UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo para alumnos con experiencia profesional

TÍTULO: Seguridad en Interoperabilidad de aplicaciones del ecosistema e-Sidif

AUTOR: Martín Ignacio Vacas

DIRECTOR ACADÉMICO: Andrés Rodríguez

DIRECTOR PROFESIONAL: Daniel Horrisberger

CARRERA: Licenciatura en Sistemas

Resumen

El e-Sidif es una aplicación Enterprise, implementada por la DGSIAF dentro del Ministerio de Economía. En su ecosistema existen diversas aplicaciones Web, estas se comunican e intercambian información con él para realizar distintas tareas. El presente trabajo de tesis tiene como principal objetivo exponer el análisis, diseño e implementación de un sistema de seguridad centralizado para dichas aplicaciones.

Palabras Clave

- *Single Sign On (SSO)*
- *OAuth 2.0*
- *Arquitectura de Microservicios*
- *Web Services*
- *Sistemas Enterprise.*

Conclusiones

Con la utilización de un framework maduro, como lo es Spring Security, y haciendo uso de distintos componentes que este brinda, se logró desarrollar la aplicación DAUT. Permitiendo así a los usuario de las aplicaciones Web de la DGSIAF que usen a este como Servidor de Autenticación y Autorización, tener un único punto de entrada, y contar con un ambiente seguro que brinda SSO.

Trabajos Realizados

Se describió el origen del e-Sidif, se hizo un recorrido histórico sobre su evolución hacia una arquitectura de Microservicios.
*Se presentó **DAUT**, aplicación que centraliza la autenticación de los usuarios de las aplicaciones web de la DGSIAF, implementa políticas de seguridad para las mismas, y brinda un entorno de SSO.*
Se describió cómo fueron adaptadas algunas funcionalidades y características provistas por el framework SpringSecurity para implementar dicha aplicación.
*Se describieron los pasos a seguir para que una aplicación Web de la DGSIAF haga uso de DAUT usando una librería llamada **daut-connector.***

Trabajos Futuros

Como trabajo futuro podría analizarse la posibilidad de extender DAUT para dar soporte a otro tipo de aplicaciones, cómo pueden ser apps móviles u otro cliente distinto a las aplicaciones Web, como podría ser backends de otras aplicaciones del estado.

Se podría realizar una comparación con otro framework open source maduro en la industria y analizar costos, beneficios, eficiencia, robustez y simpleza en la implementación.

Fecha de la presentación: Agosto 2022

UNIVERSIDAD NACIONAL DE LA PLATA

Facultad de Informática

Seguridad en interoperabilidad de aplicaciones del ecosistema e-Sidif

Alumno: Martín Ignacio Vacas.
Director Académico: Andrés Rodríguez.
Director Profesional: Daniel Horrisberger.

ÍNDICE

CAPÍTULO 1 - Introducción

1.1 - Motivación

El e-Sidif [1] es un sistema de gran tamaño, diseñado para sustentar la diversidad de tareas y roles que requiere la administración financiera de la Administración Pública Nacional. El proyecto se desarrolla en el ámbito de la Dirección General de Sistemas Informáticos de Administración Financiera (en adelante DGSIAF [2]) desde hace más de 15 años y al presente sigue sumando servicios y nueva funcionalidad que surgen a través de nuevos requerimientos.

El e-Sidif soporta la formulación del presupuesto nacional y el registro completo de la ejecución presupuestaria.

Es un Sistema Enterprise [3] de gran escala, como consecuencia posee las características inherentes a este tipo de sistemas. En la actualidad hay diversas aplicaciones web, las cuales requieren una comunicación segura y efectiva con el sistema central en cuestión.

Las políticas de acceso, autorizaciones, y todo lo relacionado a seguridad de las aplicaciones web son centralizados en un sistema llamado DAUT.

En este sistema DAUT y en sus particularidades se va a hacer foco y se toma como caso de estudio para la realización de esta tesina.

1.2 - Objetivo

El objetivo del presente trabajo es mostrar las soluciones para un Sistema complejo que requiere interoperabilidad con un conjunto de aplicaciones, en su mayoría Aplicaciones WEB, desarrolladas en el ámbito de la Secretaría de Hacienda de la Nación (DGSIAF).

En este sentido, se prevé exponer los mecanismos de comunicación entre procesos e intercambio de datos, poniendo énfasis en los temas de seguridad asociados (identificación/autenticación de usuarios y permisos/derechos de usuarios) y la dinámica de mantenimiento y actualización de los mismos.

El análisis de los problemas derivados de la interoperabilidad y la seguridad se relaciona con los instrumentos de desarrollo empleados en el sistema y las aplicaciones, indicando aspectos favorables y dificultades.

Por último se compara con alternativas posibles para la gestión de la seguridad, justificando las decisiones adoptadas.

1.3 - Estructura de la Tesina

La estructura del presente trabajo consta de 7 capítulos, los cuales serán descritos brevemente a continuación.

En el ***capítulo 1 "Introducción"***, como su nombre indica se hace una introducción al presente trabajo, se presenta la motivación que llevó a realizar dicho trabajo, se presentan los objetivos a cubrir y se describe la estructura de la tesina.

En el ***capítulo 2 "El ecosistema e-Sidif"***, se hace una introducción a la Dirección General de Sistemas Informáticos de Administración Financiera (DGSIAF), se presenta la evolución desde los principios del proyecto hasta la actualidad. Se describen características del sistema e-Sidif, como este fue evolucionando hacia una arquitectura orientada a microservicios, en la cual entran en juego distintas aplicaciones web que se comunican con el.

El ***capítulo 3 "Seguridad"*** introduce conceptos básicos de seguridad, la política de seguridad adoptada en e-Sidif, como es el manejo de permisos y autorizaciones, y como se empieza a pensar en una aplicación que se encargue de todo lo relacionado a el acceso, autorizaciones y seguridad.

En el ***capítulo 4 "DAUT"*** se describe cómo surgió la aplicación, qué objetivo cumple y el alcance de la misma. Se hace una breve descripción del concepto de SSO (Single Sign On) el cual es llevado a cabo en el ámbito de las aplicaciones Web de la DGSIAF gracias a DAUT. También se hace una descripción del protocolo OAuth 2.0, estándar en el que se basa el desarrollo de la presente aplicación.
Se describe el flujo para la obtención de un token de acceso, presentando un ejemplo real con su descripción paso a paso.

El ***capítulo 5 "Componentes de Spring Security"***, describe cómo fueron implementados los distintos componentes de **DAUT** y la librería ***DAUT-CONNECTOR*** para llevar a cabo el flujo de obtención de token de acceso. Antes se hace una introducción al manejo de filtros y la cadena de filtros en aplicaciones basadas en Servlets. Luego se describen conceptos de Spring

Security, que es el framework sobre el que fueron implementados tanto **DAUT** como la librería ***DAUT-CONNECTOR.***

En el ***capítulo 6 "Implementación del sistema de seguridad centralizado"***, se exponen detalles de la implementación y configuración de los principales componentes que conforman el sistema. Se describen los pasos a seguir para configurar ***DAUT-CONNECTOR*** dentro de una aplicación web para lograr la comunicación y obtención del token de acceso provisto por **DAUT**.

Por último se dejan las conclusiones finales en el ***capítulo 7 "Conclusiones"***.

CAPÍTULO 2 - El ecosistema e-Sidif

2.1 - Introducción

La denominada “DGSIAF” es la Dirección General de Sistemas Informáticos de Administración Financiera. Desarrolla y mantiene el sistema integrado de Administración Financiera (SIAF) y un conjunto de sistemas que dan soporte a la gestión presupuestaria financiera y contable del sector público.

Dentro del conjunto de sistemas que hace referencia el párrafo anterior, se encuentra el e-Sidif y un conjunto de aplicaciones satélites las cuales cooperan para llevar a cabo el objetivo de formular el presupuesto nacional y el registro completo de la ejecución presupuestaria.

2.2 - Evolución histórica del e-Sidif

En el año 1991 se crea la Unidad Informática a través del Decreto 2737/92 encargada del desarrollo, la puesta operativa y el mantenimiento de la infraestructura tecnológica de las primeras versiones del sistema de administración financiera. En el mismo año también fue sancionada la Ley Nº 24.156 de Administración Financiera y de los Sistemas de Control del Sector Público Nacional a partir de allí, se inició el camino en la implementación del sistema e-Sidif, desde sus inicios el mismo fue variando, atravesando diversas transformaciones, hasta lo que es en la actualidad.

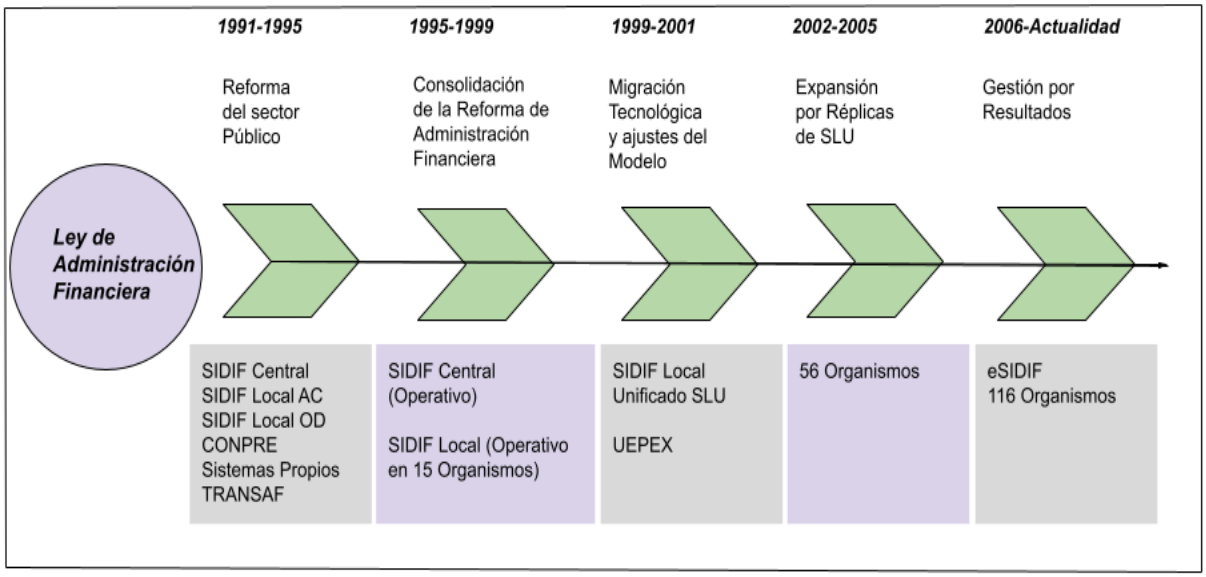

Figura 1 - Evolución Histórica

Entre los años 1991 y 1995 se realizó una renovación del Sector Público, la cual dio origen al Sistema Integrado de Información Financiera (SIDIF).

El SIDIF tenía como principal objetivo la formulación presupuestaria nacional y el registro de la ejecución presupuestaria. El mismo fue pensado inicialmente como un sistema integrado, compuesto por varios subsistemas y módulos dentro de un enfoque funcional, con una base de datos central, la cual estaba relacionada con bases de datos locales de los Servicios de Administración Financiera (SAF).

La Secretaría de Hacienda se conectaba con los sistemas locales y el resto de las aplicaciones a través del SIDIF Central. Las bases de datos locales guardaban la información de gestión y la base central agrupaba el registro de la ejecución presupuestaria.

Dentro de esta estructura, coexisten una diversidad de sistemas locales en los distintos organismos con diferentes características, esto generaba un incremento en el costo de mantenimiento y el retardo que implicaba replicar los ajustes que eran necesarios para los sistemas locales.

En el periodo comprendido entre los años 1999 y 2001 se produjo una migración tecnológica y ajustes en el modelo, que marcaron un hito en la evolución de este sistema. En ese momento la Subsecretaría de Presupuesto comenzó a desarrollar un nuevo sistema de administración financiera para los SAF, llamado SIDIF Local Unificado ***"SLU"***.

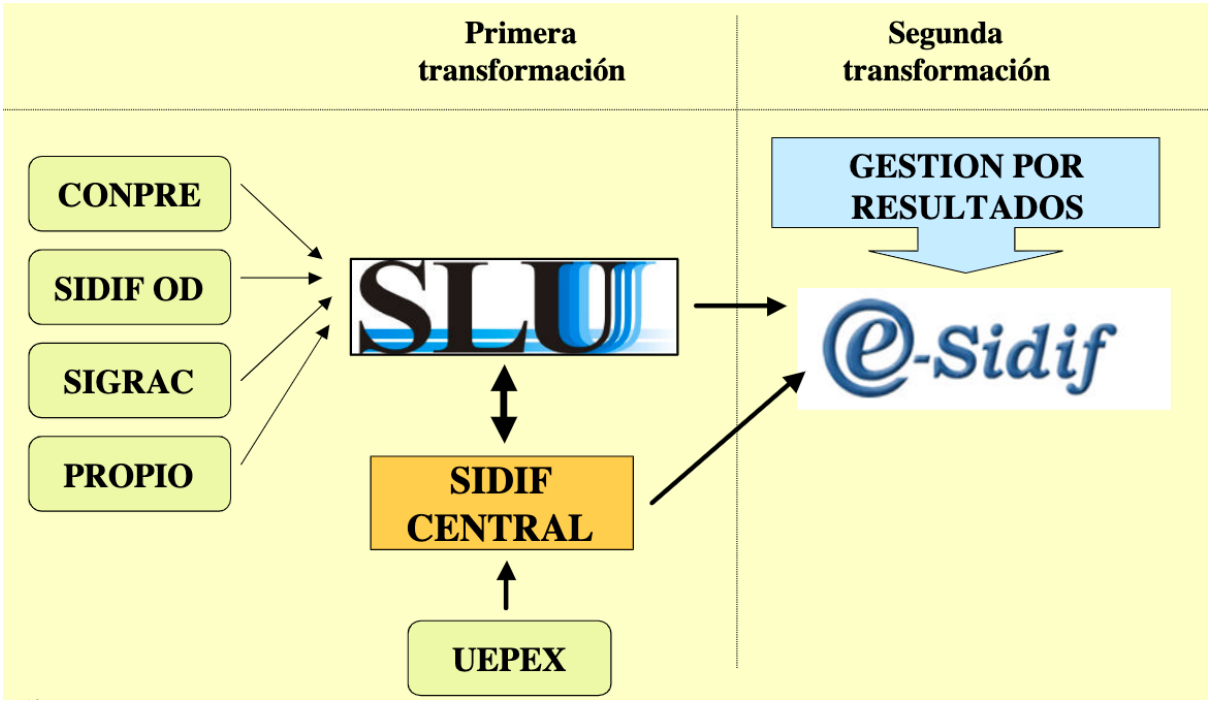

Figura 2 - Primera y Segunda Transformación del e-Sidif

En el periodo comprendido entre los años 2002 y 2005, la Secretaría de Hacienda comenzó un proceso de réplica masiva del SLU en Organismos de la Administración, con el objeto de reemplazar los sistemas existentes y racionalizar el mantenimiento de los mismos.

Cumplida esta primera etapa de transformación la cual introdujo importantes mejoras a través del SLU, se plantea un nuevo desafío, la denominada ***"Segunda transformación"***, que se materializa, con la extensión del alcance y renovación tecnológica, en el desarrollo del sistema e-Sidif, llevado a cabo a principios del año 2006.

Desde el punto de vista tecnológico, e-Sidif fue concebido como un producto adaptable a diferentes entornos y tamaños organizacionales, con características visuales y de navegación que facilita su uso, con una base de datos que centraliza toda la información de la gestión de los organismos ejecutores y de los órganos rectores, con una arquitectura en capas que simplifica y reduce los costos de mantenimiento y evolución del sistema.

Fue incorporada además la capacidad de interoperar con sistemas de otros organismos del Estado Nacional, a través del uso de servicios web.

Ya dentro del último período presentado en la línea del tiempo, en 2009, una vez logrado un avance significativo en el desarrollo, despliegue y madurez de e-Sidif, la

subsecretaría de Presupuesto promovió la incorporación de soluciones modernas de inteligencia de negocios (BI).

