UNIVERSIDAD TÉCNICA FEDERICO SANTA MARIA

SEDE VIÑA DEL MAR - JOSÉ MIGUEL CARRERA

ESCRITORIO DE TRABAJO CENTRALIZADO, BAJO ARQUITECTURA DE MICROFRONTEND

Trabajo de Titulación para optar al título de Técnico Universitario en Informática.

Alumno: Juan Pablo Rojas de la Fuente

Profesora guía: Lais San Martín Navarro

2026

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): X Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Escritorio centralizado bajo arquitectura microfrontend

Nombre del candidato(a): ___Juan Pablo Rojas de la Fuente

Carrera / Grado: __Técnico universitario en informática

Campus: _Sede Viña del mar_____ **Departamento:** __Departamento de electrotecnia e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Lais San Martín Navarro____, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

X El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: __13/01/2026_____ **Firma:** _____

Estudiante o Candidato(a):

Fecha: __13/01/2026_____ **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Resumen

El presente trabajo de titulación aborda el diseño y la modernización de la plataforma tecnológica operativa de la empresa, enfrentando la problemática de un ecosistema fragmentado y basado en arquitecturas monolíticas. Actualmente, esta situación genera una alta carga cognitiva para los colaboradores, ineficiencias operativas y riesgos de seguridad debido a la dispersión de accesos y credenciales.

La solución propuesta consiste en el diseño de un Escritorio de Trabajo Centralizado basado en una arquitectura de microfrontends. Este sistema actúa como una capa unificadora (shell) que integra dinámicamente las distintas funcionalidades de negocio, permitiendo modernizar la experiencia de usuario sin necesidad de reconstruir la totalidad de los sistemas legados (legacy).

Para garantizar la viabilidad técnica, el diseño se sustenta en una infraestructura *Cloud Native* utilizando Azure Kubernetes Service (AKS) para la orquestación elástica de contenedores Docker, y una estrategia de integración continua (CI/CD). A nivel de datos, se mantiene la robustez transaccional mediante Sybase 16, mientras que la seguridad se fortalece implementando un inicio de sesión único (SSO) gestionado por Azure Active Directory.

El resultado de este proyecto es una hoja de ruta arquitectónica que no solo resuelve la inconsistencia visual y funcional de las aplicaciones actuales, sino que dota a la organización de una plataforma escalable, segura y modular, preparada para responder ágilmente a los desafíos futuros de la compañía.

Para evidenciar la viabilidad de la solución, este documento se estructura en dos capítulos principales:

- Capítulo 1: Aspectos relevantes del análisis. Define la organización, detalla la situación actual y los problemas que genera el contexto tecnológico. A partir de ese diagnóstico, establece los objetivos del sistema propuesto y sus requerimientos funcionales clave.
- Capítulo 2: Aspectos relevantes del diseño. Presenta el *blueprint* de la solución, detallando la arquitectura de Microfrontend, el modelo de datos normalizado (modelo relacional), los recursos de hardware y software utilizados y los lineamientos de la Experiencia de Usuario a través del diagrama de menús y vistas.

Este informe sienta las bases técnicas para el desarrollo, proporcionando la visión detallada que guiará a la compañía en su camino hacia una operación más inteligente, eficiente y humana.

Índice de Contenido

Índice de Figuras

Índice de Tablas

Glosario

Microfrontend: Estilo arquitectónico donde una aplicación monolítica de *frontend* se descompone en pequeñas aplicaciones individuales y autónomas. Permite que cada módulo (o *microfrontend*) sea desarrollado, desplegado y gestionado por separado, aumentando la agilidad.

Shell Principal: El contenedor central o "cascarón" (host) de la arquitectura *Microfrontend*. Su rol es cargar y orquestar los módulos (*microfrontends*) de manera dinámica, asegurando la navegación consistente y gestionando la autenticación compartida (SSO).

Design System: Conjunto completo de estándares, patrones y componentes reutilizables (colores, tipografía, botones) utilizados para construir interfaces. Garantiza la Consistencia Visual en todos los *microfrontends*.

Single Sign-On (SSO) (Acceso Único): Mecanismo de autenticación que permite a un usuario iniciar sesión una sola vez para acceder a múltiples aplicaciones y servicios (en este caso, todos los *microfrontends*), utilizando un único proveedor de identidad

(**Azure AD**). **Active Directory (AD) / Azure AD:** El servicio de directorio e identidad basado en la nube. Actúa como el proveedor de identidad (IdP) que gestiona y valida las credenciales de los usuarios para el SSO.

API Gateway: Componente que actúa como un único punto de entrada para todas las solicitudes externas. Enruta las peticiones a los microservicios correctos, proporciona seguridad y desacopla al cliente de la complejidad interna de la arquitectura.

API Manager: Herramienta de gobernanza que administra las APIs (API Gateway). Se encarga de la seguridad, el *rate limiting*, el versionado y la recopilación de métricas y *logs* de todas las llamadas de los *microfrontends*.

Docker: Plataforma que utiliza la virtualización a nivel de sistema operativo para empaquetar código y todas sus dependencias en un contenedor ejecutable y portátil.

Imagen Docker: Un paquete inmutable y ejecutable que incluye todo lo necesario para correr una aplicación (código, *runtime*, librerías). Es la unidad de despliegue utilizada en los *pipelines* CI/CD.

Kubernetes (K8s): Sistema de código abierto para la automatización del despliegue, escalamiento y gestión de aplicaciones en contenedores (Docker). Es el orquestador del *cluster*.

AKS (Azure Kubernetes Service): Servicio gestionado de Kubernetes ofrecido por Microsoft Azure. Proporciona la Elasticidad y Escalabilidad necesaria para el entorno de producción de los *microfrontends*.

Pod: La unidad más pequeña de despliegue en Kubernetes. Un POD contiene uno o más contenedores Docker que se ejecutan como una única entidad lógica y compartida.

Gitlab: Repositorio de código fuente (Git) y plataforma integrada de DevOps utilizada para la gestión de CI/CD y la trazabilidad del código.

Pipeline: Conjunto de pasos automatizados (construcción, prueba y despliegue) que permite a los desarrolladores llevar el código desde el repositorio (Gitlab) hasta la implementación en el entorno de producción (AKS) de forma rápida y confiable.

CI/CD: (Integración Continua / Despliegue Continuo) Práctica de DevOps que automatiza el proceso de desarrollo de software, desde la integración del código (CI) hasta la entrega e implementación en producción (CD), permitiendo lanzamientos rápidos y frecuentes.

DevOps (Desarrollo y Operaciones)**:** Es una metodología de trabajo que busca automatizar e integrar los procesos entre los equipos de desarrollo de software (Dev) y los equipos de operaciones de TI (Ops). Su objetivo es acortar el ciclo de vida del desarrollo de sistemas, mejorando la calidad y permitiendo lanzamientos rápidos y confiables. En este proyecto, se implementa mediante Gitlab y los *pipelines* CI/CD que despliegan en AKS.

Visual Studio Code (VS Code): Editor de código fuente ligero, de código abierto y multiplataforma de Microsoft. Es la herramienta principal utilizada por los desarrolladores para escribir, depurar y probar el código de los *microfrontends* (Angular/React) y los servicios de *backend* (Java/APIs), integrándose con Gitlab y Docker.

Webpack 5 - Module Federation: Herramienta de arquitectura que permite a múltiples aplicaciones compartir código y exponer funcionalidades en tiempo de ejecución. Es clave para la carga dinámica de módulos y la autonomía de despliegue.

MFA - Autenticación de Múltiples Factores: Mecanismo de seguridad que requiere que el usuario demuestre su identidad proporcionando dos o más factores de verificación (ej., contraseña + código

móvil). En este proyecto, refuerza la seguridad del SSO gestionado por Azure AD, mitigando el riesgo de acceso no autorizado y resolviendo los *Problemas de Seguridad* detectados.

OAuth 2.0 (Open Authorization 2.0): Es un estándar de autorización que permite a los usuarios otorgar acceso limitado a sus recursos (información) a aplicaciones de terceros sin necesidad de compartir sus credenciales (contraseña). En el proyecto, se utiliza para validar los tokens de acceso emitidos por Azure AD y consumidos por los API Gateway.

mTLS (Mutual Transport Layer Security): Protocolo de seguridad que establece una autenticación mutua entre el cliente y el servidor. A diferencia del TLS estándar (donde solo el servidor se autentica), mTLS requiere que ambas partes (el microfrontend o el API Gateway) presenten y verifiquen un certificado digital. Esto garantiza que solo los servicios internos autorizados puedan comunicarse entre sí.

Introducción

Génesis del proyecto y alineación estratégica

El presente trabajo de titulación se enmarca en la iniciativa de transformación digital de la compañía, una institución con el firme propósito de ser un aliado clave en el cuidado de la salud de sus afiliados. Es una institución real, que, en este texto, se deja en el anonimato.

Históricamente, el compromiso con el servicio ha chocado con la realidad de una plataforma tecnológica fragmentada: aplicaciones dispersas, datos inconsistentes y la constante necesidad de navegar múltiples sistemas.

Esta realidad tecnológica se traduce directamente en ineficiencia operativa y en última instancia, afecta la calidad de atención que reciben los clientes. La misión de este proyecto, por lo tanto, se alinea con dos pilares estratégicos de la compañía: mejorar la experiencia del colaborador mediante la centralización de sus herramientas de trabajo, y modernizar las plataformas para garantizar la seguridad y la agilidad de respuesta en un mercado en constante cambio.

Planteamiento del problema y solución propuesta

La situación actual se caracteriza por la plataforma monolítica, que genera una carga cognitiva excesiva en los colaboradores: múltiples accesos en sus escritorios, contraseñas fragmentadas y la ausencia de una interfaz única y coherente. El problema central no es solo técnico, sino de Experiencia de Usuario (UX).

La solución propuesta es el diseño de un escritorio de trabajo centralizado. Este no es un nuevo "sistema" más, sino la capa unificadora que actúa como puerta de entrada inteligente. Para lograr esta unificación sin tener que demoler los sistemas críticos y heredados por distintas olas tecnológicas, se opta por la arquitectura de microfrontend. Esta metodología permite construir una plataforma elástica, segura y ágil, donde cada funcionalidad (o *microfrontend*) puede ser desarrollada, desplegada y mantenida por equipos distintos, garantizando la autonomía y la velocidad de entrega.

El objetivo principal de este informe es, por lo tanto, diseñar la hoja de ruta arquitectónica y de datos que haga factible esta transformación, asegurando la escalabilidad, la seguridad y la consistencia visual del nuevo ecosistema de trabajo.

1 Aspectos relevantes del análisis.

1.1 Descripción de la organización.

1.1.1 Antecedentes generales

La empresa donde se desarrolla este proyecto es una Institución de Salud Previsional (ISAPRE) que nació en 1981, inscribiéndose en el Fondo Nacional de Salud (FONASA) como una de las instituciones pioneras del sistema de salud previsional chileno. En la actualidad cuenta con más de 680.000 beneficiarios distribuidos a lo largo del país, en sus 60 sucursales desde Arica a Punta Arenas.

1.1.2 Portafolio de servicios y coberturas

Anualmente: más de 70.000 exámenes preventivos se realizan sin costo para nuestros beneficiarios, se crean programas de detección precoz de enfermedades a los que pueden acceder más de 100.000 afiliados y se gestionan más de 2 millones de asesorías (presupuestos hospitalarios, consultas y asesorías en canales presenciales y remotos), que permiten que nuestros beneficiarios saquen el máximo provecho a su plan, fortaleciendo nuestro compromiso con la prevención y la reducción del gasto del bolsillo.

