<CIS1910CP01>

WS-GUARDIAN CLIENT'S AGENT

CHRISTIAN GIOVANNY ROJAS DÍAZ
JUAN DAVID GAMA PEÑA
MICHAEL ANDRÉS VARGAS BUITRAGO
RIE KANEKO BOJACÁ
WILLIAM ALEXANDER MORENO PRIETO

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
PROGRAMA DE INGENIERÍA DE SISTEMAS
BOGOTÁ, D.C.
2019

<CIS1910CP01>

WS-GUARDIAN CLIENT´S AGENT

Autores:

Christian Giovanny Rojas Díaz
Juan David Gama Peña
Michael Andrés Vargas Buitrago
Rie Kaneko Bojacá
William Alexander Moreno Prieto

MEMORIA DE PROYECTO DE GRADO PARA EL CUMPLIMIENTO DE UNO DE LOS REQUISITOS PARA OPTAR AL TÍTULO DE INGENIERÍA DE SISTEMAS

Director

Ing. Javier Humberto Galindo

Jurados del Proyecto Final de Pregrado

Ing. Samuel Gutiérrez Tibaquirá

Ing. Oscar Morales Quintero

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
PROGRAMA DE INGENIERÍA DE SISTEMAS
BOGOTÁ, D.C.
Noviembre, 2019

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
PROGRAMA DE INGENIERÍA DE SISTEMAS

Rector de la Pontificia Universidad Javeriana

Jorge Humberto Peláez Piedrahita, S.J.

Decano de la Facultad de Ingeniería

Ing. Lope Hugo Barrero Solano

Directora de la carrera de Ingeniería de Sistemas

Ing. Mariela Josefina Curiel Huérfano

Director de Departamento de Ingeniería de Sistemas

Ing. Efraín Ortiz Pabón

Artículo 23 de la Resolución No. 1 de junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTO

A lo largo de la carrera tuvimos la fortuna de tener el apoyo de varias personas que nos hicieron crecer, tanto académica como personalmente. Primero, queremos agradecer a nuestros padres, quienes nos han apoyado y traído hasta este punto. A nuestros maestros, que sin duda nos han dado sus conocimientos y siempre han estado pendientes de nosotros, resolviendo dudas y dándonos consejos para formar nuestro camino profesional. A nuestros amigos y compañeros, por la colaboración en clases, trabajos en grupo, y también por el apoyo fuera de lo académico durante los momentos divertidos y tristes. Siempre creciendo juntos como familia. Queremos agradecer a la Pontificia Universidad Javeriana por darnos la oportunidad de conocer estas excelentes personas, y brindarnos las herramientas necesarias para realizar trabajos como este, siendo una base para seguir saliendo adelante. Por último, agradecemos a la empresa ITAC S.A. por su colaboración y paciencia durante el desarrollo de este proyecto.

CONTENIDO

ABSTRACT

The use of Internet becomes more and more indispensable daily, being one of the most important communication channels at the time of sending data. Most of the bytes transported are sensitive information of users, which is increasingly valuable, but with more risks of being violated. For this reason, the issue of security to guarantee the confidentiality and integrity of this type of data protection interactions is essential, however, it is also largely ignored on several occasions due to its complexity of implementation. *WSG-Client's Agent* with the support of *WS-Guardian*, applies and processes security policies to messages directed to SOAP services facilitating their safe consumption.

I- INTRODUCCIÓN

El proyecto busca externalizar la implementación de técnicas de seguridad, logrando que el consumo de *Web Services* protegidos por *WS-Guardian* de la empresa *ITAC S.A* sea más transparente frente al usuario, al este poder consumir el servicio de forma normal, sin necesidad de añadir técnicas de seguridad manualmente. El prototipo se encargará de ver y aplicar las políticas que cada servicio requiera. De esta forma se reducen los costos que recaen sobre el cliente, al no tener que modificar ni adaptar cada aplicación que hace uso de un servicio SOAP protegido por el servidor *WS-Guardian*.

El presente documento muestra el desarrollo del proyecto de grado a lo largo del año, tanto su planeación como su desarrollo. Aquí se explican las fases y como se abordaron en la realidad, además de los recursos utilizados, especificación de procesos, resultados de los casos de prueba y conclusiones. Es por esto que se encuentra dividido en diferentes partes, iniciando con la descripción general donde se muestra desde la problemática hasta la solución del proyecto; luego sigue con una descripción más clara del mismo, y se muestran los objetivos y los entregables, todo esto como primera y segunda fase.

En la tercera fase se habla sobre el contexto en que se trabajó junto con la justificación del uso del producto, lo cual ya fue mencionado en la propuesta del proyecto (VFP). En la cuarta fase se encuentra el análisis del problema, expuesto con más detalle en el SRS quien menciona donde se encuentran los requerimientos. En la quinta fase se muestra el diseño del producto final desde diferentes perspectivas como despliegue, lógica y proceso. Estos mismos se pueden encontrar con más detalle en el SDD.

La sexta fase muestra la metodología empleada para el ciclo de desarrollo del proyecto, junto a las tecnologías usadas en la implementación, detalles sobre los acuerdos, y el proceso realizado semana a semana. Los detalles y control de la etapa de desarrollo se encuentran en el informe PostMortem. Para finalizar, una vez que se ha explicado lo anterior, se muestra el producto final, los resultados de los casos de prueba y su análisis como séptima fase.

II- DESCRIPCIÓN GENERAL

1. Oportunidad, Problema

1.1. Contexto del Problema

La empresa *ITAC* S.A. actualmente ofrece al mercado el producto *WS-Guardian* (en adelante WSG), el cual es una herramienta que potencia la *Arquitectura Orientada a Servicios* (SOA) protegiendo el tráfico de entrada y salida del lado del servidor de *Servicios Web* (SW), bajo criterios de confianza y privacidad a nivel de mensajes, por medio de técnicas semánticas y basadas en *Web Services Security* tales como *encriptación*, *username-token*, *timestamp*, *firmas* y *certificados digitales*. WSG se caracteriza por brindar gobernabilidad y administración de políticas de seguridad sobre los *Web Services,* y que esté asegurada la transferencia de archivos e información sin la necesidad de alterar las aplicaciones utilizadas por el receptor implicado [1]–[3].

Como se mencionó anteriormente, WSG aplica controles de acceso y seguridad semántica a nivel del mensaje para asegurar que estos sean protegidos completamente, y no solo a nivel de tráfico como lo hace el protocolo https. Para que la información viaje protegida desde el origen hasta el destino, ya sea en comunicaciones cliente/WSG o WSG/servidor, el emisor debía tener un importante conocimiento de ciberseguridad para implementar las técnicas de seguridad mencionadas en el párrafo anterior [4]. Esto afecta negativamente la adopción de la seguridad en las interacciones de los servicios web, dado que el incluir políticas de seguridad en los servicios web implica cambios en las aplicaciones que consumen los consumen, y da un motivo más para que una compañía no implante medidas de aseguramiento en el uso de SW.

1.2. Formulación del Problema

Los clientes de los servicios asegurados mediante *WSG* eran los responsables de implementar los principios de seguridad: *autenticación, confidencialidad, no repudio e integridad* por medio de *tokens de usuario, certificados y/o firmas digitales.* Muchos clientes cuentan con su propio despliegue de las aplicaciones, y por ende, hacer una modificación e implementación de estos sistemas era costoso en materia de tiempo, dinero y esfuerzo. Pero, *WS-Guardian Client's Agent* siendo la solución propuesta de este proyecto, permite incluir estas características de seguridad en los consumidores de servicios web (clientes de WSG), sin la necesidad de realizar modificaciones en dichas aplicaciones.

1.3. Propuesta de Solución

Para dar solución a la problemática se propuso implantar un Agente denominado *WS-Guardian Client's Agent* (en adelante *WSG-Client*), el cual es ejecutado en la máquina del

cliente, para poder aplicar las políticas que aseguren los principios de seguridad mencionados en la sección anterior. Esto es en los mensajes emitidos hacia *WSG*. *WSG-Client* brinda ciertas funcionalidades como: ser capaz de solicitar, descargar e integrar las funcionalidades que requiera, [5] para cumplir con las restricciones de seguridad en el momento de consumir un SW. Adicionalmente, este agente es desplegado en la máquina del cliente de manera separada de sus aplicaciones.

La *Figura 1: Infraestructura original* ilustra la arquitectura de consumo de servicios web sin el uso de *WSG-Client* mediante mensajes *http* o *https*. *WSG* aparece como una capa intermedia para la inclusión de mecanismos de seguridad en el consumo *servicios web* (servicios del negocio del cliente que requieren seguridad para ser consumidos) expuestos por una organización.

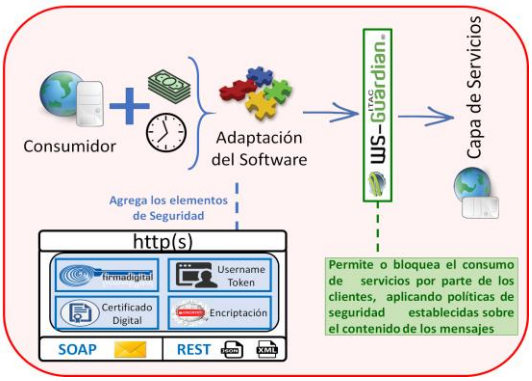

Figura 1: Infraestructura original

En la *Figura 2: Arquitectura conceptual de la solución* se ilustra la interacción a partir de la inclusión de *WSG-Client* y la capa de exposición de servicios. *WSG-Client* reemplaza la adaptación del sistema que debe hacer el cliente en su aplicativo para incluir las políticas de seguridad.

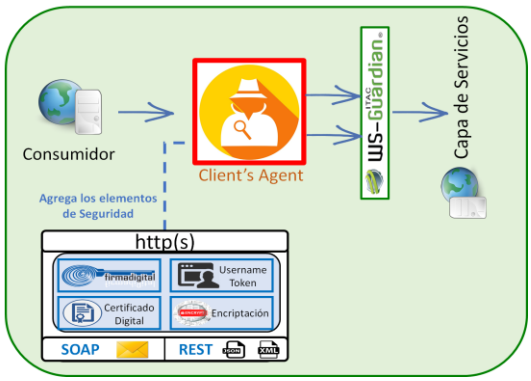

Figura 2: Arquitectura solución

1.4. Justificación de la Solución

En el contexto del uso de servicios web existen consumidores que no tienen experiencia en la implementación de concepto de ciberseguridad, y/o sus planes de implementación de software no tienen como prioridad el aseguramiento que el tráfico de mensajes requiere. Por esto, normalmente peticiones viajan por la red sin usar el nivel de protección adecuado.

Para fomentar la seguridad en el uso de los servicios web con apoyo del producto *WSG,* surgió *WSG-Client.* Este le ahorra al cliente la implementación de técnicas de aseguramiento en su sistema, y le evita hacer modificaciones al mismo. *WSG-Client* trabaja con los protocolos http y https siendo soportados por los SW. La solución fue planteada como un componente que se instala en la máquina del cliente, para que la petición del servicio pueda ser asegurada desde su origen hasta su destino, evitando que viaje de forma vulnerable.

2. Descripción del Proyecto

2.1. Objetivo General

Diseñar un agente que se ejecute en el lado del consumidor de servicios, el cual proporcionará principios de seguridad como *autorización, confidencialidad, integridad y no repudio* sobre los mensajes emitidos hacia WS-Guardian.

