UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISPONIBILIZACIÓN DE SERVICIOS PARA CORREDORES EN EL ÁMBITO DE SEGUROS GENERALES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

PABLO ALEJANDRO MUÑOZ SOTO

PROFESORA GUÍA:

JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:

ÉRIC TANTER

JOSÉ BENGURIA DONOSO

SANTIAGO DE CHILE

2023

Resumen

Para esta memoria se realizó un proyecto para la compañía FID Seguros, una empresa de seguros generales. Actualmente algunos corredores ocupan servicios expuestos por FID para cotizar y emitir pólizas, lo cual genera ventas para la compañía. Sin embargo, cada nuevo corredor que quiere consumir los servicios de FID requiere un nuevo desarrollo dado que las APIs tienen funcionalidades a medida para cada corredor. Esto implica que este tipo de integraciones solo son rentables para FID si el corredor tiene una gran participación en el mercado que indique a la compañía que la inversión del tiempo de desarrollo está justificada. Es debido a esto que corredores con una menor participación de mercado no son considerados para este tipo de integraciones.

El objetivo principal de este proyecto es desarrollar un sistema modular que facilite el desarrollo y paso a producción de nuevos servicios de consulta de datos de FID. Además, se deben desarrollar y exponer tres APIs de consulta de datos que hagan uso de este sistema. Sumado a esto, la solución a implementar debe minimizar el impacto de estas nuevas APIs de consulta sobre la performance del Core de la compañía. Para cumplir estos objetivos se implementó un mecanismo de autenticación y enrolamiento. Adicionalmente, se definieron prácticas de desarrollo basadas en el estilo arquitectural REST. Para minimizar el impacto a la performance del Core, se levantó una segunda base de datos, la cual contiene una réplica de los datos del Core de la compañía. Para mantener la información actualizada se implementó un sistema de migración de datos en vivo. Este sistema es el encargado de replicar cualquier inserción/modificación/borrado que ocurra en el Core dentro de la segunda base de datos, de tal manera que la información entre ambas sea consistente. Finalmente, se diseñaron e implementaron las tres APIs de consulta de datos, una para siniestro, una para cuotas y otra de póliza. Estas APIs obtienen la información de la segunda base de datos levantada, minimizando el impacto en la performance del Core.

Para realizar el proyecto, el memorista lideró un equipo de dos desarrolladores con la supervisión del jefe de sistemas de la compañía. Se llevaron a cabo reuniones semanales con otros miembros de la empresa para informar los avances y discutir los diseños de los componentes de la solución. Durante estas reuniones, se identificaron debilidades u oportunidades de mejora en los diseños que fueron abordados segundas iteraciones de estos.

Para validar la solución, se realizó una encuesta dirigida a las personas involucradas en el proyecto. Se formularon preguntas con el objetivo de evaluar en qué medida la solución logra cumplir con los objetivos establecidos al inicio del proyecto. La mayoría de las respuestas indican que la solución satisface de manera razonable los objetivos propuestos. Sumado a esto, se monitoreó el Core de compañía para validar que el sistema de migración de datos en vivo no ponga en riesgo la continuidad operacional, las métricas recolectadas indican que el sistema de migración de datos no representa un riesgo en este aspecto.

En conclusión, el proyecto aborda de manera exitosa los objetivos propuestos y permite a FID dar el primer paso hacia el paradigma de Open Insurance. Se requerirán esfuerzos adicionales por parte de FID para seguir expandiendo el proyecto y ver el máximo potencial de este.

Agradecimientos

El trabajo presentado en esta memoria fue posible gracias al apoyo de muchas personas, por lo que quiero aprovechar de agradecer profundamente toda la ayuda que recibí durante este proceso.

En primer lugar, quiero agradecer y reconocer a todos mis compañeros de trabajo, principalmente por su aporte a la realización del proyecto, mediante el desarrollo de las componentes, participación en las discusiones de diseño, ideas para refinar los sistemas, etc. Agradezco en particular la buena disposición, ánimo y compañerismo que se reflejaba en los involucrados en el proyecto. Durante la realización de la memoria aprendí muchas cosas de mis supervisores y compañeros de equipo que sin duda me serán de gran ayuda durante toda mi carrera profesional, por lo que me siento afortunado de haber tenido la posibilidad de trabajar con profesionales que tengo en alta estima.

Adicionalmente, quiero expresar mi gratitud a mi profesora guía Jocelyn Simmonds, la cual fue un apoyo fundamental al momento de redactar y revelar oportunidades de mejora en la presentación de la información. En particular, me gustaría destacar la disposición y dedicación al momento de entregar comentarios bien pensados, los cuales permitieron que esta memoria fuera una de mayor calidad.

Sumado a esto, quisiera agradecer a mi familia por su apoyo incondicional durante todo este proceso y mi trayectoria educativa en general. En particular, me gustaría agradecer a mis padres y hermana, los cuales desde que era joven, han tenido una actitud alentadora y de apoyo cada vez que me ha surgido una oportunidad para desarrollarme como persona. Esta red me otorga una seguridad que considero fundamental al momento de abordar los desafíos del pasado, presente y futuro.

Finalmente, pero no menos importante, me gustaría agradecer a todos mis amigos, conocidos y familiares con los que interactúe durante la realización de esta memoria. Estos estuvieron presentes en múltiples actividades de distensión que fueron fundamentales para desconectarme temporalmente del proyecto, para que al volver tuviera la mentalidad y energía renovadas. En particular, agradezco a mis amigos más cercanos, los cuales siempre han sido un impacto positivo en mi vida, mediante sus consejos, apoyo y risas, y estoy seguro que en el futuro van a seguir siendo una parte importante en mi vida.

Tabla de Contenido

1. Introducción

1.1. Contexto

Para esta memoria se realizó un proyecto para la compañía FID Seguros, una empresa que vende seguros generales. Estos seguros son ofrecidos por corredores asociados a la compañía, los cuales corresponden a personas independientes que, a cambio de una remuneración cumplen el rol de mediador entre el cliente final y la compañía aseguradora para concretar la venta de un seguro [1].

Múltiples rubros se han visto en la necesidad de digitalizar y renovar sus tecnologías, entre estos el área de Seguros Generales, en donde incluso desde el año 2017 se empezaban a ver artículos enfatizando la importancia de invertir recursos en modernizar las compañías y procesos [2]. A raíz de esto, ha comenzado a aparecer en los mercados las "insurtech", definidas como compañías aseguradoras que aplican las nuevas tecnologías al mundo de los seguros [3].

Un enfoque particular de este cambio de paradigma es lo conocido como "Open Insurance", esta visión, inspirada del concepto análogo en el ámbito bancario "Open Banking", corresponde a "un conjunto de normas que permite proporcionar datos y servicios a socios, comunidades y startups para crear servicios, aplicaciones y modelos de negocio innovadores." [4]. Mediante estos servicios disponibilizados por la compañías de seguros se fomenta la innovación de nuevos participantes del mercado, permite aumentar el alcance de venta de los productos y recolectar datos que permitan informar las decisiones de estas.

FID Seguros empezó sus operaciones en 2020 y cerraron el año 2022 con ventas del orden de 100 millones dólares [5], este crecimiento se explica dado que cuentan con el apoyo de la aseguradora portuguesa Fidelidade que tiene más de 200 años de historia y el grupo de inversión FOSUN [6]. FID se cataloga de la siguiente manera "nos definimos como una insurtech, una empresa que ha incorporado la tecnología en sus procesos y de manera ágil" [6], por lo que la transformación digital está en el centro de la visión de la compañía y el paradigma de Open Insurance ha comenzado a generar interés dentro del empresa.

1.2. Problema

Actualmente la compañía expone APIs a corredores con un gran volumen de ventas para que ellos puedan generar cotizaciones y emisiones de los seguros que venden en nombre de FID. Cada vez que se quiere habilitar estas funcionalidades para un nuevo corredor, se deben realizar modificaciones a las APIs a exponer, ya que estas no fueron construidas con un esquema escalable en mente, sino que personalizadas para cada corredor en particular. Esto lleva a que se tengan que utilizar horas del equipo de desarrollo para cubrir las necesidades específicas de cada corredor y se genere una gran cantidad de replicación de código. Debido al costo y tiempo asociado de exponer estas funcionalidades a los corredores, estos desarrollos están limitados solo para los corredores que tengan un volumen de ventas que justifique el tiempo y esfuerzo necesario para concretar la integración, eliminando la posibilidad de considerar a corredores o startups más pequeñas.

Este proyecto busca abordar el problema descrito anteriormente, abaratando los costos de integración por cada corredor de tal manera que sea factible para FID ofrecer estos servicios para corredores más pequeños, ampliando el alcance de venta de sus productos. Adicionalmente se deben sentar las bases arquitecturales, infraestructurales y los lineamientos de desarrollo adecuados para expandir el trabajo realizado en un proyecto posterior, con el objetivo de implementar la estrategia de Open Insurance dentro de la compañía en el largo plazo.

1.3. Objetivos

1.3.1 Objetivo General

El objetivo principal de este proyecto es diseñar e implementar un sistema modular que facilite el desarrollo y paso a producción de nuevos servicios de consulta de datos de FID, haciendo uso de buenas prácticas de desarrollo de software. Además se deben desarrollar y exponer 3 APIs de consulta de datos que hagan uso de este sistema.

1.3.2 Objetivos Específicos

- Implementar un flujo de enrolamiento que permita a los corredores obtener acceso a las APIs del proyecto sin que FID tenga que destinar tiempo de sus desarrolladores para cada nueva integración.
- El flujo de enrolamiento para obtener acceso a las APIs debe ser intuitivo y fácil de usar, los corredores deben ser capaces de obtener accesos a las APIs con tan solo una capacitación de 10 min.
- Implementar un esquema de seguridad que permita autenticar e identificar a los corredores realizando la solicitud, permitiendo que las APIs puedan tomar decisiones con respecto a qué información puede o no ser mostrada dependiendo del corredor identificado.
- Definir lineamientos de diseño en cuanto a firmas para que sean consistentes con el esquema REST. Adicionalmente, se deben definir prácticas de desarrollo de APIs que minimicen las vulnerabilidades de los servicios a exponer. Lo anterior con el objetivo de evitar filtraciones de datos y que se proyecte una buena imagen de la cultura de desarrollo de la compañía.
- La solución debe minimizar el impacto de esta sobre la performance del Core de la compañía, de tal manera que no se produzcan riesgos a la continuidad operacional.

Además se deben desarrollar y exponer 3 APIs, estas corresponden a:

- Póliza API: Servicio que permite consultar información sobre 1 póliza en particular a partir del número de póliza, sólo debe devolver información sobre las pólizas asociadas al corredor identificado en la solicitud.

- Cuota API: Servicio que permite consultar información sobre las cuotas asociadas a las pólizas manejadas por el corredor identificado.
- Siniestro API: Servicio que permite consultar información sobre 1 siniestro en particular a partir del número de siniestro, sólo debe devolver información sobre los siniestros relacionados a pólizas asociadas al corredor identificado en la solicitud.

1.4. Solución Desarrollada

Para implementar el proceso de autenticación y enrolamiento se necesita involucrar a la empresa en prácticas de API Management, esto se define como "API management o gestión de API es el proceso de diseño, publicación, documentación y análisis de API, en un entorno seguro. Mediante una herramienta de administración de API, se puede garantizar que tanto APIs públicas como internas funcionen correctamente y sean seguras." [7]. Para facilitar y acelerar la implementación del proyecto se utilizó una herramienta de API Management, con las funcionalidades de autenticación, administración y enrutamiento de las APIs.

Con respecto a la facilidad de integración para los corredores, se implementó una pantalla en el portal de corredores, el cual corresponde a un sitio Web de la compañía donde los corredores pueden consultar información sobre sus pólizas, clientes, cuotas, siniestros, etc. Se eligió implementarla ahí porque corresponde a una plataforma familiar para los corredores que ya están familiarizados. Esta nueva pantalla permitirá a los corredores generar API keys que permitan otorgar acceso a los servicios a exponer.

Para abordar el impacto en la performance del Core de la compañía, se levantó una segunda base de datos más robusta y escalable que contenga una copia de los datos originales del Core, la cual será actualizada a medida que ocurran cambios en este. Para esto se diseñaron dos esquemas, uno para la migración de datos históricos y otro para traspasar los cambios en vivo desde el origen al destino con un pequeño desfase. Esta funcionalidad se desarrolló haciendo uso de la funcionalidad de CDC (Change Data Capture) de SQL Server, que permite monitorear los cambios ocurridos en la base de datos de origen.

Con esto en mente se diseñaron las 3 APIs requeridas de tal manera que obtengan la información desde la segunda base de datos, para que la carga adicional generada por el flujo de solicitudes en las nuevas APIs no produzca un impacto en la performance del Core de la compañía.

La definición de lineamientos de desarrollo de APIs tuvo un carácter más investigativo en donde se evaluaron recomendaciones de organizaciones, comentarios en foros y artículos especializados para APIs REST. Luego, se compararon con los casos de uso específicos de la compañía para validar si es aplicable o no. De la misma forma para verificar la seguridad de los sistemas se investigó sobre las vulnerabilidades más comunes, y se realizó una comparación con los desarrollos anteriores para detectar posibles problemas/vulnerabilidades que puedan ser corregidas mediante sus contramedidas.

1.5. Metodología

Para implementar la memoria el memorista trabajó con un equipo de dos desarrolladores. El rol del memorista dentro del equipo fue principalmente de liderazgo, en donde se incluyen responsabilidades como la definición de los objetivos semana a semana, delegar tareas, evitar bloqueos al equipo, etc. Adicionalmente, el equipo estaba siendo supervisado por el jefe de sistemas de la compañía, el cual estaba presente en cualquier actividad de coordinación del equipo.

Para monitorear avances y comentar posibles bloqueos dentro del equipo se implementó un esquema de reuniones diarias basadas en la metodología Scrum. Estas reuniones diarias en Scrum reúnen al equipo de trabajo para que cada miembro responda que hizo ayer, que piensa hacer hoy y si hay bloqueos para cumplir sus tareas [8]. Notar que solo se incluyó esta ceremonia Scrum dentro del esquema de trabajo.

Adicionalmente, se realizó una reunión al final de cada semana, en donde además del equipo de trabajo mencionado anteriormente asistía la subgerenta de tecnología, el gerente de finanzas y tecnología, el arquitecto de la compañía y un arquitecto consultor externo. La agenda de esta reunión era definida por el memorista. Entre las actividades a realizar se incluía: comentar los avances de la semana, cuáles eran los problemas actuales del equipo, cuáles eran los siguientes pasos y aclarar dudas del contexto de negocio relacionado al proyecto. Adicionalmente, la reunión era utilizada para obtener feedback y refinar los análisis y diseños requeridos para la solución. En ocasiones estos comentarios mostraban debilidades en el diseño original que eran luego corregidas en un nuevo diseño. Este nuevo diseño era luego revisado en la siguiente reunión semanal para confirmar que la debilidad había sido abordada correctamente.

1.6. Estructura del documento

El documento comienza con el capítulo del estado del arte, en este se presentarán las tecnologías y estándares que deben ser conocidos para poder entender el resto de la memoria.

Posterior a esto, el documento continúa con el capítulo de análisis y diseño. Este capítulo está dividido en tres partes, una parte para la situación antes del inicio del proyecto, otra para análisis y finalmente la parte de diseño.

En la situación antes del inicio del proyecto se explica el contexto de la compañía al momento de iniciar el proyecto, se incluyen detalles sobre el estado actual de las integraciones de la compañía, la infraestructura utilizada en sus sistemas y las prácticas de desarrollo utilizadas en ese momento.

En la sección de análisis se revisaron si las tecnologías y estándares revisados en el capítulo de estado del arte eran adecuados para cumplir los objetivos del proyecto. Adicionalmente, se examinó la información presentada hasta el momento para derivar consideraciones al momento de diseñar la solución.

En la sección de diseño se modelaron las distintas componentes necesarias para implementar la solución y su interacción entre ellas.

Posterior a estas tres secciones que engloban el capítulo de análisis y diseño, se continúa con el capítulo de implementación. En este capítulo se detalla el proceso de creación y de las distintas componentes del proyecto, indicando cuales fueron los principales desafíos durante el proceso de la implementación. Adicionalmente, se proporcionan detalles sobre las tecnologías finalmente utilizadas.

Luego de esto, se continúa con el capítulo de validación. En este capítulo se busca corroborar que la solución implementada aborda la problemática central de la memoria. Para esto se realizó una encuesta dirigida a los miembros del área de tecnología con preguntas que apuntan a validar que los objetivos fueron abordados por la solución. En el capítulo se discuten los resultados de esta encuesta para determinar si se cumplieron o no dichos objetivos.

Tras el capítulo de validación, se procede a cerrar el contenido principal de la memoria con el capítulo de conclusión, en donde se resume el trabajo realizado y se presentan reflexiones sobre este.

Finalmente, se incluye el capítulo de referencias que incluye todas las fuentes utilizadas en los capítulos anteriores.

2. Estado del arte

El presente capítulo detalla el estado del arte, en donde se comentarán las tecnologías y estándares que son actualmente relevantes para solucionar el problema presentado en esta memoria. En la sección 2.1 se realiza una introducción al concepto de APIs RESTful, para proceder a discutir mecanismos de control de acceso en la sección 2.2. Para finalizar, en la sección 2.3 y 2.4 se explican las tecnologías de Cloud computing y contenedores, respectivamente.

2.1. APIs RESTful

Una Application Programming Interface, o API por sus siglas en Inglés, está definida como un conjunto de definiciones y protocolos que se usa para diseñar e integrar el software de las aplicaciones [9], y corresponden a la principal forma de disponibilizar servicios o funcionalidades a otras aplicaciones.

REST, o Representational State Transfer por sus siglas en Inglés, es un estilo arquitectural utilizado para diseñar APIs, y corresponde a uno de los enfoques más utilizados al momento de implementar servicios [10].

Entre los beneficios de diseñar APIs que siguen la arquitectura REST se encuentran:

- Interoperabilidad: Las APIs REST están basadas en los mismos estándares utilizados en la Web. Debido a esto, son altamente interoperables e interactúan de manera sencilla entre ellas [11].
- Flexibilidad: Las APIs REST pueden comunicarse usando cualquier formato de datos. Esto implica que pueden ser adaptadas a casi cualquier aplicación en la Web, independiente de su formato, lenguaje o arquitectura [11].
- Escalabilidad: Las APIs REST procesan solicitudes sin la necesidad de sesiones de usuarios, dado que cada solicitud lleva su propio contexto. Esta arquitectura promueve el escalamiento horizontal, facilitando la expansión de recursos a medida que se vea necesario [12].

En el contexto de esta memoria, las APIs que posee FID fueron diseñadas basadas en la arquitectura REST, por lo qué este es el enfoque que se utilizará para implementar las APIs de esta memoria y definir los lineamientos de diseño.

2.1.1 Recursos

La abstracción clave de información en REST es un recurso. Cualquier información que pueda ser nombrada puede ser un recurso. Por ejemplo, un recurso REST puede ser un documento, imagen, servicio temporal o una persona [10].

Una agrupación de recursos son llamados colecciones. Cada colección es homogénea, de modo que solo contenga un tipo de recurso, y sin un orden asociado. Estas colecciones pueden existir globalmente, al más alto nivel de una API, pero también pueden ser contenidos dentro de un recurso. Si este es el caso, estas colecciones son llamadas sub-colecciones, y son usualmente usadas para expresar alguna relación de "contenido en" [13].

A continuación se presenta algunos ejemplos de recursos:

/universidad_de_chile

Este recurso representa a la Universidad de Chile, notar que es un recurso único y no una colección dado que no existen múltiples entidades de la Universidad de Chile.

/universidad_de_chile/facultades

Esto hace referencia a una colección de facultades, esta es puesta posterior al recurso de la Universidad de Chile para indicar una relación de "contenido en", por lo que en particular corresponde a una sub-colección. Los recursos están jerarquizados de esta manera para indicar que las facultades no existen solas sin un contexto asociado, sino que están relacionados a la Universidad de Chile.

2.1.2 Los seis principios REST

REST está basado en algunas restricciones y principios que promueven la simplicidad y escalabilidad en el diseño. Estos principios son presentados a continuación.

Interfaz uniforme

Al aplicar el principio de generalidad a las interfaces de los componentes es posible simplificar la arquitectura del sistema y mejorar la visibilidad de las interacciones [10]. El principio de generalidad establece que al tener que resolver un problema se debe buscar un problema más general que posiblemente esté oculto tras el problema original, puesto que puede suceder que el problema general no sea mucho más complejo (a veces puede ser incluso más simple) que el original y posiblemente la solución al problema general tenga potencial de reuso [14].

Para generar una interfaz uniforme se sugiere seguir las siguientes reglas:

- Identificación de recursos: La interfaz debe identificar de manera única cada recurso involucrado en la interacción entre el cliente y el servidor [10].
- Manipulación de recursos mediante representaciones: Los recursos deben tener representaciones uniformes en la respuesta del servidor. Los consumidores de las APIs deben usar estas representaciones para modificar el estado del recurso en el servidor [10].
- Mensajes auto descriptivos: Cada representación del recurso debería llevar suficiente información como para describir cómo procesar el mensaje. Adicionalmente, también debe

proveer información de las acciones adicionales que el cliente puede realizar en el recurso [10].

- Hypermedia como el motor del estado de la aplicación: El cliente solo debe tener la URI inicial de la aplicación. La aplicación del cliente debe dinámicamente impulsar todos los otros recursos e interacciones mediante el uso de hipervínculos [10]. En este contexto, URI es la sigla de uniform resource identifier y corresponde a una cadena que identifica un recurso [15].

Un ejemplo de estas reglas aplicadas se presenta a continuación, en donde se realiza una petición GET a http://api.domain.com/management/departments/10 y se obtiene la siguiente respuesta del servidor [16]:

```
{
  "departmentId": 10,
  "departmentName": "Administration",
  "locationId": 1700,
  "managerId": 200,
  "links": [
    {
      "href": "10/employees",
      "rel": "employees",
      "type" : "GET"
    }
  ]
}
```

En este ejemplo se puede ver que la URI única que identifica al departamento 10 corresponde a http://api.domain.com/management/departments/10, ejemplificando la regla de identificación de recursos. Adicionalmente, se ve que un departamento está compuesto por su id, nombre, ubicación, manager y sus empleados, esta estructura debe ser consistente con los demás departamentos para tener una representación uniforme. Finalmente, se incluye dentro de la respuesta las posibles acciones que pueden ser realizadas sobre este recurso. En particular, se indica que se pueden obtener los empleados del departamento añadiendo “/employees” a la URI,

por lo que el mensaje es auto descriptivo y permite al cliente solicitar la información de manera dinámica.

Las reglas presentadas en esta sección permiten a REST definir una interfaz constante y uniforme entre cliente y servidor. Un ejemplo de esto es como las APIs REST basadas en HTTP hacen uso de los métodos HTTP (GET, POST, PUT, DELETE, etc.) y las URIs para identificar los recursos [10].

Cliente-Servidor

El patrón de diseño de cliente-servidor promueve la separación de responsabilidades. Esto ayuda a que el cliente y las componentes del servidor evolucionen de manera independiente [10]. Al separar las preocupaciones de la interfaz del usuario (cliente), de las preocupaciones del almacenamiento de la información (servidor), es posible mejorar la portabilidad de la interfaz del usuario a través de múltiples plataformas y mejorar la escalabilidad al simplificar las componentes del servidor [10]. Cuando el cliente y el servidor evolucionan es importante asegurarse que la interfaz/contrato entre ellos no se rompa [10].

Stateless

Statelessness dictamina que cada solicitud desde el cliente al servidor debe contener toda la información necesaria para entender y completar la solicitud. El servidor no puede tomar ventaja de ninguna información de contexto previamente almacenada, es por esto que el estado de la sesión debe ser manejado por el cliente [10].