El impacto de nuestra gestión también se refleja en otras cifras: se financian más de 3 millones de días de licencias médicas y se da cobertura a más de 10 millones de prestaciones ambulatorias, 140.000 cirugías hospitalarias y 46.000 tratamientos oncológicos.

También se garantiza el acceso a atenciones de alto impacto, especialmente en aquellas situaciones en que las personas más lo necesitan, se realizan más de 40 mil activaciones GES, llegando a tener más de 160 mil beneficiarios con GES activos, a quienes se les entrega más de 800 mil coberturas en prestaciones, a través de una red de más de 1.300 prestadores en convenio, asegurando acceso, calidad y oportunidad en su atención.

Para el caso de enfermedades catastróficas, anualmente más de 10.000 hospitalizaciones son cubiertas por el CAEC, lo que representa un respaldo financiero superior a $ 70.000 millones.

1.1.3 Desafíos estratégicos

A pesar de la magnitud de la cobertura y el volumen de prestaciones gestionadas, el desarrollo continuo de la industria de la salud y la necesidad de modernización interna impulsan a la Isapre a establecer Desafíos Estratégicos clave para los próximos años.

Textualmente se declaran los siguientes desafíos:

- *Concretar nuevos acuerdos con prestadores que premien la calidad por sobre la cantidad, para contener costos sin afectar la atención de nuestros beneficiarios.*
- *Mejorar significativamente la transparencia y comprensión de nuestros planes, para que sean accesibles y fáciles de entender para todos.*
- *Fortalecer la confianza de nuestros afiliados, mediante una experiencia de servicio más humana, oportuna y personalizada.*
- *Alinear nuestras capacidades digitales con una estrategia de salud preventiva, que nos permita acompañar a las personas y no solo reaccionar ante eventos.*
- *Ser un actor activo y propositivo en la discusión sobre el futuro del sistema de salud en Chile, desde una visión integradora y responsable.*

1.1.4 Estructura organizacional

A continuación, se presenta el organigrama de la compañía.

Figura 1-1 Organigrama de la empresa

Fuente: https://www.xxxxx.cl/informacion-institucional/

1.2 Descripción de situación actual.

Para entender la necesidad de este proyecto, primero se debe describir cómo opera la organización hoy en día, tanto a nivel operativo como tecnológico. Esta sección presenta una radiografía objetiva del entorno de trabajo de la compañía.

1.2.1 Contexto tecnológico actual

Actualmente, la compañía gestiona su operación mediante un ecosistema tecnológico extenso y heterogéneo, compuesto por 187 aplicaciones web. Esta dispersión obliga a que los usuarios deban administrar, conocer y mantener accesos directos independientes para cada uno de estos enlaces en sus escritorios de trabajo.

Estas herramientas no surgieron de un plan de diseño unificado, sino que son el resultado de implementaciones independientes realizadas a lo largo de los años por diversas áreas y proveedores externos.

En términos de ingeniería de software, este entorno se caracteriza por el predominio de aplicaciones monolíticas.

Nota técnica: Se entiende por arquitectura monolítica aquel modelo tradicional en el que la interfaz de usuario, la lógica de negocio y el acceso a datos se encuentran estrechamente acoplados en una única unidad de software autónoma.

Como consecuencia de este diseño, cada sistema opera en la práctica como una "isla" tecnológica: posee su propia base de datos, su interfaz y diseño visual particular y lo que resulta más crítico para la operación, su propio mecanismo de autenticación. Al carecer de una capa transversal de integración, no existe comunicación fluida entre estos aplicativos.

1.2.2 Descripción de flujo operativo de atención.

En la operación vigente, el ciclo de atención a un afiliado se desarrolla mediante la interacción secuencial con distintos aplicativos independientes. Cuando un ejecutivo (usuario) debe gestionar múltiples requerimientos para un mismo cliente (por ejemplo, gestionar la venta de un bono y posteriormente la recepción de una licencia médica), el procedimiento estándar implica los siguientes pasos operativos:

1. **Acceso y Autenticación:** El colaborador debe ingresar sus credenciales de acceso individualmente en cada aplicativo que la atención requiera, dado que no existe una sesión compartida entre los sistemas de bonos, licencias o beneficios.
2. **Ingreso de datos del afiliado:** En cada aplicativo abierto, se debe solicitar e ingresar nuevamente el identificador del cliente (RUT). Al no existir comunicación entre las interfaces, el dato ingresado en el sistema de bonos no se transfiere automáticamente al sistema de licencias médicas.
3. **Ejecución de la transacción:** El colaborador opera la transacción específica en la interfaz correspondiente, manteniendo abiertas las ventanas de los otros sistemas en paralelo para consulta.
4. **Reingreso manual ante cambios:** Si durante la atención es necesario cambiar de contexto o corregir un dato, el colaborador debe replicar manualmente la corrección en cada una de las plataformas activas para mantener la consistencia de la información visualizada.

1.2.3 La necesidad de transformación

Frente a un entorno que exige crecimiento y agilidad digital, la compañía ha reconocido la necesidad de mitigar estos riesgos estructurales. Por ello, se ha iniciado un proceso de transformación tecnológica orientado a mejorar la experiencia interna de los usuarios, reducir los tiempos de operación y optimizar la gestión de información. Este proceso busca migrar hacia soluciones modernas, modulares y sostenibles.

1.3 Problemas detectados

Basado en el análisis de la situación actual expuesto en la Sección 1.2, el diagnóstico tecnológico de la empresa revela una serie de problemas estructurales que impactan directamente tanto la gestión del área de TI como la productividad de los colaboradores. Estos problemas son la justificación principal para el desarrollo del sistema propuesto. En los siguientes puntos, 1.3.1 y 1.3.2, se exponen los problemas detectados y lo que se concluye.

1.3.1 Problemas de arquitectura y mantenimiento

La persistencia de múltiples arquitecturas monolíticas aisladas genera una deuda técnica que limita la capacidad de respuesta del área de TI:

- **Baja escalabilidad y rigidez:** Al estar la lógica de negocio acoplada en bloques monolíticos, no es posible escalar funcionalidades específicas de alta demanda sin duplicar la aplicación completa. Esto impide una gestión eficiente de los recursos ante peaks de operación.
- **Redundancia funcional y de desarrollo:** Debido a la falta de comunicación entre sistemas (las "islas" descritas en 1.2.1), se ha incurrido en la reprogramación constante de componentes comunes. Funcionalidades básicas como menús, formularios de búsqueda o validaciones de RUT existen repetidas veces en el código de las 187 aplicaciones, elevando innecesariamente los costos de mantenimiento.
- **Complejidad en el despliegue:** La actualización de un pequeño componente dentro de un monolito obliga a redesplegar todo el sistema, aumentando el riesgo de caídas de servicio y ralentizando la entrega de nuevas soluciones al negocio.

- **Riesgo de inconsistencia de datos:** El reingreso manual de información crítica (como el RUT del afiliado) al cambiar de aplicativo introduce un margen de error humano inevitable. Esto afecta la integridad de la información y puede derivar en la gestión de beneficios o licencias para el beneficiario incorrecto, generando un riesgo operacional directo para la compañía.

1.3.2 Problemas de seguridad y experiencia de usuario

Estos problemas tienen un impacto directo en el riesgo operacional y en la eficiencia de los colaboradores:

- **Vulnerabilidades en el mecanismo de acceso:** La existencia de mecanismos de autenticación independientes para cada aplicación impide aplicar políticas de seguridad unificadas. Al no estar integrados centralizadamente con el Directorio Activo (AD), el sistema es susceptible a riesgos por claves débiles, falta de rotación de contraseñas y dificultad para revocar accesos de exempleados en todas las plataformas simultáneamente.
- **Escasa interoperabilidad:** La falta de mecanismos estandarizados de comunicación impide el flujo de información en tiempo real entre los sistemas, lo cual es esencial para una atención a clientes eficiente.
- **Alta carga cognitiva y fatiga operativa:** La necesidad de administrar accesos directos para múltiples aplicaciones y adaptarse a interfaces visualmente inconsistentes obliga al colaborador a realizar un esfuerzo mental adicional. Esto no solo reduce la velocidad de atención, sino que aumenta la curva de aprendizaje para nuevos empleados.

Estos problemas estructurales derivan en ineficiencia operativa, retrabajo y menor trazabilidad de los procesos, lo que afecta directamente la capacidad de respuesta de la organización. Por lo tanto, el sistema propuesto (el escritorio centralizado) está diseñado para servir a dos grupos principales:

1. **Colaboradores internos:** Al proporcionar una experiencia de usuario consistente y centralizada, optimizando su eficiencia y reduciendo la carga cognitiva.

2. **Área de tecnología:** Al permitir la modernización de la arquitectura, la reducción de costos de mantenimiento y la mitigación de riesgos de seguridad a través de un sistema de acceso unificado.

1.4 Descripción del sistema propuesto

1.4.1 Definición de arquitectura microfrontend

Para abordar la complejidad descrita en el diagnóstico, la solución se fundamenta en el patrón de arquitectura conocido como microfrontends.

Este estilo arquitectónico extiende los conceptos de los microservicios al desarrollo del *frontend* (interfaz de usuario). En lugar de construir una única aplicación web gigante y monolítica, la arquitectura de microfrontends propone descomponer la interfaz en módulos pequeños, autónomos e independientes, donde cada uno representa una funcionalidad de negocio específica (por ejemplo: un módulo para "Venta de Bonos" y otro distinto para "Revisión de Licencias").

Estos módulos individuales son orquestados por un contenedor principal llamado shell o "Escritorio Anfitrión", en este caso se denominará “Escritorio Centralizado”, el cual es responsable de cargar estos microfrontends dinámicamente, unificarlos visualmente y gestionar la autenticación compartida. Esto permite que, para el usuario final, la experiencia se perciba como una sola aplicación cohesiva, aunque técnicamente esté compuesta por múltiples piezas de software independientes.

1.4.2 Justificación de su elección para este proyecto

La elección de esta arquitectura no es arbitraria, sino que responde directamente a las limitaciones del ecosistema actual de la empresa (descritas en el punto 1.3). La implementación de microfrontends es la estrategia idónea por las siguientes razones:

1. **Modernización Gradual (Estrategia Anti-Big Bang):** Dada la existencia de 187 aplicaciones legadas, intentar reescribir todo el sistema de una sola vez ("Big Bang") sería inviable en costos, tiempo y riesgos. Los microfrontends permiten una migración progresiva: se pueden desarrollar nuevas funcionalidades en tecnologías modernas (Angular/React) e integrarlas en el Escritorio Centralizado, mientras conviven temporalmente con los sistemas antiguos, sin detener la operación.
2. **Desacoplamiento y Autonomía de Equipos:** Al separar las funcionalidades en módulos autónomos, distintos equipos de desarrollo pueden trabajar en paralelo. Un equipo puede actualizar el módulo de "Afiliaciones" y desplegarlo en producción sin riesgo de romper el módulo de "Cobranzas". Esto resuelve el problema de la *rigidez en el despliegue* detectado en los monolitos actuales.
3. **Unificación de la Experiencia de Usuario (UX):** El principal dolor de los colaboradores es la fragmentación visual y de acceso. Esta arquitectura, a través del shell, fuerza una consistencia visual. Aunque los módulos vengan de distintos equipos, todos se cargan bajo el mismo marco, con el mismo menú y la misma barra de navegación, eliminando la carga cognitiva de "saltar" entre aplicaciones dispares.
4. **Resiliencia Operativa:** En un monolito, un error crítico puede botar toda la aplicación. En esta arquitectura, si el microfrontend de "Noticias" falla, el resto del Escritorio (ej. "Venta de Bonos") sigue funcionando perfectamente, garantizando la continuidad operativa del negocio.