2.2. Objetivos Específicos

- Diseñar el agente para que pueda ser ejecutado en dos o más Plataformas
- Implementar un prototipo del agente para dos o más Plataformas
- Implementar dos técnicas de seguridad dentro del agente
- Determinar la validez de los requerimientos funcionales y no funcionales por medio del prototipo
- Demostrar la calidad de software mediante una metodología de pruebas

2.3. Entregables, Estándares y Justificación

Entregable	Estándares asociados	Justificación
PMP (Software Project Management Plan)	ISO/IEC/IEEE 16326-2009 UML BPMN	Para el documento PMP se tomó como base el estándar IEEE 1058.1-1987, adaptando su contenido para ese proyecto. En este documento se describe una vista general, contexto y administración del proyecto. De la misma manera se definen herramientas, estándares y metodología de trabajo para su desarrollo [6]. Se utiliza estándar UML [7] y BPMN [8] para representación de diagramas de secuencia y procesos que se llevarán a cabo en el proyecto.

SRS (Software Requerements Specification)	IEEE 830-1998 UML BPMN	La plantilla para SRS es tomada del estándar IEEE 830-1998. Aquí se especifican todas las funcionalidades y atributos pertenecientes al sistema a desarrollar, junto con los lineamientos acordados con el cliente (*ITAC S.A.*) [9]. Se utiliza estándar UML [7] y BPMN [8] para representación de diagramas de secuencia y procesos que se llevarán a cabo en el proyecto.
SDD (Software Design Description)	IEEE 1016-2009 UML BPMN	Se adoptó el estándar IEEE 1016-2009 para el SDD, con el fin de establecer la arquitectura y las interfaces lógicas de usuario que serán diseñadas e implementadas [10]. Se utiliza estándar UML [7] para la representación de diagramas de secuencia y procesos que se llevarán a cabo en el proyecto.
Informe de Pruebas		En este documento, se pretenden mostrar las pruebas y resultados del testing realizados sobre el prototipo funcional, con el fin validar el correcto funcionamiento de los módulos implementados, y el cumplimiento de los requerimientos. Se utiliza la plataforma Spira, la cual genera un reporte de pruebas definiendo *casos de prueba*. Esto debido a que es utilizado por el cliente (ITAC S.A.).
Prototipo Funcional "WSG-Client´s Agent"	Versionado Semántico 2.0.0-rc.2	Prototipo funcional acorde a los requerimientos especificados en el documento SRS junto con el cumplimiento de los objetivos establecidos. El estándar de versionamiento semántico, define como asignar y aumentar los números de versión a lo largo del desarrollo del software [11].
Reporte Postmorten		Reporte generado por la plataforma Spira, el cual muestra el resumen de los trabajos realizados (cada spring, esfuerzo y tiempo) de cada integrante del equipo.
Memoria	Plantilla dado por la Universidad Javeriana	Documento cual es leído por los jurados del proyecto de grado, el cual muestra el trabajo realizado a lo largo de la planeación y desarrollo del trabajo. La plantilla es entregada por parte de la carrera de ingeniería de sistemas.

Tabla 1: Entregables y Estándares

III- CONTEXTO DEL PROYECTO

1. Contexto

Actualmente, cuando una empresa instala políticas de seguridad sobre sus *SW*, la responsabilidad de implementar técnicas de seguridad para consumirlos queda a cargo del cliente del servicio. Esto hace que el incluir aseguramiento en sus servicios web sea costoso en materia de tiempo, dinero y esfuerzo, al tener que hacer los cambios en las aplicaciones consumidoras.

Con *WSG-Client* se quiso reducir los costos adicionales (tiempo, dinero y esfuerzo) al momento de consumir un *WS*. Al lograr automáticamente conocer qué políticas requiere el servicio para ser consumido (estas pueden añadirse, modificarse y/o ser removidas desde *WS-Guardian*) y aplicándolas sobre el mismo, se disminuyen costos para el cliente ya que no tiene que modificar su sistema.

El *WSG-Client* crea la comunicación con el servidor *WS-Guardian* de *ITAC S.A.* conociendo su dirección IP, mencionado en las secciones anteriores. Esto es para conocer tanto los servicios como las políticas que se deben aplicar, a cada una de las operaciones. También puede obtener (descargar desde el servidor) el código en tiempo de ejecución para aplicar las políticas de seguridad.

Este proyecto tiene en cuenta que la empresa *ITAC S.A.* está interesada en el *WSG-Client,* debido a que son los dueños del servidor *(WS-Guardian)*. El prototipo de *WSG (módulo de administración)* se conecta con este servidor. Este debe ser aceptado y desplegado por *ITAC S.A.* en los clientes interesados. Para poder tener un prototipo estable, no se tuvieron en cuenta cambios externos durante el desarrollo.

El *WSG-Client* no es soporte de comunicación tipo REST sino SOAP 1.1, debido a que este es el protocolo de comunicación que en este momento se encuentra estandarizado en la industria, y es el protocolo que maneja el estándar *Web Servicie Security* [12].

Hay que tener en cuenta que *WSG-Client* no cubre la totalidad de las políticas de seguridad que maneja el WSG, sino dos de ellas únicamente; las cuales fueron mencionadas en los objetivos específicos. De igual manera, no se contará con motor de base de datos ni conexión para centralizar los logs.

2. Análisis del Contexto

Como se mencionó en las secciones anteriores, el servidor WSG aplica controles de acceso y seguridad, para proteger completamente la información de los servicios sensibles del cliente; cabe aclarar que no solo es a nivel de tráfico sino también a nivel del mensaje. Esto caracteriza al servidor por brindar gobernabilidad y administración de políticas de seguridad sobre los servicios web.

WSG-Client realiza de manera automática la aplicación, validación y eliminación de las políticas de seguridad en los mensajes SOAP, los cuales son verificados y procesados por WSG según las necesidades del cliente. Esto permite mejorar la adopción de la seguridad, y evitar futuras modificaciones que puedan generar gastos extra al cliente. Es decir, ayuda a reducir costos en materia de tiempo, dinero y esfuerzo.

Programas como *Oracle Web Services Manager* o *Digital Guardian Data Cloud Protection,* realizan soluciones similares ante una problemática similar. Pero, cada uno posee diferentes características:

Oracle Web Services Manager permite administrar políticas y asegurar los servicios web en toda una organización. A pesar de que puede ser usado como un componente o desplegado como otra aplicación, está sujeto a usar solamente tecnología Oracle; lo que significa que su portabilidad (capacidad de ser migrada en diferentes tipos de hardware y software) es media [13]. Esto no es aplicable como solución a nuestro problema debido a que no se puede obligar a los clientes a usar Oracle en sus servidores.

Digital Guardian Data Cloud Protection permite cifrar, procesar y firmar digitalmente archivos confidenciales almacenados en la nube como Office 365, antes de transferirlos de manera automática. Este producto está relacionado con otros productos de *Digital Guardian* como *Digital Guardian Network DLP* para monitorear y controlar el flujo de datos [14], [15]. Al igual que el anterior, esto no es aplicable como solución a nuestro problema debido a que se requieren más de un programa para su uso. Además el monitoreo y control de flujo es realizado por WSG actualmente.

Adicionalmente, ninguno de estos productos es flexible (no es fácil realizar cambios en sus configuraciones) debido a que son desarrollados por empresas privadas.

WSG-Client´s Agent es un programa implementado en lenguaje JAVA (OpenJDK 8), lo que permite que se pueda ejecutar sin importar el sistema operativo que maneja el servidor. Esto lo hace portable a diferencia de los dos programas mencionados. Adicionalmente, es flexible ya que es de código abierto.

Como objetivo de WSG-Client's Agent*,* el programa debe ser ligero para poder ejecutarlo en la mayoría de los servidores del cliente. Por esto, WSG-Client inicialmente solo tiene la conexión directa con el servidor WSG, donde está almacenado su componente de administración con cada política de seguridad en un JAR. Esta se descarga sólo si la necesita para luego usarla con Java Reflexión.

IV- ANÁLISIS DEL PROBLEMA

En la presente sección se resume el problema con detalles importantes como los requerimientos y limitaciones sobre las que se trabajó en el producto final. Los detalles más específicos son contados en el documento de especificación de requerimientos de software (SRS).

1. Requerimientos

Teniendo en cuenta el contexto y la solución de la propuesta, se plantearon una serie de requerimientos para cumplir con las funciones principales según lo establecido en los objetivos específicos.

Los requerimientos establecidos son funcionales y no funcionales, los cuales se utilizaron para la creación de las Historias de Usuario. Esto además fue algo solicitado por *ITAC S.A.* debido a la flexibilidad que permiten para realizar cambios a lo largo de la implementación. Las Historias de Usuario se pueden observar con detalle en el Anexo 1 – Historias de Usuario.

Para la especificación de los requerimientos se tuvieron en cuenta tres actores: administrador, cliente y desarrollador. El primero es el encargado de configurar el producto de *WSG-Client* y su módulo de administración en *WSG;* el segundo consume servicios web en *WSG* a través de *WSG-Client*; y el tercero sigue el desarrollo y/o mantenimiento del *WSG-Client.* Este último es un miembro de la empresa I*TAC S.A.,* quien tiene acceso al código fuente del aplicativo. Tanto el administrador como desarrollador deben ser empleados de *ITAC S.A.,* y el cliente es el propietario de los servidores donde se va a desplegar *WSG-Client*.

Para el actor cliente se tiene un archivo con parámetros definidos, el cual está ubicado en el directorio raíz del aplicativo. Este es usado para la configuración inicial de *WSG-Client's Agent* y sólo permite entradas por teclado. El actor administrador, al ser el encargado de administrar *WSG,* cuenta con una interfaz web donde puede registrar los diferentes WS-Client haciendo uso de su respectiva ID y dirección física (MAC). Esta interfaz estará contenida en *WS-Guardian*.

Para la implementación de *WSG-Client*, se tuvo en cuenta el uso de lenguaje JAVA sobre el servidor JRE con el fin de que este pueda ser ejecutado de manera híbrida. Más explicación sobre esto en el documento Software Project Management Plan (SPMP), sección 3.

La interfaz de comunicación entre *WSG-Client* y *WSG* utiliza el estilo arquitectural REST para descargar los componentes (JAR) a través del protocolo https. *WSG-Client,* además cuenta con una interfaz para recibir peticiones http (contenidas de mensajes SOAP 1.1) del cliente al

que está asignado. Esto es para incorporar las técnicas de seguridad necesarias sobre el mensaje SOAP, y luego consumir el *Servicio Web* protegido por *WS-Guardian*.

2. Restricciones

El proyecto contaba con restricciones de software que limitan el alcance del desarrollo. Es importante que estas restricciones fueran discutidas y planeadas dentro del equipo y con *ITAC S.A.* desde el inicio del proyecto, para así iniciar con el desarrollo del *WSG-Client*.

Las restricciones que se tomaron en cuenta para el desarrollo del proyecto son las siguientes:

Tiempo

El desarrollo del trabajo tiene una duración de 10 meses, este tiempo está dividido en dos periodos. Siendo el primer semestre de 2019 la etapa de definición y planeación general, y el segundo semestre de 2019 la implementación del producto final, junto con la creación de los documentos necesarios mencionados en el VFP y al inicio del presente documento.

En el proyecto se combinaron las metodologías RUP y SCRUM, tomando las fases de desarrollo de la primera *(Inicio, Elaboración, Construcción y Validación y Cierre)* y las buenas prácticas de la segunda. Cada sprint tenía un rango de 8 a 15 días de duración. Esta planeación es especificada en el documento SPMP y en el informe de PostMorten.

Cliente

A nivel de software, como *WSG-Client* hace uso de ciertas herramientas específicas mencionadas en el documento SPMP, el cliente debe tener *Java Runtime Environment v1.8* con el fin de ejecutar el producto final. A nivel de hardware, se exige tener un espacio de 1GB de RAM mínimo y 1GB mínimo de espacio en disco.