Más tarde, en 2011, se dio comienzo a la implementación gradual de firma digital. Este avance profundizó el proceso de despapelización de la Administración Nacional y la descentralización de la carga de información.

Acorde a las innovaciones tecnológicas, en el marco del gobierno electrónico, en 2012, se implementó un servicio de consulta por internet accesible desde computadoras o equipos móviles para los proveedores y contratistas del Estado, llamado e-Prov.

Con el correr de los años y a partir del surgimiento de nuevas necesidades, se fueron desarrollando nuevos sistemas vinculados al e-Sidif destinados a diferentes usuarios (empresas públicas, ciudadanos, entre otros).

Paralelamente se desarrollaron sistemas para usuarios internos de la administración pública nacional, que ayudan a la consulta, seguimiento y gestión de información.

2.3 - Características

Como se menciona anteriormente en el capítulo 1, el e-Sidif es un Sistema Enterprise de gran tamaño, este tipo de sistemas presentan ciertas características que las distinguen de otros sistemas:

- Persistencia de datos
    - Los datos que se van generando a través de las distintas operaciones que brinda el sistema deben ser almacenados. La persistencia de los datos asegura que los mismos queden disponibles para futuras operaciones. Los datos deben soportar cambios de hardware y migraciones a nuevas estructuras de persistencia.

- Grandes volúmenes de datos
    - Los procesos administrativos generan importantes volúmenes de datos. Es por esto que muchas de las operaciones que brindan las aplicaciones enterprise generan gran cantidad de información. De hecho, estas aplicaciones suelen proveer la posibilidad de ejecutar las

operaciones de forma masiva y de forma batch, generando más información en menos tiempo y con menos trabajo.

- Grandes volumenes de usuarios
  - Este tipo de aplicaciones es usada por una gran cantidad de usuarios, los cuales presentan diferentes perfiles con sus respectivos permisos, esto les permite acceder y/o manipular un cierto conjunto de datos.

- Acceso concurrente de datos
  - Debido a que las aplicaciones enterprise son multiusuario, la información será accedida por varios usuarios a la vez. Durante este acceso concurrente se deben garantizar las 4 propiedades ACID: Atomicidad, Consistencia, Aislamiento y Durabilidad.
    - Atomicidad. Una transacción debe ser tratada como una unidad de procesamiento. Por lo tanto, todas las acciones de la transacción se realizan o ninguna de ellas se lleva a cabo. Asimismo, si una transacción se interrumpe por una falla, sus resultados parciales deben ser desechados.
    - Consistencia. Los recursos del sistema deben estar en un estado consistente y no-corruptos al comenzar la transacción y finalizada su ejecución. Las transacciones no deben violar las restricciones de integridad de un sistema.
    - Aislamiento. Una transacción en ejecución no puede revelar sus resultados a otras transacciones concurrentes antes de su completitud. Más aún, si varias transacciones se ejecutan concurrentemente, los resultados deben ser los mismos que si ellas se hubieran ejecutado de manera secuencial (seriabilidad).
    - Durabilidad. Se debe asegurar que una vez que una transacción finaliza, sus resultados son permanentes y no pueden deshacerse.

- Presentación de los datos de diferentes maneras
  - Es necesario presentar los datos de diferentes maneras considerando los distintos perfiles de los usuarios y las distintas operaciones que provee el sistema.

- Integración con otros sistemas
    - Es muy común en sistemas enterprise consumir y/o exponer servicios con otros sistemas. Esta interacción podrá ser con otros sistemas de la misma organización, con sistemas de otras organizaciones que proveen servicios y con aplicaciones que tienen como fin vender servicios a aplicaciones enterprise entre otro tipos de integración. El uso de Web Services facilita la integración de aplicaciones merced a la estandarización de protocolos y formatos utilizados para la comunicación e interpretación de los pedidos y respuestas.

- Lógica de negocio muy compleja y evolutiva
    - La lógica de negocio en este tipo de sistemas es muy compleja, es común el uso de reglas. Estas reglas describen las operaciones, definiciones y restricciones que se aplican para alcanzar el objetivo del sistema.

2.4 - Arquitectura

Como se describió en puntos anteriores, el sistema e-Sidif surgió como una etapa de actualización de un software de la administración financiera del estado. Esta etapa ocurrió como una transición en donde el nuevo sistema debió convivir y relacionarse con otros sistemas existentes a los cuales tuvo que proveer de una interfaz clara y definida.

A diferencia del sistema que reemplazó, el e-Sidif utiliza una Base de Datos única que centraliza toda la información para una gestión unificada, y además permite una descentralización operativa.

La aplicación tiene un alto enfoque transaccional y de integración, que está basado sobre estándares ampliamente difundidos en la industria. Uno de los conceptos mencionados es el de SOA [4].

SOA está basado sobre dos principios, ***modularidad*** – subdivide el sistema en partes más pequeñas (módulos) – y ***encapsulamiento*** – utiliza una interfaz claramente definida para aislar las características internas del exterior.

Estos principios permiten que las aplicaciones sean de fácil desarrollo y mantención, puesto que se puede alcanzar una clara separación de roles y alcanzar el desarrollo de componentes que luego puedan ser reutilizados. Al ser independiente de la tecnología permite que mundos diferentes puedan intercomunicarse.

El sistema e-Sidif está dividido en módulos funcionales, cada uno cumple una función específica y además cooperan con otros módulos para llevar a cabo otras tareas.

A continuación se enumeran algunos módulos del sistema:

- Pagos
- Gastos
- Compras
- Conciliación Bancaria
- Contabilidad
- Fondos Rotatorios

Dichos módulos exponen interfaces definidas a servicios las cuales son usadas para la comunicación e intercambio de información entre módulos del sistema.

Estas interfaces nombradas anteriormente, también son expuestas al exterior, para que cualquier aplicación del ecosistema que requiera el intercambio de información con algún módulo específico, pueda consumir los servicios expuestos mediante el protocolo SOAP [5].

Asimismo la evolución de la arquitectura del e-Sidif incorporó :

- Modelo de N-Capas permitiendo mayor modularidad y flexibilidad [6].
- Independencia o portabilidad de sus componentes de infraestructura de software y hardware.
- Capacidades más sencillas de deployment.

El estilo arquitectónico adoptado para el e-Sidif organiza el sistema en capas, las cuales se comunican para poder completar la funcionalidad requerida. Cada capa tiene una funcionalidad y un alcance bien definido.

A continuación se describe a alto nivel un esquema que divide al sistema en 3 capas:

- Presentación
- Negocio
- Datos

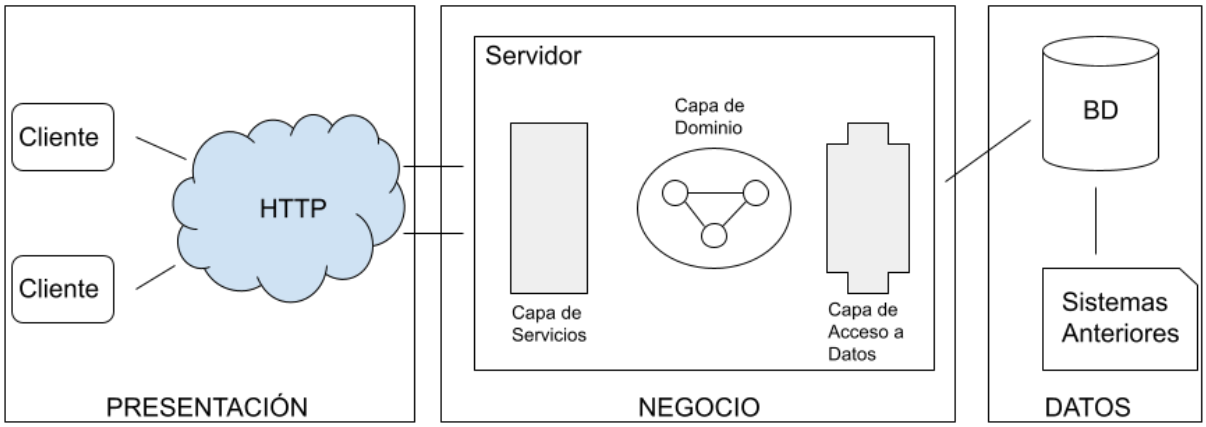

Figura 3 - Arquitectura de 3 capas

La capa de presentación conforma parte de la “vista” o interfaz gráfica [GUI] de la aplicación. El hecho de tener la presentación como una capa separada de la lógica de negocios permite desacoplar la lógica de negocios de la presentación permitiendo alterar la interfaz de la aplicación o agregar un nuevo canal sin necesidad de alterar la lógica de negocios. Esta independencia del modelo de servicios permite acoplar rápidamente nuevas tecnologías de presentación preservando la inversión a realizar.

Entre los clientes del e-Sidif se destacan sistemas que consumen web services y el cliente de escritorio que utilizan a diario los usuarios operativos de este sistema. Este cliente, está basado en tecnología Java-RCP (Rich Client Platform) [7]. Para evitar la instalación de este cliente en las computadoras de los usuarios y hacerlo accesible a través de la web, se utiliza una tecnología de virtualización (Citrix).

En la capa de Negocio se ejecutan las reglas específicas del negocio. Recibe peticiones desde la capa de presentación que deberá resolver aplicando las reglas que correspondan. Se debe garantizar la ejecución de reglas controlando los permisos definidos como así también el registro de la ejecución para una posterior auditoría.

Esta capa brinda acceso a las reglas de negocio no solo a la capa de presentación sino también a otros sistemas autorizados que así lo requieran. Para esta integración utiliza web services.

La capa de datos se implementa sobre una base de datos Oracle. Esta capa es la responsable de almacenar los datos de la aplicación. Además existen otras fuentes/destinos de datos, entre las cuales se encuentra el sistema legado.

2.5 - Hacia un enfoque orientado a microservicios

Conforme la industria fue avanzando, se fue migrando desde una arquitectura monolítica hacia una arquitectura orientada a microservicios.

Un enfoque monolítico define una estructura en la que toda la funcionalidad está fuertemente acoplada y sujeta a una sola implementación. Por ende se tiene un único WAR a desplegar. En este tipo de arquitectura cualquier pequeño cambio o actualización requiere compilar, implementar y desplegar completamente una nueva versión de la aplicación.

Además, las aplicaciones monolíticas son más difíciles de escalar. Cuando una aplicación monolítica alcanza una limitación de su capacidad, como el rendimiento en el procesamiento de datos o algún otro cuello de botella, la única alternativa práctica es tener otra instancia de la aplicación monolítica y administrar el tráfico entre las instancias utilizando balanceadores de carga.

La arquitectura de microservicios tiene como objetivo aislar los distintos componentes de una aplicación, con el fin de que cada uno sea una aplicación por sí misma. Cada componente se ejecuta en un proceso único y generalmente administra su propia base de datos. El paradigma de microservicio proporciona un enfoque más descentralizado para construir software. Los microservicios permiten que cada servicio sea aislado, construido, desplegado y administrado de forma independiente.

Podríamos considerar que los microservicios son una evolución del Service Oriented Architecture o SOA, cuya función se basa en desarrollar servicios independientes para el negocio, estando cada uno de estos asociados o unidos a una misma aplicación.

A diferencia de SOA, los microservicios permiten que un componente específico del mismo, evolucione más allá de sus capacidades, ya sea dividiéndolo en elementos más pequeños o dotándola de mayores recursos.

Con un horizonte definido, las nuevas funcionalidades se comenzaron a pensar e implementar bajo estas políticas.

El enfoque orientado a microservicios adoptado, presenta grandes ventajas respecto a otros enfoques, entre ellas:

- Reutilización de componentes.
- Interoperabilidad entre aplicaciones con tecnologías heterogéneas.

- Reducción en los costos de mantenimiento.
- Facilidad de adaptación a cambios.
- Escalabilidad.
- Se despliegan fácilmente.
- Requieren menos tiempo de desarrollo.
- Contienen mejor aislamiento de fallas.
- Puede ser desplegado en equipos relativamente pequeños.
- Trabaja bien con los contenedores.

Este nuevo enfoque, permitió tener aplicaciones Web con responsabilidades más acotadas, y desacoplar comportamiento que antes estaba centralizado en un único sistema. Estas aplicaciones interactúan con el e-Sidif para llevar a cabo sus respectivos objetivos.

Algunas de las aplicaciones en cuestión son:

- SIRECO. [8]
- SICHE. [9]
- MENU
- SICDE. [10]

CAPÍTULO 3 - Seguridad

3.1 - Introducción

En este capítulo se definen algunos conceptos básicos relacionados a la seguridad en sistemas de grandes organizaciones, que en este caso también fueron adoptados por la política de seguridad implementada en e-Sidif. La mayoría de estos conceptos son comunes a todos los sistemas en los cuales se implementa una política de seguridad.

3.2 - Conceptos básicos

Los mecanismos de seguridad adoptados en los sistemas de grandes, medianas o pequeñas organizaciones, deben garantizar en todo momento la confidencialidad, integridad de datos y disponibilidad.

Confidencialidad: La información solo debe de ser vista y/o utilizada por los usuarios autorizados para tener acceso a ella.

Integridad: Prevenir e identificar cualquier cambio a la información por un usuario no autorizado.

Disponibilidad: La información debe estar disponible cuando los usuarios autorizados la necesiten.

Algunos de los mecanismos utilizados para garantizar los objetivos enumerados anteriormente son:

- Autenticación
- Autorización
- Integridad
- Auditoría

Autenticación: se define como la verificación de la identidad del usuario cuando entra en el sistema. Generalmente para entrar en un sistema informático se utiliza un nombre de usuario y una password. Pero, cada vez más se están utilizando otras

técnicas más seguras, como por ejemplo, usar un segundo factor de autenticación, esto significa añadir un paso más al proceso de la autenticación a través de nombre de usuario y password .

Autorización: se define como el proceso por el cual se determina qué, cómo y cuándo, un usuario autenticado puede utilizar los recursos del sistema.

El mecanismo o el grado de autorización puede variar dependiendo de qué sea lo que se está protegiendo. No toda la información de la organización donde se está implementando el mecanismo de seguridad es igual de crítica. Los recursos en general y los datos en particular, se organizan en niveles y cada nivel debe tener una autorización.

La autorización debe asegurar la confidencialidad e integridad, ya sea dando o negando el acceso en lectura, modificación, creación o borrado de los datos.

Por otra parte, solo se debe dar autorización a acceder a un recurso a aquellos usuarios que lo necesiten para hacer su trabajo, y si no se le negará. Aunque también es posible dar autorizaciones transitorias o modificarlas a medida que las necesidades del usuario varíen.

Auditoría: se define como la continua vigilancia de los servicios en producción y para ello se recaba información y se analiza.

Este proceso permite verificar que las técnicas de autenticación y autorización utilizadas se realizan según lo establecido y se cumplen los objetivos fijados por la organización.

Monitorear la información registrada o auditar se puede realizar mediante medios manuales o automáticos, y con una periodicidad que dependerá de lo crítica que sea la información protegida y del nivel de riesgo.

Integridad: se define la integridad de la información como el conjunto de procedimientos establecidos para evitar o controlar que los datos sufran cambios no autorizados y que la información enviada desde un punto llegue al destino inalterada.

3.3 - Capacidades, Roles y Usuarios

A continuación se describen los conceptos de Capacidades, Roles y Usuarios, utilizados en los mecanismos de autenticación y autorización en e-Sidif.

La ***capacidad*** representa un permiso de ejecución de una acción (servicio) o conjunto de ellas. En cierta forma específica la visión que tendrá el usuario, ya que en función de las capacidades asignadas, el usuario verá habilitadas las acciones que podrá ejecutar en el sistema.

Los ***roles*** se clasifican en:

- **Simples**: contienen una única capacidad.
- **Compuestos**: agrupa varios roles, los cuales pueden ser simples o compuestos.

Los roles se asignan a los usuarios y de esta manera se les otorgan los permisos de ejecución sobre las distintas operaciones del sistema.

El **Rol** es el elemento de seguridad que poseen los usuarios a través de una asignación, según la funcionalidad los roles en e-Sidif se clasifican en ***roles ejecutores*** *o* ***roles otorgantes***.

- Un usuario que posea un ***rol ejecutor*** determinado, podrá ejecutar el servicio o ver las opciones del sistema indicadas por las capacidades incluidas en el rol.

