal en las que serán desarrollados los microfrontends.

Nginx que servirá de orquestador entre los microfronts shell y remote, configuraciones de seguridad, redirecciones etc. También se utilizará Stencil que es un compilador para componentes web, que se utilizará principalmente para los recursos de webcomponents `as-wc-a`. Finalmente se utilizará Docker para empaquetar las shell, remotes, librerías en contenedores que serán desplegados y orquestados en un AKS.

La capa DEVOPS, representará a Bitbucket que será nuestra fuente de repositorios de los recursos, donde podremos almacenar código, hacer code review, generar los pull request y tener un control óptimo de las versiones. Por otro lado, tendremos Bitbucket pipelines con quienes manejaremos el CI/CD, la automatización de la construcción, prueba y empaquetado de los artefactos.

Finalmente contaremos con la capa de la nube, por lo cual tendremos a disposición todo el ecosistema de Microsoft Azure, donde se hará uso de sus diversos servicios que esta nube provee tales como las CDN, storage account, web-app, redis, api management entre otros. Como punto adicional, el sistema estará constantemente monitoreado por new relic, un servicio que nos mostrará métricas y gráficas sobre el sistema, su funcionamiento, caídas, tráfico entre otras unidades de medición.

Cabe resaltar que, una vez finalizada la diagramación de la nueva arquitectura, esta fue documentada y registrada en la herramienta Confluence, con el propósito de

centralizar la información técnica y asegurar su disponibilidad para los equipos involucrados.

3.2.5.3. Actualización tecnológica y configuración de microfrontends

En esta etapa iniciamos con el desarrollo de la evolución tecnológica, que fue el uso de módulo federados en la arquitectura de microfrontends. Por lo cual se debe tener en consideraciones pasos previos como la migración a una versión de angular que soporte el uso de módulos federados, homologación de librerías que soporten la nueva versión y algunas configuraciones iniciales que se requiera para dicha evolución tecnológica.

El proyecto estaba desarrollado en Angular en su versión 12.0.3, el cual permitía el uso de microfrontends mediante iframes, importaciones estáticas o repositorios monolitos. Pero para el uso de módulos federados, se requería al menos la versión 13, para poder usar dicha tecnología.

Se optó por usar la versión 15.2.2 de Angular, con el fin de usar la última versión estable que el framework tenía publicado, además de gozar de ventajas que traía consigo, las cuales se describen en la Tabla 21.

Tabla 21

Diferencia entre Angular 12 y 15

Aspecto	Angular 12	Angular 15
Compilador	Usaba View Engine; menor rendimiento.	Solo Ivy; compilación más rápida.
Webpack	Versión 4 por defecto; soporte 5 experimental.	Webpack 5 integrado y estable.
Microfrontends	Integración manual y fuertemente acoplada.	Integración dinámica con <i>Module Federation</i> .
Arquitectura	Estructura monolítica y poco flexible.	Arquitectura federada y escalable.

Dependencias	TypeScript 4.2 y RxJS 6.	TypeScript 4.9 y RxJS 7.
CLI y builds	Builds más lentos y pesados.	CLI optimizada y compilaciones ágiles.
Estabilidad	Limitada para entornos modernos.	Ecosistema más maduro y estable.

En la Tabla 21 resumimos todas las ventajas que trae consigo la migración de Angular 12 a 15, donde podemos resaltar el uso de webpack, módulos federados y compilaciones más rápidas.

Uno de los mayores retos que se tuvo que afrontar, fue verificar que las dependencias que se usaban en Angular 12 funcionen correctamente después de la migración a Angular 15, debido a que este deprecaba el uso de algunas librerías tales como TSLint, una librería que se encargaba de corregir errores de sintaxis en TypeScript, malas prácticas de desarrollo, etc.

Debido a la migración a Angular 15, se adoptó a ESLint en su totalidad. ESLint al igual que TSLint cumple las mismas funcionalidades, con la diferencia que ESLint posee más reglas de sintaxis, es más flexible y cuenta con soporte oficial. Por ello la migración nos obliga a dejar de usar la librería TSLint para implementar la librería ESLint, en todos los 9 microfrontds.

Otra de las actividades que se realizaron fue la actualización de algunas dependencias, en su mayoría propias de angular como evidenciamos en la Tabla 22. Recordando que se optó por usar la versión de Angular 15.2.2, considerándose la más estable en su momento, esto conllevó la actualización de 9 dependencias de Angular.

Tabla 22*Listado de dependencias de Angular actualizados*

Dependencia	Versión en Angular 12	Versión en Angular 15
@angular/animations.	~12.0.3	^15.2.2
@angular/common.	~12.0.3	^15.2.2
@angular/compiler.	~12.0.3	^15.2.2
@angular/core.	~12.0.3	^15.2.2
@angular/elements.	~12.0.3	^15.2.8
@angular/forms.	~12.0.3	^15.2.2
@angular/platform-browser.	~12.0.3	^15.2.2
@angular/platform-browser-dynamic.	~12.0.3	^15.2.2
@angular/router.	~12.0.3	^15.2.2

Otras de las dependencias que fueron actualizados, fue la dependencia de apollo-angular que nos ayuda a conectar desde Angular hacia los contratos de GraphQL, ng-recaptcha que nos ayuda para la validación en la autenticación, entre muchas otras librerías que no son propiamente de Angular, tal como se ve en la Tabla 23.

Tabla 23*Listado de dependencias que no son propias de Angular actualizados*

Dependencia	Versión en Angular 12	Versión en Angular 15
apollo-angular	^2.6.0	^4.2.1
graphql	^15.3.0	^15.8.0
ng-recaptcha	^8.0.0	^11.0.0

Iniciamos con la adopción de los módulos federados, comenzando con la instalación de las dependencias de módulos federados, seguidamente de la configuración del archivo webpack.

Para hacer uso de los módulos federados se requirió la instalación de la librería `@angular-architects/module-federation`, que nos permitirá generar el archivo de configuración de Webpack para el microfront shell y los remotes, así como compartir dependencias y facilitar la integración dinámica de microfrontends.

Una vez instalada la librería, se generó automáticamente el archivo `webpack.config.js` que consta de 3 secciones, importación, exportación y plugin de librerías compartidas.

En la Figura 10 vemos la sección de importación, donde se importó `microfront`, que facilita la integración con Angular, `ModuleFederationPlugin`, el plugin central de webpack 5 que permite la federación del microfront y `path` para utilizar la ruta de los archivos.

Figura 10

Sección de importación

```
const mf = require('@angular-architects/module-federation/webpack');
const ModuleFederationPlugin = require('webpack/lib/container/ModuleFederationPlugin');
const path = require('path');
const share = mf.share;
const sharedMappings = new mf.SharedMappings();

sharedMappings.register(path.join(__dirname, 'tsconfig.json'), [
  /* mapped paths to share */
]);
```

En la Figura 11 vemos la sección de exportación, donde `uniqueName` es el nombre identificador del microfronted dentro de la federación, `publicPath` en `auto` que permite cargar dinámicamente los archivos de diversas rutas y `runtimeChunk` evita la división de los runtimes para la carga de los microfronts remotos.