Herramientas

WSG-Client debía ser desarrollado sobre *Openjdk 8* y las políticas de seguridad debían ser aplicadas de acuerdo con el estándar *WS-Security.* Esto porque *ITAC S.A.* en su producto *WS-Guardian* usa este estándar, para la aplicación y validación de la seguridad sobre los mensajes SOAP. Esto crea limitaciones en los lenguajes de programación (programar en JAVA) y en los estándares de seguridad (uso de *WS-Security*).

ITAC S.A.

Siendo el *WS-Guardian* un producto de la empresa *ITAC S.A.,* se solicitó acceso a este sistema para realizar la configuración de servicios SOAP y realizar pruebas de calidad sobre el producto final. Adicionalmente, se solicitó la estructura de la base de datos manejada por *WS-Guardian* para adaptarla al módulo de administración.

Se firmaron derechos patrimoniales con la empresa, con el fin de tener un acuerdo sobre la propiedad intelectual del producto final y sobre la confidencialidad, debido a que se tuvo acceso a un servidor de su propiedad para realizar las pruebas finales. Todo esto puede ser observado en el documento VFP, sección 4.

Recursos

A lo largo de la implementación no se tuvo presupuesto monetario para el desarrollo del producto. Pero existieron sanciones monetarias a nivel de Scrum Team, al momento de incumplir con algún reglamento del establecido inicialmente. Todo esto también es explicado con mayor detalle en el SPMP.

3. Especificación Funcional

Los requerimientos fueron divididos en funcionales y no funcionales, y se utilizaron historias de usuario, para la creación de estas se tuvieron en cuenta los actores anteriormente mencionados en la sección Requerimientos.

En la *Tabla 2: Características*, se pueden observar los requerimientos funcionales de alta prioridad (definidos como prioridad *Must have*). Todos los requerimientos funcionales se muestran en el *Anexo 1 – Historias de Usuario.xls*, hoja “Características”.

ID	Historia de Usuario	Comportamiento Esperado
F-01	Como administrador, quiero registrar en WS-Guardian los clientes WSG-Client's Agent para permitirles el acceso.	WSG-Client's Agent registrado para un cliente en WS-Guardian.
F-02	Como administrador, quiero ver el listado de servicios en WS-Guardian a los que tiene acceso un WSG-Client's Agent, para obtener la configuración del cliente.	Mostrar la lista de servicios a la que tiene acceso WSG-Client's Agent.

F-03	Como administrador, quiero cargar en WS-Guardian los archivos JAR asociados a las políticas, para enviar la configuración a un WSG-Client's Agent.	Llave dentro del Jar que WSG Client's Agent usará.
F-04	Como desarrollador, quiero que WSG-Client's Agent use una estructura con los métodos para aplicar y soportar las políticas.	Que las nuevas políticas implementadas tengan las mismas estructuras de métodos.
F-05	Como cliente, quiero que WSG-Client's Agent descargue y guarde la información de configuración (servicios web y sus políticas) de WS-Guardian, para no configurar manualmente los servicios.	Información de servicios y políticas WSG-Client's Agent cargada en memoria.
F-06	Como cliente, quiero que el WSG-Client's Agent descargue de WS-Guardian el JAR con la lógica para aplicar políticas de seguridad al mensaje.	Obtener Jar de la Política en el cliente.
F-08	Como cliente, quiero que el WSG-Client's Agent cargue la lógica de cada política de seguridad desde un JAR, para aplicar las políticas a mensajes dirigidos a los WS protegidos por WSG.	Cargar lógica en tiempo de ejecución de WSG-Client's Agent.
F-09	Como cliente, quiero que WSG-Client's Agent compruebe la expiración de validez de cada política y actualice su información cada vez que se requiera, para mantenerse con la configuración pertinente.	Actualización de las políticas en WSG-Client's Agent
F-15	Como cliente, quiero que el WSG-Client's Agent aplique y quite las políticas de seguridad a un mensaje SOAP de acuerdo a la lógica descargada de WS-Guardian, para consumir un servicio web.	WSG-Client's Agent realiza petición hacia WS-Guardian con políticas aplicadas, WSG-Client's Agent responde al cliente.
F-16	Como administrador, quiero configurar en WSG-Client's Agent la ip y puerto de conexión con WS-Guardian, para establecer la comunicación	WSG-Client's Agent realiza la conexión a WS-Guardian.
F-17	Como cliente, quiero que el WSG-Client's Agent rechace o ignore cualquier mensaje que NO provenga del cliente al que esté asignado para que el servicio solo sea consumido por quien corresponde.	WSG-Client's Agent rechace la petición.

F-18	Como administrador, quiero cargar en WS-Guardian los archivos Jar asociados a las políticas, para enviar la configuración a un WSG-Client's Agent.	Interfaz que permita cargar archivos y asociar el archivo a una política.

Tabla 2: Características

A continuación, se muestra el mapeo y distribución de los requerimientos de *WSG-Client*, relacionando funcionalidades a sus correspondientes módulos del sistema:

- Configuración: Definida para obtener y dar soporte a la configuración (servicios y su relación con las políticas de seguridad).
- Comunicación: Este módulo contendrá toda la lógica para atender las peticiones hechas por el cliente consumidor de WS.
- Políticas: Este módulo es destinado para el uso y gestión de las políticas de seguridad (archivos JAR).
- Auditoría: La finalidad de este módulo es generar las líneas de registro de eventos (logs) que ocurren en el sistema y que son almacenadas en los archivos planos con estructura definida.

	Módulos				**Historias de Usuario**
	Configuración	**Comunicación**	**Políticas**	**Auditoría**	**Id**
Descargar archivos de configuración	✓	✓		✓	F-6, F-10, F-13, F-16
Descargar los JAR con políticas de seguridad	✓	✓	✓	✓	F-7, F-9, F-13
Agregar políticas de seguridad a los mensajes SOAP			✓		F-5, F-15, F-17
Retransmitir mensaje SOAP con políticas hacia WS-Guardian		✓	✓		F-19
Quitar las políticas de seguridad a los mensajes SOAP recibidos de WS-Guardian			✓		F-17
Actualizar políticas de seguridad	✓	✓		✓	F-10, F-11, F-13
Registrar eventos ocurridos en el sistema				✓	F-14

Tabla 3: Mapeo y distribución de requerimientos

En la *Tabla 4: No Funcionales*, se pueden observar los requerimientos funcionales de alta prioridad (definidos como prioridad *Must have*). Todos los requerimientos no funcionales se muestran en el *Anexo 1 – Historias de Usuario.xls*, hoja “No Funcionales”.

ID	Historia de Usuario	Criterio de Aceptación
NF-1	Como usuario, quiero ejecutar el WSG-Client's Agent en plataformas con sistema operativo Windows y Linux, para que pueda ser usado por varios clientes.	El sistema pueda ejecutar en sistemas operativos Windows y Linux.
NF-2	Como desarrollador, quiero que el producto use el protocolo HTTPS para comunicarse con WS-Guardian.	La información viaje encriptada.
NF-3	Como desarrollador, quiero que la arquitectura sea Cliente / Servidor.	El WS-Guardian Client's Agent solicita servicio y WS-Guardian los presta.
NF-4	Como usuario, quiero tener un manual de instalación y ejecución para manipular WSG-Client´s Agent.	El manual de instalación y ejecución sea entendible para el usuario.
NF-8	Como desarrollador, quiero que todo JAR (implementación de políticas) este junto a una firma, para que se valide la integridad del código que será ejecutado en WSG-Client´s Agent.	Todos los JAR sea firmado con la misma llave privado.
NF-9	Como administrador, quiero que para obtener la configuración WSG-Client´s Agent se comunique con WS-Guardian vía REST.	Para la configuración, WSG-Client's Agent sea comunicaco por REST con WS-Guardian.
NF-10	Como desarrollador, quiero que el WSG-Client's Agent utilice SOAP para enviar las peticiones del cliente a WS-Guardian	Se haga uso de SOAP para el envió de peticiones de WSG-Client's Agent para WS-Guardian.

Tabla 4: No Funcionales

V- DISEÑO DE LA SOLUCIÓN

En esta sección muestran y explican los diseños de la arquitectura sobre la que se trabajó, teniendo en cuenta que gracias a la metodología SCRUM se pudieron realizar cambios a lo largo de su desarrollo. La presente sección se puede ver a mayor detalle en el SDD.

1. Stakeholders

Dentro de la aplicación existen dos principales *stakeholders*.

Administrador de WSG-Client´s Agent

Quien se encarga de monitorear todos los *WSG-Client* registrados en cada cliente.

Cliente

Quien es el que hace uso de *WS-Guardian*, asegurando así que los servicios consumidos cuentan con el uso de políticas de seguridad. Estos son los interesados en hacer uso de la aplicación *WSG-Client* dentro de su servidor para la automatización de la aplicación se políticas de seguridad.

2. Arquitectura del diseño

2.1. Vista General

En esta sección se va a mostrar los flujos del producto de manera genérica para tener una idea global del funcionamiento. Esto se va a ser explicado con mayor detalle en las siguientes secciones.

Figura 3: Vista general_Configuración

Como se puede observar en la *Figura 3: Vista general_Configuración*, cuando *WSG-Client's Agent* inicia su ejecución, hace la solicitud de descarga de configuración al módulo de administración (servidor). Este módulo responde con los datos de configuración en formato JSON. Una vez que la aplicación obtiene este archivo, guarda la información y envía peticiones al servidor para descargar las políticas (JAR) de seguridad que usará sobre los mensajes SOAP. Como respuesta recibe archivos JAR contenidos de código ejecutable, ficheros de configuración y certificados. Para finalizar su proceso, *WSG-Client's Agent* hace una última solitud al administrador para obtener el WSDL. Cuando lo recibe la respuesta en formato XML, actualiza las direcciones de los servicios y lo guarda.

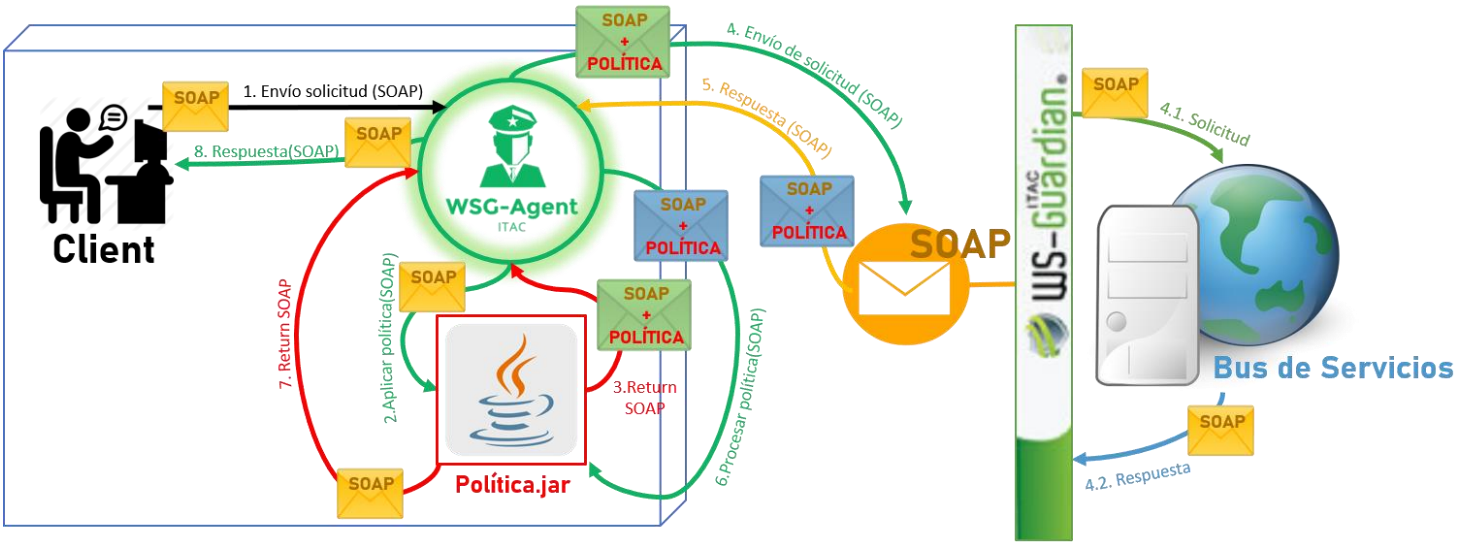

Figura 4: Vista general_Aplicar/Procesar Política

En la *Figura 4: Vista general_Aplicar/Procesar Política,* se muestra el flujo general sobre la aplicación y procesamiento de las políticas de seguridad del servicio del cliente. Primero el cliente (consumidor) envía la solicitud SOAP a *WSG-Client's Agent*, quien se apoya en el(los) JAR correspondiente(s) según su configuración, enviándole el envelope con el fin de aplicar la(s) política(s) requerida(s) para consumir el servicio. Cada JAR responde un mensaje con las etiquetas de aseguramiento agregadas. Luego, *WSG-Client's Agent* envía la petición a la interfaz SOAP de *WS-Guardia*. Este la procesa (valida las restricciones de seguridad) y obtiene una respuesta (mensaje SOAP) del bus de servicios, para luego aplicar las políticas de seguridad y responder la solicitud.