1.4.3 Objetivo principal y objetivos específicos

El objetivo principal debe ser una solución directa al problema de la fragmentación tecnológica y la baja seguridad.

1.4.3.1 Objetivo principal

Implementar un escritorio de trabajo centralizado bajo una arquitectura de *microfrontend*, con el fin de unificar la experiencia de usuario e integrar las aplicaciones corporativas de la Isapre, mitigando la fragmentación tecnológica y los riesgos de seguridad asociados a los sistemas actuales antes descritos.

1.4.3.2 Objetivos específicos

1. Seguridad y acceso: Centralizar la autenticación de usuarios mediante un mecanismo de punto único de acceso corporativo (SSO= Single Sign-On, inicio de sesión único), integrando una capa de autorización interna basada en roles que gestione el acceso granular a las funcionalidades eliminando la necesidad de múltiples inicios de sesión y fortalezca la seguridad del sistema.
2. Experiencia de usuario: Asegurar que todas las aplicaciones integradas mantengan una consistencia visual y funcional mediante un diseño unificado.
3. Modularidad y agilidad: Implementar un modelo de microfrontend modular que permita el desarrollo autónomo y el despliegue independiente de cada componente, lo que mejorará la escalabilidad.
4. Interoperabilidad: Asegurar una comunicación eficiente y estandarizada entre los módulos integrados a través de APIs y eventos globales.
5. Nuevos lineamientos: Establecer las pautas para la arquitectura que se utilizará en los futuros desarrollos de la empresa.

1.4.4 Descripción de la arquitectura propuesta

Como solución a los desafíos operacionales y técnicos, se propone el desarrollo de un escritorio de trabajo centralizado basado en la arquitectura de microfrontend. Este sistema actuará como un entorno unificado que integra de forma dinámica las distintas aplicaciones corporativas de la compañía dentro de una única interfaz visual y funcional.
El modelo permitirá que los usuarios accedan a todas sus herramientas desde un solo punto de entrada, con autenticación compartida, navegación consistente y una experiencia visual uniforme.

El escritorio funcionará como un shell principal o contenedor que gestionará la carga y la comunicación entre los microfrontends. Cada uno de estos microfrontends representará una aplicación o módulo independiente (por ejemplo, gestión de afiliados, reportes médicos, reembolsos, atención comercial, entre otros).

La Figura 1-2 ilustra el flujo de acceso, autorización y la construcción dinámica del escritorio de trabajo centralizado, abordando directamente el objetivo principal de unificar la experiencia de usuario y mejorar la seguridad.

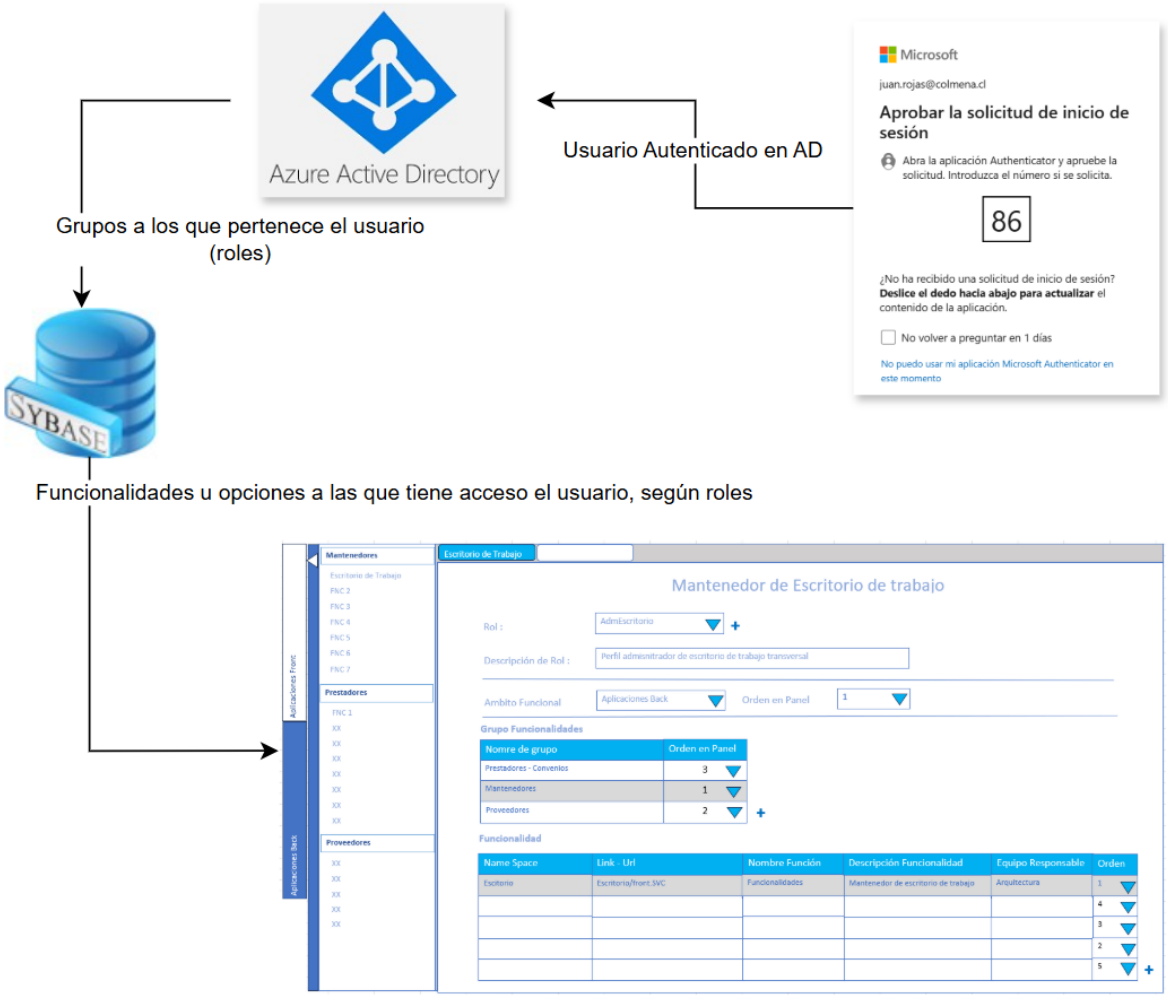

Figura 1-2 Esquema general de la solución propuesta

Fuente: Elaboración propia

El funcionamiento se desarrolla en las siguientes fases interconectadas:

1. **Autenticación centralizada (SSO):**

    - El proceso inicia cuando un usuario autenticado en AD intenta acceder al escritorio. El sistema no gestiona credenciales localmente, sino que delega la autenticación a Azure Active Directory (AD), el proveedor de identidad corporativo.

    - Como se muestra en la imagen (notificación de Microsoft), la autenticación puede incluir factores adicionales (MFA), fortaleciendo la seguridad del acceso.

    - Este flujo aborda el punto 1 de los objetivos específicos (ver punto 1.4.3.2) eliminando la duplicidad de inicios de sesión y las vulnerabilidades asociadas a sistemas de autenticación fragmentados.

2. **Autorización basada en roles:**

    - Una vez que Azure AD autentica al usuario, se obtienen los roles del usuario (ej. "Administrador de Escritorio", "Ejecutivo Comercial").
    - Estos roles se cotejan con una base de datos local que contiene las funcionalidades asociadas a los roles. Esta base de datos es gestionada a través del "Mantenedor de escritorio de trabajo", una interfaz que permite al administrador definir qué "Ámbitos Funcionales", "Grupos de Funcionalidades" y "Funcionalidades" específicas son accesibles para cada "Rol".
    - Esto garantiza que el contenido del escritorio centralizado se personalice según los permisos del usuario.

3. **Construcción dinámica del escritorio:**

    - El mantenedor de escritorio de trabajo (lado derecho de la figura) no solo define las autorizaciones, sino también la estructura de presentación. Establece el orden en panel para los grupos y funcionalidades, así como el Link Llamado (URL) de cada *microfrontend*.
    - Con esta información, el escritorio centralizado procede a la carga dinámica de los *microfrontends* autorizados para el usuario. Esto cumple con el punto 2 y 3 de los objetivos específicos (ver punto 1.4.3.2), ya que el *shell* (escritorio centralizado) orquesta la integración y presentación de las distintas Micro-Apps (ej. la de Angular) dentro de una interfaz unificada y coherente.

Este esquema valida la capacidad de la arquitectura para resolver la fragmentación de sistemas y la inconsistencia visual (indicada en 1.3 Problemas detectados), ofreciendo un entorno de trabajo seguro, personalizado y eficiente para los colaboradores de Isapre.

1.4.5 Pilares del sistema y beneficios

La propuesta arquitectónica se sustenta en los siguientes pilares técnicos y beneficios directos:

Modularidad: Separación de funcionalidades en módulos autónomos, lo que simplifica el desarrollo y el mantenimiento.

Autonomía de despliegue: Actualización independiente de cada módulo, permitiendo la entrega continua de valor sin afectar el conjunto del sistema.

Consistencia visual: Uso de un *Design System (diseño)* común para mantener la coherencia gráfica, mejorando la experiencia del usuario.

Interoperabilidad: Comunicación entre módulos mediante APIs y eventos globales, resolviendo la escasa comunicación entre sistemas legados.

Escalabilidad y elasticidad: Posibilidad de crecimiento de la infraestructura (*Elasticidad*) y la incorporación progresiva de nuevas aplicaciones (*Escalabilidad*) sin reestructurar el entorno.

En conjunto, la arquitectura propuesta permitirá reducir los costos de mantenimiento, mejorar los tiempos de despliegue y aumentar la satisfacción y eficiencia de los usuarios internos**.**

Comunicación global de eventos: El escritorio centralizado debe exponer un bus de eventos que permita a los microfrontends comunicarse entre sí (ej., el módulo "Búsqueda de Afiliado" debe notificar a todos los demás módulos la selección de un RUT).

1.4.6 Requerimientos funcionales

RF01 - Gestión de sesión única (SSO): El sistema debe implementar un mecanismo de autenticación centralizado que permita a los usuarios acceder a todos los *microfrontends* con un solo inicio de sesión.

RF02 - Carga dinámica de módulos: El escritorio centralizado debe ser capaz de cargar y descargar de forma dinámica cualquier *microfrontend* registrado sin requerir un reinicio del sistema.

RF03 - Sesión de cliente seleccionado: Cuando el escritorio esté activo en la pestaña de "Aplicaciones Front", el sistema contendrá la funcionalidad de poder buscar y seleccionar un cliente o beneficiario para el cual se generará la atención; cada trámite que el cliente necesite será resuelto por una distinta micro aplicación que se cargará en el menú base, donde a modo de ejemplo se puede mencionar algunos trámites que estarán disponibles:

- Emitir/Pagar un bono para atención médica
- Hacer la recepción de una licencia médica manual
- Ver la ficha del cliente
- Pagar cotizaciones.
- Etc.