Cacheable

Cachear es el proceso de almacenar información en un cache, el cual es un almacenamiento temporal que facilita un más rápido acceso a la información con el objetivo de mejorar el rendimiento de la aplicación y el sistema [17]. En REST, la acción de cachear debería ser aplicado a los recursos siempre que sea posible, y estos recursos deben declararse a sí mismos como cacheables [18].

Sistema por capas

REST permite usar un sistema de arquitectura por capas, en donde las APIs son desplegadas en un servidor A, la información es almacenada en un servidor B y las solicitudes son autenticadas en un servidor C [18].

Código on demand

REST permite al servidor retornar código ejecutable para apoyar a una parte de la aplicación [18]. Este código permite simplificar los clientes, dado que reduce el número de funcionalidades que necesitan ser pre-implementadas. Los servidores pueden proveer parte de las funcionalidades en forma de código, y el cliente solo tendría que ejecutar este [10].

2.2. Control de acceso

Control de acceso se define como un mecanismo de seguridad que regula quien o que puede ver o utilizar recursos en un ambiente computacional [19]. En el contexto de esta memoria, se necesita implementar un mecanismo de control de acceso para manejar quién puede realizar solicitudes a las APIs expuestas, de tal manera que se evite que los datos de la compañía sean públicos y evitar que los corredores vean información de clientes que no son suyos.

Para aplicar mecanismos de control de acceso a las solicitudes a las APIs es posible apoyarse en un API gateway. Este actúa como mediador entre las aplicaciones del cliente y las APIs. Está encargado de controlar las solicitudes y respuestas a las APIs mediante el manejo del tráfico de esta, y de hacer cumplir políticas de seguridad sobre las solicitudes [20]. De ahora en adelante cuando en la memoria se haga referencia a un gateway, se debe asumir que se está hablando específicamente sobre un API gateway.

A continuación se muestran los mecanismos de control de acceso para autenticar solicitudes de APIs REST que fueron revisados para efectos de esta memoria.

API keys

Una API Key es un identificador que sirve para la autenticación de un usuario para el uso de un servicio [21]. Son principalmente utilizadas para prevenir intentos de uso malicioso de las APIs [22]. Estas son enviadas junto con la solicitud para proceder a ser rechazada o validada dependiendo de los permisos y límites de usos asociados a la API Key. Poseen la desventaja de que en caso de que alguna API Key haya sido filtrada, el atacante tendría acceso al servicio hasta que la API Key sea revocada [23].

JWT

Json Web Token, o JWT por sus siglas en Inglés, es un estándar que define un mecanismo para propagar de manera segura la identidad y permisos de un determinado usuario entre dos partes [24]. Estos corresponden a una cadena de texto compuestas por 3 partes codificadas en base64 y separadas por un punto [23]:

- Header: Contiene información del algoritmo utilizado.
- Payload: Contiene la información de la identidad y permisos del usuario.
- Signature: Contiene la firma utilizada para verificar el JWT.

La desventaja principal de un JWT, es que una vez que han sido generados no existe una forma de revocarlos que no impliquen realizar un esfuerzo significativo. Esto es debido a que su proceso de validación no depende de llamadas a bases de datos [25].

2.3. Cloud computing

Cloud computing es la disponibilidad bajo demanda de recursos de computación como servicios a través de Internet. Esta tecnología evita que las empresas tengan que encargarse de aprovisionar, configurar o gestionar los recursos y permite que paguen únicamente por los que usen [26]. Como se verá en el siguiente capítulo, la infraestructura de FID reside en un proveedor de servicios de Cloud Computing, por lo que es fundamental entender esta sección para seguir las definiciones arquitecturales de esta memoria.

El uso de una infraestructura cloud conlleva una serie de beneficios. Entre ellos se incluyen tiempos de despliegue más rápidos, mayor escalabilidad y flexibilidad, ahorro de costes, mejor colaboración, seguridad avanzada y prevención de la pérdida de datos [26].

Sin embargo, también incluye limitaciones. Entre estas se incluye el riesgo de depender de otros proveedores, menos control sobre la infraestructura subyacente, preocupaciones sobre los riesgos de seguridad, complejidad de integración con sistemas actuales y costes imprevistos. Es importante recalcar que estas limitaciones pueden ser abordadas investigando y analizando cuidadosamente a los proveedores de servicios y sus modelos de servicio [26].

Actualmente, los proveedores de servicios de Cloud Computing más populares corresponden a [27]:

- Google Cloud Platform (GCP).
- Amazon Web Services (AWS).
- Microsoft Azure.

Las nubes pueden ser catalogadas como públicas o privadas. Las nubes públicas son infraestructuras compartidas, múltiples clientes del mismo proveedor acceden a la misma infraestructura, sin embargo, la información no es compartida entre ellos. En contraste, una nube privada es un servicio ofrecido exclusivamente a solo una organización [28]. Los proveedores de servicios Cloud mencionados anteriormente están basados en una infraestructura de nube pública [29] [30] [31]. Sin embargo, estos proveedores permiten a sus usuarios configurar una Virtual Private Cloud, o VPC por sus siglas en Inglés. Esta corresponde a una nube privada aislada y securizada que está alojada dentro de una nube pública [28]. Esto permite a sus usuarios emular las condiciones ofrecidas por una nube privada mientras se utiliza los servicios de estos proveedores.

En particular para FID Seguros, se está utilizando una infraestructura alojada en GCP, en donde se tienen múltiples VPCs que alojan los recursos utilizados por la compañía.

2.4. Contenedores

Los contenedores son paquetes de software que incluyen todos los elementos necesarios para ser ejecutados en cualquier entorno. Debido a que virtualiza el sistema operativo estos pueden ser ejecutados en cualquier parte, desde un centro de datos privado hasta la nube pública o incluso el portátil personal de un desarrollador [32].

Las tecnologías de contenedores más relevantes actualmente corresponden a [33]:

- Docker.
- Podman.
- CRI-O
- containerd.
- runc.
- runhcs.

Los contenedores son comúnmente manejados mediante una plataforma de contenedores. Estas son herramientas que crean, manejan y ofrecen seguridad a aplicaciones de contenedores. Con esto se logra una conectividad más fácil y rápida, además de permitir su orquestación [34].

FID actualmente está levantando sus servicios utilizando contenedores Docker. Para la orquestación de estos, están utilizando la herramienta de Google Kubernetes Engine, o GKE por sus siglas en Inglés, la cual corresponde a una implementación de GCP de la plataforma de organización de contenedores Kubernetes [35].

3. Análisis y Diseño

El presente capítulo detalla el proceso de análisis y diseño de la solución, en donde se comentará la situación anterior al proyecto mencionando todos los aspectos a considerar, para luego proceder a discutir posibles soluciones a las problemáticas del proyecto basadas en el estado del arte. Finalmente se detalla el diseño a utilizar en los distintos sistemas de la solución.

3.1. Situación antes del inicio del proyecto

La presente sección detalla el contexto existente en la compañía antes del inicio del proyecto. Se abordan diversos aspectos, tales como las integraciones en curso, la configuración de la infraestructura general de la compañía, las prácticas de desarrollo que están siendo empleadas en las APIs revisadas, y el método de autenticación actualmente utilizado por dichas APIs. Estos detalles son presentados a profundidad en las siguientes secciones.

3.1.1 Integraciones

Actualmente FID Seguros mantiene un acuerdo con Seguros Falabella y Mesos (https://www.mesos.cl/) que les permite vender seguros de FID a través de sus respectivos sitios Web. Para lograr esto, los sistemas de estas compañías utilizan una API de cotización de seguros que FID puso a disposición de estas empresas, que les permite obtener el precio del seguro que el cliente quiere contratar. Posterior a esto, estas compañías informan a FID cuáles de esas cotizaciones resultan en una contratación de parte del cliente. En ese momento, FID realiza la emisión de la póliza correspondiente, con los detalles informados. Este proceso permite a Seguros Falabella y Mesos actuar como corredores de seguros de FID.

Adicionalmente, para hacer efectivo un seguro solicitado por un cliente, se debe pasar por la inspección del ítem asegurado. Para el caso de seguros de vehículos, esto corresponde al proceso por el cual se determina el estado en que se encuentra un vehículo, al momento de adquirir una póliza con la compañía [36]. La mayoría de los seguros vendidos por FID son automotrices, por lo que esta situación es la más común.

Actualmente, FID no es la entidad que realiza este proceso de inspección, este procedimiento es realizado por empresas inspectoras especializadas como RAC (https://www.racasistencia.cl/) o LET (https://www.letchile.cl/). Estas empresas luego informan la cantidad de inspecciones realizadas para pólizas de FID y sus resultados utilizando servicios expuestos por la aseguradora.

Estos terceros y sus sistemas que consumen algún servicio expuesto por FID serán referenciados como integradores de ahora en adelante.

En particular para corredores grandes, la integración de estos requiere de un gran nivel de coordinación y colaboración entre ambas partes interesadas, por lo que cada una de estas implica destinar tiempo del equipo de desarrollo de FID para llevarlas a cabo. Esto se debe a que cada nueva integración involucra las siguientes tareas:

- Definición de la firma de entrada: Involucra reuniones entre los equipos de desarrollo de ambas compañías y los representantes del área de negocio correspondiente. El poder de negociación de FID en este aspecto depende del volumen de ventas del integrador. En el caso de corredores con grandes volúmenes de ventas, como Seguros Falabella, FID básicamente tiene que adaptarse al formato que ellos utilizan. Situaciones como esta contribuyen a que no exista una entrada estándar para los servicios proporcionados por FID. En cambio, se debe generar un endpoint dentro de las APIs para cada integrador.

- Implementación: Dado que por lo general cada nueva integración conlleva una firma distinta, dentro de la API existe un endpoint por cada integrador. Dependiendo de la firma de entrada con la que se esté trabajando se debe realizar un proceso de homologación, lo cual corresponde a traducir códigos utilizados en la compañía integradora a los reconocidos por FID. Un ejemplo de esto es como en Falabella un auto de color verde puede ser simbolizado por “G”, pero en FID, ese auto se simboliza con “V”, por lo que se tiene que implementar la lógica para realizar esta transformación.

- Pruebas de lo desarrollado: Todos los desarrollos tienen que ser aprobados por el equipo de Quality Assurance de FID. Quality Assurance, o QA por sus siglas en Inglés, es el equipo responsable de asegurar la calidad del software y de prevenir fallos en él [-]. En FID, el equipo de QA levanta un plan de pruebas para cada nuevo desarrollo en una API. Una vez que este ha sido completado y aprobado, se realiza una sesión de pruebas conjuntas con el integrador que va a utilizar el servicio. Si el desarrollo ha pasado por los dos procesos sin problemas, este puede ser desplegado en producción.

Este proceso toma alrededor de 2-3 meses y se repite por cada nuevo corredor que quiere integrarse.

Lo mencionado anteriormente implica que cada nueva integración tiene un costo alto para la compañía, lo que incide en que las integraciones están reservadas solo para corredores que cumplan una o más de las siguientes condiciones:

- El corredor debe poder asignar un equipo de desarrollo capaz de integrar su sistema con los servicios expuestos por FID.

- El corredor debe haber realizado una integración con una o más compañías de seguros distintas de FID.

- El corredor debe presentar una oportunidad de generar ventas de seguros de FID significativas.

Estas razones conllevan a que los corredores que no cumplan estas condiciones no sean considerados para las integraciones con FID. Por lo tanto, este trabajo de memoria busca disminuir los costos de integración por cada corredor, de tal manera que se abra la posibilidad para que corredores más pequeños puedan obtener acceso a los servicios de FID.

3.1.2 Infraestructura

La infraestructura de la compañía se encuentra principalmente en GCP. En la figura 1 a continuación se muestran los principales componentes, frameworks y tecnologías utilizadas, además de cómo se comunican entre ellas. Esto es relevante porque corresponde a la base infraestructural a partir de la cual se diseñó la solución.

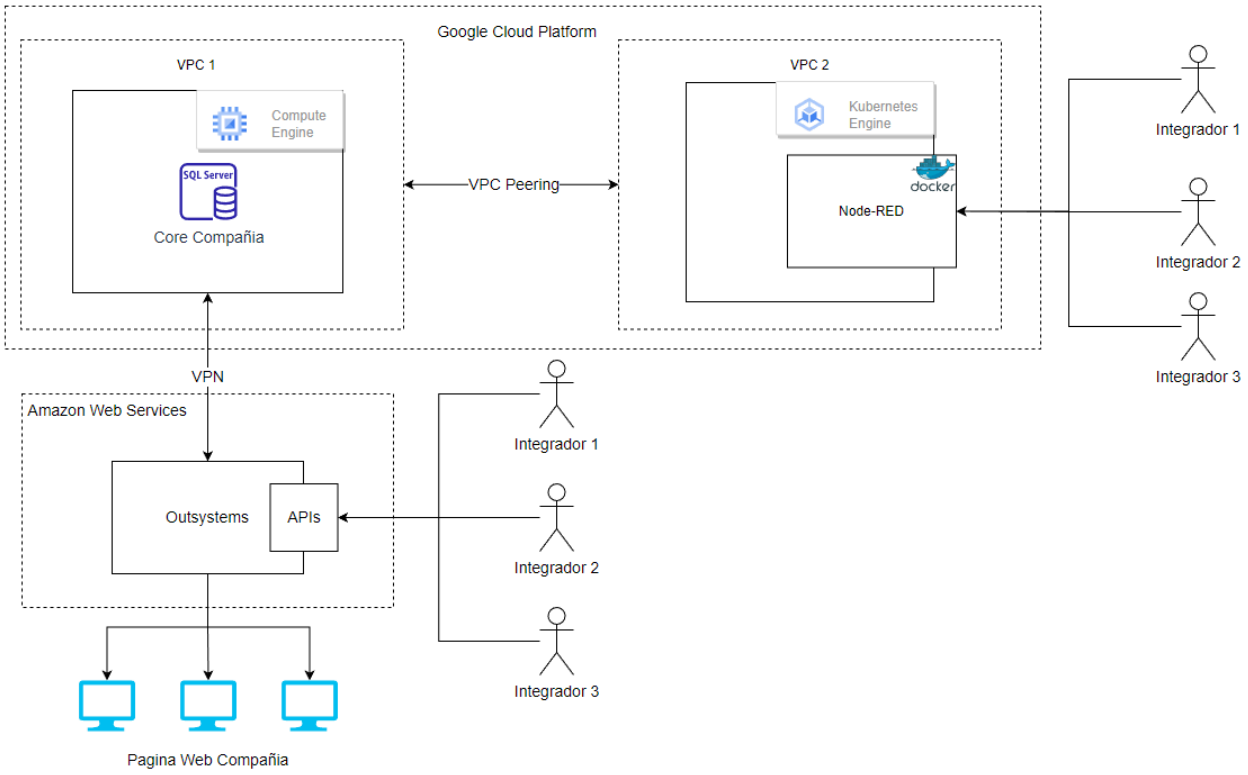

Figura 1: Arquitectura FID Seguros

A continuación, se presenta una breve explicación de los elementos más relevantes del diagrama.

Core compañía

El Core de la compañía es una solución que fue contratada a SISTRAN, la cual es una compañía especializada en la venta de soluciones para el ámbito de seguros [38]. Esta solución incluye APIs y pantallas Web que permiten realizar acciones como: emitir una póliza, registrar los datos de una persona, informar un siniestro, etc. La información de las pólizas, personas, siniestros y otros objetos del mundo de los seguros son almacenados en un modelo de datos proveído por SISTRAN que está siendo servido por un motor de base de datos de SQL Server que también es parte de la solución contratada. Este motor de base de datos será referenciado como BD a partir de este punto de la memoria.

El Core reside dentro del servicio de máquinas virtuales de GCP (Compute Engine), dentro de una VPC que será referenciada como VPC 1. Por lo que esta máquina virtual se encarga de alojar la instancia de SQL Server, las APIs y las pantallas Web.

El modelo de datos contiene alrededor de 8.000 tablas organizadas principalmente en un solo esquema. La cantidad de tablas dentro del modelo de datos está explicado por las siguientes razones:

- Muchas tablas dentro de la BD son temporales, estas son generadas por procedimientos almacenados que son ejecutados dentro del motor, pero no son eliminadas tras la ejecución de estos.
- SISTRAN incluye dentro de las tablas los modelos de datos para otras áreas dentro de los seguros. Un ejemplo de esto es como FID actualmente trabaja dentro del ámbito de seguros generales, pero el modelo también incluye las tablas y las relaciones necesarias para seguros de vida.
- A partir de las tablas base del modelo de SISTRAN se agregaron tablas adicionales que actúan como una expansión de estas, incluyendo columnas que contienen información que FID utiliza y no está incluida dentro del modelo base.

Las tablas principales en este contexto contienen datos fundamentales relacionados con el negocio de seguros. Un ejemplo de ello sería una tabla que contiene información sobre personas que estén relacionados con algún aspecto del negocio de seguros, como un contratante de una póliza o un liquidador de siniestros. Dentro de estas tablas, se utilizan códigos para categorizar la información. Por ejemplo, existe una tabla llamada "pv_header" donde está almacenada la información general de las pólizas, dentro de esta tabla existe una columna llamada "cod_ramo", la cual indica a qué agrupación de riesgo o ramo pertenece una póliza, siendo ejemplos de estos vehículo, incendio, sismo, etc. Esta columna es una llave foránea hacia la tabla "tramo", en donde esta última contiene los códigos asociados a diferentes ramos, acompañados de sus respectivas descripciones. En este caso, "tramo" actúa como una tabla auxiliar.

Adicionalmente, el motor de BD contiene alrededor de 9.000 procedimientos almacenados, en donde sus usos son variados: algunos ejecutan consultas y devuelven los resultados, mueven/modifican información dentro de la BD o generan vistas para simplificar la presentación de los datos. Estos procedimientos son utilizados por las APIs y pantallas Web expuestas por el Core para poder realizar sus funciones.

Los servicios de FID obtienen la información directamente de la BD sin utilizar las APIs proveídas por el Core. Sin embargo, no existe un estándar para la forma de obtener esta información, dependiendo del sistema que se está revisando, este puede utilizar procedimientos almacenados para obtenerla o realizar consultas directas al motor. Esta discrepancia ocurre porque inicialmente la compañía estaba siguiendo la política de utilizar procedimientos almacenados. Con la llegada de un arquitecto de software, se definió que desde ese momento en adelante solo se utilizarían

consultas directas, por lo que cualquier servicio desarrollado antes de la llegada del arquitecto fue implementado usando procedimientos almacenados.

El Core de la compañía ha presentado múltiples problemas de performance, los cuales pueden ser atribuidos a las siguientes razones:

- No existe un gestor de base de datos dentro de la empresa encargado de realizar mantenimiento y optimizaciones al modelo, consultas y procedimientos. En particular, los procedimientos hacen uso de malas prácticas que afectan la eficiencia de estos. Un ejemplo de esto es el uso extensivo de cursores, lo cual no es recomendable por el efecto que tiene en la performance [39].
- Los procedimientos almacenados dentro del sistema contienen lógica de negocios dentro de estos, aumentando el tiempo que toma su ejecución.

Kubernetes Engine

Como se comentó en la sección 2.4, GKE es una herramienta de manejo de contenedores de GCP. Actualmente se usa para administrar contenedores Docker que exponen algunas APIs de la compañía. Estos contenedores residen en una VPC distinta al Core, que será referenciada como VPC 2.

Estas APIs ofrecen servicios como:

- ingresar/modificar información sobre una inspección
- solicitar una cotización para un seguro
- emitir una póliza, entre otros

Las APIs que están alojadas en GKE tienen acceso a la información del Core de la compañía por IP privada. Dado que estas APIs residen en la VPC 2 y el Core reside en la VPC 1, es necesario el uso del servicio de VPC peering ofrecido por GCP. Este es un servicio que permite la conectividad por IP Privada entre dos VPCs distintas [40] .

La información requerida para responder las solicitudes de cada API es obtenida mediante el uso de procedimientos almacenados presentes en la BD de la compañía, o mediante consultas directas de SQL.

Los servicios alojados en Kubernetes fueron desarrollados usando Node-RED, la cual es una herramienta open source y low-code basada en NodeJS [41]. Node-RED es una plataforma low-code, lo que significa que reduce al mínimo el desarrollo de código de forma manual, porque ciertos componentes ya vienen construidos y preconfigurados. Estos se combinan mediante una interfaz visual, usando características integrables mediante drag and drop, etc. [42]. En el caso particular de Node-RED, las características integrables corresponden a “nodos” que ejecutan una función en particular, como por ejemplo, una serie de condicionales. Estos “nodos” son luego

unidos usando cables para especificar el flujo de ejecución [41]. La mayoría de los servicios presentes en la compañía fueron desarrollados con esta herramienta, esto incluye tanto a los servicios expuestos a terceros, como a los que son utilizados por los sistemas internos de FID, por lo que corresponde a la herramienta más utilizada dentro de la compañía para el desarrollo de software.

Outsystems

Es una plataforma low-code contratada como software como servicio o SaaS, por sus siglas en Inglés. Saas es un modelo de entrega de software basado en la nube, en el que el proveedor de la nube desarrolla y mantiene el software de las aplicaciones en la nube, proporciona actualizaciones automáticas del mismo y lo pone a disposición de sus clientes a través de Internet con un sistema de pago por uso [43].

Outsystems en particular es una herramienta que funciona como backend y frontend para el desarrollo de aplicaciones Web, actualmente se usa como servidor de las pantallas de la página Web de la compañía, junto con exponer algunas de las APIs utilizadas por los integradores. La razón por la cual algunos de los servicios son expuestos por Outsystems es que en los primeros meses de la compañía ciertos equipos desarrollaban en Outsystems y otros en Node-RED, esto debido a que no había un estándar bien definido. Actualmente, la instrucción del arquitecto es que cualquier nueva API debe ser desarrollada en Node-RED.

Una de las funcionalidades expuestas actualmente por Outsystems es “el portal de corredores”. En este portal, los corredores tienen acceso a diversas funcionalidades, tales como, consultar sus pólizas administradas, ver su cartelera de clientes, consultar cuotas impagas, etc. Este portal de trabajo es usado regularmente por los corredores de seguros, por lo que están acostumbrados a su funcionamiento. El acceso a este es manejado por FID, y permite controlar de manera granular a qué módulos del sistema cada corredor tiene acceso. Cada corredor posee un usuario que tiene asignado ciertos roles, estos roles definen a qué módulos tiene acceso el usuario, limitando o habilitando qué funcionalidades están permitidas para cada corredor. Estos roles y usuarios son manejados mediante un panel de control dentro del sitio, el cual es manejado por un administrador del sistema.

Con respecto a la conectividad, dado que Outsystems reside en AWS (Amazon Web Services) se utiliza una VPN (Virtual Private Network), la cual corresponde a una conexión de red privada entre dispositivos a través de Internet [44]. En el caso de FID, la VPN es utilizada para que Outsystems pueda acceder a la información de la BD de la compañía de manera segura, la cual se obtiene realizando consultas directas en el motor o mediante procedimientos almacenados dependiendo del servicio.

3.1.3 Prácticas de desarrollo

Con respecto a las APIs ya existentes, estas han sido desarrolladas bajo el esquema REST y se revisaron alrededor de 5 APIs para esta memoria. Estas corresponden a todos los servicios

actualmente expuestos a corredores e inspectores como RAC o LET, de manera que sean lo más representativos de la calidad del código usado por integradores.

Al examinar los servicios se identificaron varias inconsistencias en las APIs ya desarrolladas y las prácticas definidas en el esquema REST, en donde cada API presenta una o más desviaciones de lo recomendado. Sumado a esto, se nota una falta de consistencia en cuanto a los lineamientos de desarrollo. Por ejemplo, el formato usado para nombrar variables es distinto entre APIs, por lo que no se está siguiendo un lineamiento definido por FID.

