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

DISEÑO E IMPLEMENTACIÓN DE UNA ESTRATEGIA DE SEGURIDAD MEDIANTE POLÍTICAS DE AUTENTICACIÓN Y AUTORIZACIÓN PARA UNA EMPRESA DE SEGUROS

TESIS PARA OPTAR AL GRADO DE MAGISTER EN TECNOLOGÍAS DE LA INFORMACIÓN

MANUEL ANTONIO CASTILLO GUTIÉRREZ

PROFESOR GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
CRISTIÁN ROJAS POBLETE
DANIEL PEROVICH GEROSA
ISMAEL FIGUEROA PALET

SANTIAGO DE CHILE
2018

RESUMEN

En la actualidad, la necesidad de una sólida estrategia de gestión de acceso e identidad se ha convertido en parte integral de los recursos de tecnología de una empresa. Las soluciones para estas estrategias pueden permitir a las empresas aumentar la satisfacción de los usuarios y reforzar sus posturas generales de seguridad. Sin embargo, el crecimiento de la computación en la nube y una mano de obra móvil cada vez más distribuida hacen que la gestión de acceso e identidad sea más compleja en el día a día.

Una empresa de seguros de Chile se enfrenta al problema anterior. La empresa desea proveer de acceso seguro y controlado a una API de servicios por parte de aplicaciones de terceros. También busca resolver una debilidad en los procesos de identificación con respecto a las aplicaciones de terceros y es cuando los usuarios desean autenticarse. Cada proceso de ingreso exige distintas identidades ocasionando que un mismo usuario tenga más de una cuenta.

Existen distintas alternativas y soluciones en el mercado, las mismas realizan desde la gestión de la identidad hasta la gestión de la autorización. El protocolo OAuth 2.0 es parte de estas soluciones, es el protocolo estándar de la industria para autorizar accesos limitados a las aplicaciones y los datos, motivo por el cual ha sido escogido como parte de la estrategia de seguridad propuesta. El protocolo se basa en la autorización delegada, es decir poder otorgarles a las aplicaciones acceso a la información en nombre de un usuario en vez de usar una cuenta por aplicación.

Se realiza una validación funcional de la solución en una aplicación móvil de terceros. La aplicación, mediante el protocolo, puede acceder satisfactoriamente a servicios expuestos en la API de la empresa. Se logra mejorar la gestión de la identidad al permitir que los usuarios de esta aplicación utilicen la cuenta propia que poseen con la empresa. Cumpliendo con estos objetivos se obtiene la aceptación de la solución para la empresa de seguros y se toma como estándar para futuras aplicaciones de terceros.

Este trabajo de tesis me deja como enseñanza la diferencia entre los protocolos de autenticación y autorización, como abordar la gestión de la identidad en los sistemas informáticos y sobre todo como interactuar con una API protegida con el protocolo OAuth.

AGRADECIMIENTOS

Agradezco a Dios por ser guía en mi camino y darme la fortaleza para cumplir mis metas. A mis padres por estar siempre apoyándome y motivándome día a día a pesar de la distancia.

Quiero agradecer a mis profesores Sergio y Daniel por brindarme su apoyo y conocimientos en varias etapas de la maestría. Un agradecimiento muy especial a Jocelyn por su orientación, paciencia y dedicación de manera que pueda realizar este trabajo de forma satisfactoria.

Tabla de contenido

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Capítulo I: Introducción

1.1 Contexto

Esta tesis se desarrolla en el marco de una empresa de seguros en Chile, que posee una trayectoria de muchos años en el mercado. La empresa ha aprovechado los recursos tecnológicos, la experiencia y el conocimiento adquirido para situarse como una empresa líder con un alto estándar de competitividad. También ha venido demostrando un constante crecimiento y expansión, siempre cumpliendo con el objetivo estratégico de brindar servicios de calidad a sus clientes, entre los cuales se encuentran los sistemas informáticos. El área de tecnología informática, en búsqueda de la mejora continua ha apostado por aumentar y diversificar los canales de información y así ofrecer servicios cada vez más atractivos a los consumidores.

La empresa de seguros ha acumulado mucho valor a lo largo de años de desarrollo de productos, servicios y reglas de negocio. Por lo cual decidió concentrar la información en una API (Application Programming Interface) de servicios Web REST (Representational State Transfer) con la finalidad de reestructurar y organizar sistemas sobre todo internos. A su vez, decidió comenzar a apoyar proyectos nuevos e innovadores de una manera uniforme, reducir costos de mantenimiento y aumentar la agilidad para el desarrollo de aplicaciones. Esta API cuenta con servicios que van desde la consulta de los datos de un cliente hasta la ejecución de operaciones como el cambio de plan de cobertura del mismo.

Esta empresa ha comprendido que los usuarios de hoy en día requieren de nuevos servicios contextuales e integrados porque empiezan a utilizar múltiples formas de acceder a ellos; no solo en aplicaciones propias de la empresa, sino también en aplicaciones de terceros, sobre todo en el canal móvil. Además, que las aplicaciones de terceros no necesariamente reemplazan las propias, sino que extienden sus servicios, como por ejemplo, una aplicación que permite que los usuarios de la empresa posean un repositorio en la nube para los documentos solicitados.

Con la llegada de aplicaciones construidas por terceros que necesitan acceso a la información que ofrece la API de servicios, surge una problemática que actualmente no está controlada por parte de la empresa de seguros, que es que el intercambio de información no está protegido de forma segura y eficaz. Bajo esta problemática se ha decidido implementar una estrategia de seguridad para el acceso a los recursos ofrecidos por la API de servicios.

1.2 Problemática a Resolver

En la actualidad la empresa de seguros tiene varios problemas que surgen con la llegada de aplicaciones de terceros. Por ejemplo, uno de los problemas es que la API de servicios implementada no tiene un acceso granular, por ende, lo normal es que el profesional de la empresa de seguros que necesite usarla deba identificarse usando algún método específico y obtenga acceso a servicios que van más allá de los necesarios. En el modelo tradicional, la aplicación cliente interactúa con la API utilizando los datos de autenticación con el servidor para acceder a él e intercambiar información. En este escenario, si las credenciales propias de la aplicación llegan a ser comprometidas, se obtiene total acceso a la información, abriendo una brecha de seguridad que se convierte en el punto más importante a resolver con el manejo de aplicaciones de terceros.

Por otra parte, se denotó una de las debilidades de los procesos de identificación que existen con respecto a las aplicaciones de terceros y es el enorme desgaste que se da en los usuarios que desean autenticarse en varios servicios o plataformas a la vez. Cada proceso de ingreso exige distintas identidades ocasionando que un mismo usuario tenga más de una cuenta.

En la Figura 1, se muestra la situación de la empresa de seguros con los problemas encontrados. Específicamente, el primer problema es que no existe una forma determinada y segura para que se puede acceder a la API de servicios desde una aplicación de terceros, por ende, una aplicación de terceros no puede solicitar recursos que se encuentran en la API. El segundo problema se da del lado de las aplicaciones de terceros, dichas aplicaciones fueron creadas con sus propias credenciales de usuario y que son distintas a las credenciales de la empresa de seguros, así un mismo usuario de la empresa podría tener más de una cuenta.

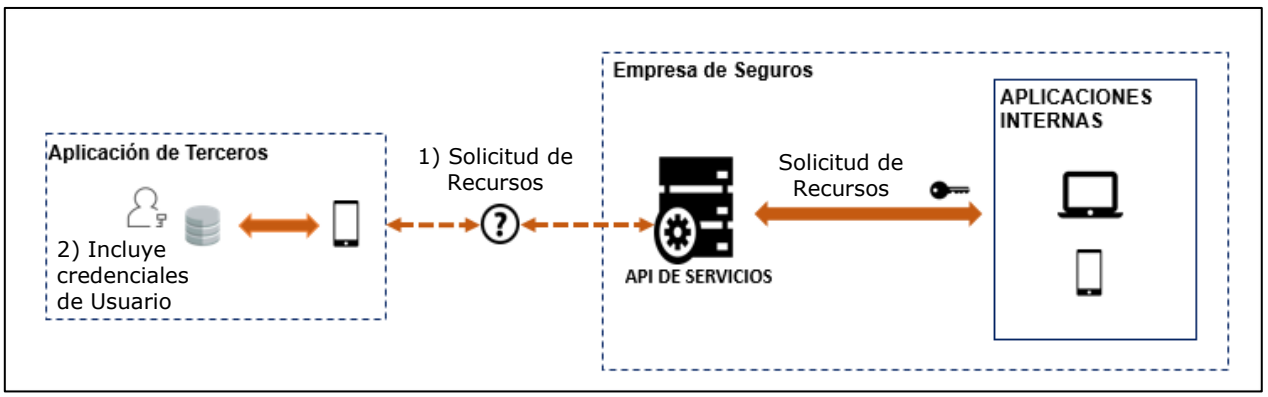

Figura 1 – Problemas encontrados en la empresa de seguros

Existe entonces, un desafío para la empresa de seguros, donde se precisa implementar un sistema que provea de un mecanismo de comunicación y validación de las peticiones, en este caso de aplicaciones de terceros. El sistema debe asegurar el control de acceso correcto a los recursos compartidos. Por otro lado, el canal móvil es uno de los más importantes que desea impulsar la empresa de seguros, por lo tanto, el mecanismo a implementar debe ser capaz de operar sobre aplicaciones móviles independiente de las múltiples plataformas actuales en el mercado.

1.3 Justificación de la Solución

Los problemas encontrados radican en la gestión de la identidad, es decir, en el proceso y empleo de tecnologías para gestionar información sobre la identidad de los usuarios. Existen varias tecnologías que brindan una solución a estos problemas como OpenID [1], OAuth [2], SAML [3], OpenID Connect[4]. OpenID []es un protocolo estándar para la autenticación, mientras que OAuth es el protocolo estándar para la autorización, SAML y OpenID Connect manejan ambos protocolos. La autenticación es verificar que alguien es quien dicen ser, mientras que la autorización es decidir a qué recursos un usuario debería poder acceder y qué se les debería permitir hacer con esos recursos.

Dado que el objetivo principal definido por la empresa de seguros es brindar autorización para que las aplicaciones de terceros puedan utilizar servicios expuestos en la API, se decidió por el protocolo OAuth. OAuth admite la autorización delegada entre aplicaciones mediante un código de seguridad. OAuth utiliza este código en lugar de confiar en una sola contraseña como clave maestra para cada aplicación que accede a los recursos ofrecidos por una API. El protocolo tiene la capacidad para que la primera vez que se emite el código a una aplicación, se solicite autenticación del usuario.

Se justifica el desarrollo de este trabajo para que proporcione una solución integral mediante el análisis, diseño e implementación del protocolo estándar OAuth 2.0 para la empresa de seguros, así pues, por medio del uso adecuado de las políticas de autenticación y autorización pertinentes se aporta confidencialidad e integridad a la hora de mantener una comunicación con la API y acceder a la información que ésta ofrece. Este protocolo implica todo un entorno para la validación de peticiones, restricción de acceso a recursos y gestión del acceso tanto de usuarios como de terceros.

La solución propuesta permite el manejo de un solo canal de autenticación, el protocolo realizará la autenticación del usuario utilizando el servicio que la empresa de seguros ya tiene implementado para sus sistemas. Esto evita la

duplicidad de cuentas y por lo tanto genera mayor confianza entre el usuario y la aplicación de terceros. En forma adicional, se implementa la librería para ser utilizada por aplicaciones móviles de la plataforma Android, con la finalidad de proveer el mecanismo de comunicación entre estas aplicaciones y el servidor de autorización.

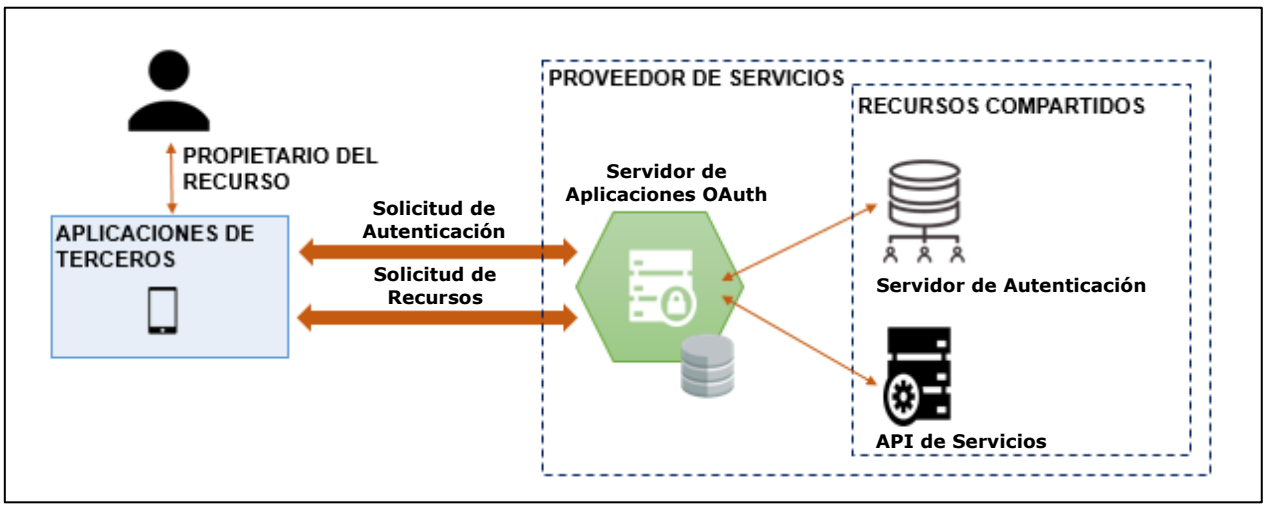

Figura 2 - Arquitectura General de la Solución Propuesta

En la Figura 2 se muestra la arquitectura general de la solución. Hay múltiples entidades involucradas en el protocolo OAuth. Una entidad es el usuario que autoriza el acceso a los recursos protegidos y es llamado propietario del recurso. Otra entidad es la aplicación que accederá a los recursos protegidos de un usuario. La aplicación antes de acceder a los recursos compartidos deberá solicitar al propietario del recurso que se autentique y también la autorización correspondiente para que la aplicación pueda acceder en nombre del usuario.

Para la solución aparece la figura llamada Servidor de Aplicaciones OAuth. Este servidor se encarga de exponer los servicios que sirven para obtener un código de seguridad, de la validación de las peticiones a los recursos protegidos y de la gestión y configuración de las aplicaciones de terceros que podrá acceder a los recursos compartidos.

Los recursos compartidos que pueden ser accedidos por las aplicaciones de terceros son los servicios expuestos por el Servidor de Autenticación y por la API de Servicios. Ambos servidores han sido implementados por la empresa de seguros y son accedidos por aplicaciones internas.

1.4 Objetivos de la Tesis

El objetivo general del trabajo de tesis es diseñar e implementar una estrategia de seguridad mediante el uso de políticas de autenticación y autorización para la empresa de seguros.

Los objetivos específicos definidos para el alcanzar el objetivo general son los siguientes:

1) Analizar, diseñar e implementar el protocolo de autorización OAuth 2.0 en la empresa de seguros mediante las aplicaciones llamadas servidor de autorización y servidor de recursos.
2) Desarrollar sistema Web que permita la gestión de las políticas de autorización entre las aplicaciones de terceros y la API de servicios.
3) Desarrollar la librería para Android que permita la comunicación entre las aplicaciones móviles y el servidor de autorización.

1.5 Descripción de la Solución

Para aclarar el alcance de la solución se especifican los siguientes productos que se obtienen como resultado del presente trabajo:

- Aplicación Servidor de Autorización, es el responsable de generar códigos de seguridad y validar usuarios y credenciales.
- Aplicación Servidor de Recursos, es capaz de aceptar y responder peticiones usando el código de acceso que debe venir en la petición.
- Sistema Web de Gestión de Políticas de Autorización, permite la configuración para las aplicaciones que deseen acceder al API de servicios mediante el utilizando el protocolo OAuth 2.0.
- Librería para la plataforma Android, permite la comunicación entre las aplicaciones móviles de terceros y el servidor de autorización.

1.6 Estructura del Documento

La tesis se estructuró en cada uno de los capítulos que se describen a continuación:

En el capítulo II se encuentra la revisión de los conceptos involucrados en el desarrollo de esta tesis, así como las consideraciones a tener en cuenta en la implementación de la solución. El capítulo III contiene el análisis de la situación de la empresa junto con los recursos de tecnología que posee para la implementación de la solución. Se detalla la especificación y análisis de los

requerimientos tanto funcionales y de calidad que se debe cumplir para la solución. Muestra la arquitectura general de la plataforma y los principales componentes implementados en la concepción de la misma.

El capítulo IV describe la implementación de la solución, especifica los componentes desarrollados y las tareas necesarias que se realizaron durante la etapa de desarrollo. En el capítulo V se muestra el análisis de la solución, su evaluación y las pruebas realizadas. Se presenta los resultados de la evaluación y la verificación del cumplimiento de los objetivos.

Finalmente, en el capítulo VI se presentan las conclusiones, se exponen las lecciones aprendidas y lineamientos para futuros trabajos.

Capítulo II: Marco Teórico

El presente capítulo contiene los conceptos principales que se deben tener en cuenta a lo largo del desarrollo de este trabajo. Está dividido en tres subsecciones, la primera aborda los aspectos teóricos con respecto a lo que posee la empresa en su infraestructura de servicios. La segunda subsección contiene la información contextual sobre el problema que se le presenta a la empresa y finalmente la última sección específica la tecnología principal que se usa en la solución a los problemas presentados.

2.1 API de Servicios Web REST

Para un mejor entendimiento de lo que representa una API de servicios Web REST se ha dividido los temas involucrados en tres secciones, la primera explica qué es una API y sus beneficios, la siguiente qué es un servicio Web, y por último los conceptos que engloba REST.

2.1.1 API (Application Programming Interface)

API significa interfaz de programación de aplicaciones. Una API es un conjunto de reglas y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas. Sirve de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción humano-sistema.

Una API proporciona un nexo para que desarrolladores de aplicaciones puedan acceder a datos y servicios, como ejemplos se tienen a las API de Twitter y Facebook. La popularidad de las APIs se basa en las funcionalidades que ofrecen o los datos a los que se tiene acceso.

Existen dos tipos de API: privadas y públicas. Se considera que APIs privadas son la variedad más prevalente [5]. A continuación, se discuten ambos tipos de API en forma separada.

2.1.1.1 APIs Privadas

La API privada es una interfaz que ofrece datos y funcionalidad del back-end de una organización, con el objetivo de que los desarrolladores internos usen esta información o funcionalidad. La interfaz administrativa de una API privada no está disponible para quienes no trabajan directamente para el propietario de la misma.

Las APIs privadas pueden reducir significativamente el tiempo y los recursos necesarios para:

- Integración de sistemas internos.
- Construcción de nuevos sistemas.

Estos beneficios extienden el alcance del mercado y agregan valor a las ofertas existentes de una organización. En lugar de crear aplicaciones desde cero, los desarrolladores pueden diseñar a partir de un conjunto común de activos de software internos, que es más rápido y más barato.

2.1.1.2 APIs Públicas

La API pública es una interfaz que ha sido diseñada para que su uso sea por desarrolladores dentro de la organización o por desarrolladores externos. Para mencionar un ejemplo se tiene el caso de Trello y Slack. Trello es una herramienta para la gestión de proyectos y Slack un servicio de mensajería en tiempo real. Slack utiliza la API de Trello para consultar los cambios que se realizaron en un determinado proyecto y así comunicar los cambios a los participantes del proyecto, siendo Slack una aplicación que no pertenece a Trello.

Una API pública generalmente busca alcanzar la creciente comunidad de desarrolladores de aplicaciones de terceros. Esto permitirá a la organización estimular el desarrollo de soluciones innovadoras que generen valor para el negocio principal, sin invertir directamente en los esfuerzos de desarrollo. Es importante resaltar que, en la mayoría de los casos, el éxito de una API pública dependerá de la funcionalidad y datos que ofrece.

2.1.2 Servicios Web

Los servicios Web surgieron como un conjunto de protocolos, estándares y recomendaciones, definidos por la W3C [6] (World Wide Web Consistorium) y OASIS [7] (Organization for the Advancement of Structured Information Standards), para lograr la interoperabilidad en la interacción entre máquinas, sistemas de software y aplicaciones a través de la red [8].

Así, una posible definición sería hablar de los servicios Web como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la red. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos.

Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar. Se puede encontrar diversos estilos de servicios Web. Para el marco de este trabajo de tesis se detalla el estilo basado en REST y que se explica en la siguiente sección.

La seguridad es considerada un concepto clave dentro del aseguramiento de calidad de un servicio Web [9]. Los criterios de seguridad básicos son la autenticación, autorización, confidencialidad, integridad e irrenunciabilidad. A continuación, se explica brevemente cada uno de ellos:

Autenticación: Los servicios Web por definición tienen mucha heterogeneidad, lo que provoca que los sistemas de autenticación tengan que ser flexibles. Si imaginamos un servicio Web que necesita comunicarse con otro servicio, este podría solicitar al demandante credenciales junto a una demostración de que es el propietario de las mismas. Resulta necesario conseguir una estandarización de los protocolos y en los formatos a utilizar. Otro problema remanente es definir un modelo de autenticación de forma que un servicio que necesita comunicarse con otros servicios Web, no tenga la necesidad de estar continuamente autenticándose y logre completar el proceso de negocio en un tiempo de respuesta aceptable.

Autorización: Con frecuencia, es necesario aplicar criterios que permitan controlar el acceso a los diferentes recursos. En combinación con la autenticación, permite a las identidades conocidas realizar las acciones para las que tienen permisos. Con frecuencia se definen políticas de acceso en base a jerarquías de usuarios.

Confidencialidad: Es necesario asegurar que el contenido incluido en los mensajes que se intercambian se mantiene como información confidencial. Es muy habitual emplear técnicas de cifrado. Obviamente, la confidencialidad del mensaje va más allá del canal por el que es transmitido.

Integridad: Esta propiedad garantiza que la información que se ha recibido, es exactamente la misma que se envió desde el cliente.

Irrenunciabilidad: En una comunicación donde se realizan transacciones, es necesario registrar la acción que se ha realizado y quién es el autor que la ha ejecutado. En el caso de los servicios Web, trasladamos esta política al uso del servicio. Se debería poder comprobar que cierto cliente hizo uso de un servicio a pesar de que éste lo niegue, así como demostrar que la ejecución se llevó a cabo.

2.1.3 REST (Representational State Transfer)

REST [8] se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen cómo los recursos son definidos y especificados. El término REST frecuentemente es utilizado en el sentido de describir a cualquier interfaz, que transmite datos específicos de un domino sobre HTTP, sin ninguna capa adicional. Los principios REST son descritos a continuación:

Cliente – Servidor: Al ser sistemas independientes solo necesitan comunicarse con un lenguaje de intercambio.

Sin Estado: Cada petición desde el cliente contiene la información necesaria para que el servidor sea capaz de preparar la respuesta.

Información cacheable: Para mejorar la eficiencia en el tráfico de red, las respuestas del servidor deben tener la posibilidad de ser marcadas como cacheables. Esta información es utilizada por los clientes REST para decidir si hacer una copia local del recurso con la fecha y hora del último cambio de estado del recurso.

Operaciones bien definidas: Los recursos de información son accedidos mediante operaciones como GET para obtener una representación del recurso, DELETE para eliminarla, POST para actualizar o crear representaciones y PUT para crear representaciones del recurso.

Acceso a recursos por nombre: Un sistema REST está compuesto por recursos que son accedidos mediante URL, y éstas deben ser intuitivas, predecibles y fáciles de entender y componer.

Recursos relacionados: Los recursos accesibles en el servidor suelen estar relacionados unos con otros. Por tanto, la información de estado de un recurso debería permitir acceder a otros recursos.

Respuesta en un formato conocido: La representación de un recurso refleja el estado actual del mismo y sus atributos en el instante en el que el cliente ha realizado la solicitud. La información debe ser entregada al cliente en un formato comprensible para ambas partes.

2.2 Gestión de la Identidad

La gestión de la identidad se refiere al proceso y empleo de tecnologías emergentes para gestionar la información sobre la identidad de los usuarios y controlar el acceso a los recursos de una organización. El objetivo de la gestión de la identidad es mejorar la seguridad al tiempo que reduce los costes asociados con la gestión de los usuarios y sus identidades, atributos y credenciales [10].

La gestión de la identidad incluye todo el proceso de decidir quién debe tener acceso a los recursos y a qué recursos; proporcionar, cambiar y terminar dicho acceso cuando sea apropiado; gestionar el seguimiento para el cumplimiento de las políticas internas y externas de la organización. Esto generalmente se aplica a situaciones en las que una persona tiene que identificar quién afirma ser por medio de una identidad verificada, como un pasaporte o una tarjeta de identidad en el control fronterizo, credenciales de acceso para la banca electrónica, identificación biométrica para el acceso a la cuenta en un cajero automático, y así entre otros.

La gestión de la identidad tiene dos componentes principales: la gestión de la identificación y la gestión de la autorización. La gestión de la identificación es el proceso de emisión y utilización de identidades digitales y credenciales (como nombres de usuario y contraseñas) para la autenticación. La gestión de la autorización combina la identidad probada del usuario con su autorización, con el fin de conceder acceso a los recursos protegidos por las políticas de seguridad de la organización. En las siguientes secciones, se describe con mayor detalle los dos componentes mencionados en la gestión de la identidad.

2.2.1 Gestión de la Identificación

Hoy en día, el uso de muchos de los servicios en Internet tales como envío de correo electrónico, banca en línea o compras a través de Internet, requiere demostrar primero que la persona es quien dice ser. A este proceso de demostrar nuestra identidad se le conoce como identificación.

Los métodos de identificación están en función de lo que se utiliza para la verificación de una identidad. Estos se dividen en tres categorías:

Factor de Conocimiento: Ejemplo, una contraseña.