Figura 11

Sección de exportación

```
module.exports = {
  output: {
    uniqueName: 'xxxx',
    publicPath: 'auto'
  },
  optimization: {
    runtimeChunk: false
  },
  resolve: {
    alias: {
      ...sharedMappings.getAliases()
    }
  },
  experiments: {
    outputModule: true
  },
}
```

Como tercera sección, tenemos el apartado de plugins donde podemos decir que es la parte más importante de la configuración, responsable de mostrar cómo se va a compartir las dependencias en todo el sistema a través del shared. Permitiendo así instancias únicas similares al patrón singleton. Evitando la duplicidad y garantizando compatibilidad de versiones.

Desde una perspectiva arquitectónica, esta configuración respondió no solo a una necesidad de integración técnica, sino también a criterios de gobernanza en un entorno bancario. Se priorizó la compartición de dependencias críticas bajo el principio de instancia única, evitando la coexistencia de múltiples versiones del framework en tiempo de ejecución y reduciendo riesgos de incompatibilidad entre microfrontends.

Asimismo, el versionamiento independiente de los microfrontends permitió mantener ciclos de despliegue desacoplados y habilitar mecanismos de rollback controlado ante fallos parciales. El uso de manifest.json dinámico y publicación versionada en la nube contribuyó a mitigar problemas de caché obsoleta, reforzando atributos de calidad como disponibilidad, mantenibilidad y resiliencia operativa del sistema.

Esta configuración permite la carga dinámica de remotos y asegura que todos los microfrontends utilicen versiones coherentes de las dependencias compartidas. Tal como vemos en la Figura 12, la configuración del microfront shell.

Figura 12

Sección de configuración de dependencias compartidas en un microfront shell

```
plugins: [  
  new ModuleFederationPlugin({  
    library: { type: 'module' },  
  
    // For remotes (please adjust)  
    name: 'xxxx',  
  
    shared: share({  
      '@angular/core': { singleton: true, eager: true, strictVersion: true, requiredVersion: 'auto' },  
      '@angular/common': { singleton: true, eager: true, strictVersion: true, requiredVersion: 'auto' },  
      '@angular/common/http': { singleton: true, eager: true, strictVersion: true, requiredVersion: 'auto' },  
      '@angular/router': { singleton: true, eager: true, strictVersion: true, requiredVersion: 'auto' },  
      '@apollo/client': { singleton: true, eager: true, includeSecondaries: true, version: 'auto', strictVersion: false },  
      'apollo-angular': { singleton: true, eager: true, strictVersion: true, requiredVersion: 'auto', includeSecondaries: true },  
      rxjs: { singleton: true, strictVersion: false, eager: true, requiredVersion: 'auto' },  
      'rxjs/operators': { singleton: true, strictVersion: false, eager: true, requiredVersion: 'auto' },  
      '@angular/forms': { singleton: true, eager: true, strictVersion: true, requiredVersion: 'auto', includeSecondaries: true },  
  
      ...sharedMappings.getDescriptors()  
    })  
  ]),  
  sharedMappings.getPlugin()  
]
```

Así mismo en la configuración de un microfront remote, notamos un ligero cambio, puesto que un microfront remote, lo que hace es compartir las librerías q se usaran en todo el sistema y a su vez exportar módulos.

Aplicando el principio de reutilización y evitar la duplicidad de código. En el apartado de “exposes” tal como vemos en la Figura 13, exponemos el nombre del módulo que se va a utilizar en otras instancias. Lo cual nos permite tener el control de aquello que se va a exportar a demanda y que el “remoteEntry.js” sea más ligero, evitando exportar algo que no se va a utilizar.

Figura 13

Sección de configuración de dependencias compartidas en un microfront remote

```
plugins: [  
  new NodePolyfillPlugin(),  
  new ModuleFederationPlugin({  
    library: { type: 'module' },  
    name: 'xxxxxx',  
    filename: 'remoteEntry.js',  
    exposes: {  
      SaveModule: './projects/remote01/src/app/workflows/mod01.module.ts',  
      PriceModule: './projects/remote01/src/app/workflows/mod02.module.ts',  
      ShowModule: './projects/remote01/src/app/workflows/mod03.module.ts',  
      SaveModalsComponent: './projects/remote01/src/app/shared/mod04.component.ts'  
    },  
  },  
  shared: share({  
    '@angular/core': { singleton: true, strictVersion: false, eager: true, requiredVersion: 'auto' },  
    '@angular/common': { singleton: true, strictVersion: false, eager: true, requiredVersion: 'auto' },  
    '@angular/common/http': { singleton: true, strictVersion: false, eager: true, requiredVersion: 'auto' },  
    '@angular/router': { singleton: true, strictVersion: false, eager: true, requiredVersion: 'auto' },  
    '@apollo/client': { singleton: true, eager: true, includeSecondaries: true, version: 'auto', strictVersion: false },  
  })  
],
```

Adicionalmente en “filename” se nombra como remoteEntry.js a ese archivo que va a ser compilado y transpilado a js que representa como tal el microfront remote que se utilizará en todo el sistema.

3.2.5.4. *Desarrollo e integración de microfrontends*

En esta etapa, pasamos a integrar los microfrontends remotes en el microfront shell. Considerando que ya tenemos configurado previamente los webpack de cada microfront, donde nombramos el microfront y seleccionamos los módulos que se van a exponer. Hasta esta etapa, ya tenemos registrado el total de microfrontends, una arquitectura adecuada, definida y con todos los microfrontends completamente configurados. Dando paso a la integración de los microfrontends.

Antes de hablar de la integración de los microfrontends, es necesario saber sobre la creación de un archivo llamado manifest.json, que cumplirá la función de registrar las ubicaciones de los demás microfronts mediante su remoteEntry.js.

En la Figura 14 podemos ver la estructura del manifest.json, donde incluye el nombre del microfront, que es el que se asigna en la configuración webpack en la Figura 10, así como la dirección del remoteEntry.js que obtenemos por cada microfront.

Figura 14

Manifest json, con rutas locales

```
{
  "web_micro_sale": "http://localhost:4202/remoteEntry.js",
  "web_micro_watch": "http://localhost:4203/remoteEntry.js",
  "web_micro_market": "http://localhost:4204/remoteEntry.js"
}
```

Para fines de desarrollo local, se apunta al localhost debido a que el levantamiento de cada microfront es independiente. Posteriormente las urls deberán apuntar al remoteEntry.js que se encuentran ya desplegadas.

Como un ejemplo la Figura 15, donde las direcciones corresponden al remoteEntry.json ya desplegados, dejando de lado las direcciones locales.

Figura 15

Manifest json, con rutas remotas

```
{
  "web_micro_sale": "https://straeu2c001.blob.core.windows.net/micro_sale/3.3.20/remoteEntry.js",
  "web_micro_watch": "https://straeu2c001.blob.core.windows.net/micro_watch/3.3.28/remoteEntry.js",
  "web_micro_market": "https://straeu2c001.blob.core.windows.net/micro_market/1.1.3/remoteEntry.js"
}
```

La integración se dará de 2 formas, la primera será integrando un módulo de un microfront dentro de un componente y la segunda será integrando los módulos de un microfront directamente en el routing aprovechando el lazyloading.

A continuación, en la Figura 16 podemos observar cómo pasamos a integrar los módulos expuestos de un microfront remote en un microfront shell.