Estos aspectos son relevantes dado que afectan la experiencia de los integradores al interactuar con las APIs, y se espera que al seguir las recomendaciones REST y mantener una consistencia en los servicios expuestos, se puede lograr una mayor facilidad de uso de los servicios entregados por FID, disminuyendo la ambigüedad sobre la función de cada endpoint.

Adicionalmente, como se mencionó en la Introducción, este proyecto es el primer acercamiento de la compañía al movimiento de Open Insurance, y es parte de una estrategia a largo plazo para generar canales de ventas y mejores servicios para el cliente final. Es por esto que la integración de terceros a estos servicios será impulsada por el área comercial, lo que podría llevar a que las APIs desarrolladas en el proyecto tengan una mayor exposición que los servicios ya existentes. Este punto es relevante porque el hecho de publicar APIs que tengan malas prácticas de desarrollo proyecta una mala imagen de la empresa, desincentivando la integración de terceros y yendo en contra del objetivo principal del proyecto.

A continuación se presenta un detalle de los problemas encontrados en las APIs examinadas.

Uso incorrecto de verbos HTTP

El primer problema encontrado es que no se están utilizando los verbos HTTP correctos para describir las acciones realizadas por un cierto método de la API. Por ejemplo, el método POST se debería utilizar para crear un nuevo recurso [45]. Sin embargo, en múltiples APIs, este verbo está siendo utilizado para operaciones de obtención de información, lo que implica que existan métodos como:

POST /obtenerInspeccion

En este tipo de situaciones, el método correcto debería ser GET.

Verbos en URIs

Algunas URIs definidas incluyen el verbo de la acción que se quiere realizar, a pesar de que sea redundante con la acción asignada al verbo HTTP. REST recomienda utilizar sustantivos al momento de definir los recursos a acceder [46]. Volviendo al ejemplo anterior:

GET /obtenerInspeccion

La palabra “obtener” dentro del recurso no provee ninguna información adicional sobre el método, dado que se entiende que una operación GET corresponde a la obtención de información.

URIs no organizadas por subrecursos

En múltiples APIs se observa que no existe una organización jerárquica de recursos y sub recursos, por ejemplo:

GET /polizas

GET /items?poliza_id=10

En el segundo endpoint se quiere acceder los ítems de una póliza en particular, para esto se otorga el id de la póliza como parámetro de query. Como se comentó en la sección 2.2.1, es posible utilizar una jerarquía de sub-colecciones para indicar que un recurso está "contenido en" otro.

REST recomienda utilizar el siguiente formato para lograr la misma función [46]:

GET /polizas

GET /polizas/{poliza_id}/items

El formato inicialmente presentado no utiliza sub-colecciones, lo que conlleva a que la organización jerárquica de los recursos no sea intuitiva para el usuario, afectando la usabilidad de esta.

Consistencia en nombramiento de parámetros

Se nota que no existe una regla a nivel empresa que dictamine qué formato deben tener los nombres de los parámetros de las APIs, los formatos más populares actualmente corresponden a [47]:

- Camel Case: Se escribe la primera palabra con minúscula y cualquier palabra adicional comienza con una mayúscula.
- Snake Case: Todas las palabras son con minúsculas y están separadas por un guión bajo.
- Pascal Case: Todas las palabras empiezan con mayúsculas y el resto con minúsculas.

En las APIs revisadas, se encuentra una mezcla de Camel Case, Snake Case y Pascal Case dependiendo de la API. Es importante que esta situación no ocurra en las APIs del proyecto, porque de lo contrario se arriesga transmitir que la compañía presenta un desorden con respecto a las prácticas de desarrollo internas de la empresa. Esto eventualmente podría llevar a los integradores a no hacer uso de las APIs de FID.

Ausencia de mecanismo de paginamiento en APIs de consulta

En general, las APIs revisadas no permiten limitar los resultados devueltos por una API de consulta. Por lo que existe una oportunidad para incorporar mecanismos de paginamiento al entregar los resultados de la consulta, ayudando a reducir la latencia y optimizar el rendimiento del servidor, especialmente cuando se solicitan y transmiten grandes cantidades de datos [48].

Reutilización de usuarios de BD

Las APIs al acceder a la BD de la compañía para trabajar sobre la información utilizan credenciales de BD. En las APIs revisadas se encontraron casos en donde estas credenciales son el usuario y contraseña personales de desarrolladores. Sumado a esto, estas mismas credenciales estaban siendo utilizadas por múltiples APIs al mismo tiempo.

Adicionalmente, se encontraron casos en donde los usuarios de BD usados por las APIs tenían más permisos de los que son mínimamente necesarios para su funcionamiento. Un ejemplo de esto es como una API de consulta de información de pólizas usaba un usuario que poseía permisos de escritura, siendo que la API solo necesitaba permisos de lectura para realizar su función.

Esto es un problema porque aumenta el impacto que puede generar la filtración de alguno de estos usuarios, cuando esto podría minimizarse para cubrir solo lo mínimamente requerido por el sistema para poder funcionar.

Uso incorrecto de códigos de estado HTTP

Se encontró evidencia de que los códigos de estado HTTP no están siendo usados correctamente en las respuestas, el ejemplo más común de esto es que están utilizando el código 200 (OK) al momento de indicar que falta un parámetro obligatorio en la petición. Para estos casos el código correcto es el 400 (Bad Request).

Un ejemplo de esto corresponde al siguiente endpoint:

POST /PostInspecciones

En este endpoint el parámetro “Vehiculo” es obligatorio dentro del cuerpo de la solicitud. Si este no se envía, la API entrega la siguiente respuesta:


```
{
  "Codigo": 400,
  "ExcepcionMensaje": "Campo Vehiculo es obligatorio"
}
```


Sin embargo, el código de estado devuelto es un 200 (OK), esto a pesar de que en el cuerpo de la respuesta se informe como un 400.

Esta inconsistencia abre la posibilidad de que integradores califiquen errores como solicitudes correctamente ejecutadas dentro de sus desarrollos.

3.1.4 Método de autenticación

Para mantener seguras las APIs expuestas a Internet, la compañía distribuye API keys a los integradores, en donde el concepto de API Key fue introducido en la sección 2.2. Esta es enviada

como un parámetro al momento de consultar las APIs expuestas, otorgándoles acceso a las funcionalidades.

Dependiendo de la API, esta verificación se hace de 2 maneras, las cuales son presentadas a continuación.

Verificación por software

En este caso la autenticación es realizada dentro de la API. Al momento de recibir una solicitud, se hace una comparación entre la API Key recibida y el valor configurado para esta. Si existe alguna diferencia, se deniega el acceso a la API.

Esta alternativa tiene la desventaja de que múltiples integradores ocupan la misma API key para acceder al servicio, por lo que no existe trazabilidad de quien está realizando cada llamado a la API. Además con esta opción no hay una modularización entre la autenticación y la funcionalidad de la API, afectando la escalabilidad de esta.

Verificación por Google Endpoints

Este caso corresponde a cuando la autenticación es realizada por un servicio de GCP llamado Google Endpoint. En donde Google Endpoint es un sistema de administración de API que ayuda a proteger, supervisar, analizar y establecer cuotas para las APIs propias, mediante la misma infraestructura que usa Google para sus propias API [49]. Para autenticar una solicitud la API key es enviada como parámetro de la solicitud y es validada por la componente de la nube de Google, permitiendo o denegando el acceso antes de ingresar a la API.

Esta alternativa tiene la ventaja de que permite generar múltiples API keys para acceder al mismo servicio, permitiendo dar trazabilidad y monitorear quienes y en qué volumen están consumiendo los servicios expuestos. Además, dado que la autenticación está separada del código principal de la API, se logra la modularización de las componentes, dando como resultado una mayor escalabilidad.

Actualmente cualquier nuevo desarrollo de APIs debe incluir este tipo de autenticación, dado que es un lineamiento establecido por el arquitecto de la compañía.

3.2. Análisis

La presente sección detalla el proceso de análisis de la solución, en donde revisaron las condiciones particulares del proyecto y se discutieron aspectos a considerar al momento de diseñar la solución final.

El análisis presentado es un trabajo colaborativo entre el memorista, el gerente de finanzas y tecnología, la subgerenta de tecnología, el arquitecto, el jefe de sistemas, 2 desarrolladores de la empresa y un consultor externo. En particular, el memorista fue el encargado de planear los temas de discusión en las reuniones semanales descritas en la sección 1.5 y guiar al equipo de trabajo en base a los comentarios expuestos en esas reuniones.

3.2.1. Acceso a datos

En esta sección se discuten los aspectos a considerar con respecto al proceso de autenticación y acceso a los datos, explicando el tipo de usuario objetivo y revisando posibles opciones para asegurar la confidencialidad de la información.

Tipos de integradores

Uno de los requisitos del sistema a desarrollar es que soporte la integración de los corredores de FID a los servicios expuestos. Sin embargo, el sistema debe ser escalable para acomodar distintos tipos de integradores que podrían querer ser soportados en el futuro, en donde integradores se refiere al sistema o usuario que está haciendo uso de las APIs a exponer.

Para esto se analizaron y discutieron los posibles usuarios que podrían actuar como integradores los cuales serán descritos a continuación.

Integrador Corredor

Corresponde a los integradores que son corredores de la compañía o son un sistema que pertenece a ellos, en este caso el integrador sólo podría tener acceso a la información que es normalmente visualizable por el corredor desde su pantalla de administración en el portal de corredores.

A continuación, en la figura 2 se muestra un caso de múltiples integraciones de este tipo de integrador.

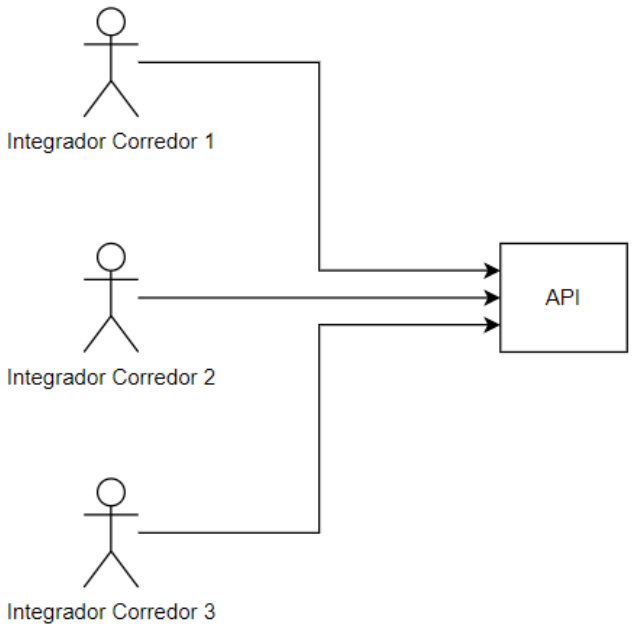

Figura 2: Integrador Corredor

Un ejemplo de este tipo de integrador corresponde a un corredor que está vendiendo seguros de FID y quiere tener un panel de administración personalizado en algún sistema suyo, este panel puede ser alimentado con la información devuelta por las APIs expuestas por FID.

Integrador representando a corredor

Corresponde a los integradores que no son corredores de la compañía o sistemas que no pertenecen directamente al corredor, pero han sido autorizados para actuar u obtener información por uno o más corredores de FID. La información a la que pueden acceder depende del grado de permisos que se le haya sido otorgado por los corredores y la información que cada uno de estos puede visualizar normalmente desde el portal de corredores.

A continuación en la figura 3 se muestra el caso donde 1 integrador ha recibido autorización de 3 corredores para acceder a las APIs de FID, permitiéndole realizar solicitudes a los servicios en representación de ellos.

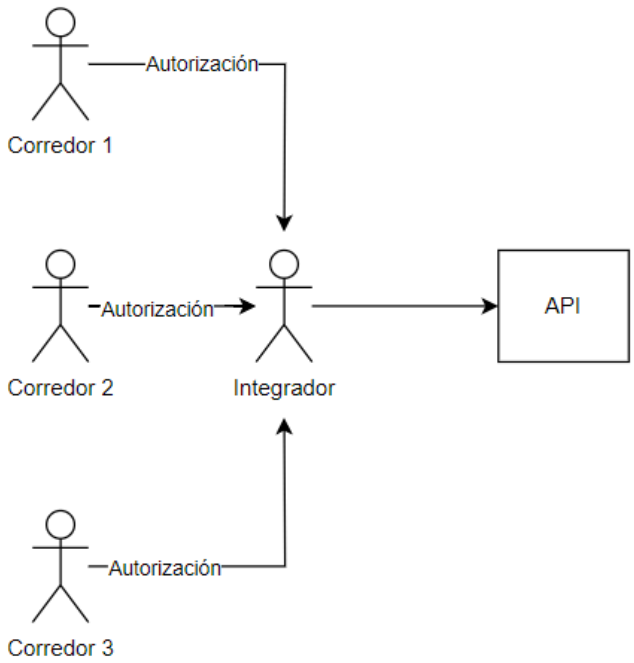

Figura 3: Integrador representando a corredores

Un ejemplo de este tipo de integrador podría corresponder a Brokeris, el cual es una plataforma que ofrece funcionalidades de gestión para corredores [50]. En este caso los corredores de FID otorgan permiso a Brokeris para acceder a su información, y Brokeris puede luego usar esta información para alimentar una pantalla de gestión que después es utilizada por el corredor.

Confidencialidad de la información

La confidencialidad de la información es un factor importante a considerar en el diseño del sistema de autenticación, dado que se quiere evitar que un integrador utilice los servicios expuestos para obtener información a la que no debería tener acceso. Debido a esto el sistema de autenticación debe ser capaz de:

- Identificar al sistema/usuario realizando la solicitud
- Identificar al corredor/cliente que le otorgó los permisos al integrador para poder actuar en su nombre o ver información relacionada a sus pólizas

En particular se consideraron las siguientes opciones para manejar estos requerimientos

- Json Web Token (JWT)
- API key con metadata

A continuación, se discuten ambas alternativas.

Json Web Token (JWT)

Corresponde a un estándar que permite validar que la información enviada entre dos partes no ha sido modificada mediante el uso de firmas digitales.

Implementar esta opción requiere que dentro de la solución se incluya un sistema que permita generar JWTs. Los integradores usarían este sistema para solicitar JWTs con su información que les permitan acceder a los servicios expuestos.

Esto tiene la ventaja de que permite centralizar las solicitudes de autenticación en un sistema externo a los servicios a exponer, promoviendo la separación de responsabilidades en las componentes. Adicionalmente reduce el procesamiento necesario por cada solicitud, dado que no es necesario re-autenticar las solicitudes si el integrador está usando un JWT que aún es válido. Sumado a esto, los JWT permiten asegurar que la información contenida en el payload no ha sido modificada. Esto es logrado mediante la firma digital incluida al final de cada JWT, cualquier cambio en el payload causaría que la firma digital deje de ser válida, invalidando el JWT.

Sin embargo posee la desventaja de que la información dentro del token solamente está firmada, pero no encriptada. Esto implica que si un tercero obtiene acceso a un JWT, puede recuperar la información guardada en este e identificar a quien realizó la petición u obtener indicios sobre el funcionamiento interno del mecanismo de autenticación de FID. Esto es de particular relevancia ya que los sistemas que estarán interactuando con las APIs son construidos por equipos de desarrollo que no van a estar siendo monitoreados/auditados por FID, por lo que no se puede asegurar que el sistema que utilice el JWT no esté expuesto a una vulnerabilidad de seguridad. Adicionalmente, dado que la información dentro del token no está encriptada, FID debe considerar que cualquier información que este contenga puede ser leída por los integradores, lo cual limita a FID sobre qué información en particular puede ser añadida al JWT.

API Key con metadata

Algunos gateways tienen la posibilidad de asignar metadata a cada API Key, de modo que cada vez que una solicitud es autenticada por el gateway, esta información sea añadida a la solicitud antes de entrar a la API.

La posibilidad o no de implementar esta opción depende de las capacidades del gateway que se quiera utilizar, o de otro modo se estaría forzado a desarrollar un gateway que incluya esta característica.

Esta alternativa tiene la ventaja de que la información almacenada como metadata es completamente oculta al integrador, y es solo accesible por los servicios internos de la compañía, protegiendo la identidad del integrador y corredor involucrados en la solicitud y otorgándole más libertad a FID con respecto al tipo de información que puede ser asignada como metadata.

La desventaja es que esta opción limita la selección de la herramienta de gateway a una que posea esta funcionalidad.

Luego de considerar las ventajas y desventajas de ambas alternativas se optó por ocupar API keys con metadata, principalmente por las siguientes razones:

- Restricciones en la información contenida: Dado que la información en un JWT no está encriptada, cualquier información a guardar debe ser considerada como pública para los integradores, lo que limita el tipo de información que puede ser ingresada en el JWT.
- Facilidad de integración: Desde el punto de vista de los integradores, el desarrollo necesario para realizar consultas para las APIs es más simple con la autenticación por API key que por un flujo de autenticación que ocupe JWT.
- Menor exposición de la información: Si algún sistema desarrollado por un integrador posee una vulnerabilidad de seguridad y se está usando un sistema de autenticación por JWT, el atacante tiene total acceso a la información contenida en este. En cambio, si el sistema de autenticación está basado en API keys con metadata, la información asociada a la API Key no es accesible para el atacante, dado que esta es solo agregada una vez que es autenticada por el gateway. En esta situación una API Key con metadata protege a FID de las vulnerabilidades de ciberseguridad que puedan existir en los sistemas de los integradores.

Uso de herramienta de API Management

El proyecto requiere de un sistema de autenticación y enrolamiento para los integradores para cumplir con los objetivos definidos. Para acelerar el proceso de desarrollo se decidió contratar una herramienta de API Management que provea estos servicios. El uso de una herramienta de API Management facilita el desarrollo de estas componentes dentro de la solución implementada.

3.2.2. Performance del Core

Uno de los objetivos específicos del proyecto es que el tráfico generado por las nuevas APIs no afecte significativamente la performance del Core de la compañía. Como se mencionó en la sección 3.1.2, la BD utilizada por la compañía es parte de una solución contratada a SISTRAN. La solución del Core está pensada para que el motor de BD, las APIs y las pantallas Web de esta residan en la misma máquina virtual. Esto implica que no fue posible aislar la BD dentro de un servicio Cloud, como Cloud SQL por ejemplo. En donde Cloud SQL corresponde a un servicio gestionado por Google para BD relacionales [51]. De haberse levantado la BD en Cloud SQL en vez de una máquina virtual se hubiera facilitado el mantenimiento y el escalado de esta [51].

Para evitar estas limitaciones, se decidió levantar una segunda BD en Cloud SQL. Esta segunda BD contendría la información actualizada de la BD del Core y tendría los beneficios descritos anteriormente. Los datos contenidos en la segunda BD serían un subconjunto del modelo de datos presente en el Core, sin modificaciones. Esta estrategia permitiría diseñar las APIs de tal manera que la información sea obtenida de la segunda BD, disminuyendo el impacto sobre la performance del Core.

Es importante recalcar que esta segunda BD no puede ser simplemente una réplica del Core, esto debido a que la segunda BD debe contener la información lo más actualizada posible. Esto significa que si en el Core se añade/borra/modifica un registro, este cambio debe verse reflejado en la segunda BD.

Para manejar este requisito se decidió diseñar una solución basada en una herramienta extract, transform and load, o ETL por sus siglas en Inglés. Una herramienta de ETL permite extraer y mover datos desde múltiples fuentes, transformarlos y cargarlos a un destino [52]. Esta herramienta permitiría tomar información desde el Core de la compañía, y transferirla a la segunda BD en Cloud SQL para que esta contenga la información lo más actualizada posible. Hasta este punto en la memoria, se están utilizando exclusivamente las funciones de extracción y carga de la ETL. La funcionalidad de transformación cobra relevancia en la sección 3.2.4, por lo que durante el proyecto se aprovechan todas las capacidades de la herramienta.

Se nota que la información de la segunda BD no estaría completamente igual a la del Core debido al desfase de la migración de la información, pero se definió que esto es un compromiso adecuado dentro de la solución siempre y cuando el tiempo de desfase no sea mayor a 10 minutos.

3.2.3. Selección de herramientas y tecnologías

Para definir qué herramientas de API Management y ETLs iban a ser utilizadas se revisaron los análisis realizados por la compañía de consultoría Gartner, la cual tiene como misión su misión habilitar decisiones rápidas e inteligentes mediante su guía de expertos [53]. Para este análisis se revisó su clasificación de competidores para API Management y ETLs mediante el Gartner Magic Quadrant, el cual busca resumir el proceso de análisis realizado por ellos dentro de un mercado tecnológico en particular [54]. El acceso a este documento fue proveído por FID dado que

corresponde a una fuente privada. Sumado a esto se realizó una búsqueda de información en línea, se mantuvieron conversaciones con equipos de venta de las herramientas y se levantaron proof of concept, o POCs por sus sigles en Inglés, en caso de ser posible. En donde una POC es una prueba que permite comprobar la viabilidad técnica de una idea, por medio de la evidencia de su funcionalidad y potencial [55].

A continuación, se presentan los resultados de esta exploración.

Revisión de API Managers

Como se mencionó en el párrafo anterior, se utilizó el Gartner Magic Quadrant para guiar la elección de la herramienta de API Management. Esta clasificación realizada por el cuadrante mágico se muestra en la figura 4.

Figura 4: Cuadrante mágico de Gartner para API Managers (Proveído por FID Seguros)

El diagrama clasifica a las herramientas en cuatro cuadrantes, los cuales corresponden a [54]:

- Leaders: Ejecutan bien respecto a su visión actual y están bien posicionados para el futuro.
- Visionaries: Entienden donde el mercado está apuntando o tienen una visión para cambiar las reglas del mercado, pero no la han ejecutado bien.
- Niche Players: Han enfocado sus esfuerzos en un pequeño segmento de manera exitosa, o no están enfocados y no innovan o rinden mejor que otras opciones.

- Challengers: Ejecutan bien al día de hoy o podrían dominar un gran segmento, pero no demuestran un entendimiento de la dirección del mercado .

Las herramientas son mejor evaluadas por el cuadrante entre más arriba y a la derecha se encuentren frente a las otras opciones.

De las opciones revisadas destacan como líderes las herramientas de Apigee, MuleSoft y Kong.

Adicionalmente, para complementar la información se trabajó con la consultora Prodigio, compañía que define su trabajo de la siguiente manera "Trabajamos con su organización en desarrollar, acelerar y rentabilizar capacidades tecnológicas para transformar su negocio en un negocio digital." [56]. Se discutió el objetivo y alcance del proyecto con ellos, y a partir de esta información nos recomendaron considerar a Tyk como opción a revisar. Esta herramienta fue clasificada dentro de "Visionaries" dentro del Gartner Magic Quadrant.

A continuación, se presenta un resumen de las opciones consideradas:

- MuleSoft: Cumple con los requisitos buscados como API Manager, y es considerada la mejor herramienta dentro del cuadrante de gartner. Adicionalmente sumado a las funcionalidades de API Manager actúa como plataforma de desarrollo low-code, con versionamiento y manejo de extensiones incluido. Sin embargo, al consultar por el coste de la herramienta, este sobrepasaba los 100 millones de pesos mensuales, un valor que estaba fuera del presupuesto del proyecto según el gerente de finanzas y tecnología.

- Apigee: Cumple con los requisitos buscados como API Manager, además posee la ventaja de ser una herramienta que pertenece a Google. Esto asegura que haya compatibilidad para conectarse con los servicios de GCP que son usados en la compañía. Sin embargo, su coste estaba alrededor de 100 millones de pesos mensuales, un valor que estaba fuera del presupuesto del proyecto según el gerente de finanzas y tecnología.

- Kong: Cumple con los requisitos buscados como API Manager. Esta alternativa requiere instalar la infraestructura necesaria para su funcionamiento, dado que no sigue un esquema SaaS. Esto implica que elegir esta opción requiere realizar la instalación de la herramienta dentro de la nube de la compañía. En cuanto al precio de la herramienta, se mandaron múltiples correos al área de ventas para solicitar una versión de prueba y consultar precios, pero ninguno de estos fueron respondidos por Kong, generando preocupaciones por la calidad del servicio al cliente incluso antes de poder probar la herramienta.

