UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO
FACULTAD DE INGENIERÍA

"Documentación de la implementación de una aplicación web para la administración de usuarios y perfiles para el manejo de pagos al extranjero de una institución bancaria"

Memoria de experiencia laboral

Presenta:

Valery Michelle Sierra Almazán

Asesor:

Dr. Marcelo Romero Huertas

Toluca, México, agosto 2021

Resumen

En este trabajo se plasmaron las experiencias profesionales durante la implementación de una aplicación *web* basada en Angular y Spring Boot para la administración de usuarios y perfiles de cuatro sistemas de pago al extranjero de una institución bancaria. Antes del desarrollo documentado en esta memoria, se tenía cada sistema por separado, cada uno era un monolito diferente y cada perfil se generaba en el sistema correspondiente al igual que los usuarios, por lo que un mismo usuario podía tener varios perfiles en los diferentes sistemas. La solución para esto fue unificar los 4 sistemas en un mismo módulo de creación de perfiles y usuarios, de modo que se tuvieran cada una de las diferentes opciones para cada sistema.

Dicha solución fue diseñada con microservicios y con una arquitectura de tres capas llamada Api Led Connectivity, se trabajó sobre Spring Boot con Java 8, se generó una nueva base de datos centralizada utilizando Oracle 12c y se utilizó Angular y SAAS para el desarrollo de la parte visual de la aplicación web. Como resultado, se obtuvo una aplicación capaz de administrar los usuarios y perfiles para los mismos en cada uno de los sistemas con la capacidad de generar múltiples perfiles sin tener que gestionar cada uno por separado. Además, implementar este módulo con microservicios resultó beneficioso para diferentes sistemas de la empresa, ya que sólo con modificar las llaves de acceso dentro de la base de datos podría ser adaptado a diferentes funcionalidades.

Con el desarrollo de esta solución pude obtener el conocimiento necesario para la implementación de microservicios y su puesta en marcha, ya que dentro de las actividades que se realizaron están el análisis y levantamiento de requerimientos que se tenían de cada una de las funcionalidades, así como el desarrollo y comunicación entre los compañeros desarrolladores y con el cliente; además, se tuvo que realizar el despliegue en el servidor de aplicaciones; y finalmente se hicieron pruebas que ayudaron a confirmar una mayor eficiencia gracias a esta solución.

Índice

Introducción

Este trabajo es una documentación que describe la problemática, solución y los resultados del desarrollo de una aplicación web para administrar usuarios y perfiles, derivados de mi experiencia profesional trabajando en una institución del sector financiero. Va dirigida a los desarrolladores y analistas que buscan una solución de acceso a los sistemas con una arquitectura que tenga la flexibilidad de cambios continuos, por lo que es necesario que el lector tenga conocimiento básicos-intermedios sobre el desarrollo de sistemas web para mejorar la comprensión.

En el mundo de las finanzas, existen diferentes sistemas para la administración y manejo de todas y cada una de las transacciones que se realizan día tras día, principalmente en el sector bancario. Hay pagos que se realizan en el banco directamente, pero actualmente también los hacemos de manera digital, ya sea desde una aplicación móvil en el celular, o desde la banca electrónica en una computadora, esto solo por mencionar algunos ejemplos del amplio mundo de las Tecnologías de la Información (IT) que cada vez son más presentes y necesarios en el día a día de los seres humanos.

Derivado de lo anterior, hoy en día tenemos la tarea de llevar a cabo y facilitar actividades que son parte de las tareas diarias de las personas, por ejemplo, el trámite que realiza un cliente cuando acude a una sucursal, las actividades que realiza el personal financiero que labora dentro de diferentes instituciones, o bien, pueden ser aquellos movimientos que se hacen para la gestión de procesos e información de las personas, entre otros. Si multiplicamos estas acciones por las miles de personas en el mundo que realizan una tarea de este tipo al día, y luego lo multiplicamos por los días que tiene un año, podemos darnos cuenta de que los problemas pueden ser mayores en el caso de un mal manejo de información, más aún cuando en el diseño de los sistemas no se contemplan escenarios bajo los cuales es necesario operar una aplicación de tal responsabilidad.

Cuando una institución o empresa ha trabajado durante un largo periodo bajo las mismas reglas, normas o herramientas, la idea de un cambio puede resultar una mala práctica por los riesgos que éste conlleva. Es por eso que hoy en día, aún bajo el inmenso mundo de la tecnología y una creciente demanda de evolución, existen quienes todavía se rehúsan al cambio por miedo a disminuir el porcentaje de consumidores. A pesar de que puede ser seguro seguir bajo los mismos estándares y las mismas herramientas de trabajo que han operado por años, es necesario construir un cambio e ir en pro de los avances tecnológicos para estar siempre a la vanguardia y con la pila tecnológica más actual.

¿Cuántas veces no hemos visto que cuando hacemos una visita a algún banco para realizar un trámite, los expertos en el sistema hacen uso de varias herramientas y sistemas en los que a veces no se hace más que llevar a cabo actividades similares? Derivado de la pregunta anterior, es necesario reconocer que las personas encargadas de la gestión de permisos deben tener control de lo que cada persona puede o no hacer dentro de cada uno de los sistemas, lo que resulta en un conjunto de tareas que se añade a la carga de trabajo y finalmente, como un todo, termina siendo algo difícil.

Existen ciertas herramientas que, por el valor que tenían años atrás, se programaron de forma aislada y fueron asignadas para realizar una única cosa bien, pero actualmente las diferentes operaciones que se encargaban de hacer esto, se han estado descartando y actualizando, lo cual también implica una nueva planeación, así como una definición de arquitectura y el uso de herramientas y tecnologías que, además de brindar seguridad, agilicen las operaciones con la única finalidad de satisfacer las necesidades de los clientes.

Desde hace unos años, la actualización de las tareas y actividades han estado buscando ser similares y que además se vuelvan genéricas para que varios clientes puedan hacer uso de ellas de una forma más sencilla.

Es por lo anterior que el presente proyecto se desenvuelve en el ámbito financiero y más específicamente en el sector bancario, y se tiene como objetivo la documentación

del desarrollo e implementación de una aplicación web, cuya finalidad es tener una administración y control de accesos para cada una de las personas que interactúan con los sistemas propios del banco. Cabe recalcar que un buen control garantiza un correcto funcionamiento del sistema y, principalmente, la integridad de la información de los clientes, por lo que se puso en marcha un plan en el que se tenía que modificar la forma de controlar los accesos al sistema de envío de dinero al extranjero.

El desarrollo consistió en realizar la gestión de todos los usuarios integrando diferentes sistemas o módulos que interactúan con el sistema mencionado. Esto considerando que hoy en día, la operación de estos sistemas se ha vuelto muy tediosa por la forma en la que se necesita interactuar con ellos.

A continuación, se explicará brevemente y de forma general la forma en la que se administraban los accesos anteriormente: se necesitaban los permisos necesarios para cada uno de los sistemas y que las personas encargadas de esta gestión generaran la creación de cada usuario en cada sistema (módulo) y le generaran el perfil o rol indicado en cada uno de ellos.

Con este proyecto se buscó facilitar la administración de todos los usuarios en estos módulos, así como la gestión que se realiza al asignar un rol específico de cada uno, en solo este sistema.

Para el desarrollo de este sistema se utilizó la arquitectura de microservicios tomando como base los servicios REST, y es así como parte de la infraestructura Backend, la cual se integra para tener la capacidad de realizar cambios en cada una de las acciones sin generar repercusiones en las demás. Más adelante se encontrarán más detalles sobre esta nueva arquitectura y su implementación.

Por otro lado, y como parte del Frontend, se realizó el diseño y maquetado de las pantallas para un mejor manejo de los servicios, dándole al usuario una mejor vista y generándole una mejor experiencia al usuario dentro del sistema. Sin embargo, éste no será un tema que se podrá abordar a detalle puesto que el cargo que desempeñé

fue principalmente como desarrolladora de Backend. Este proyecto tuvo una duración aproximada de 9 meses.

Objetivo general de la memoria laboral

Plasmar las experiencias profesionales durante la implementación de una aplicación web utilizando la plataforma de desarrollo Angular y Spring Boot, para la administración de usuarios y perfiles en el manejo de pagos al extranjero de una institución bancaria.

Alcances y limitaciones

Dentro de la empresa donde se realizó el proyecto, existían varios grupos encargados de la validación del sistema, desde el equipo que realizaba las modificaciones necesarias de los sistemas anteriores, hasta las personas que ocupaban estos sistemas día a día; pasando por los equipos encargados de seguridad, arquitectura e infraestructura.

La planeación del proyecto fue determinada por el cliente de la institución bancaria, y teniendo en cuenta los cambios en el sistema, fue que se decidió emplear tecnología más actual, ya que los sistemas utilizaban versiones antiguas. Una vez identificados cada uno de los puntos a acatar para la implementación, estos fueron plasmados bajo el uso de metodologías ágiles y, para los cuales, en este caso se hizo uso de SCRUM.

Algunas de las modificaciones fueron determinadas por el usuario mientras que el resto se mantuvo bajo el funcionamiento actual de cada sistema, intentando homologar todos los servicios en donde se realiza la configuración de usuarios y perfiles.

Directriz de la memoria

Esta memoria de experiencia laboral va dirigida a los desarrolladores y analistas que buscan una solución de acceso a los sistemas con una arquitectura que tenga la flexibilidad de cambios continuos.

Organización del documento

El contenido de esta memoria está estructurado de la siguiente forma:

Capítulo 1. Antecedentes: muestra una descripción detallada de la problemática.

Capítulo 2. Diseño de la solución*:* muestra una descripción de las arquitecturas propuestas para la solución del problema, así como el diseño final que se construyó.

Capítulo 3. Resultados: muestra el *stack* tecnológico usado y los resultados de las pruebas.

Capítulo 1. Antecedentes

En este capítulo muestro una descripción detallada de la problemática solucionada y la empresa objetivo.

1.1 Empresa objetivo

En este trabajo documento el desarrollo del *software* realizado para una empresa del sector financiero, que por razones de confidencialidad no se revela su identidad ni los datos y procesos no públicos.

El proyecto tuvo una duración de 9 meses y el rol que desempeñé lleva por nombre "analista de desarrollo de aplicaciones", en donde mis actividades específicas fueron:

- Análisis de la aplicación a migrar para obtener los requerimientos funcionales.
- Diseño de microservicios para cumplir con los requerimientos utilizando una arquitectura de API Led Connectivity.
- Diseño y construcción de base de datos.
- Desarrollo de microservicios con Spring Boot utilizando Java 8.
- Despliegue de microservicios en el servidor.
- Apoyo en la solución de incidencias detectadas por el equipo de *testing*.

Dentro del tema laboral y hablando estrictamente de las empresas que se encargan de ofrecer servicios, se encuentra una clasificación, la cual hace referencia al sector en el que se desarrolla. En esta ocasión, la implementación de la aplicación en cuestión la realizamos para una empresa que, a nivel global, se desenvuelve en el giro financiero, por lo que puede utilizarse para los negocios que se dedican a la intermediación financiera, pueden ser empresas de financiamiento comercial, corporaciones de ahorro y vivienda, bancos, entre otros.

Los servicios que presta el sistema financiero son múltiples y cada día son más necesarios para el comportamiento económico de particulares y empresas. Para ello,

este sistema realiza 3 subsunciones: Captación, Canalización y Asignación de Recursos Financieros (UNEP, 2010).

Como ya se mencionó, dentro de las 3 subsunciones, la captación de capital se refiere a la cantidad de recursos disponibles para la inversión, donde depende la capacidad del sistema para captar ahorro. La canalización debe de tener una muy buena estructura diversificada para poder ofrecer a los inversores un acceso sencillo a las fuentes de financiación, ya sea para ahorros o inversiones, con un coste de intermediación mínimo. La asignación de recursos se debe realizar para que se obtengan las mejores oportunidades de inversión, para el impulso de las empresas. La creación de la liquidez se deriva de que el 10% de la liquidez total es dinero no físico, el resto del dinero puede ser creado por entidades financieras con la concesión de préstamos y créditos y la disponibilidad de fondos captados a los ahorradores.

Tener en cuenta los principios anteriores es fundamental para entender cómo la empresa ha entrado a ser parte de este sector, logrando tener varias áreas que le permitan ejercer diversas funciones que se interpreten en servicios para los clientes de todo el mundo.

Gracias al éxito que año tras año ha ido creando la institución bancaria, hoy en día es una de las mayores organizaciones en el sector financiero del mundo; gracias al número de clientes que, a través de sus negocios globales, ha ido adquiriendo y a los más de 64 países en los diferentes continentes en los que está presente. Su objetivo es ir a la par del crecimiento tecnológico, crear oportunidades para los clientes e impulsar a las empresas y a las economías, siendo su inspiración ayudar a la gente a hacer sus sueños realidad.

Gracias a la estrategia de la institución, ésta se ha posicionado como uno de los bancos internacionales líderes del mundo, y es que sus valores hacen que siempre se aseguren de que, tanto sus empleados como sus clientes, se sientan impulsados para hacer lo correcto y actuar con integridad y valentía.

Además de aportar en el sector financiero y bancario, ha ayudado a que tan solo aquí en México, el porcentaje de profesionistas que consiguen su primer empleo sea mayor, ya que su compromiso con las personas va más allá de atraer clientes, de modo que ayuda y gestiona diferentes programas de estudio para todas y cada una de las personas que deseen desarrollar sus habilidades y lograr sus ambiciones profesionales en un entorno diverso e inclusivo.

Sus negocios se dividen en los siguientes grupos:

- **Finanzas personales:** Suministra servicios financieros a más de 125 millones de clientes alrededor de todo el mundo. Estos servicios incluyen cuentas corrientes, cuentas de ahorro, hipotecas, seguros, tarjetas de crédito, préstamos, pensiones e inversiones.
- **Banca comercial:** El banco tiene como clientes casi 2.5 millones de pequeñas y medianas empresas.
- **Corporativo, banca de inversión y mercados:** En esta área de negocios, el banco provee servicios financieros a clientes corporativos e institucionales referidos a mercados globales, banca corporativa e institucional, transacciones bancarias globales y banca de inversión global.
- **Banca privada.**

La evolución que ha tenido la empresa le ha ayudado a seguir en el sector global brindando todos estos servicios, además de tener un crecimiento considerable tanto en el sector financiero, como en todo lo que conlleva estos nuevos años.

Derivado del éxito de la empresa del que ya se ha hablado, es que surgen nuevas ideas, metas y objetivos por estar siempre disponible para los clientes y además competir por los primeros lugares de reconocimiento junto con otras empresas.

1.2 Problemática

Una parte importante de las empresas es que deben ir transformándose a la par de la evolución del mundo, si no es así, pueden llegar a quebrar porque ya no son autosustentables, lo cual produce que haya pérdida de empleos, además de otras consecuencias.

Como sabemos, el mundo y la tecnología se encuentran en constante cambio y evolución, y en este caso, dentro de este proyecto, la tecnología juega un papel muy importante. De igual forma, los sistemas de información deben ser necesariamente actualizados para garantizar la disponibilidad y seguridad que los clientes requieren. En ocasiones, estos cambios y actualizaciones tienen que ver con temas de seguridad, o simplemente con un tema de rendimiento, por lo que algunas veces se requiere que los cambios sean desde la raíz, en temas de arquitectura e infraestructura, o a veces solamente con adaptaciones en el *software*. Esto también se deriva de la necesidad de transformación que tienen otras empresas, o incluso personas particulares.