Figura 16

Integración por módulo

```
export class SaleComponent implements OnInit {
  @ViewChild('templateSales', { read: ViewContainerRef }) viewContainer!: ViewContainerRef;
  constructor(private manifest: ManifestService) {}
  async ngOnInit(): Promise<void> {
    await this.load();
  }

  async load(): Promise<void> {
    const { remoteEntry } = this.manifest.get(ManifestNameType.SALE);
    const expose = await loadRemoteModule({
      type: 'module',
      remoteEntry,
      exposedModule: 'SaleComponent'
    });
    this.viewContainer.createComponent(expose.SaleComponent);
  }
}
```

Dentro de un componente, en este caso llamado “SaleComponent”, obtenemos una referencia de un contenedor dinámico en angular “templateSales”, que será dónde se cargue el módulo a importar.

De acuerdo con el ciclo de vida de angular, el ngOnInit se ejecutará posterior al constructor. Ejecutando el método “load()”, donde mediante un servicio del manifest, obtendremos la dirección que corresponde al microfront remote.

Y mediante el método “loadRemoteModule()” de “@angular-architects/module-federation”, se mandará por parámetro un objeto de 3 campos, el primero “type”, que hace referencia al tipo de recurso que se busca importar, en este caso es un “module” que también está definido en la configuración de webpack en la figura 14.

El segundo campo es “remoteEntry”, donde se colocará la dirección del remoteEntry del microfront remote. Este se obtiene mediante un servicio, indicando que microfront se requiere.

En el tercer campo “exposedModule”, se especifica el módulo a importar dentro de todos los módulos que el microfront remote expone en su configuración de webpack. Finalmente, se creará el componente que importamos desde un microfront remote en el

microfront shell.

Otra de las formas que se importó un microfront, es directamente desde una ruta como vemos en la Figura 17, dónde también invocamos el método “loadRemoteModule()”, que tiene por parámetro un objeto con 3 campos.

Figura 17

Integración por routing

```
{
  path: 'solicitud',
  loadChildren: () => import('./workflows/request/request.module').then((m) => m.RequestModule)
},
{
  path: 'productos',
  loadChildren: () =>
    loadRemoteModule({
      type: 'module',
      remoteEntry: manifest.get(ManifestNameType.SALE).remoteEntry,
      exposedModule: 'SaleModule'
    })
    .catch(() => import('@applications/error/error.module').then((m) => m.ErrorModule))
    .then((m) => m.SaleModule)
},
}
```

Aprovechando así la carga dinámica y el lazy loading. Este enfoque facilitó el despliegue a demanda y mejoró la escalabilidad de la arquitectura frontend. Las configuraciones previamente descritas deberán ser replicadas en los 9 microfronts registrados en el punto 3.2.5.1.

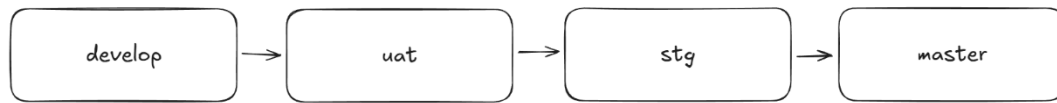
3.2.5.5. *Actualización de pipelines y despliegue en la nube*

En esta etapa nos encargaremos de describir los ambientes de desarrollo que contamos en la entidad financiera, posteriormente actualizaremos los pipelines considerando los pasos que el pipeline contiene, así como los cambios que estos sufrieron a raíz de la migración. La configuración de los pipelines de los microfronts shell y microfronts remotos.

Durante el desarrollo de la solución contamos con 4 ambientes de desarrollo, las cuales se muestran en la Figura 18.

Figura 18

Ambientes de desarrollo



Cada ambiente de desarrollo que se muestran en la Figura 17 tiene una funcionalidad en particular y está dirigido a un equipo de trabajo, los cuales se detallan en la Tabla 24.

Tabla 24

Descripción de los ambientes de desarrollo

Ambiente	Función
develop	Este ambiente es de uso extensivo por parte de los desarrolladores que constantemente están subiendo sus cambios y viendo el comportamiento de ellos en un ambiente desplegado.
uat	Este ambiente es de uso exclusivo por parte del equipo de QA, quienes se encargaron de realizar la certificación de lo desarrollado por el equipo de desarrollo.
stg	El ambiente stg o de preproducción, es de los ambientes mas estables que se tiene, puesto que solo los cambios que han sido certificados por el equipo de QA, pueden subir. Además de ser un ambiente que es utilizado para testear errores de producción.
master	El ambiente productivo, es el ambiente final donde el usuario final podrá hacer uso del sistema.

A continuación, se describirá la estructura que cuenta el pipeline. Cabe resaltar que la estructura de pasos es el mismo entre un microfront shell y remote. El pipeline estará en un archivo llamado *bitbucket-pipelines.yml* ubicado en la raíz del proyecto.

Figura 19

Pasos del pipeline

```
definitions:  
  steps:  
    - step: &InstallPackages ...  
    - step: &LintingCode ...  
    - step: &QualityCode ...  
    - step: &Build ...  
    - step: &Deploy ...
```

En la Figura 19, podemos ver que el pipeline consta de 5 pasos, de los cuales el primero es *InstallPackages*, que se encargará de instalar las dependencias propias de node y librerías externas. El paso *LintingCode* que se encargará de corregir errores de sintaxis y del uso de buenas prácticas de desarrollo.

El paso *QualityCode* se encargará de correr todas las pruebas unitarias y que estos se ejecuten de manera satisfactorio cumpliendo con el coverage programado. El paso *Build* se encargará de generar la carpeta dist que es el empaquetado de todo lo que se quiere desplegar en el siguiente paso.

Finalmente, el paso *Deploy*, es la más importante debido a que en este punto se dará paso al despliegue del microfront, referenciando los recursos de Azure, versionamiento y orquestación mediante el manifest.json.

Cabe resaltar que lo descrito líneas arriba, no necesariamente se deben cumplir en ese orden, puesto que son solo pasos. En la Figura 20 podemos evidenciar en el apartado de *pipelines*, las ramas en donde se ejecutarán los pasos descritos. Las ramas coinciden con los ambientes de desarrollo develop, uat, stg y master.

Figura 20

Estructura del pipeline

```
pipelines:
  default: ...
  branches:
    develop:
      - step: *InstallPackages
      - step: *QualityCode
      - step:
        <<: *Deploy
        name: Deploy to Dev
        deployment: Dev
    uat:
      - step: *InstallPackages
      - step:
        <<: *Deploy
        name: Deploy to Uat
        deployment: Uat
    stg:
      - step: *InstallPackages
      - step:
        <<: *Deploy
        name: Deploy to Stg
        deployment: Staging
        trigger: manual
    master:
      - step: *InstallPackages
      - step: *Build
      - step:
        <<: *Deploy
        name: Deploy to Production
        deployment: Production
        trigger: manual
```

Existe una particularidad en las ramas stg y master, ya que en ambos casos la ejecución del proceso de *Deploy* debe realizarse de manera manual mediante la directiva *trigger: manual*. Esto se debe a que dichos entornos corresponden a ambientes preproductivos y productivos, donde es necesario mantener un mayor control sobre los despliegues y evitar ejecuciones automáticas no autorizadas.

En contraste, los ambientes develop y uat cuentan con despliegues automáticos, al considerarse entornos de desarrollo y pruebas con un nivel de riesgo menor. Cabe resaltar que la modalidad de despliegue ya sea automática o manual, es totalmente configurable para cualquier ambiente, y su definición depende de los lineamientos y políticas establecidas en la implementación de los pipelines.