Cuando *WSG-Client's Agent* recibe la respuesta del servicio SOAP invoca nuevamente al JAR de la(s) política(s), para procesar (validar y quitar) las etiquetas de seguridad que *WS-Guardian* agregó al mensaje. El resultado de este proceso (respuesta SOAP sin políticas) es retornado al consumidor final del servicio.

2.2. Vista de despliegue

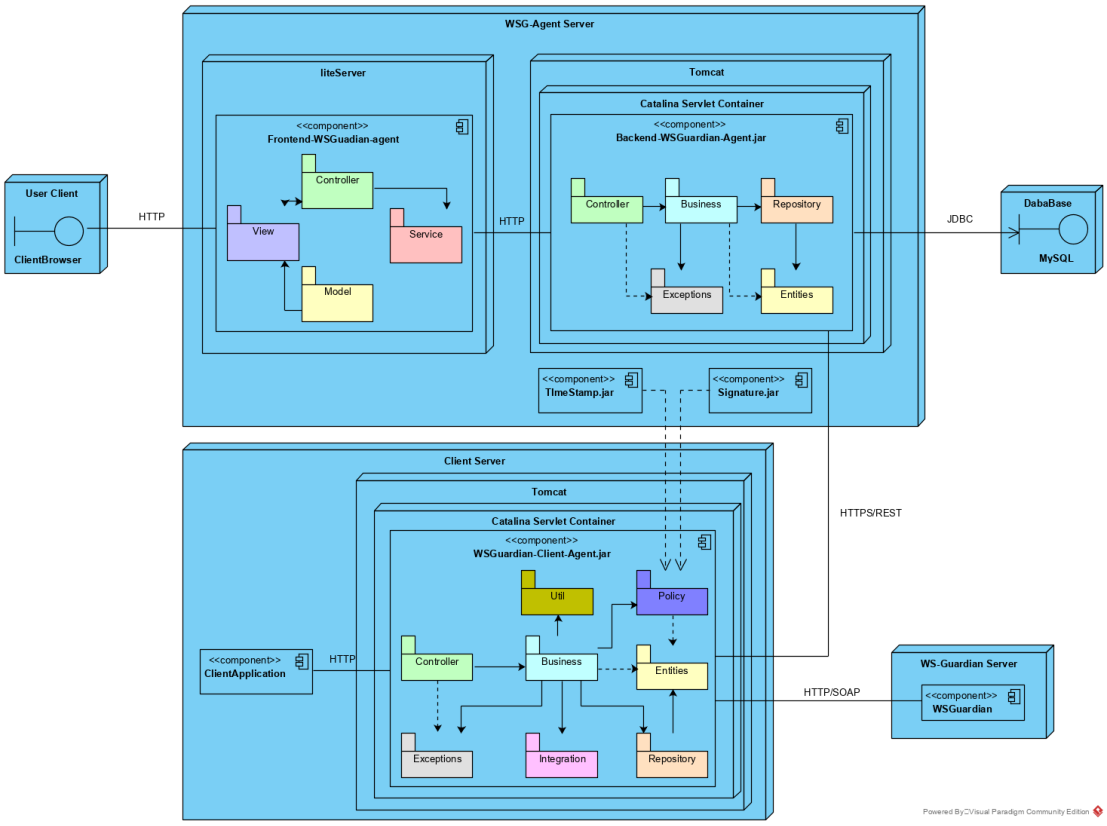

Figura 5: Diagrama de despliegue

En la *Figura 5: Diagrama de despliegue* (*Anexo 2 – Diagrama de despliegue.pdf*) muestra la relación de todos los nodos junto a los componentes y paquetes utilizados para la *WSG-Client's Agent* y su administración. También incluye los nodos externos como lo son: el servidor *WS-Guardian* y el componente de la aplicación del cliente.

Nodos

Nodo	Descripción	Protocolo
User Client	El nodo donde el administrador de *WSG-Client's Agent* hace uso del módulo de administración.	Esto es conectado con el nodo WSG-Agent Server utilizando el protocolo HTTP.
WS-Guardian Server	Servidor de WS-Guardian.	Hace la conexión con el nodo Client Server por HTTPS/SOAP.
WSG-Agent Server	El servidor donde está implementado el administrador de *WSG-Client´s Agent*.	Utiliza el protocolo HTTPS/REST para realizar la conexión con el servidor User client.
Client Server	El servidor del cliente donde se despliega el *WSG-Client's Agent* para consumir los servicios de WS-Guardian y así provee o	La conexión con el WSG-Agent Server se realiza por el protocolo HTTPS/REST

	quita la seguridad cuando viene el servicio.	y con WS-Guardian Server por el protocolo HTTPS/SOAP.
MySQL	Nodo que representa la base de datos que se utiliza en *WSG*. Esto es una base de datos relacionados llamado MySQL.	Tanto con el servidor del cliente como el servidor de WSG-Agent hace uso de JDBC para la consulta.

Tabla 5: Descripción de nodos

Componentes

Componente	Descripción
Frontend-WSGuardian-Agent	Componente que representa el FrontEnd para la administración de *WSG-Client's Agent* dentro del servidor *WSG*.
Backend-WSGuardian-Agent	Componente que representa el BackEnd de la administración de *WSG-Client's Agent* dentro del servidor *WSG*.
WSGuardian-Client-Agent	Componente que representa el producto *WSG-Client's Agent* cual es desplegado dentro del servidor del cliente.
WSGuardian	Componente donde se tiene toda la lógica de *WS-Guardian*. Esto se encuentra dentro del nodo WS-Guardian Server.
ClientAplication	Componente que contiene la lógica de consumo de los servicios que expone WSG-Client's Agent. Esto se encuentra dentro del nodo Client Server.
Signature	Componente que contiene las clases para aplicar/procesar la firma de los servicios. Esto se encuentra dentro del nodo *WS-Guardian Server* y cuando el cliente lo requiere, hace la descarga de este.
TimeStamp	Componente que contiene las clases para aplicar/procesar la marca de tiempo de los servicios. Esto se encuentra dentro del nodo *WS-Guardian Server* y cuando el cliente lo requiere, hace la descarga de este.

Tabla 6: Descripción de componentes

2.3. Vista lógica

WSG-Client´s Agent

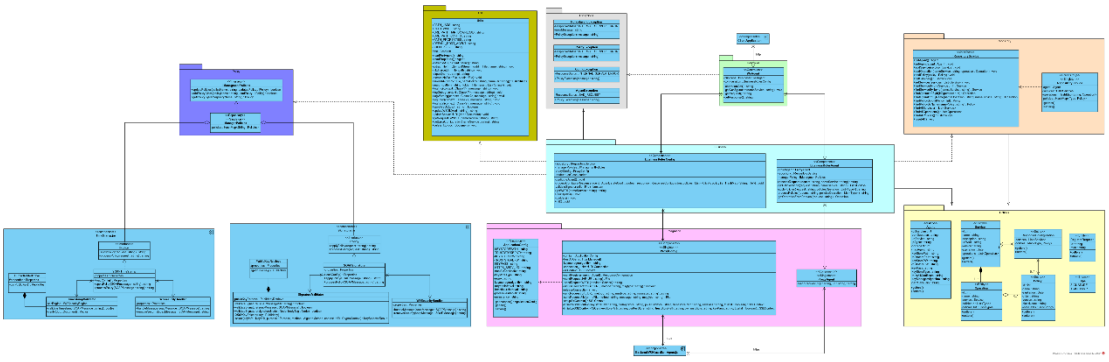

Figura 6: Diagrama de clases_WSG-Client´s Agent

En la *Figura 6: Diagrama de clases_WSG-Client´s Agent* (*Anexo 3 – Diagrama de clases_WSG-Client´s Agent.pdf*) se muestra el diagrama de clases de la aplicación *WSG-Client's Agent* que se encuentra desplegado dentro del servidor del cliente, como se mencionó en la sección anterior. A continuación, se muestra la descripción de cada clase.

El paquete *Business* contiene las clases con la lógica de negocio. Esta está dividida en dos clases: una para la configuración inicial de la aplicación, esta se comunica con el *WSAgent* para realizar la configuración inicial como se mencionó anteriormente; y otra para las políticas de seguridad (aplicar y procesar), esta se comunica con el *WSAgent* para obtener o enviar la operación.

El paquete *Controller*, contiene clases de controladores que se encargan de comunicarse con la aplicación del cliente; y el paquete *Integration*, contiene clases que se encargan de integrar con otras clases que tiene el servidor de *WS-Guardian*. Dentro de ello, se encuentra la clase *ProxyConfig* que hace la conexión con el servidor *WS-Guardian* para la configuración inicial del WSG-Client's Agent y *ProxyAgent* que se encarga de mandar el *request* al servidor *WS-Guardian*.

En el paquete *Util,* se encuentra la clase Útil con variables globales que son usadas en las clases de negocio y también contiene métodos que son importantes la lógica. Se encuentran en este paquete ya que dentro de ellas no debería haber métodos como los de manejo de logs, guardar o actualizar el WSDL, guardar el JAR o guardar y leer archivo.

Otro paquete importante para este diagrama es el paquete *Policy*. Este contiene clases que se encargan de administrar las políticas de seguridad, tanto para aplicar como para quitar.

Todas las clases entidades se almacenan en el paquete *Entities* y en el paquete *Repository* contiene las clases de repositorios de servicios del cliente.

También existe el paquete *Exceptions*, donde se encuentran las clases de excepciones para tener un buen manejo de los errores.