- Un usuario administrador que posea un ***rol otorgante*** determinado, podrá asignar roles ejecutores a un usuario.

Los ***Usuarios*** son entidades que interactúan con el sistema, lo administran y ejecutan los distintos casos de uso.

Existen diferentes tipos de usuario:

- Usuarios Comunes
- Administradores Locales
- Administradores Generales
- Sistemas Externos

Usuarios Comunes: su función principal consiste en ejecutar los casos de uso de la aplicación. Los cuales serán visibles en función de los roles ejecutores que tenga asignados.

Administradores Locales: su tarea principal consiste en la administración de los usuarios comunes de un organismo. Sus principales funciones para llevar a cabo sus tareas son:

- Crear Usuarios
- Dar de baja Usuarios
- Rehabilitar la password de un Usuario
- Crear roles ejecutores simples y compuestos a partir de capacidades públicas
- Asignar roles ejecutores a Usuarios

Administradores Generales: su tarea principal consiste en la administración de usuarios administradores. Sus principales funciones para llevar a cabo sus tareas son:

- Crear y administrar Usuarios Administradores y Sistemas Externos
- Dar de baja Usuarios
- Rehabilitar Usuario administradores
- Crear roles otorgantes simples y compuestos
- Asignar roles otorgantes a Usuarios administradores

Sistemas Externos: consumen información del e-Sidif a través de web services expuestos por este. Estos sistemas pertenecen a un organismo del estado. A este tipo se les asignan Capacidades no roles.

3.4 - Autenticación y Autorización en e-Sidif

Autenticación (Quién soy?)

A través de un nombre de usuario y password se determina que el usuario es quien dice ser para permitirle ingresar al sistema.

La administración de nombres de usuarios y password, es llevada a cabo mediante el uso de un Servicio de Directorio a través de LDAP.

Un directorio es una base de datos, pero en general contiene información más descriptiva y basada en atributos. La información contenida en un directorio normalmente se lee mucho más de lo que se escribe. Como consecuencia estos no implementan normalmente los complicados esquemas para transacciones o esquemas de reducción que las bases de datos utilizan para llevar a cabo actualizaciones complejas de grandes volúmenes de datos.

Los directorios están para proporcionar una respuesta rápida a operaciones de búsqueda o consulta. Los directorios pueden tener capacidad de replicar información de forma amplia, con el fin de aumentar la disponibilidad y fiabilidad, y a la vez reducir tiempo de respuesta.

En e-Sidif, se hace uso de LDAP mediante OpenLDAP, que es una implementación del estándar.

LDAP ("Lightweight Directory Access Protocol") es un protocolo de tipo cliente-servidor para acceder a un servicio de directorio. LDAP es un estándar abierto de la industria que define un método para el acceso y la actualización de información en un directorio.

Características de un servidor LDAP:

- Operaciones de lectura muy rápidas. Debido a la naturaleza de los datos almacenados en los directorios las lecturas son más comunes que las escrituras.
- Datos relativamente estáticos. Los datos almacenados en los directorios no suelen actualizarse con mucha frecuencia.
- Entorno distribuido, fácil replicación.
- Estructura jerárquica. Los directorios almacenan la información de forma jerárquica de forma nativa.
- Orientadas a objetos. El directorio representa a elementos y a objetos. Los objetos son creados como entradas, que representan a una colección de atributos.
- Esquema estándar. Los directorios utilizan un sistema estándar que pueden usar fácilmente diversas aplicaciones.
- Replicación. Muchos de los servidores LDAP permiten que se realicen escrituras o actualizaciones en múltiples servidores.

Ventajas en el uso de LDAP:

- Es muy rápido en la lectura de registros.
- Permite replicar el servidor de forma muy sencilla y económica.
- Muchas aplicaciones de todo tipo tienen interfaces de conexión a LDAP y se pueden integrar fácilmente.
- Dispone de un modelo de nombres globales que asegura que todas las entradas son únicas.
- Usa un sistema jerárquico de almacenamiento de información.
- Permite múltiples directorios independientes.

Autorización (Qué puedo hacer?)

Se determina qué operaciones puede ejecutar el usuario en base a su tipo (administrador, común o súper usuario) y a los roles que tiene asignados.

Las operaciones que un usuario puede llevar a cabo en e-Sidif se representan mediante servicios, los cuales se clasifican en CRUD (create, read, update y delete) sobre una entidad y No CRUD.

Los servicios o acciones que un usuario del e-Sidif puede invocar, están agrupados en sus respectivas capacidades. Para determinar estos, se hace uso de un concepto denominado PIS (permiso de invocación a servicios) el cual se configura a través del uso de una herramienta denominada Co.S.E (configurador de seguridad) que la usan los desarrolladores de la aplicación.
Por lo que un usuario de la aplicación para ejecutar un servicio determinado, deberá contar con una capacidad que tenga el PIS correspondiente.

A continuación, en la imagen siguiente se muestra la configuración de un PIS y la capacidad necesaria para que un usuario pueda ejecutar un servicio, en este ejemplo se describe el servicio para realizar una consulta de Facturas de Gastos.

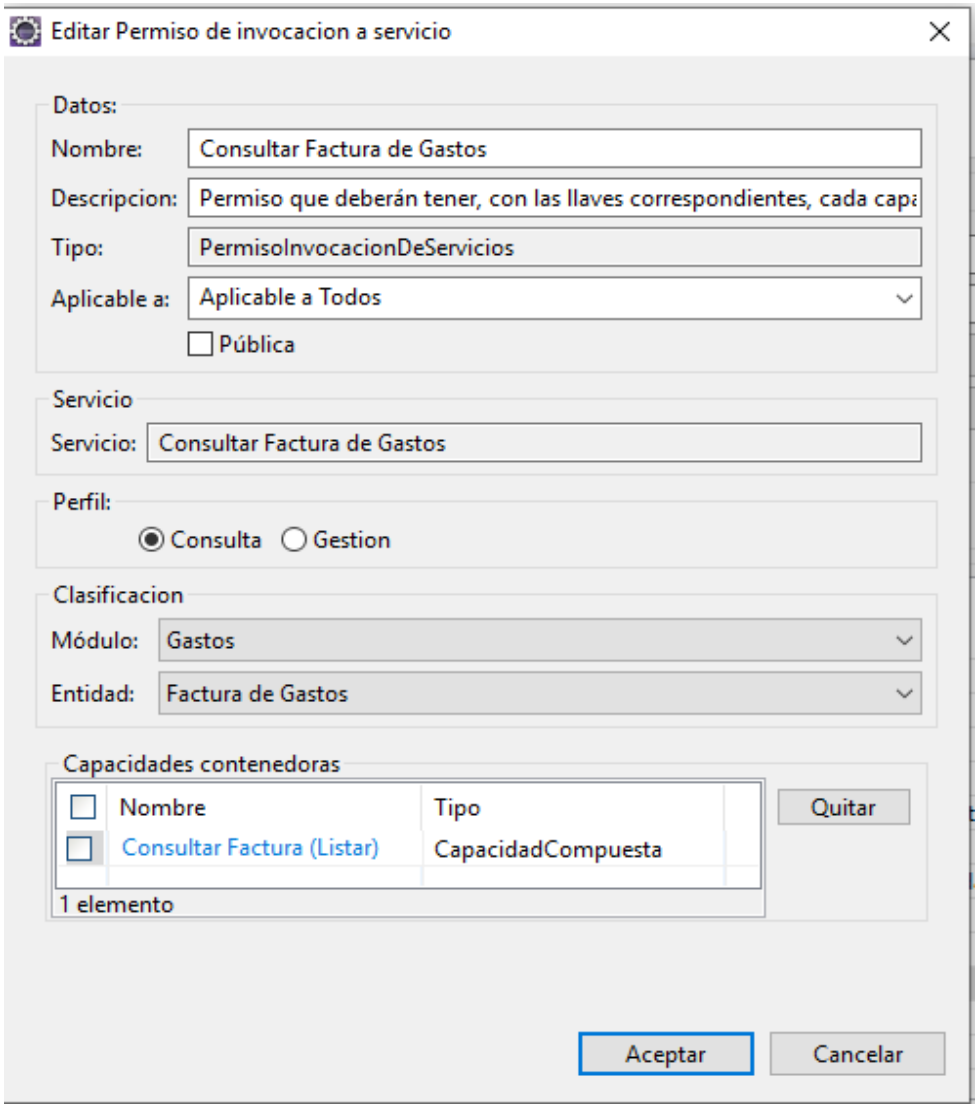

Figura 4 - Configuración de Capacidad

Para poder asignar el permiso para la lectura de Facturas, el PIS debe estar en una capacidad compuesta:

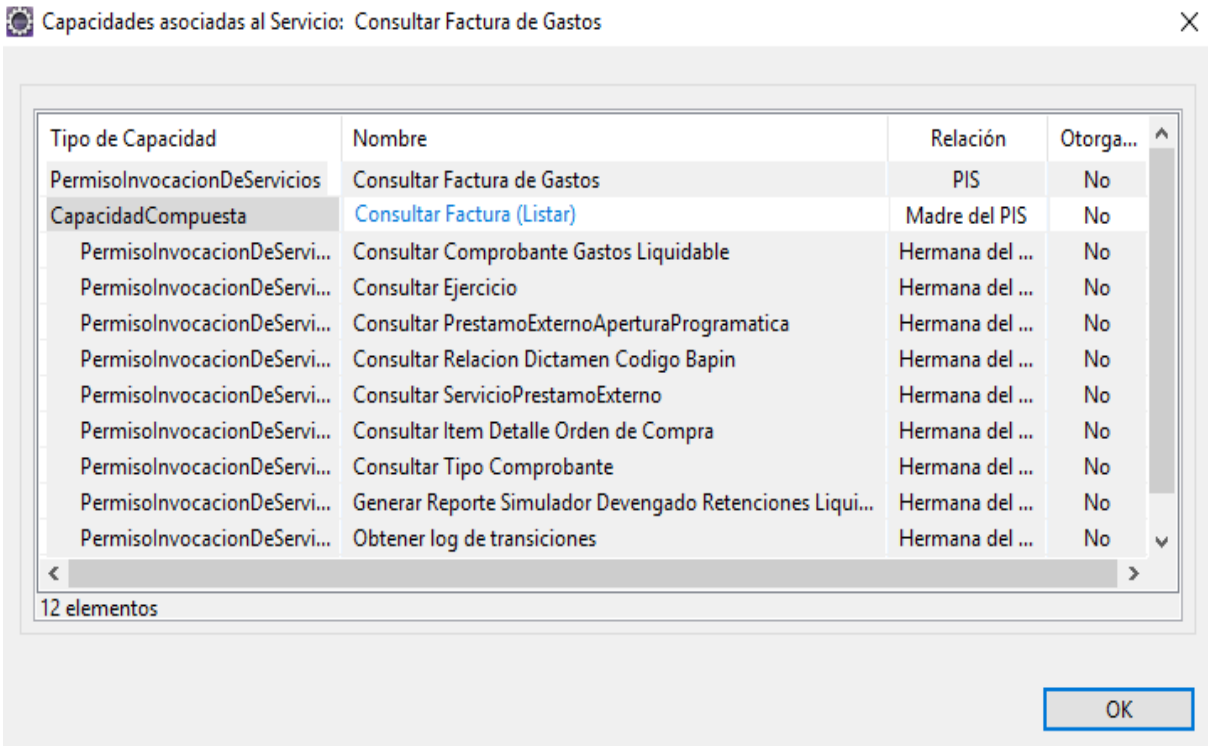

Figura 5 - Capacidades de Consulta Factura

Por último el usuario debería tener asociado un ROL que contenga dicha capacidad, y ya con este rol asignado, va a poder listar las Facturas de Gastos.

CAPÍTULO 4 - DAUT

4.1 - Introducción

En este capítulo se describe cómo surgió la aplicación ***DAUT,*** el objetivo para el cual fue desarrollada, el alcance y funcionamiento de la misma. También se describe el funcionamiento de una librería desarrollada en la DGSIAF llamada ***daut-connector.*** Esta librería debe ser importada por cualquier aplicación WEB que desee comunicarse con DAUT.

DAUT fue desarrollada haciendo uso de cierta funcionalidad que provee el framework Spring Security [11], adoptando características del estándar OAuth 2.0 [12].

Se describirán el propósito y funcionamiento de este protocolo, distintos flujos de trabajo, lo que ayudará a comprender mejor las distintas partes involucradas en el mecanismo de seguridad adoptado.

4.2 - Hacia un mecanismo de Seguridad centralizado

Como se mencionó en anteriores capítulos, la evolución del sistema e-Sidif hacia una arquitectura de microservicios, dio origen al desarrollo de Aplicaciones Web que conviven, comparten e intercambian información para llevar a cabo su propósito.

Todas estas Aplicaciones son utilizadas por el mismo conjunto de usuarios, por lo que requieren el chequeo de las políticas de acceso y seguridad que ya están implementadas en e-Sidif. También se requiere tener sincronizado los datos de login del usuario entre todos los sistemas.

Tener estas políticas replicadas en diferentes bases de datos y sistemas LDAPs, ó tener replicado la implementación del chequeo de autenticación y autorización, hace que la sincronización requerida no sea un objetivo fácil de cumplir.

Así es como surge DAUT, una aplicación que se encarga de centralizar la autenticación, las políticas de acceso y autorización de las aplicaciones web en el ámbito de la DGSIAF.

DAUT provee un único punto de autenticación para los usuarios que deseen acceder a las aplicaciones web provistas por la DGSIAF. Centralizando de esta manera, la implementación de las políticas de seguridad de usuario; y permitiendo tener una sola instancia de identificación de usuario, esto implementando el concepto de Single Sign On (SSO) el cual se describe en el punto 4.4.

4.3 - Alcance y Objetivo

En la primera etapa del proyecto, DAUT brindará apoyo solamente a aplicaciones WEB, permitirá Single Sign On (SSO) y validará usuarios ya dados de alta en e-Sidif. De esta manera los datos de login entre el e-Sidif y las aplicaciones WEB permanecerán unificados.

También en la primera etapa del proyecto, DAUT permitirá o rechazará el acceso de un usuario a una aplicación determinada, de acuerdo a las capacidades del mismo. Cada aplicación WEB define cuando un usuario tiene acceso o no a la misma.

Para lograr el objetivo por el cual fue desarrollado, DAUT se apoya y adopta algunas características del estándar OAuth 2.0 (el cual será descrito en puntos posteriores) para implementar un servidor de autorización para entregar un token de acceso. Esta y otras funcionalidades de OAuth 2.0, son implementadas haciendo uso del soporte que provee el framework Spring para el manejo de la seguridad (Spring Security).

Cuando un usuario quiera acceder a una determinada aplicación WEB de la DGSIAF, está lo redireccionará a DAUT, en el que se generará un token de acceso con la información que la aplicación necesita para conceder o no el acceso a dicho usuario, dependiendo de sus capacidades y otras políticas implementadas en DAUT, las cuales serán descritas en puntos posteriores.

4.4 - Single Sign On (SSO)

El concepto de SSO permite complementar los sistemas de seguridad de las organizaciones, incorporando un único punto de acceso a las diversas aplicaciones y/o recursos existentes.

El principal objetivo de implementar un esquema de SSO dentro de una organización, es proveer al usuario final la habilidad de contar con un único punto de acceso a las aplicaciones y/o recursos que necesita. En un esquema SSO, la identidad de los usuarios es verificada sólo una vez y su identidad es transferida a los diferentes recursos de forma segura y automática. Se logra así una mejor experiencia de usuario en el uso de las diferentes aplicaciones.

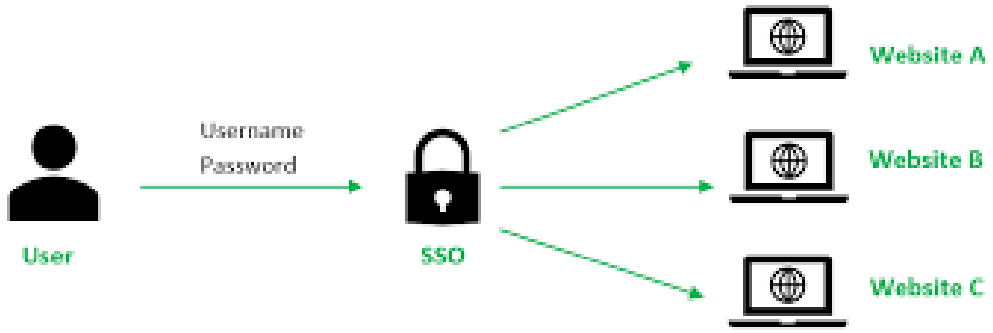

Figura 6 - Single Sign On

SSO es un proceso de autenticación en el que un usuario provee sus credenciales una sola vez para acceder a múltiples aplicaciones a las cuales tiene permitido.