Un ejemplo muy sencillo pero que en la vida cotidiana se puede presentar infinidad de veces, es cuando tenemos que realizar una investigación de cierta información a través de internet, y es así como surge la siguiente pregunta: ¿Qué es lo que hacemos cuando una página de internet no funciona, o simplemente no nos proporciona lo que queremos? La respuesta es clara: cambiamos o buscamos alternativas que nos sean útiles. Esto mismo pasa con los sistemas de información, con los sitios web o con las aplicaciones que en el día a día tenemos que utilizar.

Una vez dicho lo anterior, en el sector financiero no existe ninguna excepción, porque cada vez más y más servicios se mueven de manera más eficaz, tanto aquí como en otros lados del mundo.

Hoy en día, dentro de la empresa para la que se llevó a cabo este proyecto, el realizar cambios y/o actualizaciones a los sistemas que ya se tenían implementados, fue un

reto muy costoso, ya que se utilizaba tecnología antigua y la arquitectura de los sistemas se construyó de tal manera que no veía más allá de posibles cambios.

Ocasionalmente, dentro de las prácticas de desarrollo de *software* se adoptan técnicas que usualmente resultan salidas rápidas, pero que no es lo correcto. Derivado de esto surge uno de los términos más usados dentro del desarrollo de *software*: "parchar código", que implica agregar y/o quitar mejoras que con anterioridad se le aplicaron al código fuente, lo cual fue necesario ya sea por errores, por cambios que surgieron posterior a la planeación o simplemente por una actualización. Hacer uso de esta técnica dentro de los diferentes sistemas de la empresa, puede ser con la finalidad de que el funcionamiento siga siendo el mismo, por lo que cada vez que se genere un cambio, esto implica más y más costo.

Otro de los problemas que surgen dentro del mantenimiento y desarrollo de *software* es que cuando existen aplicaciones o sistemas ya productivos a los que se requiere realizar mejoras, se presenta el caso de que, pasado tanto el tiempo, ya no existe ninguna persona que conozca en su totalidad la funcionalidad de todo el sistema, y se crean "huecos" de conocimiento para poder cambiarlo, esto como consecuencia de que los desarrolladores iniciales ya no laboran en la empresa o porque solo estuvieron por ciertos periodos como consultores, o porque simplemente nunca se realizó la documentación adecuada para poder darle seguimiento y mantenimiento en caso de requerirse.

En este proyecto se tomó en cuenta uno de todos los sistemas que existían dentro de la empresa y que se encargaba de realizar pagos al extranjero. El sistema de envío de pagos al extranjero estaba conformado por cuatro módulos diferentes principalmente, y cada uno de ellos enfrentaba una situación diferente de gestión del mismo, lo que implicaba que, aunque se realizaran actividades similares, no todos las trabajaban igual.

Estos sistemas eran utilizados por empleados de la misma empresa, cuya función era realizar las validaciones necesarias si algún envío no se podía hacer

automáticamente, y administrar las empresas con las que se trabajaba en otros lugares del mundo, el monto, entre otras características.

Sin embargo, los usuarios que trabajaban con un sistema tendían a trabajar con otro de estos, o hasta tenían un perfil de cada sistema, por lo que era problemático encontrar la página de inicio de cada uno de ellos, además de que era muy tardado por el sistema de acceso.

Para poder entrar a cada uno de los diferentes módulos existentes, era necesario pedir un permiso levantando un *ticket* para que se generaran un usuario y una contraseña con ciertos permisos en un sistema totalmente aparte. La Figura 1 muestra el diagrama del proceso para la creación de usuarios. Éste podía tardar de uno a tres días en realizarse y requería de la aprobación del gerente a cargo.

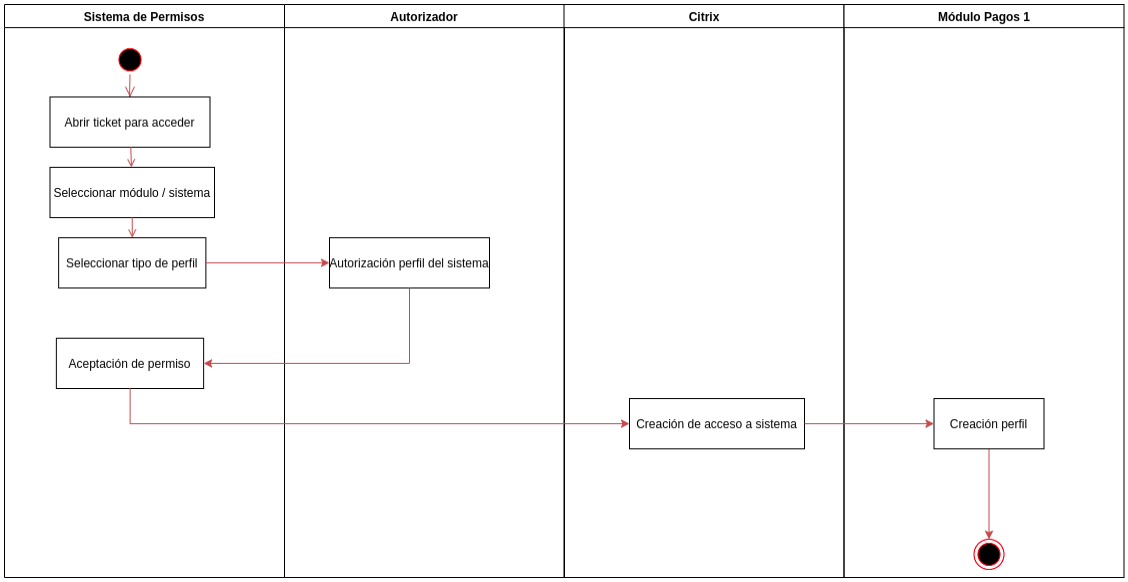

Figura 1. Diagrama del Proceso para Creación Perfil Anterior

Después de este proceso, se necesitaba contactar con la persona encargada de cada sistema con el perfil correspondiente que tenía los permisos necesarios para poder registrar al nuevo usuario en el sistema. Es importante recalcar que éste era un proceso por cada sistema al que se requería acceso, por lo que podía ser una persona diferente para cada módulo, sin embargo, en ese momento había sólo un equipo encargado de este trabajo.

Para poder entrar, se utilizaba la plataforma Citrix, que suministra tecnologías de virtualización de servidores, conexión en red, software-como-servicio (SaaS) e informática en la nube, entre las que se cuentan los productos de código abierto.

La nube es un suministro de servicios informáticos (incluidos servidores, almacenamiento, bases de datos, redes, *software*, análisis e inteligencia) a través de internet, cuyo objetivo es ofrecer una innovación más rápida, recursos flexibles y economías de escala.

Citrix, a pesar de brindar tantos beneficios, puede resultar lenta dependiendo de la red de la empresa, además de otros factores. El problema principal era que cada que se desconectaba del perfil, se tenía que esperar cierto tiempo para poder ingresar a la plataforma de nuevo, y así acceder al sistema requerido. En la Figura 2 muestro el diagrama de Funcionamiento de Acceso al Sistema que se tenía. Adentrándose a estos sistemas, se puede observar que cada módulo tenía diferentes formas de dar de alta a un usuario, ya que algunas pedían más datos que otros, por lo que en algunos módulos se tenía información escasa acerca de los usuarios.

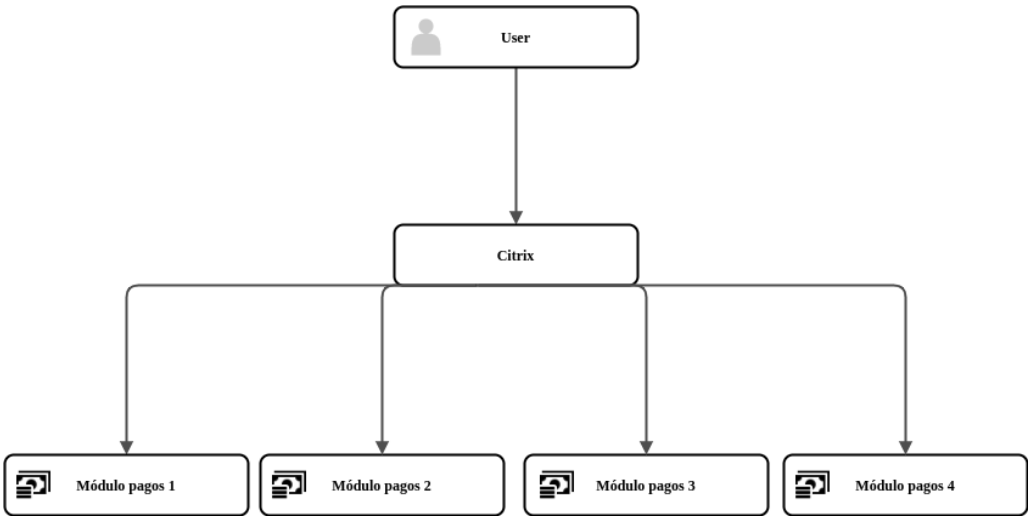

Figura 2. Diagrama Funcionamiento de Acceso Sistema Anterior

De la misma forma, la manera en la que se creaban los perfiles era totalmente diferente en cada módulo, pues había ciertos módulos en donde, seleccionando ciertos roles, se generaban las opciones de cada perfil, y había otros en los que era

necesario elegir cada opción para dar permisos a los mismos. En la Figura 3, se muestra un ejemplo de ciertas diferencias que había en los módulos.

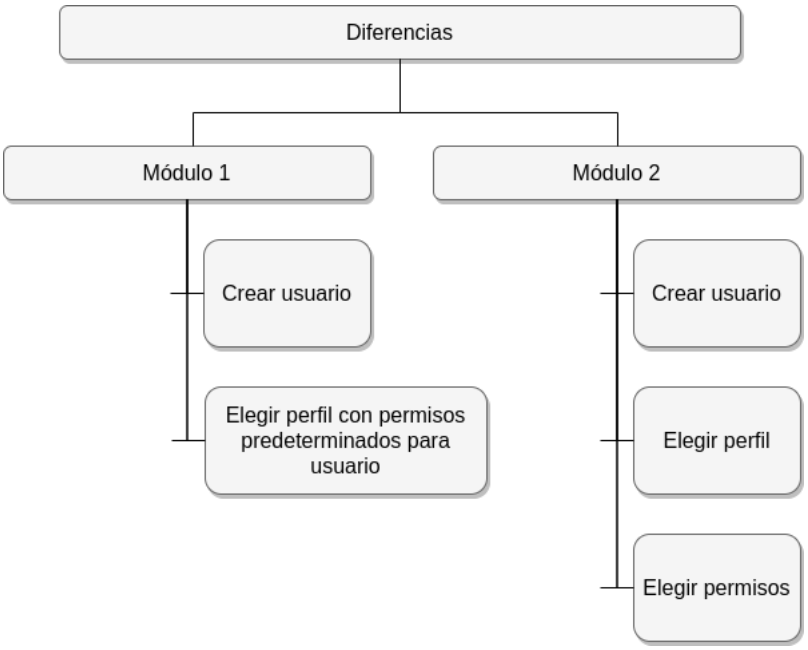

Figura 3. Diagrama diferencias entre Módulos

Durante el análisis tuvimos accesos limitados para la revisión total del sistema, pero detectamos esos conflictos entre módulos en su mayor parte.

El proyecto anterior estaba dividido en cuatro proyectos monolitos (más adelante hablaré de este tipo de arquitectura), los cuales empleaban un desarrollo que utilizaba Java 1.6 y JSP (tecnología orientada a crear páginas web en Java) para el desarrollo de las pantallas. Los servicios estaban llegando a ciertas funciones desarrolladas en lenguaje Cobol (por lo general orientada a archivos y aplicaciones), que se encargan del desarrollo de ciertas acciones a sistemas globales, y al mismo tiempo estaban llegando a una base de Datos DB2. En la Figura 4 muestro el diagrama de comunicación que tenían los módulos. DB2 es una base de datos relacional que proporciona funciones avanzadas de gestión de datos y analítica para las cargas de trabajo transaccionales.

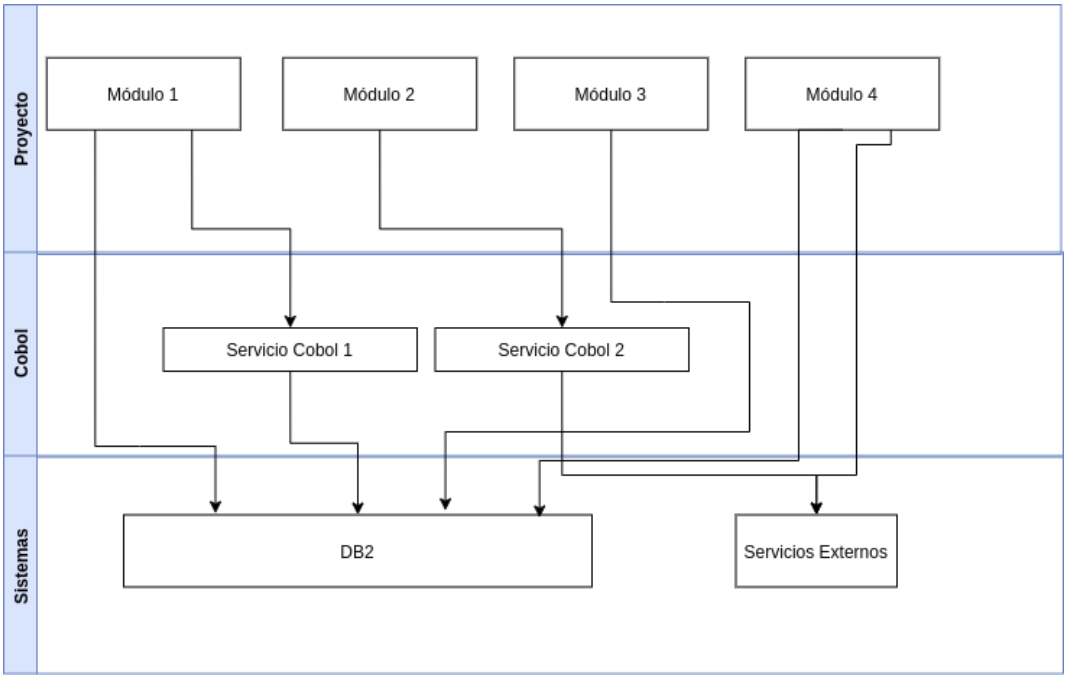

Figura 4. Diagrama de Comunicación de Sistema Anterior

Es importante mencionar que cada una tenía una manera diferente de administrar usuarios hablando de desarrollo y lógica en el código, por lo que, mientras en unas usaban sesiones dentro de la aplicación, en otras solo ocupaban ciertos métodos de una librería interna de la empresa, que ya no estaba actualizada, por lo que ya no tendría utilidad en el corto plazo y quedaría obsoleta.

También es importante añadir que la forma en la que se daban los permisos a los menús de cada aplicación estaban dentro de un archivo estático, en el que cada menú es verificado con un código dentro de la misma aplicación, por lo que cada que se tenía un menú agregado o sin uso en la aplicación, debían entrar a moverlo directamente en el código.

Asimismo, la forma de entrar a los menús estaba puesta por código dependiendo de los perfiles que se tenían, de modo que, en vez de llevarlos por perfiles, los llevaban por códigos.

Había diversas funciones en desuso, pero ya que estaban trabajadas como monolito, solo se llegaban a comentar las partes del código que accedían a ellas en los menús, para no romper el funcionamiento de las demás acciones.

Capítulo 2. Diseño de la solución

En este capítulo muestro una descripción de las arquitecturas propuestas para la solución del problema, así como el diseño final que se construyó.