Factor de Propiedad: Ejemplo, una tarjeta de identidad, una tarjeta inteligente (smartcard), dispositivo tipo token o dongle criptográfico.

Factor de Característica: Ejemplo, verificación de voz, de escritura, de huellas, de patrones oculares.

La gestión de la identificación es el manejo de todo el ciclo de vida de las identidades en una organización. Este ciclo de vida abarca varias operaciones para diversas actividades. Por ejemplo, la cuenta de inicio de sesión de un empleado para acceder a la red de la empresa es creada, mantenida, sincronizada y eliminada en múltiples sistemas o plataformas. Dado lo anteriormente mencionado, para la gestión de la identificación existen modelos que se adecuan según las necesidades del negocio y la confianza que se desee entregar a la entidad dueña de las credenciales y que se describen a continuación.

2.2.1.1 Modelo de Gestión de Identidades Aisladas

Este modelo requiere que cada usuario posea un identificador para el acceso a cada servicio aislado. Este sistema se utiliza mucho en los servicios y recursos en línea, ya que es relativamente simple para los proveedores de servicios de administrar, pero se está convirtiendo en inmanejable para los usuarios. El crecimiento exponencial de los servicios en línea ha llevado a que los usuarios estén sobrecargados de identificadores y credenciales, por ejemplo, diferentes usuarios y contraseñas que se necesitan recordar y administrar. En la Figura 3 se puede visualizar un ejemplo, en el que un mismo usuario debe manejar tres cuentas distintas para tres servicios diferentes.

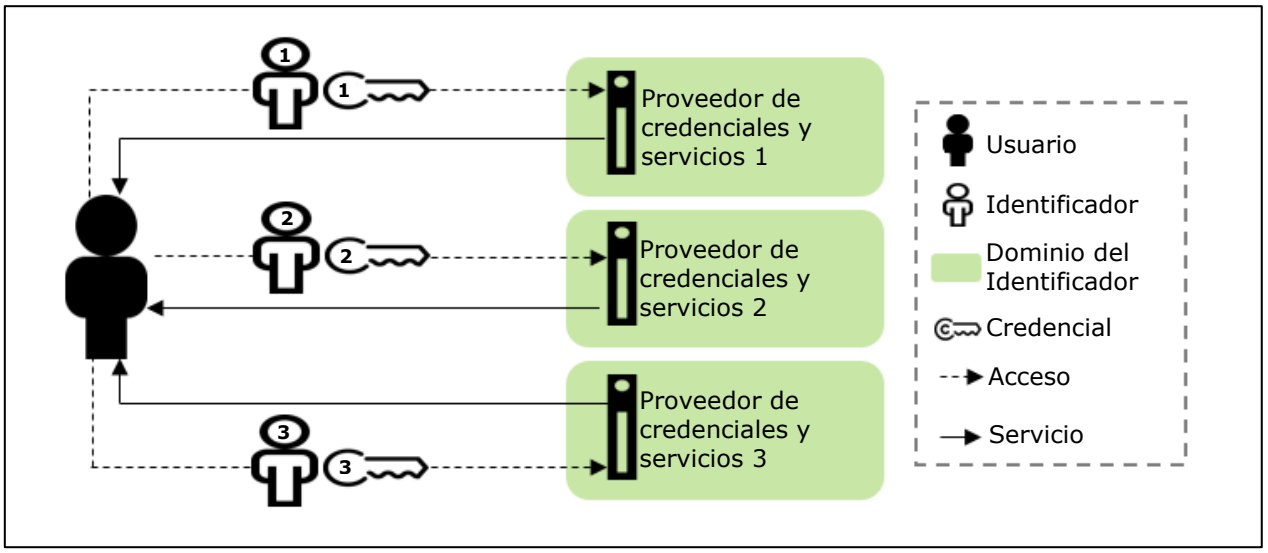

Figura 3 - Modelo de Gestión de Identidades Aisladas

2.2.1.2 Modelo de Gestión de Identidades Federadas

La administración de identidades federadas simplifica el problema de administración de cuentas. Un conjunto de acuerdos y normas se definen entre un grupo de proveedores de servicios que reconocen los identificadores de usuarios entre sí. Un cliente de un proveedor de servicios en particular podría acceder a todos los servicios proporcionados por otro proveedor de servicios en el grupo con un solo identificador. Para que tales métodos estandarizados de intercambio de información funcionen dentro del grupo, es necesaria la implementación de una norma tecnológica común.

En la Figura 4 se puede apreciar que existe una única cuenta que siempre se identifica con el proveedor principal y gracias a esta cuenta se puede acceder a los demás servicios. Cada proveedor es capaz de reconocer la cuenta principal y dar por autorizado el acceso, sin embargo, en cada proveedor, se sigue manejando la cuenta propia internamente y que se realiza mediante el mapeo entre proveedores. Por ejemplo, el acceso a diversas aplicaciones con una única cuenta de Google.

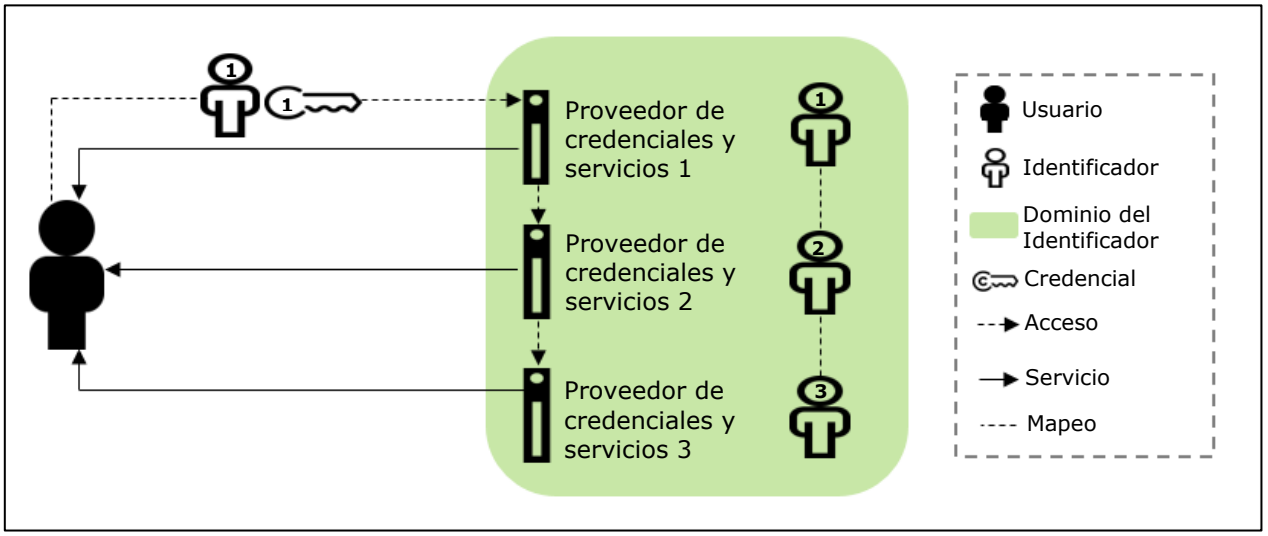

Figura 4 - Modelo de Gestión de Identidades Federadas

2.2.1.3 Modelo de Gestión de Identidades Centralizada

En este modelo, el mismo identificador y credencial son utilizados por cada proveedor de servicios. En un dominio de identificador centralizado, existe un único identificador, así como credencial que se utilizan por todos los servicios. Los dominios de identificadores centralizados pueden ser implementados en una serie de formas diferentes como, por ejemplo, dominio común del identificador

En la Figura 5 se observa que no se necesita tres cuentas individuales como en la Figura 3, sino que existe una cuarta cuenta que las reemplaza. Y a diferencia con la Figura 4 no se maneja mapeo entre cuentas en los proveedores, sino que se tiene un proveedor adicional de credenciales el cual maneja la cuenta número 4.

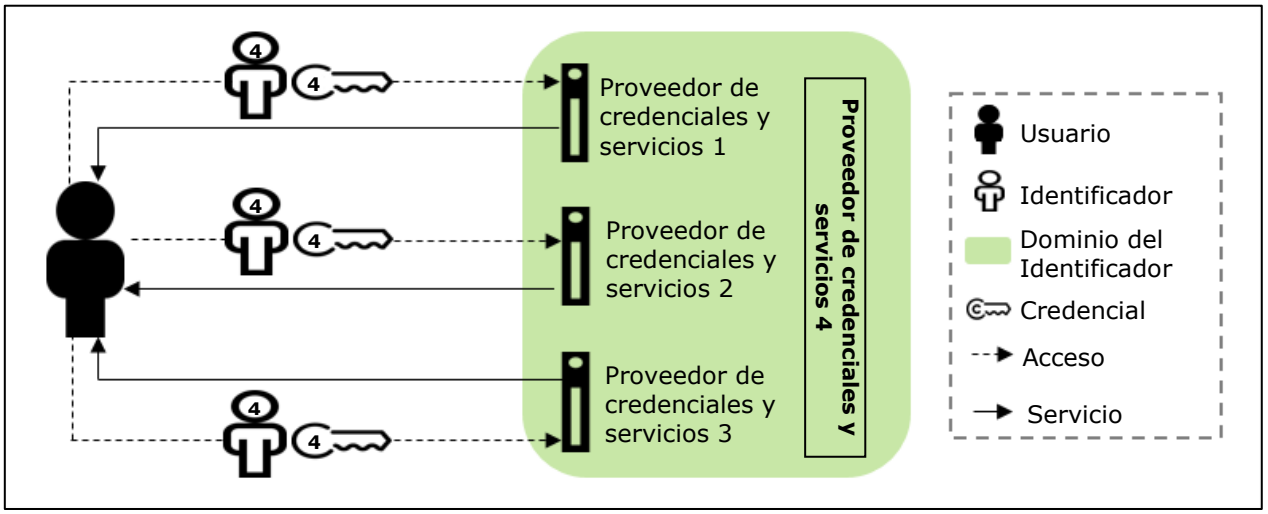

Figura 5 - Modelo de Gestión de Identidades Centralizadas

2.2.2 Gestión de la Autorización

Los sistemas de gestión de autorización van más allá de los mecanismos de autenticación tradicionales como contraseñas, tarjetas inteligentes y claves públicas o privadas. La gestión de la autorización determina si un usuario tiene permiso para acceder a un recurso y qué acciones puede realizar sobre este recurso. Una cosa es saber quién es, que se obtiene con la identificación, y otra, es saber lo que se le permite hacer una vez identificado.

La autorización delegada es el proceso para permitir que una persona o una aplicación pueda realizar acciones en nombre de otra persona. En la actualidad la autorización delegada no es algo que puedan realizar solo las personas; existen sistemas dotados de autonomía lógica con acceso protegido a procesos y repositorios de información, por lo que estos sistemas deben disponer del nivel de complejidad y calidad adecuado para lo que se desea proteger.

Tanto si lo que se quiere autorizar es una persona o una aplicación, se debe mantener un buen equilibrio entre la complejidad técnica de implementación y el proceso de negocio implicado. En caso contrario se diseñará sistemas muy robustos, pero tan complicados de implementar y usar que dejen de ser útiles. O, por el contrario, simplificaremos tanto el proceso de negocio que el sistema diseñado será en exceso frágil y, por tanto, la autorización será inútil, por

ejemplo, autorizar a un usuario que pueda realizar todas las acciones sobre el sistema.

2.3 Protocolo OAuth

OAuth [2] es un protocolo estándar para autorizar accesos limitados a las aplicaciones y los datos. Está diseñado de modo que los usuarios finales puedan conceder un acceso restringido a los recursos que poseen, por poner un ejemplo, darle permiso a un cliente externo para ver imágenes subidas a Facebook, este cliente externo puede ser un sitio de impresión de fotos. El protocolo se encuentra dentro del modelo de gestión de identidades federadas.

Empezó a definirse en 2006 y en 2007 se publicó la primera versión oficial. OAuth es importante porque deja la gestión de la delegación en manos del auténtico propietario de los recursos. El usuario establece las conexiones entre sus cuentas en distintas aplicaciones sin la implicación directa de los administradores de seguridad de cada sitio. Además, tiene la capacidad de alargar esta relación en el tiempo cuanto desee, pero también de terminarla con facilidad. Una de las grandes mejoras que aporta OAuth a la comunidad es la formalización del proceso por el que se delega la asignación de identidades a los usuarios.

Twitter y Facebook constituyen ejemplos de OAuth en la práctica. Un usuario, dispone de cuentas separadas en estas dos plataformas. Es posible que su nombre de usuario puede ser igual en las dos plataformas, pero al fin y al cabo son cuentas distintas, gestionadas en sitios diferentes. Por lo tanto, ¿cómo puede configurar todo lo necesario para que los tuits del usuario aparezcan al instante en su muro de Facebook?

Hace años se habría tenido que guardar el nombre de usuario y contraseña de Facebook en el perfil de Twitter. De esta forma, siempre que se publicara un nuevo tuit, la aplicación de Twitter podría iniciar sesión en nombre del usuario para publicar al mismo tiempo en Facebook. Confiar a Twitter la contraseña de Facebook tan solo otorga a esta aplicación mucho más poder. Twitter y Facebook utilizan OAuth para hacer frente a este reto, el protocolo proporciona un modelo de autorización delegada que permite que Twitter publique en su muro de Facebook, pero sin realizar ninguna otra acción. Este es un claro ejemplo y se muestra en la Figura 6. En general con OAuth se tienen más ventajas que se describen en las siguientes secciones.

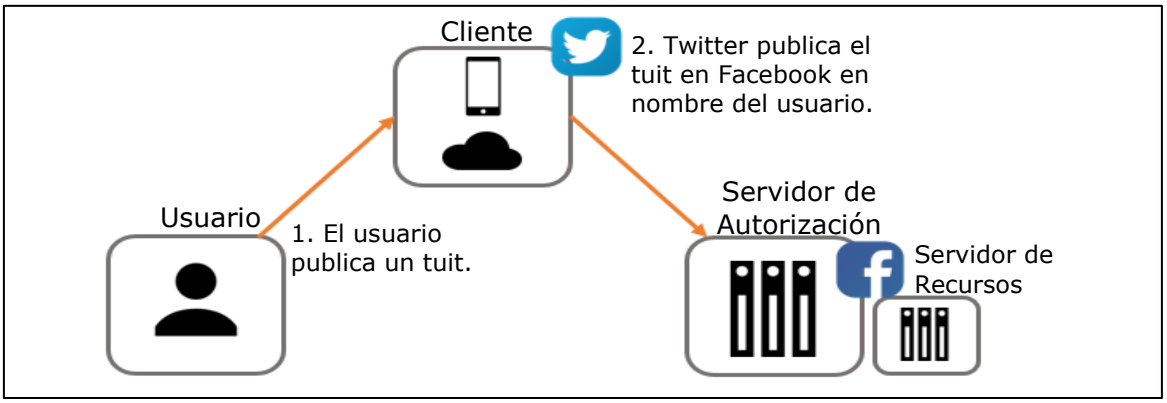

Figura 6 - Modelo de Autorización Delegada

Para concluir, OAuth es un protocolo abierto que permite la autorización segura desde aplicaciones de escritorio, web y móviles hacia una API, a través de un método simple y estándar. Fue construido sobre los estándares IETF [11] donde el núcleo de la implementación es un código de seguridad, en el protocolo el código de seguridad es llamado token de acceso [12] y es una cadena de caracteres. El token de acceso tiene derechos limitados que el usuario puede revocar en cualquier momento si se vuelve sospechosa la aplicación que está utilizando para acceder a su información.

OAuth admite la autorización delegada entre aplicaciones mediante el token de acceso. En lugar de confiar en una sola contraseña como clave maestra para cada aplicación que accede a una API, OAuth utiliza este token. El token sirve para el acceso a los servicios de una API en nombre de un usuario.

2.3.1 OAuth 2.0

En esta sección se menciona el porqué de un cambio de versión en el protocolo, los problemas que se resolvieron y las definiciones con las que finalmente el protocolo OAuth 2.0 se mantiene. Se detalla los flujos de trabajo del protocolo, así como también las consideraciones de seguridad que hay que tener en cuenta.

2.3.1.1 Definiciones de OAuth 2.0

OAuth 2.0 se centra en la simplicidad de su desarrollo al tiempo que proporciona flujos de autorización específicos para aplicaciones Web, aplicaciones de escritorio, teléfonos móviles y dispositivos de sala de estar. De esta manera se resuelve un problema con la especificación anterior que se volvió compleja durante su implementación. La especificación de OAuth 2.0 está desarrollada y publicada dentro del IETF como RFC 6749 [13] en el año 2012.

También OAuth 2.0 separa las funciones de autorización del control de acceso. Ahora, el servidor de autorización puede funcionar separado del servidor de recursos. Además de la discriminación lógica de las funciones, esta especificación promueve también el uso de un servidor de autorización central y la distribución de varios servidores de recursos. Esto nace para solucionar uno de los problemas que se presentaba en la versión anterior, OAuth 1.0 requiere que los puntos finales de recursos protegidos tengan acceso a las credenciales del cliente para validar la solicitud.

En la Tabla 1 se muestra las definiciones que juegan un papel en el proceso:

Rol	Descripción
Propietario del Recurso	Es el usuario que hace uso del recurso de forma convencional.
Token de acceso	Concedido por el servidor de autorización como respuesta a una solicitud correcta del cliente, permite que más tarde el servidor del recurso pueda identificar quién está intentando acceder al recurso y en nombre de quién. Tiene un tiempo de vida específico.
Servidor de Recursos	El servidor que aloja los recursos protegidos, capaz de aceptar y responder a peticiones de recursos protegidos usando tokens de acceso.
Cliente	Una aplicación que hace solicitudes de recursos protegidos en nombre del propietario del recurso y con su autorización. El término "cliente" no implica ninguna característica particular de implementación, por ejemplo, si la aplicación se ejecuta en un servidor, un escritorio u otro dispositivo.
Servidor de Autorizaciones	El servidor que emite tokens de acceso al cliente después de autenticar al propietario del recurso y obtener la autorización.
Código de autorización	Es un código entregado por el servidor de autorizaciones que permite a la aplicación poder negociar el token de acceso.
Token de Actualización	Es un código entregado por el servidor de autorización a las aplicaciones, este código es usado para actualizar el token de acceso una vez expirado su tiempo de uso.

Tabla 1 - Roles del Proceso OAuth 2.0

2.3.1.2 Tokens de Acceso y Actualización

Los tokens de acceso son credenciales utilizadas para acceder a recursos protegidos. Un token de acceso es una cadena que representa una autorización expedida al cliente. La cadena suele ser opaca para el cliente. Los tokens representan los alcances específicos y también la duración del acceso, concedidos por el propietario de recursos, y aplicada por el servidor de autorización y recursos.

El token de acceso proporciona una capa de abstracción que reemplaza métodos de autorización, por ejemplo, nombre de usuario y contraseña, por un solo token entendido por el servidor de recursos. Esta abstracción permite expedir tokens de acceso más restrictivos y, así como eliminar la necesidad para comprender una amplia gama de métodos de autenticación del servidor de recursos [12].

Los tokens de actualización son credenciales utilizadas para obtener tokens de acceso. Los tokens de actualización son emitidos al cliente por el servidor de autorización y utilizados para obtener un nuevo token de acceso, esto cuando el token de acceso actual queda invalidado o caduca. También, sirven para obtener tokens de acceso adicionales con un alcance idéntico o menor.

A diferencia de los tokens de acceso, los tokens de actualización son solo para uso con servidores de autorización y nunca se envían a los servidores de recursos.

2.3.1.3 Registro de Aplicaciones Clientes

Antes de iniciar el protocolo, el cliente debe registrarse en el servidor de autorización. Típicamente involucra una interacción con un formulario de inscripción. El registro del cliente no requiere una interacción directa entre el cliente y el servidor de autorización. Cuando es apoyado por el servidor de autorización, el registro puede depender de otros medios para establecer confianza y obtener las propiedades del cliente requeridas, por ejemplo, URI de redirección, tipo de aplicación cliente, y aspectos de identificación requeridos como nombre de la aplicación, logo, términos legales, entre otros.

Los tipos de aplicaciones clientes en los que se ha basado la especificación son las aplicaciones web, las aplicaciones basadas en un agente de usuario y las aplicaciones móviles nativas. Para estos tipos de aplicaciones cliente existe un flujo determinado del protocolo OAuth 2.0.

2.3.1.4 Flujo General

El flujo general para el protocolo OAuth 2.0 se encuentra ilustrado en la Figura 7 y el detalle del flujo está descrito a continuación.

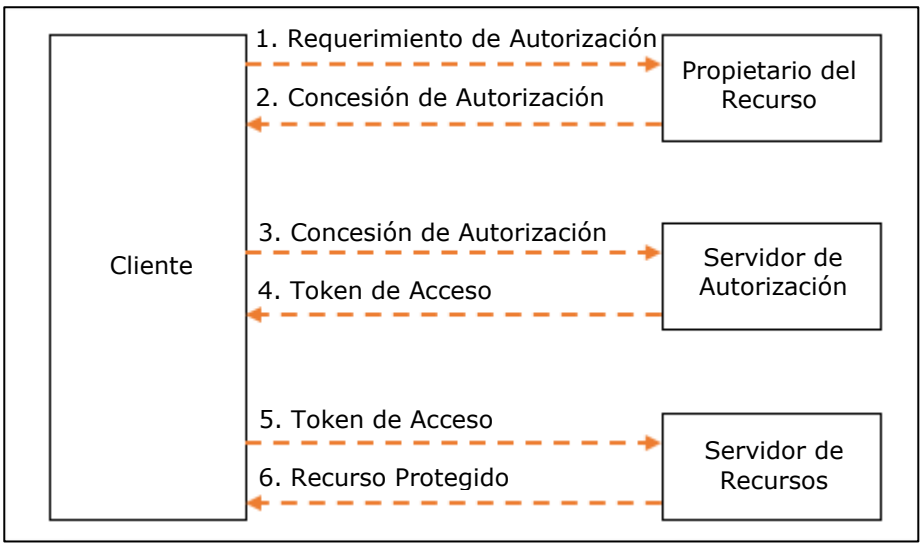

Figura 7 - Flujo General del Protocolo OAuth 2.0 [13]

1. El Cliente solicita autorización al Propietario del Recurso. Las solicitudes de autorización se pueden hacer directamente al propietario del recurso, o preferiblemente indirectamente a través del servidor de autorización como intermediario.
2. El Cliente recibe una concesión de autorización, que es una credencial que representa la autorización del propietario del recurso, expresado utilizando uno de los cuatro tipos de concesiones definidos en la especificación RFC 6749 [13] o utilizando un tipo de concesión de extensión. Los cuatro tipos de concesiones se explican en las siguientes secciones.
3. El Cliente solicita un token de acceso mediante la autenticación con el Servidor de Autorización y presentando la concesión de autorización.
4. El Servidor de Autorización autentica al Cliente y valida la concesión de autorización y, si es válida, emite un token de acceso.
5. El Cliente solicita el recurso protegido al Servidor de Recursos y se autentica presentando el token de acceso.
6. El Servidor de Recursos valida el token de acceso y, si es válido, responde la solicitud con el recurso protegido.

2.3.1.5 Concesiones de Autorización

Las aplicaciones clientes según su perfil deben utilizar el flujo apropiado del protocolo. El objetivo es obtener la autorización del propietario del recurso y acceder a sus datos. OAuth 2.0 define cuatro tipos principales de concesión que se utilizan para obtener autorización. Los tipos principales son descritos a continuación.

2.3.1.5.1 Credenciales del Cliente

En este escenario únicamente interviene el cliente y el servidor de autorización. Se utiliza cuando es el mismo cliente el que quiere acceder a datos del servidor de autorización sin necesidad de hacerlo en nombre de un propietario de recursos. Este tipo de concesión se usa principalmente para interacciones de servidor a servidor que se deben ejecutar en segundo plano, sin la interacción inmediata con un usuario. A menudo, estos tipos de aplicaciones se conocen como demonios o cuentas de servicio. En la Figura 8 se muestra el flujo y continuación la descripción.

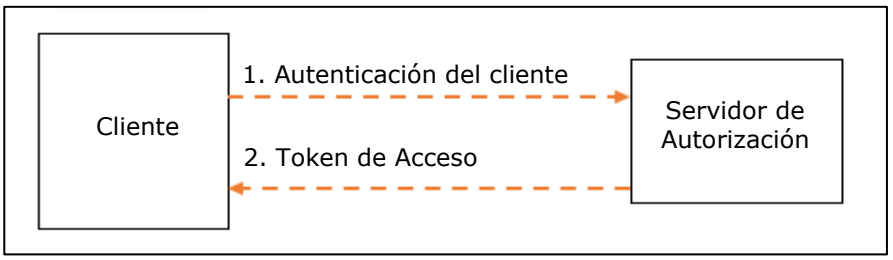

Figura 8 - Concesión de Credenciales del Cliente

1. El Cliente autentica con el Servidor de Autorización y solicita un token de acceso.
2. El Servidor de Autorización autentica al cliente, si es válido, responde un token de acceso.

2.3.1.5.2 Credenciales del Propietario del Recurso

En este flujo, el cliente recibe un token de acceso cuando el nombre de usuario / contraseña del usuario es validado por el servidor de autorización. Este flujo se recomienda para aplicaciones totalmente confiables. Una ventaja de este flujo sobre la autenticación básica, es que el usuario sólo presenta su nombre de usuario / contraseña una vez. A partir de entonces, se utiliza el token de acceso. La Figura 9 representa el flujo de esta concesión y a continuación su detalle.