Este proceso es llevado a cabo a través del uso de sesiones, que elimina los requerimientos a autenticaciones subsiguientes cuando el usuario cambia de aplicación en una sesión en particular.

Los beneficios de SSO son:

- Proveer una política de autenticación/autorización uniforme para toda la organización.
- Elimina la necesidad de que los desarrolladores de aplicaciones tengan que implementar el módulo de autenticación/autorización en sus aplicaciones.
- Reducción drástica de los pedidos de recuperación de passwords.
- En consecuencia mejora la usabilidad de las aplicaciones.

4.5 - OAuth 2.0

OAuth 2.0 es un estándar abierto de autorización, el cual a través de distintos flujos permite la autorización segura desde aplicaciones web, móviles y de escritorio de una manera simple.

Su funcionamiento se basa en delegar la autenticación de usuarios a otra aplicación, a través del consentimiento de un usuario, este permite que se realicen ciertas operaciones en su nombre (“Protocolo de Autorización Delegada”). Por ejemplo, permite que los usuarios compartan con otra aplicación sus recursos o datos privados (lista de contactos, documentos, fotos, vídeos, etc.), sin que estos tengan que informar sus credenciales (generalmente, el nombre de usuario y la contraseña).

Como se dijo anteriormente, OAuth 2.0 provee flujos de autorización tanto para aplicaciones Web, desktop y móviles.

4.5.1 - Entidades involucradas

En los diferentes flujos de OAuth hay varios actores claves involucrados, entre ellos:

- Resource Server (Servidor de recursos): Es un servidor que aloja los recursos protegidos de los usuarios, y estos están protegidos por OAuth. Suele ser una API que bloquea y protege los datos.

- Resource Owner (Propietario de los recursos): Normalmente es el usuario de una aplicación, el propietario del recurso tiene la capacidad de permitir el acceso a sus propios datos, alojados en el servidor de recursos. Es quien da autorización a una determinada aplicación para acceder a sus datos y poder hacer algunas operaciones en su nombre.

- Client (Cliente/Aplicación): Una aplicación que hace peticiones a la API para realizar acciones sobre recursos protegidos, en nombre del propietario del recurso y con su autorización.

- Authorization Server (Servidor de autorización): El servidor de autorización tiene el consentimiento del propietario del recurso y emite tokens de acceso a los clientes para acceder a los recursos protegidos alojados en un servidor de recursos

4.5.2 - Flujo abstracto de OAuth 2.0

A continuación se muestra el flujo general para la obtención de un recurso protegido. En este se describe cómo interactúan los roles anteriormente descritos:

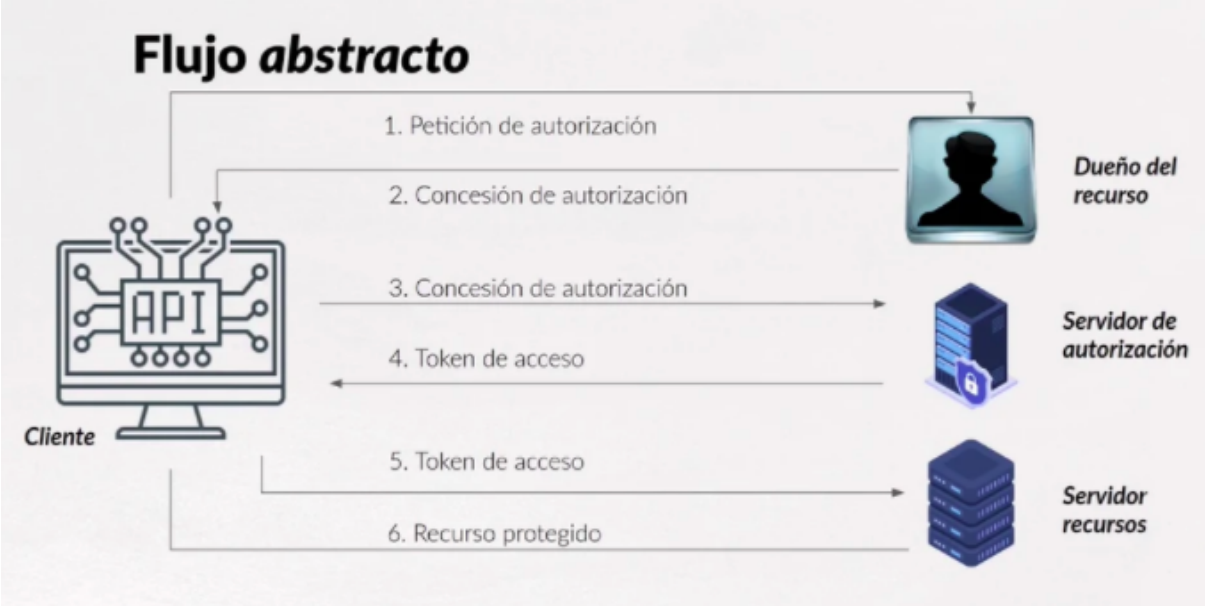

Figura 7 - Flujo Abstracto OAuth

1. El cliente solicita la autorización del propietario del recurso. La petición de autorización se puede hacer directamente al propietario del recurso (como muestra la imagen anterior), o de manera indirecta, la cual es preferible, usando como intermediario el Servidor de Autorización.

2. El cliente recibe una concesión de autorización (Authorization Grant), que es una credencial que representa la autorización del propietario del recurso, expresado mediante uno de los cuatro tipos de concesión definidos en este protocolo. El tipo de concesión depende del método utilizado por el cliente para solicitar autorización y los tipos soportados por el Servidor de Autorización.

3. El cliente solicita un token de acceso mediante la autenticación con el Servidor de Autorización presentando la concesión de autorización.

4. El Servidor de Autorización autoriza al cliente y valida la concesión de autorización. Si es válido, emite un token de acceso.

5. El cliente solicita un recurso protegido al Servidor de Recursos y se autentica mediante la presentación del token de acceso.

6. El Servidor de Recursos valida el token de acceso y, si es válido, atiende la solicitud

El flujo descrito anteriormente es una idea general de cómo funciona el estándar, puede diferir dependiendo de los tipos de concesiones usados.

4.5.3 - Concesión de Autorización (Authorization Grant)

Una concesión de autorización no es más que una credencial que representa la autorización dada por el propietario del recurso para que se pueda acceder a sus recursos, la cual va a ser usada por el cliente para obtener un token de acceso.

OAuth 2.0 define 4 tipos diferentes de concesiones de autorización. Para obtener cada uno de ellos existe un mecanismo o flujo definido, es decir, dependiendo del tipo de concesión de autorización, será el flujo que se lleve a cabo.

- Authorization Code.
- Client Credential.
- Device Code.
- Refresh Token.

En la presente tesina, se explicará Authorization Code, que es el método adoptado para la implementación de DAUT, con algunas variaciones y adaptaciones.

4.5.4 - Código de autorización (Authorization code):

Es un tipo de concesión de autorización que se utiliza para obtener tokens de acceso. Como se trata de un flujo basado en la redirección, el cliente debe ser capaz de interactuar con el agente de usuario del propietario del recurso (típicamente un navegador web) y capaz de recibir solicitudes (a través de la redirección) desde el servidor de autorización.

El código de autorización se obtiene usando un servidor de autorización como intermediario entre el cliente y el propietario del recurso. En lugar de hacer la petición de autorización directamente al propietario del recurso, el cliente redirige al propietario del recurso al servidor de autorización, que a su vez redirige al propietario del recurso de vuelta al cliente con el código de autorización.

Antes de dirigir al propietario del recurso de vuelta al cliente con el código de autorización, el servidor de autorización autentica al propietario del recurso y obtiene autorización. Las credenciales del propietario del recurso nunca se comparten con el cliente, dado que el propietario del recurso solo se autentica con el servidor de autorización.

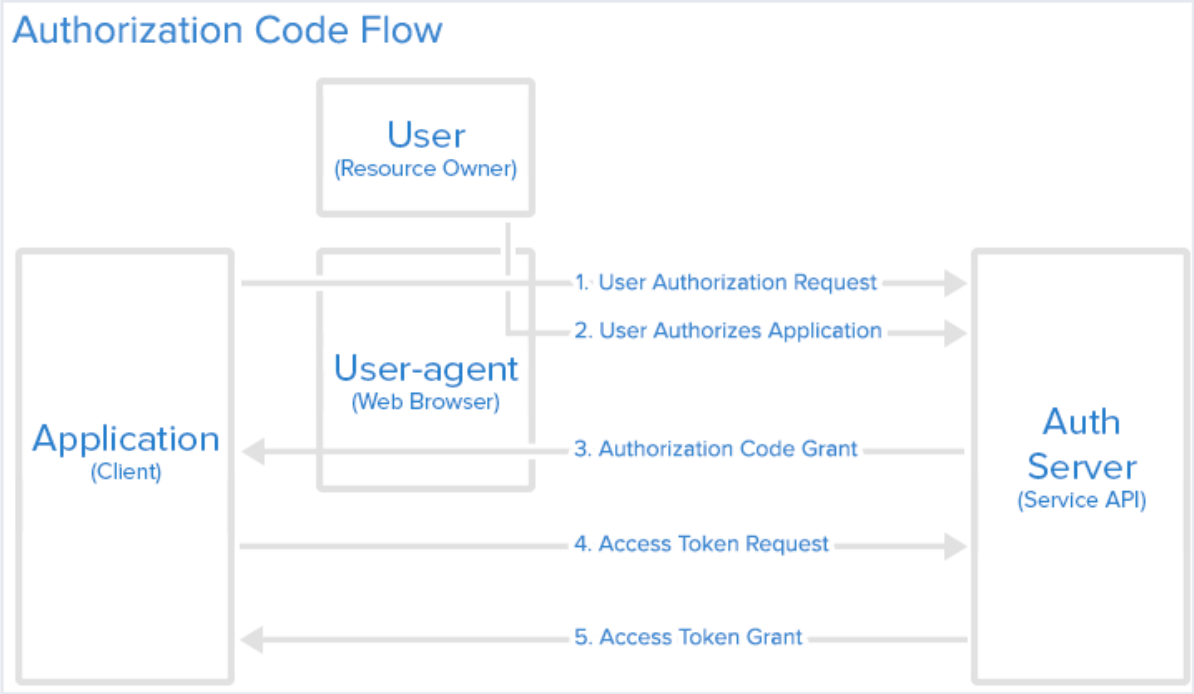

Figura 8 - Flujo Codigo de Autorizacion

1. El cliente inicia el flujo redireccionando el agente de usuario del propietario del recurso al punto de redirección (típicamente un formulario). El cliente introduce su identificador, su ámbito de acceso, estado y una URI de redirección a la que el servidor de autorización mandará al agente de usuario una vez el acceso finalice (satisfactoriamente o erróneamente).

2. El servidor de autorización autentica al propietario del recurso (a través del agente de usuario) y establece si el propietario del recurso concede o deniega la petición de acceso del cliente.

3. Asumiendo que el propietario del recurso concede el acceso, el servidor de autorización redirecciona el agente de usuario de vuelta al cliente usando la URI de redirección proporcionada anteriormente (en la petición o durante el registro del cliente). La URI de redirección incluye un código de autorización y cualquier estado local proporcionado por el cliente anteriormente.

4. El cliente pide un token de acceso al servidor de autorización incluyendo el código de autorización en la petición, obtenido en el paso anterior. Cuando hace la petición, el cliente se autentica con el servidor de autorización. El cliente incluye la URI de redirección usada para obtener el código de autorización, con propósitos de verificación. El servidor de autorización auténtica al cliente, valida el código de autorización y se asegura que la URI de redirección recibida coincide con la usada para redirigir al cliente en el paso 1.

5. Si todo es correcto, el servidor de autorización responde con un token de acceso y, opcionalmente, con un token de refresco. El código de autorización ofrece unos añadidos importantes de seguridad, tales como la capacidad de autenticar al cliente, o la transmisión del token de acceso directamente al cliente sin pasar por el agente de usuario del propietario del recurso y potencialmente exponerlo a otros, incluyendo al propietario del recurso

4.5.5 - Token de Acceso (Access token)

Los tokens de acceso son credenciales que se utilizan para acceder a recursos protegidos. Un token de acceso es una cadena de caracteres que representa una autorización emitida al cliente. Representan alcances y duraciones de acceso específicos, garantizadas por el propietario del recurso, y reforzadas por el servidor de recursos y el servidor de autorización.

El token de acceso proporciona una capa de abstracción, en sustitución de las diferentes formas de autorización (por ejemplo, nombre de usuario y contraseña) con un único token entendido por el servidor de recursos. Esta abstracción permite la emisión de tokens de acceso más restrictivos que la concesión de autorización utilizada para su obtención, así como la eliminación de la necesidad de que el servidor de recursos comprenda una amplia variedad de métodos de autenticación.

4.5.6 - Token de Refresco (Refresh token)

Los tokens de refresco son credenciales utilizadas para obtener tokens de acceso. Los mismos son emitidos al cliente por el servidor de autorización, y sirven para que una vez caducado el token de acceso, poder obtener un token de acceso nuevo.

A diferencia de los tokens de acceso, los tokens de refresco están provistos para ser usados sólo con servidores de autorización y nunca se envían a los servidores de recursos.

4.6 - Authorization Code en el ámbito Web de la DGSIAF

Análogamente a las entidades involucradas en el flujo de obtención del token de acceso OAuth 2.0, en el ámbito de las aplicaciones Web de la DGSIAF, se encuentran las siguientes entidades:

- Resource Owner (Propietario de los recursos): es el usuario final de e-Sidif, como se describió en puntos anteriores, las aplicaciones web son usadas por el mismo conjunto de usuarios del e-Sidif, y bajo las mismas políticas de seguridad.

- Client (Cliente/Aplicación): son las aplicaciones Web de la DGSIAF, entre las que se encuentran:

    - Menú -> Portal contenedor de aplicaciones
    - Siche -> Consulta de Sistemas Heredados
    - Sireco -> Registro de Cuentas Oficiales
    - Sicde -> Sistema de Captura de Datos Externos

- Authorization Server (Servidor de autorización): El servidor de autorización, es DAUT, este tiene el consentimiento del propietario del recurso y emite tokens de acceso.

- Resource Server (Servidor de Recursos): en este ámbito, se encarga de desencriptar el Token, y bloquear el acceso a los datos protegidos.
  Esta funcionalidad debe estar en todas las aplicaciones que usen a DAUT como proveedor de seguridad.

La siguiente imagen muestra las entidades involucradas. Este flujo se basa en redirecciones, esto significa que la aplicación cliente debe ser capaz de interactuar con el user-agent del usuario, enviando respuestas HTTP con código de estado 302,

y también debe ser capaz de aceptar request (via redirecciones) del servidor de autorización.