- Tyk: Cumple con los requisitos buscados como API Manager, es notablemente más económico que las otras opciones. Sin embargo, su posición en el cuadrante de gartner no está al mismo nivel que las otras opciones, y al ser un producto relativamente nuevo no está tan refinado como las otras alternativas.

Sumado a lo anterior, se tuvo la posibilidad de probar tanto Apigee como Tyk, dado que contaban con versiones free-trial, por lo que se pudieron realizar POCs de posibles diseños del sistema de autenticación.

Para las herramientas de Apigee y MuleSoft, estas tuvieron que ser descartadas, dado que su coste estaba fuera del presupuesto del proyecto.

El API Manager Kong generó desconfianza en cuanto a la calidad del servicio al cliente, debido al constante envío de correos que no fueron respondidos por el equipo de Kong cuando se intentó iniciar el contacto para conocer más sobre la herramienta.

Dado que Tyk cumple con las funcionalidades requeridas como API Manager y teniendo en cuenta los problemas encontrados con las otras alternativas, se decidió seleccionar Tyk como la herramienta final.

Existe un nivel de riesgo al contratar una herramienta relativamente nueva dentro del mercado, pero la clasificación en "Visionaries" dentro del cuadrante de Gartner, además de la POC realizada entrega la confianza suficiente para justificar la decisión.

Revisión de ETLs

Como se mencionó en la sección 3.2.2, la estrategia para mitigar el impacto en la performance del Core requiere el uso de una herramienta de ETL.

Se definió que la herramienta a elegir debía ser capaz de ser instalada dentro de la nube de la compañía, para que los datos a migrar en ningún momento estuvieran fuera de la red interna por temas de seguridad, cualquier opción que no cumpliera con este criterio sería automáticamente descartada.

Sumado a esto, se definieron las siguientes directrices para tomar la decisión:

- Precio.
- Tiempo mínimo entre cada ejecución de un proceso de migración, de manera que la información en la segunda BD no tenga un desfase mayor a 10 minutos.
- Conectores con distintos repositorios de datos, dado que en el futuro se podría querer mover datos entre otras fuentes de datos. Un ejemplo de esto es BigQuery, un servicio de GCP del almacenamiento de datos empresarial que provee funciones de análisis de datos [57].
- Complejidad de implementación.
- Infraestructura Requerida.
- Complejidad de la operación.

Para las herramientas de ETL se consideraron las siguientes opciones:

- Hevo Data
- Talend
- Stitch Data
- Cdata
- Informatica

Se conversó con los equipos de venta de las herramientas y se buscó información en línea de sus características. A partir de esto se descartaron las opciones de Hevo Data, Stitch Data, Cdata e Informatica, dado que todas estas corresponden a soluciones Cloud. Esto implica que la transmisión de la información debía dejar la red privada de la compañía durante la ejecución del proceso de migración.

Dado lo anterior, Talend quedó como única opción dentro de las inicialmente consideradas.

Aunque es importante notar que a pesar de que las otras alternativas fueron descartadas por ser plataformas Cloud, Talend seguía teniendo un cumplimiento más completo de las directrices definidas en comparación a las otras opciones. Estas características ya hacían que estuviera siendo considerada como la opción preferida independiente del problema con las otras alternativas.

Se contactó al equipo de ventas de Talend para solicitar una versión free-trial para poder confirmar las funcionalidades y ver si eran adecuadas para el proyecto. A raíz de estas reuniones, se le otorgó la posibilidad al equipo de desarrollo de utilizar 2 semanas Talend Academy, una plataforma diseñada para ayudar con el proceso de onboarding de la herramienta, en donde se pueden realizar tutoriales que terminan con un "hands-on" para poder aplicar lo aprendido [58]. Con esto el equipo de desarrollo se pudo familiarizar con Talend y se confirmó que la herramienta cumplía con lo necesario para implementar las migraciones diseñadas, por lo que se eligió como la opción adecuada para el proyecto.

3.2.4. Definición motor de BD

Como se comentó en la sección 3.2.2, se decidió levantar una segunda BD usando el servicio de Cloud SQL de GCP.

El servicio de Cloud SQL soporta tres motores de BD:

- MySQL
- PostgreSQL
- SQL Server

La BD del Core de la compañía está en un motor de SQL Server, pero se consideró la posibilidad de usar otro motor para la nueva BD. Es por esto que se realizó una exploración de las ventajas y desventajas de cada opción, las cuales concluyeron en la elección de MySQL como el motor para la nueva BD, las razones fueron las siguientes:

- Menor coste: Dado que tanto MySQL como PostgreSQL cuentan con una licencia Open Source, el uso de estos motores dentro de Cloud SQL tiene un coste significativamente menor en comparación al uso de SQL Server [59].

- Funcionalidades: Las funcionalidades soportadas por Cloud SQL varían entre motor de BD. En el momento que se realizó esta memoria, se identificaron varias funcionalidades que no eran soportadas en SQL Server ni PostgreSQL, pero sí en MySQL [60]. De estas funcionalidades, la más relevante y la que más influyó en la decisión es la capacidad de generar réplicas de lectura. Esta funcionalidad permite a la BD escalar horizontalmente para manejar consultas de información, lo cual se consideró una cualidad fundamental al momento de elegir el motor.

Dado que esto implica un cambio de motor entre el origen y el destino se revisaron las consideraciones necesarias y se realizó una POC en donde se leyeron, transformaron y se almacenaron los datos de manera exitosa finalmente cerrando la elección de MySQL como motor.

3.2.5. Enrolamiento y generación de API keys

Enrolar está definido por la RAE cómo “Alistarse, inscribirse en el Ejército, en un partido político u otra organización.” [61], en el contexto de esta memoria se refiere específicamente a la acción de inscribir a un corredor para que pueda generar API keys de acceso a los servicios de FID.

Uno de los objetivos principales del proyecto es proveer un proceso de enrolamiento y generación de API keys lo más simple posible para los corredores. Es por esto que se tomó la decisión de integrar el proceso de enrolamiento dentro del portal de corredores, de esta forma se logra que las nuevas funcionalidades existan dentro de una plataforma familiar para los usuarios. Adicionalmente, esto permite a FID controlar quién tiene acceso a estas funcionalidades mediante el sistema de accesos del portal de corredores.

El módulo permite a los corredores realizar las siguientes acciones:

- Generar API Keys para el corredor, en donde inicialmente solo se permitirá generar API Keys que otorguen acceso a las 3 APIs del proyecto.

- Visualizar las API keys activas del corredor.

- Revocar API Keys del corredor previamente generadas.

El proceso de generación y revocación de API Keys será manejado por el API Manager, el cual fue elegido en la sección 3.2.3. Se decidió desarrollar un wrapper del API Manager para agregar un

nivel de abstracción entre Outsystems y el API Manager, de tal manera que se reduzca el impacto en el sistema si en el futuro se decide cambiar de herramienta de API Management.

Para poder implementar el wrapper, este debe tener acceso a un modelo de datos que contenga información sobre:

- API Keys.
- Corredores.
- APIs existentes.
- Permisos, en donde se especifica a qué APIs está otorgando acceso cada API Key.

Esta información debe almacenarse para no generar una dependencia de la información dentro de la herramienta de API Management. El modelo de datos usado por el wrapper debe contener toda la información necesaria en caso de que se quiera realizar una migración a una nueva herramienta de API Management.

Un factor a considerar es que las API Keys que van a ser guardadas deben estar encriptadas, de tal manera que si se produce una brecha de seguridad, estas no se vean comprometidas. Las API Keys a guardar deben estar encriptadas y no hasheadas para que el proceso sea reversible y se pueda obtener la API Key original. Esto es un requisito para poder implementar la funcionalidad de visualización de las API keys activas en el portal de corredores, dado que se debe revertir el proceso antes de mostrarla al usuario.

Para la encriptación se usó el algoritmo AES, dado que es considerado muy robusto ante ataques que no son fuerza bruta, siendo además el algoritmo estándar utilizado por el gobierno de los Estados Unidos [62] [63] [64].

3.3. Diseño

La presente sección detalla el diseño de solución, la cual está dividida 5 partes:

- Arquitectura de las componentes.
- Acceso a las APIs.
- Prácticas de desarrollo.
- Migración de datos.
- APIs de consulta.

La primera parte de la solución corresponde al acceso a las APIs. Esto incluye al diseño del proceso de enrolamiento de los corredores y el flujo de autenticación. Para que un corredor pueda ser enrolado este tiene que dirigirse al portal de corredores y realizar el proceso de generación de API

Keys. Una vez que ha sido enrolado y tenga una API Key para acceder a los servicios de FID, puede enviar una solicitud a las APIs siguiendo el flujo de autenticación diseñado.

Las APIs que son llamadas por los integradores fueron implementadas siguiendo las prácticas de desarrollo definidas en esté capítulo. Estas están enfocadas en estar alineadas con lo definido en el esquema REST, estandarizar las firmas de entrada y corregir vulnerabilidades de seguridad.

Una vez que empieza el procesamiento de la solicitud por las APIs, estas realizan consultas a la BD MySQL que es manejada por el servicio de Cloud SQL de GCP. Esta BD MySQL contiene una réplica de la información del Core y está constantemente siendo actualizada por la segunda parte de la solución, el sistema de migración de datos. Este sistema está encargado de aplicar cualquier cambio que ocurra sobre la información presente en el Core en la BD MySQL. Estos cambios aplicados por el sistema de migración permiten que las APIs consulten información que como máximo está desfasada 5 minutos en comparación a la presente en Core.

Los diseños presentados son un trabajo colaborativo entre el memorista, el gerente de finanzas y tecnología, la subgerenta de tecnología, el arquitecto, el jefe de sistemas, 2 desarrolladores de la empresa y 1 consultor externo. En particular, el memorista, junto con el apoyo de los 2 desarrolladores, fue el encargado de proponer los diseños iniciales. Estos diseños eran luego discutidos y refinados en las reuniones semanales.

3.3.1. Arquitectura de la solución

En esta sección se describe la arquitectura utilizada por los servicios y herramientas usadas de la solución, los cuales corresponden a:

- APIs de consulta
- Talend
- TYK

Dado que algunas componentes de las herramientas son administradas por su respectivo proveedor en su propia nube, las arquitecturas incluyen la comunicación entre componentes presentes en las nubes de GCP y AWS.

Arquitectura de las APIs

Las APIs están ejecutándose en contenedores Docker administrados por GKE, se definió que se crearía un cluster de Kubernetes distinto al que usan el resto de los servicios de la compañía para las 3 APIs del proyecto. En donde un cluster corresponde a un conjunto de máquinas de nodos que ejecutan aplicaciones en contenedores [65]. Esto con el fin de reducir el impacto de una posible brecha de seguridad, dado que estarían separados de los servicios estándar de la compañía. Sin embargo, es importante notar que tanto las APIs del proyecto como los servicios anteriormente existentes viven en la misma VPC.

Debido a que la instancia de Cloud SQL no vive en la misma VPC que los clusters de Kubernetes, es necesario incluir dentro de los despliegues de las APIs un contenedor que actúe como proxy hacia la BD [66]. Usando este proxy las APIs pueden realizar consultas a la instancia MySQL por IP Privada a pesar de no estar en la misma VPC.

A continuación, en la figura 5 se presenta un diagrama resumiendo lo mencionado anteriormente:

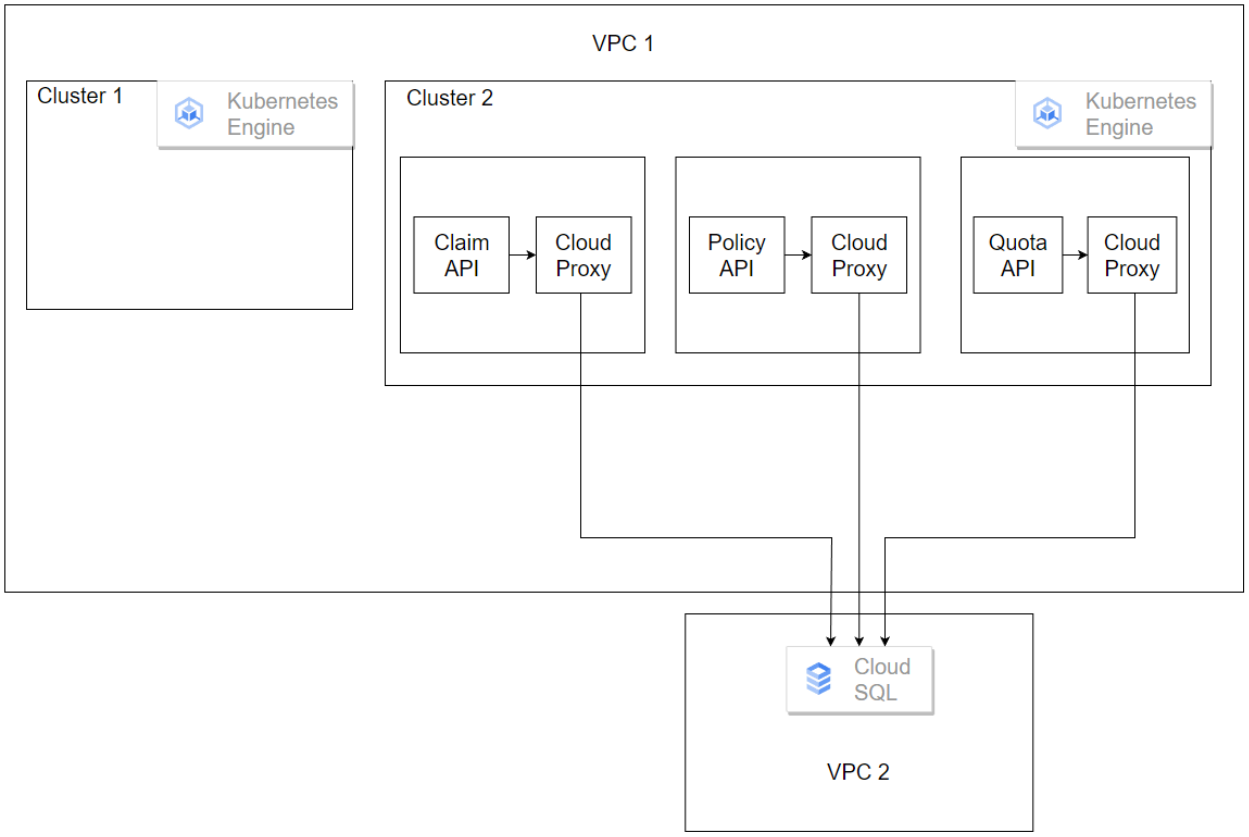

Figura 5: Arquitectura APIs

Arquitectura de Talend

Talend corresponde a la herramienta de ETL elegida en la sección 3.2.3. Si bien una componente de la herramienta reside en la nube de Amazon, la arquitectura diseñada asegura que los datos nunca salgan de la nube de FID al momento de ser migrados.

La herramienta contiene 2 componentes principales:

- Remote Engine: Corresponde a una máquina virtual que actúa como runtime para la ejecución de los procesos de migración de datos. Esta fue levantada usando el servicio de Compute Engine de GCP, la cual es el servicio de la nube de Google para proveer máquinas virtuales [67].

- Talend Cloud: Componente orquestador administrada por Talend en AWS. Esta puede ser ingresada mediante el navegador con las credenciales de la compañía. Cumple la funcionalidad de un panel de control, en donde se permite programar ejecuciones de los procesos de migración de datos, configurar el Remote Engine a utilizar, administrar ambientes, entre otras funcionalidades.

La comunicación entre estos componentes es realizada mediante HTTPS, siendo siempre el remote engine el que inicia la conexión, por lo que no es necesario habilitar conexiones entrantes para la máquina virtual donde se ejecutan los procesos de migración.

Adicionalmente, el remote engine tiene que ser capaz de conectarse tanto a la BD del Core de la compañía y la instancia Cloud SQL para poder realizar la migración de datos. Para permitir la conectividad por IP Privada entre estos, es necesario configurar el servicio de VPC peering ofrecido por Google. Dado que ambas BD se encuentran en VPCs distintas a la del remote, este servicio es necesario para lograr la conectividad.

A continuación, en la figura 6 se muestran las componentes mencionadas y cómo se comunican entre ellas:

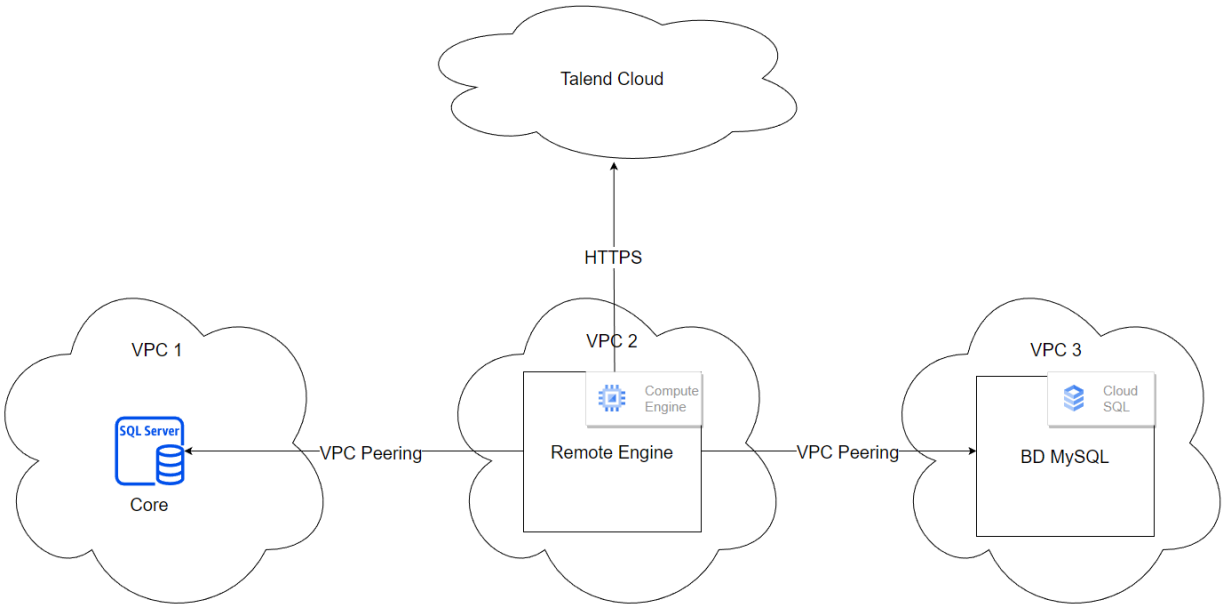

Figura 6: Arquitectura Talend

Arquitectura de Tyk

TYK corresponde a la herramienta de API Management elegida en la sección 3.2.4. Esta herramienta posee múltiples formas de ser instalada, como Cloud, Hybrid u On-premise [68] [69] [70]. Para este proyecto se decidió implementar la solución Cloud de la herramienta para facilitar

la mantención y la instalación. Esto significa que las componentes de TYK están alojadas en AWS y son administradas por el proveedor.

El plan que fue contratado por FID incluye 3 ambientes, los cuales fueron destinados a desarrollo, QA y producción. Estos ambientes son administrados mediante un panel de control accedido por el navegador Web con las credenciales de la compañía llamado TYK Cloud. Este permite realizar la configuración y despliegue de las componentes de cada ambiente.

De estos componentes, los relevantes para la memoria corresponden a:

- Dashboard: Cumple el rol de un panel de control del funcionamiento general del API Manager, hay 1 por cada ambiente. Permite realizar variadas funciones, como la generación/revocación/modificación de API Keys, crear definiciones de APIs, crear definiciones de permisos, visualizar métricas, entre otras. Adicionalmente, disponibiliza APIs que permiten utilizar las funcionalidades descritas anteriormente. Estas funcionalidades son utilizadas por el sistema de enrolamiento para realizar la generación/revocación de API Keys.
- Edge gateway: Gateway encargado de realizar la validación de las API Keys y aplicar las transformaciones a la solicitud. Su funcionamiento está basado en la configuración presente en el Dashboard, hay 1 por cada ambiente.

A continuación, en la figura 7 se muestra la organización de estas componentes y TYK Cloud:

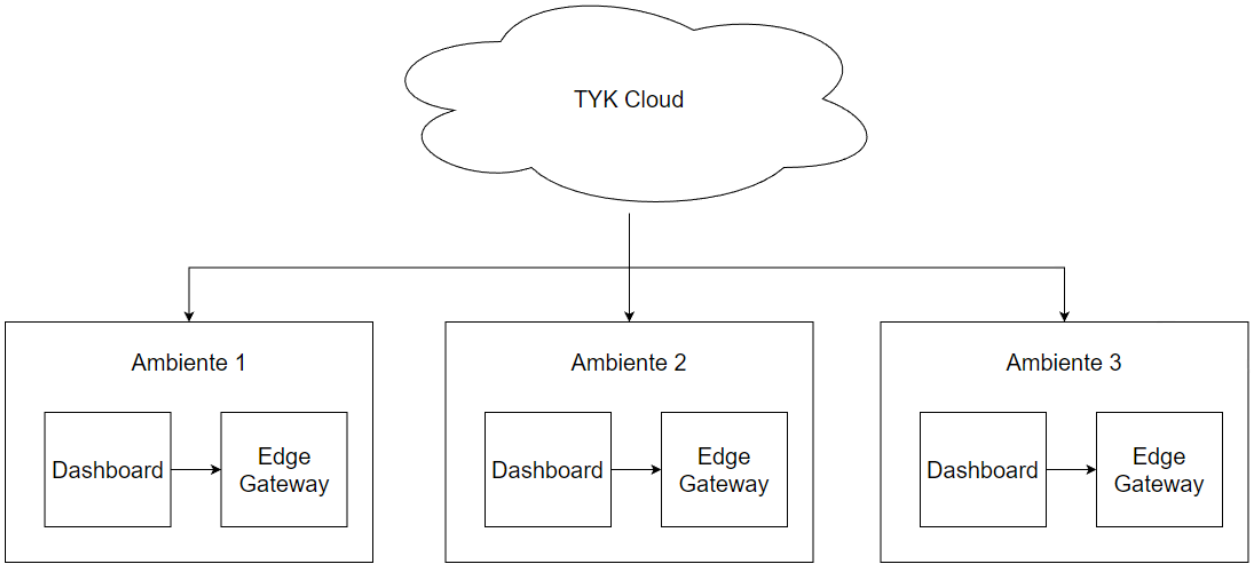

Figura 7: Organización componentes TYK

Una vez que la solicitud es autenticada, el gateway envía una solicitud mediante https a las APIs que residen en GCP. Dado que el gateway está alojado en AWS, las APIs del proyecto deben estar expuestas a internet para que puedan ser llamadas por el gateway. Para impedir el tráfico hacia las

APIs no proveniente desde el API Manager se configuró la herramienta Cloud Armor de GCP, esta corresponde a un servicio de protección de aplicaciones que protege a aplicaciones Web de ataques Denial of service (DDOS) y otras amenazas [71]. Esta herramienta permite bloquear todo el tráfico entrante y solo permitir las conexiones que cumplan con ciertas condiciones.

El Cloud Armor está asignado a las 3 APIs del proyecto (Claim, Policy, Quota) y solo permite ingresar el tráfico que cumple con las siguientes condiciones:

- La IP de origen debe estar dentro del rango de IPs conocido para el edge gateway, esto es posible dado que la IP del gateway de cada ambiente es fija y fue informada por TYK para poder hacer la configuración correspondiente.
- La solicitud debe incluir el header "password" y su valor debe ser idéntico a un string que depende del ambiente. Este header es inyectado/sobreescribido por el gateway al momento de redirigir las solicitudes de los integradores a las APIs.