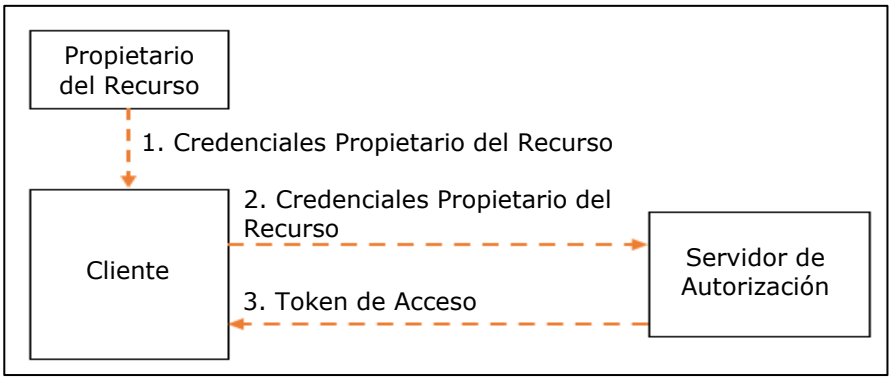

Figura 9 - Concesión de Credenciales del propietario del recurso

1. El Propietario del Recurso proporciona al Cliente sus credenciales de usuario.
2. El Cliente solicita un token de acceso al Servidor de Autorización incluyendo las credenciales del Propietario del Recurso. Con la solicitud el Cliente se autentica con el Servidor de Autorización.
3. El Servidor de Autorización autentica al Cliente y valida las credenciales, y si es válido, emite un token de acceso.

2.3.1.5.3 Código de Autorización

El código de autorización es el tipo de concesión que se puede usar en aplicaciones que se instalan en un dispositivo para obtener acceso a recursos protegidos, como las API. En este flujo, el usuario se autentica con el servidor de recursos y da a la aplicación, el consentimiento para acceder a sus recursos protegidos sin divulgar nombres de usuario y contraseñas.

El tipo de concesión de código de autorización se utiliza para obtener tokens tanto de acceso como de actualización. Dado que se trata de un flujo basado en la redirección, el cliente debe ser capaz de interactuar con el propietario del recurso mediante un agente, típicamente un navegador Web, ya que este es capaz de recibir las peticiones entrantes vía redirección, desde el servidor de autorización.

Este flujo se ve cada vez que la aplicación abre un navegador a la página de inicio de sesión del servidor de recursos e invita a iniciar sesión en su cuenta real (por ejemplo, Facebook o Twitter). Si inicia sesión correctamente, la aplicación recibirá un código de autorización que puede utilizar para negociar un token de acceso con el servidor de autorización. En la Figura 10 se puede observar el flujo de esta concesión y su descripción a continuación.

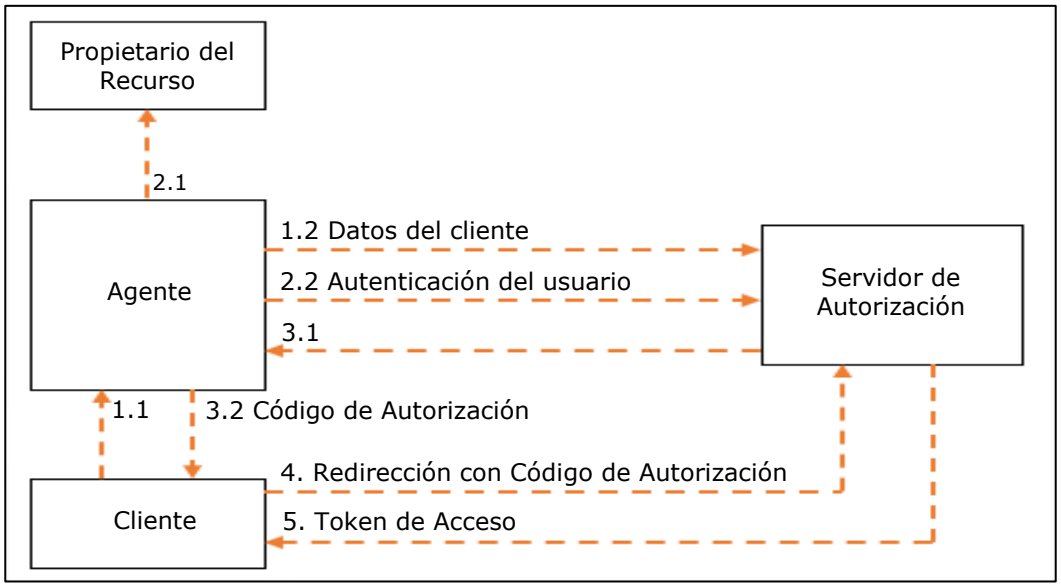

Figura 10 - Concesión de Código de Autorización

Nota: Las líneas que ilustran los pasos 1, 2 y 3 se dividen en dos partes a medida que pasan por el Agente.

1. El Cliente inicia el flujo dirigiendo al Propietario del Recurso al Servidor de Autorización mediante el Agente. El Cliente incluye su identificador de cliente, y una URI de redirección, la URI sirve para que el Servidor de Autorización redirija de vuelta al Agente una vez que el acceso se conceda o se deniegue.
2. El Servidor de Autorización autentica al Propietario del Recurso mediante el Agente y establece si el Propietario del Recurso concede o niega la solicitud de acceso del Cliente.
3. Asumiendo que el Propietario del Recurso concede acceso, la autorización redirecciona el agente de nuevo al Cliente usando la URI de redirección proporcionada anteriormente, en la solicitud o registro de clientes. La redirección incluye un código de autorización.
4. El Cliente solicita un token de acceso al Servidor de Autorización mediante la inclusión del código de autorización recibido en el paso anterior. Al hacer la solicitud, el cliente se autentica con el Servidor de Autorización. El cliente incluye la URI de redirección utilizada para obtener el token de acceso.
5. El Servidor de Autorización autentica al Cliente, valida el código de autorización y asegura que el URI de redirección recibido coincide con el URI utilizado para redirigir al cliente en paso 3. Si es válido, el Servidor de Autorización responde con un token de acceso y, opcionalmente, un token de actualización.

2.3.1.5.4 Implícito

Se considera una versión simplificada de la concesión código de autorización. Normalmente, este tipo de concesión se utiliza cuando la aplicación reside en el cliente. Por ejemplo, el código de la aplicación se implementa en un navegador que utiliza JavaScript u otro lenguaje de scripting. En este flujo de tipo de concesión, el servidor de autorización devuelve un token de acceso directamente cuando el usuario es autenticado, en lugar de emitir primero un código de autorización. La Figura 11 representa el flujo de esta concesión y a continuación su descripción.

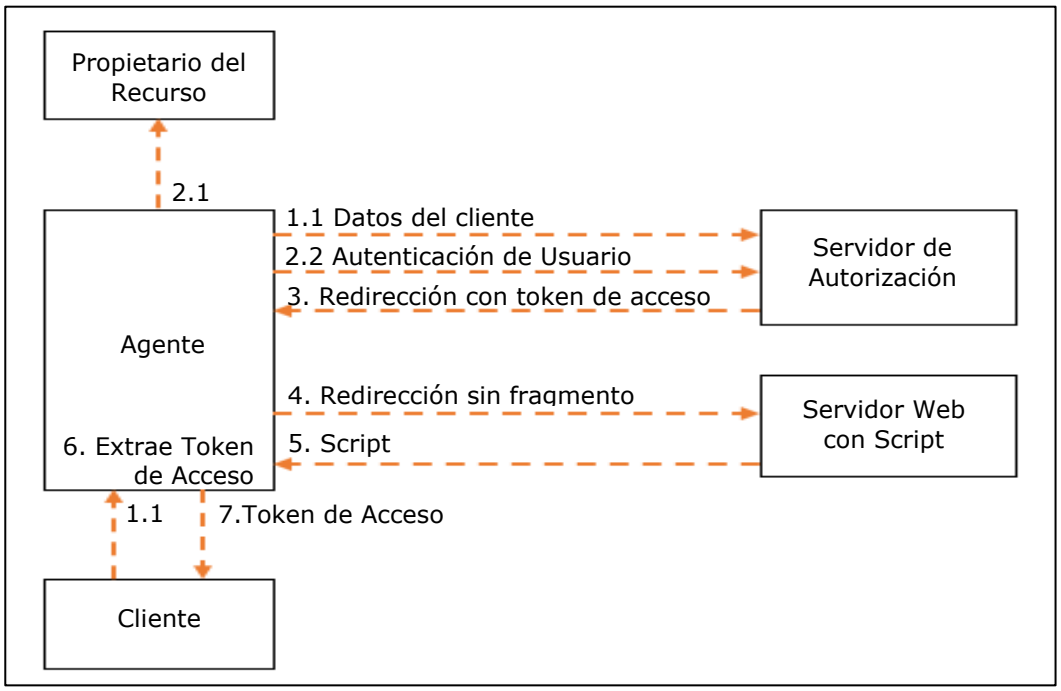

Figura 11 - Concesión Implícita

Nota: Las líneas que ilustran los pasos 1 y 2 se dividen en dos partes a medida que pasan por el Agente.

1. El Cliente inicia el flujo dirigiendo al Propietario del Recurso al Servidor de Autorización mediante el Agente. El Cliente incluye su identificador de cliente, ámbito solicitado, estado local y una URI de redirección, la URI sirve para que el servidor de autorización redirija de vuelta al agente una vez que el acceso se conceda o se deniegue.
2. El Servidor de Autorización autentica al Propietario del Recurso, mediante el Agente y establece si el Propietario del Recurso concede la solicitud de acceso del cliente.
3. El Servidor de Autorización redirecciona el Agente de nuevo al Cliente usando la URI de redirección proporcionada anteriormente, en la

solicitud o registro de clientes. El URI de redirección incluye un código de autorización y cualquier estado local proporcionado por el Cliente anteriormente.
4. El Agente sigue las instrucciones de redirección haciendo una solicitud al Servidor Web con Script. El agente retiene el fragmento de información localmente.
5. El Servidor Web con Script devuelve normalmente un documento HTML con un script incrustado capaz de acceder a la URI de redirección completa incluyendo el fragmento retenido por el Agente, y así extraer el token de acceso y posiblemente otros parámetros.
6. El Agente ejecuta el script localmente y extrae el token de acceso.
7. El Agente pasa el token de acceso al Cliente.

Esta concesión tiene el inconveniente de que el propietario de los recursos podría llegar a obtener el token de acceso y usarlo a su beneficio. Otra consideración es que no permite la emisión de tokens de actualización debido a que está pensado para clientes conocidos que operan en una URI de redirección particular.

2.3.1.6 Consideraciones de Implementación

Para poder asegurar que este protocolo de autorización es seguro, la especificación [12] define una serie de medidas y recomendaciones que deben cumplir todos aquellos que quieran utilizarlo. Algunas de las principales de medidas de seguridad son:

- Todo el intercambio de tokens con el servidor de autorización debe hacerse bajo el protocolo de comunicaciones TLS (Transport Layer Security).
- El servidor de autorización es el encargado de autenticar al usuario de forma segura.
- La generación de los tokens por parte del servidor de autorización debe garantizar que no puedan ser generados, modificados o adivinados por terceras partes.
- El tiempo de validez de los tokens o códigos de acceso debe ser corto y de un solo uso.
- Los clientes que se autentiquen con el servidor de autorización deben validar el certificado TLS del servidor.
- Los clientes no deben almacenar los tokens en sitios vulnerables o accesibles.

- El servidor de autorización debe comprobar que las URIs de redireccionamiento usadas tanto al conseguir el código de autorización como el token de acceso deben coincidir.

2.3.2 Relación entre los Conceptos

Con los conceptos claros, ahora se ahonda en la relación entre ellos para establecer de manera más concreta la situación del problema planteado y su respectiva solución en el marco de este trabajo de tesis.

2.3.2.1 Gestión de la Identificación y API de Servicios

En la empresa de seguros, la API nació de tipo privada para gestionar procesos internos y tener la oportunidad de reestructurar y modernizar su negocio. Detrás de este enfoque en la organización, el concepto clave fue desentrañar funcionalidades y servicios del negocio en módulos y componentes, con la finalidad de permitir la creación de nuevas cadenas con valor de negocio, al componer estos servicios, datos y funcionalidades en nuevas configuraciones.

Con el crecer de la empresa, se tomó la decisión de que ciertos servicios de la API fueran públicos para aplicaciones construidas por terceros. Esto conllevó a notar que la apertura de una interfaz para desarrolladores externos aumentaría significativamente los desafíos de administración y seguridad asociados a esa API.

Con el modelo de la empresa, en el que siempre se utiliza un usuario y contraseña para la consulta de servicios de la API, se determina que otorgarle estos datos a una aplicación de terceros generaría un gran problema de confianza, por los inconvenientes de seguridad que aparecen.

En los principales sistemas manejados en la empresa, tanto para usuarios internos como para clientes, ha sido implementada la autenticación federada, obteniendo ventajas como la centralización de sus usuarios y sistemas de autenticación y el refuerzo de políticas de seguridad; otra ventaja de cara al usuario final es la presentación de una única pantalla de inicio de sesión, cuando acceden a cualquier aplicación del dominio corporativo.

Para aprovechar estas ventajas, se considera que para el acceso de aplicaciones de terceros a la API de servicios se realice mediante el modelo de autenticación federado y que los datos para autenticar no sean propios de la aplicación sino de un usuario final.

2.3.2.2 Gestión de la Autorización y API de Servicios

Es una situación casi universal que hoy en día los usuarios hacen uso de muchas aplicaciones, algunas de las cuales almacenan datos para el usuario y algunas de las cuales consumen datos de un usuario. Existe un interés creciente en reunir estos dos tipos de aplicación, es decir, al permitir que ciertas aplicaciones que consumen datos de usuario accedan a datos almacenados por otras aplicaciones, en este caso la API de servicios.

Esta situación ha creado la necesidad de un mecanismo para delegar la autorización. En dicho mecanismo, el propietario de un conjunto de recursos puede delegar el acceso de tales recursos a una aplicación de terceros designada. Hay dos requisitos previos básicos de tal mecanismo de delegación:

1. Las aplicaciones que almacenan datos de usuario deben tener nociones separadas de propiedad y permisos de acceso, esto por supuesto, no está sujeto a la estandarización.
2. En segundo lugar, debe haber una forma de que las aplicaciones cliente soliciten acceso y que los usuarios ordenen a los servidores para que concedan dicho acceso.

El segundo requisito es el aspecto principal en el que se enmarca este trabajo de tesis y para lo cual se ha implementado el protocolo estándar OAuth 2.0, utilizando la concesión código de autorización para las aplicaciones de terceros. Consiguiendo un modelo de seguridad de autorización delegado en la empresa de seguros.

Capítulo III: Análisis y Diseño de la Solución

En este capítulo se especifican los objetivos y restricciones de la solución en base a los requisitos solicitados por la empresa de seguros. Junto a los objetivos también se describe la concepción de la solución y sus características principales mediante el diseño elaborado.

3.1 Análisis de Requisitos

La solución que se plantea en este trabajo de tesis es analizada en base a los requerimientos de la empresa de seguros. No obstante, se busca conseguir que la solución permita ser de guía para otras organizaciones que se encuentren con un problema similar. Se implementará el protocolo de autorización OAuth 2.0 en la empresa de seguros para exponer al público los servicios que poseen en su API privada. Esto permitirá que aplicaciones tanto propias como de terceros puedan utilizar dichos servicios.

La información de la situación de la empresa y el problema que tiene, fue obtenida mediante las reuniones de toma de requerimientos al inicio del proyecto. Se puede asociar los temas mencionados en el capítulo anterior junto con la información recolectada en dos problemas específicos para la empresa.

El primero, existen aplicaciones desarrolladas por terceros que necesitan acceder a servicios Web implementados por la empresa de seguros. Exponer estos servicios con el método actual de autenticación, que es mediante usuario y contraseña representa un riesgo de seguridad, en caso que se comprometa la aplicación. La gestión de la autorización para el acceso a los servicios no es manejada, cuando se da acceso, es un acceso total.

El segundo, es que las aplicaciones de terceros poseen sus propios mecanismos de autenticación, ocasionando múltiples cuentas para un mismo usuario de la empresa.

En base a lo mencionado anteriormente se han definido objetivos que vienen dados por la especificación del protocolo OAuth 2.0, mientras que los requerimientos específicos para el desarrollo de la solución se encuentran en el Anexo (A.1) del presente documento. Los objetivos basados en la funcionalidad se detallan a continuación:

a) **Acceso desde servidor Web:** Permitir el acceso al servidor de autorización y al de recursos desde una aplicación cliente que se ejecuta en un servidor Web. La aplicación Web es accedida por el propietario del

recurso (usuario) y la aplicación hace las llamadas apropiadas a la API de servicios desde el lenguaje de programación del lado del servidor.

b) **Acceso desde navegador Web:** Permitir el acceso al servidor de autorización y al de recursos desde una aplicación cliente que se ejecuta en el navegador Web de un usuario, donde la aplicación cliente tiene acceso a las solicitudes de la API.

c) **Acceso desde aplicación móvil nativa:** Permitir el acceso al servidor de autorización y al de recursos desde una aplicación cliente que se ejecuta en un dispositivo móvil con el sistema operativo Android, donde la aplicación cliente tiene acceso a los servicios de la API.

d) **Obtención de token de acceso:** Permitir obtener el token de acceso mediante los cuatro flujos de concesiones de OAuth 2.0: código de autorización, implícita, credencial del propietario del recurso y credenciales del cliente.

e) **Registro de aplicaciones clientes:** Permitir el registro de aplicaciones clientes, para obtener las credenciales que se utilizan para autenticar las solicitudes hechas al servidor de autorización.

En base a los requerimientos no funcionales solicitados se obtienen los siguientes objetivos:

a) **Bajo Acoplamiento:** La solución diseñada debe mantener un bajo acoplamiento entre sus componentes y así no generar dependencia entre sus funcionalidades.

b) **Eficiente:** La solución debe conservar o mejorar los tiempos de respuesta de las llamadas a los servicios de la API que se realizan desde una aplicación interna de la empresa.

c) **Robusto:** La solución debe ser capaz de reaccionar apropiadamente ante condiciones de excepción o de error y evitar así caídas o no disponibilidad de los servicios.

d) **Mantenible:** Con el paso del tiempo se irán agregando más recursos compartidos, por lo cual la solución debe permitir agregarlos al servidor de recursos de manera organizada.

e) **Confidencialidad:** La información almacenada estará protegida y será accesible únicamente por el personal autorizado.

3.2 Diseño de la Solución

En esta sección se presenta el diseño de la solución que cumple con los objetivos y restricciones mencionadas en la sección anterior. Se describe los servidores y aplicaciones que interactúan entre sí, los componentes implementados y su respectiva funcionalidad.

3.2.1 Arquitectura de la Solución

La arquitectura física de la solución está compuesta por cuatro servidores, el Servidor de Aplicaciones OAuth que es exclusivo para las aplicaciones de la implementación del protocolo, el Servidor de Base de Datos, el Servidor de Autenticación (LDAP) que maneja las cuentas de los usuarios de la empresa y el Servidor de API de Servicios. En la Figura 12 se muestra los servidores que interactúan en la solución. El Servidor de Aplicaciones es el único que debe tener acceso a los otros servidores.

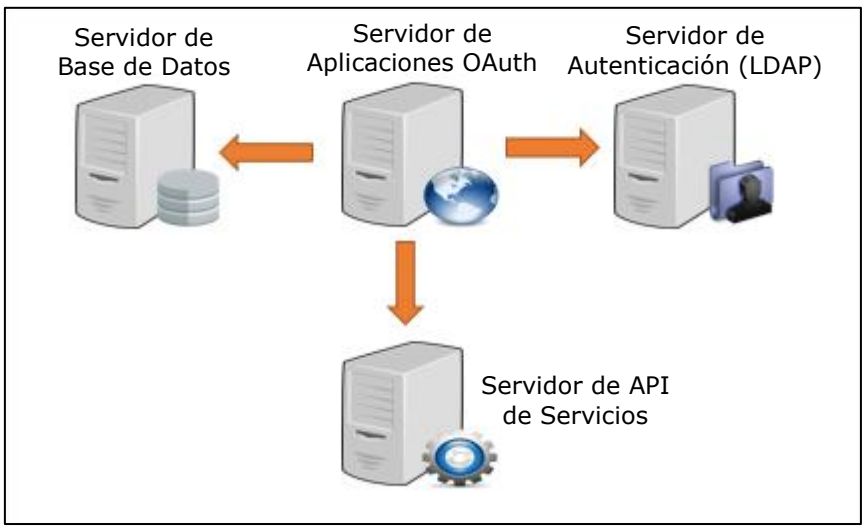

Figura 12 - Arquitectura Física de la Solución

A continuación, se describe el uso de cada uno de los servidores:

Servidor de Bases de Datos: Es el servidor que almacena en un solo esquema de datos la información que se utiliza en el protocolo como por ejemplo los tokens generados, las aplicaciones autorizadas, las políticas de

autorización entre otros datos que se describen en las siguientes secciones. El motor de base de datos es IBM DB2 e9.7.

Servidor de Aplicaciones OAuth: Es un servidor dedicado exclusivamente para la implementación del protocolo OAuth 2.0. Es el encargado de contener a las aplicaciones llamadas Servidor de Autorización y Servidor de Recursos. Por falta de recursos también se instala en este mismo servidor, la aplicación Mantenedor Políticas de Autorización. El Servidor de Aplicaciones OAuth posee el sistema operativo Red Hat Enterprise Linux 7 y como servidor de aplicaciones JBoss EAP 7. Más adelante se describe detalladamente cada aplicación de este servidor.

Servidor de Autenticación (LDAP): Este servidor ya es parte de la empresa de seguros y es utilizado para procesar las consultas acerca de los usuarios internos y de usuarios finales de la empresa de seguros, específicamente el servidor de aplicaciones lo usa para verificar la identidad de los usuarios. Las consultas se realizan mediante las especificaciones del protocolo LDAP (Protocolo Ligero/Simplificado de Acceso a Directorios) [14]. Un directorio es un conjunto de objetos con atributos organizados en una manera lógica y jerárquica, en este caso posee información de nombre de usuario y contraseña.

Servidor de API de Servicios: Es el servidor encargado de exponer servicios Web REST. Este servidor ya existe en la empresa de seguros y sus servicios son los que las aplicaciones pueden acceder según las políticas de autorización configuradas. Los servicios expuestos inicialmente para la solución son los siguientes:

- Consulta de información básica de un usuario, como nombres y apellidos, dirección de correo, entre otros.
- Consulta de estado de cuenta.
- Modificación de alternativa de fondo de ahorros.

3.2.2 Servidor de Aplicaciones OAuth

En esta sección se describe el Servidor de Aplicaciones OAuth, que tiene tres aplicaciones independientes. Como se puede apreciar en la Figura 13, existe una primera aplicación que cumple con las tareas de validación de peticiones y generación del token de acceso para el protocolo OAuth 2.0 y que es llamada Aplicación Servidor de Autorización. Una segunda aplicación que hace la validación de las peticiones de los recursos compartidos y que es llamada Aplicación Servidor de Recursos, y finalmente la tercera aplicación que se encarga de la gestión de las políticas de autorización y es llamada Aplicación

Mantenedor de Políticas de Autorización. Las tres aplicaciones tienen acceso directo a los otros servidores participantes de la solución.

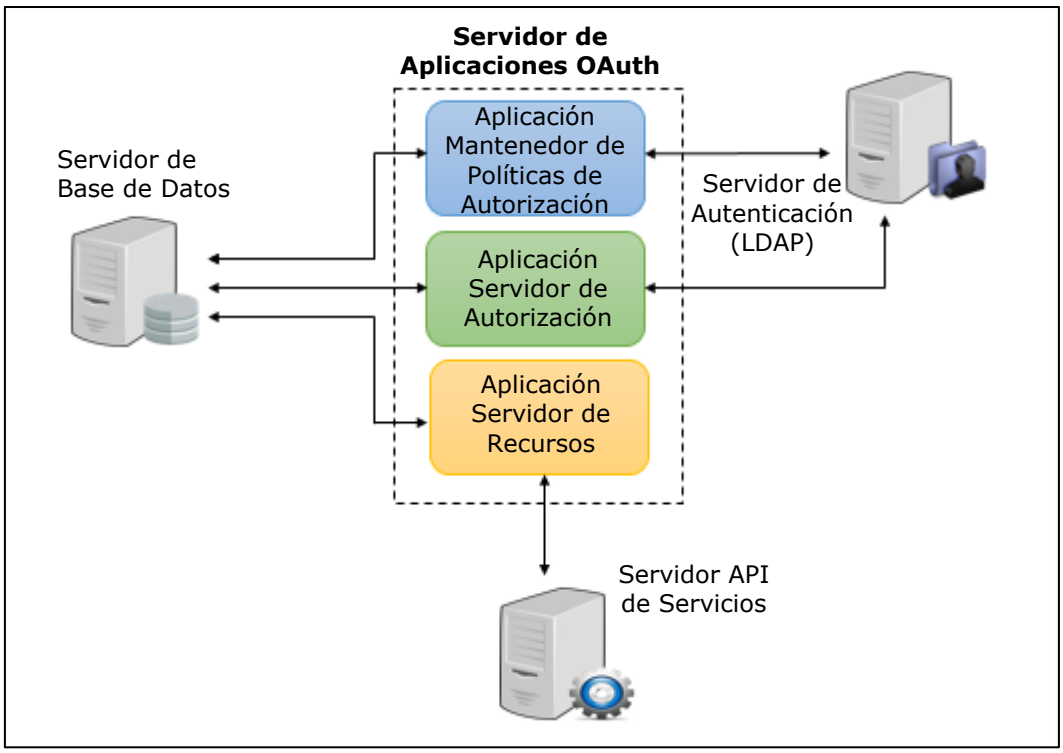

Figura 13 - Servidor de Aplicaciones OAuth

3.2.2.1 Aplicación Mantenedor de Políticas de Autorización

Es la aplicación que sirve para gestionar los clientes que pueden utilizar los recursos compartidos, permite configurar a cuáles de los recursos compartidos puede acceder, así como también gestiona a los usuarios internos de la empresa de seguros que pueden utilizar la aplicación. Esta aplicación utiliza el patrón de diseño MVC (Modelo-Vista-Controlador).