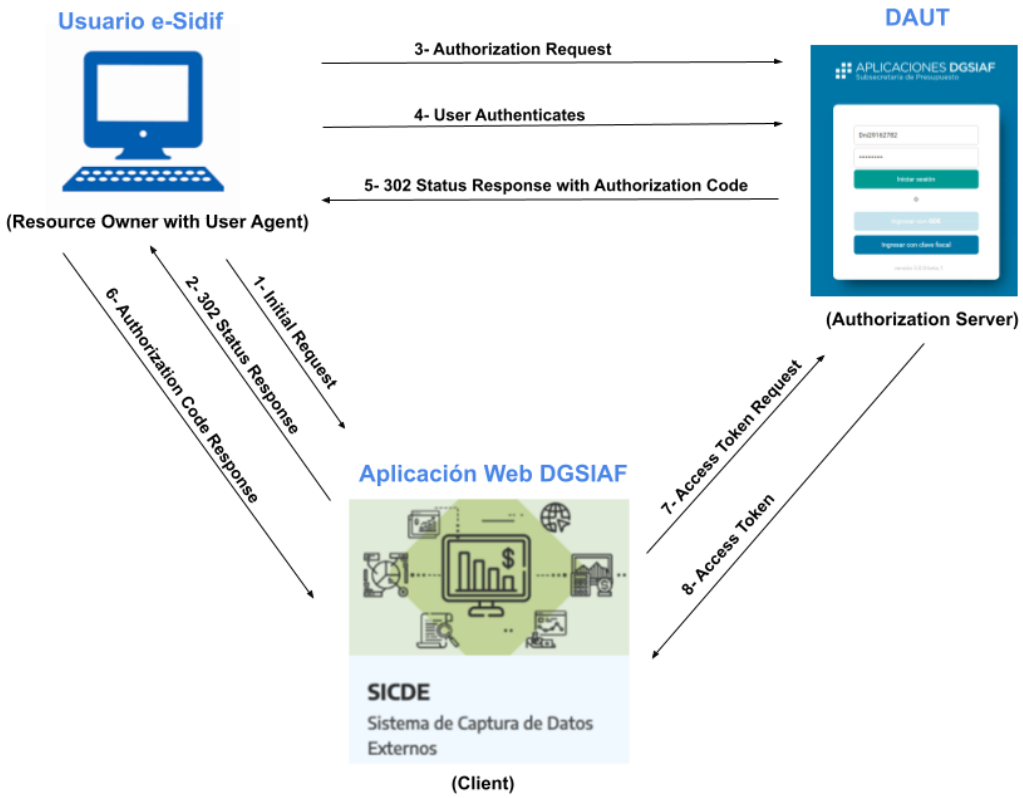

Figura 9 - Flujo Authorization Code en DGSIAF

1- Initial Request

El flujo comienza cuando un usuario de la DGSIAF desea ingresar en una aplicación Web, en el caso del gráfico anterior, la aplicación en cuestión se llama SICDE.

2- 302 Status Response

La aplicación SICDE responde con un código de estado 302, indicando una redirección hacia el servidor de autorización. Esta funcionalidad está encapsulada en la librería ***DAUT-CONNECTOR*** la cual se detalla en puntos posteriores.

3- Authorization Request

El servidor de autorización DAUT recibe el authorization request y responde con un página de login.

4- User Authenticate

El usuario se autentica con éxito y otorga consentimiento para que el cliente acceda a sus recursos (o un subconjunto de ellos).

5- 302 Status Response with Authorization Code

Recibida la autorización del usuario, el servidor de autorización DAUT responde con una redirección vía user-agent. Además anexa a la misma un código de autorización.

6- Authorization Code Response

La aplicación cliente recibe el código de autorización.

7- Access Token Request

Con el código recibido, la aplicación cliente solicita ahora un access token al servidor de autorización mediante una petición de tipo POST.

8- Access Token Response

El servidor de autorización verifica que todos los datos de la petición sean válidos y responde el access token.

4.6.1 - Access Token entregado por DAUT

Un Access Token es la llave que le permite al cliente (en el caso descrito en la anterior imagen, la aplicación web SICDE) acceder a recursos protegidos.

Para esta tarea, parte de la funcionalidad contenida en la librería DAUT-CONNECTOR es la encargada de desencriptar el token que llega de DAUT.

El token que DAUT arma es un JWT (Json Web Token) [13] a él que le agrega información que la aplicación cliente necesita para operar, como las aplicaciones a las que el usuario tiene permiso, dependiendo de sus capacidades.

En la siguiente imagen, se ilustran los datos que provee el token:


```
{
  "USER_ROL_DOMAIN": {
    "ROLE_USER_ADMIN": [
      "DGSIAF-DGSIAF"
    ],
    "ROLE_USER_CONSULTA": [
      "DGSIAF-DGSIAF-DGSIAF"
    ]
  },
  "provider": "daut",
  "user_name": "ar.gob.mecon.dgsiaf.daut.model.User@5b68a3ac[login=DNI29162782,cuit=20291627820]",
  "scope": [
    "read"
  ],
  "ati": "8c849482-c0b6-4d72-9358-81d5dbcd9681",
  "exp": 1660066979,
  "APP_LIST": "[{\"name\":\"SICDE\",\"url\":\"/sicde/\",\"descripcion\":\"Sistema de Captura de Datos Externos\",\"AUTHORIZED\":1},
  "USER": "{\"LOGIN\":\"DNI29162782\",\"NAME\":\"Vacas Martin\",\"ID\":99420849,\"USER_TYPE\":\"DGSIAF\"}",
  "jti": "309fddaf-f90a-4d52-be84-2137edf34ce0",
  "authorities": [
    "ROLE_USER_ADMIN",
    "ROLE_USER_CONSULTA"
  ],
  "client_id": "SICDE"
}
```


Figura 10- Token JWT obtenido de DAUT

4.7 - Single Sign On en el ámbito Web de la DGSIAF

Para brindar un contexto de SSO en las aplicaciones Web de la DGSIAF, se debe proveer a todas las aplicaciones web, un único punto de entrada para la autenticación de los usuarios. También se debe implementar un mecanismo que permita a los usuario proveer sus credenciales una sola vez y poder acceder a múltiples aplicaciones a las cuales tiene permitido el acceso.

Todas las aplicaciones Web que hagan uso de la librería DAUT-CONNECTOR, tendrán el mismo punto de acceso a la autenticación y los request para autenticarse serán redirigidos al servidor DAUT.

La siguiente imagen ilustra un flujo de SSO, en el que un usuario ya inició sesión en DAUT para usar la aplicación SICDE, y luego desea usar otra aplicación web, en este ejemplo se trata de la aplicación llamada SIRECO.

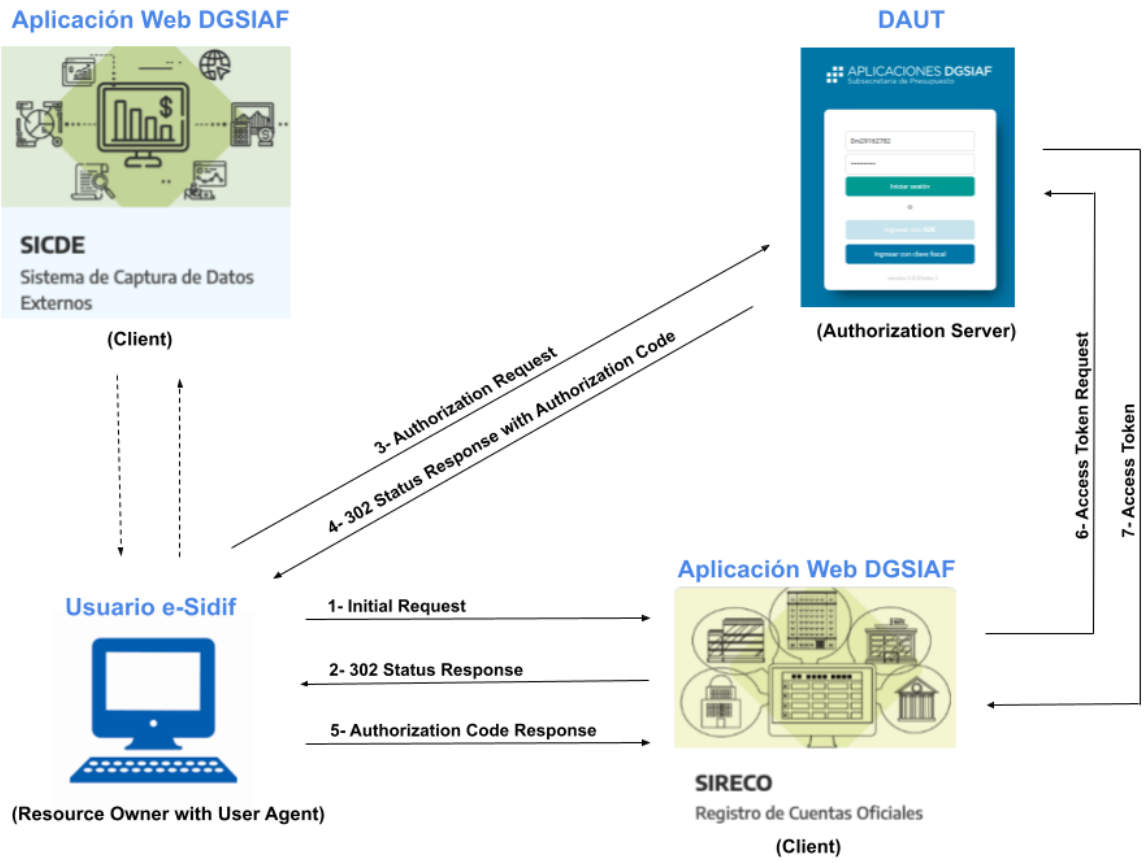

Figura 11 - SSO

Asumiendo que el usuario ya se autenticó para usar la aplicación SICDE, se ve que los pasos para iniciar sesión en SIRECO son exactamente los mismos que se describieron en el punto 4.6, sólo difiere en el punto 3.

En este flujo, el servidor de autorización DAUT, cuando recibe un request de autorización, se da cuenta que el usuario que inicio el flujo, ya tiene una sesión iniciada (a través de una cookie que pasa el user-agent) y entonces responde directamente con un código de autorización en vez de redirigir al usuario a la página de login.

Con el código de autorización obtenido de DAUT, los pasos para obtener un token de acceso son los mismos que el flujo descrito anteriormente.

CAPÍTULO 5 - Componentes de Spring Security

5.1 - Introducción

Tanto en el proyecto **DAUT** como en ***daut-connector***, se usa el soporte de OAuth2 que brinda el framework de Spring Security [11] para implementar el flujo de "*Authorization Code*" explicado en puntos anteriores.

Spring Security es un framework que provee autenticación, autorización y brinda soporte para securizar aplicaciones basadas en Spring.

A continuación se analizará la arquitectura de alto nivel de Spring Security dentro de las aplicaciones basadas en Servlet, y las componentes que la conforman.

Spring Security se integra al ServletContainer mediante el uso de una cadena de filtros, esta es conocida como "*FilterChain*", por lo que es útil observar primero el papel de los filtros en general.

5.2 - Filtros

Un filtro es un objeto que se utiliza para interceptar las solicitudes y respuestas HTTP de una aplicación [14]. Al usar filtros, es posible realizar distintas operaciones en dos instancias:

- Antes de enviar la solicitud al controlador.
- Antes de enviar una respuesta al cliente.

Los filtros son creados, manejados y destruidos por el ServletContainer.
La siguiente imagen muestra los filtros por los que pasa un request HTTP, hasta llegar a destino.

El cliente hace un request HTTP a la aplicación, el contenedor crea la cadena de filtros y servlet que atenderá dicho request, basándose en la URI indicada en el request.

Este request pasa a la cadena de filtros (*FilterChain)* donde cada uno de los filtros procesa el request (también son capaces de procesar la respuesta) y pasa al filtro que le sigue en la cadena y eventualmente el request llegará al servlet destino.

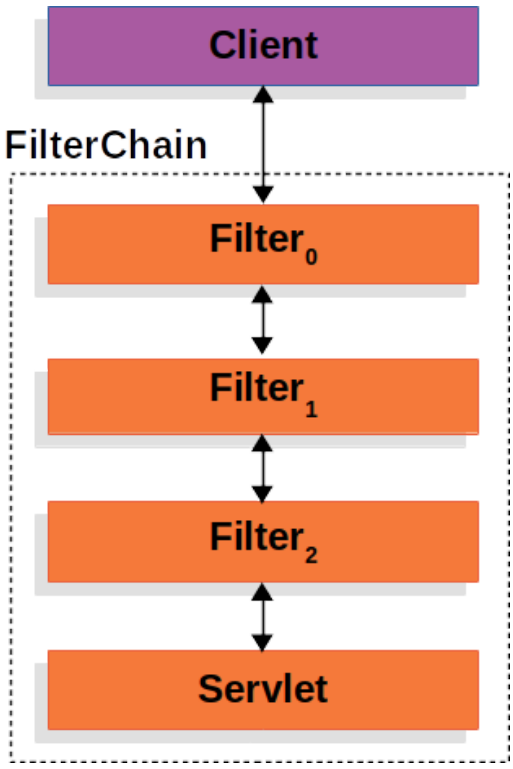

Figura 12 - Filtros y Cadena de Filtros

Dado que un filtro solo afectará a los filtros posteriores y al servlet, el orden en que estos se ejecutan es extremadamente importante.

5.3 - Manejo de Filtros

Spring proporciona una implementación de filtro denominada DelegatingFilterProxy, dicha implementación permite establecer un puente entre el ciclo de vida del contenedor de Servlet y el ApplicationContext de Spring [15].

El soporte de Servlet de Spring Security está contenido en FilterChainProxy. Este es un filtro especial proporcionado por Spring Security que permite delegar a muchas instancias de filtro a través de SecurityFilterChain.

FilterChainProxy utiliza SecurityFilterChain para determinar qué Filtro de seguridad provisto por Spring deben invocarse para el request entrante.

La siguiente imagen ilustra como este filtro especial (FilterChainProxy) delega el request entrante a los filtros de seguridad de Spring, los cuales suelen ser beans que están en el ApplicationContex de la aplicación.

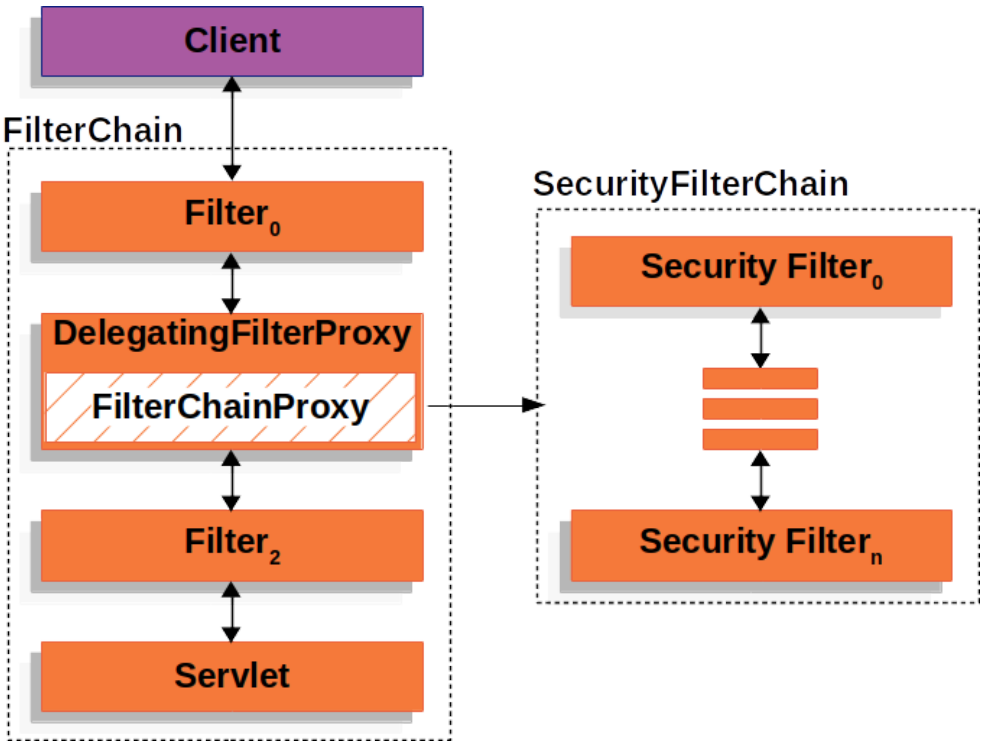

Figura 13 - Filter Chain Proxy

Como se dijo anteriormente, los filtros de seguridad en SecurityFilterChain suelen ser Beans, y están registrados con FilterChainProxy en lugar de DelegatingFilterProxy, este proporciona más flexibilidad para determinar cuándo se debe invocar una filtro de seguridad perteneciente a la cadena de filtros *SecurityFilterChain*.

En un contenedor de servlet, los filtros se invocan basándose únicamente en la URL. Sin embargo, FilterChainProxy puede determinar la invocación en función de cualquier cosa que venga en el Request aprovechando la interfaz RequestMatcher.