RF04 - Persistencia de información: Cuando se tiene un cliente seleccionado para su atención, internamente el sistema dispondrá de variables globales y objetos en memoria que permitan la persistencia de la información seleccionada que será accesible por las micro aplicaciones, permitiendo de este modo la mensajería entre micro aplicaciones.

RF05 - Cambio de cliente seleccionado: Al momento de cambiar el cliente seleccionado para atención, el sistema deberá limpiar todo referente a la atención anterior, eliminando las instancias de micro aplicaciones cargadas en el menú e internamente variables globales, objetos en memoria, quedando disponible para una nueva atención.

RF06 - Administración del catálogo de microfrontends: El Administrador del escritorio debe poder registrar, actualizar y desactivar las referencias de las Micro-Apps (es decir, sus NameSpace y LinkLlamado) en la base de datos de orquestación.

RF07 - Gestión de autorización: El sistema debe utilizar los roles del usuario (obtenidos de Azure AD) para filtrar la visualización de los grupos de funcionalidades en el menú de navegación (basado en la tabla ROLFUNCIONALIDAD).

RF08 – Habilitación de funcionalidades programada y controlada: El sistema debe permitir al Administrador del Escritorio programar el momento exacto (fecha y hora) en que cada Micro-App registrada en el catálogo estará disponible para los colaboradores (usuarios internos). Esto asegura que los usuarios vean el enlace de la nueva funcionalidad (*LinkLlamado*) *exactamente* a la hora

prometida (ej., lunes a las 9:00 AM), maximizando la confianza en el sistema y previniendo el acceso prematuro a herramientas que, aunque ya estén desplegadas en AKS (Azure Kubernetes Service), no deben estar activas para el negocio.

1.4.7 Arquitectura de despliegue y ejecución

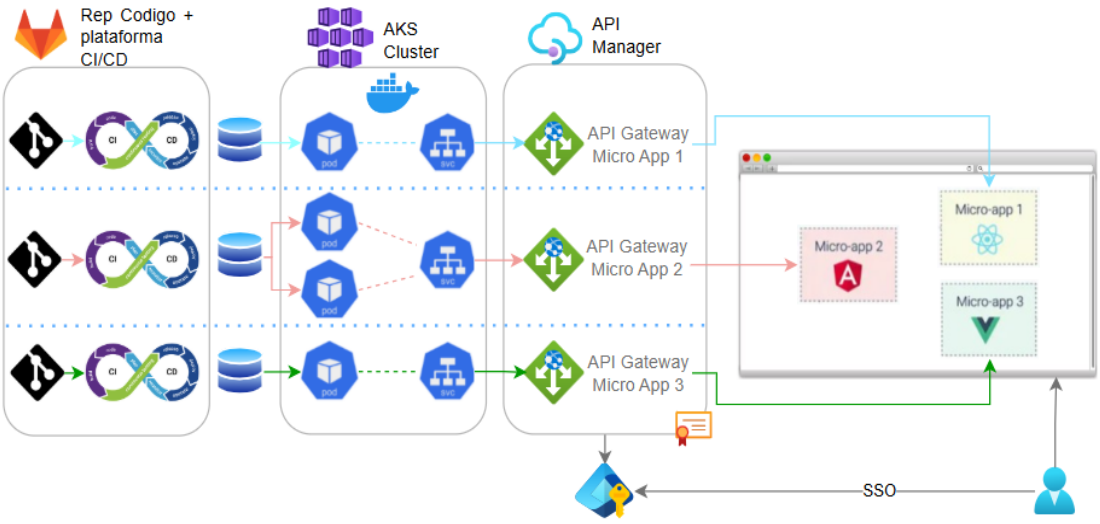

Figura 1-3 Arquitectura de despliegue y ejecución
Fuente: Elaboración propia

La Figura 1-3 representa la arquitectura de ejecución (*runtime*) y despliegue (*DevOps*) del sistema, diseñada para garantizar la modularidad, la autonomía y la escalabilidad de cada *microfrontend*. Esta arquitectura se basa en tres pilares interconectados:

A. Pilar de ingeniería de software (DevOps y CI/CD)

Este pilar garantiza la agilidad y la autonomía de despliegue para cada equipo de desarrollo.

- **Repositorio y pipeline (Gitlab):** Gitlab sirve como repositorio de código central y plataforma de integración continua y despliegue continuo (CI/CD).

- **Proceso:** Cuando el desarrollador hace commit en el repositorio de una Micro-App, el pipeline automatizado compila el código (Java 21 para APIs, Angular/React para frontend), ejecuta las pruebas unitarias, genera la imagen Docker del componente y la despliega.
- **Contenedorización (Docker):** El uso de Docker asegura que cada Micro-App sea un paquete aislado e inmutable, listo para ser ejecutado en cualquier entorno, garantizando la consistencia.

B. Pilar de orquestación y gobernanza

Este pilar gestiona la infraestructura elástica y expone las APIs de manera segura y controlada.

- **Cluster de orquestación (AKS):** Los contenedores Docker se despliegan en el Cluster AKS (Azure Kubernetes Service). AKS actúa como el motor de *runtime*, proporcionando la elasticidad y escalabilidad necesaria, permitiendo que la plataforma crezca o decrezca automáticamente según la demanda.
- **API manager y gateway:** Los *pods* (instancias de aplicaciones) de cada micro-App exponen sus servicios a través de un API Gateway dedicado. Toda esta capa es gobernada por un API Manager.
    - El API Gateway actúa como el único punto de entrada a las APIs, desacoplando al cliente de la complejidad interna de los microservicios.
    - El API Manager centraliza la seguridad, el *rate limiting* y el registro de métricas y *logs*, garantizando la gobernanza técnica de la solución.

C. Pilar de presentación y seguridad

Este pilar se centra en la experiencia del colaborador interno o usuario y el cumplimiento del objetivo de seguridad.

- **Acceso y SSO:** El usuario accede al escritorio de aplicaciones y la autenticación se realiza mediante SSO (Single Sign-On) a través del IdP corporativo (Azure AD). Esto resuelve las vulnerabilidades críticas detectadas en el acceso (ver punto 1.3.2).
- **Interacción unificada:** El escritorio de aplicaciones actúa como el *shell* principal, que integra de forma dinámica los *microfrontends* (ej. micro-app 1 en React, micro-app 2 en Angular, etc.).

- **Consumo de APIs:** El escritorio y las micro-apps utilizan el *token* de acceso emitido por el SSO para consumir las APIs de negocio a través de los API Gateway.

1.4.8 Casos de uso

Actor principal: Para este contexto se define que el escritorio será sólo de uso interno, diferenciando en dos grupos de actores principales:

- Usuario de negocio
- Usuario Administrador de accesos

Actores secundarios:

- Azure Active Directory (Proveedor de Identidad)

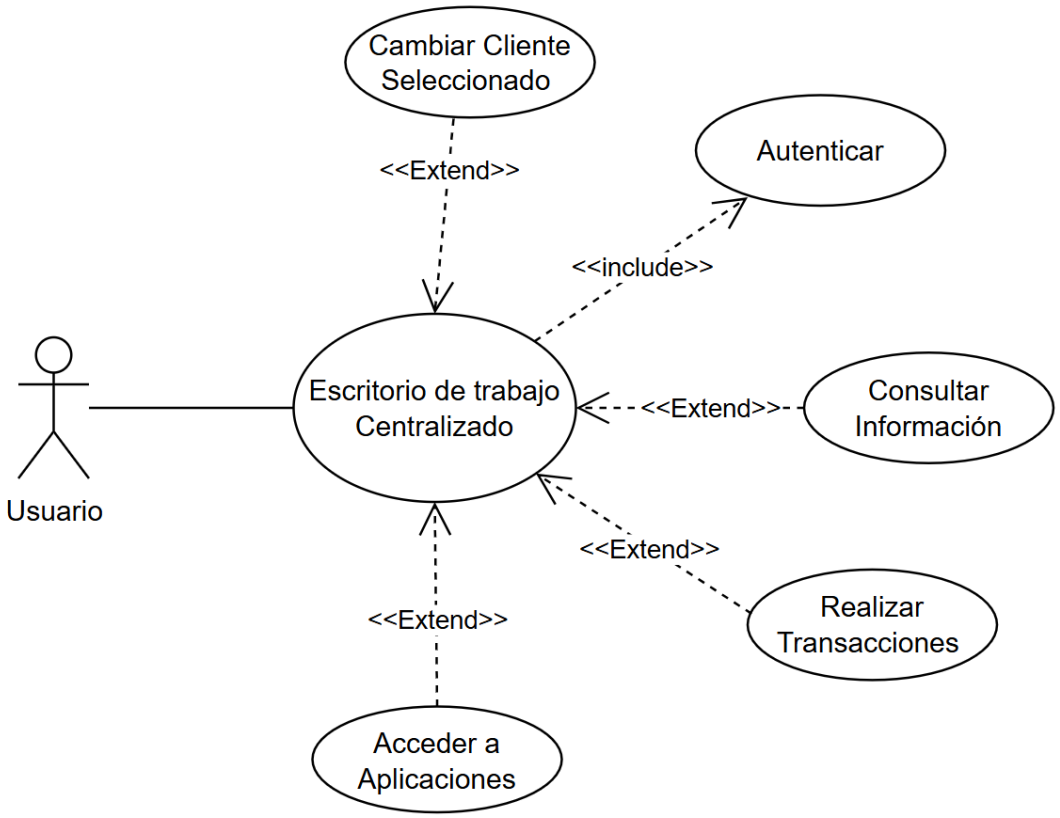

Figura 1-4 Casos de uso Escritorio Centralizado

Fuente: Elaboración propia

CU01 – Autenticarse en el sistema (SSO)

Actor: Usuario Interno

Descripción:

Permite que el usuario acceda al escritorio mediante autenticación corporativa basada en Azure AD.

Flujo principal:

1. El usuario ingresa a la URL del Escritorio Centralizado.
2. El sistema redirige automáticamente al login de Azure AD.
3. El usuario ingresa sus credenciales corporativas.
4. Azure AD valida la identidad (incluye MFA cuando aplica).
5. El sistema recibe el token de acceso.
6. El escritorio se carga con los permisos y funcionalidades del usuario.

Postcondición:

El usuario tiene acceso seguro y personalizado al escritorio.

CU02 – Consultar Información del cliente

Actor: Usuario Interno

Descripción:

Permite obtener la ficha unificada del cliente/beneficiario.

Flujo Principal:

1. El usuario ingresa a “Aplicaciones Front”.
2. Busca un cliente mediante RUT.
3. El sistema recupera los datos desde las APIs.
4. La información se presenta en la Ficha Consolidada.
5. Las variables internas quedan actualizadas.

Postcondición:

El cliente queda seleccionado como “cliente activo de la sesión”.

CU03 – Realizar transacciones para el cliente

Actor: Usuario Interno

Descripción:

Permite ejecutar trámites relacionados con el cliente seleccionado.

Flujo Principal:

1. El usuario selecciona un cliente (CU02).
2. Selecciona una funcionalidad transaccional del menú.
3. El escritorio carga dinámicamente el microfrontend correspondiente.
4. El microfrontend obtiene los datos del cliente.
5. El usuario completa la transacción.
6. La API confirma la operación.

Postcondición:

La transacción queda registrada y auditada.

CU04 – Acceder a aplicaciones (microfrontends)

Actor: Usuario Interno

Descripción:

Permite acceder a diferentes microaplicaciones sin iniciar sesión nuevamente.