Administrador de WSG-Client´s Agent

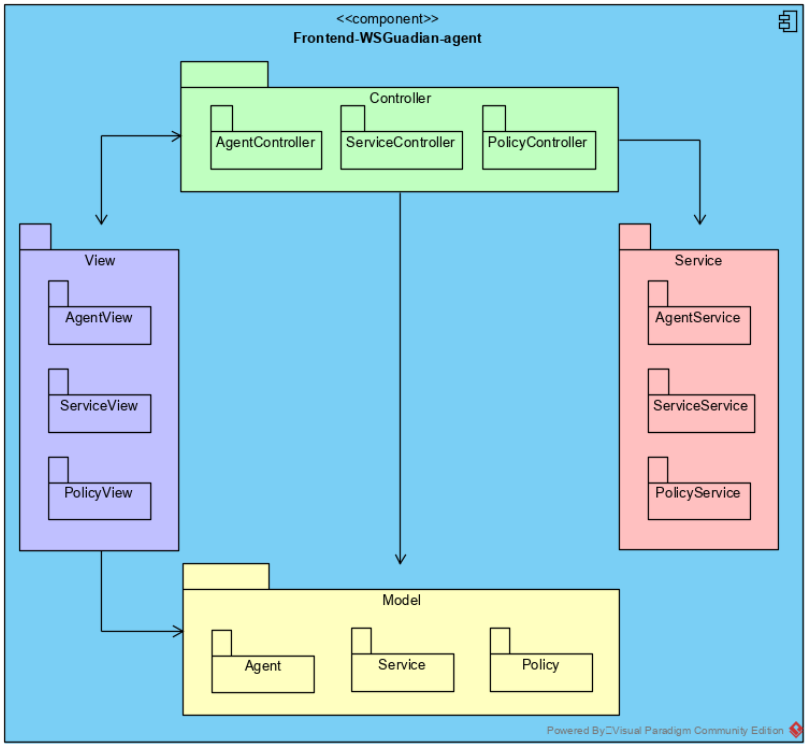

Figura 7: Diagrama de paquetes - administración del WSG-Client´s Agent (Frontend)

En la *Figura 7: Diagrama de paquetes - administración del WSG-Client´s Agent (Frontend),* muestra el diagrama de paquetes que utilizados dentro del componente *WSClientAplication*, representa el *FrontEnd* para la administración de los agentes, servicios de los clientes y las políticas de seguridad. Para esto, se utilizó el modelo de MVC (modelo, vista y controlador) y cada uno de ellos tiene paquetes divididos por funcionalidad.

Service: permite administrar los servicios registrados del cliente. Muestra datos como id, nombre del servicio, versión y la fecha de expiración.

Policy: permite administrar los datos de las políticas de seguridad mostrando los datos como el id, la fecha de expiración, el nombre de la política, el nombre del jar y su versión.

Agent: permite administrar los agentes que está registrados para cada cliente permitiendo eliminar o editar los datos como el id, IP, MAC y estado.

Esto al final se envía al componente *WSGServer* a través del paquete *Service*, explicado anteriormente.

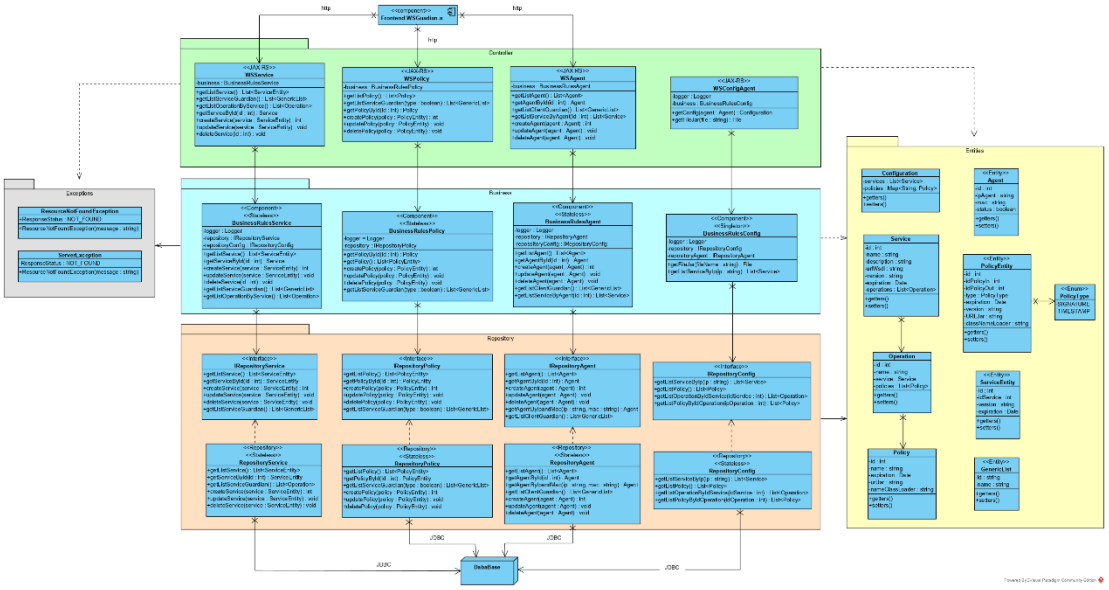

Figura 8: Diagrama de clases _administración del WSG-Client´s Agent

En la Figura 8: *Diagrama de clases _administración del WSG-Client´s Agent* (*Anexo 4 - Diagrama de clases _administración del WSG-Client´s Agent.pdf), se* muestra el diagrama de clases de la administración del *WSG-Client´s Agent, el* cual está desplegado dentro del nodo *WSG-Agent Server*. En este diagrama de clases se contienen 5 paquetes: *Controller*, *Business*, *Repository*, *Entities* y *Exception*.

El paquete *Controller* hace la conexión con el *frontend* de la administración vía http, también se encarga de mandar los datos obtenidos del *frontend* a *business*. Este también hace la conexión con el paquete de integración del WSClientAgent.jar del nodo del cliente por vía https.

El paquete *Business* contiene clases de lógica de negocio para la administración de *WSG-Client's Agent*.

Una vez que se obtiene datos desde la base de datos (*MySQL*) desde el paquete *Repository*, es guardado en el paquete *Entities* en sus respectivas clases.

Se tiene paquete llamado *Exceptions* cual contiene clases de excepciones para buen manejo de los errores.

2.4. Vista del proceso

Configuración

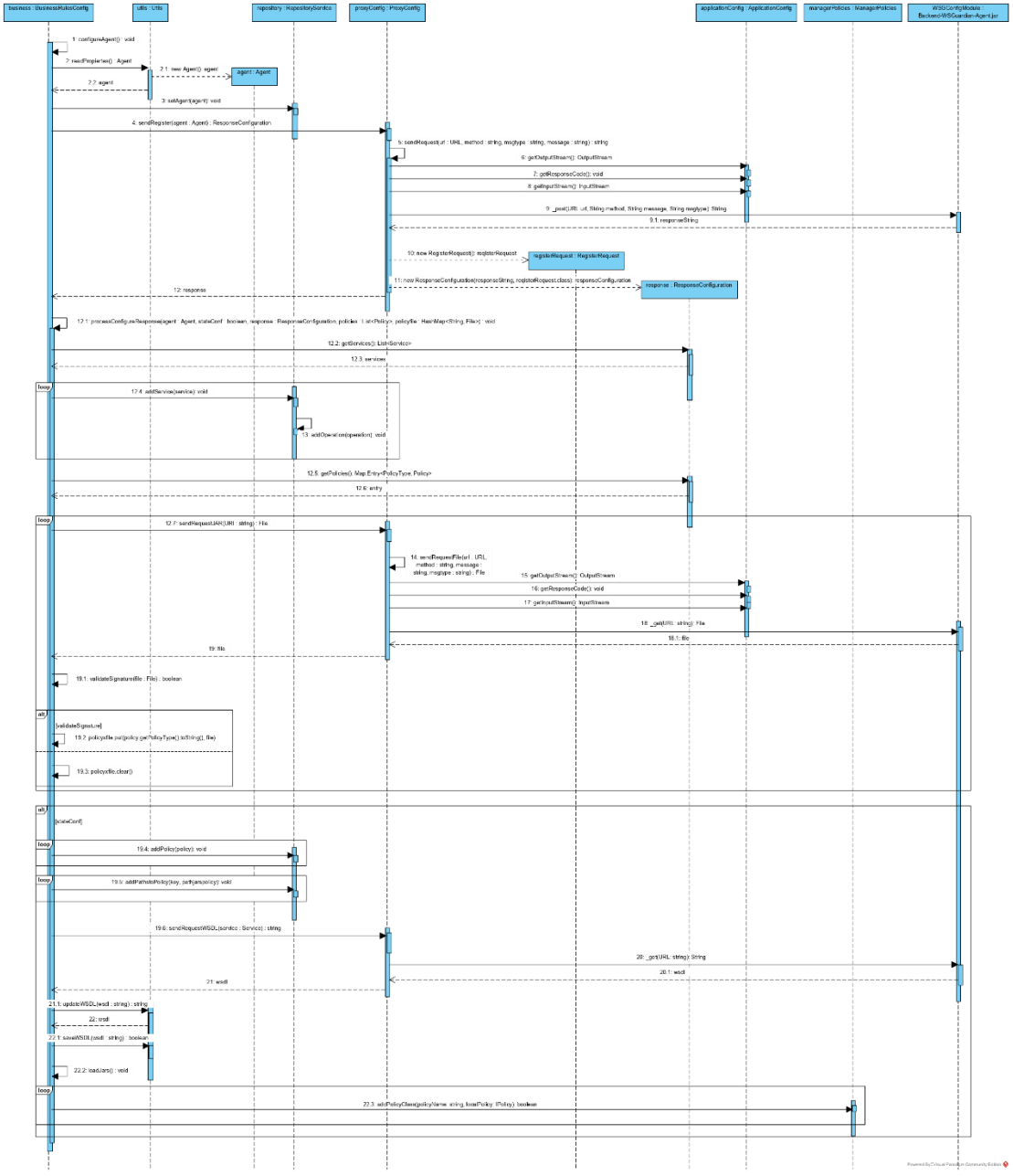

Figura 9: Diagrama de secuencia_Configuración

En la Figura 9: *Diagrama de secuencia _Configuración* (*Anexo 5 - Diagrama de secuencia _Configuración.pdf) se* muestra el diagrama de secuencia de la configuración inicial. El *Main* se

comunica con la clase *BussinesRulesConfig* donde realiza la lógica del negocio, aquí inicializa los datos del sistema y realiza las siguientes acciones:

1. Leer propiedades
    i. Desde la clase *Utils*, se lee la propiedad inicial de la aplicación para crear la clase *Agent* que contiene información básica, tanto de *WS-Guardian* como del cliente donde se está corriendo la aplicación.
    ii. El *Agent* creado es guardado en la clase *Repository*.
2. Conexión con el servidor *WS-Guardian*
    i. Una vez que se obtuvo la información necesaria para la conexión con el servidor *WS-Guardian*, a través del *ProxyConfig* hace la conexión https.
    ii. Cuando *WS-Guardian* responde a la aplicación, se crea la clase *ResponceConfiguration* que contiene la información de servicios y políticas necesarias del cliente.
3. Obtener los servicios (ciclo)
    i. A través del *ProxyConfig*, se manda la solicitud del WSDL del servicio (información obtenida en el paso anterior) al *WS-Guardian.*
    ii. Las direcciones http y https del WSDL obtenido son actualizadas para que el consumidor que obtiene este documento envíe las solicitudes a *WSG-Client's Agent*.
    iii. El WSDL obtenido anteriormente se guarda en *RepositoryServices*, junto con las operaciones y políticas de seguridad necesarias mencionado en WSDL.
4. Obtener las políticas (ciclo)
    i. Se hace uso del ProxyConfig, para enviarle la URL como parámetro, se dirige al servidor de *WS-Guardian* para obtener el archivo JAR, que contiene las clases y archivos necesarios para aplicar la política.
    ii. Una vez que se obtienen los JAR, se verifican las firmas para asegurar que estos códigos no fueron modificados en el transcurso del viaje.
    iii. Lo anterior es guardado con la clase *RepositoryService*, y se realiza de manera iterativa hasta tener todos los archivos de las políticas.