De hecho, FilterChainProxy se puede usar para determinar qué SecurityFilterChain se debe usar. Esto permite proporcionar una configuración totalmente separada para diferentes segmentos de su aplicación.

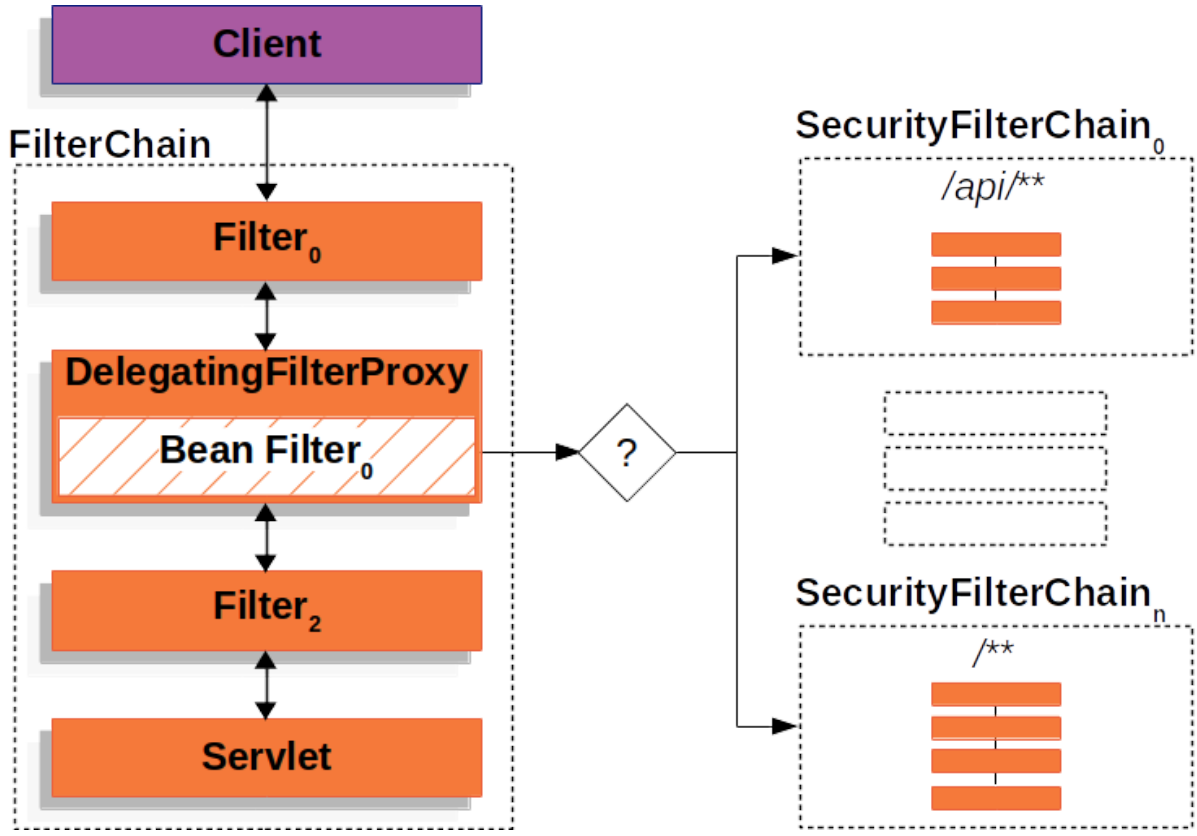

Figura 14 - Filtros de Seguridad de Spring

FilterChainProxy decide qué SecurityFilterChain debe usarse. Si se solicita una URL del tipo: ".....api/messages/", coincidirá con el patrón de /api/** de *SecurityFilterChain 0*, por lo que solo se invocará *SecurityFilterChain 0*.

Si se solicita una URL de ".....messages/", no coincidirá con el patrón de /api/** de *SecurityFilterChain 0*, por lo que FilterChainProxy continuará probando cada SecurityFilterChain.

5.3.1 - Filtros de Seguridad

A continuación se detalla una lista con algunos de los filtros de seguridad más relevantes usados por Spring Security:

- SecurityContextPersistenceFilter
- HeaderWriterFilter
- CorsFilter
- LogoutFilter
- OAuth2AuthorizationRequestRedirectFilter
- UsernamePasswordAuthenticationFilter
- AnonymousAuthenticationFilter
- OAuth2AuthorizationCodeGrantFilter
- SessionManagementFilter
- ExceptionTranslationFilter
- SwitchUserFilter

SecurityContextPersistenceFilter es el filtro de entrada al mecanismo de autenticación en Spring Security. Básicamente, administra el ciclo de vida de SecurityContext en un request, obteniendo el mismo de la actual HttpSession y asociándolo con el proceso de ejecución actual en el SecurityContextHolder.

HeaderWriterFilter es responsable de escribir algunos headers de seguridad en la respuesta actual para habilitar la protección del navegador.

CsrfFilter su función es evitar los ataques de falsificación de solicitudes entre sitios. Este filtro se ejecuta para cualquier solicitud que permita cambiar de estado.

LogoutFilter se ejecuta si el request entrante coincide con el RequestMatcher de este filtro, que está predeterminado en /logout. Este filtro tiene algunos controladores predeterminados que se ejecutarán durante el cierre de sesión y también puede escribir sus propios controladores de cierre de sesión personalizados para casos de uso específicos.

OAuth2AuthorizationRequestRedirectFilter este filtro inicia la concesión del código de autorización al redirigir el agente de usuario al endpoint de autorización del servidor de autorización. De forma predeterminada, este filtro responde a las solicitudes de autorización en el URI /oauth2/authorization/ mediante el ***OAuth2AuthorizationRequestResolver*** predeterminado.

UsernamePasswordAuthenticationFilter el RequestMatcher de este filtro está predeterminado en /login. Y si se realiza una solicitud de inicio de sesión, este filtro lee el nombre de usuario y la contraseña del cuerpo de la solicitud; si existe, autentica al usuario.

AnonymousAuthenticationFilter, SpringSecurity crea un objeto de autenticación (Authentication) para las solicitudes autorizadas. Por ejemplo, si inicia sesión a través de un formulario con formLogin o autenticación básica, existe un **UsernamePasswordAuthenticationToken** en el contexto SecurityContext como objeto de autenticación. Sin embargo, un usuario no autorizado puede acceder a recursos sin autenticación. Esta vez no existe ningún objeto de autenticación en SecurityContext. Un usuario no autorizado puede acceder a recursos sin privilegios de una de las siguientes dos maneras:

1. llamando a HttpSecurity.anonymous() y otorgando el rol ANÓNIMO a los recursos relacionados.

2. llamando a permitAll() y dando acceso a todos los roles.

OAuth2AuthorizationCodeGrantFilter es un filtro para la concesión del código de autorización de OAuth 2.0, gestiona el procesamiento de la respuesta de autorización de OAuth 2.0.

SessionManagmentFilter detecta que un usuario se ha autenticado desde el inicio de la solicitud, si lo ha hecho, llama a la **SessionAuthenticationStrategy** configurada para realizar cualquier actividad relacionada con la sesión, como activar mecanismos de protección de fijación de sesión o comprobar si hay varios inicios de sesión simultáneos.

ExceptionTranslationFilter maneja cualquier excepción de tipo **AccessDeniedException** y **AuthenticationException** lanzada dentro de la cadena de filtros.

Este filtro es necesario porque proporciona el puente entre las excepciones de Java y las respuestas HTTP.
Si se detecta una **AuthenticationException**, el filtro iniciará el proceso de autenticación.

5.4 - Componentes principales

A continuación se describirán algunos de los componentes principales de Spring Security, entre ellos el SecurityContextHolder, SecurityContext y Authentication, que tienen un importante rol en el manejo de la seguridad.

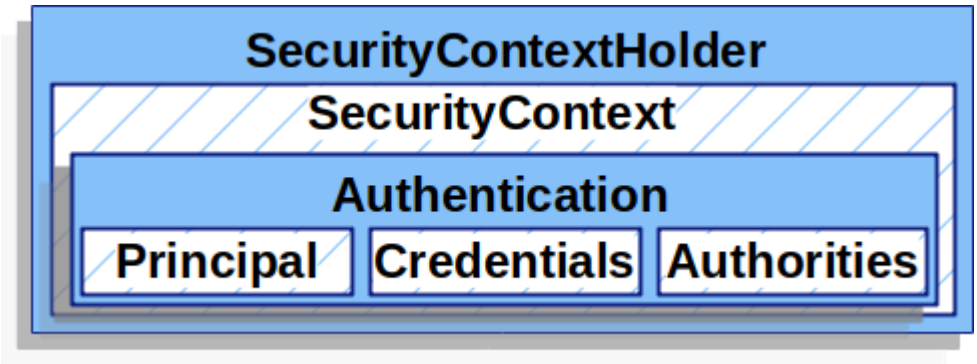

Figura 15 - Componentes Core de Spring Security

En el corazón del modelo de autenticación de Spring Security se encuentra el ***SecurityContextHolder***, dicho componente contiene a el ***SecurityContext***, que es donde se almacenan los detalles del contexto de seguridad de la aplicación, el cual incluye detalles del usuario principal.

De forma predeterminada, SecurityContextHolder usa un ThreadLocal para almacenar esta información, lo que significa que el contexto de seguridad siempre está disponible para los métodos en el mismo hilo de ejecución, incluso si el contexto de seguridad no se pasa explícitamente como argumento a esos métodos.

Dentro de SecurityContextHolder se almacena información del usuario principal que interactúa actualmente con la aplicación. Spring Security utiliza un objeto de autenticación (***Authentication***) para representar esta información.

Normalmente no es necesario crear un objeto de autenticación, en general se recupera del contexto de seguridad, para obtener ciertos datos del usuario autenticado se procede como se indica en la siguiente figura:

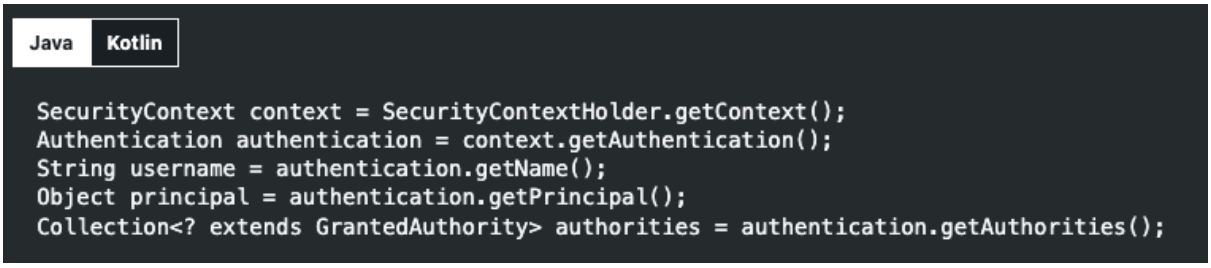

Figura 16 - Recuperando un usuario del Contexto de la aplicación

El objeto Authentication tiene dos propósitos principales dentro de Spring Security:

1. Sirve para proveer las credenciales del usuario al AuthenticationManager.

2. Representa al usuario autenticado actualmente. La autenticación actual se puede obtener de SecurityContext como se describe en la imagen anterior.

El objeto Authentication a su vez está compuesto de otros componentes:

- ***principal*** - identifica al usuario. Cuando se autentica con un nombre de usuario/contraseña, a menudo se trata de una instancia de ***UserDetails***.

- ***credentials*** - a menudo una contraseña. En muchos casos, esto se borrará después de que el usuario se autentique para garantizar que no se filtre.

- ***authorities***: son permisos de alto nivel que se otorgan al usuario.

Los permisos se pueden obtener con el método Authentication.getAuthorities(). Este método proporciona una colección de objetos ***GrantedAuthority***.

Un objeto GrantedAuthority representa una autoridad que se otorga al usuario principal. Estas autoridades suelen ser "roles", como por ejemplo: ***ROLE_ADMINISTRATOR*** o ***ROLE_HR_SUPERVISOR***. Estos roles se configuran posteriormente para la autorización web, la autorización de métodos y la autorización de objetos de dominio. Otras componentes dentro de Spring Security son capaces de interpretar estas autoridades y actuar en consecuencia.

Cuando se utiliza la autenticación basada en nombre de usuario/contraseña, las autorizaciones otorgadas generalmente se cargan mediante ***UserDetailsService***.

Por lo general, los objetos GrantedAuthority son permisos para toda la aplicación. No son específicos de un objeto de dominio dado.

Otro componente clave es **AuthenticationManager**, este componente es el encargado de definir cómo los filtros de Spring Security realizará la autenticación. El objeto Authentication que es retornado, luego es almacenado en SecurityContextHolder por el controlador que invocó al AuthenticationManager.

ProviderManager es la implementación más utilizada de AuthenticationManager, este delega a una lista de proveedores de autenticación. Cada AuthenticationProvider tiene la oportunidad de indicar que la autenticación es exitosa, fallar o indicar que no puede tomar una decisión y por lo tanto permitir que un AuthenticationProvider mas abajo en la cadena decida.

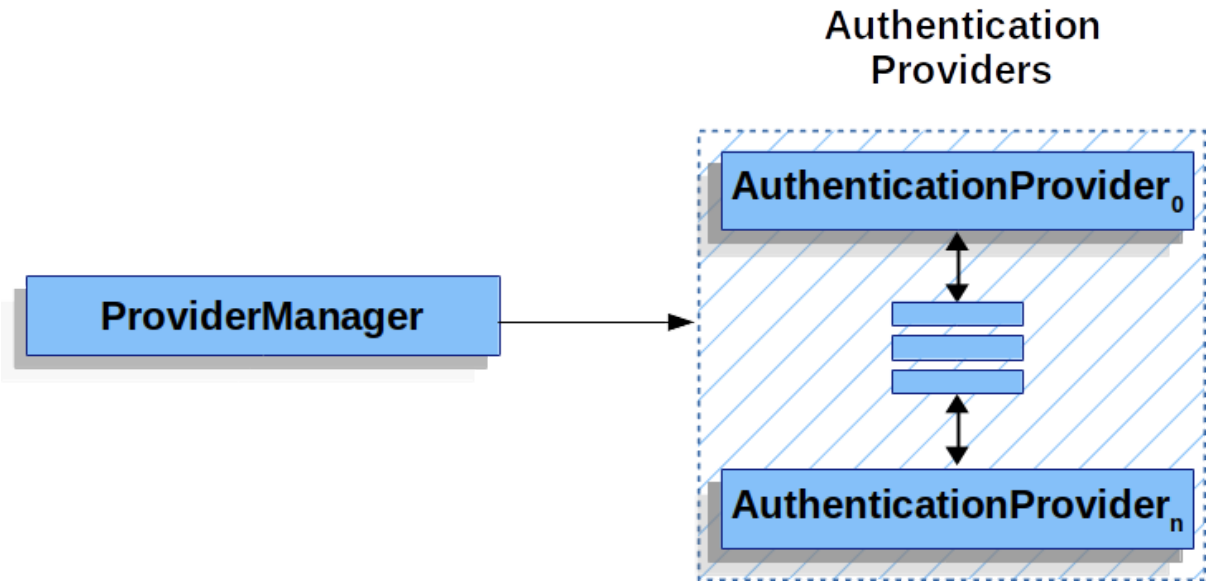

Figura 17 - Proveedores de Autenticación

Se pueden inyectar varios AuthenticationProvider, cada uno realiza un tipo específico de autenticación. Por ejemplo, **DaoAuthenticationProvider** admite la autenticación basada en nombre de usuario y contraseña, mientras que **JwtAuthenticationProvider** admite la autenticación de un token JWT.

5.5 - Soporte para OAuth 2.0

El soporte de OAuth brindado por Spring Security [16], se basa en el uso de filtros y algunos de los componentes descritos anteriormente.

A continuación se describen algunos de los Componentes principales involucrados en proveer soporte OAuth a través de Spring.