A continuación, en la figura 8 se muestra un resumen de lo descrito anteriormente:

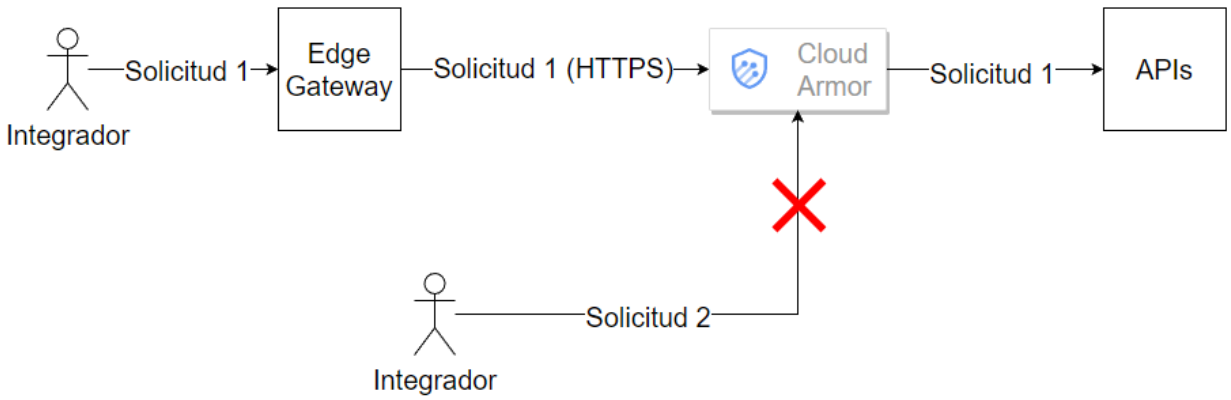

Figura 8: Conectividad TYK-GCP

Las reglas definidas en el Cloud Armor aseguran que a pesar de que las APIs están expuestas a internet, estas solo reciban solicitudes que hayan pasado por el edge gateway, evitando que se reciban solicitudes no autenticadas. Adicionalmente, dado que la solicitud entre el gateway y las APIs es realizada mediante HTTPS, esta está encriptada mediante el protocolo SSL/TLS. Esto permite otorgar un canal seguro entre dos computadoras o dispositivos que operan a través de Internet o de una red interna [72]. Y en particular en este caso, permite mantener una conexión segura entre el edge gateway y las APIs.

3.3.2. Acceso a las APIs

Enrolamiento y generación de API keys

Como se comentó en la sección 3.2.5, el proceso de enrolamiento y generación de API Keys es realizado por los corredores en el portal de corredores.

Este no se comunica directo con la herramienta de API Management, sino que utiliza un wrapper que abstrae las funcionalidades de la herramienta. Por lo que el portal de corredores solo interactúa con el wrapper mediante peticiones GET, POST o DELETE mediante HTTPS.

El wrapper además se comunica con un modelo de datos que reside en la BD MySQL, en este se encuentra toda la información necesaria para manejar las solicitudes provenientes del portal de corredores.

A continuación, en la figura 9 se muestra un diagrama de interacción del proceso de generación de una API KEY por un corredor, mostrando la interacción de las partes involucradas:

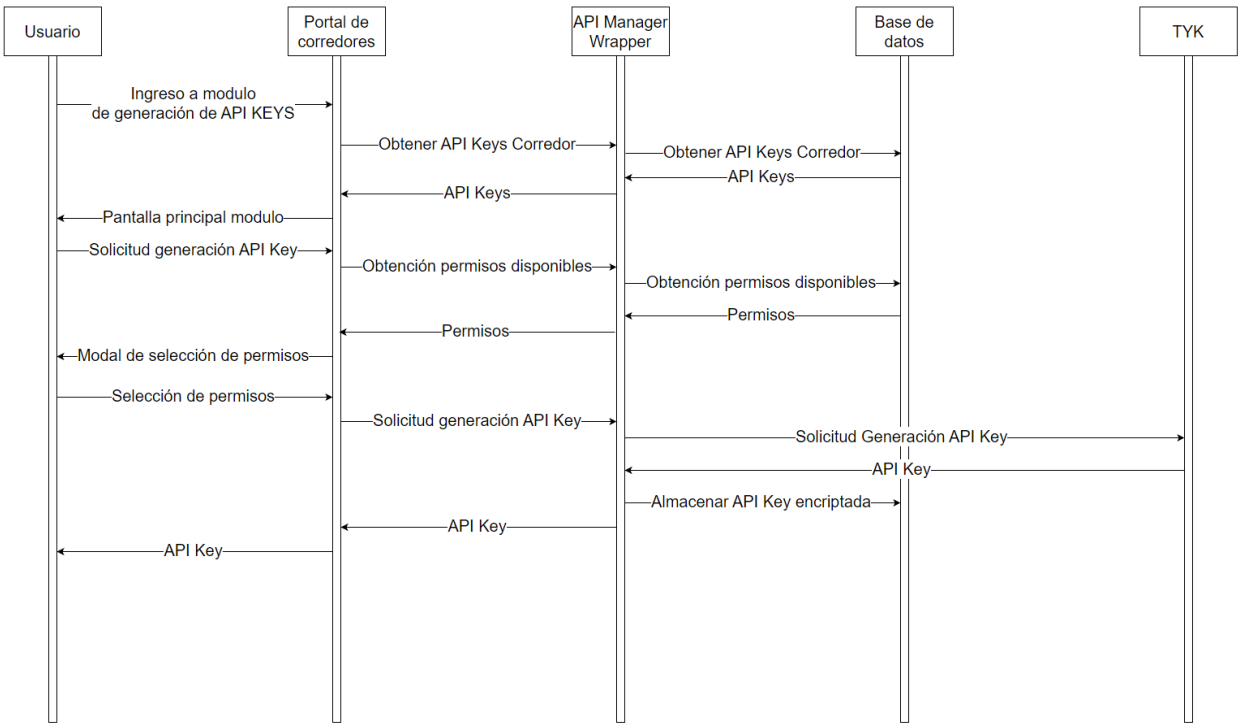

Figura 9: Pantalla principal generación de API Keys

El detalle de las componentes involucradas y cómo realizan estas funciones será descrito en las siguientes secciones.

API Manager Wrapper

Para abstraer la lógica detrás del manejo de API Keys se decidió implementar un wrapper, de tal manera que la pantalla del portal de corredores se integre con el wrapper y no directamente con el API Manager. Esto facilita la integración y minimiza el impacto de un posible cambio en la herramienta de API Manager.

El wrapper es capaz de:

- Generar y asignar permisos a API keys.
- Entregar las API keys activas del corredor.
- Revocar API Key.

Para realizar estas funciones el wrapper se comunica con APIs expuestas por Tyk. Estas APIs exponen funcionalidades normalmente realizadas en el dashboard descrito en la sección 3.3.1. Es mediante el uso de estas APIs de TYK que el wrapper puede realizar las funcionalidades requeridas.

Adicionalmente, el wrapper maneja un modelo de datos para almacenar la información relacionada a:

- API keys
- Corredores
- APIs
- Permisos
- Integradores
- Funcionalidades de APIs

A continuación, en la figura 10 se presenta un diagrama entidad relación de este:

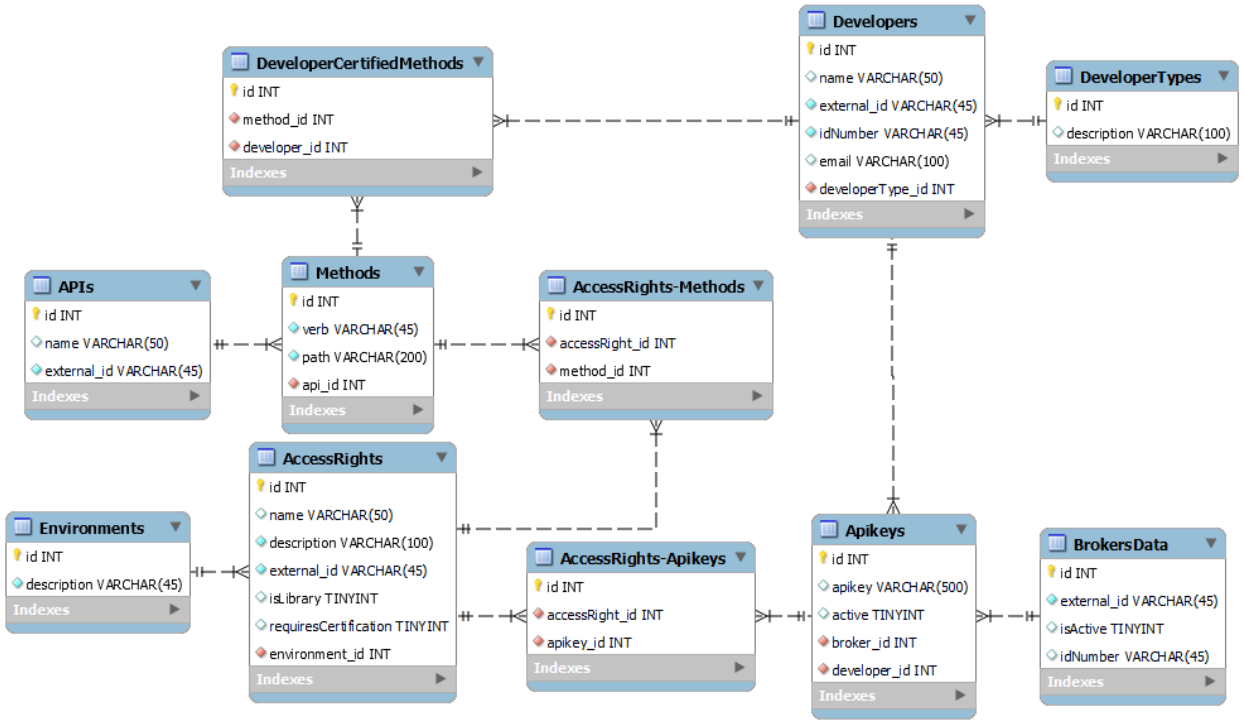

Figura 10: Diagrama ER del modelo de datos del API Manager Wrapper

En el contexto de este modelo de datos, se debe asumir que desarrollador es sinónimo del concepto de integrador utilizado hasta este momento en la memoria.

A continuación se detalla una breve descripción de las tablas principales:

- Developers: Contiene la información de los desarrolladores, indicando a qué tipo corresponden.
- DeveloperTypes: Contiene los tipos de desarrolladores mencionados en la sección 3.2.1. Actualmente solo contiene al tipo integrador-corredor.
- Apikeys: Contiene todas las API Keys encriptadas, indicando a qué corredor y desarrollador están asociadas. Pueden tener asignados uno o más permisos de la tabla AccessRigths.
- BrokersData: Contiene la información de los corredores.
- APIs: Contiene la información de las APIs disponibilizadas.
- Methods: Contiene la información de las funcionalidades ofrecidas por cada API, donde cada funcionalidad está identificada por la ruta, el verbo HTTP utilizado en la solicitud y a qué API pertenece.
- DeveloperCertifiedMethods: Contiene la información de qué métodos han sido certificados para cada desarrollador. Esto permite que en el futuro se pueda restringir qué permisos

pueden ser asignados a una API Key dependiendo de las certificaciones del desarrollador asociado a ella.

- AccessRights: Contiene la información de los permisos que pueden ser asignados a las API Keys generadas por los corredores, indicando en qué ambiente y funcionalidades otorga acceso. La columna “requiresCertification” indica si el permiso requiere que todas las funcionalidades asociadas estén certificadas para el desarrollador antes de generar una API Key.

- Environments: Contiene la información de los posibles ambientes en donde actúa un permiso. Actualmente sólo contiene el ambiente de producción, pero en el futuro podría ser añadido un ambiente de sandbox o testing.

Este modelo de datos permite al wrapper ser independiente del API Manager que está abstrayendo, simplificando el proceso de migración si en el futuro se desea cambiar la herramienta.

Como último comentario, las API Keys presentes en la tabla Apikeys están encriptadas con el algoritmo AES, según se comentó en la sección 3.2.6. Dado que el wrapper también posee la funcionalidad de obtener las API Keys de un corredor, también se encarga de desencriptarlas antes de devolverlas en la respuesta.

Portal de corredores

El proceso de enrolamiento se realizará mediante el portal de corredores mediante un módulo que debe ser añadido a este. Para realizar las funciones relacionadas a obtención/creación/revocación de API Keys el servidor realiza peticiones mediante HTTPS al API Manager Wrapper.

Cuando los corredores ingresan al módulo llegan a la pantalla principal. El wireframe de esta pantalla es presentado a continuación en la figura 11:

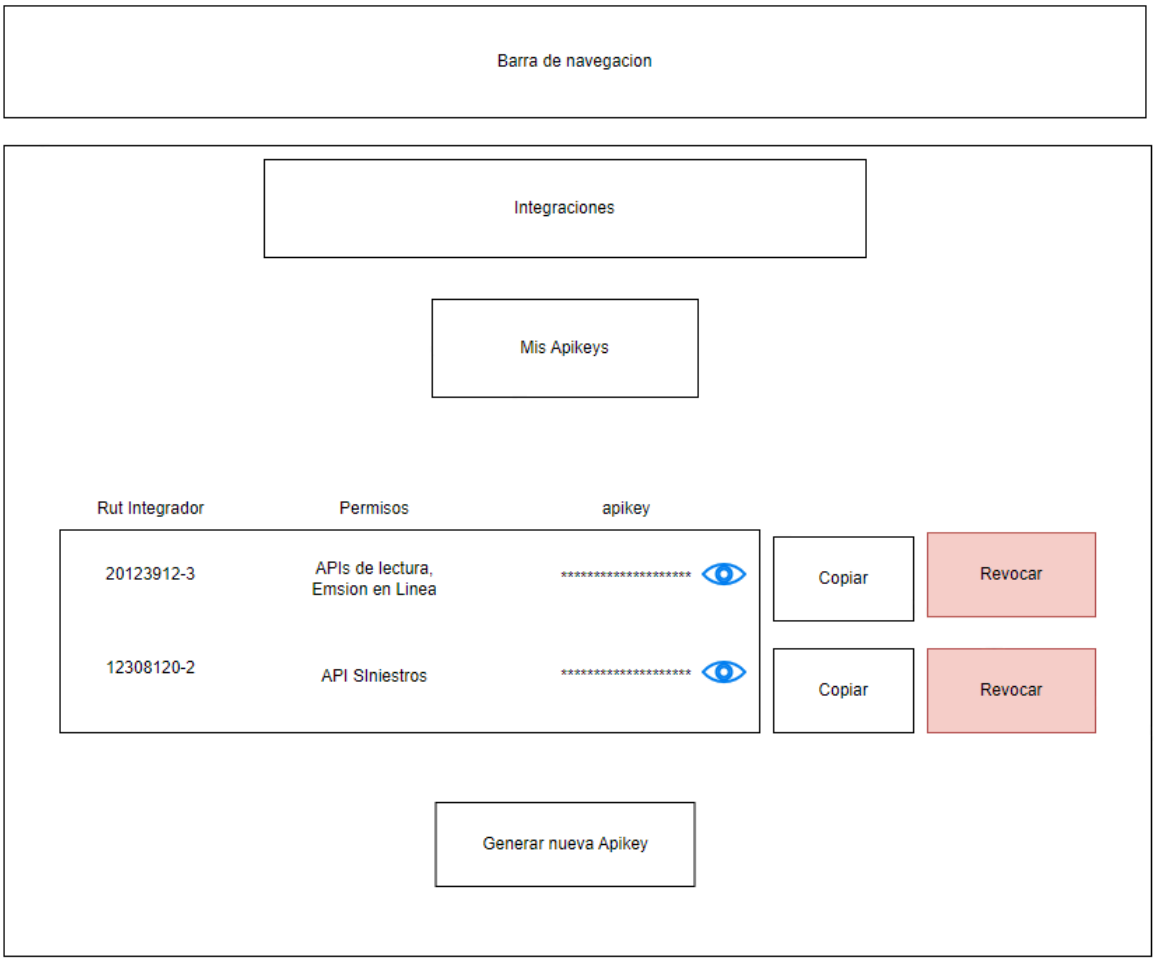

Figura 11: Pantalla principal generación de API Keys

Las secciones/partes principales de esta pantalla corresponden a:

- Listado de API Keys: Acá se presentan todas las API Keys activas del corredor, indicando que permisos tienen asignados y a que integrador está registrada. Adicionalmente se permite visualizar la API Key en cuestión al presionar el símbolo del ojo.
- Botón “Copiar”: Este botón es un atajo que les permite a los usuarios copiar la API Key al portapapeles del computador.
- Botón “Revocar”: Este botón permite a los usuarios revocar la API Key en cuestión, esto la deshabilita dentro del API Manager y la marca como inactiva dentro del modelo de datos.
- Botón “Generar nueva API KEY”: Este botón levanta una ventana modal que permite al usuario asignar permisos para generar una nueva API KEY.

Para generar una API Key el corredor debe presionar el botón “Generar nueva API Key”, posterior a esto se levantará una pantalla modal en donde se mostrarán los permisos actualmente disponibles a asignar.

En la figura 12 a continuación se presenta el diagrama wireframe de la pantalla modal de generación de API Keys:

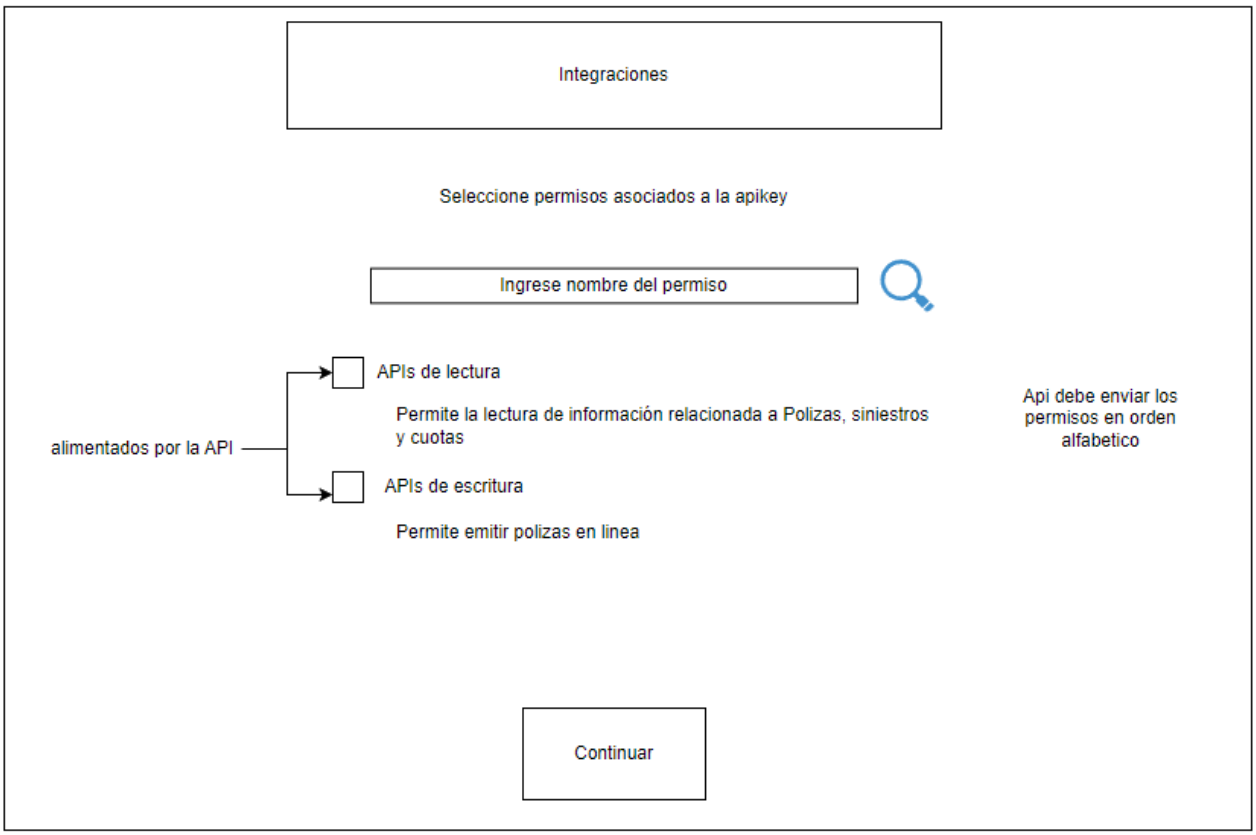

Figura 12: Ventana modal generación de API Keys

Esta muestra los permisos actualmente disponibles junto a una pequeña descripción. También permite utilizar un buscador para reducir la cantidad de permisos mostrados. El usuario luego debe seleccionar uno o más permisos que serán asignados a la API Key y presionar el botón “Continuar”.

Posterior a esto, se generará y se mostrará la API Key como se puede ver a continuación en la figura 13:

Figura 13: Ventana modal con API Key generada

La información sobre las API Keys activas del corredor y los permisos asignables a las API Keys son proveídos por el API Manager Wrapper. El portal de corredores obtiene esta información realizando peticiones GET al API Manager Wrapper mediante HTTPS. De la misma manera las funcionalidades de generación y revocación de API Keys también son proveídas por el API Manager Wrapper, y son ejecutadas cuando el portal de corredores realiza una petición POST o DELETE para creación y revocación respectivamente.

Dado que para el proyecto solo se consideran las integraciones del tipo corredor integrador, todas las API Keys generadas quedan configuradas con el corredor como integrador.

Flujo de autenticación

Una vez que un corredor ha sido enrolado y tiene una API Key valida a su disposición, este puede realizar solicitudes a las APIs habilitadas por los permisos asignados a la API Key.

Para realizar esta autenticación el corredor debe enviar el header “Authorization” con la API Key en la solicitud. Esta es luego validada por el gateway de Tyk para proceder a leer la metadata asociada a la API Key, la cual es adjuntada a la solicitud en los headers brokerIdNumber e integratorIdNumber. Esto causa que la API que recibe la solicitud posea la información de quienes son el corredor e integrador involucrados en la petición.

A continuación, en la figura 14 se muestra la visualización de este proceso:

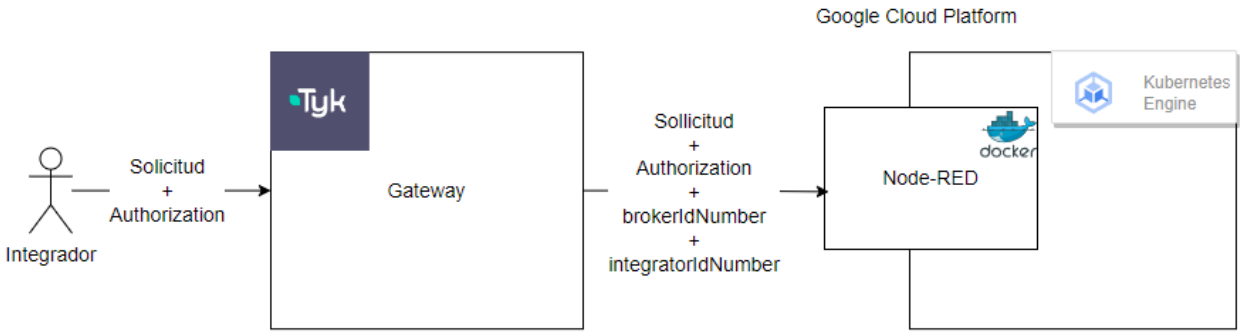

Figura 14: Flujo de autenticación

Este flujo permite a las APIs determinar qué información puede o no ser mostrada para cada solicitud, manteniendo la confidencialidad de la información mencionada en la sección 3.2.1.

3.3.3. Prácticas de desarrollo

Lineamientos de diseño

Para mantener una consistencia en las firmas de las APIs se definieron una serie de lineamientos de diseño, estos están basados en las prácticas recomendadas de APIs REST y la solución al problema de reutilización de usuarios descrito en la sección 3.1.3.