Aplicar/Procesar Política

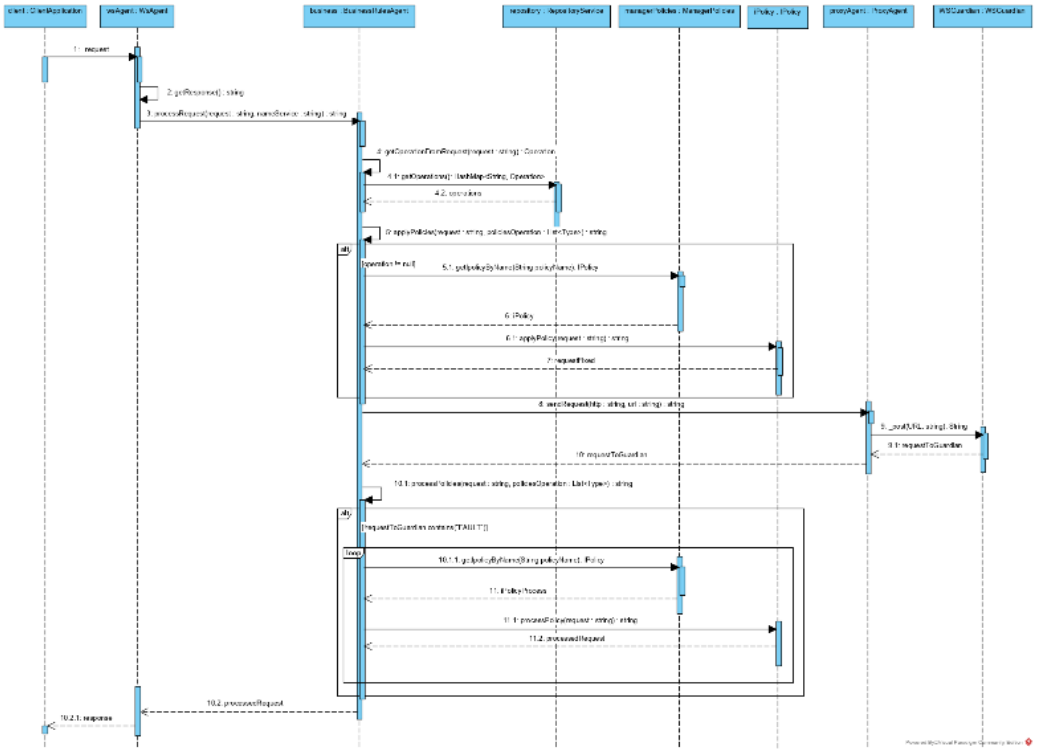

Figura 10: Diagrama de secuencia _Aplicar/Procar política

En la *Figura 10: Diagrama de secuencia _Aplicar/Procar política* (*Anexo 6 - Diagrama de secuencia _Aplicar/Procar política.pdf*) *se* muestra el diagrama de secuencia de la aplicación y procesamiento de la(s) política(s) de seguridad del servicio del cliente.

1. Obtener petición del cliente
    i. Por la clase *WSAgent* se obtiene la solicitud SOAP enviada por del cliente.
    ii. Esto es procesado con la clase *BusinessRulesAgent* para obtener el nombre del servicio y de la operación.
2. Obtener Operación
    i. Con los datos obtenidos anteriormente, se recupera la *Operation* con el recurso *RepositoryService*. Si retorna nulo, se manda una excepción. Si no, sigue al siguiente paso.
3. Aplicar política de seguridad (ciclo)
    i. Una vez que se obtiene *Operation*, se obtiene la lista de políticas.
    ii. Se hace la llamada de *ManagerPolicy* para obtener la referencia a la(s) política(s) correspondientes.
    iii. Se llama el método *ApplyPolicy* (en cada referencia obtenida anteriormente) para aplicar la(s) política(s) de seguridad obtenida(s) en el paso anterior.

4. Enviar solicitud a *WS-Guardian*
    i. Se envía la petición usando el método *SendRequest* de la clase *ProxyAgent*.
5. Obtener respuesta del *WS-Guardian* de la clase *ProxyAgent*.
    i. El *ProxyAgent* obtiene la respuesta del *WS-Guardian* luego de enviar la solicitud, la cual se pasa a la clase *BusinessRulesAgent.*
6. Procesar política (ciclo)
    i. Se hace la llamada de *ManagerPolicy* para obtener la referencia de la política que corresponde según la configuración.
    ii. Se llama el método *ProcessPolicy* para procesar la política de seguridad obtenida anteriormente.
7. Enviar al cliente
    i. Se responde al cliente con el mensaje devuelto por el método invocado anteriormente, ya se ha validado su aseguramiento y no contiene etiquetas de seguridad.

Actualización

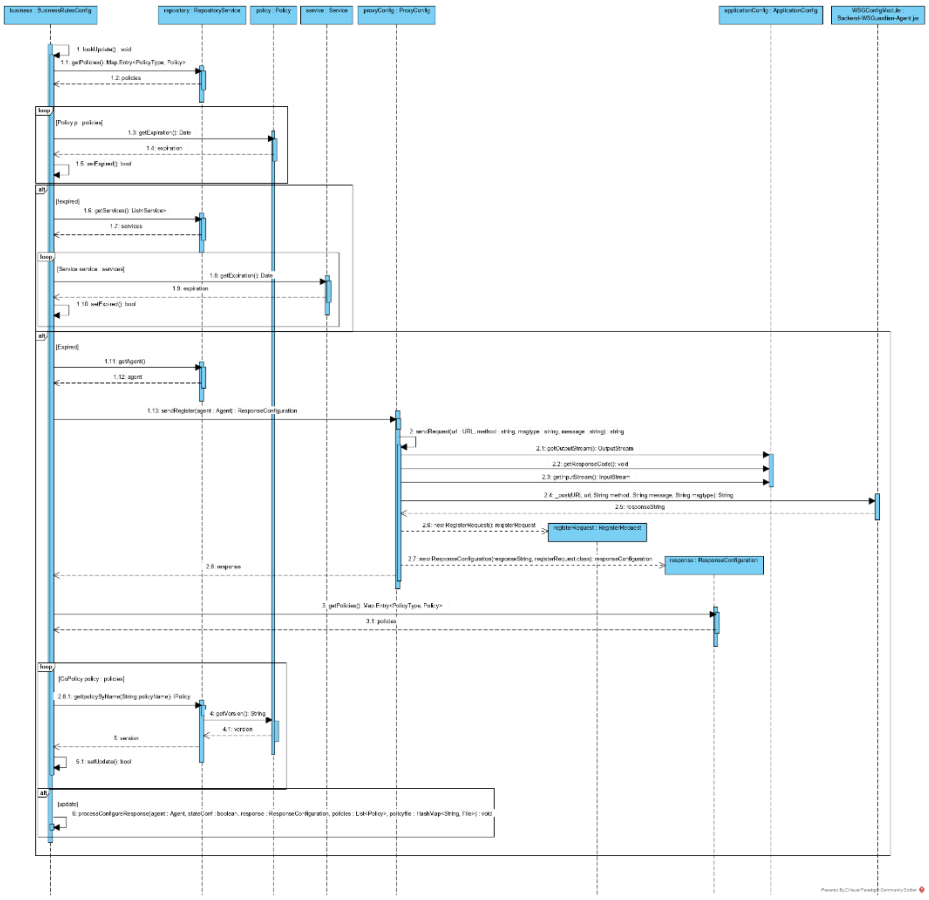

Figura 11: Diagrama de secuencia _Actualización

En el *Figura 11: Diagrama de secuencia _Actualización* (*Anexo 7 - Diagrama de secuencia _Actualización.pdf*) se muestra el diagrama de secuencia para la actualización de las políticas de seguridad. Estas tienen tiempo de expiración o necesitan de una actualización de los servicios relacionados a dichas políticas cuando se vence el tiempo.

1. Verificación de las políticas
    i. Se obtiene la lista de políticas almacenadas dentro del repositorio.
    ii. Se verifica política por política si su tiempo de expiración no se ha completado.
2. Verificación de los servicios
    i. Si lo anterior es falso, se verifica entonces el versiona miento de los servicios.
    ii. Se obtiene la lista de servicios almacenados dentro del repositorio.
    iii. Se verifica servicio por servicio si su tiempo de expiración no se ha completado.
    iv. Si el tiempo se ha vencido, se llama al *ProxyConfig* por el método *sendRegister*, para que obtenga las nuevas políticas desde el servidor *WS-Guardian*.
    v. Se verifican las versiones de cada política para conocer si se necesita nuevos JAR.
3. Actualización
    i. En el caso de que se necesite actualización de la(s) política(s), se llama al *ProcessConfigureResponse* desde *BusinessRulesConfig* para la descarga de nuevos JAR.
    ii. En caso de no necesitar actualización de la(s) política(s), se termina el proceso son afectar la configuración de *WSG-Client's Agent* ni los componentes de las políticas.

VI- DESARROLLO DE LA SOLUCIÓN

1. Ejecución

1.1. Metodología

Durante el desarrollo se empleó la metodología SCRUM con el fin monitorear procesos y avances, ajustar la estimación de las tareas programadas, y mitigar los eventos externos no deseados que pudieran afectar la ejecución del proyecto.

El ciclo de desarrollo inició con la definición del *product backlog*, para luego realizar el *planning* de *sprint* con tareas específicas, asignando un estimado de tiempo y un responsable a cada una. Cada *sprint* tenía una duración de ocho (8) o quince (15) días. Al vencer este periodo de trabajo se realizaban reuniones de cierre para observar los resultados obtenidos, evaluar el avance, tomar decisiones de implementación y programar el siguiente *sprint* (ceremonia de *sprint planning*). Esto fue dirigido por el *Scrum Master*. Adicionalmente, se realizaron reuniones cada quince (15) días con el *product owner* para revisar los avances de la implementación, y recibir sugerencias y retroalimentaciones.

Para el control y programación de los *sprint* en el ciclo de desarrollo se utilizó la plataforma *Spira Team*, como se puede observar en la *Figura 12: Sprint 1*, debido a que esto fue requerido por la empresa *ITAC S.A.*

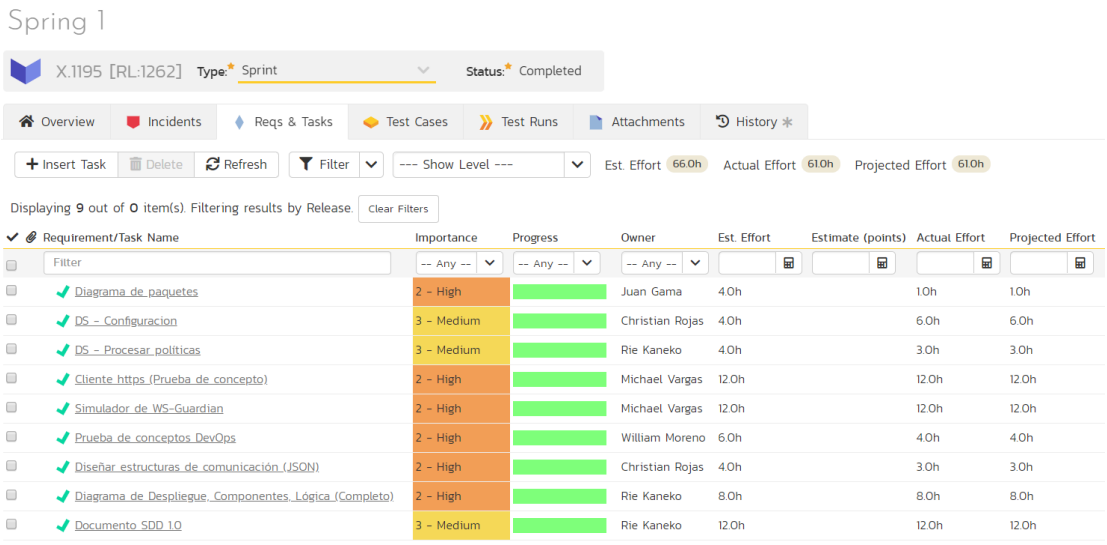

Figura 12: Sprint 1

En cada reunión se generaba una minuta con los aspectos tratados durante la junta, y así tener constancia de las interacciones. De la misma manera al finalizar una fase del proyecto se realizaron controles de riesgo y retroalimentación del trabajo cooperativo. Las minutas de las reuniones, los informes de controles de riesgo y retroalimentación de los trabajos se puede observar con mayor detalle en el informe de postmortem.