Este patrón separa los datos de la aplicación, la interfaz de usuario y la lógica de negocio. Es un patrón muy maduro y que ha demostrado su validez a través de los años. El modelo es la representación de la información con la cual la aplicación opera, la vista es la representación visual de la información para la interacción con el usuario, mientras que el controlador actúa de intermediario entre el modelo y la vista, gestionando el flujo de información entre ellos.

A continuación, en la Figura 14 se muestra la colaboración entre las distintas capas en el patrón de diseño MVC y el usuario; el controlador recibe una solicitud de un usuario para realizar una acción, procesa y manipula la información para lo cual utiliza el modelo establecido. Cuando el modelo es

actualizado, este puede arrojar eventos que notifican a la vista para que se actualice. Una vez que la información se encuentra procesada, el controlador envía a la vista dicha información para que pueda ser presentada al usuario de forma entendible.

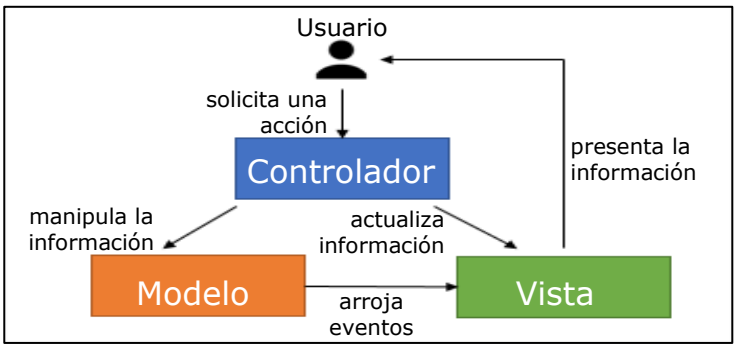

Figura 14 - Colaboración entre Capas - Patrón MVC

Las acciones que se pueden solicitar en la aplicación son las siguientes:

- Inicio de sesión
- Consulta de aplicaciones cliente
- Ingreso/Actualización de aplicaciones clientes
- Eliminación de aplicaciones clientes
- Consulta de usuarios autorizados en la aplicación
- Ingreso/Actualización de usuarios autorizados en la aplicación
- Eliminación de usuarios autorizados en la aplicación.

Para el controlador las acciones vienen representadas por servicios y la localización de cada uno de ellos como los parámetros de entrada y salida se encuentran en la especificación técnica en el Anexo (B).

3.2.2.2 Aplicación Servidor de Autorización

Este sistema está diseñado para manejar los cuatro tipos de concesiones de la solución, es decir, su objetivo es validar las peticiones que realice un cliente que desea obtener un token de acceso.

El sistema posee una arquitectura en capas, donde se definen jerárquicamente los roles y responsabilidades de cada una y así para proporcionar una división organizada de las funcionalidades implementadas. Cada capa tiene un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema; este diseño permite que algunos de los componentes puedan ser reutilizados en las otras aplicaciones de este servidor.

En la Figura 15 se muestra la arquitectura de la Aplicación Servidor de Autorización para la solución, donde se visualiza tres capas lógicas que se explican a continuación. Transversalmente a las capas lógicas de la aplicación se manejan los requerimientos de seguridad, trazabilidad, y manejo de excepciones o errores en cada una de ellas.

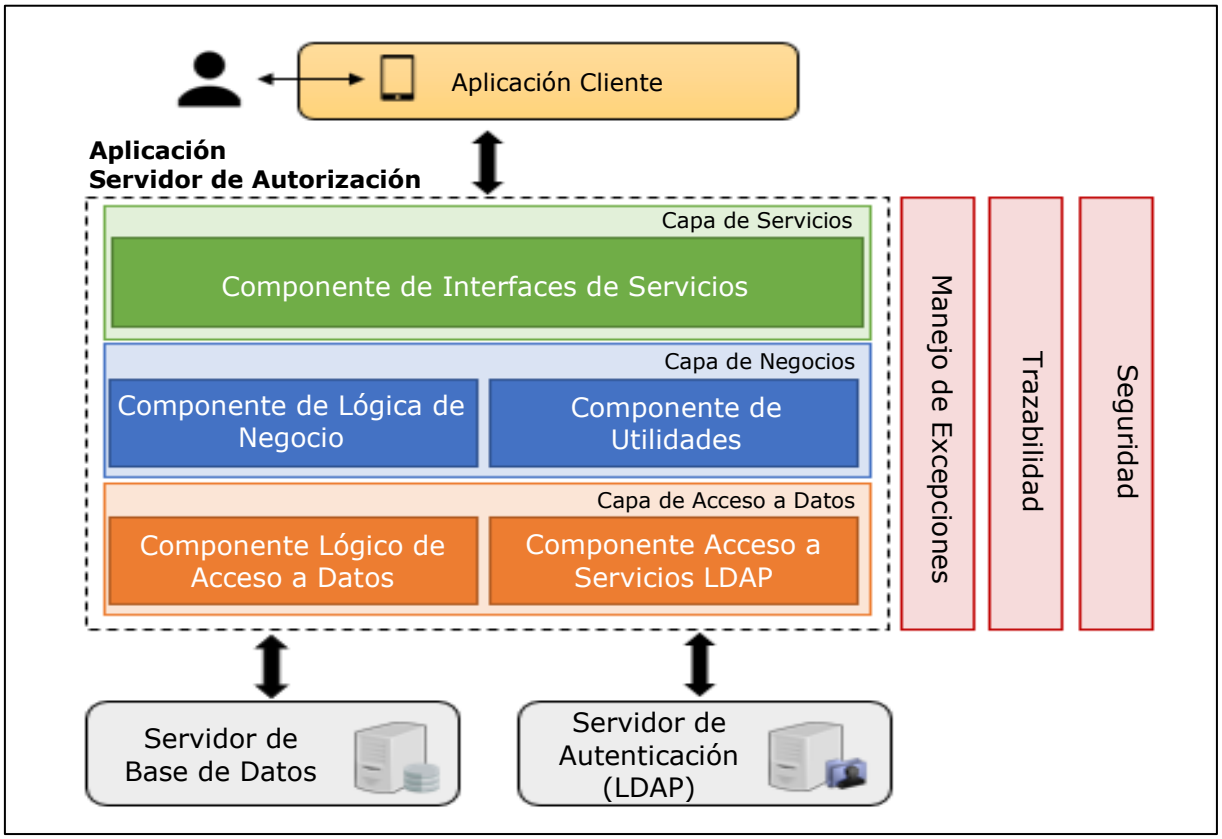

Figura 15 - Arquitectura de Aplicación Servidor de Autorización

Capa de Servicios: Proporciona una capa de abstracción para el acceso a datos que es independiente del esquema físico, está formada por la lógica de aplicación que es capaz de procesar los requerimientos que provienen de las aplicaciones clientes para luego realizar el envío de los datos de forma acorde con lo que necesita la capa de negocios. Su función principal es realizar la verificación de que los parámetros que deben llegar en las solicitudes lleguen y en caso contrario responder con mensaje de error.

Capa de Negocios: En esta capa es donde se establecen todas las reglas que deben cumplirse. Por ejemplo, una de ellas es la validación de que una aplicación este correctamente registrada. Esta capa se comunica con la capa de servicios para recibir y responder las solicitudes mientras que con la capa de acceso a datos lo hace para almacenar o recuperar información.

Capa de Acceso a Datos: Está formada por los servicios que proporcionan acceso a los datos persistidos y que son utilizados por la capa de negocio. Los servicios son para la persistencia de los datos en el Servidor de Bases de Datos y servicios para recuperar información de recursos almacenados en un directorio ordenado de datos, en este caso el Servidor de Autenticación (LDAP) de la Figura 15.

Las capas mencionadas anteriormente poseen componentes, los cuales han sido trabajados modularmente para garantizar la reusabilidad de algunos de ellos. Los componentes de cada capa de la aplicación son los siguientes:

Componente de Interfaces de Servicios: Este componente pertenece a la Capa de Servicios y está encargado de exponer los servicios que una aplicación de terceros puede consumir para solicitar el token de acceso. Las interfaces de los servicios creados para la Aplicación Servidor de Autorización cumplen con las funcionalidades de generación de código de autorización y de generación de token de acceso. Estos servicios son conocidos como *Authorization Endpoint* y *Token Endpoint* en la implementación.

Estas interfaces son utilizadas por cualquiera de los flujos de concesión del protocolo OAuth 2.0, su funcionalidad dependerá de los parámetros recibidos. Para este componente, tanto la localización de los servicios como los parámetros de entrada y salida se encuentran en la especificación técnica en el Anexo (B).

Componente de Lógica de Negocio: Este componente pertenece a la Capa Negocios y está encargado de realizar validaciones según las concesiones del protocolo, por ejemplo, verificación de código de autorización. También contiene reglas de negocio como tiempo de expiración de un token de acceso. Tiene como principal finalidad determinar los flujos de intercambio de información en base a los valores de los parámetros recibidos.

Componente de Utilidades: Este componente pertenece a la Capa Negocios y representa el conjunto de información con propiedades y comportamiento común para la implementación del protocolo. Posee la funcionalidad encargada de la encriptación del token de acceso, de actualización, el código de autorización y para el identificador de la aplicación cliente. Posee los valores constantes que se utilizan en las tres aplicaciones del Servidor de Aplicaciones OAuth.

Componente Lógico de Acceso a Datos: Este componente pertenece a la Capa de Acceso a Datos y suministra una interfaz común entre la aplicación y el Servidor de Base de Datos de la Figura 15. Posee los métodos necesarios

para realizar el ingreso, la actualización y la eliminación de los datos que son utilizados en el protocolo, por ejemplo, el ingreso de un token de acceso o la actualización de una aplicación cliente.

Componente Acceso a Servicios LDAP: Este componente pertenece a la Capa de Acceso a Datos y suministra una interfaz común entre la aplicación y el Servidor de Autenticación (LDAP) que se visualiza en la Figura 15. Este componente tiene la lógica para utilizar el protocolo LDAP, lo que permitirá buscar la información de los usuarios de la empresa de seguros.

3.2.2.3 Aplicación Servidor de Recursos

Este sistema está diseñado para el acceso a los recursos compartidos. Su funcionalidad principal es validar el token de acceso y si este tiene permiso para consumir un servicio. Al igual que el servidor de autorización, este sistema posee una arquitectura en capas.

En la Figura 16 se visualiza que la Aplicación Servidor de Recursos posee una arquitectura similar a la Aplicación Servidor de Autorización, tiene componentes que son reutilizados.

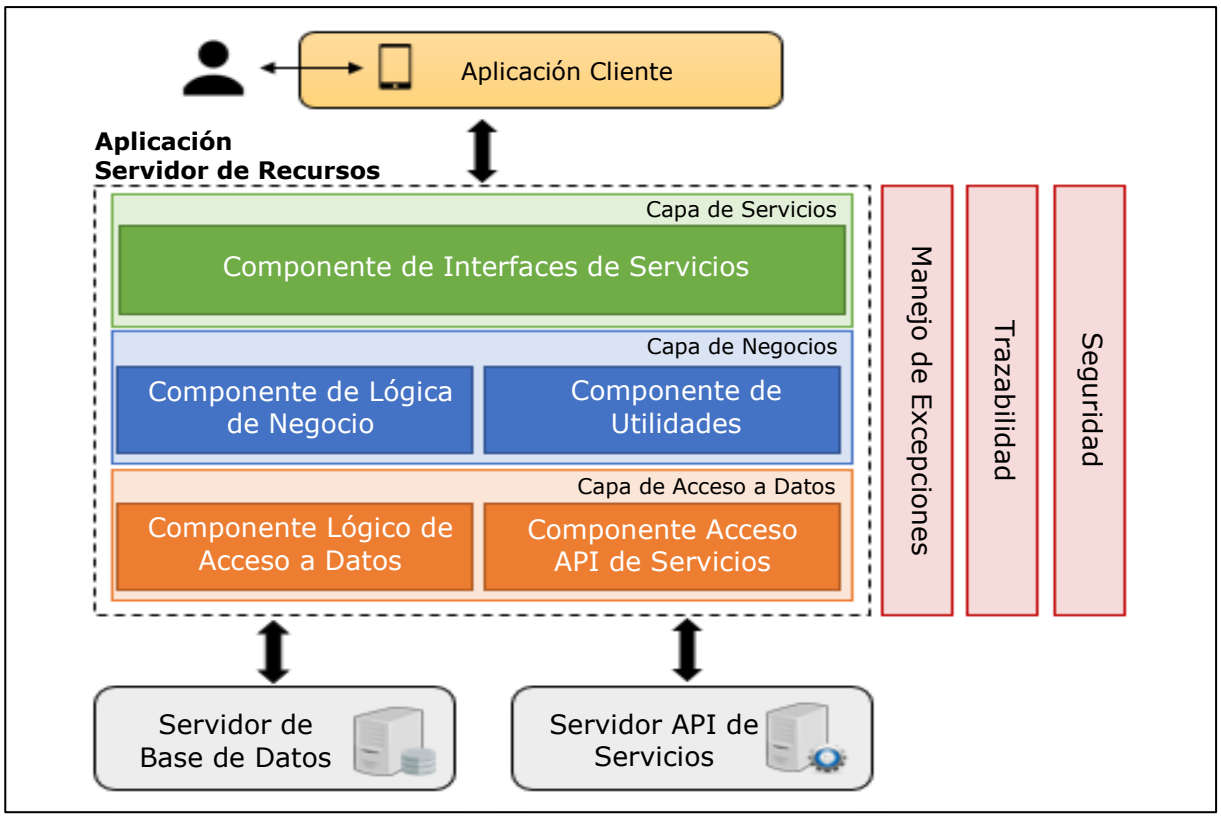

Figura 16 - Arquitectura de Aplicación Servidor de Recursos

A diferencia de exponer servicios para negociar el token de acceso, esta aplicación expone los servicios de los recursos compartidos y para ello necesita el acceso al Servidor API de Servicios de la empresa.

Componente de Interfaces de Servicios: Este componente está encargado de exponer los servicios Web que son compartidos a las aplicaciones clientes. Las funcionalidades de las interfaces expuestas son:

- Consulta de información básica de un usuario.
- Consulta información del estado de cuenta de un usuario.
- Modificación del tipo de fondo de una cuenta de un usuario.

Para este componente, tanto la localización de los servicios como los parámetros de entrada y salida se encuentran en la especificación técnica en el Anexo (B).

Componente de Lógica de Negocio: Este componente está encargado de realizar la verificación del token de acceso que es enviada entre los parámetros de la solicitud por una aplicación y si posee las respectivas autorizaciones para acceder a un servicio expuesto.

Componente de Utilidades y Componente Lógico de Acceso a Datos: Estos componentes son reutilizados de la Aplicación Servidor de Autorización.

Componente Acceso API de servicios: Este componente suministra una interfaz común entre la aplicación y el servidor API de servicios de la Figura 16. Se invoca los servicios expuestos por la API reenviando los mismos parámetros recibidos en el Servidor de Recursos.

3.2.3 Servidor de Base de Datos

La persistencia de los datos es la acción de preservar la información de una forma permanente, pero también se refiere a la capacidad de recuperar dicha información para volver a utilizarla. En este sentido; persistir los datos que son generados y utilizados de forma constante en los flujos del protocolo OAuth 2.0 toma relevancia dentro del diseño de la solución.

Realizar la validación de un token de acceso generado previamente es un claro ejemplo de la importancia de que se persistan los datos de la solución, para lo cual se decide por diseñar e implementar una base de datos con el objetivo de poder obtener un conjunto de datos y un conjunto de operaciones que permitan satisfacer los requerimientos de la solución.

La base de datos es del tipo relacional, es decir, se representa la información en un conjunto de tablas y cada una de ellas contiene atributos de diferentes categorías. El diseño de la base de datos posee características que proporcionan acceso eficaz a los datos, mantienen la integridad de la información a lo largo del tiempo y evita el almacenamiento de datos redundantes.

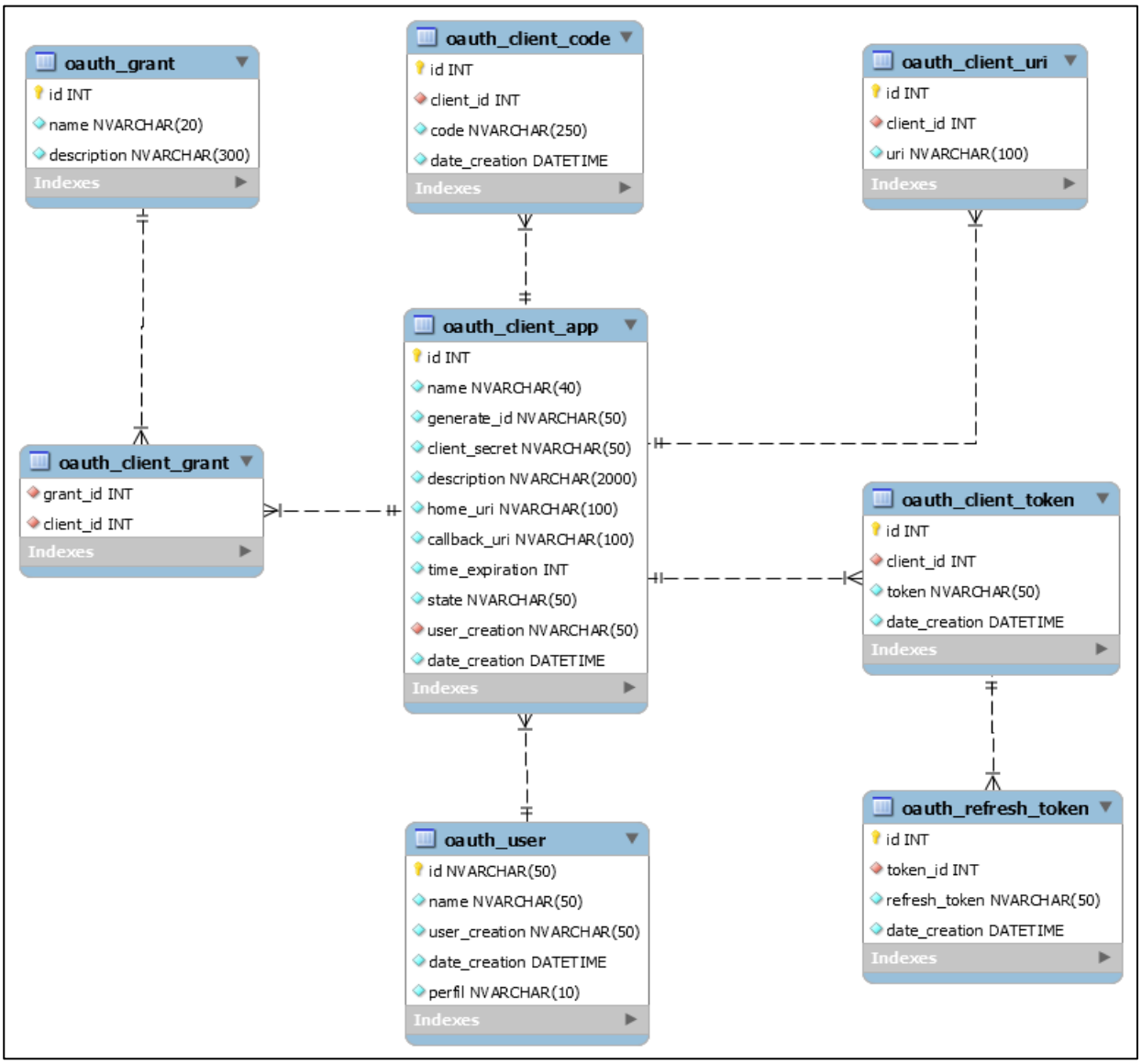

Figura 17 - Modelo Lógico de la base de datos de la solución

En la Figura 17 se muestra el modelo lógico de la base de datos, donde se refleja la estructura de la solución. Se puede visualizar que posee una tabla principal llamada *oauth_client_app* que corresponde a los datos de las aplicaciones clientes, aquellas aplicaciones que van a poder acceder a los

servicios. Se registran los datos de la aplicación cliente como nombre, descripción, URL principal de la aplicación, una URL callback que es utilizada para cuando se retorna el código de autorización, el tiempo que durará el token de acceso y dos códigos generados por el componente de utilidades que son el identificador y el código secreto que se le entrega a la aplicación. Además, posee el campo para el manejo de estados de la aplicación, creado, modificado o eliminado y campos de auditoria como usuario y fecha de creación. Los datos no se guardan encriptados.

La tabla *oauth_client_app* posee relaciones con otras tablas como por ejemplo la tabla de los códigos de autorización llamada *oauth_client_code* que se ve en la Figura 17. Una aplicación cliente puede tener muchos códigos de autorización, pero cuando se maneja la verificación de un token válido es contra el ultimo registro guardado, los demás registros quedan para auditoría. La misma relación se refleja con la tabla que almacena los tokens de acceso que se llama *oauth_client_token*.

Se tiene la tabla llamada *oauth_client_uri* para el control de las URLs de los servicios expuestos por la API y que una aplicación puede acceder, la tabla *oauth_user* sirve para la gestión de usuarios de la Aplicación Mantenedor de Políticas de Autorización, en esta tabla no se registra la contraseña y para la autenticación se utiliza el Componente de Acceso a Servicios LDAP. Finalmente, la tabla donde se almacena las concesiones llamada *oauth_grant* y la tabla para la relación entre aplicación y que concesión ha sido configurada llamada *oauth_client_grant*.

3.2.4 Tecnologías Utilizadas

Las siguientes son las tecnologías escogidas y a utilizar en la implementación de la solución. Como principio fundamental se ha elegido herramientas y librerías libres para su uso, este fue requerimiento de la empresa de seguros. A continuación, se describe las tecnologías a usar:

Java: Las aplicaciones del Servidor de Aplicaciones OAuth están desarrolladas con el lenguaje de programación Java. Java es un lenguaje orientado a objetos y que es independiente de la plataforma.

Esencialmente, este lenguaje ha sido escogido porque es soportado por el servidor de Aplicaciones JBoss, el mismo que es un requerimiento de la empresa de seguros.

Apache OLTU: Es un framework que ha sido creado para la implementación del protocolo OAuth 2.0 en Java. La idea de Apache OLTU es la construcción y

validación de las solicitudes. Aunque este framework maneja la funcionalidad como la verificación de los parámetros recibidos según el tipo de concesión, queda a cargo del desarrollador operaciones como por ejemplo el almacenamiento de la información, la administración del tiempo de expiración o la encriptación de los tokens.

RESTEasy: Framework para construcción de servicios Web RESTful. El servidor de aplicaciones JBoss incluye entre sus módulos el proyecto RESTEasy y por ende se tiene el soporte para la generación de servicios Web ligeros. Otra ventaja es que puede usar el almacenamiento en caché para URLs o consultas y que es de gran utilidad cuando existe un alto volumen de transacciones. Es utilizada en la solución para exponer los servicios que será consumidos por las aplicaciones de terceros.

Spring Framework/Hibernate: Proporciona un modelo completo de programación y configuración para aplicaciones J2EE (Java 2, Enterprise Edition), en cualquier tipo de plataforma de implementación. Se utiliza en las tres aplicaciones que se encuentran en el Servidor de Aplicaciones OAuth y su principal uso es con Spring ORM (Object Relational Mapping), que permite la gestión de recursos e implementación de objetos para el acceso a la base de datos. Como marco de mapeo de objetos y relaciones, Hibernate se ocupa de la persistencia de los datos. Ayuda a mantener un código más fácil de entender en cuanto al trato con los objetos de la base de datos.

AngularJS: Framework que permite crear vistas dinámicas para aplicaciones Web, se utiliza en la aplicación mantenedor de políticas de seguridad. Su principal ventaja es la creación de aplicaciones pensadas con el objetivo de lograr la mayor fluidez posible en UX (User eXperience), esto se logra haciendo que la comunicación entre cliente y servidor se realice de forma transparente al usuario, logrando que se tenga la sensación de no abandonar nunca la página principal de la aplicación.

Apache Maven: Es una herramienta de software para la gestión y construcción de proyectos en Java. Posee la capacidad de realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Es decir, hace posible la creación de software con dependencias incluidas dentro de la estructura del proyecto. El uso de Apache Maven mejora el mantenimiento y la reusabilidad en los proyectos.

A continuación, en el capítulo IV se describirá la implementación de los componentes, su reusabilidad, la codificación de las aplicaciones y los resultados obtenidos.

Capítulo IV: Implementación y Resultados Obtenidos

En el presente capítulo se describe la implementación de la solución, la codificación de la Aplicación Servidor de Autorización, Aplicación Servidor de Recursos y la Aplicación Mantenedor de Políticas de Autorización. Se expone el detalle de la organización del código y descripción del código relevante. También se describe las consideraciones para la codificación de la librería que se utiliza en las aplicaciones móviles junto con su instalación.

4.1 Servidor de Aplicaciones OAuth

El Servidor de Aplicaciones OAuth consta de tres aplicaciones como se mencionó en el capítulo anterior. Se decide desarrollar algunos componentes como proyectos independientes por su reusabilidad en las aplicaciones. Estos componentes son descritos a continuación y una vez que se tiene claro la funcionalidad de cada uno de ellos se describen las aplicaciones tanto en funcionalidad como en codificación.

4.1.1 Componente de Utilidades

Su objetivo principal es la implementación de funcionalidades comunes como el manejo de errores con descripciones concretas y configurables. Otra funcionalidad es proveer de los métodos de encriptación de los datos sensibles como también la definición de la forma para crear un token de acceso, de actualización o código de autorización y sus validaciones.