Flujo Principal:

1. El usuario accede al menú.
2. El sistema muestra solo las aplicaciones autorizadas.
3. El usuario selecciona un módulo.
4. El microfrontend se carga dinámicamente.
5. El usuario interactúa con la aplicación.

Postcondición:

El módulo queda disponible para la sesión actual.

CU05 – Cambiar cliente seleccionado

Actor: Usuario Interno

Descripción:

Permite modificar el cliente activo de la sesión.

Flujo Principal:

1. Usuario selecciona “Cambiar Cliente”.
2. El sistema pide confirmación.
3. Limpia todas las instancias activas de microfrontends.
4. Limpia variables internas.
5. El usuario selecciona un nuevo cliente.

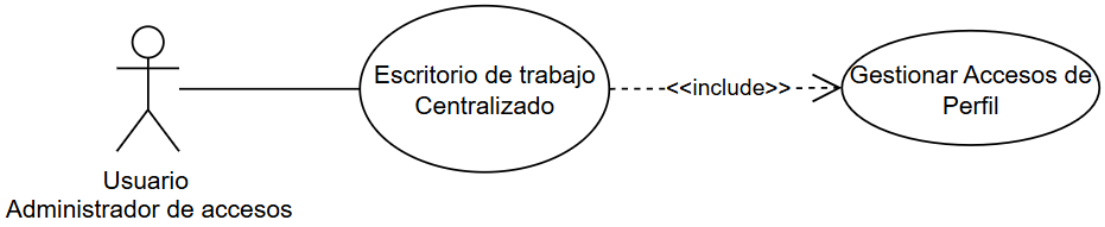

Figura 1-5 Caso uso Gestionar Accesos de Perfil

Fuente: Elaboración propia

CU06 – Gestionar Accesos de Perfil (Administrador de Escritorio)

Actor: Administrador

Descripción:

Permite administrar roles, grupos y funcionalidades.

Flujo Principal:

1. El administrador accede al Mantenedor del Escritorio.
2. Consulta y edita roles, grupos y funcionalidades.
3. Configura orden, URLs, ámbitos.
4. El sistema actualiza la base de datos.

Postcondición:

La estructura del escritorio queda actualizada.

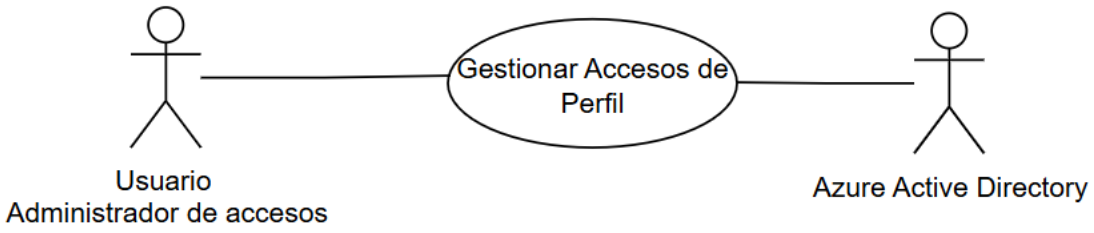

Figura 1-6 Caso uso Gestionar Perfil de Usuarios

Fuente: Elaboración propia

CU07 – Gestionar perfil de usuario

Actores: Usuario administrador de accesos (Principal), Azure Active Directory (Secundario).

Descripción: Permite revisar y gestionar información personal del usuario, asegurando que los cambios se sincronicen con el proveedor de identidad corporativo.

Flujo Principal:

1. El usuario abre “Mi Perfil”.
2. El sistema consulta y muestra la información vigente obtenida desde Azure Active Directory.
3. El usuario modifica los parámetros permitidos.
4. El sistema valida y registra los cambios impactando directamente en Azure AD.

Postcondición: La ficha del usuario queda actualizada en el directorio corporativo.

1.4.9 Información que manejará

Salidas (descriptivo básico)

El sistema generará salidas de información principalmente a nivel de interfaz de usuario y telemetría:

- **Escritorio centralizado:** Interfaz unificada que presenta la ficha consolidada del cliente (integrando datos de diferentes *microfrontends*).
- **Logs y métricas:** Datos generados por el API Manager y el cluster AKS sobre la latencia, el tráfico y los errores de las APIs, esenciales para la gobernanza del sistema.

Entradas (descriptivo básico)

Las entradas del sistema serán gestionadas a través de la capa de APIs:

- **Credenciales de acceso:** Datos de autenticación proporcionados por el usuario, validados por el sistema de Identidad (Azure AD).
- **Datos transaccionales:** Información capturada a través de los formularios de cada *microfrontend* (ejemplo número de licencia médica, Rut del cliente, numero de caso, etc.), enviada vía APIs para ser registrada en el *cluster* de base de datos.

2 Aspectos relevantes del diseño

2.1 Entidades de datos.

A continuación, se presenta el modelo relacional que da soporte al Escritorio Centralizado. Este diagrama detalla las entidades encargadas de la orquestación de los microfrontends y la gestión de permisos de seguridad.

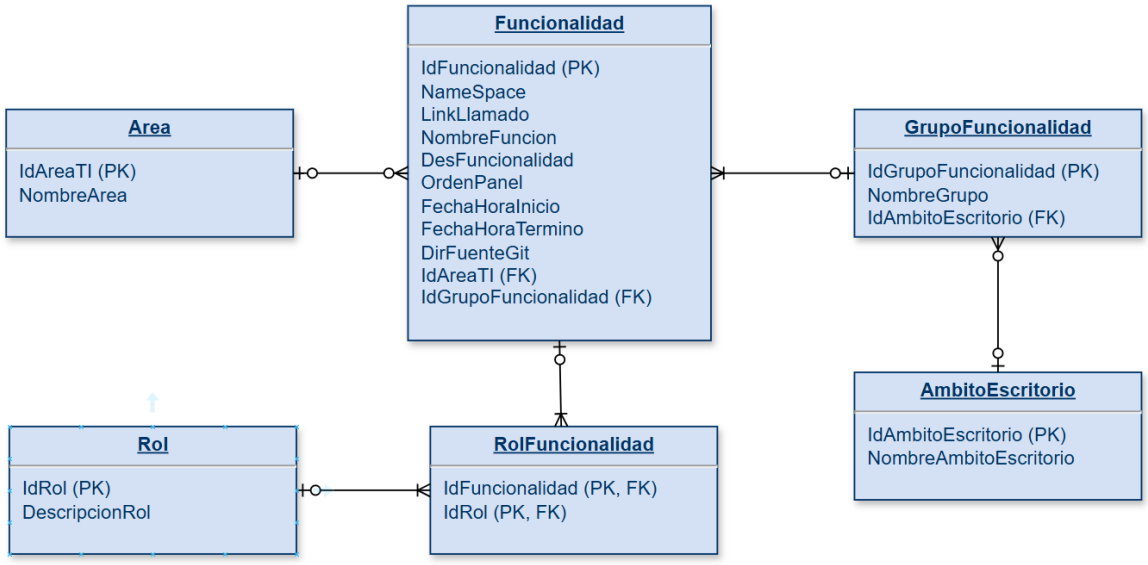

Figura 2-1 Modelo Relacional

Fuente: Elaboración propia

2.2 Descripción de los recursos utilizados

2.2.1 Configuración del sistema

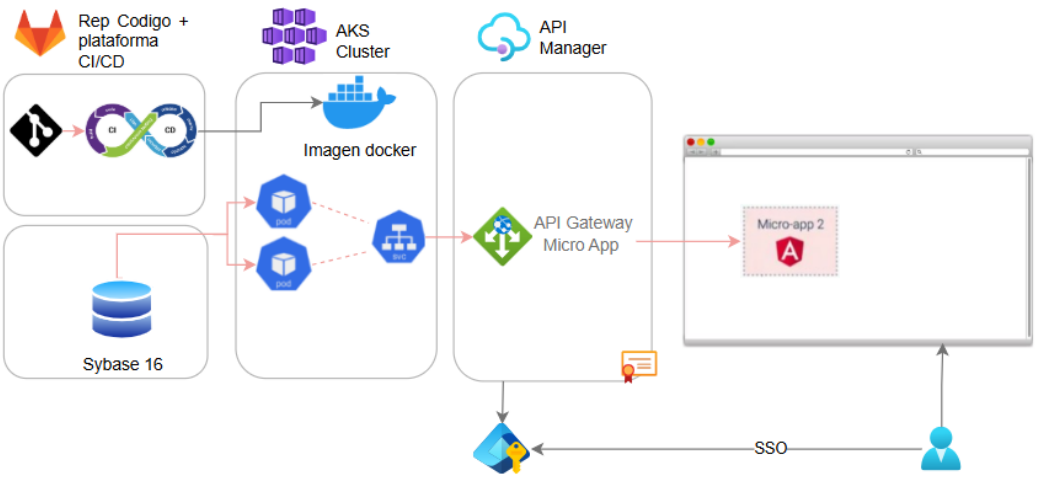

Figura 2-2 Recursos utilizados y sus funciones

Fuente: Elaboración propia

La figura 2-2 muestra una representación de los recursos usados en la solución y las componentes que la conforman. A continuación, se comentan algunas de sus características principales.

2.2.1.1 Entorno de aplicación (plataforma elástica)

El diseño del nuevo escritorio centralizado se apoya en una infraestructura de nube (Cloud Native), seleccionada no solo por su capacidad técnica, sino por su facilidad de dar elasticidad tanto horizontal como vertical, permitiendo ajustar la capacidad según la demanda.

1. Plataforma de cómputo y orquestación: Azure Kubernetes Service (AKS)

AKS es el motor que alberga el escritorio, asegurando que la plataforma esté siempre disponible y siempre responda con la velocidad que se necesita. AKS es el componente que elimina la preocupación por los límites de capacidad.

- **Elasticidad:** AKS elimina el miedo a los peak de demanda que consumen todos los recursos el sistema anterior. Si una funcionalidad específica (*microfrontend*) experimenta un

aumento repentino de usuarios, la plataforma automáticamente genera nuevas instancias (*Pods*) para absorber la carga. Esto garantiza una experiencia de usuario fluida y sin demoras, independientemente del volumen de trabajo.

- **Resiliencia:** El sistema está diseñado para auto-protegerse. AKS monitorea la "salud" de cada componente las 24 horas. Si un módulo falla, la plataforma aísla el error y reinicia automáticamente el contenedor afectado. Esto se traduce en una tranquilidad operacional crucial, ya que el escritorio principal permanece activo y funcional, protegiendo al usuario de interrupciones generalizadas.

2. Almacenamiento primario y unidad de despliegue (Contenedores Docker)

Todos los componentes de las aplicaciones (servicios y *microfrontends*) serán empaquetados como imágenes Docker la cual proveerá de un espacio adecuado para los componentes a utilizar.

Se utilizará contenedor Docker ya que es la solución a la clásica frustración de los desarrolladores: *"¿Por qué funciona en mi equipo, pero no en producción?", donde se busca:*

- **Garantía de consistencia:** Cada aplicación (*shell y microfrontends*) se empaqueta en una Imagen Docker, que es una caja sellada e inmutable que contiene todo lo necesario para ejecutar el código (librerías, sistema operativo, configuración).
- **Portabilidad Inmediata:** Almacenadas en el Azure Container Registry (ACR), estas imágenes son la unidad estandarizada de despliegue. Esto garantiza que lo que se prueba en desarrollo es exactamente lo que se ejecuta en el ambiente productivo. Desde una perspectiva humana, esto elimina las interminables horas de solución de problemas de ambiente y acelera significativamente el tiempo que el código tarda en llegar a manos del usuario.