5.5.1 - Componentes Arquitectónicos para la Autenticación

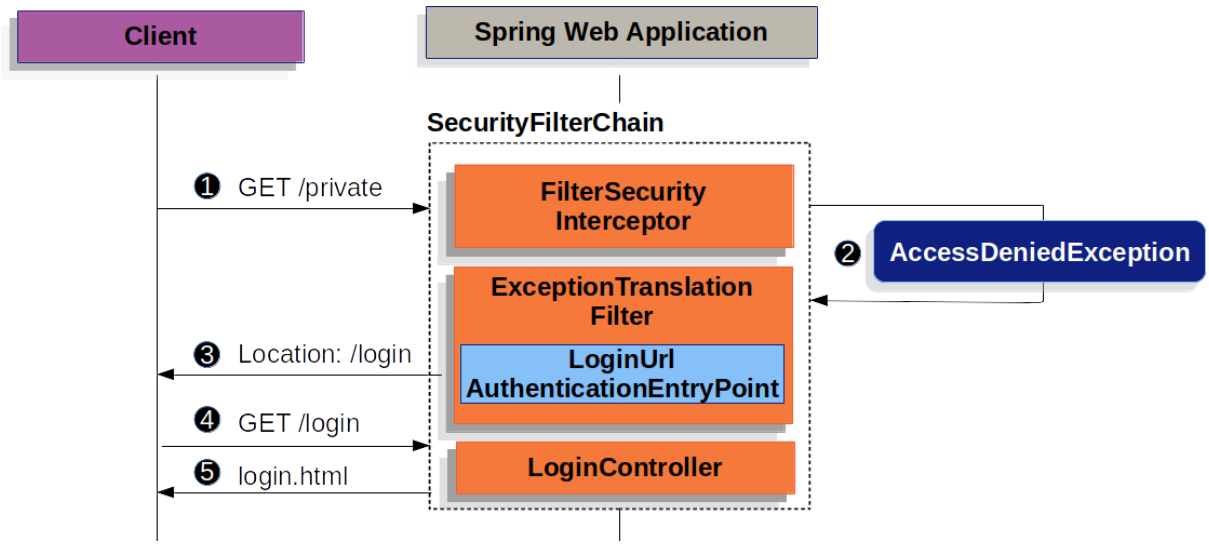

Figura 18 - Flujo de redirección

1. Un usuario que no está autenticado, hace un request a un recurso privado

2. El filtro de seguridad de Spring FilterSecurityInterceptor detecta que el request viene sin autenticación, y lanza una exception AccessDeniedException

3. El filtro ExceptionTranslationFilter inicia la Autenticación, y redirecciona a una página de login, configurada por el AuthenticationEntryPoint

4. El browser lleva a cabo la redirección

Luego que el usuario ingrese su nombre de usuario y password, entra en juego otro filtro de seguridad UsernamePasswordAuthenticationFilter.

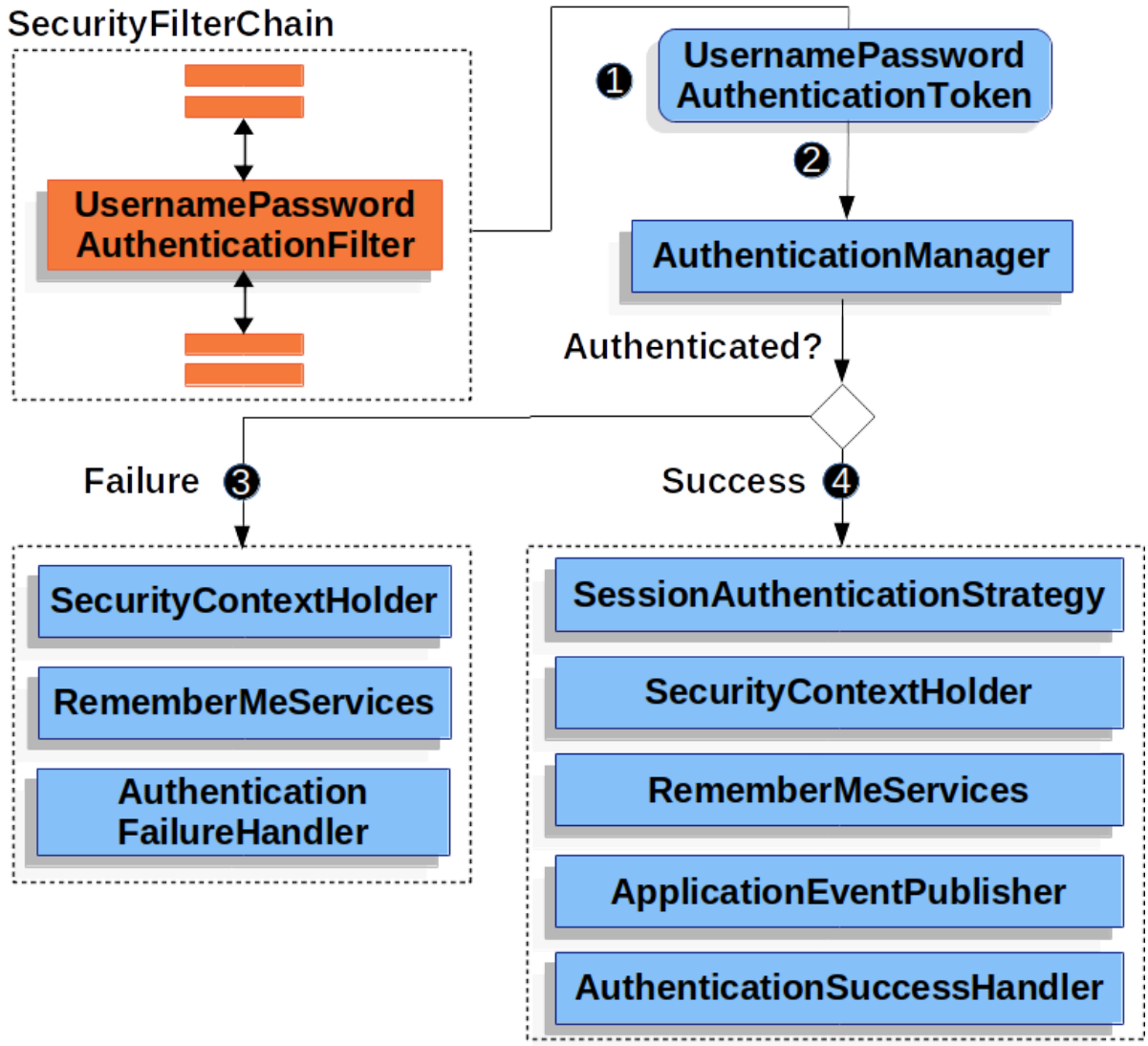

Figura 19 - Authentication Username y Password

1. Cuando el usuario ingresa su nombre de usuario y password, el filtro crea un UsernamePasswordAuthenticationToken que es de tipo Authentication (explicado en anteriores puntos) extrayendo los datos del usuario del HttpServletRequest

2. El objeto Authentication creado en el punto anterior, es pasado al AuthenticationManager, el cual se encarga de verificar la autenticación

3. Falló la Autenticación

4. Autenticación Exitosa

5.5.2 - Componentes Arquitectónicos para la Autorización

La siguiente imagen describe los componentes que integran el flujo de autorización de un request http.

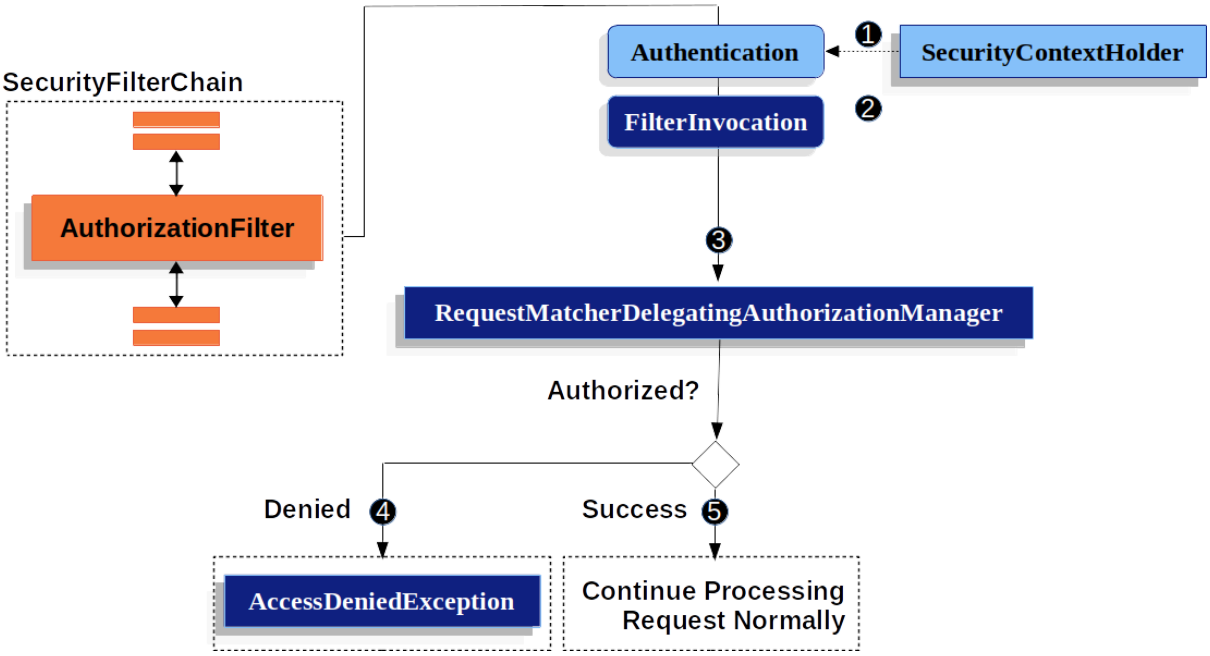

Figura 20 - Autorización HttpServletRequest

1. El AuthorizationFilter obtiene el objeto Authentication del contexto SecurityContextHolder

2. Se crea un FilterInvocation

3. Se pasan tanto el Authentication como el FilterInvocation al AuthorizationManager

4. Si no está autorizado, se lanza una AccessDeniedException, que va a ser manejada por el filtro ExceptionTranslationFilter

5. Si se garantiza el acceso, continúa el proceso normalmente

CAPÍTULO 6 - Implementación del sistema de seguridad centralizado

6.1 - Introducción

En este capítulo se describen detalles de implementación y configuración de las principales componentes que conforman este sistema de seguridad centralizado adoptado en el ámbito Web de la DGSIAF.

Como se describió en el capítulo anterior, tanto DAUT como daut-connector, fueron implementados adoptando componentes que provee el framework de Spring Security.

6.2 - Configuración

Para disponer de la funcionalidad que brinda Spring Security, para la implementación del flujo de OAuth 2.0, se deben configurar las dependencias correspondientes. En la siguiente imagen se muestra cómo incluir las dependencias usando Gradle, que es el mecanismo de construcción utilizado.

Dependencias Gradle configuradas en ambos proyectos:

```
apply from: 'versions.gradle'

dependencies {

    implementation group:'org.springframework.security.oauth', name:'spring-security-oauth2'
    implementation group:'org.springframework.security', name:'spring-security-jwt'
    implementation group:'org.springframework.security', name:'spring-security-core'
    implementation group:'org.springframework.security', name:'spring-security-config'
    implementation group:'org.springframework.security', name:'spring-security-taglibs'
    implementation group:'org.springframework.security', name:'spring-security-web'
    implementation group:'commons-codec', name:'commons-codec'

}
```

Figura 21 - Dependencias Gradle

Agregando las dependencias indicadas en la figura anterior, ya se tendrá en el contexto de la aplicación una cadena de filtros, los cuales son los encargados de

filtrar los requerimientos y delegar en los componentes de seguridad indicados, tanto en la Autenticación como en la Autorización.

Al iniciar la aplicación, se va a registrar el **DispatcherServlet** de Spring Framework que es el que hace el rol de Front Controller y el **DelegatingFilterProxy** que es el filtro que habilita Spring Security en nuestra aplicación.

Una vez registrados, el filtro protege el acceso al DispatcherServlet y por consecuencia al resto de la aplicación.

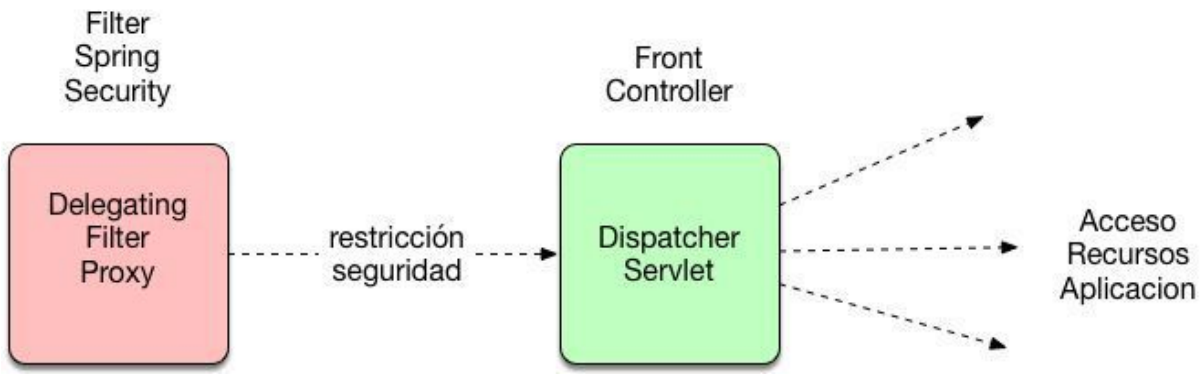

Figura 22 - Spring Security en acción

6.3 - Roles OAuth 2.0

La siguiente imagen describe los roles involucrados

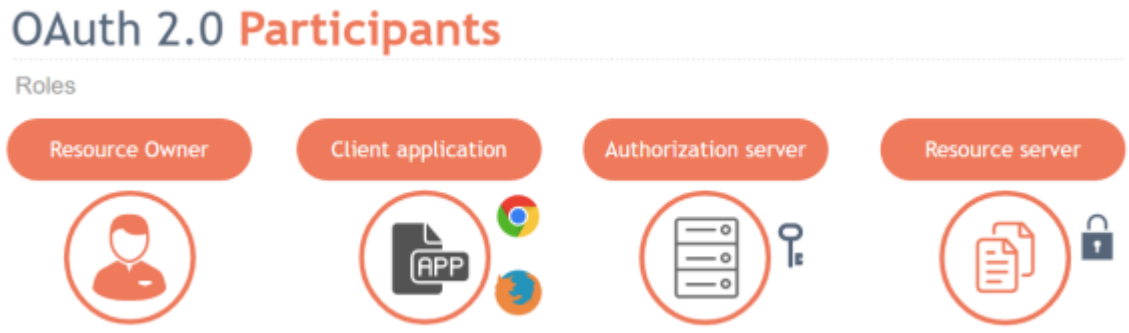

Figura 23 - Roles OAuth 2.0

En el contexto de la DGSIAF cada uno de estos roles es representado por una o varias entidades descritas a continuación:

- ***Resource Owner***: representado por una entidad que puede dar consentimiento para el acceso en su nombre, a un recurso protegido. En este contexto es el Usuario e-Sidif, con sus roles y capacidades, el cual al autenticarse en DAUT da su consentimiento y podrá acceder a los recursos determinados por sus capacidades.

- ***Client Application:*** son las aplicaciones Web de la DGSIAF, las cuales requieren de un token de acceso, para operar sobre recursos protegidos. Para esto la librería DAUT-CONNECTOR incluida en la aplicación cliente, redirecciona a DAUT para la obtención del token de acceso.

- ***Authorization Server:*** en este caso es DAUT, que actúa como servidor de autorización, entregando un token de seguridad. Dicho token tiene información agregada

- ***Resource Server:*** en este contexto, el resource server está representado por las aplicaciones cliente, este es responsable de desencriptar el token de acceso que le llega y permitir (o prohibir) el acceso a recursos protegidos.

El soporte de OAuth 2.0 que provee el Spring Security, se basa en la configuración por medio de anotaciones específicas, a continuación se detalla parte de la configuración de cada entidad involucrada en el flujo de obtención del token.

6.3.1 - Authorization Server.