En la Figura 18 se visualiza la estructura del proyecto junto con las clases que la componen. OauthException.java representa el objeto que maneja las condiciones anómalas que podrían ocurrir. Security.java y Aesconfig.java sirven para realizar la encriptación de los parámetros, se usa el algoritmo de cifrado simétrico AES (Advanced Encryption Standard).

Por último, se tienen las clases Constants.java que posee los atributos que no cambian entre todas las aplicaciones, por ejemplo, se tienen las constantes para manejar la descripción de mensajes de error y códigos de error. La clase Utils.java que posee métodos para transformación de datos y la clase Validator.java posee métodos para realizar las comparaciones entre tokens. Como resultado se tiene un proyecto Java con un identificador de artefacto *oauth2Util* para Maven. Este componente es utilizado en las tres aplicaciones del Servidor de Aplicaciones OAuth.

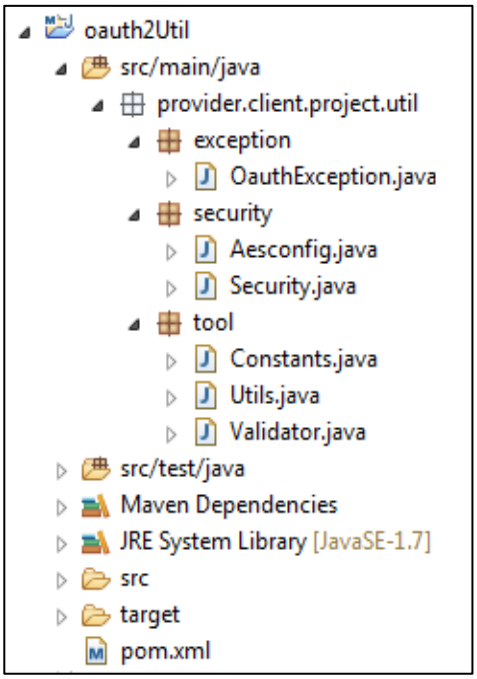

Figura 18 - Componente de Utilidades - Estructura de archivos

4.1.2 Componente Lógico de Acceso a Datos

La finalidad de este componente es proveer una interfaz que las aplicaciones puedan utilizar para manipular el conjunto de datos que representa la información almacenada. Incluye código que crea una conexión a la base de datos y código mediante el cual se puede realizar las operaciones de ingreso, consulta, modificación y eliminación de datos.

Se ha implementado el patrón de diseño Data Access Object (DAO) que se basa en los principios de diseño de abstracción y encapsulación de Java. La ventaja de usar objetos de acceso a datos se da porque los objetos de la lógica de negocio no requieren conocimiento directo del destino final de la información que manipula. Junto con el framework Spring y también Hibernate se logra detallar cómo es el modelo de datos y qué relaciones existen, se realiza un mapeo entre los objetos de este componente y las tablas de la base de datos con el objetivo de que se pueda mover la información de un lado a otro.

El mapeo se efectúa mediante anotaciones Java que se codificaron manualmente. Una anotación Java es una forma de añadir metadatos al código fuente y que están disponibles para la aplicación en tiempo de ejecución. Esto permite indicarle al programa cómo debe comportarse. El detalle técnico de los tipos de datos, longitudes de los campos y de las tablas de la base de datos se encuentra en el Anexo (C).

En la Figura 19 se muestra parte del archivo *pom.xml*, en este archivo se incluyen las librerías de las que depende un componente. Dentro de estas librerías se encuentra el driver de conexión para la base DB2, librerías del framework Spring y también las librerías para Hibernate.

```
<!-- DB2 -->
<dependency>
        <groupId>com.ibm.db2.jcc</groupId>
        <artifactId>db2jcc</artifactId>
        <version>3.8.47</version>
</dependency>
<!-- Spring framework -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.1.RELEASE</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>3.1.1.RELEASE</version>
</dependency>
<!-- Spring AOP dependency -->
<dependency>
        <groupId>cglib</groupId>
        <artifactId>cglib</artifactId>
        <version>2.2</version>
</dependency>
<!-- Hibernate -->
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>3.6.9.Final</version>
</dependency>
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>3.6.9.Final </version>
</dependency>
<!-- Hibernate library dependency start -->
<dependency>
        <groupId>dom4j</groupId>
        <artifactId>dom4j</artifactId>
        <version>1.6.1</version>
</dependency>
<dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.2.2</version>
</dependency>
```

Figura 19 - Componente Lógico de Acceso a Datos - Dependencias

El componente tiene un archivo de configuración llamado *Spring.xml* donde se encuentran las definiciones para la base de datos .En este archivo se maneja atributos como el nombre de la conexión a la base de datos, este nombre está configurado en el servidor. También en el archivo se encuentra el nombre de los objetos que son utilizados para el mapeo con las tablas de la base de datos.

En la Figura 20 se observa la estructura del proyecto. Dentro del paquete *provider.client.project.admin.model* existen las clases que representan las tablas de la base de datos por mencionar un ejemplo la clase *Client.java* representa a la tabla *oauth_client_app* de la Figura 19. Esta clase tiene el mapeo de sus atributos con los campos de la tabla. Bajo el paquete con nombre

provider.client.project.admin.dao se encuentra las clases que poseen los métodos que se pueden ejecutar en la base de datos, algunos de los métodos son los tradicionales guardar, eliminar y consultar.

La clase *ServiceLocator.java* permite que la configuración como la conexión a la base de datos pueda ser cargada una sola vez. El resultado de este componente es un proyecto java empaquetado utilizando Maven, el identificador del artefacto es *oauth2DAO*. Este componente es utilizado en las tres aplicaciones del Servidor de Aplicaciones OAuth.

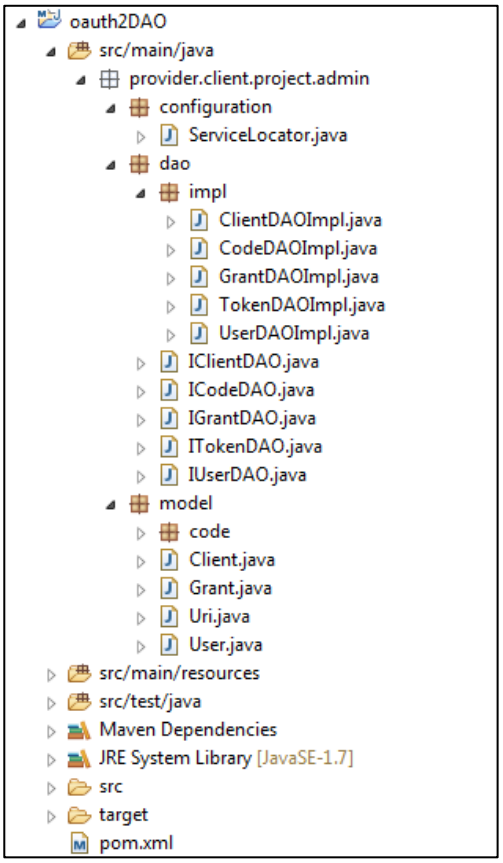

Figura 20 - Componente Lógico de Acceso a Datos - Estructura de Archivos

4.1.3 Componente de Acceso a Servicios LDAP

Este componente es un proyecto pequeño que se encarga de validar el usuario y contraseña contra el directorio de usuarios de la empresa de seguros. Este componente valida a usuarios que son clientes y a usuarios internos. En la Figura 21 se puede apreciar que consta de la clase LdapService.java que tiene la lógica para invocar al servicio en el Servidor LDAP y hacer las validaciones correspondientes. El paquete *provider.client.project.webservice* tiene la implementación del servicio Web que usa para averiguar el estado y tipo de

usuario para la empresa. En el archivo config.xml se encuentra los datos para la conexión con el servidor LDAP como por ejemplo la URL y el puerto.

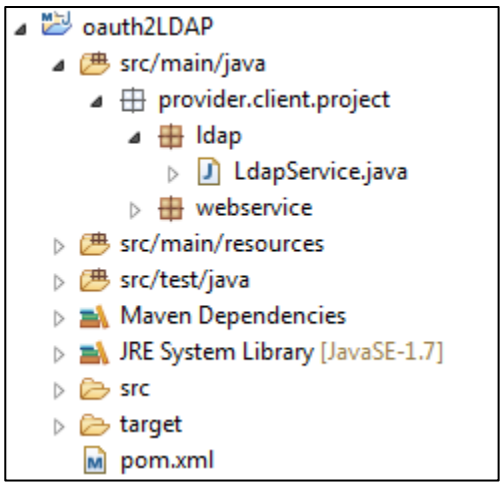

Figura 21 - Componente de Acceso a LDAP - Estructura de archivos

El resultado es un proyecto Java empaquetado con Maven y con el identificador *oauth2LDAP*. Este componente se utiliza en la Aplicación Servidor de Autorización y en la Aplicación Mantenedor de Políticas de Autorización.

4.1.4 Aplicación Mantenedor de Políticas de Autorización

La funcionalidad principal es agregar las aplicaciones clientes que pueden tener acceso a los recursos compartidos, para su implementación se utiliza el patrón de arquitectura Modelo Vista Controlador. El registro de una aplicación de terceros permite obtener las credenciales de la aplicación cliente como identificador de la aplicación y el código secreto, que se utilizan para autenticar solicitudes hechas al Servidor de Autorización.

Estas credenciales son fundamentales para proteger la autenticidad de las solicitudes al realizar operaciones tales como solicitar un token de acceso o actualizar tokens de acceso. El registro de una aplicación de terceros también almacena información para mejorar la experiencia del usuario durante el proceso de autorización, se almacena el nombre de la aplicación como la URL de la página principal de la aplicación y que se puede mostrar al momento de solicitar autorización.

La parte de la vista está compuesta por la estructura que se visualiza en la Figura 22. Existe un archivo principal llamado *index.html* que maneja el inicio de sesión o mostrar la página de inicio en caso de que el usuario ya haya ingresado al sistema. El archivo *client.html* se encarga de las operaciones que un usuario puede realizar con las aplicaciones de terceros mientras que el

archivo *user.html* se encarga de las operaciones de los usuarios que pueden usar esta aplicación.

Los archivos que manejan las peticiones que realiza un usuario se encuentra dentro del directorio *scripts*, mientras que los estilos de la aplicación se encuentran en el archivo *main.css*.

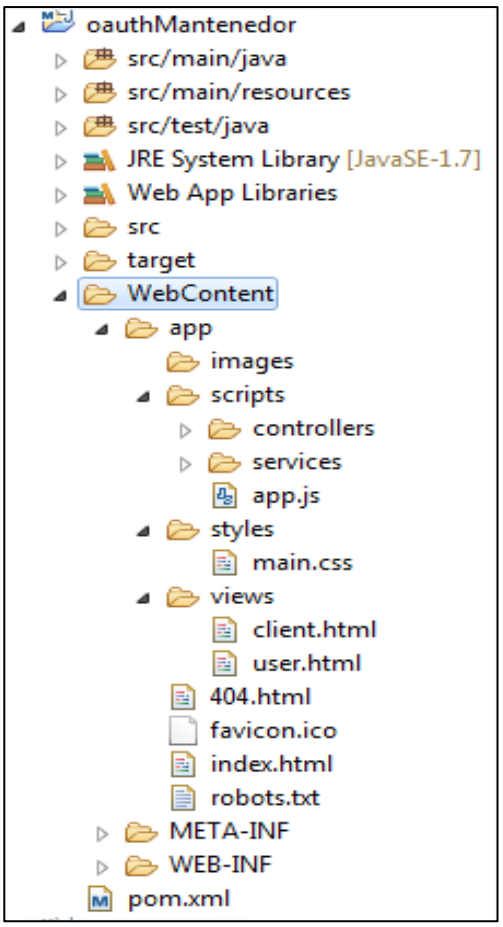

Figura 22 - Aplicación Mantenedor de Políticas de Autorización - Archivos para la Vista

El backend de esta aplicación está en el mismo proyecto, dentro de las carpetas *src/java/main*. En la Figura 23 se muestra el backend y se tiene que los controladores están representados por las clases bajo el paquete con el nombre *provider.client.project.core.endpoint*. Estas clases se encargan de la revisión y validación de los parámetros recibidos, de manejar la información y poder armar los objetos del modelo de datos que usa el Componente de Acceso a Datos. Las clases bajo el paquete *provider.client.project.core.pojo* son *FormUser.java* que genera un objeto con los campos para el inicio de sesión y *GenericResponse.java* que sirve para crear las respuestas que se envían a la vista. La clase *InitApp.java* bajo el paquete *provider.client.project.core.servlet*

sirve para cargar configuraciones iniciales del proyecto como por ejemplo la ruta de los logs.

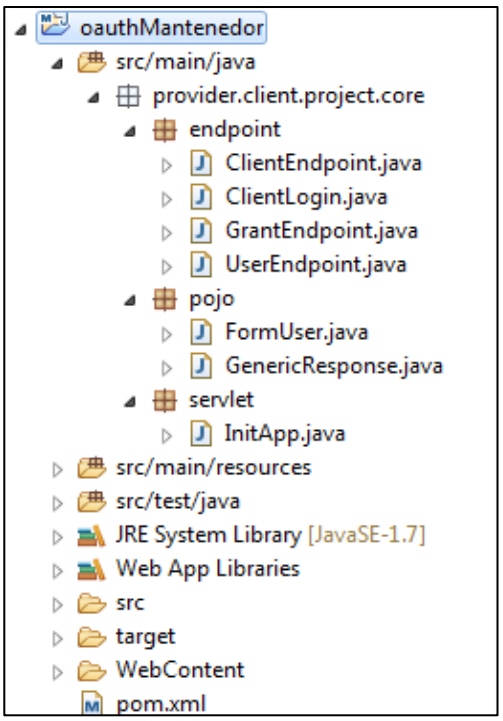

Figura 23 - Aplicación Mantenedor Políticas de Autorización - Estructura de Archivos Backend

Las vistas principales se muestran en las Figuras 24, 25 y 26. Las Figuras tienen ofuscado el logo de la empresa de seguridad y se visualizan datos de prueba no reales por temas de confidencialidad. La Figura 24 tiene la presentación del inicio de sesión para ingresar a la Aplicación Mantenedor de Políticas de Autorización y solo admite usuarios internos de la empresa de seguros que hayan sido dados de alta.

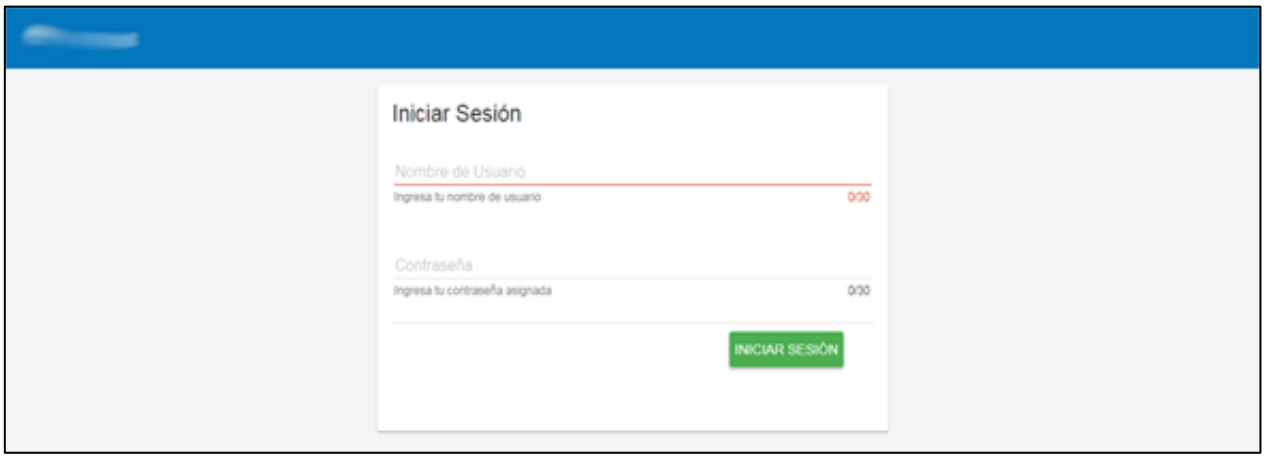

Figura 24 - Aplicación Mantenedor de Políticas de Autorización - Vista Inicio de Sesión

Figura 25 - Aplicación Mantenedor de Políticas de Autorización - Vista Página de Inicio

La Figura 25 presenta la página inicial de la aplicación, se puede visualizar las aplicaciones de terceros que se encuentran registradas. Además, permite agregar una nueva aplicación de terceros.

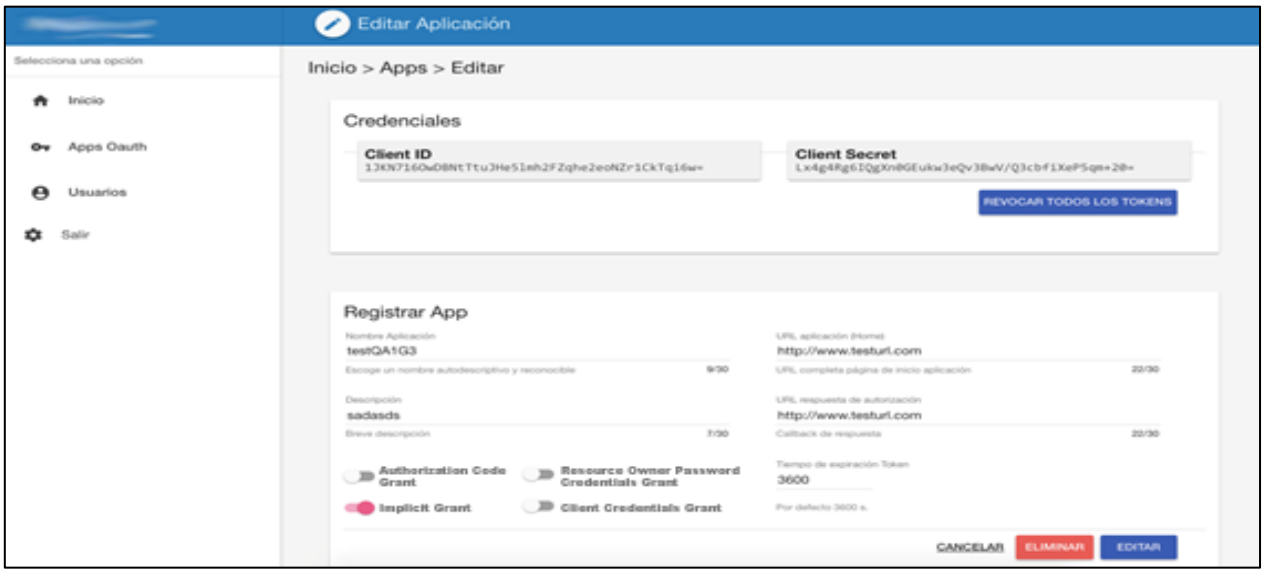

Figura 26 -Aplicación Mantenedor de Políticas de Autorización - Vista Crear/Editar Aplicación Cliente

La Figura 26 muestra los campos para crear o editar una aplicación de terceros, dentro de credenciales está el campo identificador de aplicación de terceros y el código secreto de la aplicación. Estos datos son autogenerados una vez que se crea una aplicación y no se pueden editar. Los otros campos son las características de una aplicación de terceros, se puede observar el nombre y la URL principal de la aplicación, la URL de respuesta del código de autorización, una breve descripción, el tipo de concesión que utilizará para

obtener el token de acceso y el tiempo que puede durar el token de acceso generado.

4.1.5 Aplicación Servidor de Autorización

La Aplicación Servidor de Autorización tiene como definición principal dos interfaces para permitir las cuatro concesiones del protocolo OAuth 2.0:

Authorization Endpoint: Esta interfaz debe ser la encargada de validar las aplicaciones clientes y a los propietarios de los recursos y así generar el código de autorización.

Token Endpoint: Esta interfaz se encarga de proveer tokens de acceso, normalmente a cambio de códigos previamente expedidos por el Authorization Endpoint, también tiene el método para la anulación del token.

En la Figura 27 se muestra la estructura de los archivos del proyecto para la Aplicación Servidor de Autorización. Se observa las dos interfaces mencionadas *AuthEndpoint.java* y *TokenEndpoint.java* y que se encuentran bajo el paquete *provider.client.project.oauth.core.endpoint*. Las clases *AuthLogic.java* y *TokenLogic.java* poseen la lógica de negocio y las clases bajo el paquete *provider.client.project.oauth.core.endpoint*.response son destinadas para formar la respuesta a las peticiones.

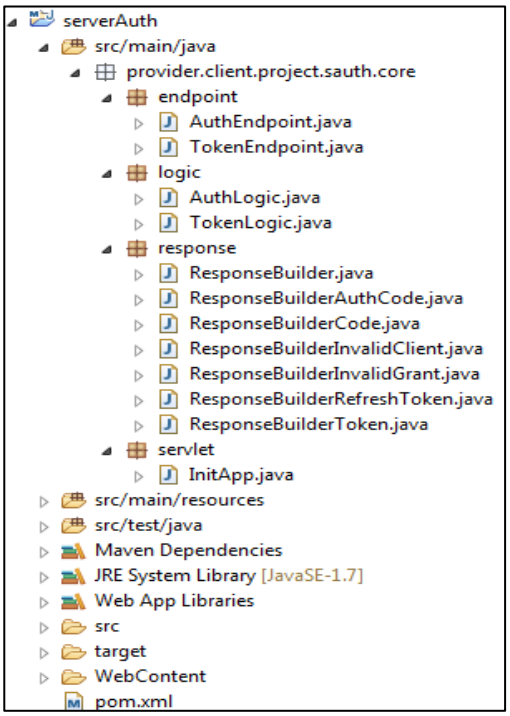

Figura 27 - Servidor de Autorización - Estructura de Archivos

Para la implementación se usaron nombres de los archivos que guarden relación entre sí, tanto para la capa de servicios como para la capa lógica con las interfaces de esta aplicación. En la Figura 28 se observa el diagrama de secuencia para la solicitud de código de autorización y se describe a continuación.

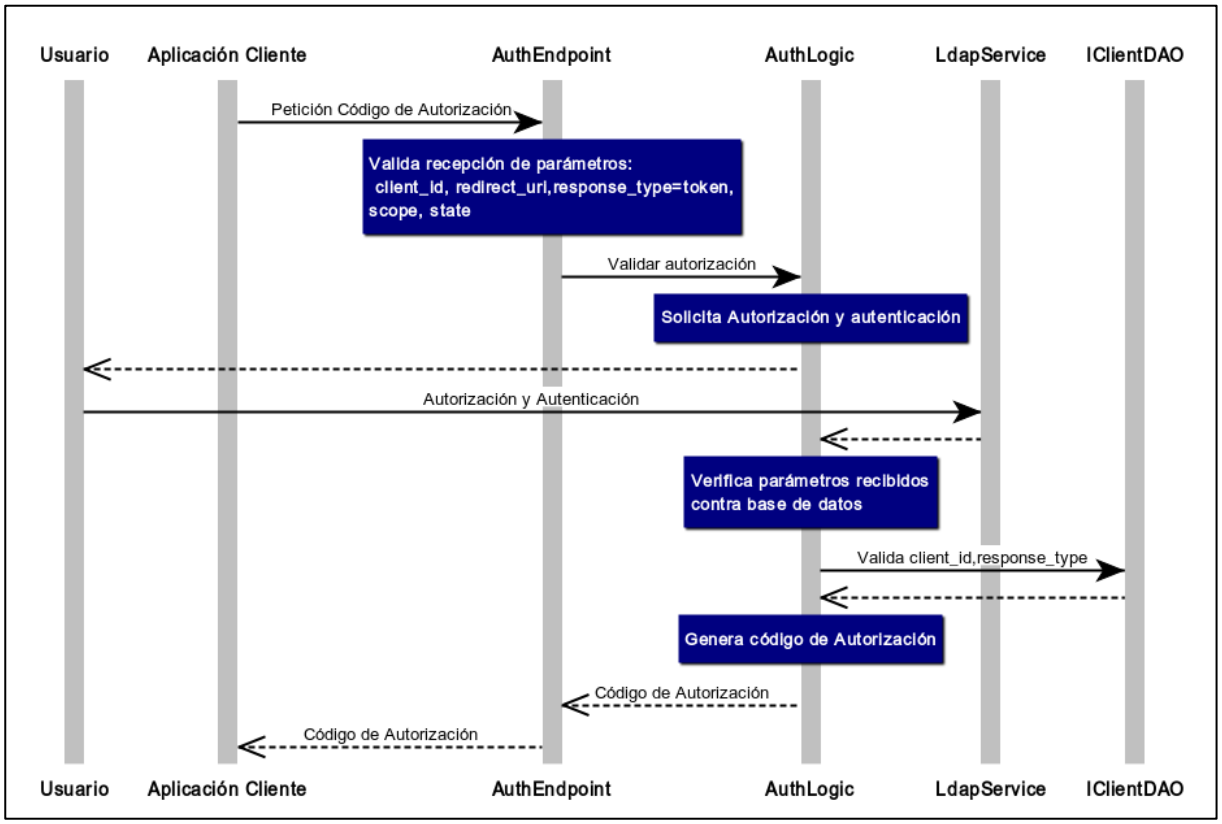

Figura 28 - Aplicación Servidor de Autorización – Authorization Endpoint

- La petición llega a la interfaz de código de autorización donde se realiza un primer análisis con el framework Apache OLTU. Este se encarga de verificar que lleguen los datos necesarios en la petición.
- Si la petición es correcta, la aplicación realiza como lógica de negocio las validaciones para saber si el propietario del recurso ha autorizado el acceso.
- En caso de no existir autorización se procede autenticar al propietario del recurso y pedirle su autorización.
- Si se ha autorizado se procede a validar los datos de la petición con respecto a lo almacenado en la base de datos.
- Comprobado la validez de los datos proporcionados, se genera el código de autorización.