2.1 Aplicación Monolítica vs. Microservicios

Como parte fundamental de la problemática presentada en este documento es necesario mencionar que se tenían diferentes propuestas de soluciones y después de un profundo análisis, pudimos elegir la más adecuada con base en las herramientas de trabajo con las que contábamos. La arquitectura que se utilizaba por la empresa era monolítica, la cual explicaré detalladamente más adelante.

De acuerdo con el enfoque tradicional para el desarrollo de aplicaciones basadas en aplicaciones monolíticas, estos sistemas se conforman de todas las partes de la aplicación que se pueden implementar, las cuales están contenidas dentro de esa única aplicación.

Como definición tenemos que el término *monolito* o *monolítico* se refiere a un estilo de arquitectura o a un patrón (o anti-patrón) de desarrollo de *software*. Cabe añadir que diferentes estilos de arquitectura o patrones de desarrollo de *software* se clasifican en diferentes tipos de vista (conjunto o categoría de cómo se visualiza una arquitectura de software) para su correcta implementación.

Una de las ventajas más importantes que poseen los sistemas de aplicaciones monolíticas es que el desarrollo implica un solo proyecto, de modo que se logra un solo despliegue y, por su misma naturaleza, es rápido de ejecutar (Bucchiarone, 2018, pp. 50-55). Debido a la simpleza de su estructura, el desarrollo de aplicaciones monolíticas suele ser menos costoso que sus alternativas; sin embargo, también tiene sus desventajas: mientras más compleja es la aplicación por la cantidad de información que maneja, así como por las métricas de desarrollo que utiliza, se

dificulta más solucionar con rapidez los problemas que se presentan y agregar funciones nuevas.

La asignación monolítica se refiere a que, una vez que el código fuente ha sido compilado y está listo para ser liberado, existe una única versión del empaquetado, el cual contiene todos los componentes necesarios para su ejecución.

También, dentro de la asignación monolítica, todos los componentes en ejecución tienen la misma versión del *software* que se ejecuta en un momento determinado, por lo que la asignación monolítica es independiente de si la estructura (a nivel código) del módulo es un monolito o no, debido a una de las dos posibles asignaciones siguientes:

- Despliegue de un único compilado de un único código fuente.
- Despliegue simultáneo de múltiples compilados de distintos códigos fuente.

Por otro lado, la asignación de memoria no monolítica implica desplegar diferentes compilados (con diferentes versiones de código o no), en diferentes nodos o equipos y en diferentes momentos, además, la afectación del reinicio del sistema sólo afecta al nodo donde se está realizando el despliegue.

Para resumir, encontramos que los módulos monolíticos se refieren a que todo el código de un sistema se encuentra en un único código base o código fuente que es compilado y produce un único artefacto o pieza de *software*. Además, el código fuente puede estar bien estructurado mediante clases y paquetes, y puede estar escrito implementando las mejores prácticas de desarrollo; sin embargo, no se encuentra dividido en distintos módulos para su mantenimiento, compilación, despliegue y ejecución.

Una vez explicadas las bases y características de la arquitectura monolítica, es tiempo de mencionar otro método o solución de diseño basado en microservicios, el cual se rige en la división de módulos del sistema. Estos tienen como ventaja el impulso del desarrollo y la capacidad de respuesta.

En un sentido estricto, el diseño de un módulo no monolítico mantiene el código fuente del sistema dividido o separado en múltiples módulos o librerías, los cuales pueden ser compilados, mantenidos, desplegados y ejecutados de manera separada.

A su vez, el diseño de un módulo no monolítico puede estar almacenado en diferentes códigos base y diferentes repositorios, lo que permite que puedan ser referenciados o modificados de forma independiente cuando sea necesario.

Comparando estas dos arquitecturas de diseño, la asignación monolítica se refiere a que todo el código fuente del módulo o sistema monolítico ha sido empaquetado y desplegado en el mismo momento, al mismo tiempo. A continuación, en las Figuras 5 y 6, podremos observar un ejemplo sobre la diferencia entre las dos formas de asignación de memoria:

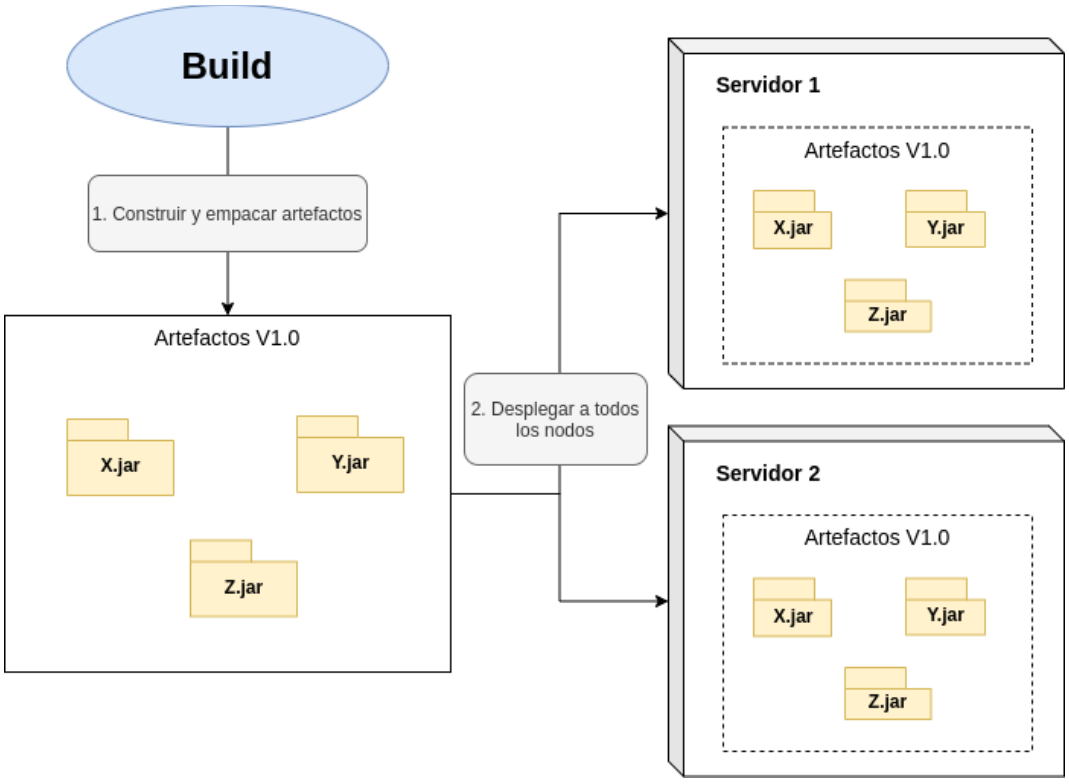

Figura 5. Diagrama de un proyecto Monolítico

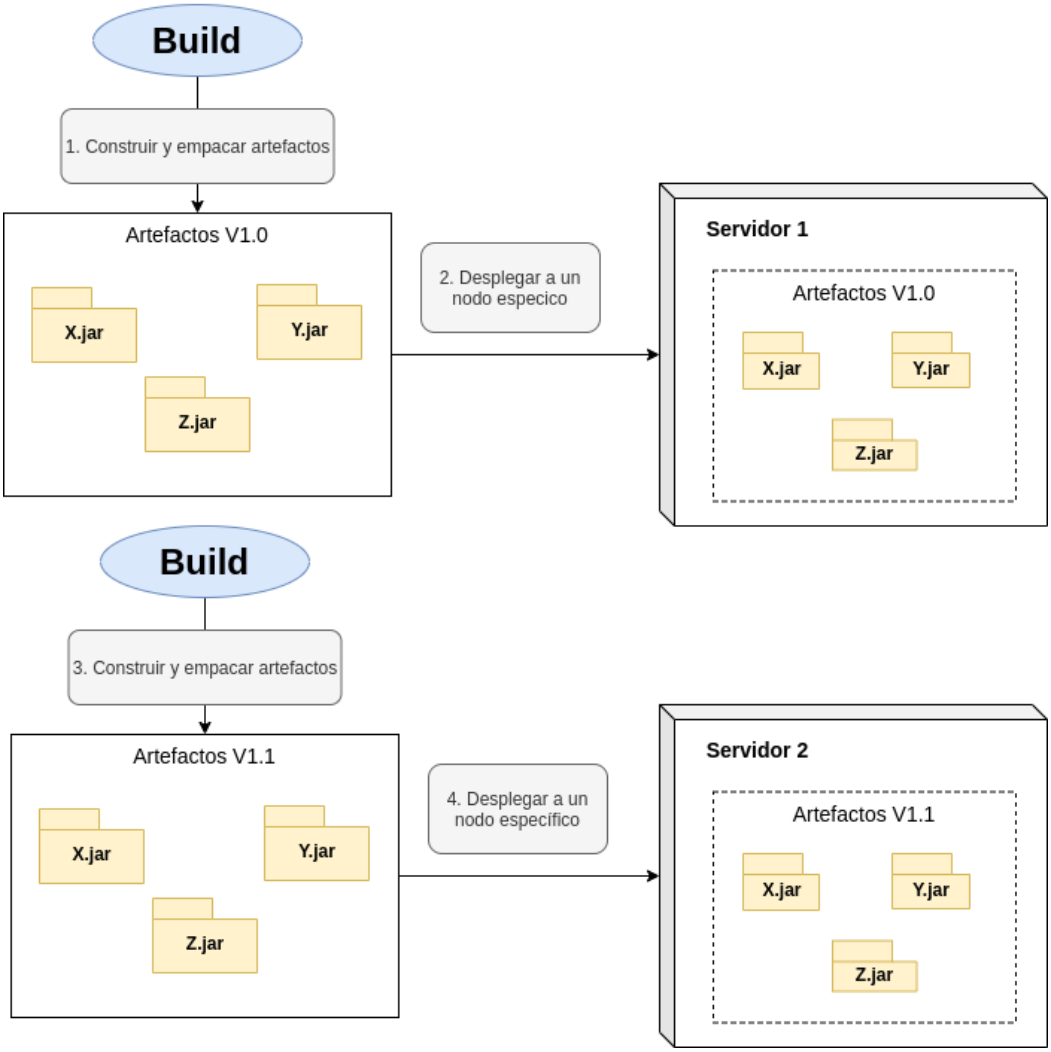

Figura 6. Diagrama de un proyecto con microservicios

Continuando con la arquitectura de microservicios (no monolítica), estos representan un estilo de arquitectura y un modo de programar un *software*. Con los microservicios, las aplicaciones se dividen en sus componentes más pequeños y son independientes entre sí.

A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, como menciona Levcovitz en su texto *Towards a technique for extracting microservices from monolithic enterprise systems* (2016), los microservicios son independientes y funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de estos elementos o procesos es un microservicio. Este enfoque privilegia el nivel de detalle, la sencillez y la capacidad de compartir un

proceso similar en varias aplicaciones. Es un componente fundamental para la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube.

El objetivo es distribuir un *software* de calidad con mayor rapidez. A pesar de que esto es posible con los microservicios, se deben considerar otras cuestiones. Dividir las aplicaciones en microservicios no es suficiente; por lo que es necesario administrarlos y coordinarlos, así como gestionar los datos que ellos crean y modifican. “Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Los servicios son independientes entre sí y cada uno debe implementar una funcionalidad de negocio individual” (Dragoni, 2017, 195).

De cierto modo, los microservicios son la evolución natural de las arquitecturas orientadas a servicios, pero hay ciertas diferencias entre los microservicios y estas arquitecturas. Éstas son algunas de las características que definen un microservicio:

- En una arquitectura de microservicios, los servicios son pequeños e independientes y están acoplados de forma flexible.
- Cada servicio es un código base independiente que puede administrarse por un equipo de desarrollo pequeño.
- Los servicios pueden implementarse de manera independiente. Un equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación.
- Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.
- Los servicios se comunican entre sí mediante API bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios.
- No es necesario que los servicios compartan la misma pila de tecnología, las bibliotecas o los marcos de trabajo.

Para la solución del sistema en cuestión, se optamos por la solución de los microservicios, ya que además de ayudar a la compilación individual de cada funcionalidad del sistema, también consideramos el uso de estos para ocuparlos en otros sistemas de la empresa en el futuro.

En este proyecto también involucramos el uso de Servicios REST, como Backend, que a continuación explico a detalle.

2.1.1 Servicios REST

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que dispone de una alta capacidad de procesamiento, pero también de mucha complejidad.

Los servicios SOAP, mejor conocidos como Servicios Web, son servicios que basan su comunicación bajo el protocolo SOAP (Protocolo de acceso de objetos simples), el cual se entiende como el protocolo estándar que define cómo es que dos objetos en diferentes procesos pueden comunicarse por medio del intercambio de datos XML (Blancarte, 2020). Por otro lado, REST es una tecnología más flexible que transporta datos por medio del protocolo HTTP; éste permite utilizar los diversos métodos que proporciona HTTP para comunicarse, como GET, POST, PUT, DELETE, PATCH y, a la vez, utiliza los códigos de respuesta nativos de HTTP (404, 200, 204, 409, etc.).

REST es tan flexible que permite transmitir prácticamente cualquier tipo de datos, ya que el tipo de datos está definido por el encabezado Content-Type, lo que nos permite mandar XML, JSON, binarios (imágenes, documentos), texto, etc.; lo que contrasta con SOAP que solo permite la transmisión de datos en formato XML (Upadhyaya, 2011).

A pesar de la variedad de tipos de datos que podemos mandar con REST, la mayoría de estos se transmite en JSON por un motivo muy importante: JSON es

interpretado de forma natural por JavaScript, lo que ha hecho que *frameworks* como Angular y React se aprovechen al máximo, pues pueden enviar peticiones directas al servidor por medio de AJAX y obtener los datos de una forma nativa. Los formularios de HTML pueden ser apuntados a los servicios REST sin ningún problema.

Debido a lo anterior y a que REST cuenta con mayores beneficios, decidimos utilizar este método, además de que JSON nos proporcionó un mejor rendimiento y calidad en el envío de datos. Las características de REST las describo a continuación:

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo cliente-caché-servidor sin estado, es decir, existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como “cacheables”, con el objetivo de que el cliente pueda ejecutar la misma respuesta para peticiones idénticas en un futuro. Sin embargo, que exista la posibilidad no significa que sea lo más recomendable.
- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado o, por ejemplo, para compartir su ubicación exacta con terceros.
- Interfaz uniforme: para la transferencia de datos en un sistema REST, éste aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

- Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
- Uso de hipermedios: hipermedia es un término acuñado por Ted Nelson en 1965 y se refiere a una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario pueda navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones para proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

De forma más detallada, JSON (por sus siglas JavaScript Object Notation - Notación de Objetos de JavaScript) “es un formato ligero de intercambio de datos. La principal característica es que es sencillo de leer y de escribir para los humanos, mientras que las máquinas hacen el trabajo de interpretación y procesamiento de forma más simple” (JSON, 2017).

Está basado en un subconjunto del lenguaje de programación JavaScript y es un formato de texto completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidas por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, entre otros. JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla *hash*, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Éstas son estructuras universales; todos los lenguajes de programación las soportan de una forma u otra virtualmente. Es razonable que un formato de

intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

Con base en lo anterior, decidimos que la tecnología REST es la mejor opción actualmente para trabajar con las diferentes tecnologías que se pueden manejar para este proyecto.

2.1.2 LDAP

Además de todo lo explicado en el apartado anterior, para el acceso al nuevo sistema buscamos una forma de recurrir a los usuarios utilizando los ya existentes en los sistemas del banco, y no tener que realizar un nuevo trámite de generación de usuario para el sistema, como se tenía que hacer anteriormente.

Esto se define con el término de "directorio activo", el cual utiliza Microsoft para referirse a su implementación de servicio de directorio en una red distribuida de computadores. Utiliza distintos protocolos, principalmente LDAP, DNS, DHCP y Kerberos.