La asignación de tareas fue concorde a los roles que cada integrante del *scrum team* desempeñaba. El tiempo estimado para cada *sprint* y actividad estuvieron alineadas con el periodo de trabajo según la fase a la que correspondían. Cabe aclarar que en ocasiones se extendieron las duraciones asignadas por motivos de retraso de tareas, mala estimación de tiempos, y errores imprevistos de implementación. Una solución que se propuso para para mitigar estas contingencias fue sancionar a quien cometió la falta con una penitencia. Para conocer en detalle sobre los roles, calendarización de las fases, y reglas de trabajo interno del equipo referirse al documento SPMP e informe PostMortem.

1.2. Repositorio del Código

La implementación del código fuente fue manejada con la plataforma *GitHub*, donde se aloja un repositorio privado contenido de siete (7) ramas *(master, dev, feature/guardian, policy/signatura, policy/timestamp, update/agent, update/styles)*. En la *figura 13: Git Flow* se ilustra una parte del tráfico de código que hubo entre cada una de las ramas.

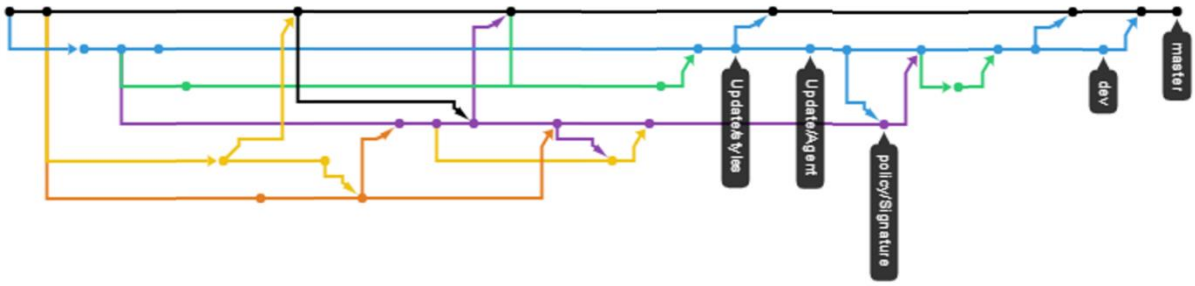

Figura 13: Git flow

1.3. Herramientas

Para la implementación de los componentes (JAR) encargados de aplicar y validar las políticas *Signature* y *TimeStamp*, fue necesario usar las librerías de Apache WSS4J. Por este motivo, se realizó la búsqueda de algoritmos que agregarán y verificarán el estándar WS-Security sobre los mensajes SOAP 1.1 usando WSS4J. Se encontró un repositorio contenido de un algoritmo encargado de aplicar la firma digital a los mensajes SOAP, usando las librerías WSS4J de Apache. Este fue adaptado a la solución del problema para generar el JAR de *Signature*, y se usó

como guía para generar el componente de *TimeStamp* ya que usan las mismas dependencias [16].

El producto fue desarrollado en una gran parte en Java. El *framework Spring* fue usado para implementar *WSG-Client´s Agent* y el *BackEnd* del Módulo de Administración de *este,* aprovechando su característica de portabilidad. Además, los lenguajes *TypeScript*, *HTML*, *CSS* y *JavaScript* se incluyeron en el *FrontEnd* donde se usó *Angular* como entorno de trabajo. La *figura 14: Frameworks y IDE de implementación,* muestra la distribución de los marcos e IDE de trabajo usados. Las herramientas utilizadas son explicadas con mayor detalle en el documento SPMP.

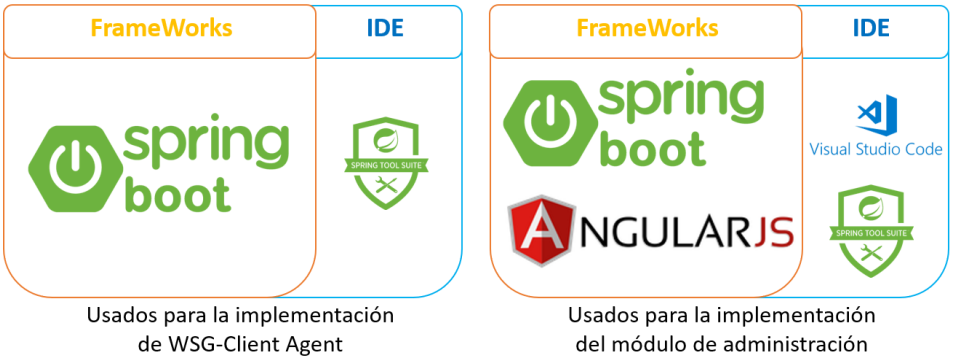

Figura 14: Frameworks y IDE de implementación

Como resumen del uso de las herramientas, la *figura 15: Lenguaje de Programación usados*, muestra los lenguajes de programación involucrados en el desarrollo, junto a los sus porcentajes de participación en la solución (código fuente).

Figura 15: Lenguajes de Programación usados

2. Alistamiento de Sistemas y Componentes Externos

2.1. Configuración de JAR de Política

La configuración y los componentes que *WSG-Client* requiere para cumplir sus funciones son descargados automáticamente del módulo de administración, por medio del protocolo https. *WSG-Client* toma los parámetros de entrada definidos en el archivo "*conf.properties*" para establecer la conexión y descargar sus recursos.

Una vez que *WSG-Client* descarga su configuración en formato JSON, este procede a guardar los datos de los servicios que va a proveer, e identificar las políticas que debe aplicar o validar. Con la información capturada anteriormente, se procede a descargar los componentes encargados de agregar o verificar la seguridad de los mensajes SOAP según corresponda. Es necesario aclarar que existe un componente por cada política, y cada componente está encapsulado en un archivo JAR contenido de: ejecutable en Java (JAR), archivo de configuración (*conf.properties*), librerías requeridas para la ejecución y certificados en el aco de que la política los requiere.

Antes de guardar un JAR proveniente del módulo de administración, pasa por el proceso de validación de firma digital. La llave y el hash de los archivos internos necesarios para la verificación están contenidos en la carpeta "*META-INF*" situada en el interior del archivo descargado. Si la firma es correcta se almacena el componente en el disco; pero en caso de no serlo, *WSG-Client* repite el proceso una vez más y si vuelve a fallar genera un registro en el log y se cierra.

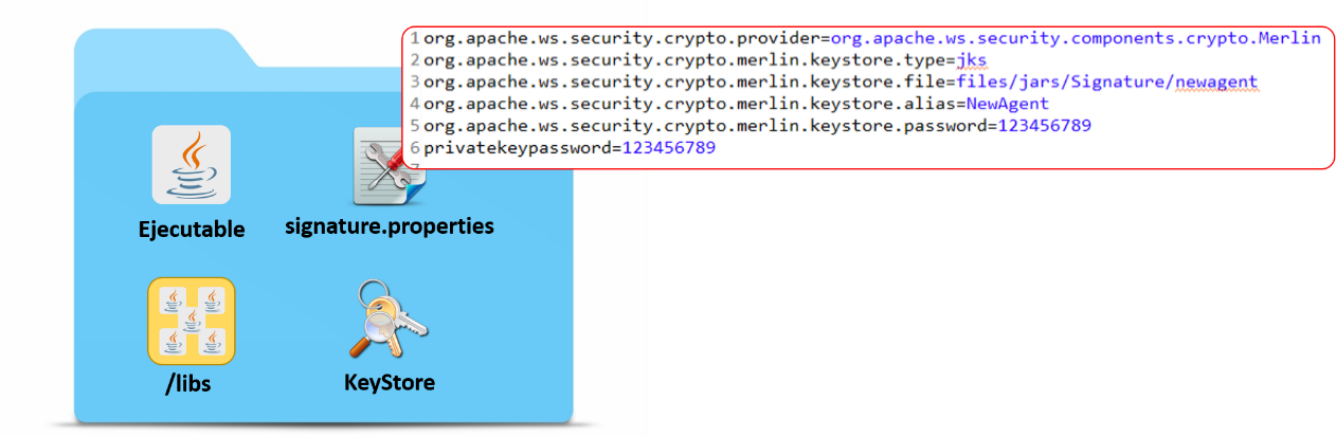

Figura 16: Encapsulamiento del componente de Firma Digital

La *figura 16: Encapsulamiento del componente de Firma Digital*, muestra los archivos pertenecientes al componente de firma digital (*SOAPSignature.jar*), y el detalle del archivo de configuración "*signature.properties*" cual esto son:

- **Ejecutable:** Código compilado con la funcionalidad agregar, validar y remover las etiquetas de *Signature* en mensajes SOAP.
- **KeyStore:** Certificado contenido de llaves públicas para aplicar y verificar la firma en mensajes SOAP.
- **signatura.properties:** Es un archivo de configuración que contiene parámetros de entrada para las funciones que realiza el ejecutable.

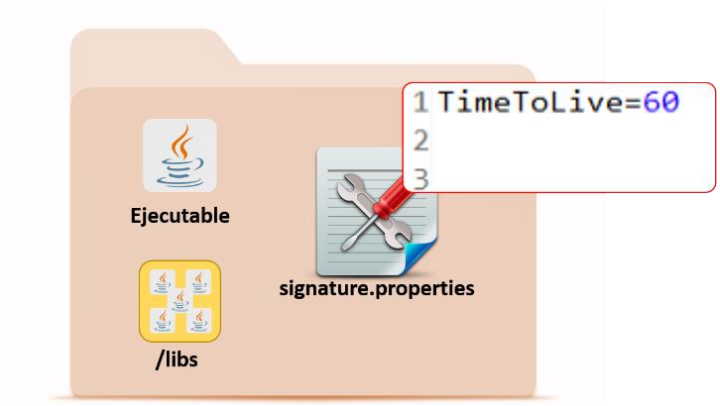

Figura 17: Encapsulamiento del componente de TimeStamp

La *figura 17: Encapsulamiento del componente de TimeStamp*, muestra los archivos pertenecientes al componente de TimeStamp (*SOAPTimeStamp.jar*), y el detalle del archivo de configuración "*timestamp.properties*". Estos son:

- **Ejecutable:** Código compilado con la funcionalidad agregar, validar y remover la etiqueta *TimeStamp* en mensajes SOAP.
- **timestamp.properties:** Es un archivo de configuración que contiene parámetros de entrada para la aplicación del *TimeStamp*.
- **/lib:** directorio contenido con las librerías usadas por el ejecutable.

Para ver más el detalle de los archivos pertenecientes a cada componente (*Signature* y *TimeStamp*), y detalles de los archivos de configuración se encuentra en el documento Manual de Usuario.

2.2. Datos previos en WS-Guardian

Para poder agregar un nuevo *WSG-Client* y su configuración en el módulo de administración, es necesario que previamente el administrador de *WS-Guardian* agregue el cliente consumidor de los servicios web, registrando: la IP desde la que envía las peticiones, los servicios que puede consumir, y las políticas de entrada y salida de cada operación en los WS.

Luego de que un cliente está registrado en el administrador de *WS-Guardian*, se puede incluir el nuevo *WSG-Client* y relacionar los servicios a los cuales agrega su configuración a través del módulo de administración.