El pipeline correspondiente a un microfront de tipo shell presentó únicamente cambios menores, debido a que este sigue desplegándose sobre el mismo clúster de Azure Kubernetes Service (AKS). En este sentido, la infraestructura de despliegue y la dirección

de publicación en Azure se mantuvieron sin modificaciones significativas. Los cambios realizados estuvieron orientados principalmente a adecuar el proceso de construcción y validación a la nueva versión de Angular y a la integración con módulos federados

Los pasos *InstallPackages*, *LintingCode*, *Build*, *Deploy*, tienen en común el uso de la imagen de node para las acciones que corresponden, considerando que anteriormente se usaba Angular 12, la imagen de node utilizado en el pipeline era el node:12-alpine, una versión suficiente para la arquitectura de ese entonces.

Posterior a la migración a Angular 15, la imagen de node:12-alpine era insuficiente para seguir compilando, por ello se hizo un cambio a la imagen de node:16, versión suficiente para soportar la arquitectura basado en Angular 15.

En el paso de *QualityCode*, debido a la migración tecnológica del frontend, la cual requirió actualizar el entorno de ejecución para mantener compatibilidad con Angular 15. En este contexto, se reemplazó la imagen basada en Node 12(*zenika/alpine-chrome:86-with-node-12*) y Chrome 86 por una versión más reciente con Node 16 (*zenika/alpine-chrome:102-with-node-16*) y Chrome 102, garantizando la correcta ejecución de pruebas unitarias y herramientas de calidad de código.

El pipeline correspondiente a un microfront de tipo remote, fue ajustado para soportar la nueva versión de Angular y principalmente incorporar un flujo de despliegue desacoplado del microfront shell, permitiendo así que el artefacto se construya y publique de manera independiente. Mejorando el flujo de despliegue que la arquitectura actual tenía.

Figura 21

Versionamiento independiente

```
- export VERSION=$(cat ./version.txt)
- export DEPLOY_URL=$CDN_URL/js/micro-sale/$VERSION/
- sed -i "s|DEPLOY URL|$DEPLOY URL|g" devops/dist/main.js
```

En la Figura 21 podemos visualizar un extracto del paso *Deploy* de un microfront

remote, dónde se genera su versión propia del microfront con la finalidad de mantener su propio ciclo de vida e independencia del microfront shell. Así como la construcción de la URL de publicación, donde se concatena la versión previamente generada, lo cual nos permitirá convivir con varias versiones del microfront, facilitar rollbacks y no sobrescribir a los microfronts. Finalmente, se relacionará la URL con el dist generado.

Figura 22

Despliegue independiente

```
- pipe: microsoft/azure-storage-deploy:2.0.0
variables:
  SOURCE: 'devops/dist/*.js'
  DESTINATION: '$AZURE_BLOB_CONTAINER/js/micro-sale/$VERSION/'
  DESTINATION_SAS_TOKEN: $LOCAL_SAS_TOKEN
  EXTRA_ARGS: '--overwrite=true'
  DEBUG: 'true'
```

En la Figura 22 notamos que el microfront remote se despliega como artefacto estático con su versión única, con ello logramos un despliegue independiente sin afectar a la totalidad del sistema al momento de su despliegue, evitando desplegar nuevamente el shell, recompilar todo el sistema y dependencia del AKS.

Figura 23

Despliegue del manifest.json actualizado

```
- npx ds manifest --cdn=$CDN_URL/js --name=web_micro-sale --url=$AZURE_FILE_CONTAINER/nginx-conf --path=/
- cat devops/manifest.json
- pipe: microsoft/azure-storage-deploy:2.0.0
variables:
  SOURCE: 'devops/manifest.json'
  DESTINATION: '$AZURE_FILE_CONTAINER/nginx-conf/manifest.json'
  DESTINATION_SAS_TOKEN: $LOCAL_SAS_TOKEN
  EXTRA_ARGS: '--overwrite=true'
```

En la Figura 23 notamos un segundo despliegue, pero del manifest.json actualizado, con las nuevas versiones de los microfronts remotes actualizadas en tiempo de ejecución. Permittiéndonos así que el sistema completo esté disponible en todo momento sin comprometer el funcionamiento general y reforzando el desacoplamiento entre los microfronts shell y los microfronts remotes.

Posteriormente el equipo de desarrollo se dispuso a realizar el *Pull Request*

respectivo en los 9 microfronts registrados, con el objetivo de desplegar los cambios al ambiente *develop* y así probar el sistema en un ambiente remoto, como paso previo a la certificación a cargo del equipo QA.

Los pipelines de integración y despliegue continuo fueron documentados y registrados en la herramienta Confluence, con el objetivo de centralizar la información relacionada con los procesos de construcción, validación y despliegue del sistema. Esta documentación permite estandarizar los flujos de trabajo, facilitar el mantenimiento de los pipelines y asegurar que los equipos cuenten con una referencia clara para futuras modificaciones, optimizaciones o ampliaciones del proceso de entrega continua.

3.2.5.6. *Pruebas funcionales e integración continua*

En esta etapa, se procedió con las pruebas funcionales para validar la estabilidad y el correcto funcionamiento del sistema que soporta la nueva arquitectura. Considerando que, hasta este momento, el sistema ya se encontraba migrado a Angular 15, se tenía configurado los webpack e implementado módulos federados en los 9 microfronts y desplegado en el ambiente UAT, bajo aprobación del equipo de desarrollo en el ambiente de *develop*.

En el ambiente de UAT, el QA del proyecto presentó una matriz de prueba de cerca de 40 escenarios. De los cuales se presentarán 20 de los escenarios mas representativos en valor, que nos aseguren que la arquitectura propuesta tiene los resultados esperados, los cuales se detallan en la Tabla 25.

Tabla 25*Casos de pruebas testeados*

ID	Caso de prueba	Descripción	Resultado esperado	Estado
CP-01	Carga inicial del shell	Verificar que el shell se renderiza correctamente al acceder a la aplicación.	El shell carga sin errores y muestra la vista principal.	Superado
CP-02	Descarga del remoteEntry	Validar la obtención del remoteEntry.js del microfrontend remoto.	El archivo se descarga con estado 200 y sin errores de red.	Superado
CP-03	Render de módulo remoto por ruta	Navegar a una ruta que carga un microfrontend remoto.	La vista del microfrontend se muestra correctamente.	Superado
CP-04	Render de componente dinámico	Cargar un componente remoto mediante loadRemoteModule.	El componente se renderiza sin errores.	Superado
CP-05	Fallback ante remote caído	Simular la indisponibilidad del remote.	Se muestra una pantalla de error controlada.	Superado
CP-06	Validación del manifest	Verificar la correcta lectura del manifest.json.	El manifest se carga y contiene URLs válidas.	Superado
CP-07	Cambio de versión del remote	Desplegar una nueva versión del remote sin redeplegar el shell.	El shell consume la nueva versión automáticamente.	Superado
CP-08	Navegación interna del remote	Validar rutas internas del microfrontend.	La navegación funciona sin errores ni recargas.	Superado
CP-09	Refresh en ruta remota	Recargar el navegador dentro de un remote.	La aplicación vuelve a cargar correctamente.	Superado
CP-10	Navegación entre remotes	Cambiar entre distintos microfrontends.	No se generan errores ni duplicación de vistas.	Superado