De forma simplificada, se puede decir que es un servicio o función establecido en uno o varios servidores en donde se crean objetos tales como usuarios, equipos o grupos, con el objetivo de administrar los inicios de sesión en los equipos conectados a la red, así como para la administración de políticas en toda la red.

Su estructura jerárquica permite mantener una serie de objetos relacionados con componentes de una red, como usuarios, grupos de usuarios, permisos y asignación de recursos y políticas de acceso.

El directorio activo permite a los administradores establecer políticas a nivel de empresa, desplegar programas en diferentes ordenadores y aplicar actualizaciones críticas a una organización entera. Un directorio activo almacena información de una organización en una base de datos central, organizada y

accesible. Pueden encontrarse desde directorios con cientos de objetos para una red pequeña, hasta directorios con millones de objetos.

En este caso utilizamos LDAP, Protocolo Ligero de Acceso a Directorio por sus siglas en inglés. Se trata de un conjunto de protocolos de licencia abierta que se utilizan para acceder a la información que está almacenada de forma centralizada en una red. Este protocolo se usa a nivel de aplicación para acceder a los servicios de directorio remoto o directorio activo.

2.2 Estructura Base de Datos

Antes de ahondar en la estructura, es importante mencionar que la base de datos que elegimos fue de tipo relacional y fue Oracle 12C. Esta es la nueva versión que estaba manejando en la empresa, ya que anteriormente se tenía DB2, y sobre Oracle la más reciente era 11G.

> Una base de datos relacional es un tipo de base de datos que almacena y proporciona ingreso a puntos de datos conectados entre sí. Se basan en el modelo relacional: un modo intuitivo y directo de representar datos en tablas. (Oracle, 2019)

En una de estas bases, cada fila de la tabla es un registro con un ID único llamado “clave”. Las columnas de la tabla contienen los atributos de los datos y cada registro tiene normalmente un valor para cada atributo, lo que permite instaurar cómodamente las relaciones entre los puntos de datos.

Las bases de datos relacionales se usan para buscar inventarios, procesar transacciones de comercio electrónico y dirigir enormes cantidades de información crítica de los clientes, entre otras cosas. Se puede utilizar para cualquier necesidad de información en la que los puntos de datos se relacionan entre sí y deban gestionarse de forma segura, en función de ciertas reglas y de manera razonable.

En una base de datos de este tipo, se tiene más control sobre los datos, el manejo de estos, además de que nos ayuda a verificar que no haya duplicidad en los registros y favorece la normalización por ser más comprensible y aplicable.

Estás son algunas de las razones por las que fue la mejor decisión, además de que, por ser una migración, ya se tenían los datos que iban a ser utilizados.

La estructura de la base de datos la diseñamos a partir de la solución que elegimos anteriormente. Tomamos en cuenta toda la información que se requería para un inicio de sesión que abarcara los cuatro módulos y que tuviera todas las opciones de perfiles de cada uno.

Las tablas que construimos contenían los usuarios, módulos y perfiles. A continuación, en la Figura 7 muestro el modelo de base de datos que generamos.

En esta estructura, tomamos en cuenta la separación y las restricciones que debía de haber en la aplicación web de Administración de Usuarios y Perfiles. Empezando por la tabla de APP, la cual hace referencia a la tabla de aplicaciones, que en ese momento es la que se empleó para este proyecto. Esto se hizo con el fin de que esta aplicación se pudiera utilizar en otros proyectos dentro de la organización.

Después tenemos la tabla de módulos, los cuales eran creados dependiendo de la aplicación; en este proyecto tuvimos los 4 módulos registrados que íbamos a migrar más este administrador, que lo tomamos en cuenta como un nuevo módulo. Enseguida, tenemos la tabla de perfiles, la cual contenía todos los perfiles de todos los módulos de la aplicación.

Aparte de éstas, tenemos las tablas que hacen la relación entre ellas, en donde se encuentra la tabla de REL_MENU_OPC, que se encargaba de tener guardados todos los menús y submenús de la aplicación, en donde ya se utilizaba la relación con cada módulo y perfil. Para hacer relación con los perfiles, se tiene REL_MENU_OPC_PER, la cual tiene como llave foránea toda la referencia de módulo, perfil y *app*.

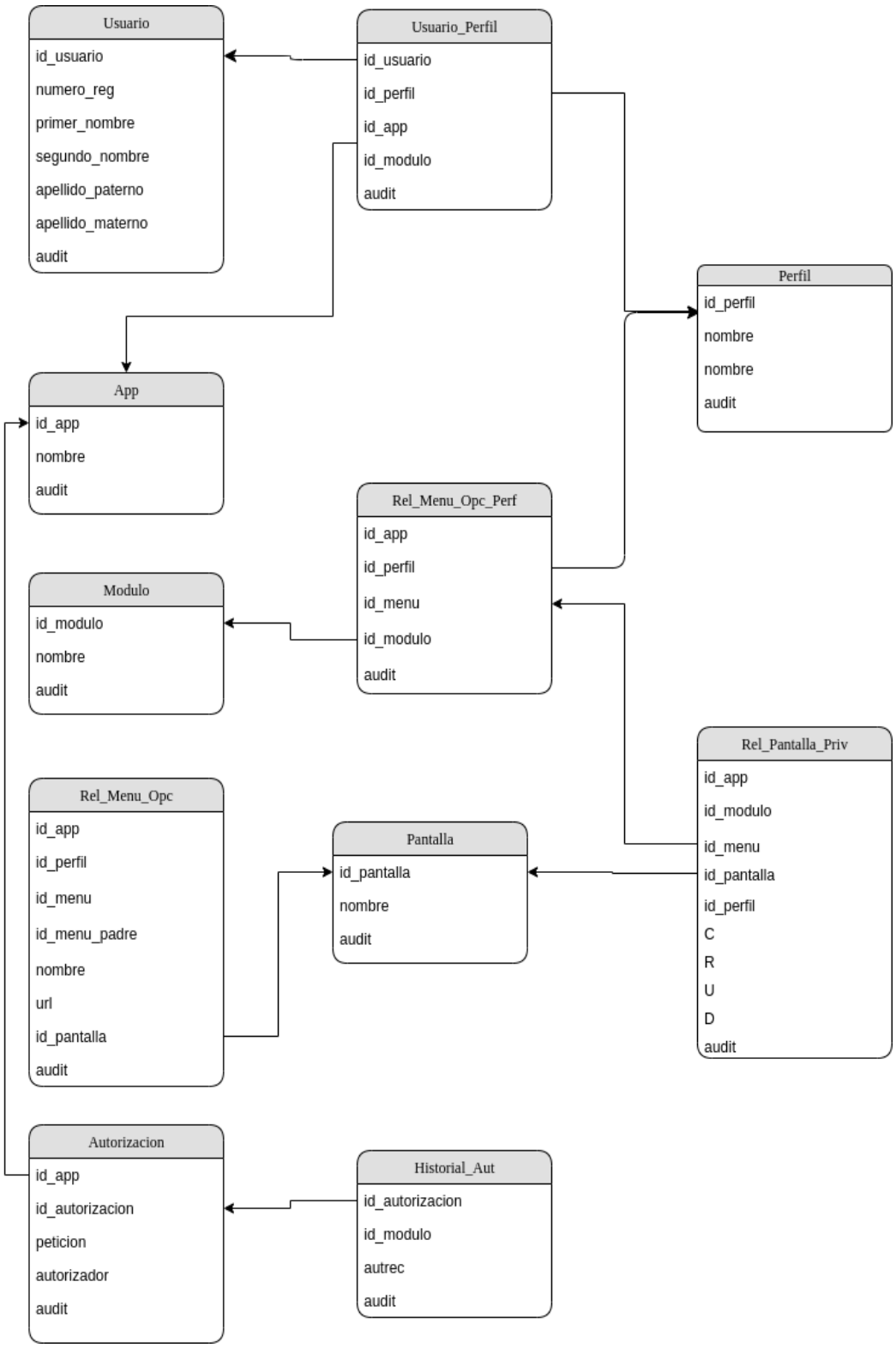

Figura 7. Diagrama de Base de Datos

Es importante mencionar la existencia de la tabla de permisos, ésta guardaba los permisos que se tenía en cada menú, referente a CRUD (Crear, Leer, Modificar y Eliminar) y a los permisos para autorizar, entro otros que existan dentro de cada módulo.

Finalmente, pero no menos importante, tenemos la tabla de usuarios, que tenía el registro de usuarios asociados a la aplicación utilizada. Y a su vez la tabla de USUARIO_PERFIL, la cual hacía relación entre usuarios y perfiles.

Todas las tablas contienen campos de auditoría para saber quién las creó, así como un campo para poder activar y desactivar registros, con la facilidad de realizar borrados lógicos.

Por otro lado, tenemos las tablas de autorización, en donde se guardaba la petición de las operaciones realizadas por un usuario, esto quiere decir que guardaba el método, el cuerpo, la url, entre otros datos, de la petición, para poder realizar la autorización dependiendo del perfil.

Además, se tiene una tabla de historial, donde se guardan las fechas, así como si son autorizadas o rechazadas las peticiones por el perfil determinado.

En la mayoría de las tablas utilizamos como llave primaria el conjunto de identificadores de las diferentes tablas, esto lo decidimos así, para que no hubiera ningún registro duplicado.

Esta base de datos la diseñamos pensando tanto en el usuario que iba a entrar a los módulos actuales del banco, como en el administrador que podía tener referencia de las acciones de los usuarios para poder autorizarlas. De esto hablo detalladamente más adelante.

2.3 Diseño de Interfaces

Para el diseño de las interfaces utilizamos Angular con SASS; y Angular Framework para aplicaciones web desarrollado en TypeScript, de código abierto, que se usa para crear y mantener aplicaciones web de una sola página.

Los módulos anteriores se trabajaron con JSP y Facelets (lenguaje de declaración de página que se utiliza para crear vistas por medio de plantillas de estilo HTML y árboles de componentes) para la parte de las interfaces. Es importante mencionar que la mayor parte del código ya estaba “parchado”, por lo que hacía muy difícil referenciar cada uno de los campos a su código de *backend*.

Como ya mencioné, al ser un proyecto monolítico se tenía la codificación del *backend* y la del *frontend* en el mismo proyecto. Además de que los módulos eran pesados, era muy complicado poder comprender cada una de las clases que conformaban el proyecto, ya que no se tenía un orden y documentación para relacionar las interfaces con su código.

Para este proyecto decidimos tener el desarrollo de *frontend* totalmente aislado de los microservicios, es decir, que no convivan con el *backend*. Anteriormente, al estar todo junto dentro de un mismo proyecto, se compilaba en un proyecto Maven y posteriormente se desplegaba como una aplicación en el servidor de WAS 9 (Websphere Application Server Versión 9).

En esta actualización de proyecto decidimos utilizar el servidor Was Liberty, para compilar cada uno de los microservicios y de las interfaces de forma independiente en diferentes instancias, continuando con el uso de Maven igualmente.

El diseño visual de las pantallas lo realizamos con varios esquemas que ya se han trabajado para otros sistemas más nuevos en el banco, manteniendo solo los colores y el formato permitido por el mismo.

En nuestro equipo, tuvimos un diseñador de experiencia de usuario que nos ayudó a mejorar la velocidad con que el usuario pudiera hacer las acciones. El objetivo era hacer el menor número de *clics* posibles para realizar una acción.

Al tener cuatro módulos diferentes que se habían estado modificando con el tiempo, encontramos que había acciones muy diferentes para funcionalidades similares, por lo que optamos por tomar la versión más sencilla para manejar al usuario y así crear las funcionalidades correspondientes en un ambiente más amigable. De este modo, de tener múltiples pantallas para realizar una acción, intentamos crear más visibilidad en cada una de las pantallas y así lograr menos número de *clics*.

La comunicación que el *front* tenía con el *backend* lo hicimos a través de los servicios REST, estos llaman a los microservicios de la capa de experiencia, la cual es una capa de la arquitectura que está diseñada para que la interfaz o un usuario en específico llegue a un microservicio. Hablaré más de esto más adelante.

Cada una de las pantallas o interfaces maquetadas tenía sus propias funciones, de modo que el objetivo era mantener la mejor experiencia para el usuario sin necesidad de saturarlo de información, así como mantenerlo en espera de una respuesta; es por eso que varios microservicios debían ser consumidos de manera asíncrona por detrás. Un ejemplo era el llenado dinámico de los *combobox*, pues anteriormente se necesitaba cambiar de página para que se realizara la acción, por lo que en esta nueva versión trabajamos para simplificar este proceso.

Dentro de las nuevas características y/o funcionalidades que le añadimos al sistema, el usuario debía ingresar su número de usuario y contraseña asignado, y así el sistema que hacía uso de LDAP validaba que las credenciales ingresadas fueran correctas y que el usuario estuviera registrado en el sistema. Una vez que se validaba la identidad del usuario, éste ya podía ingresar al portal. Dentro de éste, se debía seleccionar el módulo de “administración de usuarios y perfiles” dependiendo del perfil con el que contara y una vez hecho esto, se podía visualizar lo siguiente:

Para un usuario con perfil **operador:**

En la pantalla de inicio tenía habilitados los siguientes menús: “estado de autorizaciones” y “catálogos” (usuarios y perfiles). El operador podía realizar todas las operaciones CRUD (Crear, leer, modificar y hacer borrados lógicos) de perfiles y usuarios, es decir, podía dar de alta, desactivar, reactivar, modificar y consultar todos los perfiles (incluyendo los propios del módulo), así como usuarios.

Cabe resaltar que las operaciones no eran efectuadas al momento, pues al ser un sistema con dualidad se regía por un perfil que operaba, para que éste envíe una solicitud en cada operación. Posteriormente, la solicitud era resuelta por otro perfil, con la finalidad de ser autorizada o rechazada. También podía consultar el estado de las solicitudes que había efectuado.

Acciones definidas para un **operador:**

- Consultar catálogos de usuarios
- Buscar por número de registro
- Buscar por nombre de usuario
- Buscar por filtrador (usuarios activados/ usuarios desactivados)
- Puede ver los perfiles asociados al usuario
- Agregar un perfil al usuario
- Editar un perfil asociado al usuario
- Eliminar un perfil asociado al usuario
- Modificar un usuario después de buscar las actualizaciones que éste tiene
- Desactivar o reactivar el usuario
- Dar de alta un usuario con un perfil específico
- Modificar los privilegios de un perfil
- Dar de alta un perfil o algún módulo
- Puede consultar catálogos de perfiles
- Buscar por módulos
- Buscar por nombre de perfil
- Buscar con filtrado activado/desactivado para conocer el estado de los usuarios y perfiles

- Al consultar puede ver los usuarios desde el catálogo de perfiles
- Desactivar o reactivar un perfil
- Modificar los privilegios de un perfil
- Dar de alta un perfil a algún módulo

Para usuario con perfil **autorizador**:

En la pantalla de inicio tenía habilitado el menú de “estado de autorizaciones” en donde autorizaba o rechazaba las solicitudes que cualquier operador enviara. Además de estas acciones un autorizador podía consultar las solicitudes que se habían realizado por los demás usuarios.

Las funcionalidades principales de un autorizador eran autorizar o rechazar las solicitudes que algún operador había enviado, por lo que el perfil de autorizador era el encargado de otorgar o denegar los permisos de las solicitudes, para que se efectuaran los cambios correspondientes realizados por el perfil de operador (todas las operaciones CRUD realizadas por el perfil operador se veían reflejadas en el sistema hasta que el perfil autorizador aprobara las peticiones).