3. API Manager (Puerta única)

El uso de un API Manager es estratégico porque transforma las APIs de simples puntos de conexión técnicos en productos digitales controlados y escalables. Actúa como el guardián disciplinado y el único punto de entrada confiable al ecosistema de microservicios, desacoplando la complejidad interna de la experiencia del usuario.

Su rol se enfoca en dos tareas críticas:

1. **Seguridad centralizada y gobernanza:** Centraliza la seguridad, obligando a que toda la comunicación pase por filtros de validación robustos (incluyendo OAuth 2.0 y mTLS). Esto garantiza que solo el tráfico genuino y autorizado llegue a los servicios, blindando la arquitectura contra vulnerabilidades.

2. **Rendimiento y resiliencia:** Gestiona el flujo de tráfico mediante balanceo de carga y políticas de limitación de tasa (*Rate Limiting*). Esto previene la sobrecarga de los microservicios ante peak de demanda, asegurando que la plataforma mantenga una experiencia fluida y sin demoras para el colaborador.

4. Repositorio y CI/CD (Gitlab):

Gitlab será utilizado tanto para la custodia del código fuente como para la implementación de las prácticas de DevOps. Gitlab gestionará los *pipelines* de Integración continua y despliegue continuo (CI/CD) que automatizarán la construcción, prueba. Por este medio se implementan las imágenes Docker directamente en el cluster AKS, asegurando de este la agilidad y los tiempos de despliegue reducidos en el ciclo de desarrollo.

5. Autenticación y SSO

El sistema se comunicará directamente con el Active Directory (AD) corporativo, resolviendo el problema de las vulnerabilidades y fragmentación de accesos e implementando el entorno de SSO (Single Sign-On) para el punto único de entrada.

2.2.1.2 Infraestructura de servidores de base de datos

El sistema se conecta a la instancia de base de datos Sybase 16, la cual opera bajo una arquitectura física dedicada y de alta disponibilidad, esencial para los datos transaccionales de la Isapre.

- **Arquitectura física:** El *cluster* de base de datos se basa en una modalidad activo-pasivo con replicación a nivel de disco. Consta de dos *sites (centros de datos)* independientes y equivalentes: Server Alpha 16 (nodo productivo primario, Site Los Militares) y Server Beta 16 (nodo de contingencia y site contingencia).

- **Procesamiento y memoria:** Se utilizan servidores basados en procesadores HP Itanium II bajo el sistema operativo propietario HPUX. Cada nodo cuenta con 16 cores de procesamiento y 128 Gigas de memoria RAM.
- **Almacenamiento (storage):** El almacenamiento es de 7 Tera Bytes (TB), gestionado por tecnología 3PAR, con todos los discos del tipo SSD para asegurar la velocidad y el rendimiento transaccional.
- **Respaldo y recuperación:** Se implementan políticas de respaldo automatizado, y para las instancias productivas se obtiene una nueva imagen diariamente por medio de snapshots, la cual se almacena por 7 días para recuperación.

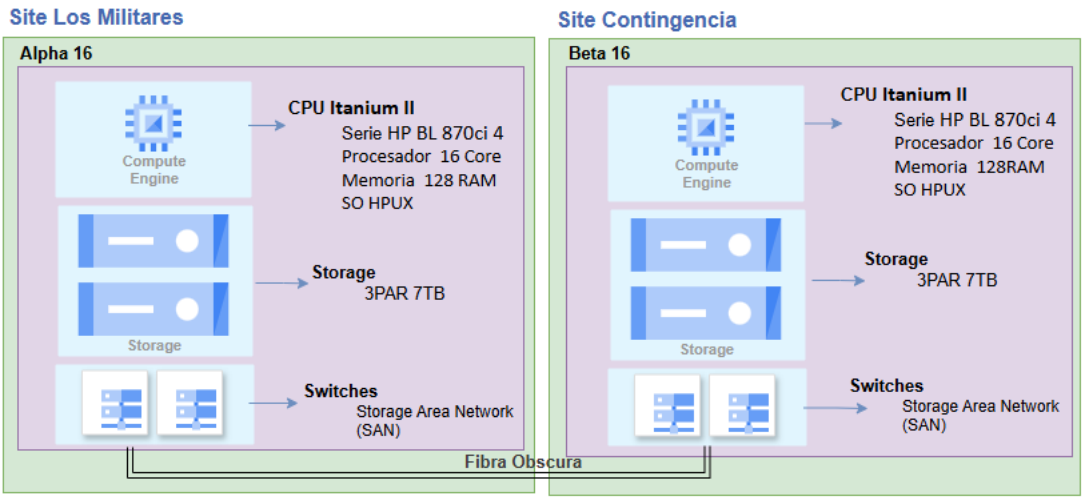

Figura 2-3 Arquitectura física de base de datos

Fuente: Elaboración propia

2.2.1.3 Entorno de desarrollo y pruebas (local/aislado)

El siguiente hardware es el estándar mínimo requerido para la estación de trabajo de cada desarrollador, garantizando que el entorno local pueda soportar la ejecución simultánea del shell principal o escritorio de trabajo y de múltiples microfrontends mediante Docker Desktop.

Este ambiente es necesario para la autonomía y las pruebas locales antes de la integración al pipeline CI/CD.

1. Estación de trabajo (Hardware)

El siguiente *hardware* es el estándar mínimo para el desarrollo, capaz de soportar la compilación y la contenerización local:

Tabla 2-1 Estación de trabajo local

Componente	**Especificación Mínima**	**Justificación Técnica**
Procesador (CPU)	Intel Core i7 (12ª generación o superior) o AMD Ryzen 7.	Necesario para la compilación de código Java/Angular y el *overhead* de Docker Desktop.
Memoria RAM	32 GB DDR4 (Recomendado).	Esencial para la ejecución simultánea del *shell* local, el *microfrontend* y el contenedor de base de datos *sandbox*.
Almacenamiento	1 TB SSD.	Crítico para la velocidad de I/O en la compilación y el manejo eficiente de imágenes Docker y repositorios de código.
Sistema Operativo	Windows 11 Pro o macOS Sonoma/Ventura.	Requerido para la seguridad corporativa y la compatibilidad optimizada con Docker Desktop.

Fuente: Elaboración propia

2. Ambiente de pruebas (Sandbox Local)

Este ambiente utiliza Docker compose para orquestar los servicios en la máquina local, permitiendo pruebas de integración aisladas:

- **Contenerización local:** Se utiliza Docker Desktop para ejecutar una instancia local del API Gateway y de los *microservicios* Java.
- **Base de datos** server de QA de la compañía, bajo el mismo sistema operativo y versión con una tercera parte de los recursos de productivo.

2.2.2 Software utilizado

El desarrollo del escritorio de trabajo centralizado se basa en un *stack* tecnológico moderno y distribuido, seleccionado específicamente para soportar la arquitectura modular de *microfrontend* y los requisitos de seguridad y escalabilidad de la Isapre.

1. Capas Core de la arquitectura (frontend /shell)

El frontend, que constituye la capa de presentación unificada, es el pilar de la nueva experiencia de usuario.

- **Framework principal (Angular):** Seleccionado como el *framework* principal para el desarrollo de la *shell* o contenedor central y para la construcción de los *microfrontends* individuales. Angular proporciona la robustez y estructura necesaria para aplicaciones de gran escala. Sin embargo, como prueba de concepto, se generará al menos una micro aplicación con la capacidad de convivir e interactuar en el framework React, ya que ambos frameworks son del tipo SPA (Single Page Application), que es una de las características que permiten la tecnología microfrontend.
- **Orquestación de microfrontends (Webpack 5 - Module Federation):** Esta es la herramienta clave que habilita la arquitectura modular. Module Federation permite la carga dinámica de módulos y el aislamiento de las aplicaciones de *microfrontend* en tiempo de ejecución. Esto garantiza la autonomía de despliegue y la flexibilidad para integrar módulos desarrollados independientemente.
- **Estilización y consistencia:** Se empleará CSS para el estilo, adhiriéndose a un sistema de diseño corporativo que garantiza la consistencia visual uniforme a través de todos los módulos, cumpliendo con uno de los objetivos estratégicos del proyecto.
- **Editor de código:** Visual Studio Code será el editor principal para el desarrollo frontend, dada su ligereza y amplio ecosistema de extensiones.

2. Lenguajes y herramientas de desarrollo (backend y datos)

El backend se enfoca en la creación de servicios de integración robustos y escalables:

- **Lenguaje de APIs (Java 21):** Seleccionado para la generación de las APIs y microservicios de *backend*. Java 21 garantiza rendimiento, estabilidad y la capacidad de integrarse con sistemas empresariales existentes (sistemas legados).
- **Base de datos (Sybase 16 y DB Artisan):** El motor de base de datos transaccional principal será Sybase 16, dada su integración y uso actual en el ecosistema corporativo de la compañía. Para la gestión y el desarrollo de *queries* y *stored procedures*, se empleará la herramienta DB Artisan.
- **Editor de código:** Visual Studio Code también será utilizado para el desarrollo backend.

2.3 Descripción de tablas de la base de datos

2.3.1 Listado de tablas con breve descripción.

Para el modelo de autorización y orquestación del escritorio centralizado, se definen las siguientes entidades principales:

- **Rol:** Almacena los perfiles de acceso que se asignarán a los usuarios para el sistema.
- **Área:** Define las áreas de TI responsables del mantenimiento de cada *microfrontend*.
- **AmbitoEscritorio:** Define las categorías de más alto nivel para organizar las funcionalidades (ej. Módulos Transaccionales).
- **GrupoFuncionalidad:** Define agrupaciones lógicas de funcionalidades dentro de un Ámbito.
- **Funcionalidad:** Catálogo maestro de los *microfrontends* y módulos que pueden ser cargados dinámicamente.
- **RolFuncionalidad:** Tabla de relación que asocia tabla Rol con tabla Funcionalidad.

2.3.2 Tipos de datos soportados por el motor de base de datos.

Sybase ASE 16 (Adaptive Server Enterprise) soporta una amplia gama de tipos de datos, que se pueden clasificar en varias categorías.

A continuación, en los siguientes puntos, se presenta un resumen de los principales tipos de datos.