ID	Caso de prueba	Descripción	Resultado esperado	Estado
CP-11	Dependencias compartidas Angular	Verificar singleton de Angular entre shell y remotes.	No se cargan múltiples instancias de Angular.	Superado
CP-12	Compatibilidad de RxJS	Validar flujos reactivos entre shell y remotes.	Los observables funcionan sin conflictos.	Superado
CP-13	Librerías compartidas	Usar contratos comunes desde librerías compartidas.	No se presentan errores de tipado o ejecución.	Superado
CP-14	Carga de estilos globales	Validar la carga de CSS desde CDN.	Los estilos se aplican correctamente.	Superado
CP-15	Web Components	Validar carga y uso de webcomponents compartidos.	Los componentes se renderizan correctamente.	Superado
CP-16	Performance de carga	Medir tiempos de carga del shell y remotes.	La carga ocurre bajo demanda con tiempos aceptables.	Superado
CP-17	Cache y CDN	Validar actualización de versiones sin cache antiguo.	Se consume la versión correcta del remote.	Superado
CP-18	Seguridad CORS	Verificar políticas CORS entre shell y remotes.	No existen bloqueos del navegador.	Superado
CP-19	Resiliencia del sistema	Forzar fallo de un remote y validar continuidad.	El resto del sistema sigue operativo.	Superado
CP-20	Monitoreo y logs	Verificar registro de errores en monitoreo.	Los errores quedan registrados correctamente.	Superado

En el transcurso del proceso de certificación, se llegaron a validar los 40 casos de prueba, de los cuales alguno de ellos arrojó respuestas no esperadas según la matriz de prueba, por lo cual se tuvo que realizar correcciones que fueron presentados al equipo de QA. Posteriormente al retest de los casos de prueba, se pudo solucionar las incidencias reportadas. Dando un total de 40 casos exitosos de 40 presentados.

Finalmente se subieron los cambios al ambiente de STG, dónde se volvieron a correr los 40 casos de prueba que fueron exitosos al 100%. Por lo cual, solamente quedaba esperar la autorización del comité de despliegues de la entidad bancaria, para realizar el despliegue correspondiente al ambiente productivo.

3.2.5.7. Monitoreo y validación post-despliegue

En la etapa de monitoreo y validación post-despliegue, se hizo uso extensivo de la herramienta New Relic, permitiéndonos supervisar el comportamiento del sistema en un entorno de producción. Para validar el correcto funcionamiento de la nueva arquitectura, se analizaron métricas como la tasa de errores por integración del frontend, los tiempos de carga de vistas y la disponibilidad de los microfrontends.

Esta etapa inició desde el momento en el cual se hizo el pase a producción, dónde pudimos ingresar a New Relic para poder ver las gráficas y resultados obtenidos a partir del despliegue. Por fines de protección de la información se presentarán datos estimados de los resultados obtenidos.

En la Tabla 26 se evidencia una reducción progresiva de errores por integración tras la adopción de la arquitectura basada en módulos federados. Esto debido a que el despliegue de un microfront, no interrumpió el funcionamiento total del sistema, evitando errores por timeout.

Tabla 26*Tasa de errores por integración del frontend*

Periodo	Errores promedio por día
Antes de la migración	20
Semana 1 post-despliegue	11
Semana 2 post-despliegue	7
Semana 3 post-despliegue	4

En la Tabla 27 notamos una reducción en los tiempos de carga de las vistas, esto debido a que los microfronts ya poseen carga dinámica. Adicionalmente, estos disminuyen su peso debido a que solo se exportará los módulos a utilizar y ya no se encuentran obligados a construir todo el microfront. Los valores expuestos pueden ser variables con el pasar del tiempo, debido a la inclusión de nuevos módulos en los microfronts haciendo crecer el tamaño de estos.

Tabla 27*Tiempos de carga de vistas*

Tipo	Microfront	Antes (ms)	Después (ms)
shell	sh_public_a1	3500	2100
shell	sh_internal_a1	3400	2000
remote	mf_au_a1	3000	2200
remote	mf_in_a1	2950	2150
remote	mf_ds_a1	3100	2250
remote	mf_de_a1	3050	2200
remote	mf_mr_a1	3150	2300

remote	mf_pw_al	2900	2100
remote	mf_pr_al	2800	2050

En la Tabla 28 observamos una mejora en la disponibilidad del sistema, debido a la nueva arquitectura que nos permite aislar fallos por microfront, manteniendo operativo el sistema ante incidencias parciales. En promedio hubo una mejora de 0.6%, que en valores de disponibilidad es muy significativo en una entidad bancaria. Además, debe considerarse que la mejora observada se presenta de manera conjunta en los nueve microfrontends que conforman la solución, lo que genera un impacto acumulativo positivo en la disponibilidad y estabilidad del sistema en su conjunto.

Tabla 28

Disponibilidad de microfrontends

Tipo	Microfront	Antes (%)	Después (%)
shell	sh_public_al	99.4	99.9
shell	sh_internal_al	99.3	99.9
remote	mf_au_al	99.1	99.8
remote	mf_in_al	99.0	99.7
remote	mf_ds_al	99.2	99.8
remote	mf_de_al	99.1	99.7
remote	mf_mr_al	99.0	99.7
remote	mf_pw_al	99.3	99.9
remote	mf_pr_al	99.4	99.9

3.3. EVALUACIÓN

3.3.1. EVALUACIÓN ECONÓMICA / EVALUACIÓN COSTO-BENEFICIO

Al considerarse información sensible del proyecto, los costos mostrados en esta sección son estimados. Incluiremos costo de servicios en la nube, remuneraciones, servicios externos, etc. Los costos por uso de equipos informáticos, internet, energía, entre otros, estarán incluidos en las remuneraciones. La moneda utilizada en la evaluación será el dólar (USD).

En la Tabla 29 mostramos los costos asociados a los servicios de Azure que fueron utilizados en la implementación de los módulos federados. Consideremos que los costos presentados en los servicios de Azure suelen variar según el tiempo, el consumo y la configuración del servicio.

Tabla 29

Costos estimados de los servicios de Azure

Servicio	Costo mensual (USD)	Costo anual (USD)
Azure Kubernetes Service (AKS)	180.00	2,160.00
Azure Blob Storage	25.00	300.00
Azure CDN	60.00	720.00
Azure App Service	40.00	480.00
Azure Cache Redis	30.00	360.00
Total	335.00	4,020.00

En la Tabla 30 mostramos los costos asociados de los servicios de Azure junto a las remuneraciones percibidas por el equipo encargado de la implementación de la evolución tecnológica, estas son estimadas a partir de datos públicos recopilados por

Glassdoor para posiciones de TI en Lima, Perú (Glassdoor, 2025).

Tabla 30

Costos estimados de la implementación de la evolución tecnológica

Posición	Costo mensual (USD)	Cantidad	Tiempo (meses)	Costo total (USD)
Arquitecto Frontend	3500.00	1	3	10,500.00
Frontend Senior	2800.00	2	3	16,800.00
Technical Lead	3800.00	1	3	11,400.00
QA Tester	2200.00	1	3	6,600.00
Servicios Azure	335.00		3	1,005.00
Total				46,305.00

Con base en el informe consolidado de costos previamente sustentado, se determinó que el proceso de implementación de la evolución tecnológica representó una inversión total de USD 46,305.00, realizada a lo largo de los tres meses que tuvo de duración dicha implementación.

La propuesta de la optimización de la arquitectura es una evolución tecnológica, por lo cual, los beneficios se traducen en ahorros operativos, estos se consideran como beneficios calculados anuales, que serán enfocado en tres ejes, menor atención de incidencias y conflictos, menor tiempo y esfuerzo por despliegue y menor retrabajo y mayor estabilidad.