En el administrado de perfiles y usuarios podían existir usuarios con perfil de **operador** y usuarios con perfil de **autorizador** como fueran necesarios. Sin embargo, la utilidad de este sistema ayudó a crear perfiles según lo que el usuario requiriera, por lo que podía existir un usuario que pudiera crear perfiles y al mismo tiempo autorizar las solicitudes de creación, modificación o inactivación de un usuario.

2.4 Pila Tecnológica

A continuación, resumo las diferentes herramientas tecnológicas utilizadas para la elaboración de este proyecto:

Para la presentación o interfaz de usuario: éstas son las herramientas utilizadas para poder realizar la vista que tenía el usuario, y la capa que se encargaba de hacer el llamado a los servicios en *backend.*

- Angular 5 Framework: se utilizó para el desarrollo web, para el diseño de las páginas y su funcionamiento, además de que era el encargado de hacer las llamadas a los servicios cuando se requiera.
- Preprocesador de estilo SAAS: se utilizó para realizar el diseño gráfico de las pantallas y el formato en donde normalmente se utiliza CSS, con éste se pueden añadir reglas y síntesis.
- Angular: MultiCanal.

Para el robustecimiento de seguridad: con estas herramientas se trabajó toda la capa de seguridad, desde la parte de usuario, hasta la parte de servicios.

- Capacidad de identificación y autenticación:
    - Angular se utilizó para adquirir los datos de entrada de un usuario (usuario y contraseña) para poder enviarlos a verificar con ayuda de los servicios.
- Identificación y administración de acceso:
    - ADAM (Directorio Activo LDAP): se utilizó para obtener la información de los usuarios registrados dentro de la empresa y así utilizar la misma información.
    -
    - E2E Trust Token: se utilizó para crear una codificación sobre la información del usuario que le permitía tener acceso a los servicios y solo sea por un tiempo determinado.
- Capacidad de confidencialidad y cumplimiento:
    - Angular: esta herramienta se utilizó para poder mostrar los errores que se tenían dentro de la interfaz, además de mostrar los errores o acciones realizadas desde los servicios.

- Log4j: esta herramienta se utilizó para hacer el *log* de todos los procedimientos y los errores, para así poder tener un control de cualquier servicio o ejecución que esté fallando.

Para la capacidad de procesamiento y ejecución de lógica: estas herramientas se utilizaron para realizar el procesamiento de todo el código.

- Tiempo de procesamiento: REST se utilizó para la programación de los servicios, siendo el responsable del tiempo que tardan en ejecutarse los servicios.
- Capacidad de particionamiento en ejecución:
  - Spring Batch: se utilizó para lanzar acciones de manera programada y que no requerían ningún tipo de intervención humana.
  - Liberty: se utilizó WAS Liberty para la ejecución de los servicios en diferentes instancias del servidor, dando como resultado la separación en microservicios.

Para la capacidad de administración e integración: estas herramientas fueron utilizadas para el control y la unión de los servicios.

- Capacidad de procesamiento de datos:
  - JDBC: utilizado para la conexión con las respectivas bases de datos, en este caso Oracle.
  - Hibernate: encargado del procesamiento de datos para entrada y salida más sencilla.
  - Spring Data: utilizado para la creación de repositorios para la integración de las entidades con las tablas en la base de datos.
  - JPA: encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB).
- Capacidad de procesamiento de almacenamiento:
  - Angular: se utilizó para recoger los datos y enviarlos al *backend*.

    - Hibernate: se encargó de procesar los datos de forma sencilla con ayuda de las entidades para enviar y recibir información de la base de datos.
    - JPA: encargado de la conversión de datos Java a instrucciones en la base de datos.
    - Oracle: ésta es la base de datos encargada del almacenamiento de la información.
- Capacidad de integración y transporte de datos:
    - Swagger: con esta herramienta se realizaron los contratos de los microservicios para tener claro cuál era la petición de entrada y salida. También ayudó para las pruebas y para tener claro qué se iba a enviar y qué se debía recibir.
    - API Gateway: con esta herramienta se realizó el envío de la información al microservicio correcto.
    - Liberty: servidor que se utilizó para guardar los microservicios en instancias como archivo WAR para poder ejecutarlos internamente.

Para el despliegue de servicios.

- Versionamiento: GitHub fue utilizado para realizar cambios en el código y al mismo tiempo realizar cambios de versiones en el mismo.
- Despliegue: Jenkins fue utilizado como servidor automatizador para realizar el despliegue de los cambios realizados en Git.

2.5 Diseño de Microservicios

La solución para el proyecto consistió en la construcción de diferentes microservicios en donde se pudieran conjuntar funcionalidades similares.

El primer paso que realizamos fue el análisis del sistema anterior, sin embargo, por cuestión de conflictos en el acceso, lo tuvimos que realizar a partir de maquetas de

las pantallas, las cuales fueron proporcionadas por el cliente. De este modo reunimos las pantallas y funciones que implicaban la administración de usuarios y perfiles y las homologamos para realizar una gestión igual para todos los módulos.

En esta parte, me di cuenta que es importante reunir todos los requerimientos del cliente y verificar con alguna persona que tenga el conocimiento de negocio que no haya ninguna validación o paso por detrás que no se vea a simple vista. Esto nos ayudó a tener una homologación correcta, en donde se pudieran crear tanto perfiles como usuarios, sin afectar ninguna función de cada sistema.

En cuanto a la arquitectura de los microservicios, el diseño y estructura de paquetes lo realizamos para que quedaran divididos en tres capas: *controller, service y persistence*. En la Figura 8 muestro la comunicación de esta arquitectura.

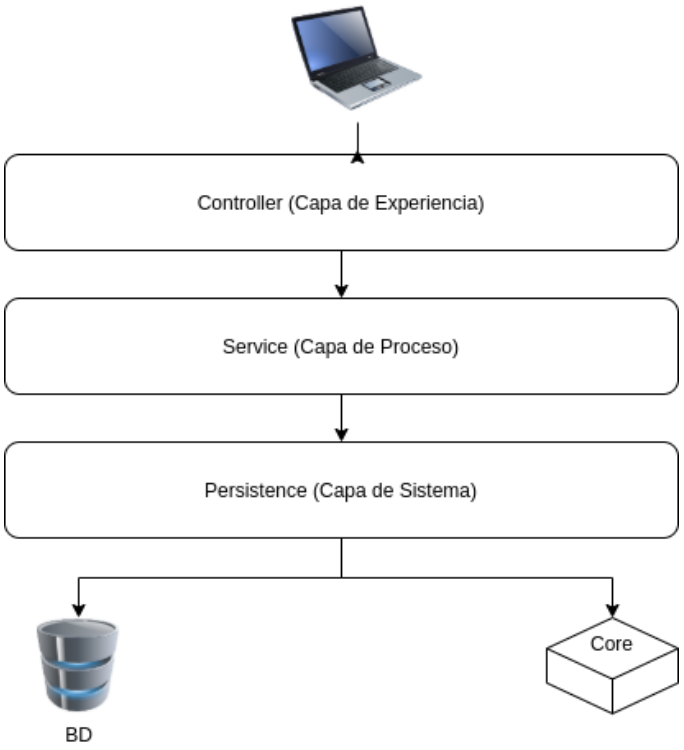

Figura 8. Diagrama de Arquitectura

Esta arquitectura se basó en una arquitectura llamada API Led Connectivity (conectividad basada en API), la cual “es una forma metódica de conectar datos a aplicaciones a través de API reutilizables y útiles. Estas API están desarrolladas para desempeñar un papel específico: desbloquear datos de sistemas, componer datos en procesos o brindar una experiencia” (Pearlman, 2019).

Al principio, cuando decidimos hacer esta arquitectura, no entendía su utilidad, sentía que solo estábamos aumentando el número de microservicios, realizando los mismos procesos. Sin embargo, conforme empezamos a realizar el desarrollo, encontré que, teniendo servicios con funciones específicas, nos ayudaba a crear microservicios mas sencillos, fáciles de entender, y además creaba menos dudas de comunicación entre que servicio debía realizar qué acción.

Las capas se dividieron en la capa de usuario o *controller*, en este caso, la capa de proceso o servicio y la capa de sistema o persistencia.

La capa *controller* era la capa a la que se llama desde el *frontend.* Esta capa recibía las peticiones del usuario y puesto que era el punto de acceso de los clientes, realizaba las validaciones de seguridad antes de enviarla a la capa de servicio. Bajo esta capa se exponían cada uno de los recursos que los clientes (llámese cliente al *frontend*) consumían para poder ejecutar el servicio.

La capa de servicio recibía las peticiones del *controller,* y manejaba toda la lógica de la información, aquí es donde se implementó cada una de las reglas del negocio. Una vez que se procesaba la información bajo la definición de requerimientos, ésta era enviada a la capa de persistencia. Tras la respuesta de la capa de persistencia, los datos eran acomodados bajo el estándar de JSON y se retornaba la respuesta a la capa inmediata superior.

La capa de persistencia recibía las peticiones de la capa de servicio y hacía la petición a la base de datos con la construcción de los *queries* necesarios. Una vez que se obtenía la respuesta de la base de datos, la información era devuelta tal cual los recibía, mandándolos a la capa de servicio.

Los servicios los dividimos de acuerdo con las funcionalidades que se identificaron. Las principales fueron: usuarios, módulos, perfiles, permisos, menús, autorizaciones y *Login*. Estos servicios los describo con mayor detalle a continuación:

- LDAP: Este servicio era el encargado de hacer la validación de usuario en el directorio activo que manejaba la empresa. Validaba que el usuario existiera y que no se encontrara bloqueado por algún motivo.
- Token: Este servicio era el encargado de generar el token de seguridad (clave de 6 caracteres o más de forma aleatoria e irreemplazable encriptada) con los perfiles del usuario y el tiempo válido para estar dentro de la sesión. Se utilizaba justo después de la validación del servicio anterior y cada que se hacía alguna petición al *backend* para verificar que seguía estando activa la sesión. Este servicio solo permitía una sesión por usuario, por lo que, al intentar ingresar en otra computadora, le rebotaba la autorización al sistema.
- Menús: Se encargaba del armado del menú que se le mostraba al usuario dependiendo del módulo y el perfil que tuviera. Como se ha mencionado, había 5 módulos y dentro de estos existían diferentes perfiles, por lo que dependiendo a dónde tuviera acceso el usuario, se armaba un JSON de salida con menús y submenús. Este servicio también proveía el llenado para agregar y modificar perfiles.
- Perfiles: Se encargaba de la creación, modificación, activación y desactivación de los perfiles; además de la consulta de los perfiles por usuario, regresando los datos de todos los perfiles con los usuarios que estuvieran asociados a ellos.
- Usuarios: Este servicio se encargaba de la creación, modificación, activación y desactivación de los usuarios, así como la asociación de un usuario a un perfil y la desactivación del mismo. También proveía la consulta de usuarios por perfil, regresando los datos de los usuarios y los perfiles a los que estaba asociado cada módulo.
- Módulos: Este servicio se encargaba de las consultas de los perfiles de cada módulo.
- Autorizaciones: Este servicio se encargaba de todas las autorizaciones que se generaban al querer hacer algún cambio en la base de datos: POST, PUT, DELETE. Éste recibía la petición y dependiendo del perfil del usuario, creaba una petición de autorización o conducía al flujo para que estas acciones fueran realizadas.

- Permisos: Este servicio se encargaba de la consulta de permisos dependiendo de un menú y un perfil, esto quiere decir que si un usuario tenía acceso a un menú, pero en éste solo podía ver l os datos, no tendría las acciones de editar, crear, ni eliminar ninguno de estos datos.
- Validaciones: Este servicio era el encargado de validar acciones específicas, como que no existieran dos perfiles que se llamaran igual en el mismo módulo, por mencionar un ejemplo.

Antes de empezar a describir detalladamente estos microservicios, quisiera añadir que es importante realizar constantes pruebas de cada uno de los *endpoints*, verificando capa por capa, para que, a la hora de realizar la prueba desde la capa superior, no haya errores donde no podamos hallar en cuál capa estuvo el error. Esto puede realizarse con herramientas como Postman y SoapUI, que son fáciles de utilizar y podemos probar servicios tanto SOAP como REST.

2.5.1 Servicio LDAP

La pantalla cargada de *Login* mostraba los campos para usuario y contraseña una vez que el usuario hubiera dado clic en “entrar”. El sistema se redirigía hacia el servicio de LDAP, donde verificaba que el usuario ingresado y la contraseña fueran correctos, esta respuesta regresaba al *front*, en donde, dependiendo de la respuesta, mandaba un mensaje de error en el caso de que los datos fueran incorrectos o si el usuario estaba bloqueado.

2.5.2 Servicio de Token

Si la verificación era exitosa, el sistema mandaba a llamar al servicio de *token,* el cual verificaba que el usuario existiera en la base de datos del sistema y buscaba los perfiles que tenía. Al igual que el servicio anterior, si no era exitoso, mandaba un error al *front*, pero si era exitoso, generaba un *token* con la fecha del sistema, el usuario, y una lista de perfiles que debía tener. Esto se realizaba usando Spring Security.

Una vez que se obtenía el *token*, éste era utilizado en todos los demás servicios para verificar que, mientras existieran peticiones hacia los demás servicios, la sesión

siempre permaneciera activa. Posteriormente, el sistema mandaba llamar al servicio de menú, que se encargaba de crear el menú con las opciones que tuviera el usuario dependiendo del perfil activo obtenido con el servicio de *token*. La estructura del JSON de respuesta era similar a la mostrada en la Figura 9:


```
{
    "regNumber":"user1",
    "modules":[
        {
            "idModule":1,
            "nameModule":"Access Management",
            "idProfile":1,
            "nameProfile":"Operador",
            "menus":[
                {
                    "idMenu": 1,
                    "name":"Consultas",
                    "submenus":[
                        {
                            "idMenu":2,
                            "name":"Usuarios",
                            "submenus":null
                        },
                        {
                            "idMenu":3,
                            "name":"Perfiles",
                            "submenus":null
                        }
                    ]

                },
                {
                    "idMenu": 4,
                    "name":"Estatus Autorizaciones",
                    "submenus":null
                }
            ]
        }

    ]
}
```


Figura 9. Ejemplo JSON

Con esta estructura, los menús dinámicos en la pantalla principal se podían formar sabiendo cuál era su menú padre. Toda la lógica se realizó en la capa de servicio, en donde obteniendo todos los datos con una consulta en la base de datos, se iban añadiendo los submenús con los datos correspondientes de sus padres.

Una vez obtenido el JSON anterior, lo que sucedía en el sistema era el ingreso a los módulos. En pantalla se mostraban como opciones los módulos a los que el usuario tenía acceso y una vez oprimiendo el botón indicado, se formaba el menú con las opciones de este módulo. En la Figura 10 muestro el diagrama de acceso al módulo.