2.3.2.1 Tipos de datos numéricos

Tabla 2-2 Tipos de datos numéricos

Tipo de Dato	**Almacenamiento Típico**	**Descripción**	**Ejemplo**
tinyint	1 byte	Números enteros pequeños sin signo. Rango: 0 a 255. Con signo: -128 a +127	Edad
smallint	2 bytes	Números enteros con signo. Rango: - 32.768 a 32.767.	Cantidad de ítems
int	4 bytes	Números enteros estándar con signo. Rango: -2.147.483.648 a +2.147.483.647	Montos, ganancias, deudas
bigint	8 bytes	Números enteros grandes con signo.	Contadores grandes
decimal / numeric	Variable	Almacena números de punto fijo (exactos), especificando precisión y escala.	Precios, porcentajes
float	8 bytes	Números de punto flotante (aproximados) de doble precisión.	Cálculos científicos
real	4 bytes	Números de punto flotante de precisión simple.	
money	8 bytes	Datos monetarios. Rango: -922.337.203.685.477,5808 - +922.337.203.685.477,5807	Montos de transacción

smallmoney	4 bytes	Datos monetarios más pequeños. Rango: -214,748.3648 a +214,748.3647	

Fuente: https://help.sap.com/docs/SAP_ASE

2.3.2.2 Tipos de datos fecha y hora

Tabla 2-3 Tipos de datos fecha y hora

Tipo de Dato	**Almacenamiento Típico**	**Descripción**
date	4 bytes	Solo almacena la fecha (sin hora).
time	3 bytes	Solo almacena la hora (sin fecha).
datetime	8 bytes	Almacena tanto la fecha como la hora. Precisión al milisegundo.
smalldatetime	4 bytes	Almacena fecha y hora. Menor rango y menor precisión (al minuto).
bigdatetime	8 bytes	Mínimo: 1 de Enero del año 0001 (A diferencia del año 1753 del datetime normal). Rangos: Máximo: 31 de Diciembre del año 9999. Hora: De 00:00:00.000000 a 23:59:59.999999.
bigtime	5 bytes	Mayor precisión para la hora que time (nanosegundos). Rangos: Mínimo: 00:00:00.000000 (Medianoche) Máximo: 23:59:59.999999 Contenido: Solo hora, minutos, segundos y microsegundos. No almacena día, mes ni año.

Fuente: https://help.sap.com/docs/SAP_ASE

2.3.2.3 Tipos de datos de cadena (String)

Tabla 2-4 Tipos de datos de cadena (String)

Tipo de Dato	Almacenamiento Típico	Descripción
char(n)	Fijo (n bytes)	Cadena de longitud fija. Se rellena con espacios si el dato es más corto.
varchar(n)	Variable	Cadena de longitud variable. Solo usa el espacio necesario.
unichar(n)	Fijo (2*n bytes)	Caracteres de longitud fija para datos Unicode (múltiples idiomas).
univarchar(n)	Variable	Caracteres de longitud variable para datos Unicode.
text	Variable (hasta 2 GB)	Cadena de caracteres muy larga. debido a su longitud, se almacena fuera de la fila (o tupla) de datos
unitext	Variable (hasta 2 GB)	Cadena Unicode muy larga.

Fuente: https://help.sap.com/docs/SAP_ASE

2.3.2.4 Tipos de datos binarios y lógicos

Tabla 2-5 Tipos de datos binarios y lógicos

Tipo de Dato	Almacenamiento Típico	Descripción
binary(n)	Fijo (n bytes)	Datos binarios de longitud fija (ej. un *hash*).
varbinary(n)	Variable	Datos binarios de longitud variable.
image	Variable (hasta 2 GB)	Datos binarios muy grandes (ej. imágenes, documentos). Similar a Tipo text.
bit	1 byte	Solo puede almacenar 0 o 1. Ideal para valores booleanos (verdadero/falso).

Fuente: https://help.sap.com/docs/SAP_ASE

2.3.2.5 Tipos Especiales

- **uniqueidentifier**: Almacena un valor GUID (Global Unique Identifier) de 16 bytes.
- **sql_variant**: Puede almacenar datos de varios tipos de sistema ASE (Adaptive Server Enterprise). Se utiliza principalmente en el código del sistema y no es recomendable para tablas de usuario.
- **xml**: Se utiliza para almacenar datos XML.

2.3.3 Codificación para el nombre de archivos, registros y/o campos

1. Convenciones Generales

- **Minúsculas/Mayúsculas:** Se utiliza Camel Case para todos los nombres multi-palabra.

  Ejemplo: fechaDeCreacion, clientesActivos.

- **Singular/Plural:** Se mantiene el uso de plural para tablas y singular para columnas.

2. Convenciones para Claves y Restricciones

Es fundamental usar prefijos claros para las restricciones (Constraints):

Tabla 2-6 Convenciones para claves y restricciones

Restricción	Prefijo	Ejemplo
Clave Primaria (PK)	PK_	PK_tblClientes
Clave Foránea (FK)	FK_	FK_tblOrdenes_tblClientes
Única (Unique)	UQ_	UQ_tblClientes_email
Chequeo (Check)	CK_	CK_tblProductos_precio

Fuente: Elaboración propia

Ejemplo Práctico

Tabla 2-7 Convenciones para claves y restricciones (Ejemplo)

Objeto	Convención (Camel Case)	Descripción
Tabla	TblProductos	Conjunto de productos.
Columna PK	ProductoID	Clave primaria.
Columna FK	CategoriaID	Clave foránea a tblCategorias.
Columna Normal	PrecioUnitario	Atributo del producto.
Vista	VwProductosConStock	Vista que filtra productos disponibles.
Stored Procedure	ActualizarStock	SP para modificar la cantidad de productos.
Restricción PK	PK_tblProductos	Nombre de la restricción PK.

Fuente: Elaboración propia

Así se vería la creación de una tabla usando la convención Camel Case con prefijos:

SQL

```
CREATE TABLE tblEmpleados (
    empleadoID          INT             NOT NULL,
    nombreCompleto      VARCHAR (100) NOT NULL,
    fechaContratacion   DATE            NULL,
    salarioBase         MONEY           NULL,
    departamentoID      INT             NULL)
-- Clave Primaria
ALTER TABLE tblEmpleados
    ADD CONSTRAINT PK_tblEmpleados PRIMARY KEY CLUSTERED (empleadoID)
-- Clave Foránea (asumiendo que existe una tabla tblDepartamentos)
ALTER TABLE tblEmpleados
    ADD CONSTRAINT FK_tblEmpleados_tblDepartamentos
    FOREIGN KEY (departamentoID)
    REFERENCES tblDepartamentos (departamentoID)
```

2.3.4 Descripción detallada de tablas

A continuación, se detalla la estructura física y lógica de las entidades clave del sistema:

2.3.4.1 Tabla: Funcionalidad (catálogo de microfrontends)

- **Nombre Lógico:** Funcionalidad
- **Descripción:** Catálogo maestro que registra cada *microfrontend* disponible para ser cargado por el *shell* principal (escritorio de trabajo), incluyendo su ubicación y trazabilidad técnica.
- **Clave Primaria (PK):** IdFuncionalidad
- **Clave(s) Foránea(s) (FK):**
    - IdAreaTI: Referencia a tabla Areas (IdAreaTI)
    - IdGrupoFuncionalidad: Referencia a tabla GrupoFuncionalidad (IdGrupoFuncionalidad)

Tabla 2-8 Tabla Funcionalidad (Catalogo de microfrontends)

Nombre Campo	Tipo de Dato	Largo	Descripción
IdFuncionalidad (PK)	INT	10	Identificador único del módulo o *microfrontend*.
NameSpace	VARCHAR	50	Nombre técnico usado por Webpack/Module Federation.
LinkLlamado	VARCHAR	255	URL de *runtime* o ruta para la carga dinámica del módulo.
NombreFuncion	VARCHAR	100	Nombre legible para el usuario de la funcionalidad.
DesFuncionalidad	VARCHAR	255	Descripción completa del módulo.
OrdenPanel	TINYINT	2	Posición en que se muestra el módulo dentro del grupo asignado.
FechaHoraInicio	DATETIME		Fecha y hora de despliegue y activación del módulo.
FechaHoraTermino	DATETIME		Fecha y hora de desactivación (para historial o baja).
DirFuenteGit	VARCHAR	255	Dirección del repositorio Git (Trazabilidad DevOps).

IdAreaTI (FK)	INT	10	Identificador del área responsable de la funcionalidad.
IdGrupoFuncionalidad (FK)	INT	10	Identificador del grupo de menú al que pertenece el módulo.

Fuente: Elaboración propia

2.3.4.2 Tabla: Rol (perfiles de acceso)

- **Nombre Lógico:** Rol
- **Descripción:** Define los perfiles de autorización que se asignan a los usuarios.
- **Clave Primaria (PK):** IdRol
- **Clave(s) Foránea(s) (FK):** (Ninguna)

Tabla 2-9 Tabla Rol (perfiles de acceso)

Nombre Campo	Tipo de Dato	Largo	Descripción
IdRol (PK)	INT	10	Identificador único del rol.
DescripcionRol	VARCHAR	50	Nombre descriptivo del perfil (ej. "Administrador Escritorio").

Fuente: Elaboración propia

2.3.4.3 Tabla: RolFuncionalidad (matriz de permisos)

- **Nombre Lógico:** RolFuncionalidad
- **Descripción:** Tabla de relación que determina qué Funcionalidades específicas son visibles y accesibles para un determinado Rol (Autorización).
- **Clave Primaria (PK):** IdFuncionalidad + IdRol
- **Clave(s) Foránea(s) (FK):**
    - IdFuncionalidad: Referencia a tabla Funcionalidad (IdFuncionalidad)
    - IdRol: Referencia a tabla Rol (IdRol)

Tabla 2-10 Tabla RolFuncionalidad (matriz de permisos)

Nombre Campo	Tipo de Dato	Largo	Descripción
IdFuncionalidad (PK, FK)	INT	10	Identificador de la Funcionalidad autorizada.
IdRol (PK, FK)	INT	10	Identificador del Rol al que se le otorga el permiso.

Fuente: Elaboración propia

2.3.4.4 Tabla: GrupoFuncionalidad (organización de menús)

- **Nombre Lógico:** GrupoFuncionalidad
- **Descripción:** Permite clasificar las funcionalidades en grupos lógicos para el menú de navegación (ej. "Mantenedores", "Prestaciones").
- **Clave Primaria (PK):** IdGrupoFuncionalidad
- **Clave(s) Foránea(s) (FK):** IdAmbitoEscritorio: Referencia a tabla AmbitoEscritorio(IdAmbitoEscritorio)

Tabla 2-11 Tabla GrupoFuncionalidad (organización de menús)

Nombre Campo	Tipo de Dato	Largo	Descripción
IdGrupoFuncionalidad (PK)	INT	10	Identificador único del grupo.
NombreGrupo	VARCHAR	50	Nombre del grupo que aparece en el menú.
IdAmbitoEscritorio (FK)	INT	10	Identificador del Ámbito superior al que pertenece el grupo.

Fuente: Elaboración propia

2.3.4.5 Tabla: Area (gobernanza y trazabilidad)

- **Nombre Lógico/Físico:** Area
- **Descripción:** Registra las áreas de tecnología y desarrollo responsables del mantenimiento y evolución de los *microfrontends*, es esencial para la gobernanza del proyecto.
- **Clave Primaria (PK):** IdAreaTI
- **Clave(s) Foránea(s) (FK):** (Ninguna)

Tabla 2-12 Tabla Area (gobernanza y trazabilidad)

Nombre Campo	Tipo de Dato	Largo	Descripción
IdAreaTI (PK)	INT	10	Identificador único del área de tecnología.
NombreArea	VARCHAR	50	Nombre del equipo o área responsable (ej. Área de Servicios Digitales, Área de Tesorería).

Fuente: Elaboración propia

2.3.4.6 Tabla: AmbitoEscritorio (categorías principales)

- **Nombre Lógico: AmbitoEscritorio**
- **Descripción:** Define las categorías de más alto nivel para organizar las funcionalidades dentro del escritorio (ej. "Módulos Transaccionales", "Consultas", "Administración").
- **Clave Primaria (PK): IdAmbitoEscritorio**
- **Clave(s) Foránea(s) (FK): (Ninguna)**

Tabla 2-13 Tabla AmbitoEscritorio (categorías principales)

Nombre Campo	Tipo de Dato	Largo	Descripción
IdAmbitoEscritorio (PK)	INT	10	Identificador único del ámbito o categoría principal.
NombreAmbitoEscritorio	VARCHAR	50	Nombre descriptivo del ámbito que agrupa a los grupos de funcionalidades.