A continuación, en la Tabla 31 se presenta la estimación del ahorro generado por la reducción en la atención de incidencias como resultado de la implementación de la nueva arquitectura, alcanzando un ahorro anual total estimado de USD 10,800.00.

Tabla 31*Estimación de ahorro por menor atención de incidencias*

Concepto	Valor estimado
Incidencias promedio por mes	6
Tiempo promedio por incidencia	4 horas
Costo hora promedio del recurso	USD 35.00
Ahorro mensual estimado	USD 840.00
Ahorro anual estimado	USD 10,080.00

De forma sucesiva, en la Tabla 32 se presenta la estimación del ahorro generado por menor tiempo y esfuerzo por despliegue como resultado de la implementación de la nueva arquitectura, alcanzando un ahorro anual total estimado aproximado de USD 8,000.00.

Tabla 32*Estimación de ahorro por menor tiempo y esfuerzo por despliegue*

Concepto	Valor estimado
Despliegues promedio por mes	4
Horas ahorradas por despliegue	5 horas
Costo hora promedio del equipo	USD 33.00
Ahorro mensual estimado	USD 660.00
Ahorro anual estimado	USD 7,920.00

Posteriormente, en la Tabla 33 se presenta la estimación del ahorro generado por menor retrabajo y mayor estabilidad como resultado de la implementación de la nueva

arquitectura, alcanzando un ahorro anual total estimado aproximado de USD 8,000.00.

Tabla 33

Estimación de ahorro por menor retrabajo y mayor estabilidad

Concepto	Valor estimado
Correcciones evitadas por mes	3
Horas promedio por corrección	6 horas
Costo hora promedio del recurso	USD 37.00
Ahorro mensual estimado	USD 666.00
Ahorro anual estimado	USD 7,992.00

Finalmente, en la Tabla 34, se estima el total de beneficios anuales por conceptos relacionados a los objetivos específicos, donde se muestra el total acumulado de los ahorros conseguidos por la implantación de la solución, dándonos un total de aproximadamente USD 26,000.00.

Tabla 34

Beneficios anuales

Concepto	Ahorro estimado anual (USD)
Menor atención de incidencias y conflictos	10,000.00
Menor tiempo y esfuerzo por despliegue	8,000.00
Menor retrabajo y mayor estabilidad	8,000.00
Total	26,000.00

3.3.2. INTERPRETACIÓN DEL VAN Y TIR

Para la estimación del Valor Actual Neto (VAN) y la Tasa Interna de Retorno (TIR) se consideró los valores previamente definidos en la sección 3.3.1 correspondiente a la evaluación económica. Los costos asociados a la implementación del proyecto, considerando servicio de Azure y remuneraciones, fueron considerados como flujo de caja negativo y los consideramos como inversión inicial del proyecto.

Por otro lado, los beneficios económicos se estimaron a partir de los ahorros operativos generados por la evolución tecnológica, que se traduce a reducción de incidencias, menor retrabajo técnico y optimización de los procesos de despliegue. Dichos ahorros fueron considerados como flujos de caja positivos en los períodos posteriores a la implementación de la solución.

Para el análisis se consideró un horizonte de evaluación de tres años y una tasa de descuento del 10 %, adecuada para proyectos de TI. Con base en estos datos, se construyó el flujo de caja y se calcularon el VAN y la TIR, cuyos resultados permiten evaluar la viabilidad económica del proyecto desde una perspectiva académica.

Tabla 35
VAN y TIR

Concepto	Valor (USD)
Inversión inicial (Año 0)	-46,305.00
Efectivo neto – Año 1	26,000.00
Efectivo neto – Año 2	26,000.00
Efectivo neto – Año 3	26,000.00
Tasa de descuento	10%
TIR	33%
VAN	21,000.00

En la Tabla 35, notamos que la inversión inicial del proyecto, considerada como flujo de caja negativo, es compensada por los ahorros operativos anuales estimados en los períodos posteriores. El análisis realizado muestra un Valor Actual Neto (VAN) positivo, lo cual indica que los beneficios económicos proyectados superan los costos de implementación. Asimismo, la Tasa Interna de Retorno (TIR) obtenida es superior a la tasa de descuento considerada, reflejando una rentabilidad adecuada del proyecto. En conjunto, estos resultados permiten concluir que la implementación de la nueva arquitectura frontend resulta económicamente viable, justificando la inversión desde la eficiencia operativa.

Para el análisis se consideró un horizonte de evaluación de tres años y una tasa de descuento del 10 %, adecuada para proyectos de TI. Con base en estos datos, se construyó el flujo de caja y se calcularon el VAN y la TIR, cuyos resultados permiten evaluar la viabilidad económica del proyecto desde una perspectiva académica.

CAPÍTULO IV – REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

La experiencia desarrollada durante la optimización de la arquitectura frontend representó un reto considerable, debido a que el uso de módulos federados era una tecnología que se encontraba en una etapa inicial de adopción y existían pocos antecedentes que sirvieran como referencia para su implementación. Asimismo, la incorporación de este enfoque implicó una migración del framework Angular, modificaciones en la arquitectura frontend y la necesidad de una planificación adecuada en términos de tiempo y esfuerzo.

Durante el desarrollo de la solución se afrontaron diversos desafíos. Uno de los principales fue la limitada documentación existente sobre la arquitectura frontend previa, lo que dificultó el análisis y la definición de una arquitectura optimizada. Adicionalmente, se presentaron incompatibilidades entre versiones de dependencias como consecuencia de la migración de Angular. Sin embargo, el reto más relevante estuvo relacionado con la gestión del tiempo, ya que la implementación de la solución debía realizarse de manera paralela a otros desarrollos del sistema que no formaban parte del proceso de evolución tecnológica.

La incorporación de módulos federados también implicó desafíos técnicos y organizacionales. La migración generó una curva de aprendizaje en el equipo y conflictos de versiones durante la actualización de dependencias, así como algunos fallos de carga en tiempo de ejecución cuando ciertos microfrontends no estaban correctamente versionados. Estas situaciones exigieron reforzar las pruebas del equipo de QA y ajustar los mecanismos de validación antes del despliegue productivo.

En este contexto, la experiencia también permitió evidenciar la importancia del trabajo colaborativo dentro del equipo de desarrollo. La adopción de una arquitectura más

desacoplada requirió una mayor coordinación entre los integrantes, así como la necesidad de compartir conocimientos técnicos y alinearse en criterios de diseño y buenas prácticas. Este proceso fomentó una comunicación más constante y un enfoque colectivo para la resolución de problemas, fortaleciendo la capacidad del equipo para adaptarse a cambios tecnológicos.

Finalmente, la implementación de la solución permitió fortalecer los conocimientos en arquitecturas frontend modernas, así como en el uso de herramientas de integración y despliegue continuo. Además, fue posible evaluar el impacto técnico y económico de una solución tecnológica en un entorno real. Esta experiencia contribuyó de manera significativa al desarrollo profesional del equipo y del respectivo autor, no solo desde el aspecto técnico, sino también al brindar una visión más integral sobre la gestión de proyectos de software en contextos reales y complejos.