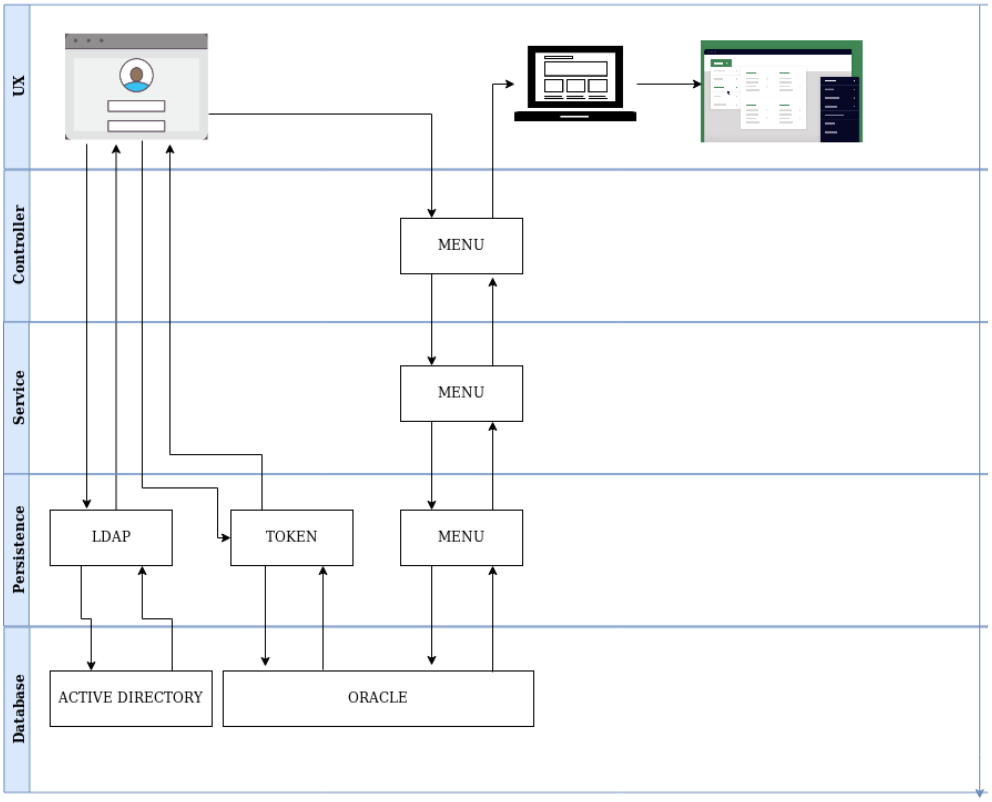

Figura 10. Diagrama de Acceso al Módulo

Cuando un usuario seleccionaba alguna opción de un menú o submenú que redirija a otra pantalla, se mandaba una petición por medio del servicio de permisos con el *id* del menú, con el cual se obtenían todas las acciones que un usuario era capaz de realizar dependiendo de su perfil, por lo que se obtenía un JSON similar al de la Figura 11.

Esto es lo que permite que la parte de *front* activara los botones que dan el permiso a las acciones indicadas, las cuales generalmente pertenecían al CRUD, para que un usuario que solo tuviera permiso de consultar los datos no pudiera modificar, crear y eliminar un registro. A continuación, muestro el diagrama de permisos en la Figura 12.

```
{
    "idMenu": 145
    "name": "Usuarios"
    "permissions":[
        {
            "idPermission":1
            "value": "Crear"
            "active": true
        },
            {
        "idPermission":1
        "value": "Modificación"
        "active": true
        },
            {
        "idPermission":1
        "value": "Eliminación"
        "active": true
        },
            {
        "idPermission":1
        "value": "Consulta"
        "active": true
        }
    ]
}
```

Figura 11. Ejemplo JSON para Permisos

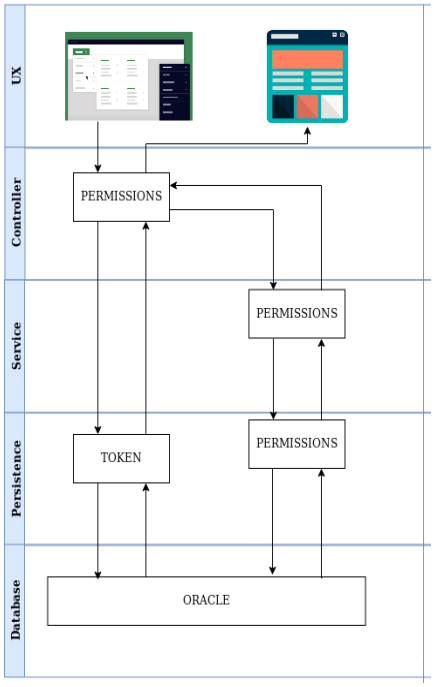

Figura 12. Diagrama de Permisos

2.5.3 Servicio de Usuarios

Cuando se deseaba ver los usuarios, se realizaba una consulta al servicio de usuarios, el cual regresaba toda la información existente y la forma en la que se mostraba en la pantalla. A modo de búsqueda, existía un campo de texto o un *textbox*,

técnicamente hablando, en el cual se podía escribir el nombre o registro de un usuario y conforme se escribía con un autocompletado, se iban mostrando las diferentes opciones de registros que se encontraban y que tenían relación con las letras o palabras escritas. En esta pantalla también se mostraban ciertos filtros como radio *buttons*, para buscar usuarios activos o inactivos.

Al seleccionar alguno, se pintaba en la pantalla una tabla mostrando los perfiles de módulos de un usuario, así como la opción de modificar y eliminar este usuario, o bien, modificar la relación que éste tenía con un perfil.

Para eliminar y modificar un usuario, se seleccionaba el botón indicado y éste mandaba llamar de nuevo al servicio de usuarios (a su respectivo método), el cual hacía el cambio en la base datos si era una modificación, o cambiaba el campo de activo a falso si era una eliminación, ya que solo se manejaban eliminaciones lógicas por cuestiones de seguridad e integridad de la demás información que persistía en la base de datos.

Para la creación de un usuario, el servicio te llevaba a una pantalla donde se encontraba un formulario que pedía el registro de empleado del usuario y automáticamente se llenaban los campos de nombre y apellidos, ya que el sistema mandaba llamar al servicio de LDAP, donde consultaba que fuera un registro activo y esa misma petición le entregaba los datos del usuario. Después de esto, en el formulario también se elegía un módulo y un perfil correspondiente a este módulo.

Para estas últimas peticiones (CUD) se necesitaba una autorización, por lo que antes de llegar a hacer cambios en la base de datos, se llamaba desde el servicio de usuarios en la capa de *controller* al servicio de autorizaciones, el cual generaba una petición para guardar estos datos en la base de datos y en las tablas de autorizaciones respectivas, las cuales esperaban un llamado del autorizador para proseguir con la ejecución de las instrucciones correspondientes.

Cuando se quería modificar la relación de un perfil con un usuario, se da *clic* en el botón a lado del perfil, éste mandaba a otra pantalla, en la cual aparecía un formulario

donde, a partir del módulo respectivo del perfil, dejaba un *combobox* con los perfiles de este módulo, éste se llenaba llamando al servicio de módulos y finalmente regresaba los módulos con sus perfiles en un formato JSON.

2.5.4 Servicio de Perfiles

Al seleccionar el cambio de perfil, éste generaba una petición que se guardaba en autorizaciones, como muestro en la Figura 13, y una vez aceptada es llevada al servicio de usuarios donde se hacía una modificación de perfil por usuario.

Al crear una asociación de perfil a un usuario, se daba clic en el botón indicado y se mostraba una pantalla similar a la anterior, dejando seleccionar el módulo en el respectivo *combobox,* que llamaba al mismo servicio de la acción anterior, como muestro en la Figura 14.

Cuando se quería crear un perfil, se mostraba una pantalla similar a la de búsqueda de usuario, solo que se podían buscar los perfiles que se consultan desde el servicio de perfiles. En la tabla de resultado se mostraban los usuarios asociados al perfil seleccionado. De la misma forma, se mostraban los filtros para poder ver solo los perfiles activos o inactivos.

Para eliminar y modificar, se debía dirigir a los servicios correspondientes en el servicio de perfiles, el cual se encargaba de llegar a la base de datos y hacer los cambios específicos solicitados. Cabe mencionar que al igual que todos los procesos, estos llegaban primero al servicio de autorizaciones y cuando se realizaba la opción de autorizar desde el perfil de autorizador, se proseguia con las acciones mencionadas anteriormente.

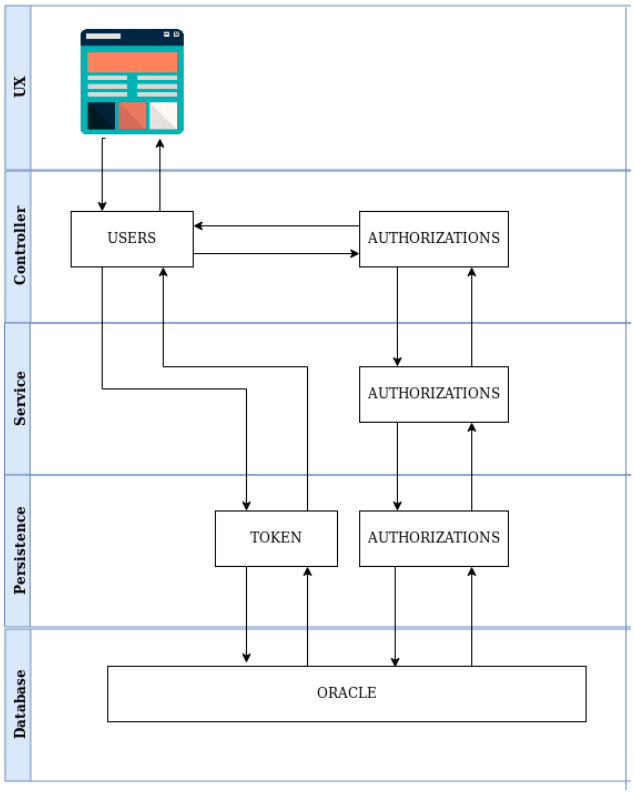

Figura 13. Diagrama de Autorizador

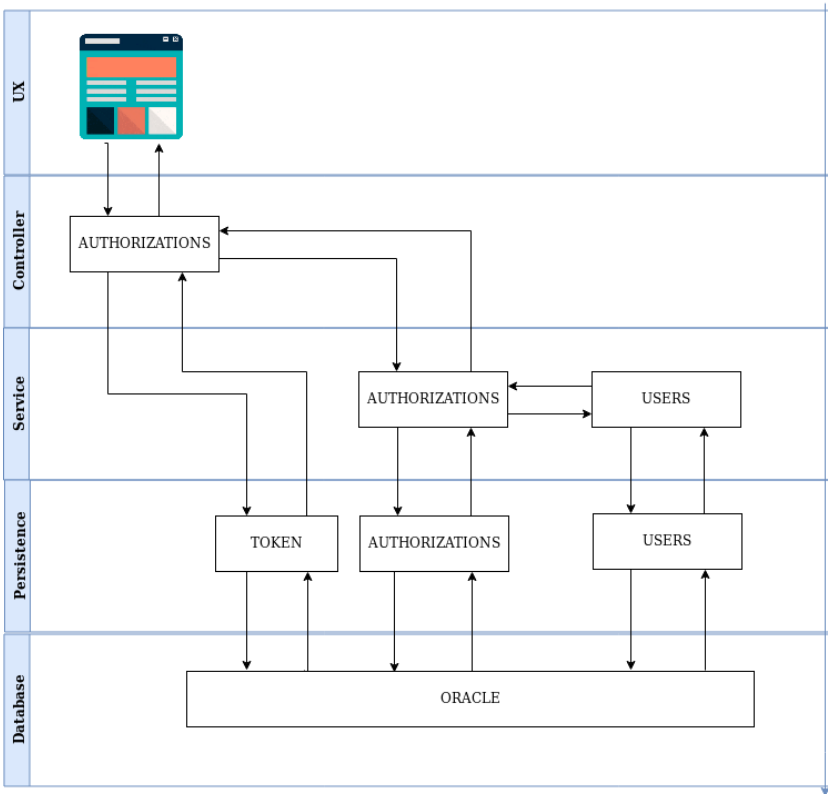

Figura 14. Diagrama de Perfil

Para crear se hacía un llamado al servicio de menús, ya que éste proveía toda la información de los módulos, los menús, y sus permisos correspondientes. Esto con el fin de darle al usuario una pantalla con la cual pudiera observar todas las opciones de menú y pudiera dar los permisos respectivos de CRUD de todas las opciones.

Es importante mencionar que existían diversas validaciones, desde la parte de negocio en donde un perfil con la opción de crear, modificar o eliminar en específico, no podía autorizar esa misma opción. Después de que el usuario eligiera las opciones que deseaba para el perfil, ésta mandaba una petición, que llegaba a autorizaciones para que posteriormente, con el visto bueno del autorizador, se generara la llamada al servicio de perfiles, como muestro en la Figura 15.

Cuando se seleccionaba a un usuario, permitía editar, al igual que en la parte de usuarios, el perfil de este.

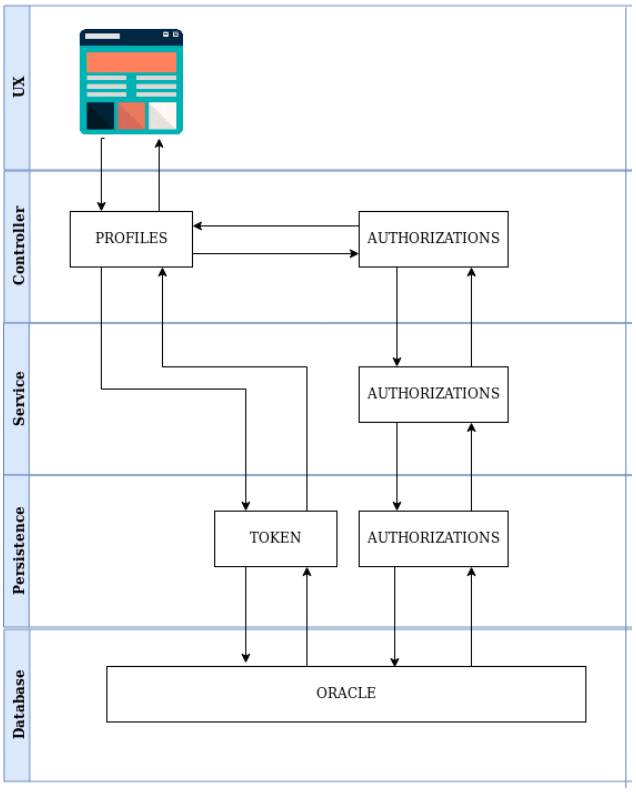

Figura 15. Diagrama de Autorización

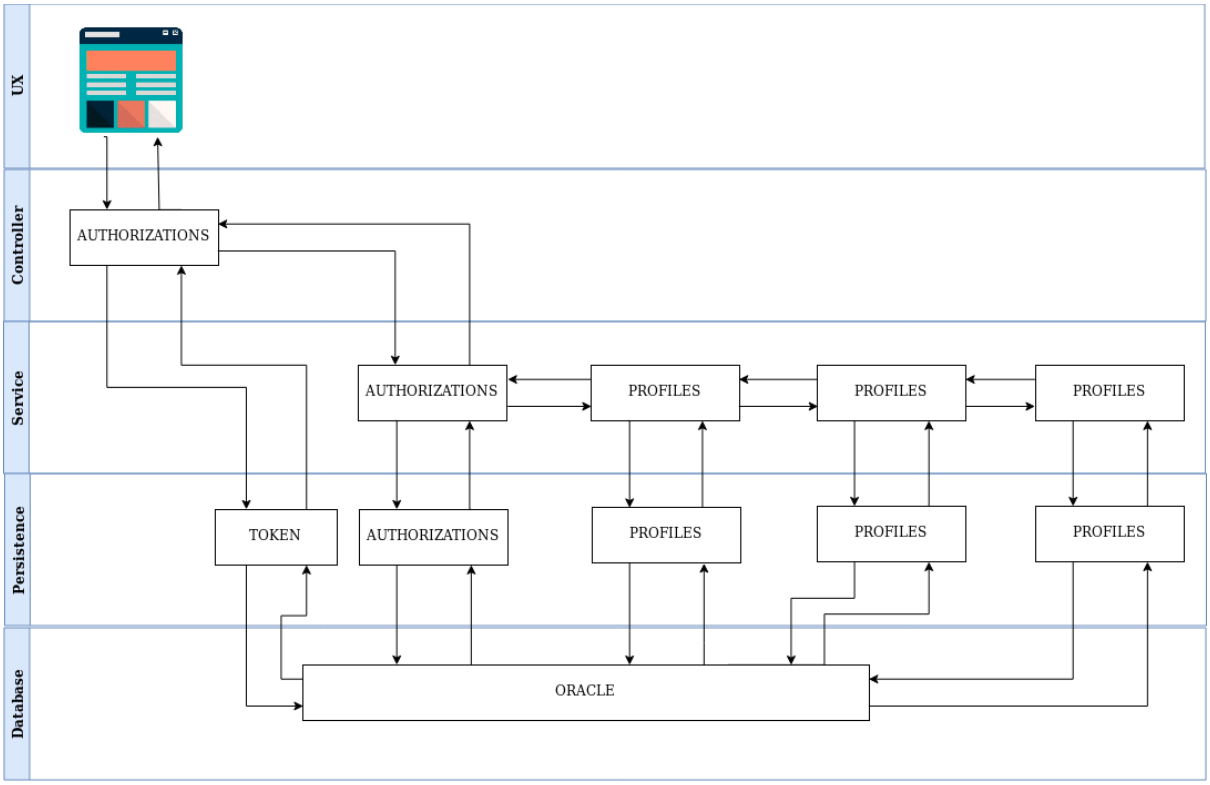

Figura 16. Diagrama de Autorización de Usuarios

Para autorizar las acciones, el usuario debía ingresar con perfil de autorizador de Administración de Usuarios y Perfiles, éste hacía un llamado al servicio de *token* para saber si seguía activa su sesión y entonces hacía el llamado para mostrar las solicitudes pendientes y el historial. En la Figura 16 muestro el diagrama de autorización de usuarios.