- La respuesta a la petición se da en sentido inverso con los datos generados.

En la Figura 29 se observa el diagrama de secuencia para la solicitud de token de acceso y se describe a continuación.

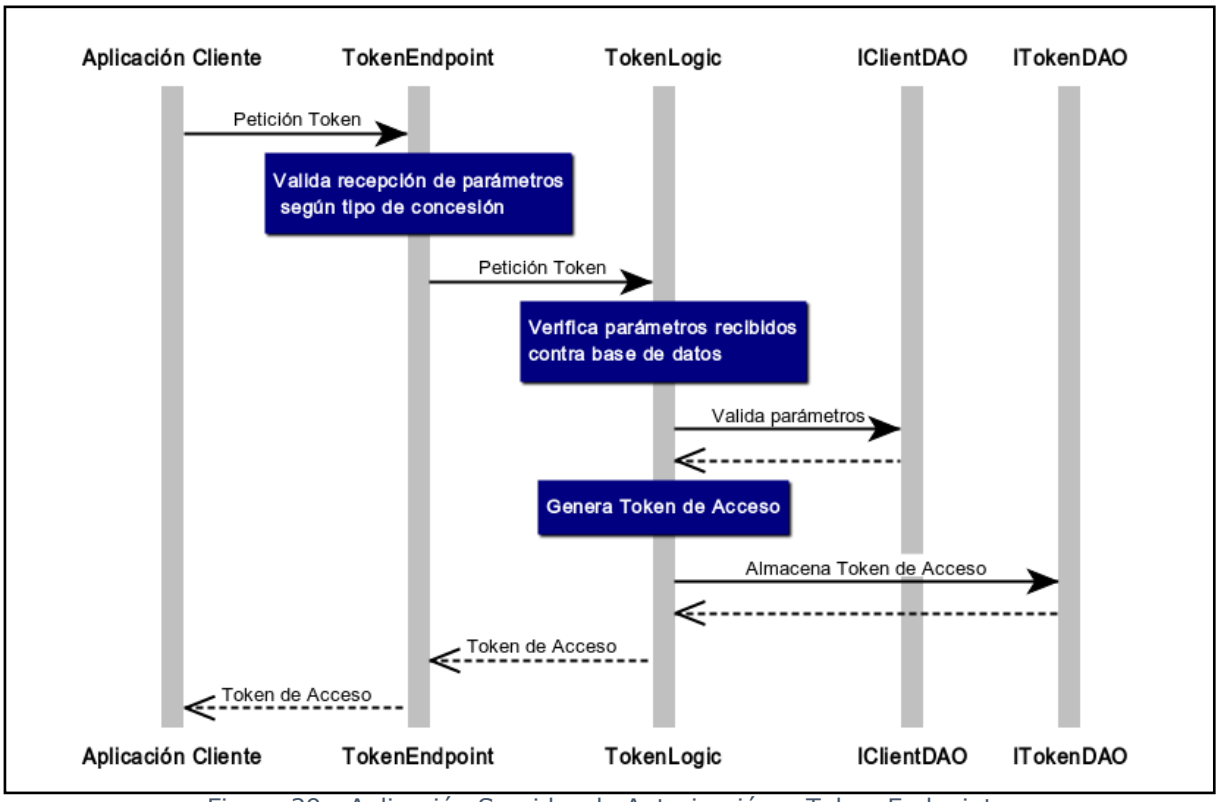

Figura 29 - Aplicación Servidor de Autorización – Token Endpoint

- La petición llega a la interfaz de petición de token de acceso, se realiza un primer análisis con el framework Apache OLTU, este se encarga de verificar que lleguen los datos necesarios en la petición.
- Se procede a validar los datos de la petición con respecto a lo almacenado en la base de datos.
- Comprobado la validez de los datos proporcionados, se genera el token de acceso y se almacena en la base de datos.
- La respuesta a la petición se da en sentido inverso con los datos generados.

Mediante el uso de anotaciones en las clases *AuthEndpoint.java* y *TokenEndpoint.java* se indica las características del servicio y el framework RESTEasy se encarga de crearlo y exponerlo en la aplicación. El uso de una anotación se puede ver en la Figura 30, se utiliza para el método de obtención de token dentro de la clase *TokenEndpoint.java* y la anotación indica como

debe ser la petición. El método HTTP del servicio es POST en la línea 56, mientras que los datos que se van a recibir son de la estructura de un formulario web y se ve en la línea 57. La respuesta a la petición es en formato JSON mediante la anotación de la línea 58. JSON es una estructura para la representación de la información.

```
55
56      @POST
57      @Consumes("application/x-www-form-urlencoded")
58      @Produces("application/json")
59      public Response token(@Context HttpServletRequest request)
```

Figura 30 - Servidor de Autorización - Anotaciones para el método obtención de Token

Las clases *AuthLogic.java* y *TokenLogic.java* tienen la implementación de la lógica de negocio. Aquí se invocan a los componentes que se explicaron al principio de este capítulo como por ejemplo el Componente de Acceso a Datos para almacenar el token de acceso generado por la aplicación o al Componente de Acceso a Servicios LDAP para realizar la autenticación del usuario en la aplicación.

Se han creado clases bajo los nombres de los posibles tipos de respuesta, es decir, si la validación encuentra que se está usando un código secreto de cliente que no es válido, la respuesta se basa en la clase *ResponseBuilderInvalidClient.java*, indicando el tipo del error y la descripción exacta, como se muestra en la Figura 31.

```
15  public class ResponseBuilderInvalidClient extends ResponseBuilder {
16
17      @Override
18      public Response buildResponse(HttpServletRequest request, Client client)
19              throws OAuthSystemException, OAuthProblemException, URISyntaxException {
20
21          return ResponseBuilder.buildInvalidResponse(OAuthError.TokenResponse.INVALID_CLIENT,
22                  Constants.INVALID_CLIENT_DESCRIPTION);
23      }
24  }
```

Figura 31 - Servidor de Autorización - Clase ResponseBuilderInvalidClient.java

Las dependencias para la Aplicación Servidor de Autorización se observan en la Figura 32 del archivo, se tienen las librerías de Apache OLTU. También se configura una librería para el manejo de logs. La aplicación realiza escritura de logs de las acciones realizadas, escribiendo en un archivo el paso a paso de las sentencias ejecutadas y ofuscando la información confidencial como es el caso del token de acceso generado.

```
<!-- Servlet -->
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
</dependency>
<!-- Apache OLTU -->
<dependency>
        <groupId>org.apache.oltu.oauth2</groupId>
        <artifactId>org.apache.oltu.oauth2.common</artifactId>
        <version>1.0.2</version>
</dependency>
<dependency>
        <groupId>org.apache.oltu.oauth2</groupId>
        <artifactId>org.apache.oltu.oauth2.authzserver</artifactId>
        <version>1.0.2</version>
</dependency>
<!-- Logs -->
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.16</version>
</dependency>
```

Figura 32 - Servidor de Autorización - Dependencias

4.1.6 Aplicación Servidor de Recursos

La Aplicación Servidor de Recursos en estructura y configuración es similar a la Aplicación Servidor de Autorización. Posee dos interfaces que representan los recursos compartidos por parte de la empresa y que solamente aplicaciones que ya poseen el token de acceso pueden solicitar. En la Figura 33, se muestra la estructura de archivos del proyecto y a continuación su descripción.

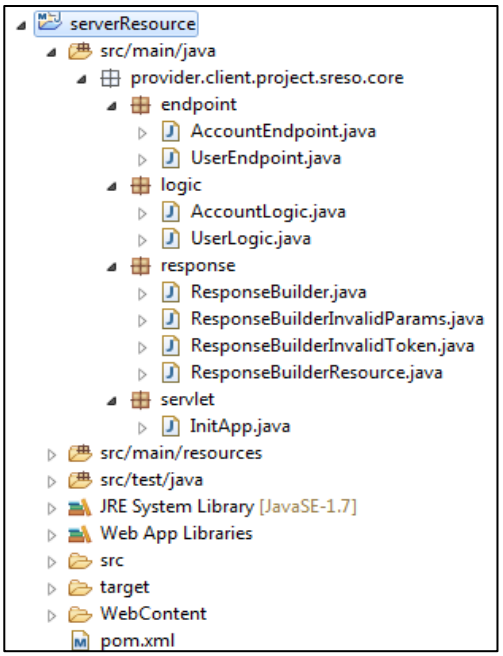

Figura 33 - Servidor de Recursos - Estructura de Archivos

Las interfaces de la aplicación que permiten acceder a los servicios de la API son representadas por la clase *UserEndpoint.java* para consultar los datos del propietario del recurso y la clase *AccountEndpoint.java* que tiene los métodos para consultar los datos del estado de cuenta y modificar el tipo de cuenta del usuario.

Al igual que el Servidor de Autorización, esta aplicación posee clases que manejan la lógica de negocio y que han sido nombradas de forma similar a las interfaces para mantener una relación, las clases UserLogic.java y AccountLogic.java tienen lógica para la llamada a los servicios de la API y también se valida el token de acceso en la base de datos. Además, hay clases para la generación de las respuestas a las peticiones

El flujo de la aplicación se muestra en la Figura 34, *UserEndpoint* y *AccountEndpoint* están referenciados por *ResourceEndpoint* de igual manera sucede con *UserLogic* y *AccountLogic* con *ResourceLogic* en la Figura.

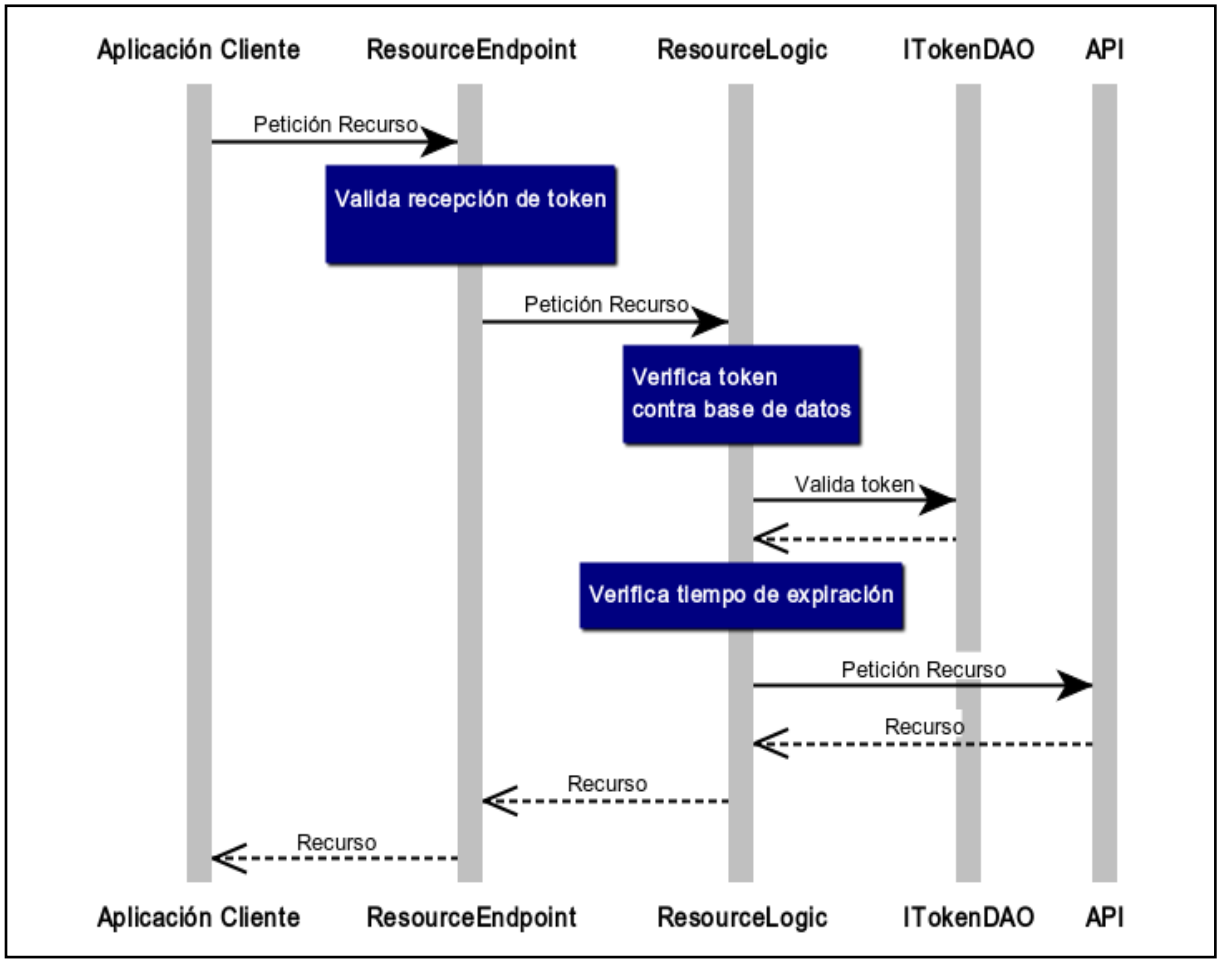

Figura 34 - Aplicación Servidor de Recursos

- La petición llega a uno de las interfaces de los recursos donde se realiza un primer análisis. Este se encarga de verificar que dentro de la petición llegue el parámetro que pertenece al token de acceso.
- Si la petición es correcta, la aplicación realiza como lógica de negocio las validaciones que corresponden a la verificación de los parámetros de cada servicio.
- Una vez verificado los parámetros se comprueba con la base de datos que el token de acceso sea el correcto.
- Se verifica que no ha expirado el token de acceso y que se tiene acceso al recurso solicitado.
- Comprobado la validez del token de acceso se realiza el llamado al API de servicios para solicitar el recurso compartido.
- La respuesta a la petición se da en sentido inverso con los datos de respuesta de la API de servicios.

4.2 Librería para Aplicaciones Móviles

Esta sección describe la implementación para las aplicaciones instaladas y ejecutadas en un dispositivo móvil que es utilizado por el propietario del recurso. Para acceder a las interfaces del servidor de autorización se requiere de la interacción entre la aplicación y el agente de usuario del dispositivo. Un agente de usuario es una aplicación informática que funciona como cliente en un protocolo de red; el nombre se aplica generalmente para referirse a aquellas aplicaciones que acceden a la World Wide Web.

Se ha decidido implementar la incorporación de un agente de usuario dentro de la aplicación utilizando Webview, que significa poder utilizar el navegador de forma embebida en la aplicación. Tanto iOS como Android soportan esta funcionalidad y ofrece una mejor usabilidad, ya que elimina la necesidad de cambiar de contexto y abrir ventanas de explorador durante el flujo para obtener el token de acceso.

Este proyecto es solo para construir la librería que se utiliza en las aplicaciones móviles de la plataforma Android y tiene la estructura de la Figura 35. Consta de 3 clases Java que se describen a continuación:

ClientConfiguration.java: Sirve para poder instanciar un objeto que posee las características de configuración para el protocolo OAuth 2.0 como, por ejemplo, el identificador de la aplicación, el código secreto de la aplicación la URI de redirección.

OAuthClient.java: Posee la lógica para el llamado a los puntos finales. Tiene los métodos para solicitar al servidor de autorización el token de acceso y también para solicitar los recursos compartidos.

OAuthConstants.java: Tiene las constantes para el manejo de los mensajes y nombre de parámetros.

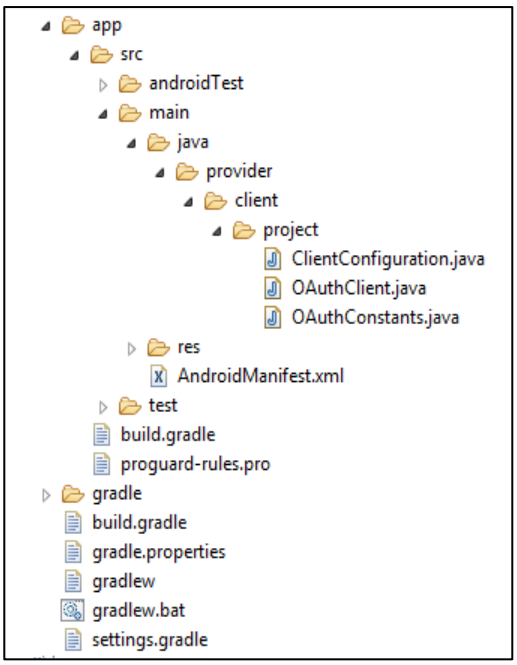

Figura 35 - Librería Android - Estructura de Archivos

Este proyecto permite generar la librería *oauthclient.aar*, el archivo generado se debe incluir dentro de la carpeta *libs* del proyecto de la aplicación cliente movil.

Para poder utilizar los métodos de la librería se agrega como dependencia en la configuración de la aplicación móvil. Se debe construir un Intent como se puede ver en la línea 11 de la Figura 36. Un Intent es una descripción abstracta de una operación a realizar. Se configura el Intent creado como se ve en la línea 13. Un Intent se puede iniciar como una actividad en Android, esto se muestra en la línea 16.

```
10  //Setting-up Intent with OauthClient.class from Oauth Library
11  Intent iOauth = new Intent(this,OauthClient.class);
12  //Add Configuration
13  i.putExtra(OAuthConstants.OAUTHCONFIG, new ClientConfiguration("<YOUR_CLIENT_ID>",
14  "<YOUR_CLIENT_SECRET>","<YOUR_REDIRECT_URI>","<YOUR_STATE>"));
15  //Init Activity
16  startActivity(iOauth,<YOUR_ID_INTENT>);
```

Figura 36 - Uso de Librería en Android - Creación Intent

En la Figura 37 se muestra el código que se debe implementar en la aplicación móvil cliente. El método en Android onActivityResult en la línea 21 sirve para una vez finalizado el flujo de la concesión se pueda obtener el token de acceso.

```
21  public void onActivityResult(int requestCode, int resultCode, Intent data){
22      super.onActivityResult(requestCode, resultCode, data);
23      TextView text = (TextView) findViewId(R.id.text);
24      if( requestCode == <YOUR_ID_INTENT>){
25          if( resultCode == RESULT_OK){
26              String error = data.getStringExtra(OAuthConstants.TOKEN);
27              /*
28               *  Save the token and use it
29               */
30          }else if( resultCode == RESULT_CANCELED){
31              String error = data.getStringExtra(OAuthConstants.ERROR);
32              /*
33               *  Do something with the error.
34               */
35          }
36      }
37  }
```

Figura 37 - Uso de Librería en Android - Obtener token de acceso

Los desarrolladores de las aplicaciones de terceros que integren la librería de OAuth deben decidir cómo guardar y usar el token de acceso obtenido, de igual manera cuando exista un error qué acción deben realizar.

4.3 Resultados de la Implementación

Como resultado del proceso de implementación se tienen cuatro piezas de software para la empresa de seguros. Tres de ellas se han instalado en un mismo servidor físico por decisión de la empresa, la Aplicación Servidor de Autorización, la Aplicación Servidor de Recursos y la Aplicación Mantenedor de Políticas de Autorización. Mientras que la librería Android ha sido integrada con una aplicación de terceros que ha sido liberada en una versión de prueba y no disponible para el mercado.

La Aplicación Servidor de Recursos posee los siguientes servicios de la API:

- Consulta de información básica de un usuario.
- Consulta información del estado de cuenta de un usuario.
- Modificación del tipo de fondo de una cuenta de un usuario.

En la Aplicación Mantenedor de Políticas de Autorización se ha configurado a la aplicación de terceros para que pueda realizar el acceso a dos de los tres servicios que ofrece el Servidor de Recurso. Esta aplicación no podrá realizar la modificación del tipo de fondo de la cuenta del usuario.

Para dar cumplimiento a las consideraciones de seguridad en la implementación del protocolo OAuth 2.0, se cumple con lo siguiente:

- Se utiliza el protocolo TLS. La empresa de seguros se encargó de la generación de los certificados e instalación en el Servidor de Aplicaciones OAuth.
- La Aplicación Servidor de Autorización se encarga de autenticar al usuario utilizando los servicios expuestos por el Servidor de Autenticación (LDAP).
- Los tokens generados son autocontenidos y utilizan el cifrado AES para ser generados. Su duración depende de la configuración en la Aplicación Mantenedor de Políticas de Autorización y se recomienda que no supere los 3600 segundos, que es un valor definido en conjunto con la empresa de seguros.

En el siguiente capítulo se menciona la evaluación de la solución, las pruebas que se han realizado con la aplicación de terceros en versión de prueba y el análisis de los aspectos funcionales y de calidad de la solución.

Capítulo V: Evaluación de la Solución

Este capítulo posee el análisis realizado con respecto a la evaluación de la solución una vez implementada y en funcionamiento. Se detallan las pruebas realizadas desde los puntos de vista de funcionalidad y calidad.

Aunque se encuentran implementadas los 4 tipo de concesiones, se ha configurado y probado exhaustivamente solo la concesión código de autorización. Al no contar en el momento con los recursos necesarios como por ejemplo una aplicación web de terceros o una aplicación interna, las otras concesiones han sido evaluadas hasta el nivel de pruebas unitarias por decisión de la empresa de seguros.

5.1 Evaluación del Funcionamiento

Con la aplicación de terceros que se encuentra integrada con la librería, se procede a verificar el correcto funcionamiento de la solución y el cumplimiento de los objetivos de calidad para lo cual se ha establecido las siguientes pruebas de manera general y que se muestran en la Tabla 2.

Nombre	Resultado Esperado
Prueba de Autenticación	El usuario de la aplicación de terceros debe poder ingresar a la aplicación con la cuenta de la empresa de seguros utilizando usuario y contraseña actual almacenada en el Servidor de Autenticación (LDAP).
Prueba de Autorización	La aplicación de terceros debe ser capaz de acceder a los servicios de los recursos compartidos que han sido previamente configurados en el mantenedor de políticas de autorización.
Prueba de Funcionalidad Concesión Código de Autorización	El flujo realizado para la obtención del token de acceso debe ser de la concesión código de autorización, se espera que se siga el flujo correctamente y mediante el token obtenido se acceda a los servicios configurados.
Cumplimiento Calidad	Se espera cumplir con los objetivos no funcionales establecidos como el bajo acoplamiento entre las aplicaciones, el manejo apropiado de los errores, que se puede agregar más recursos compartidos a la solución y la evaluación de los tiempos de respuesta.

Tabla 2 - Pruebas Generales en Aplicación de terceros

5.1.1 Pruebas de Autenticación

En primera instancia, se describen las pantallas y lógica de la aplicación de terceros de prueba y luego se detallan las pruebas realizadas y sus resultados. La presentación inicial de la aplicación de terceros posee un botón para hacer el inicio de sesión utilizando los datos de la empresa de seguros como se puede visualizar en la Figura 38, una vez presionado el botón se presenta la pantalla para ingresar las credenciales de la empresa de seguros, es decir su usuario y contraseña, esta pantalla se encuentra en la Figura 39.

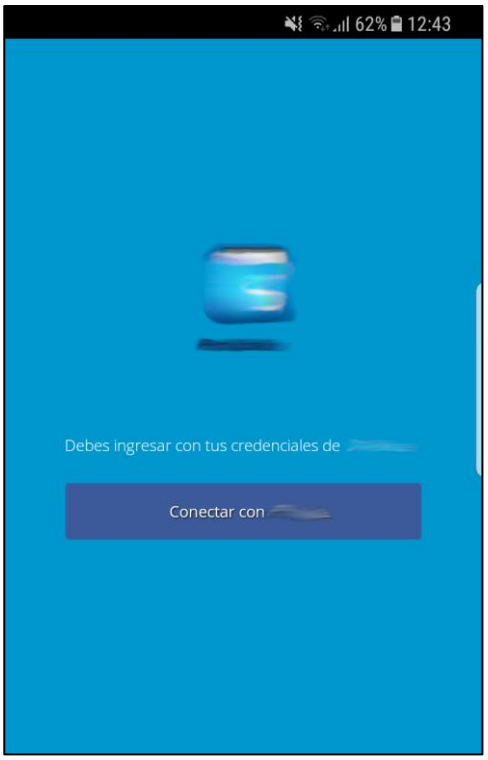

Figura 38 - Pantalla Inicial Aplicación de Terceros

Figura 39 - Pantalla de ingreso de Credenciales

Después de haber ingresado los datos correspondientes se pueden dar tres situaciones con respecto a la validación de los datos de autenticación, estos casos son:

1. Usuario y contraseña son válidos contra el servidor de autenticación.
2. Usuario y/o contraseña no válidos para el servidor de autenticación.
3. Usuario no existe en el servidor de autenticación.

Para los casos 2 y 3 se muestra un mensaje de error al usuario indicándole lo que sucede, como se muestra en la Figura 40.

Figura 40 - Pantalla de ingreso – Mensaje de error

Para el caso 1 cuando el usuario y la contraseña es correcta, la aplicación de terceros empieza la negociación para obtener el token de acceso mediante el flujo descrito para la concesión código de autorización y que fue detallado en el Capítulo 4.1.5.

Las pruebas han sido realizadas por 5 personas en total, 4 son usuarios de la empresa de seguros y la restante es externa. En este caso la persona externa es parte de un proveedor que trabaja in situ. A las 5 personas se les instaló la versión de prueba de la aplicación de terceros y se le dieron instrucciones a seguir a cada una de ellas y se muestra a continuación:

☑ Instrucción ejecutada Resultado Exitoso Resultado Fallido

Instrucciones	Usuario empresa 1	Usuario empresa 2	Usuario empresa 3	Usuario empresa 4	Usuario externo
Presionar botón Conectar con …	☑	☑	☑	☑	☑
Ingresar RUT correcto y contraseña correcta	☑	☑	☑		
Ingresar RUT correcto y contraseña incorrecta				☑	
Ingresar RUT no valido y contraseña inventada					☑
Presionar botón Continuar	☑	☑	☑	☑	☑
Resultado (Ingresar al Sistema)	✓	✓	✓	✗	✗

Tabla 3 - Pruebas de Autenticación

La Tabla 3 muestra las instrucciones que fueron ejecutadas por los participantes en las pruebas. Estas pruebas fueron diseñadas para obtener 3 casos exitosos y 2 casos no exitosos. De los casos no exitosos uno fue por no tener la contraseña correcta y el otro para validar que el RUT no pertenece a un usuario de la empresa. Los resultados de los casos de prueba fueron los esperados.