Para habilitar un servidor de autorización es necesario anotar una clase con la siguiente anotación @EnableAuthorizationServer, lo que habilita un *AuthorizationEndpoint* y un *TokenEndpoint* en el contexto de la aplicación.

```java
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter



    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private AuthorizationCodeServices authorizationCodeServices;

    @Autowired
    private DautUserApprovalHandler userAprovalhandler;

    @Autowired
    @Qualifier("dautTokenEnhacer")
    private TokenEnhancer dautTokenEnhancer;

    @Autowired
    private JwtTokenStore tokenStore;

    @Autowired
    private JwtAccessTokenConverter tokenEnhancer;
```

Figura 24 - @EnableAuthorizationServer

La clase AuthorizationServerConfigurerAdapter, es la manera que provee el framework para realizar una configuración adaptable a lo que el negocio requiera. Para esto se deben sobreescribir los métodos configure.

```java
public class AuthorizationServerConfigurerAdapter implements AuthorizationServerConfigurer {

    @Override
    public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    }

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    }

}
```

Figura 25 - AuthorizationServerConfigurerAdapter

La siguiente imagen muestra la configuración realizada en DAUT, con la cual se exponen dos endpoints:

- /oauth/authorize
- /oauth/token

```java
@Override
public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
    oauthServer
            .tokenKeyAccess("isAnonymous() || hasRole('ROLE_TRUSTED_CLIENT')")
            .checkTokenAccess("hasRole('TRUSTED_CLIENT')");
}
```

```java
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints
            .authenticationManager(authenticationManager)
            .tokenStore(tokenStore()).tokenEnhancer(tokenEnhancerChain())
            .authorizationCodeServices(authorizationCodeServices)
            .userApprovalHandler(userAprovalhandler);
}
```

Figura 26 - Configuración Authorization Server

A continuación se describe cómo está compuesto el request que llega al endpoint ***/oauth/authorize***:

http://apps.dgsiaf.mecon.gov.ar/daut/oauth/authorize?
client_id=menuID&
redirect_uri=http://apps.dgsiaf.mecon.gov.ar/menu/secure/login&
response_type=code&
scope=read&
state=Mst758'

El campo client_id representa la aplicación cliente desde la cual se está queriendo obtener un código de acceso que luego será intercambiado por un token.

Los campos restantes sirven para responder dicha solicitud con un access code, indicado en el campo response_type, se indican también un scope y un código de estado.

El endpoint ***/oauth/token*** es expuesto para la generación del token, con el código de acceso obtenido se hace un POST a DAUT para obtener el token de acceso.

HTTP POST http://apps.dgsiaf.mecon.gov.ar/daut/oauth/token
{grant_type=[authorization_code],
code=[Hz3RdN],
redirect_uri=[http://apps.dgsiaf.mecon.gov.ar/menu/secure/login]}

Cuando llega un request al endpoint ***/oauth/authorize***, se chequea si el usuario ya está autenticado, es decir si ya existe una sesión activa, para esto en dicho request el user-agent manda una cookie.

En caso que no esté autenticado, este será redireccionado a un formulario de login, a continuación se describe la configuración para dicha funcionalidad.

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public static class DautSecurityConfiguration extends WebSecurityConfigurerAdapter {

  @Autowired
  private UserAuthenticationProvider authenticationProvider;

  @Autowired
  private LoginFailureHandler loginFailureHandler;

  @Autowired
  private LogoutSuccessHandler logoutSuccessHandler;

  @Autowired
  private ChangePasswordFilter changePasswordFilter;

  @Autowired
  private DautProperties dautProperties;

  @Autowired
  public void globalUserDetails(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(authenticationProvider);
  }
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  String monitoringAccessCondition = dautProperties.getMonitoringAccessCondition();
  http.authorizeRequests().antMatchers("/actuator/**").access(monitoringAccessCondition)
      .antMatchers("/metrics**").access(monitoringAccessCondition)
      .anyRequest().authenticated().and()
      .exceptionHandling().and()
      .csrf();

  String key = UUID.randomUUID().toString();
  http.anonymous().key(key).authenticationFilter(new DautAnonymousAuthenticationFilter(key));

  FormLoginConfigurer<HttpSecurity> formLoginConfigurer = new FormLoginConfigurer<HttpSecurity>();
  formLoginConfigurer.setBuilder(http);
  formLoginConfigurer.loginPage("/login").permitAll();
  formLoginConfigurer.successHandler(loginSuccessHandler());
  formLoginConfigurer.failureHandler(loginFailureHandler);
  http.apply(formLoginConfigurer);
}
```

Figura 27 - Configuración Autenticación y FormLogin

Con esta configuración, se determinan las URLs que son accesibles sin estar autenticado y cuales requiere que el usuario se haya autenticado correctamente.

También en esta configuración se determina el form login, al cual se permite acceder todos los request. Se determinan los handlers para un login satisfactorio y uno fallido.

6.3.2 - Client Application.

La entidad Client Application, en el presente contexto, son las aplicaciones Web de la DGSIAF, las cuales requieren de un token de acceso, para operar sobre recursos protegidos. Dichas aplicaciones deben ser capaces de realizar las siguientes tareas:

- Redirigir a DAUT cuando un usuario no logueado quiere acceder a un recurso protegido de una aplicación.
- Pedir un access token a DAUT y verificar su firma.
- Extraer los datos de usuario y claims (privilegios) que vienen en el access token y cargar un usuario por defecto en el contexto de seguridad de spring.

Para estas funcionalidades se desarrolló la librería ***DAUT-CONNECTOR***, esta se integra al contexto de la aplicación web que la importa como dependencia, gracias a que implementa la interfaz WebApplicationInitializer, la cual permite una configuración programática del ServletContext.

En esta se hace uso de la anotación @EnableOAuth2Client, provista por Spring Security para implementar un cliente OAuth 2.0.

```java
@Configuration
@EnableWebSecurity
@EnableOAuth2Client
public class OAuth2SsoDefaultConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    BeanFactory beanFactory;

    private List<DautConnectorConfigurer> configurers = Collections.emptyList();

    @Autowired
    public OAuthProperties authProperties;

    /**
     * @param configurers
     *            the configurers to set
     */
    @Autowired(required = false)
    public void setConfigurers(List<DautConnectorConfigurer> configurers) {
        this.configurers = configurers;
    }
```

Figura 28 - @EnableOAuth2Client

Se sobreescriben los métodos configure() para adaptar el comportamiento , la siguiente imagen muestra parte de la configuración para indicar el servicio encargado de la decodificación del Token y de guardar un usuario en el contexto de la aplicación.

```java
@Override
protected void configure(HttpSecurity http) throws Exception {

    http.apply(new OAuth2ClientAuthenticationConfigurer(oauth2SsoFilter()));
    http.csrf().disable();
    http.logout().logoutSuccessUrl("/");
    // Captura la excepcion que lanza el filtro que se agrega debajo
    http.addFilterBefore(new DautExceptionTranslationFilter(), ExceptionTranslationFilter.class);
    http.addFilterBefore(new DautSwitchUserFilter(), FilterSecurityInterceptor.class);//
    http.antMatcher("/**").authorizeRequests().anyRequest().authenticated();

}

private OAuth2ClientAuthenticationProcessingFilter oauth2SsoFilter() {
    OAuth2RestOperations restTemplate = this.beanFactory.getBean(OAuth2RestOperations.class);
    ResourceServerTokenServices tokenServices = this.beanFactory.getBean(ResourceServerTokenServices.class);
    OAuth2ClientAuthenticationProcessingFilter filter = new OAuth2ClientAuthenticationProcessingFilter(
            authProperties.getLoginPath());

    filter.setRestTemplate(restTemplate);
    filter.setTokenServices(tokenServices);
    return filter;
}
```

Figura 29 - Configuración OAuth 2.0 Client

6.4 - Configuración de *daut-connector*

Como se describió en puntos anteriores, la librería *daut-connector* sirve de nexo entre las aplicaciones cliente y el servidor de autorizaciones DAUT. Esta librería integra al contexto de la aplicación donde se la incluya, unos filtros que redirigen los request que sean necesarios hacia el servidor de autorización.

Para hacer uso de esta, se deben seguir los siguientes pasos:

1. Depender de la última versión de daut-connector.
2. Anotar una clase con @Configuration y @DautSecurity.
3. Registrar la clase anteriormente anotada en el root application context.
4. Crear el archivo oauth.properties y ponerlo en la carpeta resources. Este archivo tiene que contener los siguientes parámetros:
    a. config.oauth2.clientID= nombre_de_la_app
    b. config.oauth2.clientSecret= secret_de_la_app
    c. config.oauth2.scope= read
5. Registrar la aplicación como cliente del DAUT.

Las siguientes imágenes muestran la configuración descrita anteriormente, realizada en la aplicación MENU (descrita en anteriores capítulos).

Primero se debe anotar una clase con @DautSecurity

```
@Configuration
@DautSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MenuDautConfig implements DautConnectorConfigurer {

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().antMatchers("/actuator/**").permitAll();
    }
```

Figura 30 - @DautSecurity

Luego se deben configurar los clientID y Secret, estos representan a la aplicación cliente en DAUT.

```
config.oauth2.clientID= menuPAF
config.oauth2.clientSecret= PasswordDelMenuPAF
config.oauth2.scope= read
```

Figura 31 - Propiedades de configuración

Con estos pasos realizados, cualquier request que llegue a la aplicación MENU y que requiere una autenticación, este es atrapado por un filtro que redirecciona a DAUT y hace la petición del token de autorización.

Luego del proceso de autenticación, en el contexto de la aplicación MENU, habrá disponible un objeto *AUTHENTICATION* el cual representa al usuario.

La siguiente imagen muestra cómo obtener el nombre de usuario del objeto Authentication que le llega.

```
@Configuration
public class TokenUtils {

    @Autowired
    private OAuthProperties oAuthProperties;

    public String getUserLogin(Authentication authentication) {

        String token = ((OAuth2AuthenticationDetails) authentication.getDetails()).getTokenValue();

        Jwt jwt = JwtHelper.decodeAndVerify(token, new RsaVerifier(oAuthProperties.getPublicKey()));

        JSONObject json = JSONObject.fromObject(jwt.getClaims());
        JSONObject user = JSONObject.fromObject(json.getString(DautConstants.USER));
        String login = user.getString(DautConstants.USER_LOGIN);

        return login.toUpperCase();

    }
```

Figura 32 - Desencriptado de token

La aplicación MENÚ, muestra las aplicaciones a las que el usuario tiene acceso, y brinda la posibilidad de navegarlas. Para esto se requiere tener una lista de las aplicaciones a las que el usuario tiene acceso. La siguiente imagen muestra un fragmento de código que obtiene una lista de estas aplicaciones en cuestión.

```
OAuth2Authentication oAuth2Authentication = (OAuth2Authentication) principal;

@SuppressWarnings("rawtypes")
Map mapa = (Map) oAuth2Authentication.getUserAuthentication().getDetails();

OAuth2AuthenticationDetails details = (OAuth2AuthenticationDetails) oAuth2Authentication.getDetails();

String token = "Bearer " + details.getTokenValue();

String apps = mapa.get("APP_LIST").toString();


ObjectMapper mapper = new ObjectMapper();
mapper.configure(SerializationFeature.ORDER_MAP_ENTRIES_BY_KEYS, true);
JsonNode rootNode = mapper.readTree(apps);
Iterator<JsonNode> elements = rootNode.elements();

List<JsonNode> lista = new ArrayList<JsonNode>();

while (elements.hasNext()) {
    lista.add(elements.next());
}
logger.info("Las aplicaciones obtenidas del token son: {}.", lista);
```

Figura 33 - Obtención lista de aplicaciones disponibles

CAPÍTULO 7 - Conclusiones

En el presente trabajo se comenzó describiendo el sistema e-Sidif, el cual fue diseñado para soportar la formulación del presupuesto nacional y el registro completo de la ejecución presupuestaria nacional. Dicho sistema lleva casi 2 décadas de desarrollo, y actualmente sigue sumando requerimientos.

Interactuando con el e-Sidif se encuentran un conjunto de aplicaciones web, las cuales hacen una tarea específica y a su vez comparten el mismo conjunto de usuarios.

Estas aplicaciones Web requieren del mismo chequeo de seguridad que está implementado en el e-Sidif. También se requiere tener sincronizado los datos de login del usuario entre todos los sistemas.

Tener estas políticas de seguridad en diferentes bases de datos y sistemas LDAPs, ó tener replicada la implementación del chequeo de autenticación y autorización, hace que la sincronización requerida no sea un objetivo fácil de cumplir, así es que se optó por implementar un sistema de seguridad centralizado, el cual va a ser usado por todas las aplicaciones Web de la DGSIAF que requieran un intercambio de información con el e-Sidif ó entre ellas para llevar a cabo su objetivo. Así mismo este sistema de seguridad brindará una experiencia de SSO al usuario final.

Así es que surge la idea del desarrollo de DAUT, una aplicación que se encarga de centralizar la autenticación, las políticas de acceso y autorización de las aplicaciones web y como complemento la librería *daut-connector* la cual permite a las aplicaciones una comunicación efectiva con los servicios que provee DAUT.

La implementación de DAUT y la librería para comunicarse con este, brinda a los usuarios finales de estas aplicaciones, un único punto de entrada para autenticarse a estas. Asimismo, los desarrolladores de aplicaciones que requieran una comunicación con el e-Sidif, no deberán preocuparse por implementar la funcionalidad para que los usuarios se identifiquen, sino que configurando su aplicación con la librería daut-connector ya dispondrán de un mecanismo de autenticacion y autorizacion seguro .

La utilización del soporte que brinda el framework Spring Security, el cual es un framework muy maduro y confiable, resultó una buena elección para la implementación del sistema de seguridad centralizado, ya que incluyendo ciertas dependencias y realizando configuraciones, Spring Security se encarga de todo lo relacionado a la seguridad. Y una de las grandes ventajas de este, es que permite

realizar una serie de parametrizaciones y ajustes para un gran abanico de posibilidades y así se logró adaptar a lo que DAUT requería.

Referencias

[1] - e-Sidif - El Sistema Integrado de Información Financiera Internet (e-SIDIF): https://www.argentina.gob.ar/economia/sechacienda/dgsiaf/e-sidif . *Último acceso 23/08/2022.*

[2] - DGSIAF - Dirección General de Sistemas Informáticos de Administración Financiera: https://www.argentina.gob.ar/economia/sechacienda/dgsiaf . *Último acceso 23/08/2022.*

[3] - Enterprise Applications - https://martinfowler.com/bliki/EnterpriseApplication.html .*Último acceso 23/08/2022.*

[4] - M. N. Huhns and M. P. Singh, "Service-oriented computing: key concepts and principles" in IEEE Internet Computing, vol. 9, no. 1, pp. 75-81.

[5] - Robert Daigneau, “Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful web services”, 2000.

[6] -N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE https://www.guru99.com/n-tier-architecture-system-concepts-tips.html . *Último acceso 23/08/2022.*

[7] - Rich Client - https://www.eclipse.org/articles/Article-RCP-1/tutorial1.html . *Último acceso 05/08/2022.*

[7] - Conceptos de LDAP - https://ldap.com/basic-ldap-concepts/ . *Último acceso 23/08/2022.*

[8] - SIRECO, Registro de Cuentas Oficiales - https://www.argentina.gob.ar/economia/sechacienda/dgsiaf/sireco. *Último acceso 23/08/2022.*

[9] - SICHE, Sistema de Consulta de Información para Sistemas Heredados - https://www.argentina.gob.ar/economia/sechacienda/dgsiaf/siche. *Último acceso 23/08/2022.*

[10] - SICDE, Sistema de Captura de Datos Externos - https://www.argentina.gob.ar/economia/sechacienda/dgsiaf/sicde. *Último acceso 23/08/2022.*

[11] - Spring Security - https://spring.io/projects/spring-security . *Último acceso 23/08/2022.*

[12] - OAuth 2.0 - https://datatracker.ietf.org/doc/html/rfc6749 . *Último acceso 23/08/2022.*

[13] - Jason Web Token JWT - https://datatracker.ietf.org/doc/html/rfc7519 . *Último acceso 23/08/2022.*

[14] - HTTP Filters - https://spring.io/guides/topicals/spring-security-architecture. *Último acceso 23/08/2022.*

[15] - Filter chain and Filter Chain Proxy - https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html .*Último acceso 23/08/2022.*

[16] - OAuth2 Spring - https://spring.io/guides/tutorials/spring-boot-oauth2/ . *Último acceso 23/08/2022.*