Dependiendo de si elija autorizar o rechazar, debía hacer el llamado al servicio de autorizaciones, el cual llamaba a los respectivos servicios de servicio y persistencia, según lo que se requiriera.

Si el usuario elegía autorizar, la petición llegaba a la base de datos y obtenía la solicitud pendiente, haciendo que ésta pudiera seguir su camino a su respectivo flujo. Si el usuario elegía rechazar, la petición se ponía en estado de rechazada en la base de datos.

Todas estas peticiones se seguían mostrando en pantalla, pero una vez autorizadas o rechazadas, solo mostraba su estado para mostrar el historial.

Todos los autorizadores eran capaces de ver todas las solicitudes, no era posible filtrar las peticiones con ningún método.

También era posible mostrar el estatus de autorizaciones con el método de historial a cada operador, sin embargo, por esto mismo no era posible filtrar por usuario.

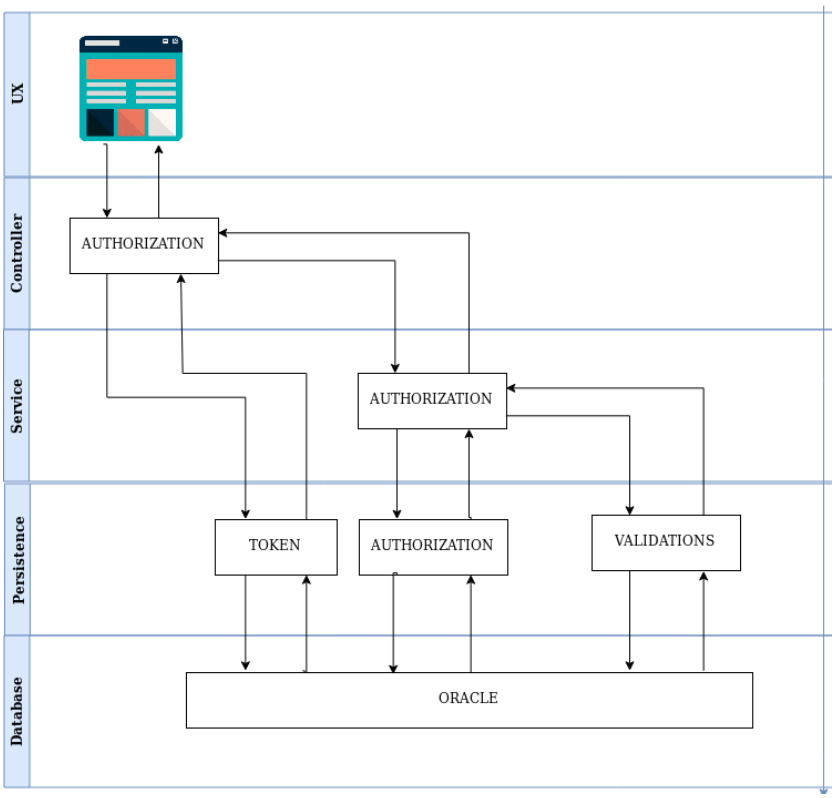

Figura 17. Diagrama de Validaciones

El servicio de validaciones era llamado desde el servicio de autorizaciones para validar antes de generar algún cambio. Éste lo muestro en la Figura 17.

Cuando se cerraba la sesión se llamaba al servicio de *token,* en el que se eliminaba el registro físicamente.

Una vez mencionado lo anterior, a continuación, enlisto en la Tabla 1 los diferentes servicios que se expusieron a través de métodos HTTP y haciendo uso de REST conjuntamente.

Tabla 1. Operaciones de los microservicios

Servicio	**Funciones**	**Método**
Perfiles	getProfile	GET
	createProfile	POST
	modifyProfile	PUT
	deleteProfile	DELETE
	getProfileUsers	GET
Menú	getMenu	GET
	getProfileOptions	GET
	getAllProfiles	GET
Permisos	getPermissions	GET
Usuarios	getUsers	GET
	createUser	POST
	modifyUser	PUT
	deleteUser	DELETE
	createUserProfile	POST
	deleteUserProfile	DELETE
	modifyUserProfile	PUT
	getUserProfiles	GET
Módulos	getModuleProfiles	GET

Servicio	**Funciones**	**Método**
LDAP	verifyUser	GET
	getUser	GET
Token	createToken	POST
	verifyToken	GET
	deleteToken	DELETE
Autorizaciones	createPetition	POST
	Authorize	PUT
	Reject	PUT
	getPetititons	GET
Validaciones	validateProfile	GET
	validateUser	GET

Capítulo 3. Resultados

En este capítulo explico los resultados de las pruebas del proyecto.

3.1 Pruebas del sistema

Para el análisis de los casos, nosotros, como equipo de desarrollo, recorrimos todo el flujo que se tenía anteriormente en los sistemas y lo comparamos con el actual, para que cada una de las validaciones, aunque fueran diferentes, se unificaran en el módulo de administración de usuarios y perfiles.

Es importante mencionar que tuvimos tres ambientes para estas validaciones: ambiente de desarrollo, UAT (ambiente pre-productivo), y producción.

Estas pruebas fueron ejecutadas de dos maneras diferentes: pruebas manuales y pruebas automatizadas.

3.2 Pruebas Manuales

Estas pruebas fueron ejecutadas desde la interfaz de usuario, en éstas se ejecutaron todas las validaciones tanto de *frontend,* como de *backend*. Se dividieron en pruebas internas, las cuales fueron ejecutadas por la parte encargada de pruebas en nuestro equipo; y pruebas externas, donde el equipo de pruebas del área de la institución bancaria se encarga de realizar ejecuciones de todos los sistemas.

Las pruebas internas las ejecutaron en diferentes ciclos dentro del ambiente de desarrollo, dependiendo el plan, se fueron ejecutando estas pruebas conforme se iba acabando el desarrollo. Una vez que se acabaron estas pruebas y se resolvieron las incidencias, se realizó un *retest* para verificar las correcciones y así poder subir nuevamente al ambiente de UAT, en donde se realizaban las pruebas externas, verificando que todo el funcionamiento fuera el esperado.

Es importante mencionar que, al ser dos equipos diferentes, cada uno tenía como objetivo diferentes casos de prueba. Mientras que en el equipo interno se revisó que todo se ejecutara de forma correcta, que los mensajes fueran los correctos, etc.; el equipo externo se encargó de verificar el funcionamiento óptimo de negocio.

3.3 Pruebas Automatizadas de Microservicios

Además del desarrollo de los microservicios, buscamos la manera de probar estos de manera automatizada, generando algunas pruebas con ayuda de Karate. “Karate es la única herramienta de código abierto que combina la automatización de pruebas de API, simulaciones, pruebas de rendimiento e incluso la automatización de la interfaz de usuario en un marco único y unificado” (GitHub, 2019).

La sintaxis BDD popularizada por Cucumber es un lenguaje neutro y que tiene una estructura que es comprensible incluso para los no programadores. Por otra parte, las poderosas aserciones JSON y XML están integradas y pueden ejecutar pruebas en paralelo para aumentar la velocidad. Esta herramienta fue muy útil para la elaboración de pruebas para cada caso involucrado en cada uno de los microservicios.

Esta herramienta es muy sencilla de utilizar y aunque el conocimiento que tenía en ese momento era casi nulo, con la ayuda de la lógica de programación fue suficiente para poder comprender la forma de trabajar con ella y solo basto con entender las palabras clave para que así ayudaran a mejorar cada una de las diferentes pruebas.

Una vez listadas las validaciones, separamos las validaciones que corresponden únicamente al funcionamiento del *backend.*

El desarrollo de las pruebas lo realizamos en un mismo proyecto, en donde la estructura las separamos por servicios y pruebas.

En la carpeta de servicios realizamos el desarrollo de la estructura de cada una de las operaciones del servicio: su método, los encabezados utilizados, el cuerpo de la

petición (si se requiere) y los parámetros de entrada, cada uno dentro de la carpeta correspondiente al nombre del microservicio.

En la carpeta de pruebas, se desarrollaron cada una de las validaciones simples: validación de encabezados y de campos, que normalmente regresarían un código 400 *Bad Request*.

Además, se realizaron diferentes pruebas de negocio, donde se validaron tanto pruebas positivas como negativas, y en el cual se validaba que se manejara un error de manera correcta.

Al final de la ejecución de las pruebas automatizadas, se creó un reporte en el que se enlistó cada una de las validaciones probadas, los pasos realizados, y si pasó o falló la prueba. Si la prueba fallaba, se visualizaba el paso en el que tuvo error. Más adelante se muestra una gráfica de pastel con el porcentaje de las pruebas pasadas y fallidas.

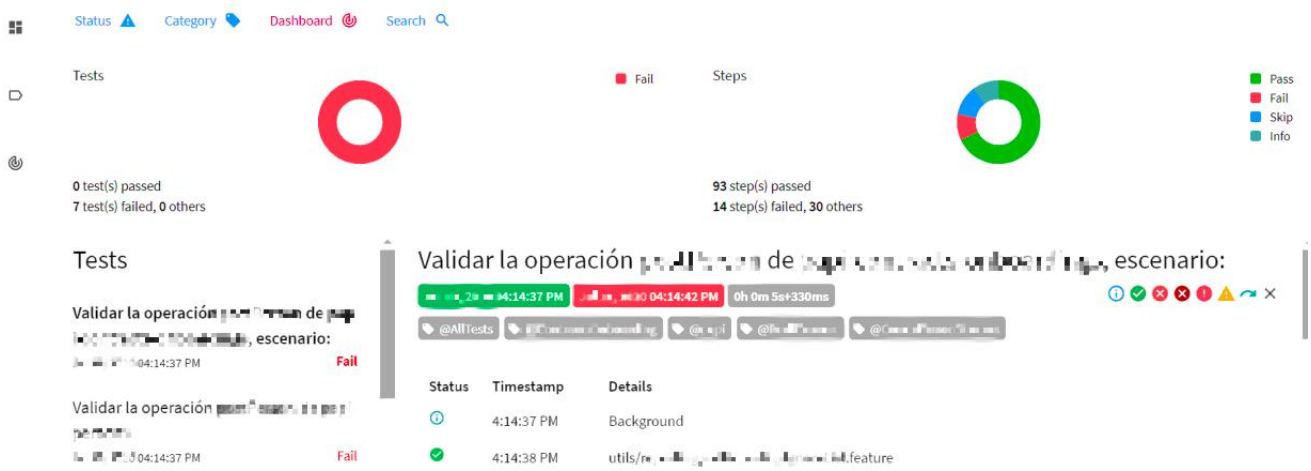

Figura 18. Ejemplo de Reporte de Automatización

Este reporte estaba en formato HTML, lo cual implicaba que se podía visualizar en cualquier navegador.

La Figura 18 mostrada anteriormente es un ejemplo de un reporte. Como se muestra en la imagen, las gráficas generales estaban en la parte superior; en una de

ellas se observaban los casos ejecutados, ya sea en rojo si son fallados y en verde si pasaron. En la segunda gráfica se mostraban los pasos ejecutados. En ésta se podían observar los pasos pasados en verde, los pasos fallados en rojo, en color azul se distinguen los pasos que se saltaron (porque algún paso anterior falló), y en color verde agua se mostraban los pasos que son meramente informativos.

En la parte inferior izquierda se mostraban los casos de prueba que se ejecutaron con su descripción y estado de la prueba.

Al seleccionar cada uno de ellos se podían visualizar, en la parte derecha, todos los pasos ejecutados de un caso, y en caso de un paso fallado, se mostraba con un tache como se puede ver en la Figura 19.

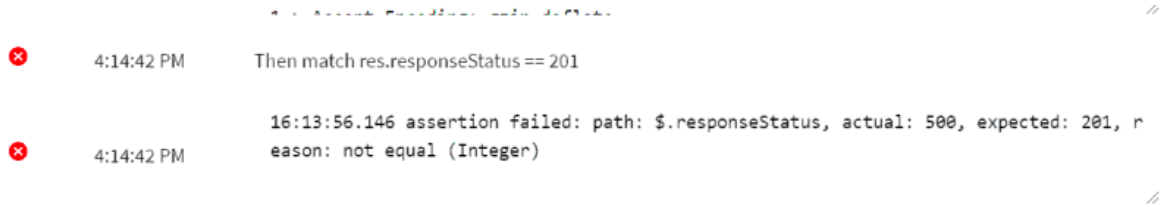

Figura 19. Ejemplo de los pasos de una prueba

Estos reportes se podían ejecutar dependiendo de los casos que se requirieran con ayuda de las etiquetas que se mostraban en la parte superior de esta sección. Estos nos ayudaban a diferenciar entre microservicios, capas, o casos en específico.

3.4 Análisis de Resultados

Las pruebas realizadas dentro del equipo estuvieron repartidas en 2 ciclos. En el primer ciclo, el 61% de las pruebas fueron aceptadas, mientras que el 24% fueron fallidas, quedando pendientes 15%, ya que tenían dependencia de las pruebas que no pasaron.

Este primer ciclo se realizó internamente para tener un entregable en una fecha específica con un porcentaje de avance mayor al 60%, por lo que las pruebas fallidas

no se corrigieron hasta el siguiente ciclo de pruebas. De estas pruebas, los casos quedaron como se muestra en la Tabla 2:

Tabla 2. Primer ciclo de pruebas

Casos de Prueba	Pasados	Bloqueados	Fallados	Prioridad Casos Fallados
334	203	51	80	Críticos - 27 Medio - 21 Bajo – 32

En la Figura 20 podemos ver la gráfica de estas pruebas.