5.1.2 Pruebas de Autorización

La autorización comprende principalmente dos escenarios, ambos dependen de la configuración de una variable en la Aplicación Mantenedor de Políticas de Autorización. La variable indica si se le pide autorización al propietario del recurso o por defecto se tiene autorizado el acceso a los recursos compartidos.

La aplicación de terceros escogida para las pruebas ha sido configurada para que solicite la autorización al propietario del recurso; el flujo inicia luego del ingreso al sistema, la aplicación muestra una pantalla con los permisos solicitados y que el usuario puede aceptar o declinar como se muestran en la Figura 41.

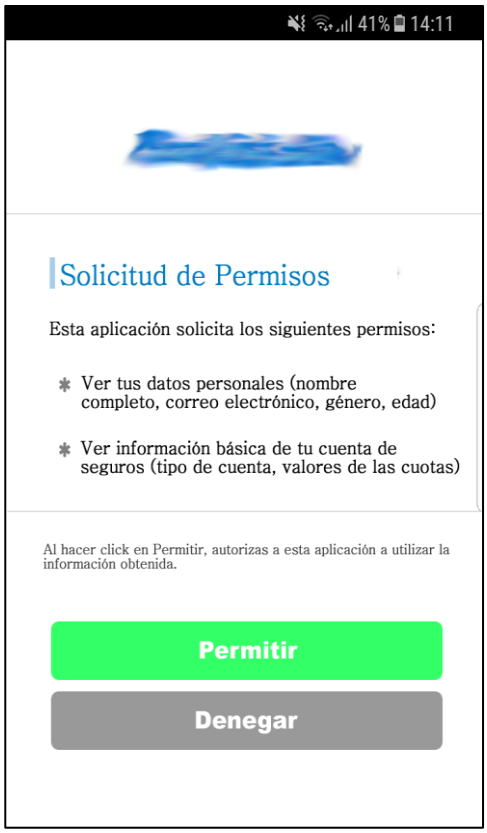

Figura 41 - Pantalla Solicitud de Permisos

Si se aceptan los permisos, se delega a la aplicación la autorización para que pueda obtener el token de acceso y así poder acceder a los recursos expuestos por la API. En el caso de denegar los permisos, la aplicación muestra la pantalla inicial de la Figura 38. El otro escenario se da cuando se configura la aplicación para que no muestre la pantalla de solicitud de permisos, y en tal caso se da una autorización por defecto permitiendo la negociación del token de acceso de una manera directa sin mostrar la pantalla de la Figura 41.

En las pruebas participaron los usuarios de la empresa de seguros que realizaron las pruebas de autenticación. Se les solicito a todos los usuarios que ingresen sus usuarios y contraseñas correctas y que validen que les aparece la pantalla como parte de la instrucción principal. La ejecución y resultados de las instrucciones se visualizan en la Tabla 4.

☑ Instrucción ejecutada 🟢 Resultado Exitoso

Instrucciones	Usuario empresa 1	Usuario empresa 2	Usuario empresa 3	Usuario empresa 4
Ingresar RUT correcto y contraseña correcta	☑	☑	☑	☑
Visualizar pantalla de permisos	☑	☑	☑	☑
Permitir autorización a la aplicación	☑	☑		
Denegar autorización a la aplicación			☑	☑
Resultado (Presentación de Pantalla de Aplicación)	🟢	🟢		
Resultado (Presentación de Pantalla de Inicio)			🟢	🟢

Tabla 4 - Pruebas de Autorización

Los resultados de las pruebas diseñadas fueron los esperados. Los usuarios que autorizaron pudieron realizar la consulta a los servicios expuestos por la Aplicación Servidor de Recursos y se muestra los resultados en las pruebas de funcionalidad.

5.1.3 Pruebas de Funcionalidad

Las pruebas de funcionalidad se han basado en la verificación de acceso a los servicios expuestos por la API para la aplicación de terceros, la obtención del token de acceso mediante la definición de la concesión de código de autorización se vuelve el factor importante a revisar. El flujo de la concesión código de autorización tiene dos iteraciones entre la aplicación de terceros y la Aplicación Servidor de Autorización, la primera es para obtener el código de autorización y la segunda para obtener finalmente el token de acceso mediante el código de autorización obtenido previamente.

El flujo inicia con la invocación del servicio *Authorization Endpoint,* donde se verifica que la aplicación cliente este registrada y la URL de respuesta sea la correcta, una vez verificado estos datos se devuelve el código de autorización. Con el código de autorización se hace la invocación al servicio *Token Endpoint* que verifica que la aplicación cliente este registrada y además que el código de autorización sea el último que le fue entregado, si todo es correcto se devuelve el token de acceso en la respuesta de la petición.

La Figura 42 muestra las trazas del log de la aplicación, el log se encuentra activado en modo DEBUG para poder constatar con mayor detalle la secuencia de obtener el token de acceso. Se omitieron logs y fue representado por puntos suspensivos en la Figura 42 por ser información no necesaria para el ejemplo mencionado. La información sensible se escribe ofuscada en los logs.

```
...
...
[20XX/08/08 13:03:06][INFO] - [AuthEndpoint] Method authorize
[20XX/08/08 13:03:06][DEBUG] - [ClientDAOImpl] Method existClientId
[20XX/08/08 13:03:06][DEBUG] - [Validator] Method checkClientId
...
[20XX/08/08 13:03:06][DEBUG] - [AuthEndpoint] Client OK
[20XX/08/08 13:03:06][DEBUG] - [Validator] Method checkCallback
[20XX/08/08 13:03:06][DEBUG] - [AuthEndpoint] Callback OK
[20XX/08/08 13:03:06][INFO] - [ResponseBuilderCode] Method buildResponse
[20XX/08/08 13:03:06][DEBUG] - [ResponseBuilderCode] Create OAuthAuthorizationResponseBuilder
[20XX/08/08 13:03:06][DEBUG] - [ResponseBuilderCode] Create authorizationCode xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
[20XX/08/08 13:03:06][DEBUG] - [CodeDAOImpl] Method saveCode
...
[20XX/08/08 13:03:06][INFO] - [ResponseBuilderCode] return Response
...
[20XX/08/08 13:03:07][INFO] - [TokenEndpoint] Method token
[20XX/08/08 13:03:07][DEBUG] - [ClientDAOImpl] Method existClientId
[20XX/08/08 13:03:07][DEBUG] - [Validator] Method checkClientId
...
[20XX/08/08 13:03:07][DEBUG] - [AuthEndpoint] Client OK
[20XX/08/08 13:03:07][DEBUG] - [TokenEndpoint] Method GrantType
[20XX/08/08 13:03:07][DEBUG] - [TokenEndpoint] GrantType AUTHORIZATION_CODE
[20XX/08/08 13:03:07][INFO] - [Validator] Method checkAuthCode
[20XX/08/08 13:03:07][DEBUG] - [AuthEndpoint] AuthCode OK
[20XX/08/08 13:03:07][DEBUG] - [ResponseBuilderCode] Create OAuthAuthorizationResponseBuilder
[20XX/08/08 13:03:07][DEBUG] - [ResponseBuilderCode] Create token xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
[20XX/08/08 13:03:07][DEBUG] - [ResponseBuilderCode] Create refreshToken xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
[20XX/08/08 13:03:07][DEBUG] - [CodeDAOImpl] Method saveToken
...
[20XX/08/08 13:03:07][INFO] - [ResponseBuilderCode] return Response
...
```

Figura 42 - Logs del Flujo Concesión Código de Autorización

Con el token de acceso obtenido, la aplicación de terceros de prueba puede realizar la consulta a los servicios expuestos por la aplicación Servidor de Recursos. Uno de los servicios a lo que tiene acceso la aplicación de prueba es la consulta de la información básica del usuario. En la Figura 43 se muestra la información obtenida por la aplicación de prueba utilizando el token de acceso. Los datos han sido ofuscados por acuerdos de confidencialidad con la empresa de seguros.

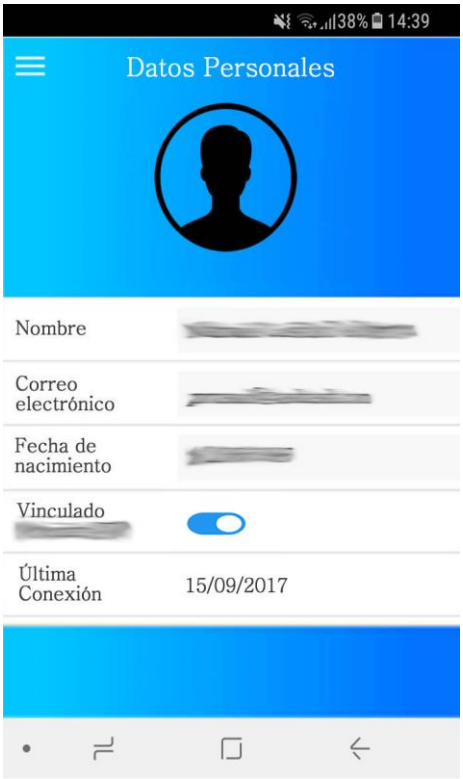

Figura 43 - Pantalla Consulta Información Básica

5.1.4 Cumplimiento de Calidad

La solución ha sido diseñada e implementada de tal forma que se pueda cumplir con los requisitos de calidad solicitados. Para evaluar el requerimiento no funcional de bajo acoplamiento se definió que la Aplicación Servidor de Autorización y Aplicación Servidor de Recursos deben ser independientes. Esta definición se cumple, aunque estas aplicaciones se han instalado en el mismo servidor físico, se han construido de forma tal que puedan ser instaladas en diferentes servidores de ser necesario. El requerimiento no funcional de obtener bajo acoplamiento entre las aplicaciones pudo ser cumplido en parte gracias al diseño, sin embargo, ambas aplicaciones poseen una dependencia fuerte con la base de datos. La evolución de la base de datos afecta a las dos aplicaciones haciendo que exista un acoplamiento que no se tomó en cuenta.

Para validar la robustez de la solución se realizaron pruebas inyectando peticiones con formato incorrecto, con código de autorización no válido, con token de acceso expirado, se intentó obtener el token de acceso con otra concesión que no sea la configurada, entre otras pruebas mediante la herramienta Advanced REST Client de Google Chrome, esta herramienta permite configurar los datos de entrada de una petición hacia un servicio Web. El manejo de errores o condiciones de excepción durante todo el proceso de

obtención del token y acceso a los recursos permitió operar correctamente ante el 100% de las pruebas realizadas, logrando que no se comprometa el sistema.

La Aplicación Servidor de Recursos ha sido diseñada para que pueda crecer en cantidad de servicios que se exponen o comparten a las aplicaciones clientes, esto se realiza mediante inclusión de código Java siguiendo el patrón de desarrollo. Para esto se debe implementar una clase Java que represente el servicio a exponer de igual manera que los otros servicios ya implementados y agregar el nuevo servicio a la configuración del proyecto. Aunque se entregó un manual para añadir nuevos servicios no se pudo evaluar la facilidad del procedimiento.

Para las pruebas de eficiencia en cuanto a tiempos de respuesta se utilizó el motor de ejecución de pruebas JMeter. El mismo corresponde a un proyecto de Apache Jakarta que puede ser utilizado como una herramienta para pruebas de carga y medir el desempeño de una variedad de servicios, con énfasis en servicios Web. Se realizó un análisis de tiempos de respuesta de los servicios expuestos por la API, se comparó las peticiones que utilizan las aplicaciones internas y que usan credenciales de aplicación contra las peticiones realizadas al Servidor de Recursos una vez que se ha obtenido el token de acceso.

Se creó en JMeter una prueba con las credenciales del sitio web de la empresa de seguros tomando como referencia que es una aplicación interna y se direccionó la prueba directo los servicios de la API, mientras que la otra prueba fue direccionando al Servidor de Recursos con un token válido que fue tomado directo de la base de datos. La Tabla 5 tiene los datos tomados de las pruebas realizadas para el servicio información básica, los resultados están expresados en segundos.

Servicio Información Básica	1 usuario	5 usuarios	10 usuarios	50 usuarios	100 usuarios
Prueba con credenciales Aplicación Interna	0,68s	0,68s	0,70s	0,73s	0,76s
Prueba con token de acceso de Aplicación de Terceros	0,65s	0,65s	0,65s	0,67s	0,69s

Tabla 5 - Pruebas de Carga Servicio Información Básica

Para los dos servicios adicionales los resultados reflejan lo mismo, se encuentra una mejora mínima en los tiempos de respuesta cuando se utiliza el protocolo OAuth. Se muestran los datos de este servicio únicamente ya que los otros dos servicios son similares en características y resultados. La definición para que se dé cumplimiento al requerimiento no funcional de eficiencia fue que los tiempos de respuesta a los servicios expuestos por la API

con el protocolo OAuth no tengan un margen superior al 3% de los tiempos de respuesta sin el protocolo OAuth. Este valor fue definido por la empresa de seguros.

5.2 Pruebas de Aceptación

La evaluación final recogida en el proyecto fue dada por una entrevista al jefe de proyecto de la empresa de seguros. En dicha entrevista se puede constatar los resultados obtenidos con la solución. El objetivo de esta evaluación no es validar el comportamiento lógico específico de un componente sino un escenario o caso de negocio. Su característica principal es que se realiza para validar que la solución permite completar las tareas y objetivos para el que fue diseñado.

Aunque el jefe de proyecto no haya ejecutado directamente algunos puntos de la entrevista, tiene pleno conocimiento de su ejecución y resultados. A continuación, los datos de la entrevista realizada al jefe de proyecto.

1. **Prueba:** Registrar una aplicación de terceros en el mantenedor de políticas de autorización.
   Descripción: Poder registrar y configurar una aplicación de terceros para que puede acceder a los recursos compartidos.

   Resultado Obtenido: *"Se configura en el mantenedor de políticas de autorización una nueva aplicación de terceros, utilizando los servicios disponibles en la API que pueden ser accedidos, los resultados obtenidos son satisfactorios, no hubo mayor demora en la respuesta del almacenamiento."*

2. **Prueba:** Iniciar sesión con cuenta de la empresa de seguros desde una aplicación de terceros.
   Descripción: Poder utilizar la cuenta principal como cliente de la empresa de seguros en una aplicación de terceros.

   Resultado Obtenido: *"Al iniciar desde la aplicación de terceros se consumen los servicios sin mayor demora con un usuario previamente registrado y con sus roles definidos, el servicio para la identificación se encuentra disponible y permite el ingreso desde la aplicación de terceros."*

3. **Prueba:** Autorizar a la aplicación de terceros para acceder a servicios.

Descripción: Poder brindar la opción de que el usuario final delegue a la aplicación de terceros el acceso a los servicios.

Resultado Obtenido: *"Se muestra de forma clara los permisos a los que se está permitiendo el acceso, lo único es que las restricciones son totales, pero se permite la delegación a la aplicación de terceros."*

4. **Prueba:** Acceder a servicios de la API desde una aplicación de tercero.
   Descripción: Poder acceder a los servicios de la API desde una aplicación de terceros.

   Resultado Obtenido: *"Se consume el servicio desde la aplicación de terceros para acceder a los servicios de información del usuario."*

5. **Prueba:** Integración de librería con una aplicación de terceros.
   Descripción: Poder usar la librería construida en una aplicación de tercero.

   Resultado Obtenido: *"Al llevarse las librerías a la aplicación de terceros, los desarrolladores utilizan los componentes para consumir las interfaces disponibles, el proceso documentado significó de gran ayuda para su uso."*

6. **Prueba:** Revocar el acceso a los servicios por una aplicación de terceros.
   Descripción: Poder eliminar el acceso a los servicios que tiene una aplicación de terceros.

   Resultado Obtenido: *"Se consumen los servicios de revocación para quitarle los privilegios a una aplicación previamente definida, se comprueba que se solicitan los tokens necesarios y que la documentación en caso de errores es clara."*

La prueba de aceptación final se realizó para dar una evaluación final de la solución y a las aplicaciones que la conforman, antes de comenzar con la puesta en producción. Los resultados descritos son la transcripción de la entrevista al jefe de proyecto. Con estos resultados se cumple los objetivos planteados para la solución y se da por aceptado el proyecto en la empresa de seguros.

Capítulo VI: Conclusiones y Trabajo Futuro

En este capítulo se presenta las conclusiones y las lecciones aprendidas que se han obtenido a lo largo del desarrollo de esta tesis, así como también el trabajo a futuro que se puede seguir realizando.

6.1 Conclusiones

Este trabajo de tesis permitió el acceso a la API de la empresa de seguros desde aplicaciones de terceros de una forma segura y eficaz mediante el uso del protocolo OAuth 2.0. Se obtuvieron ventajas como evitar la propagación de contraseñas por cada aplicación, poder limitar el acceso de forma configurable y sobre todo brindarle al usuario final la capacidad de revocar este acceso en cualquier momento.

Además, al garantizar el acceso seguro a las funcionalidades de la API para la empresa de seguros se obtuvieron beneficios adicionales, por ejemplo, crear un estándar para las llamadas a los servicios y que se puede utilizar en aplicaciones externas e internas; se genera un entorno listo para la innovación al permitir que puedan surgir aplicaciones que utilicen la información para ofrecer nuevos servicios a los usuarios.

Con la solución también se pudo mejorar el proceso de autenticación en aplicaciones de terceros; para la autorización a la aplicación se verifica si el usuario esta autenticado, en caso de no estarlo se redirige al canal principal de autenticación y se utiliza la misma identificación personal que posee para la empresa de seguros, evitando así que una misma persona tenga cuentas diferentes por cada aplicación de terceros que exista.

La etapa de análisis del protocolo OAuth 2.0 fue la más importante en el ciclo de desarrollo. El análisis detallado de las aplicaciones que participan en el protocolo ayudó al modelamiento de componentes, logrando que algunos de ellos puedan ser reutilizables. La reusabilidad de estos componentes aportó en los tiempos de desarrollo, así como también en facilitar el mantenimiento en las aplicaciones una vez finalizadas.

En las pruebas de peticiones realizadas a los recursos compartidos se denotó que no hubo mayor problema con los tiempos de respuesta. Pero en caso de que se aumente la transaccionalidad se tiene como ventaja la discriminación lógica de las funciones de la Aplicación Servidor de Autorización y la Aplicación Servidor de Recursos. Esto promueve que pueda existir una Aplicación Servidor de Autorización que sea centralizada y que pueda haber balanceo entre varias Aplicaciones de Servidores de Recursos.

6.2 Lecciones Aprendidas

La implementación del protocolo OAuth 2.0 ha servido para que me resulte, a partir de ahora, muy sencillo interactuar con una API que esté protegida con este sistema. Es importante reconocer que OAuth 2.0 solo es un componente de una completa solución de control de acceso y seguridad de APIs. Es fácil centrarse en los detalles del protocolo y perder de vista la visión global de la seguridad de una API, que abarca desde gestión del usuario hasta la auditoría, la regulación y la detección de amenazas.

Tener los cuatro tipos de concesiones implementados no garantizó que todos se vayan a usar. En este caso se detecta que el principal y único uso que se requería era para aplicaciones móviles de terceros. Para esto basta solo la concesión Código de Autorización y que en su defecto también puede ser utilizado por aplicaciones Web.

La librería desarrollada para los dispositivos móviles utiliza un agente embebido, lo cual plantea un desafío de seguridad debido a que los propietarios de los recursos se autentican en una ventana no identificada y sin acceso a las protecciones visuales encontradas en la mayoría de los agentes externos como un explorador Web. Un agente embebido educa a los usuarios finales a confiar en solicitudes de autenticación no identificados y esto facilita que ataques como phishing puedan ser fáciles de ejecutar.

6.3 Trabajo Futuro

Se tiene planificado exponer más servicios de la API para poder ser accedidos mediante el protocolo OAuth, para aquello hay que modificar el código fuente de la aplicación servidor de recursos. Una posible mejora que se plantea es poder agregar servicios a manera de configuración, evitando modificar el código se reduciría el riesgo de generar errores por los cambios realizados.

Otra mejora que se puede realizar es que la librería construida pueda verificar si en el dispositivo móvil se encuentra autenticado el usuario en la aplicación propia de la empresa, en caso de estarlo poder realizar el flujo de solicitud de token de acceso directamente sin requerir los datos de usuario y contraseña. Para realizar esta tarea se requiere de un análisis exhaustivo para no dejar abierta una brecha de seguridad.

BIBLIOGRAFÍA

[1] Página Web de OpenID [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: http://openid.net/

[2] Página Web de OAuth 1.0 [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://oauth.net/

[3] Página Web de SAML [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: http://saml.xml.org/

[4] Página Web de OpenId Connect [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: http://openid.net/connect/

[5] Daniel Jacobson, Greg Brail & Dan Woods - APIs: A Strategy Guide. O´REILLY, 2012

[6] Página Web del World Wide Web Consortium [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: http://www.w3.org/

[7] Página Web de Oasis Consortium [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: http://www.oasis-open.org/

[8] Alberto Los Santos Aransay: Revisión de los Servicios Web SOAP/REST: Características y Rendimiento. Universidad de Vigo, 2009

[9] C. Gutiérrez, E. Fernández-Medina, M. Piattini: Seguridad en Servicios Web. Universidad de Castilla. La Mancha, España, 2005

[10] Spencer C. Lee: An Introduction to Identity Management. SANS Institute, 2003

[11] Internet Engineering Task Force (IETF) [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://www.ietf.org/

[12] OAuth 2.0 [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://oauth.net/2/

[13] Revisión OAuth 2.0 [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://tools.ietf.org/html/rfc6749

[14] Revisión LDAP [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://tools.ietf.org/html/rfc4511

[15] Apache OLTU [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://oltu.apache.org/

[16] RESTEasy [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: http://resteasy.jboss.org/

[17] Spring Framework [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://projects.spring.io/spring-framework

[18] Angular JS [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://angularjs.org/

[19] Apache Maven [en línea]. Actualizada 2018 [Fecha de consulta: 06 Enero 2018] Disponible en: https://maven.apache.org/

ANEXOS

Anexo (A.1) Requisitos Específicos de la Solución

RF01: La autenticación del usuario en el cliente será mediante la cuenta principal de la empresa de seguros.

RF02: La solicitud del token de acceso para el cliente será mediante una URL en el servidor de autorización. El servidor debe ser capaz de interpretar la solicitud y procesar sus parámetros.

RF03: La pantalla de autorización que se muestra al usuario es configurable, si está activado se muestra, caso contrario se dará acceso por defecto a los recursos configurados.

RF04: El servidor de autorización identificará si el usuario está autenticado, en caso de no estarlo se procede con el **RF01**.

RF05: Con el usuario autenticado, el servidor de autorización dependiendo de la configuración puede mostrar o no la pantalla de autorización **RF03**. En dicha pantalla se acepta o se deniega los permisos para acceder a los recursos.

RF06: El servidor de autorización deberá crear y almacenar el token de acceso y de actualización a ser intercambiado con el cliente.

RF07: El servidor de autorización debe ser capaz de generar una respuesta de error por defecto, en caso de un fallo en el flujo.

RF08: Un recurso puede ser accedido solo si se cuenta con el token de acceso y el recurso está permitido dentro de las políticas de autorización para el cliente.

RF09: Las políticas de autorización serán configurables mediante una aplicación web.

RF10: Construcción de librería para la plataforma Android.

RF11: El servidor de autorización soportará la solicitud del flujo concesión código de autorización para obtener el código de autorización, dicha solicitud es de método GET. Los parámetros se muestran en la tabla 6:

Parámetro	Tipo	Descripción
response_type	Requerido	El valor debe ser "code".
client_id	Requerido	El valor identificador del cliente.
redirect_uri	Opcional	La dirección URL de redirección, esto después de que el usuario decida si autorizar o no al cliente. Si no se proporciona se obtendrá de los datos asociados al cliente en la configuración almacenada.
scope	Opcional	Permisos que quiere obtener el cliente. Si no se especifica se toman los permisos por defectos
state	Opcional	El valor utilizado por el cliente para mantener el estado desde que se hace la petición hasta que se obtiene una respuesta. El servidor de autorización devolverá el mismo parámetro en la respuesta.

Tabla 6 - Concesión Código de Autorización - Parámetros solicitud de Código de Autorización

Por ejemplo:

```
GET    /auth?response_type=code&client_id=s6BhdRkqt3
       &state=xyz&redirect_uri=https://cliente.com/callback
```

Figura 44 - Concesión Código de Autorización - Ejemplo Solicitud de Código de Autorización

RF12: Aceptada la autorización por parte del usuario o la autorización por defecto configurada, se debe enviar la respuesta al cliente en la URL especificada en la solicitud. En la tabla 7 se muestra los parámetros enviados en la respuesta a la solicitud de código de autorización:

Parámetro	Tipo	Descripción
code	Requerido	El código de autorización generado por el servidor.
state	Opcional	El mismo valor proporcionado por el cliente al realizar la petición.

Tabla 7 - Concesión Código de Autorización - Parámetros de respuesta a solicitud de Código de Autorización

Por ejemplo:

```
HTTP/1.1 302 Found
Location: /cb?code=SplxlOBeZQQYbYS6WxSbIA
```

Figura 45 - Concesión Código de Autorización – Ejemplo Solicitud de Código de Autorización

RF13: Manejo de error para la solicitud de código de autorización en la concesión código de autorización. La descripción de los códigos de error se encuentra en el Anexo (A.2). La respuesta para caso de falla tiene los parámetros de la tabla 8:

Parámetro	Tipo	Descripción
error	Requerido	El valor es un identificador según el error y puede ser: invalid_request, unauthorized_client, access_denied, unsupported_response_type, invalid_scope, server_error, temporarily_unavailable. La descripción de cada valor se encuentra en la tabla del Anexo A.
error_description	Opcional	Descripción del error ocurrido.
error_uri	Opcional	URI de una página Web con información sobre el error, utilizada para proporcionar al desarrollador del cliente información adicional sobre el error.