Fuente: Elaboración propia

2.4 Diagrama de menús

Autenticación	Aplicaciones Front	Consultas Frecuentes	Ficha del Cliente Planilla de Siniestralidad Caso GES
		Acciones	Emitir Bono Recepcionar Licencias Cambio de Dirección
		Certificados	Deuda Cotizaciones Desafiliación
	Aplicaciones Back	Prestadores -Convenios	Ingresar Prestador Pago de Bonos Dirección de atención
		Matenedores	Escritorio de Trabajo Asignar tarea Resolver Tarea
		Proveedores	Nuevo Proveedor Crear OC Pagar Factura

Figura 2-4 Diagrama de menú

Fuente: Elaboración propia

El diagrama de la figura 2-4, ilustra una propuesta de configuración de menú. Sin embargo, la constitución de este menú es completamente dinámica y los administradores del sistema pueden crear nuevas sesiones y distribuir según sean las necesidades funcionales de los distintos proyectos que sostendrá este escritorio de trabajo.

2.5 Algunas pantallas

La figura 2-5, muestra la Pantalla principal, “Aplicaciones Front” activada la cual se usa principalmente en la atención de un cliente.

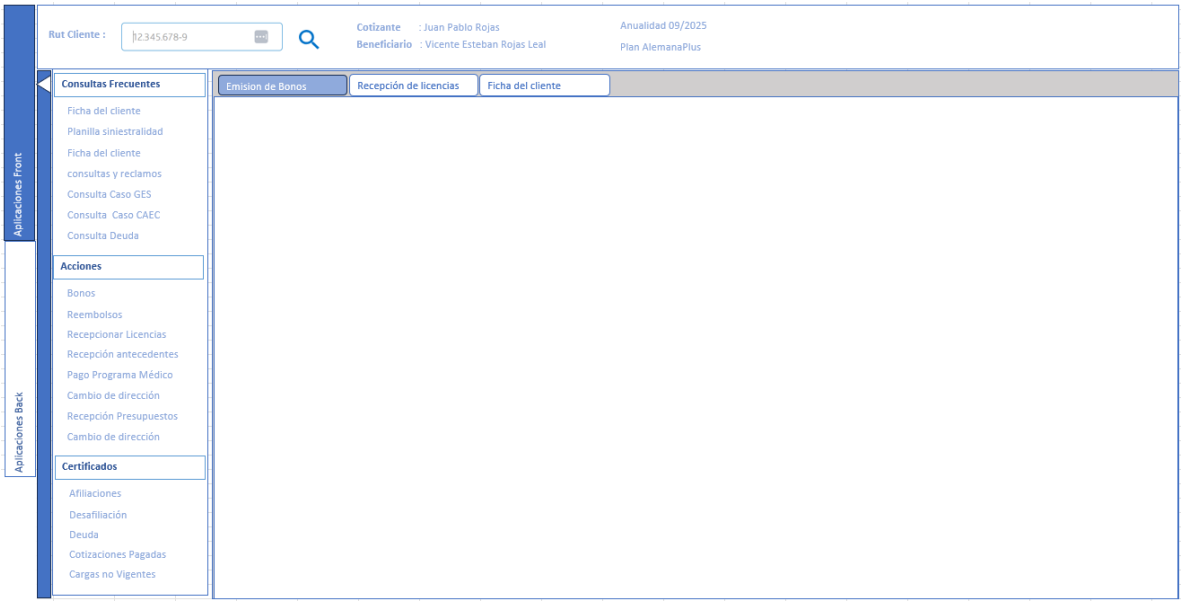

Figura 2-5 Aplicaciones Front

Fuente: Elaboración propia

La figura 2-6, muestra una vista similar a “Aplicaciones Front”, la diferencia es que en la vista de “Aplicaciones Back”: no existe un cliente seleccionado, por tanto, sólo se despliegan micro aplicaciones, pero no hay interacción entre ellas, en esta vista principalmente se despliegan mantenedores.

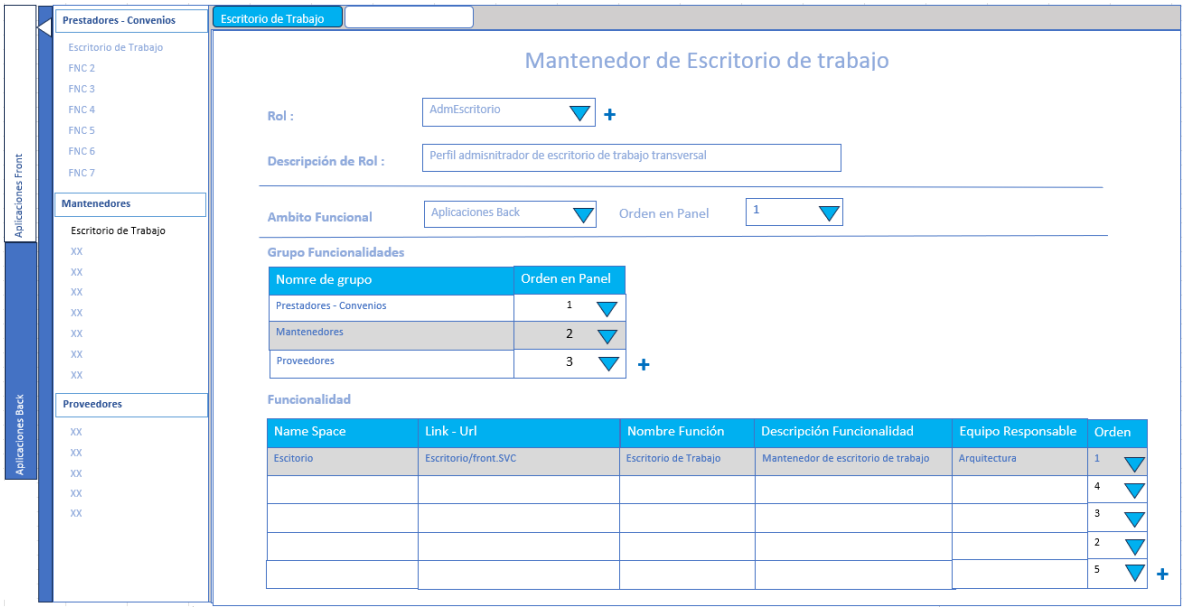

Figura 2-6 Aplicaciones Back

Fuente: Elaboración propia

Conclusión

Este trabajo de titulación ha culminado con éxito la fase de análisis y diseño integral de un nuevo escritorio de trabajo, utilizando la arquitectura de microfrontend. Más que un simple rediseño, esta propuesta es la respuesta estratégica que la compañía necesitaba para dejar atrás la rigidez de sus plataformas monolíticas y dar un salto hacia la agilidad, la seguridad y una mejor experiencia para sus colaboradores o usuarios internos.

1. El Logro principal: Objetivos cumplidos

El proceso de diseño muestra viabilidad técnica y funcional, que la solución propuesta está lista para pasar a la etapa de desarrollo. Se cumple con cada meta que se propuso en este informe:

Plataforma unificada, escalable y segura: Este era el objetivo central y se logra. El diseño arquitectónico garantiza que el nuevo sistema puede centralizar toda la experiencia del usuario e integrar las múltiples aplicaciones corporativas a través de APIs robustas. La seguridad se blinda mediante un punto de acceso único, eliminando riesgos de fragmentación.

No más a las múltiples contraseñas: Se diseñó el flujo de SSO (Single Sign-On) basado en Azure Active Directory, atacando de raíz la vulnerabilidad de autenticación fragmentada que presentaba el sistema antiguo (RF01); en resumen, tampoco habrá múltiples links sueltos para entrar a los sistemas.

Agilidad y despliegue autónomo: El mayor valor de este diseño es la independencia operativa. Al especificar la plataforma AKS y el pipeline CI/CD con Gitlab, se han sentado las bases para la autonomía de despliegue de cada microfrontend. Esto significa que los equipos podrán lanzar nuevas funcionalidades de forma rápida y elástica.

Experiencia coherente para el usuario: La solución no es solo tecnológica; es humana. El diseño del escritorio centralizado y la definición de un diseño único garantizan que, por primera vez, todos los colaboradores trabajarán con una experiencia de usuario (UX) unificada y consistente (RF04).

2. Aportes estratégicos y técnicos

Los entregables de este proyecto van más allá de un simple diagrama, ya que dejan un legado de gobernanza y trazabilidad indispensable para la futura mantención del sistema distribuido:

Modelo de gobernanza de datos: se desarrolló un modelo de datos relacional que centraliza la administración de permisos (Tablas Funcionalidad, Rol, AmbitoEscritorio). Este modelo no solo maneja transacciones, sino que actúa como un catálogo maestro de la arquitectura, indicando quién puede acceder a qué.

Trazabilidad garantizada: Se creó la base para un desarrollo sin cabos sueltos. La tabla FUNCIONALIDAD incluye campos clave como DirFuenteGit e IdAreaTI, lo que asegura la responsabilidad y trazabilidad completa del código en un ecosistema de equipos de desarrollo distribuidos.

Diseño con visión de futuro: La adopción de tecnologías Cloud Native como AKS asegura la sostenibilidad y elasticidad del sistema. Esto permite a la compañía evolucionar e incorporar nuevas aplicaciones de forma progresiva y sin la necesidad de costosas reestructuraciones de arquitectura en el futuro.

Mirando hacia el futuro: El camino a la implementación

El informe finaliza con una base detallada, sólida y fundamentada que justifica la factibilidad del proyecto. Más importante aún, sienta los lineamientos tecnológicos y de arquitectura que guiarán a la compañía en los próximos años.
El siguiente paso es la etapa de programación. El foco del trabajo futuro deberá concentrarse en el desarrollo del shell principal (escritorio de trabajo) y de los primeros microfrontends clave (como el Mantenedor de Perfil Consolidado), poniendo a prueba la integración con Azure AD y el deployment automatizado en el clúster AKS para ver a la arquitectura cobrar vida.

Bibliografía

Arquitectura de Software y Diseño Modular (microfrontends)

- Jackson, Cam & Fowler, Martin. (2019). *Micro Frontends*. martinfowler.com. Recuperado de: https://martinfowler.com/articles/micro-frontends.html
- Newman, Sam. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.
- Zalewski, Sebastian. (2019). *Microservices Architecture*. Packt Publishing.

DevOps, Orquestación y Cloud Computing (AKS, CI/CD)

- Bass, Len; Clements, Paul; Kazman, Rick. (2012). *Software Architecture in Practice*. 3rd Edition. Addison-Wesley.
- Humble, Jez; Farley, David. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.
- Microsoft. (2024). *Azure Kubernetes Service (AKS) Documentation*. Microsoft Learn. Recuperado de: https://learn.microsoft.com/en-us/azure/aks/

Bases de Datos y Seguridad

- SAP. (2024). *SAP Adaptive Server Enterprise (ASE) 16.0 Documentation*. SAP Help Portal. Recuperado de: https://help.sap.com/docs/SAP_ASE
- Ambler, Scott W. & Sadalage, Pramod J. (2006). *Refactoring Databases: Evolutionary Database Design*. O'Reilly Media.
- ISO/IEC. (2022). *ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection*. International Organization for Standardization.