Verbos HTTP

Los verbos HTTP utilizados en cada solicitud deben tener una directa relación con la funcionalidad del endpoint, se definieron los casos de uso para los siguientes verbos:

- GET: Utilizado para operaciones de obtención de información de un recurso o una lista de recursos.
- POST: Utilizado para operaciones de inserción o creación de un recurso, la respuesta debe incluir el identificador generado para el nuevo recurso.
- PUT: Utilizado para reemplazar la totalidad de un recurso, esta es una operación aplicable solo a un elemento, por lo que el identificador del recurso debe ser enviado en la solicitud.
- PATCH: Utilizado para modificar ciertas propiedades de un recurso, esta es una operación aplicable solo a un elemento, por lo que el identificador del recurso debe ser enviado en la solicitud.
- DELETE: Utilizado para eliminar un recurso, esta es una operación aplicable solo a un elemento, por lo que el identificador del recurso debe ser enviado en la solicitud.

Si en el futuro aparece un caso de uso no descrito acá, este debe ser conversado con el arquitecto de la compañía para definir qué verbo debe ser utilizado.

Paginación

Las APIs que están sirviendo peticiones GET que retornen una lista de recursos deben incluir un sistema de paginamiento. Este estará basado en los parámetros limit y offset, los cuales son usados para indicar el límite de resultados devueltos en la respuesta y la posición desde donde deben ser devueltos los resultados respectivamente. Ambos parámetros deben ser devueltos en la salida para indicar cuales fueron el limit y offset enviados o que valores por defecto se utilizaron en caso de que no estuvieran presente en la solicitud originalmente.

Estos parámetros deben tener las siguientes configuraciones dentro de las APIs:

Limit

Valor máximo: 30 | Valor por defecto: 10

Offset

Valor mínimo: 0 | Valor por defecto: 0

Adicionalmente, se debe incluir el parámetro “total” dentro de la salida, este debe indicar la cantidad total de recursos en la BD.

Los parámetros de “limit”, “offset” y “total” deben ser añadidos al nivel más alto dentro del json de respuesta. Por ejemplo, para una API que devuelve una lista de pólizas, esta debe tener una salida del siguiente formato:

```
{
        “limit”: 10,
        “offset”: 0,
        “total”: 5000,
        “pólizas”: [ … ]
}
```

URIs

Las URIs deben estructurarse de tal manera que se represente una jerarquía de recursos dentro de la API.

Se fomenta el uso de identificadores y subrecursos para realizar operaciones sobre subelementos de un objeto. Por ejemplo, si se quiere disponibilizar un servicio para obtener los ítems asociados a una póliza, la URI recomendada corresponde a:

/polizas/{identificador_poliza}/items

Notar que los recursos y subrecursos son enunciados en plural, para indicar que se está trabajando sobre el conjunto de pólizas y sobre el conjunto de ítems. Si se quiere trabajar con un recurso en particular, se debe incluir el identificador posterior al nombre del conjunto de recursos.

Adicionalmente, se debe evitar el uso de verbos dentro de las URIs, la acción a realizar dentro del recurso debe estar indicado por el verbo HTTP que acompaña la solicitud.

Códigos HTTP

Para mantener la consistencia con el reporte de errores o el estado final de una solicitud se definen los siguientes casos de uso para los códigos HTTP:

- OK (200): Utilizado para indicar que una solución fue exitosa, principalmente utilizado para peticiones GET. Notar que si una petición GET que devuelve una lista de recursos retorna un arreglo vacío, este debe ser reportado como un 200, dado que no corresponde a un error.
- Created (201): Utilizado cuando se realiza una petición de creación de recurso con POST exitosa.
- No Content (204): Utilizado cuando se realiza una petición exitosa que no requiere mayor detalle al momento de responder al usuario. Generalmente usado en peticiones PATCH, PUT o DELETE en donde basta con comunicar si la operación falló o fue exitosa.
- Bad Request (400): Utilizado cuando se produce un error al validar el formato de los parámetros de entrada de la solicitud o cuando la información enviada no cumple con las reglas de negocio definidas para el servicio.
- Unauthorized (401): Utilizado para responder a solicitudes que no contienen una API Key.
- Forbidden (403): Utilizado para responder a solicitudes en las que se envía una API Key invalida.
- Not Found (404): Utilizado cuando no se encuentra la información relacionada a un recurso, reservado para cuando se intenta realizar operaciones sobre un recurso identificado mediante el valor enviado en la URI y no existe información asociada a ese identificador.
- Internal Server Error (500): Utilizado para comunicar que ocurrió un error en el servidor al momento de procesar la solicitud. Generalmente usado cuando falla la conectividad con algún servicio o la BD utilizado por la API.

Nombramiento de parámetros

Para estandarizar el nombramiento de parámetros de las APIs se definió Camel Case como la norma a utilizar.

Usuarios de BD

Para evitar la reutilización de usuarios de BD se definió que cada API debe tener un usuario con el siguiente nombre:

{Nombre API}API

Por ejemplo, si el nombre de la API es Claim, entonces el usuario de BD debe ser ClaimAPI.

Adicionalmente, los usuarios de cada ambiente deben tener contraseñas distintas, de modo que se reduzca el impacto si se llegara a filtrar la contraseña del usuario de algún ambiente.

3.3.4. Migración de datos

Como se comentó en la sección 3.2.2, la estrategia para mitigar el impacto en la performance del Core involucra el levantamiento de una segunda BD y utilizar una herramienta de ETL para mantenerla actualizada. La herramienta elegida para llevar a cabo esta tarea corresponde a Talend, en donde las razones de esta decisión están detalladas en la sección 3.2.3.

Para entender el funcionamiento del sistema es importante conocer la funcionalidad de Change Data Capture (CDC) de SQL Server, dado que la migración está basada en esta funcionalidad. La funcionalidad de CDC es descrita en la página de Microsoft como una funcionalidad que hace uso del agente del motor para guardar un registro para cada operación de inserción, modificación o borrado que se realice sobre una tabla [73].

Los datos más importantes almacenados para cada operación son los siguientes:

- start_lsn y end_lsn: Corresponde al log sequence number (lsn) de cada cambios capturado por el CDC y actúa como el identificador único de cada cambio.
- seqval: Valor usado para identificar el orden de múltiples registros que representan una misma operación. Un ejemplo de esto son las modificaciones de información, dado que se guarda el registro que fue modificado, y el nuevo registro tras la modificación.
- operation: Valor que especifica la operación realizada (inserción, borrado o modificación).

Estos valores son utilizados por el sistema de migración para coordinar la migración de datos y mantener la consistencia entre ambas BD.

A continuación, se detallan los principales actores del sistema:

- BD de Origen: Origen de los datos, en este caso corresponde a una instancia de VM con un motor de BD de SQL Server.

- BD de destino: Destino de los datos, en este caso corresponde a una instancia de Cloud SQL de GCP con un motor de BD MySQL.
- BD de Control: BD usada para mantener la consistencia entre las BD de origen y destino. Contiene información utilizada para coordinar los procesos de migración de datos.
- Talend: Herramienta de ETL que realiza la migración de los datos desde la BD de origen a la BD de destino.

En particular la BD de control contiene 2 tipos de tablas:

- Tablas de control: Contienen información para coordinar los procesos de migración y el manejo de errores.
- Tablas intermedias: Se genera una por cada tabla actualmente migrada. Estas tienen nombre cdc_{Nombre de BD de la tabla}_{esquema}_{nombre tabla} y contiene una copia del CDC de cada tabla migrada. En donde el CDC original de cada tabla reside en la BD del Core.

La tabla de control más importante es la control_migracion, ésta contiene una fila por cada tabla actualmente migrada e indica cuál fue el último LSN traspasado desde el CDC de la BD del Core a su respectiva tabla intermedia. Adicionalmente, almacena el LSN del último cambio que ha sido aplicado desde la tabla intermedia en la BD de destino. Esta tiene las siguientes columnas:

- TablaNombre: Nombre de la tabla que está siendo migrada
- UltimoLSNProductor: LSN del último registro que ha sido traspasado desde el CDC de la BD del Core a la tabla intermedia de la tabla involucrada en el proceso de migración.
- UltimoLSNConsumidorMySQL: LSN del último registro en la tabla intermedia que ha sido aplicado dentro de la BD de destino.

Migración de datos histórica

La migración histórica de datos se lleva a cabo mediante un proceso denominado job init, el cual se genera de forma independiente para cada tabla. Este proceso se encarga de copiar tanto la tabla en sí como la tabla CDC desde la base de datos del Core hasta la base de datos de destino e intermedia, respectivamente.

Adicionalmente, ingresa la información de la tabla que se está migrando dentro de la tabla control_migración. Las columnas UltimoLSNProductor y UltimoLSNConsumidorMySQL quedan con el valor del LSN del último registro ingresado a la tabla CDC que se está copiando. Esto es necesario para que el proceso de migración de datos en vivo sepa desde qué punto en adelante tiene que aplicar los cambios.

Para evitar que la información sea modificada mientras se realiza la migración se hace uso de la funcionalidad de snapshot de SQL Server. Esta funcionalidad permite generar una vista estática de solo lectura de la data de una BD al momento de la creación del snapshot [74]. Como las tablas de CDC también están incluidas dentro de los snapshots de SQL Server, se tiene toda la información necesaria para realizar la migración.

Este diseño tiene la ventaja de que si en algún momento se produce alguna inconsistencia en la información entre el origen y destino, basta con generar un nuevo snapshot, ejecutar el job init, y reanudar la ejecución del job de migración de datos en vivo para corregir las inconsistencias.

A continuación, en la figura 15 se muestra un diagrama resumiendo el funcionamiento del job init:

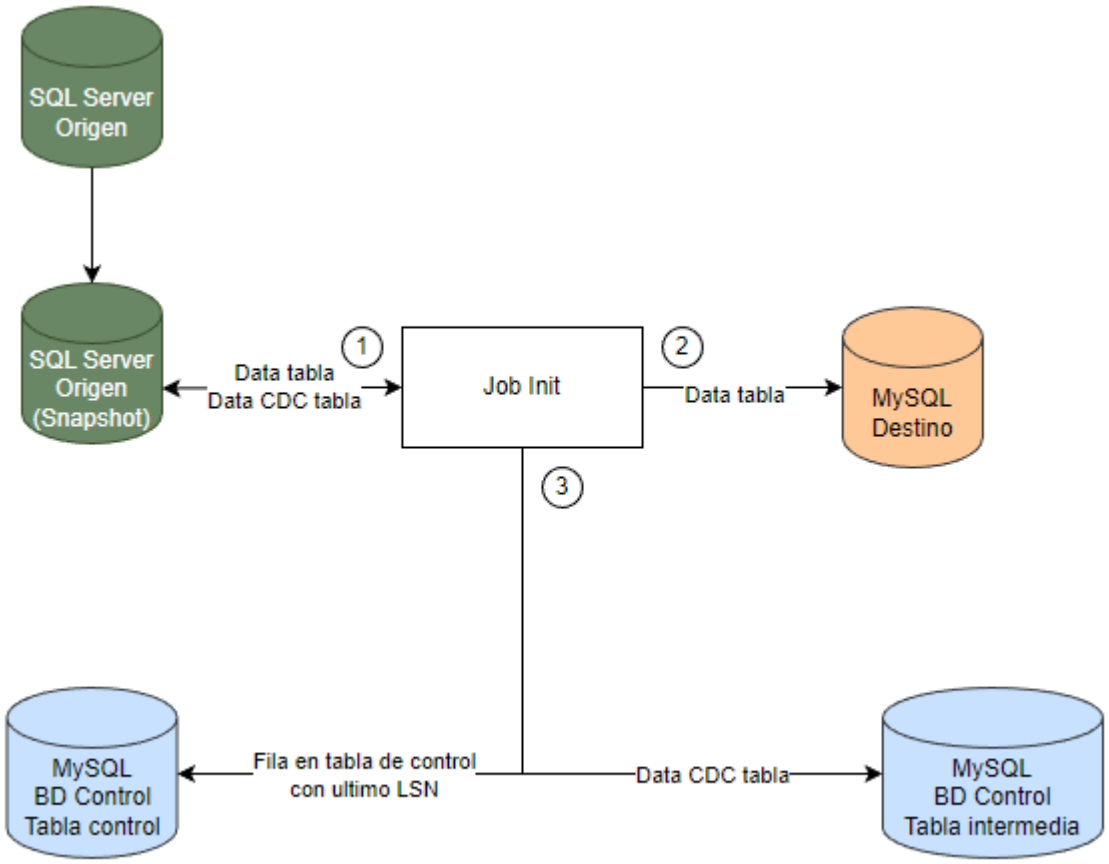

Figura 15: Diagrama de comunicación job init

Migración de datos en vivo

El sistema de migración de datos en vivo se encarga de aplicar los cambios que ocurran en los datos de la BD del Core de la compañía dentro de la BD MySQL, fue diseñado siguiendo el patrón de diseño de productor-consumidor. Este patrón está enfocado en mejorar el intercambio de datos entre múltiples ciclos que se ejecutan a diferentes velocidades [75].

El sistema de migración de datos en vivo está compuesto por los siguientes 2 procesos:

- Job productor: Mostrado en la figura 16, es un proceso encargado de poblar la tabla intermedia en la BD de control con los registros a migrar. Utiliza el UltimoLSNProductor guardado en la tabla de control y lo compara con el último LSN del CDC de la BD del Core para revisar si hay cambios que no han sido traspasados a la tabla intermedia. Si encuentra cambios pendientes, los mueve a la BD intermedia y actualiza el valor de UltimoLSNProductor.

- Job consumidor: Mostrado en la figura 17, es un proceso encargado de aplicar los cambios pendientes que viven en la tabla intermedia de la BD de control. Ocupa el valor en UltimoLSNConsumidorMySQL y lo compara con los registros en la tabla intermedia. Si existen registros que tengan un LSN mayor que UltimoLSNConsumidorMySQL, estos son aplicados en BD de destino.

A continuación, en la figura 16 y 17 se muestran los gráficos representando el funcionamiento del job productor y consumidor respectivamente:

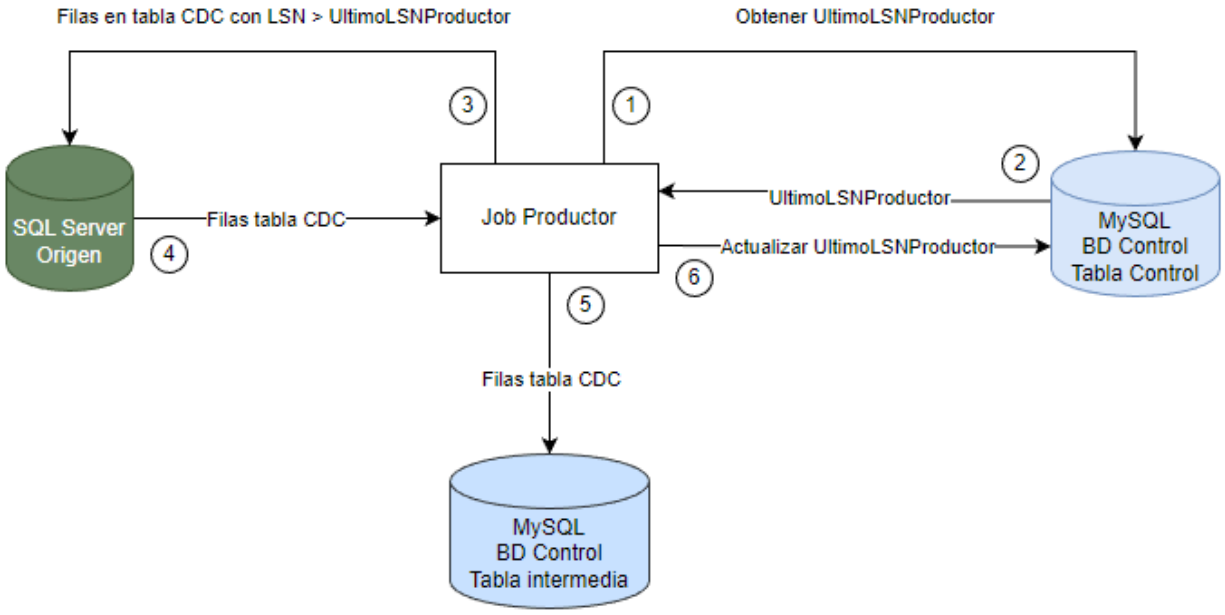

Figura 16: Diagrama de comunicación job productor

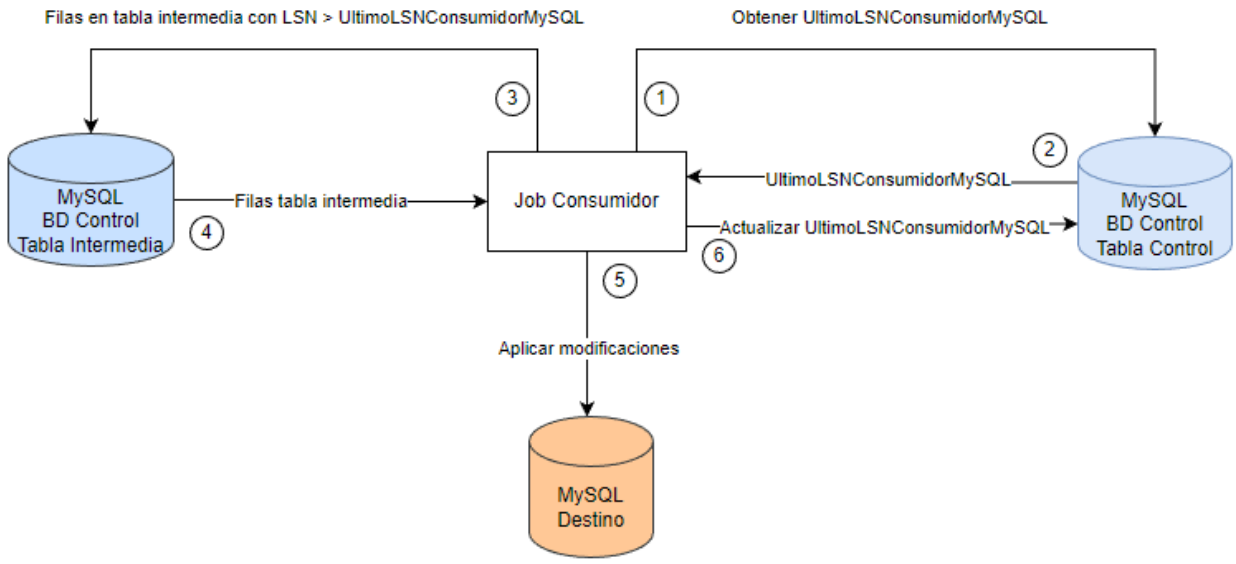

Figura 17: Diagrama de comunicación job consumidor

Se decidió utilizar el patrón de productor-consumidor en el diseño de este sistema de migración para no generar una dependencia entre la migración de datos del CDC en la BD de origen, y la aplicación de estos cambios en la BD de destino. Este enfoque posibilita una solución más modular, evitando que posibles errores en uno de los jobs afecten al otro.

Los jobs productor y consumidor se deben ejecutar frecuentemente para minimizar el tiempo de desfase entre la BD del Core y la BD MySQL. Debido a las restricciones de Talend, ambas BD tienen un desfase de alrededor de 5 min.

Manejo de errores

Todos los jobs descritos anteriormente trabajan con transacciones en las BD involucradas, si todo el proceso se ejecuta correctamente se realiza un commit al final del proceso para cada transacción, en caso contrario se realiza un rollback en todas ellas. Dado que los commits deben realizarse de manera secuencial para cada BD involucrada, existe la posibilidad de que ocurra un error posterior a algún commit, se aceptó el riesgo que esto implicaba dado que la probabilidad que esto ocurra es baja.

Adicionalmente, si ocurre un error se envía un correo de notificación a una lista de distribución, para que el equipo encargado de la mantención esté enterado del problema y pueda tomar las acciones adecuadas.

Estos correos son controlados mediante la configuración presente en las tablas control_correo y deltas_tiempos de la BD de control.

Estas tablas contienen información sobre:

- Qué tablas tienen habilitado el envío de correo en caso de error.
- Fecha y hora del último correo enviado para cada tabla.
- Veces que un correo ha sido enviado.
- Tiempo mínimo que tiene que pasar antes de enviar otro correo dependiendo de la cantidad de correos enviados, esto para no enviar una cantidad excesiva de correos por cada vez que un proceso de migración falla.

Los procesos de migración histórica y en vivo utilizan la información presente en estas tablas para decidir si deben o no enviar un correo en caso de error.

3.3.5. APIs de consulta

Las APIs desarrolladas permiten obtener información sobre siniestros, pólizas y cuotas. Para esto las APIs realizan consultas a la BD MySQL que se mantiene actualizada gracias al sistema de migración de datos en vivo.

Su proceso para manejar una solicitud puede ser dividido en 3 etapas:

- Validación: En este paso se verifica que los parámetros enviados por el usuario corresponden a los definidos en la firma de entrada de la API. Se asegura de que vengan los parámetros obligatorios, que no se hayan enviado parámetros no definidos en la firma y que los formatos de los parámetros son correctos. Adicionalmente, en esta fase se lleva a cabo cualquier validación necesaria asociada a reglas de negocio específicas relacionadas con los datos.
- Obtención de información: En esta etapa, se realizan las consultas a la BD para obtener la información del recurso solicitado. La información recuperada está filtrada para sólo incluir contenido que esté asociado al corredor identificado en la solicitud, de tal manera que se evite la filtración de información de otros corredores.
- Formateo de salida: Para finalizar se procesa la información recibida de la BD y se organiza para cumplir el esquema de la firma de salida de la API.

Fuera de ciertas particularidades del modelo de datos asociados a siniestros, pólizas y cuotas, no existe una diferencia significativa entre el funcionamiento general de cada una de estas APIs.

4. Implementación

El presente capítulo detalla el proceso de implementación de la solución, en donde se comentan las principales tareas a realizar, desafíos al momento de implementar la solución y cómo estos fueron abordados.

A excepción del portal de corredores, la implementación fue realizada por un equipo de 2 desarrolladores liderados por el memorista. Adicionalmente, se contó con la supervisión del Jefe de Sistemas de la compañía. Aparte de esto, se requirió el apoyo del equipo de infraestructura para configurar los servicios que residen en GCP y del equipo de QA para certificar los desarrollos.

4.1. Tyk

Para implementar el método de autenticación y enrolamiento se tuvieron que definir los siguientes objetos en el dashboard del API Manager:

- API Definitions: Objeto Json que se encarga de encapsular las configuraciones esenciales de cada API, así como las acciones a realizar a las solicitudes entrantes [76].
- Policies: Corresponde a un template que permite definir opciones de seguridad que son aplicadas a todas las API Keys que hayan sido o sean generadas para esa policy. Esta configuración es retroactiva, por lo que si se modifica una Policy, las API Keys que ya se habían generado para esta también son afectadas por la modificación. En estas se definen cosas como lista de accesos a APIs, lista de accesos a métodos y rutas, cantidad máxima de solicitudes en un cierto periodo de tiempo, entre otras [77].

Como se explicó en la sección 3.3.2, las API keys generadas contienen metadata que debe ser añadida a la solicitud para identificar al corredor e integrador involucrados. Para lograr esto se incluyó dentro de las API Definitions la instrucción de extraer y añadir la metadata de las API Keys al momento de pasar por el gateway.

Adicionalmente, para lograr la conectividad entre el API Manager y las APIs en GCP descrita en la sección 3.3.1, se añadió la instrucción de agregar el header “password” con la clave correspondiente a cada ambiente. Con esto se logra que las solicitudes que pasen por el gateway cumplan con las excepciones configuradas en el Cloud Armor y puedan acceder a las APIs en GCP.

Con respecto a las Policies, se definió una de nombre “APIs de lectura”, en donde se incluye el acceso a las 3 APIs del proyecto y se define un cantidad máxima de 1000 solicitudes cada 60 segundos para evitar que los sistemas sean sobrecargados por un solo integrador.