Tabla 8 - Concesión Código de Autorización - Parámetros de Respuesta en Error de Solicitud de Código de Autorización

Por ejemplo:

```
HTTP/1.1 302 Found
Location: /cb?error=access_denied&state=xyz
```

Figura 46 - Concesión Código de Autorización - Ejemplo Error en Solicitud de Código de Autorización

RF14: El servidor de autorización soportará la solicitud del flujo concesión código de autorización para obtener el token de acceso, dicha solicitud es de método POST. Los parámetros se muestran en la tabla 9.

Parámetro	Tipo	Descripción
grant_type	Requerido	El valor debe ser "authorization_code".
code	Requerido	El código de autorización recibido del servidor de autorización.
redirect_uri	Requerido	Si la URI fue incluida en la solicitud del código de autorización. Ha de ser la misma
client_id	Requerido	El valor identificador del cliente.
client_secret	Requerido	El valor del código secreto entregado a la aplicación en su registro dentro de la empresa de seguros.

Tabla 9 - Concesión Código de Autorización - Parámetros de solicitud de Token de Acceso

Por ejemplo:

POST **Content-Type: application/x-www-form-urlencoded**
grant_type=authorization_code&**code**=SplxlOBeZQQYbYS6WxSbIA
&**redirect_uri**=/cb&**client_id**=s6BhdRkqt3
&**client_secret**= M2WhfHdqa8

Figura 47 - Concesión Código de Autorización - Ejemplo Solicitud de Token de Acceso

RF15: Aceptada la solicitud de token de acceso, se debe enviar la respuesta al cliente en la URL especificada en la solicitud. En la tabla 10 se muestra los parámetros de respuesta requeridos:

Parámetro	Tipo	Descripción
access_token	Requerido	El token de acceso generado por el servidor.
token_type	Requerido	El tipo de token.
expires_in	Requerido	El tiempo de vida en segundos de duración del token de acceso.
refresh_token	Requerido	Token de actualización en caso de que el token de acceso expire.

Tabla 10 – Concesión Código de Autorización - Parámetros de respuesta a solicitud de Token de Acceso

Por ejemplo:

```
HTTP/1.1 200 OK
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache

    {
     "access_token":"2YotnFZFEjr1zCsicMWpAA",
     "token_type":"example",
     "expires_in":3600,
     "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
     "example_parameter":"example_value"
    }
```

Figura 48 - Concesión Código de Autorización - Ejemplo respuesta a solicitud de Token de Acceso

RF16: Si la solicitud de token de acceso falla el servidor de autorización responde con un error HTTP 400. La respuesta para caso de falla tiene los parámetros de la tabla 11:

Parámetro	Tipo	Descripción
error	Requerido	El valor es un identificador según el error y puede ser: invalid_request, invalid_client, invalid_grant, unauthorized_client, unsupported_grant_type, invalid_scope. La descripción de cada valor se encuentra en la tabla del Anexo A.
error_description	Opcional	Descripción del error ocurrido.
error_uri	Opcional	URI de una página Web con información sobre el error, utilizada para proporcionar al desarrollador del cliente información adicional sobre el error.

Tabla 11 - Parámetros de Respuesta en Error de Solicitud de Token de Acceso

Por ejemplo:

```
HTTP/1.1 400 OK
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache

    {
     "error":"invalid_request"
    }
```

Figura 49 - Concesión Código de Autorización - Respuesta de error Solicitud Token de Acceso

RF17: El servidor de autorización soportará la solicitud del flujo concesión implícito para obtener el token de acceso, dicha solicitud es de método GET. Los parámetros se muestran en la tabla 12:

Parámetro	Tipo	Descripción
response_type	Requerido	El valor debe ser "token".
client_id	Requerido	El valor identificador del cliente.
client_secret	Requerido	El valor del código secreto entregado a la aplicación en su registro dentro de la empresa de seguros.
redirect_uri	Opcional	La dirección URL de redirección, esto después de que el usuario decida si autorizar o no al cliente. Si no se proporciona se obtendrá de los datos asociados al cliente en la configuración almacenada.
scope	Opcional	Permisos que quiere obtener el cliente. Si no se especifica se toman los permisos por defectos
state	Opcional	El valor utilizado por el cliente para mantener el estado desde que se hace la petición hasta que se obtiene una respuesta. El servidor de autorización devolverá el mismo parámetro en la respuesta.

Tabla 12 - Concesión Implícita - Parámetros solicitud de Token de Acceso

Por ejemplo:

```
GET    /auth?response_type=token&client_id=s6BhdRkqt3
       &client_secret=M2WhfHdqa8&state=xyz
       &redirect_uri=https://cliente.com/callback
```

Figura 50 - Concesión Implícita - Ejemplo Solicitud Token de Acceso

RF18: Aceptada la autorización por parte del usuario o la autorización por defecto configurada, se debe enviar la respuesta al cliente en la URL especificada en la solicitud. En la tabla 13 se muestra los parámetros enviados:

Parámetro	Tipo	Descripción
access_token	Requerido	El token de acceso generado por el servidor.
token_type	Requerido	El tipo del token de acceso.
expires_in	Requerido	El tiempo de vida en segundos de duración del token de acceso.
scope	Opcional	Permisos que quiere obtener el cliente. Si no se especifica se toman los permisos por defectos
state	Requerido	El valor utilizado por el cliente para mantener el estado desde que se hace la petición hasta que se obtiene una respuesta. El servidor de autorización devolverá el mismo parámetro en la respuesta.

Tabla 13 - Concesión Implícita - Parámetros de respuesta a solicitud de Token de Acceso

Por ejemplo

```
HTTP/1.1 302 Found
Location: /cb?access_token=2YotnFZFEjr1zCsicMWpAA
          &state=xyz&token_type=example&expires_in=3600
```

Figura 51 - Concesión Implícita - Ejemplo de respuesta a solicitud de Token de Acceso

RF19: Manejo de error para la solicitud de token de acceso para la concesión implícita. La respuesta para caso de falla tiene los parámetros de la tabla 14:

Parámetro	Tipo	Descripción
error	Requerido	El valor es un identificador según el error y puede ser: invalid_request, unauthorized_client, access_denied, unsupported_response_type, invalid_scope, server_error, temporarily_unavailable. La descripción de cada valor se encuentra en la tabla del Anexo A.
error_description	Opcional	Descripción del error ocurrido.
error_uri	Opcional	URI de una página Web con información sobre el error, utilizada para proporcionar al desarrollador del cliente información adicional sobre el error.
state	Requerido	El mismo valor proporcionado por el cliente al realizar la petición en caso de que este parámetro estuviera presente.

Tabla 14 - Concesión Implícita - Parámetros de Respuesta en Error de Solicitud de Token de Acceso

Por ejemplo:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied&state=xyz
```

Figura 52 - Concesión Implícita – Ejemplo de Respuesta en Error de Solicitud de Token de Acceso

RF20: El servidor de autorización soportará la solicitud de la concesión credenciales del propietario del recurso para obtener el token de acceso, dicha solicitud es de método POST y la URI es /token. Los parámetros se muestran en la tabla 15:

Parámetro	Tipo	Descripción
grant_type	Requerido	El valor debe ser “password”.
username	Requerido	El usuario del propietario del recurso.
password	Requerido	La contraseña del propietario del recurso.
scope	Opcional	Permisos que quiere obtener el cliente. Si no se especifica se toman los permisos por defectos

Tabla 15 - Concesión Credenciales del Propietario del Recurso - Parámetros de solicitud de Token de Acceso

Por ejemplo:

```
POST    Content-Type: application/x-www-form-urlencoded
        grant_type=password&&username=johndoe&password=A3ddj3w
```

Figura 53 - Concesión Credenciales del Propietario del Recurso - Ejemplo Solicitud Token de Acceso

RF21: La autorización es aceptada por defecto, se debe genera una respuesta. En la tabla 16 se muestra los parámetros requeridos de la respuesta:

Parámetro	Tipo	Descripción
access_token	Requerido	El token de acceso generado por el servidor.
token_type	Requerido	El tipo de token.
expires_in	Requerido	El tiempo de vida en segundos de duración del token de acceso.
refresh_token	Requerido	Token de actualización en caso de que el token de acceso expire.

Tabla 16 - Concesión Credenciales del Propietario del Recurso - Parámetros de respuesta a solicitud de Token de Acceso

Por ejemplo:

```
HTTP/1.1 200 OK
     Content-Type: application/json;charset=UTF-8
     Cache-Control: no-store
     Pragma: no-cache

     {
       "access_token":"2YotnFZFEjr1zCsicMWpAA",
       "token_type":"example",
       "expires_in":3600,
       "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
       "example_parameter":"example_value"
     }
```

Figura 54 - Concesión Credenciales del Propietario del Recurso - Ejemplo Respuesta Solicitud de Token de Acceso

RF22: Manejo de error será igual al **RF16**.

RF23: El servidor de autorización soportará la solicitud de la concesión credenciales del cliente para obtener el token de acceso, dicha solicitud es de método POST y la URI es /token. Los parámetros se muestran en la tabla 17:

Parámetro	Tipo	Descripción
grant_type	Requerido	El valor debe ser “client_credentials”.
scope	Opcional	Permisos que quiere obtener el cliente. Si no se especifica se toman los permisos por defectos

Tabla 17 - Concesión Credenciales del Cliente - Parámetros de solicitud de Token de Acceso

Por ejemplo:

```
POST    Content-Type: application/x-www-form-urlencoded
        grant_type=client_credentials
```

Figura 55 - Concesión Credenciales del Propietario del Recurso - Ejemplo Solicitud de Token de Acceso

RF24: La autorización es aceptada por defecto, se debe genera una respuesta. En la tabla 18 se muestra los parámetros requeridos de la respuesta:

Parámetro	Tipo	Descripción
access_token	Requerido	El token de acceso generado por el servidor.
token_type	Requerido	El tipo de token.
expires_in	Requerido	El tiempo de vida en segundos de duración del token de acceso.

Tabla 18 - Concesión Credenciales del Cliente - Parámetros de respuesta a solicitud de Token de Acceso

Por ejemplo:

```
HTTP/1.1 200 OK
     Content-Type: application/json;charset=UTF-8
     Cache-Control: no-store
     Pragma: no-cache

     {
       "access_token":"2YotnFZFEjr1zCsicMWpAA",
       "token_type":"example",
       "expires_in":3600,
       "example_parameter":"example_value"
     }
```

Figura 56 - Concesión Credenciales del Cliente - Ejemplo Respuesta Solicitud de Token de Acceso

RF25: Manejo de error será igual al **RF16**.

RF26: El servidor de autorización soportará la solicitud para actualizar el token de acceso una vez que este caduque, dicha solicitud es de método POST y la URI es /token. Los parámetros se muestran en la tabla 19:

Parámetro	Tipo	Descripción
grant_type	Requerido	El valor debe ser "refresh_token".
refresh_token	Requerido	El token de actualización entregado por el servidor de autorización.
scope	Opcional	Permisos que quiere obtener el cliente. Si no se especifica se toman los permisos por defectos

Tabla 19 - Actualización de Token de Acceso - Parámetros de solicitud de Token de Acceso

Por ejemplo:

```
POST    Content-Type: application/x-www-form-urlencoded
        grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

Figura 57 - Actualización de Token de Acceso - Ejemplo Solicitud de Token de Acceso

RF27: Manejo de respuesta de token de acceso será igual al **RF21**.

RF28: Manejo de error será igual al **RF16**.

RF29: El servidor de recursos soportará la solicitud para acceso a los servicios expuestos, los parámetros de entrada serán los mismos parámetros de los servicios de la API, pero vendrán acompañados del token de acceso en el cuerpo del mensaje.

Por ejemplo:

```
POST    Content-Type: application/json
        {
        "param1":"example",
        "param2":"example",
        "accessToken":"2YotnFZFEjr1zCsicMWpAA"
        }
```

Figura 58 - Acceso Recurso Compartido - Ejemplo

RF30: La respuesta desde el servidor de recursos al cliente, será la misma respuesta que tiene el servicio en la API.

RF31: La respuesta de por error de token de acceso será igual a **RF16**.

RF32: El servidor de autorización soportará la solicitud para anular el token de acceso, dicha solicitud es de método POST. Los parámetros se muestran en la tabla 20:

Parámetro	Tipo	Descripción
refresh_token	Requerido	El token de actualización entregado por el servidor de autorización.

Tabla 20 - Anulación de Token de Acceso - Parámetros de solicitud

Por ejemplo:

```
POST    Content-Type: application/x-www-form-urlencoded
        refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

Figura 59 - Anulación de Token de Acceso - Ejemplo Anulación de Token de Acceso

RF33: La respuesta desde el servidor de autorización al cliente será con los siguientes parámetros de la tabla 21:

Parámetro	Tipo	Descripción
revoke_status	Requerido	Estado de la anulacion. [Done/Error]
description	Requerido	Descripción del resultado.

Tabla 21 - Anulación de Token de Acceso - Parámetros de respuesta

Por ejemplo:

```
HTTP/1.1 200 OK
     Content-Type: application/json;charset=UTF-8
     Cache-Control: no-store
     Pragma: no-cache

     {
       "result_status":"Done",
       "description":""
     }
```

Figura 60 - Concesión Credenciales del Cliente - Ejemplo Respuesta Solicitud de Token de Acceso

RF34: La aplicación proveerá el registro de usuarios internos de la empresa para la configuración de las políticas de autorización.

RF35: La aplicación permitirá la eliminación de usuarios registrados.

RF36: La autenticación de los usuarios será mediante el sistema principal de autenticación de la empresa.

RF37: Los usuarios registrados podrán ingresar las aplicaciones clientes que participan dentro de los flujos del protocolo OAuth 2.0.

RF38: El registro de la aplicación cliente debe tener los siguientes parámetros de la tabla 22:

Parámetro	Tipo	Descripción
name	Requerido	Nombre de la aplicación cliente.
description	Requerido	Descripción de la aplicación cliente.
home_uri	Requerido	URL de la página principal de la aplicación cliente.
callback_uri	Requerido	URL de redirección usada para el protocolo OAuth 2.0.
grant_type	Requerido	Tipo de concesión que posee la aplicaicon cliente para el protocolo OAuth 2.0
token_expires	Requerido	Tiempo de expiración del token de acceso.
authorizated_uri	Requerido	URLs de los recursos compartidos que puede acceder la aplicación cliente en caso de ser autorizada.
authorization_default	Requerido	Variable que indica si la autorización es por defecto aceptada o se solicita autorización al usuario propietario del recurso.

Tabla 22 - Parámetros de Registro de Aplicación Cliente

RF39: La aplicación podrá generar el identificador y el código secreto de un cliente.

RF40: Los usuarios registrados podrán eliminar una aplicación cliente.

RF41: La librería para aplicaciones móviles Android permitirá obtener el token de acceso mediante la concesión de código de autorización.

RF42: La librería para aplicaciones móviles iOS permitirá obtener el token de acceso mediante la concesión de código de autorización.

RNF01: La implementación permitirá que el servidor de autorización y el servidor de recursos sean independientes entre sí.

RNF02: La implementación soportará el manejo de gran cantidad de información.

RNF03: La implementación garantizará la disponibilidad de la información.

RNF04: La implementación garantizará la integridad de la información.

RNF05: La implementación permitirá agregar recursos compartidos sin dificultad.

RNF06: La implementación manejará los fallos de forma adecuada.

RNF07: La implementación permitirá la trazabilidad de los sucesos ocurridos.

RNF08: La implementación garantizará que el acceso a la información sea solo a interesados que estén autorizados.

RNF09: La implementación garantizará la confidencialidad de la información.

Anexo (A.2) Códigos de Error en Respuesta de Solicitudes Protocolo OAuth 2.0

Valor Error	Descripción
invalid_request	Falta un parámetro requerido en la solicitud, incluye un valor de parámetro no válido, incluye un parámetro más de una vez, o la solicitud esta malformada de otra manera.
unauthorized_client	El cliente no está autorizado a utilizar este método.
invalid_client	Falló la autenticación de cliente, por ejemplo, el cliente es desconocido, o la autenticación de cliente no está incluida o el método de autenticación no está soportado.
invalid_grant	Fallo por los siguientes motivos: • La concesión de autorización proporcionada, por ejemplo, código de autorización, credenciales de propietario de recurso es invalida • El token de actualización es inválido, caducado, revocado, o no coincide con la URI de redirección utilizada en la solicitud de autorización • El token de actualización es emitido para otro cliente.
access_denied	El propietario del recurso o el servidor de autorización negaron la solicitud.
unsupported_response_type	El servidor de autorización no admite la obtención del código de autorización utilizando este método.
unsupported_grant_type	El tipo de concesión de autorización no está soportado por el servidor de autorización.
invalid_scope	El ámbito solicitado es inválido, desconocido o malformado.
server_error	El servidor de autorización encontró una condición que le impidió cumplir con la solicitud. Representa al código de error HTTP 500.
temporarily_unavailable	El servidor de autorización no puede la solicitud debido a una sobrecarga o mantenimiento temporal del servidor. Representa al código de error HTTP 503.

Tabla 23 - Tabla de Códigos de Error

Anexo (B) Especificación de servicios expuestos por aplicación

Código de Autorización	
URL de acceso	/auth
Descripción	Genera el código de autorización si es la concesión código de autorización o genera un token de acceso para la concesión implícita.
Método HTTP	GET
Mensaje de Entrada	RF11-RF17
Mensaje de Salida	RF12-RF18
Mensaje de Error	RF13-RF19

Tabla 24 - Servidor de Autorización - Servicio **/auth**

Token de Acceso	
URL de acceso	/token
Descripción	Genera el token de acceso.
Método HTTP	POST
Mensaje de Entrada	RF14-RF20-RF23-RF26
Mensaje de Salida	RF15-RF21-RF24
Mensaje de Error	RF16

Tabla 25 - Servidor de Autorización - Servicio **/token**

URL de acceso	**/revoke**
Descripción	Revoca los tokens para un usuario
Método HTTP	POST
Mensaje de Entrada	RF14-RF20-RF23-RF26
Mensaje de Salida	RF15-RF21-RF24
Mensaje de Error	RF16

Tabla 26 - Servidor de Autorización - Servicio **/revoke**

Obtener Información Personal			
URL de acceso	/getInfo		
Descripción	Consulta de información básica de un usuario.		
Método HTTP	POST		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	rut	String	No
	accessToken	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	fullName	String	No
	email	String	No
	gender	String	No
	birth	String	Si
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 27 - Servidor de Recursos - Servicio **/getInfo**

Obtener Datos de la Cuenta			
URL de acceso	/getAccount		
Descripción	Consulta información del estado de cuenta de un usuario.		
Método HTTP	POST		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	rut	String	No
	accessToken	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	currentQuota	String	No
	typeAccount	String	No
	quotas	String	No
	totalQuotas	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 28 - Servidor de Recursos - Servicio **/getAccount**

Modificar Tipo de Cuenta			
URL de acceso	/setTypeAcc		
Descripción	Modifica el tipo de fondo de una cuenta de un usuario.		
Método HTTP	POST		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	rut	String	No
	newTypeAcc	String	No
	accessToken	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
	currentTypeAcc	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 29 - Servidor de Recursos - Servicio **/setTypeAcc**

Inicio de Sesión			
URL de acceso	/login		
Descripción	Inicio de sesión en la aplicación por parte de usuarios internos de la empresa de seguros.		
Método HTTP	POST		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	rut	String	No
	password	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
	user	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 30 - Mantenedor de Políticas de seguridad - Acción **/login**

Consultar Aplicaciones Clientes			
URL de acceso	/client/list		
Descripción	Consulta de aplicaciones clientes.		
Método HTTP	GET		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
	clients	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 31 - Mantenedor de Políticas de seguridad - Acción **/client/list**

Guardar Aplicación Cliente			
URL de acceso	/client/save		
Descripción	Guarda una aplicación cliente nueva, la actualiza en caso de existir.		
Método HTTP	POST		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	client	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
	clients	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 32 - Mantenedor de Políticas de seguridad - Acción **/client/sabe**

Eliminar Aplicación Cliente			
URL de acceso	/client/delete		
Descripción	Elimina una aplicación cliente.		
Método HTTP	POST		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	client	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 33 - Mantenedor de Políticas de seguridad - Acción **/client/delete**

Consultar Usuarios			
URL de acceso	/user/list		
Descripción	Consultar usuarios internos de la empresa de seguros registrados en la aplicación.		
Método HTTP	GET		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
	users	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 34 - Mantenedor de Políticas de seguridad - Acción **/user/list**

Guardar Usuario			
URL de acceso	/user/save		
Descripción	Guarda un usuario nuevo, lo actualiza en caso de existir.		
Método HTTP	POST		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	user	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
	users	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 35 - Mantenedor de Políticas de seguridad - Acción **/user/save**

Eliminar Usuario			
URL de acceso	/user/delete		
Descripción	Elimina un usuario.		
Método HTTP	POST		
Content-Type	application/json		
Mensaje de Entrada	**Parámetro**	**Tipo**	**Opcional**
	user	String	No
Mensaje de Salida	**Parámetro**	**Tipo**	**Opcional**
	result	String	No
Mensaje de Error	**Parámetro**	**Tipo**	**Opcional**
	error	String	No
	description	String	No

Tabla 36 - Mantenedor de Políticas de seguridad - Acción **/user/delete**

Anexo (C) Descripción de Tablas de Base de Datos

Tabla OAUTH_CLIENT_APP

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	INT	SI	SI	Identificador interno de la tabla
NAME	NVARCHAR(40)	SI		Nombre de la aplicación cliente
GENERATE_ID	NVARCHAR(50)	SI		Client_ID para la aplicación cliente
CLIENT_SECRET	NVARCHAR(50)	SI		Clave secreta para la aplicación cliente
DESCRIPTION	NVARCHAR(2000)	SI		Descripción de la Aplicación Cliente
HOME_URI	NVARCHAR(100)	SI		URL de la Aplicación Cliente
CALLBACK_URI	NVARCHAR(100)	SI		URL de respuesta de la Aplicación Cliente
TIME_EXPIRATION	INT	SI		Tiempo de expiración para los *Tokens* generados
STATE	NVARCHAR(50)	NO		Estado de la Aplicación Cliente
USER_CREATION	NVARCHAR(100)	SI		ID del usuario que creó la Aplicación Cliente
DATE_CREATION	DATETIME	SI		Fecha de Creación

Tabla 37 - Base de Datos - Tabla OAUTH_CLIENT_APP

Tabla OAUTH_CLIENT_CODE

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	INT	SI	SI	Identificador interno de la tabla
CLIENT_ID	INT	SI	SI	Identificador interno de tabla OAUTH_CLIENT_APP
CODE	NVARCHAR(250)	SI		Código generado para Aplicación Cliente
DATE_CREATION	DATETIME	SI		Fecha de creación del Código

Tabla 38 - Base de Datos - Tabla OAUTH_CLIENT_CODE

Tabla OAUTH_CLIENT_GRANT

Nombre	Tipo	Requerido	PK/FK	Descripción
CLIENT_ID	NUMERIC(9)	SI	SI	Identificador interno de tabla OAUTH_CLIENT_APP
GRANT_ID	NUMERIC(9)	SI	SI	Identificador interno de la tabla

Tabla 39 - Base de Datos - Tabla OAUTH_CLIENT_GRANT

Tabla OAUTH_CLIENT_TOKEN

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	INT	SI	SI	Identificador interno de la tabla
CLIENT_ID	INT	SI	SI	Identificador interno de tabla OAUTH_CLIENT_APP
TOKEN	NVARCHAR(50)	SI		Token generado para Aplicación Cliente
DATE_CREATION	DATETIME	SI		Fecha de creación de Token

Tabla 40 - Base de Datos - Tabla OAUTH_CLIENT_TOKEN

Tabla OAUTH_CLIENT_URI

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	INT	SI	SI	Identificador interno de la tabla
CLIENT_ID	INT	SI	SI	Identificador interno de tabla OAUTH_CLIENT_APP
URI	NVARCHAR(100)	SI		URL permitida.

Tabla 41 - Base de Datos - Tabla OAUTH_CLIENT_URI

Tabla OAUTH_ GRANT

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	INT	SI	SI	Identificador interno de la tabla
NAME	NVARCHAR(20)	SI		Nombre del tipo de Grant
DESCRIPTION	NVARCHAR(300)	SI		Descripción del Grant

Tabla 42 - Base de Datos - Tabla OAUTH_GRANT

Tabla OAUTH_REFRESH_TOKEN

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	INT	SI	SI	Identificador interno de la tabla
TOKEN_ID	INT	SI	SI	Identificador interno de tabla OAUTH_CLIENT_TOKEN
REFRESH_TOKEN	NVARCHAR(250)	SI		Refresh Token para refrescar Token
DATE_CREATION	DATETIME	SI		Fecha de Creación

Tabla 43 - Base de Datos -Tabla OAUTH_REFRESH_TOKEN

Tabla OAUTH_USER

Nombre	Tipo	Requerido	PK/FK	Descripción
ID	NVARCHAR(50)	SI	SI	Identificador interno de la tabla
NAME	NVARCHAR(50)	SI		Nombre del usuario
USER_CREATION	NVARCHAR(50)	SI		Usuario creador
DATE_CREATION	DATETIME	SI		Fecha de Creación
PERFIL	NVARCHAR(10)	SI		Indica Tipo de Usuario

Tabla 44 - Base de Datos - Tabla OAUTH_USER