CAPÍTULO V – CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

La implementación de la nueva arquitectura frontend basada en microfrontends y módulos federados permitió optimizar la estructura tecnológica del sistema, logrando una solución más modular, escalable y alineada a las necesidades actuales de la entidad bancaria. La migración a Angular 15 facilitó la adopción de tecnologías modernas, mejorando el soporte, la seguridad y la mantenibilidad de la plataforma, y estableciendo una base sólida para su evolución futura.

En relación con el objetivo orientado a reducir la carga y complejidad de la arquitectura frontend, la adopción de módulos federados permitió suprimir la duplicidad de dependencias y habilitar la carga dinámica de microfrontends. Esta mejora se evidenció en la reducción de los tiempos de carga de vistas, que pasaron de valores cercanos a 3500 ms a aproximadamente 2100 ms, reflejando una arquitectura más liviana y eficiente en tiempo de ejecución.

Respecto al objetivo de incrementar la eficiencia de los procesos de despliegue, la nueva arquitectura habilitó la publicación independiente de microfrontends, eliminando la necesidad de recompilaciones integrales del sistema. Este cambio se reflejó en la disminución de errores por integración, reduciéndose de aproximadamente 20 a 4 errores promedio por día, lo que demuestra una mejora significativa en estabilidad y control de despliegues.

En concordancia con el objetivo específico de fortalecer la seguridad y actualizar la base tecnológica, la migración de Angular 12 a Angular 15 permitió contar con soporte

oficial y parches de seguridad vigentes. Asimismo, la nueva arquitectura federada contribuyó a mejorar la disponibilidad del sistema, incrementándose en promedio alrededor de 0.6%, lo que evidencia una mayor resiliencia operativa frente a fallos parciales.

Desde el punto de vista económico, la evaluación realizada evidenció que el proyecto es viable, al generar ahorros operativos que compensan la inversión inicial estimada en USD 46,305.00. Los indicadores obtenidos, como un Valor Actual Neto (VAN) aproximado de USD 21,000.00 y una Tasa Interna de Retorno (TIR) del 33 %, reflejan que la optimización de la arquitectura frontend es técnica y económicamente justificable.

En conclusión, el presente Trabajo de Suficiencia Profesional demuestra que la adopción de una arquitectura frontend basada en microfrontends y módulos federados constituye una alternativa viable para la modernización de sistemas en el sector bancario. Los resultados técnicos y económicos obtenidos evidencian mejoras en la estabilidad, el rendimiento y la sostenibilidad de la plataforma, cumpliendo con los objetivos planteados.

5.2. RECOMENDACIONES

Se recomienda continuar con la investigación de mejoras en la arquitectura de microfrontends, considerando enfoques donde se combinan capacidades nativas del navegador con las funcionalidades que ofrece Angular. Permittiéndonos dejar de depender de librerías externas y mejorando aún más el rendimiento de los sistemas, mejoras en soporte, mantenibilidad y evolución de las aplicaciones.

Asimismo, se sugiere establecer lineamientos organizacionales orientados a la actualización continua de las versiones de Angular, considerando que, de acuerdo con las políticas del framework, se liberan nuevas versiones de manera periódica. Mantener las versiones actualizadas permite aprovechar nuevas funcionalidades y mejoras del ecosistema, fortaleciendo la evolución, estabilidad y sostenibilidad de las aplicaciones en

el tiempo.

Finalmente, se sugiere documentar de manera continua la arquitectura frontend y los lineamientos técnicos establecidos, con el fin de facilitar la incorporación de nuevos integrantes al equipo y asegurar la correcta evolución de la solución en el tiempo. Una documentación actualizada permitirá preservar el conocimiento adquirido y garantizar la sostenibilidad de la arquitectura implementada.

5.3. FUENTES DE INFORMACIÓN

- Agh, H., García, F., Piattini, M., & Ramsin, R. (2020). Requirements for adopting software process lines. *Journal of Systems and Software*, 164, 110546.
<https://doi.org/10.1016/j.jss.2020.110546>
- Alekseev, P. O. (2024). Using Micro Frontends for Modular Architecture of Web Applications. *Universal Library of Engineering Technology*
https://www.uopenaccess.com/ulpages/fulltextUlete?PublishID=ULETE20240102_006
- Antunes, F., Lima, M. J. D., Pereira, M. A., Taibi, D., & Kalinowski, M. (2024). Investigating benefits and limitations of migrating to a micro-frontends architecture. *arXiv*. <https://arxiv.org/abs/2407.15829>
- Banco Central de Reserva del Perú. (2024). Reporte del Sistema Nacional de Pagos y del sector Fintech. Lima: BCRP. <https://www.bcrp.gob.pe/publicaciones/reporte-del-sistema-nacional-de-pagos/rspf-setiembre-2024.html>
- Bank for International Settlements. (2020). *Annual Economic Report 2020*. Basel: BIS.
<https://www.bis.org/publ/arpdf/ar2020e.htm>
- Banks, A., Porcello, E., & Raghavendra, S. (2021). *Learning Webpack: Understand how to use Webpack to bundle JavaScript applications*. O'Reilly Media.
- Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley Professional.
- Bass, L., Weber, I., & Zhu, L. (2021). *DevOps: A software architect's perspective* (2nd ed.). Boston, MA: Addison-Wesley. ISBN 978-0135866559.
- Bhasin, A. (2021). *Webpack 5: Up and Running*. O'Reilly Media.

- https://books.google.com/books/about/Webpack_5_Up_and_Running.html
- Chava, A. (2024). CI/CD and automation in DevOps engineering. *Asian Journal of Research in Computer Science*, 17(11), 73-80.
<https://doi.org/10.9734/ajrcos/2024/v17i11520>
- Donca, I.-C., Stan, O. P., Misaros, M., Gota, D., & Miclea, L. (2022). Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors*, 22(12), 4637. <https://doi.org/10.3390/s22124637>
- Emmanni, P. S. (2024). Architectural patterns and best practices for scalable enterprise applications with Angular. *Journal of Mathematical & Computer Applications*
<https://srcpublishers.com/index.php/mathematical-computer-applicatio/article/view/5349>
- Freeman, A., & Shreve, S. (2023). *Angular: Up & Running* (2nd ed.). O'Reilly Media.
- Hangai, R. H., Corte Real, L. F., Ferreira, R. C., Rosa, T. de O., & Goldman, A. (2025). Exploring Micro Frontends: A Case Study Application in E-Commerce. *arXiv*.
<https://arxiv.org/abs/2506.21297>
<https://doi.org/10.1016/j.infsof.2021.106571>
- Jeeri, K. (2024). Approaches to automating CI/CD processes in distributed teams. *The American Journal of Engineering and Technology*, 7(04), 97-104.
<https://doi.org/10.37547/tajet/Volume07Issue04-13>
- Jones, A. (2025). *Webpack Unleashed: A Thorough Exploration of Module Bundling*. Walzone Press.
https://books.google.com/books/about/Webpack_Unleashed_A_Thorough_Exploration.html?id=nFlCEQAAQBAJ
- Jørgensen, M. (2024). A systematic literature review on characteristics of the front-end phase of agile software development projects and their connections to project success. *Journal of Systems and Software*, 216, 112155.
<https://doi.org/10.1016/j.jss.2024.112155>
- Kurian George Cheripurathu, & Sanjeev Kulkarni. (2024). Integrating microservices and microfrontends: A comprehensive literature review on architecture, design patterns, and implementation challenges. *Journal of Scientific Research and Technology*, 2(7), 1–12. <https://doi.org/10.61808/jsrt115>
- Li, P. (2022). *Jira 8 Essentials: Effective project tracking and issue management with*