El principal desafío de esta parte de la implementación es la curva de aprendizaje necesaria para familiarizarse con TYK. Para aprender a usar la herramienta se leyó la documentación disponible y se realizaron consultas al equipo de soporte de TYK.

4.2. Portal de Corredores

La implementación del módulo de generación de API Keys del portal de corredores fue realizada por una empresa de desarrollo de software llamada Pepiln (https://pepiln.com). Esta decisión fue hecha para asegurar que el módulo mantenga el estilo del resto del portal, dado que Pepiln ya había realizado desarrollos en el portal de corredores anteriormente.

Se generaron reuniones para levantar los requisitos del módulo y mostrar los diagramas wireframe incluidos en la sección 3.3.2. Posterior a eso se mantuvieron reuniones semanales con la líder técnica de Pepiln para resolver dudas, comunicar avances y entregar feedback.

El principal desafío fue la comunicación con Pepiln, se tuvo especial cuidado en transmitir correctamente el objetivo del proyecto, el tipo de usuario que usaría el módulo y que funcionalidades debian ser incluidas. Sumado a esto, dado que el portal de corredores debe integrarse con el API manager Wrapper, se les debía comunicar cómo tenían que interactuar con este para poder obtener la información y manejar las API Keys.

La comunicación debía ser constante porque el API Manager Wrapper se desarrolló de manera simultánea. Es por esto que cualquier error o modificación que ocurriera en el wrapper debía ser informado a Pepiln lo antes posible, para no interferir con el desarrollo en el portal de corredores.

4.3. API Manager Wrapper

La implementación del API Manager Wrapper fue realizada por uno de los desarrolladores del equipo, mientras que el despliegue de este fue realizado por el equipo de infraestructura.

Como se menciona en la sección 3.3.2 el API Manager Wrapper requiere de un modelo de datos que es usado para guardar la información relacionada a la generación de API Keys. Este modelo fue creado dentro de la instancia MySQL.

El wrapper en sí mismo fue implementado en Node-RED. Este debía comunicarse con TYK mediante las APIs de administración disponibilizadas por el dashboard para poder realizar operaciones relacionadas a las API Keys. Para esto el desarrollador se basó en la documentación de la página Web de TYK.

Para mantener la consistencia entre las API Keys generadas/revocadas en el API Manager y el modelo de datos se utilizó una combinación de transacciones, rollbacks y flujos de error que deshacían las operaciones realizadas en el API Manager.

El principal desafío de este desarrollo fue la integración con los servicios de TYK. En particular, ocurrió que la documentación de estos servicios no incluye especificaciones sobre cómo generar API Keys con metadata [78]. Para resolver esto se comunicó con el equipo de soporte de TYK para discutir un workaround, el cual incluía crear una API Key llamando a un endpoint en particular y posteriormente modificar la API Key mediante el uso de otro endpoint para añadir la metadata.

4.4. Servicios de GCP

Los servicios de GCP necesarios para el proyecto fueron configurados principalmente por el equipo de infraestructura. Este mantenía una comunicación constante con el equipo de desarrollo para acordar las configuraciones y definir los requerimientos.

Las servicios configurados fueron los siguientes:

- Cloud Armor: Utilizado para proteger a las APIs en el cluster de Kubernetes de ataques de denegación de servicio y solicitudes no provenientes desde el gateway del API Manager. Como se detalla en la sección 3.3.1, su principal función es impedir cualquier tráfico hacia las APIs que no haya pasado por el gateway de TYK.
- Kubernetes: Utilizado para hospedar a las APIs del proyecto. Como se indicó en la sección 3.3.1, se solicitó la creación de un nuevo cluster, separado del resto de los servicios de la compañía para reducir el impacto de una vulnerabilidad de seguridad.
- Cloud SQL: Utilizado por las APIs para obtener la información y minimizar el impacto a la performance del Core. El motor de BD usado es MySQL. Como se indicó en la sección 3.2.2, esta corresponde a la BD que es llamada por las APIs desarrolladas. De tal manera que se reduzca el impacto que el proyecto pueda tener en la performance del Core de la compañía.
- Máquina virtual para remote engine de Talend: Máquina virtual que actúa como runtime de los procesos de migración de datos. Como se mencionó en la sección 3.3.1, esta permite la migración histórica y en vivo de los datos desde la BD del Core hasta el Cloud SQL.
- VPC Peering: Servicio de GCP que permite al remote engine conectarse mediante IP Privada a la BD del Core y a Cloud SQL. Como se mencionó en la sección 3.3.1, su configuración es necesaria porque el Remote Engine, el Core y el Cloud SQL residen en VPCs distintas.

4.5. Talend

La implementación de los procesos de migración de datos fueron realizadas por ambos desarrolladores del equipo. Como se mencionó en la sección 4.4, la configuración de la máquina virtual del Remote Engine y la conectividad con la BD del Core y el Cloud SQL fueron manejadas por el equipo de infraestructura.

Para desarrollar los procesos de migración se utilizó el programa de Talend Studio incluido con la licencia de la Talend. Este programa permite diseñar los procesos de migración de datos (o Jobs como son llamados en Talend) mediante el uso de una interfaz drag-and-drop [79]. Para el control de versiones, Talend Studio tiene la posibilidad de sincronizarse con un repositorio GIT, por lo que esto fue utilizado durante el desarrollo. Un repositorio GIT corresponde a un almacenamiento virtual de un proyecto. Este permite guardar versiones del código que pueden ser accedidas según se necesite [80]. Una vez que el desarrollo de un Job está listo, este puede ser publicado a Talend

Cloud, en donde luego puede pedirse la ejecución manual en el Remote Engine o ser configurado para ejecutarse recurrentemente cada cierto tiempo.

Los procesos de migración histórica fueron nombrados como Jobs Init. Mientras que los Jobs productor y consumidor fueron agrupados en un solo Job llamado Job CDC.

Se realizó la migración de solo las tablas que iban a ser utilizadas por las APIs del proyecto, debido a esto se desarrollaron 25 Jobs Init y 25 Jobs CDC para migrar las 25 tablas requeridas.

El principal desafío de este desarrollo fue la curva de aprendizaje para utilizar Talend Studio. Existía experiencia previa utilizando Talend Studio debido al tiempo en Talend Academy como se indicó en la sección 3.2.3, pero los desarrollos realizados en ese momento eran más simples en comparación con la implementación de lo que se necesitaba para el proyecto. Esto causó que se necesitará más tiempo del inicialmente pensado para aprender a utilizar las funcionalidades de la aplicación.

4.6. APIs de consulta

La implementación de las APIs fueron realizadas por ambos desarrolladores del equipo, mientras que el despliegue de los servicios fue realizado por el equipo de infraestructura.

Se comenzó por definir la firma de entrada y salida de las APIs, para esto se mantuvieron reuniones con gente del área del negocio y se les preguntó qué información creían que sería más útil para los corredores con respecto a cuotas, pólizas y siniestros.

A partir de esta información, se construyeron las firmas de los servicios siguiendo los lineamientos de diseño definidos en la sección 3.3.3.

El desarrollo de las APIs fue realizado en Node-RED, en donde se implementaron los sistemas de paginamiento descritos en los lineamientos de diseño y se utilizaron usuarios de BD creados específicamente para cada API.

La certificación del servicio fue realizada por el equipo de QA de la compañía. Estos procesos fueron relativamente rápidos para las APIs de Póliza y Cuotas. Sin embargo, la API de siniestros pasó por al menos 5 ciclos antes de poder quedar como aprobada. Esto se debió principalmente a que el modelo de datos de siniestro es más complejo que los usados en las otras APIs. En particular, se tenía el problema de que el modelo no es intuitivo, por lo que se requería de información adicional para poder interpretar los datos contenidos en las tablas. Esto implicó que se tuvieron que realizar múltiples consultas y reuniones con otros desarrolladores y líderes de equipos TI para poder entender cómo obtener e interpretar la información definida en la firma.

5. Validación

El presente capítulo detalla la validación de la solución, en donde se busca asegurar de que los objetivos definidos en la sección 1.3 fueron realmente abordados por la solución diseñada.

5.1. Encuesta de validación

Para validar la solución se generó una encuesta de 5 preguntas, una por cada objetivo específico propuesto en la sección 1.3. Estas preguntas buscan recopilar opiniones sobre hasta qué punto el proyecto cumple con cada objetivo asociado. Se solicitó a los participantes que respondieran asignando un número del 1 al 5 para expresar en qué medida se logró dicho objetivo. Este rango está definido por los extremos 1 y 5, donde el 1 refleja una percepción negativa sobre el logro del objetivo, mientras que el 5 denota una perspectiva positiva al respecto.

La encuesta fue realizada con la herramienta de Google Forms. Esta es una herramienta de Google que permite crear formularios y visualizar los resultados de esta [81].

Esta encuesta fue realizada por los dos desarrolladores del proyecto, el jefe de sistemas, el gerente de finanzas y tecnología, la subgerenta de tecnología y el arquitecto de la compañía. Además, se incluyó la opinión de un desarrollador que no estuvo directamente involucrado en el proyecto, pero que cuenta con conocimientos sobre su funcionamiento y objetivo.

Las preguntas realizadas en la encuesta corresponden a:

- Pregunta 1: ¿Qué tanto cree que el proyecto de disponibilización de APIs acelere el proceso de integración para corredores en el futuro?, en donde 1 representa poco y 5 representa mucho.
- Pregunta 2: ¿Cómo calificaría la facilidad del proceso de enrolamiento implementado en el proyecto desde el punto de vista de un corredor?, en donde 1 representa muy difícil y 5 representa muy fácil.
- Pregunta 3: ¿Cuánta confianza tiene en que el sistema de autenticación implementado en el proyecto es capaz proveer seguridad a las APIs expuestas y también identificar al corredor e integrador involucrados en la solicitud?, en donde 1 representa poca confianza y 5 representa mucha confianza.
- Pregunta 4: ¿Qué tanto cree que las prácticas de desarrollo definidas en el proyecto impulsan la generación de código que refleje una buena imagen de la compañía a los integradores?, en donde 1 representa poco y 5 representa mucho.
- Pregunta 5: ¿Cuánta confianza tiene en que la solución del proyecto no generará problemas de rendimiento en el Core?, en donde 1 significa poca confianza y 5 significa mucha confianza.

A continuación, en las figuras 18, 19, 20, 21 y 22 se muestra la distribución de respuestas a cada una de estas preguntas.

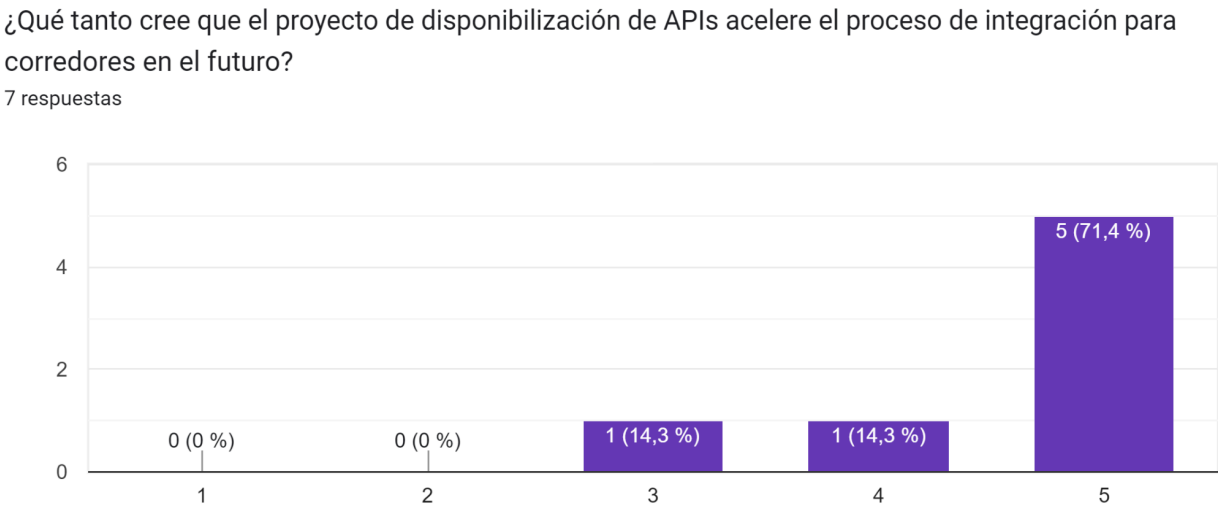

Figura 18: Respuestas pregunta 1

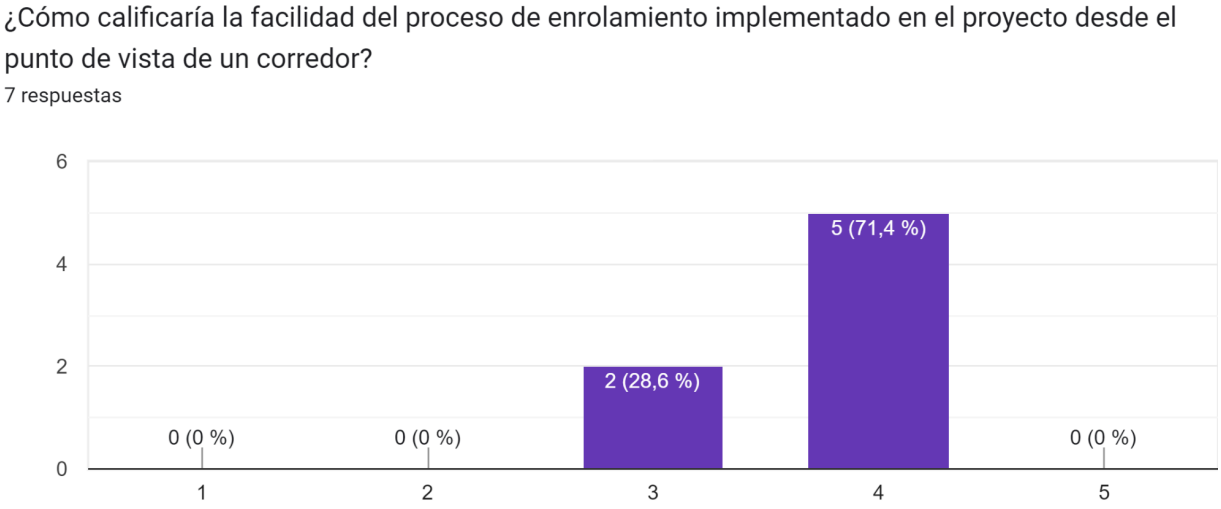

Figura 19: Respuestas pregunta 2

¿Cuánta confianza tiene en que el sistema de autenticación implementado en el proyecto es capaz proveer seguridad a las APIs expuestas y también id...orredor e integrador involucrados en la solicitud?
7 respuestas

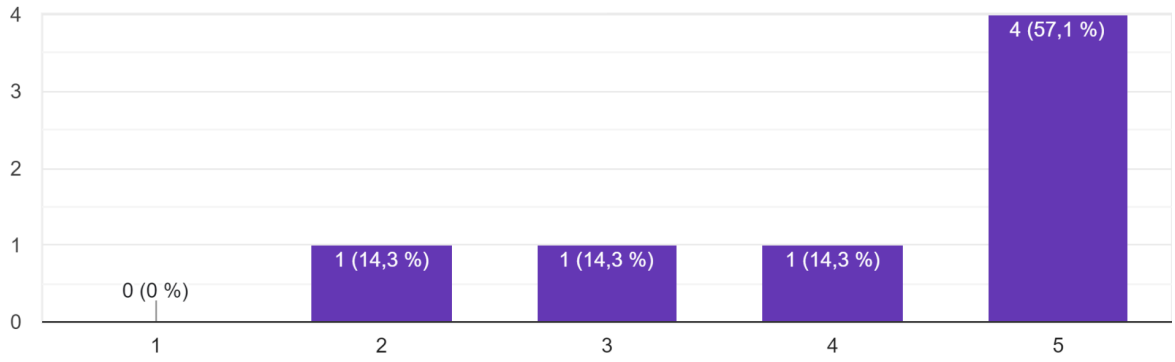

Figura 20: Respuestas pregunta 3

¿Qué tanto cree que las prácticas de desarrollo definidas en el proyecto impulsan la generación de código que refleje una buena imagen de la compañía a los integradores?
7 respuestas

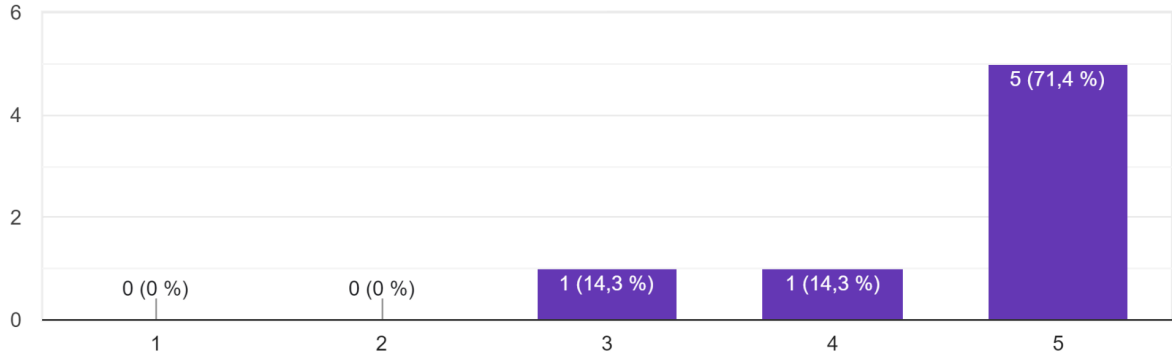

Figura 21: Respuestas pregunta 4

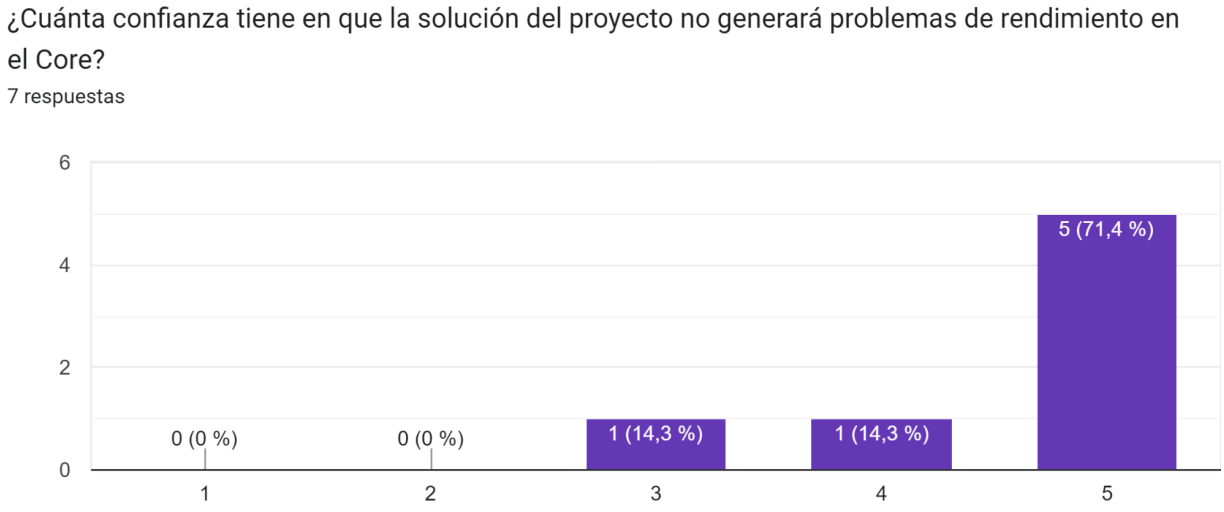

Figura 22: Respuestas pregunta 5

Se ha observado que todos los encuestados proporcionaron respuestas con calificaciones de al menos 3 en las preguntas 1, 2, 4 y 5. Además, se nota que en todas las preguntas, al menos el 70% de las respuestas recibieron calificaciones de 4 o 5. Específicamente, en las preguntas 1, 4 y 5, el 71,4% de los encuestados asignaron la calificación máxima, indicando que, según su opinión, la solución satisface completamente los objetivos asociados con estas preguntas.

En la pregunta 2, se observa que ningún participante respondió con un 5; la mayoría de las respuestas son un 4. Esto indica que, según la opinión de los encuestados, el sistema de enrolamiento presenta aspectos que dificultan que las integraciones sean lo más sencillas posible desde el punto de vista de los corredores. Esto podría deberse a que el sistema de enrolamiento requiere que los corredores tengan acceso al portal de corredores de FID, dado que esto agrega un requisito previo que no está directamente relacionado con la capacidad de un corredor para consumir los servicios. Adicionalmente, el sistema de enrolamiento no incluye ningún material de apoyo para guiar a los corredores por el proceso de integración, ni ofrece orientación sobre cómo proseguir una vez han generado una API Key. Esto es de especial relevancia si se considera que las insurtech tuvieron un gran crecimiento recién a partir del año 2016 [82], por lo que muchos corredores aún no están familiarizados con la mezcla entre aseguradoras y tecnología.

En la pregunta 3, se observa una calificación promedio menor que en las preguntas 1, 2 y 5, donde incluso un participante respondió con un 2. Esto indica que, aunque la mayoría de los participantes considera que el sistema de autenticación es confiable y seguro, algunos encuestados no comparten esa opinión. Las respuestas que reportan una baja confianza podrían deberse a que el sistema de autenticación solo está basado en la API Key, por lo que si un tercero llegara a conseguirla, tendría acceso a todos los servicios habilitados por dicha llave. Si bien los corredores

pueden revocar las llaves generadas mediante el portal de corredores, no hay una manera de que identifiquen si alguien más está utilizando alguna de sus API Keys generadas. Una posible solución para este problema sería asignar IPs a las API Keys generadas, de manera que cada API Key solo permita autenticar solicitudes provenientes de esas IPs. Esto permitiría que, si la API Key es comprometida, el atacante no tenga acceso a la información.

A partir de lo anterior, se puede concluir que la solución aborda de manera razonable los objetivos del proyecto, según la perspectiva de los encuestados, ya que la mayoría de las respuestas se sitúan en el rango de 4 y 5. No obstante, se identifican áreas de mejora, especialmente en el sistema de autenticación y enrolamiento, como se evidencia en las respuestas recopiladas en las preguntas 2 y 3 respectivamente.

5.2. Monitoreo Core de la compañía

Para validar que la solución no comprometa la continuidad operacional de la compañía, se realizó un monitoreo de los recursos utilizados por la máquina virtual donde reside el Core. Esto se realizó con el objetivo de validar que la ejecución del sistema de migración de datos no consuma una cantidad de recursos de la máquina que pueda interferir con otros procesos. En particular, el monitoreo se enfocó únicamente en el impacto del sistema de migración de datos en vivo. Esto se debe a que el sistema de migración de datos histórico solo se ejecuta en la primera migración de cada tabla y puede ser realizado fuera del horario de operación, por lo que el riesgo de que ponga en riesgo la continuidad operacional es significativamente menor.

Se recolectaron métricas del porcentaje de CPU utilizada, en donde se tomó la decisión de no monitorear la memoria utilizada. Esta decisión está basada en la forma en que SQL Server maneja la memoria. El administrador de memoria de SQL Server está diseñado para seguir adquiriendo memoria según lo requiera la carga actual. Esta adquisición se detiene si la memoria utilizada alcanza la configuración objetivo de memoria máxima del servidor o el sistema operativo indica que no existe memoria en exceso disponible. Adicionalmente, la memoria adquirida es solo liberada si se tiene más memoria que el mínimo configurado y el sistema operativo indica que hay escasez de memoria [83]. Es por esto que la cantidad de memoria reportada por el monitoreo corresponde a la máxima cantidad de memoria que ha sido utilizada por el SQL Server desde el inicio de su ejecución. Esto hace altamente improbable que esta métrica se vea afectada por la ejecución del sistema de migración de datos en vivo, por lo que se decidió omitirla en la validación.