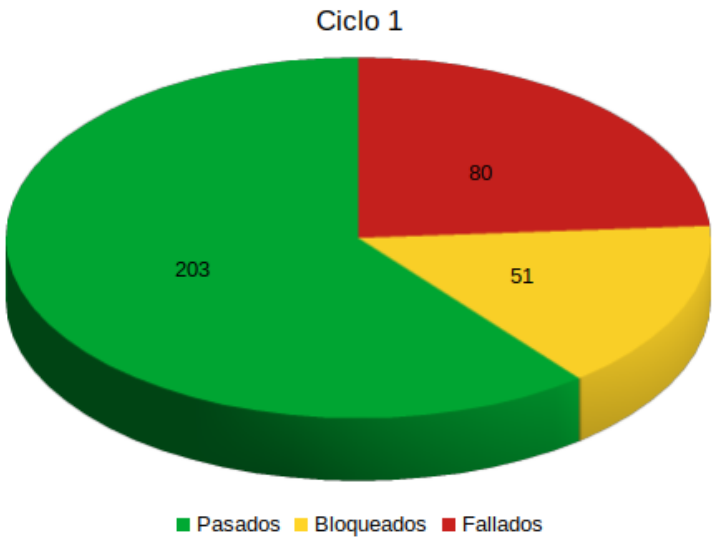

Figura 20. Gráfica de pruebas

Para un mejor entendimiento, a continuación, se describe cada columna:

- Pasados: eran aquellos casos de prueba que fueron exitosos, esto quiere decir que cumplieron con todos los puntos a verificar para ese paso y regresó la respuesta esperada.
- Bloqueados: eran aquellos casos de prueba que no se pudieron ejecutar, ya que necesitaban algún paso anterior que había fallado o no estuvo completo.
- Fallados: eran aquellos casos de prueba ejecutados pero que no regresaron la respuesta esperada, estos se dividieron en tres estados o prioridades:

- Críticos: eran aquellos en los cuales la aplicación fallaba, dejando pendiente una transacción o realizaba alguna acción no esperada afectando a otros flujos.
- Medios: eran aquellos que fallaban, pero se podía seguir ejecutando el flujo sin dependencias adicionales.
- Bajos: eran aquellos que se ejecutaban correctamente, sin embargo, el mensaje de la respuesta no era la esperada.

El segundo ciclo de pruebas se realizó para la entrega final con un 90% de casos pasados y sin tener errores críticos. Por lo cual, los datos quedan conformados de la siguiente forma:

Tabla 3. Segundo ciclo de prueba

Casos de Prueba	Pasados	Bloqueados	Fallados	Prioridad Casos Fallados
334	301	0	33	Críticos - 0 Medio - 6 Bajo - 27

En la Figura 21 podemos ver la gráfica de estas pruebas.

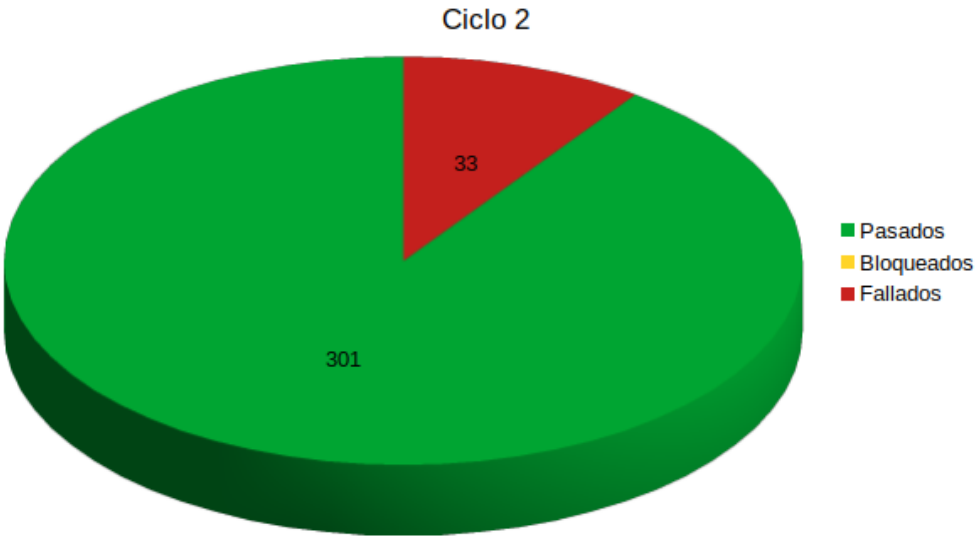

Figura 21. Gráfica de segundo ciclo de pruebas

En la mayoría de los casos, los errores sobrantes se debían a una corrección de mensajes de salida, tanto desde *backend* como desde *frontend.*

Las pruebas realizadas por automatización las utilizamos para validar la corrección de incidencias en los dos ciclos. Posteriormente, éstas serían utilizadas para nuevos cambios o pruebas que se deban hacer durante el despliegue a otros ambientes, en este caso, producción.

Después del segundo ciclo, se pasaron las pruebas al equipo encargado de *testing* del área, quienes tenían casos de prueba más relacionados con el negocio, conteniendo igualmente casos de validación más específicos.

En esta etapa, el desarrollo lo subimos a ambiente UAT, por lo que, pasando estas pruebas, se daría el visto bueno para poder subir a ambiente de producción. Este equipo era independiente del equipo en el que se trabajó, pero parte del área, por lo que las pruebas eran independientes a las nuestras.

Este mismo equipo también se encargó de las pruebas de rendimiento, dando como resultado un mejor sistema, tanto en eficiencia como en eficacia.

En cuanto a la calidad del desarrollo del aplicativo haciendo uso de microservicios y las nuevas herramientas, se determinó que la funcionalidad fue la esperada e incluso mejoró los tiempos de respuesta derivado de la arquitectura implementada.

En contraste con la arquitectura anterior, los resultados obtenidos después de una etapa de monitoreo y experiencia con el usuario, se llegó a la conclusión de que haber optado por la tecnología nueva y actualizar el sistema para que tuviera un mejor funcionamiento durante más tiempo fue la mejor opción, y el resultado fue una estructura correcta y adecuada para que el mantenimiento sea más simple.

Comentarios Finales

Con este documento se puede dar por finalizado y cumplido el objetivo de plasmar las experiencias profesionales durante la implementación de una aplicación *web*, utilizando la plataforma de desarrollo Angular y Spring Boot, para la administración de usuarios y perfiles en el manejo de pagos al extranjero de una institución bancaria. De igual manera, el desarrollo del administrador terminó con éxito, obteniendo una mejora en eficiencia dentro de los procesos de la empresa.

Este proyecto fue un reto tanto personal como para el equipo de trabajo. En lo personal, lo ha sido porque se tuvo que aprender cómo se hacía el desarrollo anteriormente y se tuvieron varias sesiones para comprender la funcionalidad de los sistemas que se querían migrar. Por otro lado, para el equipo se tuvo que poner en práctica la organización, el trabajo en equipo, el liderazgo y compromiso con las fechas acordadas, lo que requirió de horas extras de trabajo, incluso en días no laborales. Además, la tecnología que se estaba utilizando ya era antigua, y por lo tanto algunas librerías internas de la empresa ya estaban descontinuadas, dando como resultado un sistema incompleto en funcionamiento y que quedaría obsoleto en corto plazo.

Lograr hacer este trabajo implicó unificar todos los sistemas para el usuario, haciendo más sencillo el uso de éste, desde que se pide el permiso para crear un nuevo usuario con ciertos perfiles, hasta el trabajo que debe realizar cada uno, ya que, como se vio en el desarrollo del presente documento, el objetivo siempre fue crear una experiencia más limpia, agradable y que al cliente le resultara atractiva, amigable al manipular y que no fuera complicado.

Como resultado, se logró un sistema de administración de usuarios y perfiles más rápido y con mejor usabilidad. Incluso en las demos, los usuarios estuvieron de acuerdo con lo agradable que era a la vista, en comparación con el anterior.

En suma, al ser creado como microservicio, este sistema tiene la posibilidad de ser un componente adaptable para otros proyectos dentro de la empresa que, conforme al modelo de base de datos, es flexible para permitir nueva información que se requiera.

Este proyecto en específico fue solo el inicio de lo que más adelante representó una migración completa de los sistemas antiguos a la tecnología actual, teniendo en cuenta el cumplimiento de las políticas de seguridad de la empresa, para lo cual se tuvo que hacer una reingeniería profunda de cada sistema, para así lograr que el funcionamiento sea el mismo.

Además de esto, al ser un proyecto que se llevó a cabo tiempo atrás, puedo decir que además de ser un trabajo que mejoró los procesos en el banco; también representa una oportunidad para realizar futuras mejoras para elevar la seguridad y la eficiencia de tiempos en los microservicios, ésta es una observación que puedo hacer gracias a la experiencia que he ido adquiriendo en los diferentes proyectos en los que he participado después de concluir éste.

El uso de microservicios nos ayudó a mejorar la flexibilidad de los sistemas, y al haber sido un desarrollo en Spring, existen librerías que pueden mejorar su seguridad y eficiencia continuamente, además de que algunas simplifican el código para tener microservicios más comprensibles para un desarrollador y más ligeros a la hora de compilar y realizar el despliegue.

Entre otras recomendaciones y mejoras que yo hubiera realizado con el tiempo, se encuentran añadir la validación de *token* en todos los microservicios y no exclusivamente en la capa superior, para que las brechas que hayan quedado respecto a la seguridad se redujeran y se pudiera tener un mayor control de los datos que se generaban.

A pesar de haber sido un desarrollo simple, cuando se decidiera llevar a cabo la migración de los sistemas, se iban a generar más microservicios, extendiendo la comunicación entre ellos, y la administración de estos. Con ayuda de un punto de

acceso único llamado *gateway* o puerta de enlace, además de un servicio encargado del descubrimiento de otros servicios REST, se puede simplificar la dependencia de los datos en una base, o del direccionamiento directo entre microservicios.

Así como ahora tengo estos conocimientos, yo de igual manera tuve dificultades para poder entender como estábamos trabajando, apoyándome de mis compañeros en las dudas que tuviera y aprendiendo las buenas prácticas para poder tener una solución que fuera positiva al área de la empresa.

Muchas de las herramientas utilizadas, fueron por las que utilizaba el cliente en su área. Sin embargo, herramientas como Karate o Postman, fueron decisión de nuestro mismo equipo para darle un valor agregado al trabajo que estuvimos desarrollando.

De esta manera, el conocimiento obtenido me ayudó no solo a mejorar mis habilidades de desarrollo, si no que además mejoró mis habilidades de analista y solución de problemas para otros proyectos futuros en los que tuve participación.

En resumen, el trabajo se realizó con éxito y se obtuvieron los resultados esperados, además de que el crecimiento personal y profesional también fue provechoso en mi formación como ingeniera en sistemas.

Glosario

AJAX	JavaScript asíncrono y XML o Asynchronous JavaScript and XML, por sus siglas en inglés, es un conjunto de técnicas de desarrollo web que permiten que las aplicaciones web funcionen de forma asíncrona, procesando cualquier solicitud al servidor en segundo plano.
API	Interfaz de programación de aplicaciones o Application Programming Interfaces, por sus siglas en inglés, es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de *software* a través de un conjunto de reglas.
API Gateway	Sistema intermediario que proporciona una interfaz API REST o WebSocket para hacer de enrutador desde un único punto de entrada al API Gateway, hacia un grupo de microservicios y/o API de terceros definidos.
BackEnd	Parte interior de las aplicaciones que viven en el servidor y al que a menudo se le denomina como “el lado del servidor”.
Cucumber	Herramienta de *software* que admite el desarrollo basado en el comportamiento.
Facelets	Lenguaje de declaración de página potente pero ligero que se utiliza para crear vistas de JavaServer Faces utilizando plantillas de estilo HTML y crear árboles de componentes.
FrontEnd	Parte de una aplicación que interactúa con los usuarios, es conocida como el “lado del cliente”.

HTML	Lenguaje de marcado que se utiliza para el desarrollo de páginas de internet. Se trata de las siglas que corresponden a Hyper Text Markup Language, es decir, Lenguaje de Marcas de Hipertexto.
HTTP	Protocolo de Transferencia de Hipertexto o Hypertext Transfer Protocol, por sus siglas en inglés, es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML.
Java	Lenguaje de programación orientado a objetos, diseñado específicamente para permitir a los desarrolladores una plataforma de continuidad.
Javascript	Lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa.
Oracle	Sistema de gestión de base de datos de tipo objeto-relacional, desarrollado por Oracle Corporation.
Queries	Término o concepto que escribimos en un buscador cuando realizamos una búsqueda por palabra clave o *keyword.*
REST	Interfaz para conectar varios sistemas basados en el protocolo HTTP (uno de los protocolos más antiguos) y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.
SCRUM	Proceso en el que se aplican, de manera regular, un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.
Software	Programa o conjunto de programas de cómputo, así como datos, procedimientos y pautas que permiten realizar distintas tareas

en un sistema informático.

Spring Security	Marco de autenticación y control de acceso potente y altamente personalizable. Es el estándar de facto para proteger las aplicaciones basadas en Spring.
Stack	Conjunto de subsistemas o componentes necesarios para crear una plataforma completa, donde no se requiere un *software* adicional para soportar las aplicaciones.
UAT	Ambiente de pruebas de aceptación del usuario (UAT) que realiza el usuario final o el cliente para verificar o aceptar el sistema de *software* antes de mover la aplicación de *software* al entorno de producción.
WAS	Servidor de aplicaciones de *software,* de la familia WebSphere de IBM. WAS está construido por medio de estándares abiertos tales como J2EE, XML, y servicios web.
WAS Liberty	Servidor de aplicaciones Java™ EE con un entorno de tiempo de ejecución Java de bajo costo, diseñado para aplicaciones y microservicios nativos de la nube.
Web Service	Método de comunicación entre dos aparatos electrónicos en una red. Es una colección de protocolos abiertos y estándares usados para intercambiar datos entre aplicaciones o sistemas.
XML	Acrónimo de Extensible Markup Language, es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos.

Referencias

Blancarte, O. (2020). SOAP vs REST ¿cuál es mejor? *Oscar Blancarte - Software Architecture.* Recuperado de https://www.oscarblancarteblog.com/2017/03/06/soap-vs-rest-2/

Bucchiarone, A., et al. (2018). From monolithic to microservices: An experience report from the banking domain. *IEEE Software*. *35*(3), 50-55.

Dragoni, N., et al. (2017). Microservices: yesterday, today, and tomorrow. En *Present and ulterior software engineering* (195-216). Estados Unidos: Springer, Cham.

GitHub. (2019). Karate. Recuperado de https://github.com/intuit/karate

JSON. (2017). Introducción a JSON. Recuperado de https://www.json.org/json-es.html

Larrucea, X., Santamaria, I., Colomo-Palacios, R. & Ebert, C. (2018). Microservices. *IEEE Software*. *35*(3), 96-100.

Levcovitz, A., Terra, R. & Valente, M. T. (2016). Towards a technique for extracting microservices from monolithic enterprise systems. arXiv preprint arXiv:1605.03175.

Mazlami, G., Cito, J. & Leitner, P. (2017). Extraction of microservices from monolithic software architectures. En IEEE International Conference on Web Services (ICWS) (pp. 524-531). IEEE.

Oracle. (2019). ¿Qué es una base de datos relacional? Recuperado de https://www.oracle.com/mx/database/what-is-a-relational-database/

Pearlman, S. (2019). What is API-led Connectivity? *MuleSoft Blog*. Recuperado de https://blogs.mulesoft.com/dev/api-dev/what-is-api-led-connectivity/

Thönes, J. (2015). Microservices. *IEEE software*. *32*(1), 116.

UNEP. (2010). El Sector Financiero. *Monografías.* Recuperado de https://www.monografias.com/trabajos101/sector-financiero/sector-financiero.shtml

Upadhyaya, B., Zou, Y., Xiao, H., Ng, J. & Lau, A. (2011). Migration of SOAP-based services to RESTful services. In 2011 13th IEEE International Symposium on Web Systems Evolution (WSE) (pp. 105-114). IEEE.