- enhanced Jira 8.21 and Data Center features (6th ed.). Packt Publishing Ltd.
https://books.google.com/books/about/Jira_8_Essentials.html?id=fQyYEAAAQBAJ
- Li, R., Liang, P., Soliman, M., & Avgeriou, P. (2021). Understanding Architecture Erosion: The Practitioners' Perceptive. En Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension (ICPC) (pp. 24–35). IEEE.
<https://doi.org/10.1109/ICPC52881.2021.00037>
- Mane, H. R., Kumar, A., Dandu, M. M. K., Goel, P., Jain, P. A., & Shrivastav, E. A. (2024). Micro Frontend Architecture With Webpack Module Federation: Enhancing Modularity Focusing On Results And Their Implications. Journal of Quantum Science and Technology, 1(4). <https://doi.org/10.63345/jqst.v1i4.95>
- Microsoft. (2023). What is Azure?. Recuperado de <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure>
- Modi, R., Lee, J., & Skaria, R. (2020). Azure for Architects: Create Secure, Scalable, High-Availability Applications on the Cloud (3rd ed.). Packt Publishing, Limited.
https://books.google.com/books/about/Azure_for_Architects.html?id=6ifwzQEACAAJ
- Mylsamy, S., & Mehrotra, D. T. (2025). Continuous Integration and Continuous Delivery (CI/CD) Pipelines. Journal of Quantum Science and Technology, 2(2), 495–504. <https://doi.org/10.63345/jqst.v2i2.284>
- Orange Editorial Board. (2025). Ultimate Bitbucket for Automating Workflows: Master Git, Automate Git Workflows, Secure Code Quality, and Supercharge CI/CD with Bitbucket Cloud and Smart Custom App Extensibility. Orange Education Pvt Limited.
https://books.google.com/books/about/Ultime_Bitbucket_for_Automating_Workflows.html?id=_pVqEQAAQBAJ
- Organisation for Economic Co-operation and Development. (2022). Protecting and empowering consumers in the purchase of digital content products. Paris: OECD Publishing. <https://doi.org/10.1787/5k49czlc7wd3-en>
- Osei, K. A., Cherkasova, V., & Oware, K. M. (2023). Digital banking: A bibliometric review. Future Business Journal, 9(1), 1–22. <https://doi.org/10.1186/s43093->

- Peltonen, P., Mezzalira, L., & Taibi, D. (2020). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Journal of Systems and Software*, 168, 110638. <https://doi.org/10.1016/j.jss.2020.110638>
- Peltonen, S., Mezzalira, L., & Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 136, 106571.
- Razavian, M., Tamburri, D. A., & Lago, P. (2020). Migration of legacy systems towards microservices: A systematic mapping study. *Journal of Systems and Software*, 169, 110708. <https://doi.org/10.1016/j.jss.2020.110708>
- Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide: The definitive guide to Scrum: The rules of the game*. Scrum.org.
- SCRUMstudy™. (2022). *A Guide to the Scrum Body of Knowledge (SBOK® Guide) – Fourth edition*
- Sommerville, I. (2016). *Software Engineering* (10th ed.). Boston, MA: Pearson Education. ISBN 978-0133943030.
- Thallapally, N. (2021). Implementing continuous integration and continuous deployment (CI/CD) pipelines. *International Journal of Science and Research Archive*. <https://ijsra.net/content/implementing-continuous-integration-and-continuous-deployment-cicd-pipelines>
- Thallapally, N. (2023). Best practices for enhancing front-end performance in modern web development. *Journal of Computer Science and Technology Studies*, 5(2), 11–18. <https://doi.org/10.32996/jcsts.2023.5.2.4>
- Tkachenko, O., Chechet, A., Chernykh, M., Bunas, S., & Jatkiewicz, P. (2025). Scalable front-end architecture: Building for growth and sustainability. *Informatica*, 49(1), 137–150. <https://doi.org/10.31449/inf.v49i1.6304>
- Verheyen, G. (2024). *Scrum: A Pocket Guide – 4th edition: A Smart Travel Companion*. Van Haren Publishing.
https://books.google.com.pe/books/about/Scrum_A_Pocket_Guide_4th_edition.html?id=hXcWEQAAQBAJ&redir_esc=y
- World Bank. (2024). *Digital Progress and Trends Report 2023*. Washington, DC: The World Bank. <https://documents1.worldbank.org/en/publication/documents->

5.4. GLOSARIO

- **Arquitectura Frontend:** Estructura que define la organización, interacción y comunicación de los componentes del frontend de una aplicación, considerando aspectos como modularidad, rendimiento, mantenibilidad y escalabilidad.
- **Microfrontend:** Enfoque arquitectónico que divide el frontend en aplicaciones independientes, cada una con responsabilidades específicas y ciclos de desarrollo y despliegue propios.
- **Módulos Federados (Module Federation):** Mecanismo que permite compartir y cargar módulos dinámicamente entre aplicaciones, reduciendo duplicidad de dependencias y favoreciendo el desacoplamiento.
- **Shell:** Aplicación principal que orquesta la carga de microfrontends, gestiona la navegación y consolida la experiencia global del usuario.
- **Remote (Microfrontend Remoto):** Aplicación independiente que expone funcionalidades específicas y es cargada dinámicamente por el shell en tiempo de ejecución.
- **Angular:** Framework frontend basado en TypeScript, utilizado para desarrollar aplicaciones web escalables y mantenibles en entornos empresariales.
- **Webpack:** Herramienta de empaquetado de módulos que gestiona dependencias, optimiza recursos y habilita la federación de módulos.
- **Carga Dinámica:** Técnica que permite cargar módulos solo cuando son necesarios, mejorando el rendimiento y reduciendo el tiempo de carga inicial.
- **Integración Continua (CI):** Práctica que integra cambios de código de forma frecuente, validándolos mediante procesos automatizados.
- **Despliegue Continuo (CD):** Proceso automatizado que libera versiones de una aplicación de manera frecuente y controlada hacia distintos entornos.
- **Pipeline:** Conjunto de etapas automatizadas que definen el flujo de construcción, validación y despliegue dentro de CI/CD.
- **Azure:** Plataforma de servicios en la nube de Microsoft que provee infraestructura y herramientas para el despliegue y operación de aplicaciones.

- **Azure Blob Storage:** Servicio de almacenamiento en la nube usado para alojar y exponer archivos estáticos de microfrontends.
- **Azure Kubernetes Service (AKS):** Servicio administrado de Kubernetes que permite orquestar contenedores de forma escalable y confiable en Azure.
- **New Relic:** Herramienta de monitoreo y observabilidad para analizar rendimiento, disponibilidad y errores en aplicaciones productivas.
- **Disponibilidad:** Indicador que mide el porcentaje de tiempo en que un sistema permanece operativo y accesible para los usuarios.
- **Tasa de Errores del Frontend:** Métrica que refleja la frecuencia de errores en la capa frontend, como fallos de carga o errores de JavaScript.
- **Tiempo de Carga:** Intervalo entre la solicitud del usuario y la renderización completa de una vista en el navegador.
- **VAN (Valor Actual Neto):** Indicador financiero que compara beneficios futuros con la inversión inicial, considerando el valor del dinero en el tiempo.
- **TIR (Tasa Interna de Retorno):** Indicador financiero que expresa la rentabilidad de un proyecto como la tasa para la cual el VAN es igual a cero.