El monitoreo fue realizado utilizando el panel de control en GCP de la máquina virtual donde reside el Core de producción.

A continuación, en la figura 23 se muestra el uso de CPU del Core durante las 12:25 PM y 12:35 PM del dia que se realizó el monitoreo:

Figura 23: Métricas uso de CPU del Core 1

Los jobs de migración de datos en vivo fueron pausados a las 12:20 PM, por lo que el inicio del gráfico representa el uso normal de CPU del Core. Se realizó una ejecución de los jobs de migración de datos en vivo a las 12:30 PM, se nota que esto generó un incremento en el uso de CPU del sistema. A continuación, en las figuras 24 y 25 se ven los porcentajes específicos de uso de CPU antes y durante la ejecución de estos:

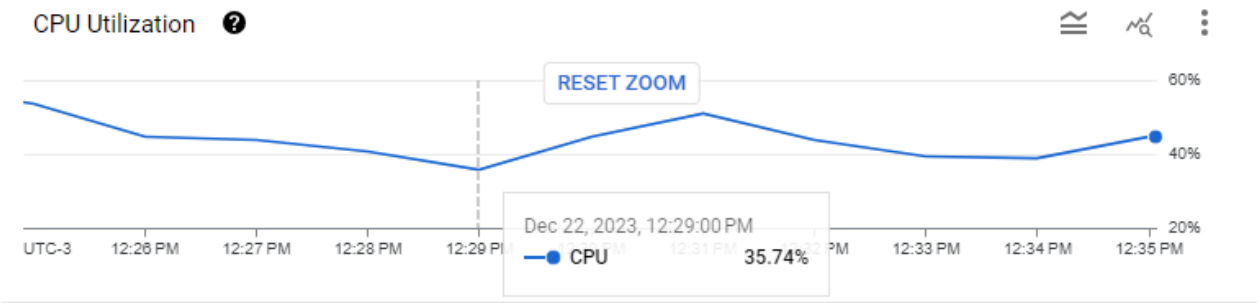

Figura 24: Métricas uso de CPU del Core 2

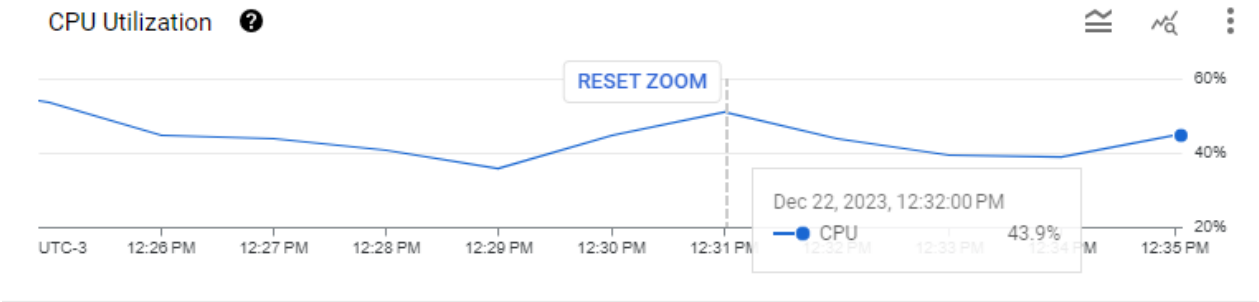

Figura 25: Métricas uso de CPU del Core 3

Como se puede ver, se genera un incremento de un 8.16% de uso de CPU entre las 12:29 y las 12:31. Es importante notar que esto corresponde a una cota máxima del incremento de uso de recursos que podrían generar los jobs de migración de datos en vivo. Dado que el motor de BD no fue aislado durante este monitoreo, parte del incremento mostrado en ese periodo puede ser atribuido a otro proceso dentro de la BD ajeno al proyecto.

A partir de lo anterior, se puede concluir que el sistema de migración de datos en vivo del proyecto no presenta un riesgo significativo para la continuidad operacional de la compañía, ya que su máximo impacto es de un 8.16% sobre la CPU del Core. Adicionalmente, el monitoreo se llevó a cabo en el entorno productivo durante el horario de operación, por lo que las condiciones presentadas reflejan adecuadamente la carga del Core en un día estándar de la compañía.

6. Conclusiones

En conclusión, el proyecto de disponibilización de servicios para corredores en el ámbito de seguros generales busca otorgar las condiciones necesarias para que se produzcan integraciones de corredores más pequeños.

Para lograr el objetivo del proyecto, se implementó un proceso de enrolamiento de fácil uso para los corredores y un sistema de autenticación que protege la confidencialidad de la información. Adicionalmente, se definieron prácticas de desarrollo alineadas con la arquitectura REST que permiten proyectar una buena cultura de desarrollo a los integradores. Esta arquitectura y prácticas de desarrollo son aplicadas en la implementación de tres APIs de consulta, una para pólizas, otra para siniestros y otra para cuotas, las cuales son genéricas para evitar futuros desarrollos a medida para los integradores.

Finalmente, la solución incluye el levantamiento de una segunda BD y la implementación de un sistema de migración de datos histórica y en vivo que permite que las APIs del proyecto no generen un impacto significativo en el Core de la compañía y puedan consumir información actualizada. Esto permite dar la seguridad de que el proyecto no ponga en riesgo la continuidad operativa de la empresa y permita a los integradores consultar información válida.

A partir de la encuesta de validación realizada se puede determinar que la solución implementada cumple razonablemente con los objetivos propuestos desde el punto de vista de los participantes. Sin embargo, se evidencian oportunidades de mejora en el sistema de enrolamiento y de autenticación. Se requerirá una evaluación acorde en el futuro para determinar si los beneficios esperados por la solución son efectivamente logrados. De momento esta evaluación no está considerada dentro de la memoria ni el proyecto por lo que dependerá de FID si se quiere realizar un análisis más detallado sobre los impactos de este.

Adicionalmente, se validó que la ejecución del sistema de migración de datos en vivo no afecta la continuidad operacional de la compañía. Se compararon los niveles de CPU utilizados por el Core sin la ejecución del sistema de migración y durante su ejecución. Las métricas recolectadas indican de manera segura que el sistema de migración de datos no representa un riesgo en este aspecto.

Se espera que el trabajo presentado en esta memoria tenga un impacto significativo en la viabilidad de ofrecer servicios a corredores con un menor volumen de ventas. Actualmente, el proyecto tiene un impacto sobre la planificación estratégica de la compañía, dado que dentro de los objetivos anuales se incluyeron metas relacionadas con la integración de corredores mediante este sistema. Esto incentiva al área comercial a promocionar los beneficios del proyecto a posibles partes interesadas, incrementando las posibilidades de que un corredor decida integrarse y utilizar los servicios expuestos.

Este proyecto corresponde al primer acercamiento de la empresa al mundo del Open Insurance. En el futuro este proyecto podría ser ampliado para incluir nuevos servicios y soportar distintos tipos de integradores. Actualmente el sistema incluye a integrador y corredor como actores, pero para

lograr incorporar completamente la compañía al paradigma de Open Insurance se debe incluir al cliente final dentro de los actores de la autorización y autenticación. Los diseños fueron creados con la intención de ser expandidos en el futuro para incluir esta nueva funcionalidad pero depende de FID otorgarle la prioridad necesaria al proyecto para poder incorporar estos aspectos.

Como continuación del trabajo presentado en esta memoria, se propone expandir el catálogo de APIs disponibles. En particular, la inclusión de una API de venta en línea de seguros sería beneficioso, dado que promovería integraciones de empresas que actualmente no son corredores de la compañía, generando nuevos canales de venta. Las tres APIs expuestas inicialmente no cumplen este propósito, ya que los servicios ofrecidos solo tienen valor para los corredores que ya pertenecen a FID.

Adicionalmente, se debe trabajar en mejorar la experiencia de integración, esto podría ser abordado mediante la implementación de un developer portal. Este corresponde a un punto centralizado en la Web que sirve como un recurso esencial para los desarrolladores. Entre sus funciones principales se encuentran: mostrar la documentación de las APIs disponibles, disponibilizar herramientas de prueba, ofrecer soporte a la comunidad, facilitar el proceso de onboarding, entre otras [84]. La inclusión de un developer portal al proyecto facilitaría y fomentaría la integración de nuevos corredores a FID.

7. Bibliografía

[1] ¿Qué es un Corredor de Seguros? (s/f). Allianz Seguros. Recuperado el 28 de agosto de 2023, de https://www.allianz.es/descubre-allianz/mediadores/diccionario-de-seguros/c/corredor-seguros.html

[2] La transformación de las compañías de seguros en la era digital. (2017, marzo 8). Deloitte Uruguay. https://www2.deloitte.com/uy/es/pages/strategy-operations/articles/La-transformacion-de-las-companias-de-seguros-en-la-era-digital.html

[3] InsurTech+. (2022, septiembre 10). Insurtech: ¿Qué es y cómo está revolucionando el sector de los seguros? Insurtech+. https://insurtech.plus/insurtech/

[4] Odreman, R. (2023, mayo 2). Todo lo que debes saber sobre Open Insurance. Nubloq. https://nubloq.co/blog/todo-lo-que-debes-saber-sobre-open-insurance

[5] Line, E. O. (2023, marzo 3). FID Seguros supera los US$ 100 millones de venta en 2022. Diario Estrategia. https://www.diarioestrategia.cl/texto-diario/mostrar/4198850/fid-seguros-supera-us-100-millones-venta-2022

[6] ¿Quiénes somos? (2021, agosto 12). FID Seguros. https://www.fidseguros.cl/nosotros/

[7] Ruiz, F. (2020, octubre 23). ¿Qué es API management? Finerio Connect. https://blog.finerioconnect.com/que-es-api-management/

[8] Ramos, J. (s/f). ¿Qué es el Daily Scrum Meeting? Programacionymas.com. Recuperado el 11 de diciembre de 2023, de https://programacionymas.com/blog/daily-scrum-meeting

[9] ¿Qué es una API y cómo funciona? (s/f). Redhat.com. Recuperado el 27 de agosto de 2023, de https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

[10] Gupta, L. (2018, mayo 29). What is REST? REST API Tutorial; Lokesh Gupta. https://restfulapi.net/

[11] Roca, C. (2023, junio 9). REST API: what it is, how it works, advantages and disadvantages. ThePower Business School; ThePowerMBA. https://www.thepowermba.com/en/blog/rest-api-what-it-is

[12] Garcia, C. (2023, septiembre 6). What are the benefits of RESTful APIs? Appmaster.Io; AppMaster. https://appmaster.io/blog/what-are-the-benefits-of-restful-apis

[13] Resources. (s/f). Readthedocs.Io. Recuperado el 8 de diciembre de 2023, de https://restful-api-design.readthedocs.io/en/latest/resources.html

[14] Introduccci ón 1 Rigor y Formalidad: 1 Separaci ón de Intereses: 2 Modularidad: 3 Abstracci Ón: 3 Anticipaci ón, al C. (s/f). Principios de la Ingeniería de Software. Edu.uy. Recuperado el 6 de diciembre de 2023, de https://www.fing.edu.uy/tecnoinf/mvd/cursos/ingsoft/material/teorico/PrincipiosIngenieriaSoftware.pdf

[15] URI - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web. (s/f). MDN Web Docs. Recuperado el 3 de diciembre de 2023, de https://developer.mozilla.org/es/docs/Glossary/URI

[16] Gupta, L. (2018c, junio 2). HATEOAS driven REST APIs. REST API Tutorial; Lokesh Gupta. https://restfulapi.net/hateoas/

[17] Sheldon, R. (2023, enero 20). Caching. Whatis.com; TechTarget. https://www.techtarget.com/whatis/definition/caching

[18] Gupta, L. (2018a, mayo 8). REST architectural constraints. REST API Tutorial; Lokesh Gupta. https://restfulapi.net/rest-architectural-constraints/

[19] Lutkevich, B. (2022, julio 7). What is access control? Security; TechTarget. https://www.techtarget.com/searchsecurity/definition/access-control

[20] What is an API Gateway? (s/f). Kong Inc. Recuperado el 9 de diciembre de 2023, de https://konghq.com/learning-center/api-gateway/what-is-an-api-gateway

[21] Distrito, K. (s/f). ¿Qué es y para qué sirve una API KEY? Distritok.com. Recuperado el 2 de diciembre de 2023, de https://www.distritok.com/blog/que-es-y-para-que-sirve-una-api-key/

[22] What is an API key? (s/f). Fortinet. Recuperado el 2 de diciembre de 2023, de https://www.fortinet.com/resources/cyberglossary/api-key

[23] API Authentication and Authorization: 6 methods and tips for success. (2023, julio 5). Frontegg. https://frontegg.com/guides/api-authentication-api-authorization

[24] Magaña, L. M. L. (2020, enero 17). Qué es Json Web Token y cómo funciona. Openwebinars.net. https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/

[25] OAuth vs. JWT: What is the difference & using them together. (2023, septiembre 8). Frontegg. https://frontegg.com/blog/oauth-vs-jwt

[26] ¿Qué es cloud computing? (s/f). Google Cloud. Recuperado el 4 de diciembre de 2023, de https://cloud.google.com/learn/what-is-cloud-computing?hl=es

[27] Zhang, M. (2023, octubre 10). Top 10 cloud service providers globally in 2023. Dgtl Infra. https://dgtlinfra.com/top-cloud-service-providers/

[28] (S/f-d). Cloudflare.com. Recuperado el 4 de diciembre de 2023, de https://www.cloudflare.com/learning/cloud/what-is-a-virtual-private-cloud/

[29] ¿Qué es una nube pública? (s/f). Google Cloud. Recuperado el 6 de diciembre de 2023, de https://cloud.google.com/learn/what-is-public-cloud?hl=es-419

[30] (S/f-e). Amazon.com. Recuperado el 6 de diciembre de 2023, de https://aws.amazon.com/es/what-is/public-cloud/

[31] (S/f-f). Microsoft.com. Recuperado el 6 de diciembre de 2023, de https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-are-private-public-hybrid-clouds

[32] ¿Qué son los contenedores? (s/f). Google Cloud. Recuperado el 4 de diciembre de 2023, de https://cloud.google.com/learn/what-are-containers?hl=es

[33] Aglave, S. (2023, agosto 3). Most popular Container Runtimes. Cloudraft. https://www.cloudraft.io/blog/container-runtimes

[34] What is Container Management - VMware Glossary. (2023, octubre 16). VMware. https://www.vmware.com/topics/glossary/content/container-management.html

[35] Descripción general de GKE. (s/f). Google Cloud. Recuperado el 7 de diciembre de 2023, de https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview?hl=es-419

[36] (S/f). Bciseguros.cl. Recuperado el 19 de noviembre de 2023, de https://ayuda.bciseguros.cl/hc/es-419/articles/360051510031--Qu%C3%A9-es-el-Proceso-de-Inspecci%C3%B3n-de-un-veh%C3%ADculo-#:~:text=Volver&text=Es%20el%20proceso%20por%20el,%C2%BFFue%20%C3%BAtil%20este%20art%C3%ADculo%3F

[37] Maluenda, R. (2022, mayo 11). Qué es y qué hace un QA en proyectos de desarrollo de software. Profile Software Services. https://profile.es/blog/analista-qa/

[38] Inicio. (2016, noviembre 25). SISTRAN. https://www.sistran.com/latam/es/inicio/

[39] Barke, E. (2016, octubre 29). Problemas de desempeño de cursores en SQL Server. SQL Shack - articles about database auditing, server performance, data recovery, and more; SQL Shack. https://www.sqlshack.com/es/problemas-de-desempeno-de-cursores-en-sql-server/

[40] Use VPC network peering. (s/f). Google Cloud. Recuperado el 15 de noviembre de 2023, de https://cloud.google.com/vpc/docs/using-vpc-peering

[41] About. (s/f). Nodered.org. Recuperado el 20 de noviembre de 2023, de https://nodered.org/about/

[42] Grabeljsek, S. (2017, enero 26). ¿Qué es una plataforma Low-Code y cómo puede ayudar a tu empresa? Grupocibernos.com. https://www.grupocibernos.com/blog/desarrollo-de-software/una-plataforma-low-code-puede-ayudar-empresa

[43] Más información sobre el SaaS. (s/f). Oracle.com. Recuperado el 20 de noviembre de 2023, de https://www.oracle.com/cl/applications/what-is-saas/

[44] (S/f-b). Amazon.com. Recuperado el 23 de noviembre de 2023, de https://aws.amazon.com/es/what-is/vpn/#:~:text=Una%20VPN%20o%20red%20privada,a%20trav%C3%A9s%20de%20redes%20p%C3%BAblicas.

[45] Gupta, L. (2018, mayo 24). HTTP methods. REST API Tutorial; Lokesh Gupta. https://restfulapi.net/http-methods/

[46] Gupta, L. (2018a, abril 1). How to design a REST API. REST API Tutorial; Lokesh Gupta. https://restfulapi.net/rest-api-design-tutorial-with-example/

[47] Stemmler, K. (2021, diciembre 21). Camel case vs. Snake case vs. Pascal case — naming conventions. Khalilstemmler.com. https://khalilstemmler.com/blogs/camel-case-snake-case-pascal-case/

[48] Paginación API. (s/f). Appmaster.io. Recuperado el 20 de septiembre de 2023, de https://appmaster.io/es/glossary/paginacion-api

[49] Documentación de Cloud○Endpoints. (s/f). Google Cloud. Recuperado el 23 de noviembre de 2023, de https://cloud.google.com/endpoints/docs?hl=es-419

[50] Brokeris - Sistema para Corredores de Seguros. (s/f). Brokeris.Cl. Recuperado el 20 de septiembre de 2023, de https://brokeris.cl/

[51] Cloud SQL. (s/f). Google.com. Recuperado el 22 de octubre de 2023, de https://cloud.google.com/sql?hl=es

[52] ¿Cuáles son las herramientas ETL? (2022, mayo 19). KeepCoding Bootcamps. https://keepcoding.io/blog/herramientas-etl/

[53] What we do and how we got here. (s/f). Gartner. Recuperado el 19 de octubre de 2023, de https://www.gartner.com/en/about

[54] Magic Quadrant research methodology. (s/f). Gartner. Recuperado el 19 de octubre de 2023, de https://www.gartner.com/en/research/methodologies/magic-quadrants-research

[55] Aguirre, M. F. (2021, enero 26). Prueba de Concepto (PoC): qué es y ejemplo de su utilidad. appvizer.es; Appvizer. https://www.appvizer.es/revista/organizacion-planificacion/gestion-proyectos/prueba-de-concepto

[56] Acelera tu negocio a un negocio digital. (2023, febrero 28). Prodig.io. https://www.prodigio.tech/

[57] Descripción general de BigQuery. (s/f). Google Cloud. Recuperado el 23 de noviembre de 2023, de https://cloud.google.com/bigquery/docs/introduction?hl=es-419

[58] Talend Academy. (2021, junio 8). Talend - A Leader in Data Integration & Data Integrity. https://www.talend.com/academy/

[59] Pricing. (s/f). Google Cloud. Recuperado el 22 de octubre de 2023, de https://cloud.google.com/sql/pricing

[60] Cloud SQL feature support by database engine. (s/f). Google Cloud. Recuperado el 22 de octubre de 2023, de https://cloud.google.com/sql/docs/feature_support

[61] (S/f). Rae.es. Recuperado el 22 de octubre de 2023, de https://dle.rae.es/enrolar

[62] Simplilearn. (2020, marzo 26). What is data encryption: Types, algorithms, techniques and methods. Simplilearn.com; Simplilearn. https://www.simplilearn.com/data-encryption-methods-article

[63] Top 8 strongest data encryption algorithms in cryptography. (s/f). Webandcrafts.com. Recuperado el 22 de octubre de 2023, de https://webandcrafts.com/blog/data-encryption-algorithms

[64] 5 common encryption algorithms and the unbreakables of the future. (2023, septiembre 19). Arcserve. https://www.arcserve.com/blog/5-common-encryption-algorithms-and-unbreakables-future

[65] ¿Qué es un clúster de Kubernetes? (s/f). Redhat.com. Recuperado el 30 de noviembre de 2023, de https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-cluster

[66] Conectarse desde Kubernetes Engine. (s/f). Google Cloud. Recuperado el 14 de noviembre de 2023, de https://cloud.google.com/sql/docs/mysql/connect-kubernetes-engine?hl=es

[67] Compute Engine. (s/f). Google Cloud. Recuperado el 17 de noviembre de 2023, de https://cloud.google.com/compute?hl=en

[68] Scalable Tyk Cloud pricing plans - API Management Platforms. (2023, marzo 6). Tyk API Management; Tyk. https://tyk.io/pricing-cloud/

[69] Multi-cloud & hybrid API management - tyk. (2021, noviembre 16). Tyk API Management; Tyk. https://tyk.io/deployment-multi-cloud-and-hybrid/

[70] Self-managed pricing. (2023, marzo 6). Tyk API Management; Tyk. https://tyk.io/pricing-self-managed/

[71] Product overview. (s/f). Google Cloud. Recuperado el 16 de noviembre de 2023, de https://cloud.google.com/armor/docs/cloud-armor-overview

[72] ¿Qué es SSL? (2022, septiembre 29). GlobalSign. https://www.globalsign.com/es/centro-de-informacion-ssl/que-es-ssl

[73] What is change data capture (CDC)? (s/f). Microsoft.com. Recuperado el 3 de noviembre de 2023, de https://learn.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server?view=sql-server-ver16

[74] WilliamDAssafMSFT. (s/f). Database snapshots (SQL Server). Microsoft.com. Recuperado el 3 de noviembre de 2023, de https://learn.microsoft.com/en-us/sql/relational-databases/databases/database-snapshots-sql-server?view=sql-server-ver16

[75] Arquitectura de productor/consumidor en LabVIEW. (2023, julio 21). Www.ni.com. https://www.ni.com/es/support/documentation/supplemental/21/producer-consumer-architecture-in-labview0.html.html

[76] API Definition object. (s/f). Tyk.Io. Recuperado el 17 de noviembre de 2023, de https://tyk.io/docs/getting-started/key-concepts/what-is-an-api-definition/

[77] Security policy. (s/f). Tyk.Io. Recuperado el 17 de noviembre de 2023, de https://tyk.io/docs/getting-started/key-concepts/what-is-a-security-policy/

[78] (S/f-c). Tyk.io. Recuperado el 28 de noviembre de 2023, de https://tyk.io/docs/tyk-apis/tyk-dashboard-api/api-keys/

[79] Welcome to talend help center. (s/f). Talend.com. Recuperado el 29 de noviembre de 2023, de https://help.talend.com/r/en-US/7.3/studio-user-guide/what-is-talend-studio

[80] Atlassian. (s/f). Como crear un repositorio de Git. Atlassian. Recuperado el 15 de diciembre de 2023, de https://www.atlassian.com/es/git/tutorials/setting-up-a-repository

[81] Qué puedes hacer con Formularios. (s/f). Google.com. Recuperado el 15 de diciembre de 2023, de https://support.google.com/a/users/answer/9302965?hl=es

[82] Forbes, I. (2023, agosto 31). Insurtech: Una revolución explosiva en el mundo de los seguros. Forbes México. https://www.forbes.com.mx/insurtech-una-revolucion-explosiva-en-el-mundo-de-los-seguros/

[83] rwestMSFT. (s/f). Memory management architecture guide - SQL Server. Microsoft.com. Recuperado el 23 de diciembre de 2023, de https://learn.microsoft.com/en-us/sql/relational-databases/memory-management-architecture-guide?view=sql-server-ver15

[84] Kiselev, R. (2023, diciembre 20). What is an API developer portal? The ultimate guide. What Is an API Developer Portal? The Ultimate Guide | Moesif Blog. https://www.moesif.com/blog/api-strategy/What-is-API-Developer-Portal/