3. Configuración y Registro del WSG-Client

3.1. Parámetro de entrada para el WSG-Client

```
1  ipGuardian=https://192.168.1.12
2  portGuardian=8080
3  urlService=http://itac.com.co:6080
4  #ipAgent=192.168.42.42
5  #macagent=00:1B:44:11:3A:B7
6  #portAgent=3124
7  log=true
8
9  keyStorePath=./client.keystore
10 truststorePath=./client.truststore
11 keyStorePW=javeriana
12 truststorePW=javeriana
13 keyPass=javeriana
14 keyStoreType=jks
15 trustAllCertificate=true
16 keyManagerAlgorithm=SunX509
```

Figura 18: Archivo conf.properties de WSG-Client

Los parámetros de entrada que tiene el *WSG-Client* son los definidos en el archivo "*conf.properties*". En la *Figura 18: Archivo conf.properties de WSG-Client* se muestra un ejemplo de configuración, donde se encierra con rojo la información usada para comunicarse con el módulo de administración y consumir los servicios; con verde se señalan los datos propios de *WSG-Client* usados para identificarse y descargar la configuración; con azul se indica el elemento que activa o desactiva la generación de log en tiempo real; y con naranja se señalan entradas requeridas para crear la conexión https.

3.2. Registro de WSG-Client y servicios de carga de políticas

Cómo se ha mencionado en la *sección Alistamiento de Sistemas y Componentes Externos*, el cliente consumidor de los servicios debe estar registrado previamente en *WS-Guardian* para poder mapearlo en el módulo de administración y asociarlo a *WSG-Clients* registrados.

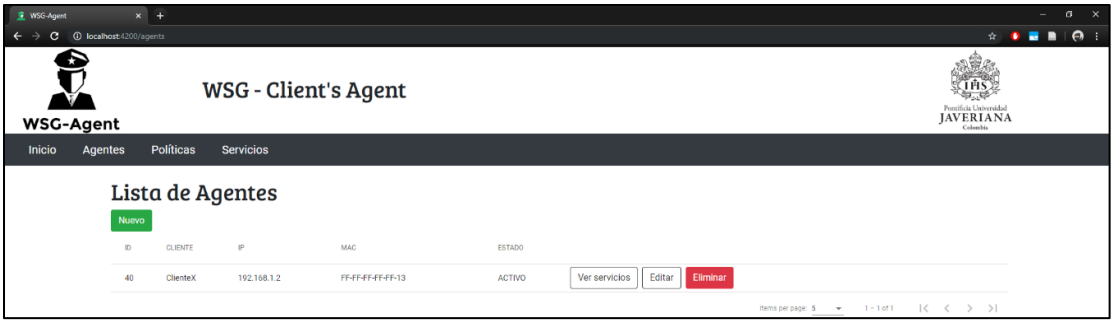

Figura 19: Lista de WSG-Clients Agent registrados

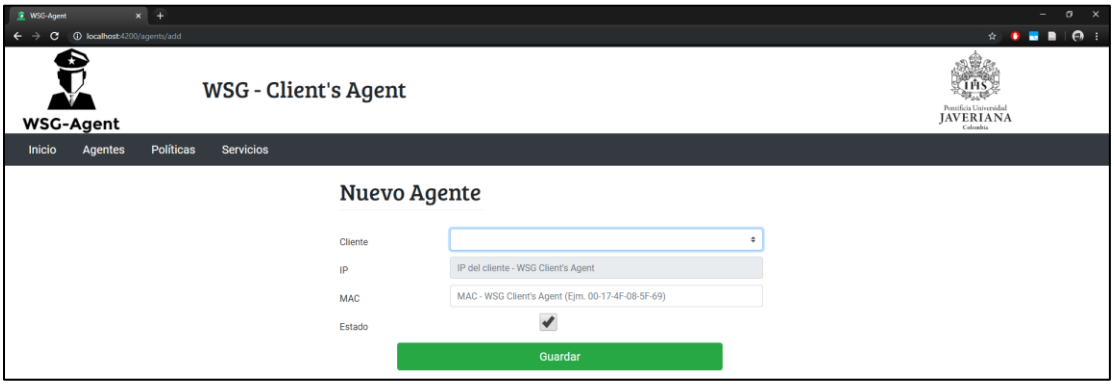

Figura 20: Formulario para registrar un WSG-Client

Para registrar un *WSG-Client* es necesario acceder al módulo de administración, luego acceder a la pestaña *Agentes* dónde se listan los agentes del sistema cómo se ve en la *Figura 19: Lista de WSG-Client Agent registrados* Seguidamente, es necesario dar click en el botón *Nuevo* para desplegar el formulario mostrado en la *Figura 20: Formulario para registrar un WSG-Client*, el cual solicita: seleccionar un cliente registrado en *WS-Guardian*, ingresar la dirección física (MAC) perteneciente al agente, y asignar un estado al nuevo *WSG-Client*.

3.3. Registro de servicios

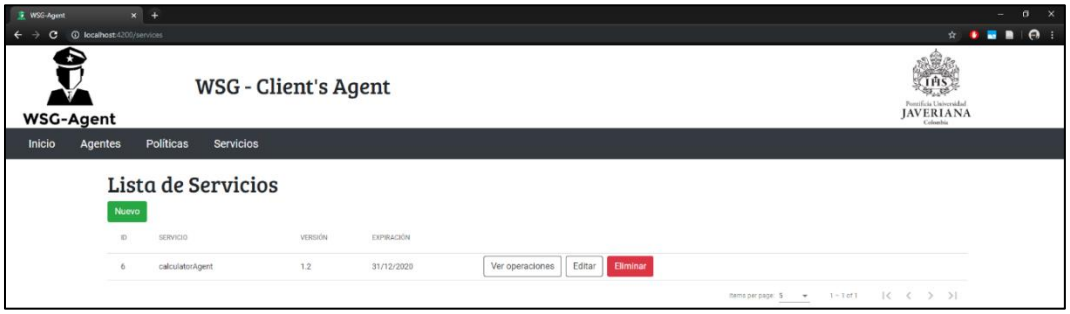

Figura 21: Lista de servicios registrados

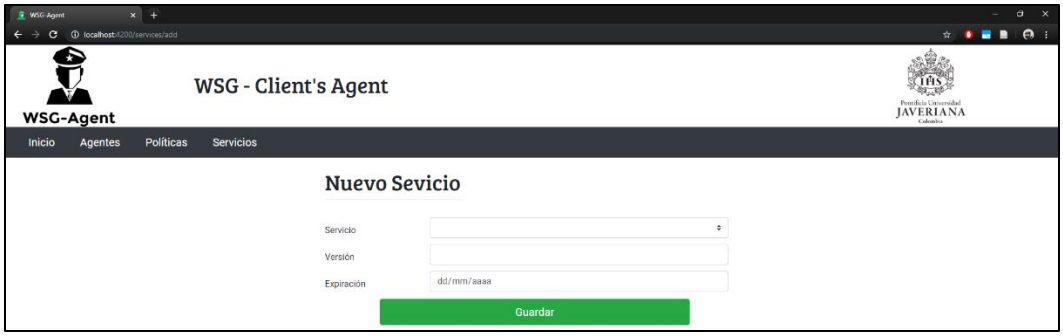

Figura 22: Formulario para registrar un nuevo servicio

El registro de los servicios se realiza mediante el módulo de administración en la pestaña *Servicios,* donde se muestran los servicios actuales del sistema cómo se ve en la *Figura 21: Lista de servicios registrados*. Seguidamente, es necesario dar click en el botón *Nuevo* para desplegar el formulario mostrado en la *Figura 22: Formulario para registrar un nuevo servicio*. En este formulario es necesario seleccionar un servicio existente en *WS-Guardian*, digitar una versión, y asignar una fecha de expiración.

3.4. Cargas de políticas

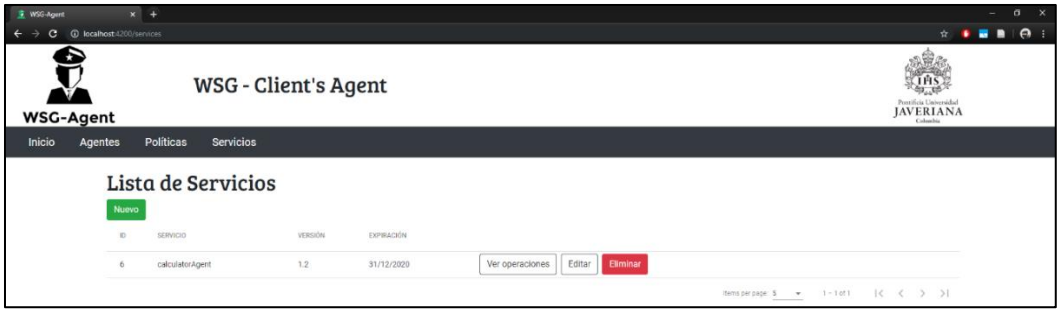

Figura 23: Lista de servicios registrados

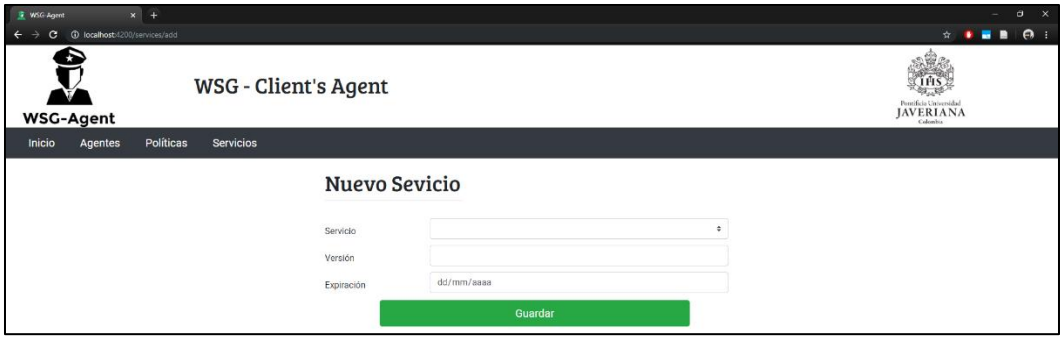

Figura 24: Formulario para registrar un nuevo servicio

La carga de los JAR correspondientes a las políticas también se hace a través del módulo de administración en la pestaña *Políticas*. Esta muestra el listado de todas las versiones de los

JAR cargados en el sistema cómo se ve en la *Figura 23: Lista de servicios registrados*. Al dar click en el botón *Nuevo* se despliega un formulario idéntico al mostrado en la *Figura 24: Formulario para registrar un nuevo servicio*, el cual solicita seleccionar el tipo de política, señalar si corresponde a la entrada o salidas, asignar una fecha de expiración, nombre de la clase que invoca al cargar el JAR y la versión de la política que se está cargando. Adicionalmente exige la carga de un archivo con extensión *'.jar'* correspondiente al código de ejecución que aplica o valida la política señalada.

3.5. Ejecución y Funcionamiento

Figura 25: Formulario para registrar un nuevo servicio

Para la ejecución de *WSG-Client* es necesario contar con los archivos ilustrados en la *Figura 25: Formulario para registrar un nuevo servicio* (*client.keystore*, *client.truststore*, *conf.properties* y *WSAgent-0.0.1.jar*), para luego ejecutar el archivo con extensión *'.jar'* y poner en marcha el software. Cuando este inicia automáticamente lee el contenido del archivo "*conf.properties*" e inicia con la descarga de sus recursos.

3.6. Trazabilidad

Figura 26: Logs generados por WSG-Client

WSG-Client genera logs en tiempo real de la ejecución, siempre y cuando el parámetro de entrada *log* en el archivo "*conf.properties*" tenga asignado el valor *true*. Cada registro que se agrega en el archivo de auditoría representa un evento ocurrido en la ejecución de la aplicación, ya sea para informar el resultado de un proceso o notificar errores. En la *Figura 26: Logs generados por WSG-Client* se muestran logs generados por *WSG-Client* en tiempo de ejecución notificando el resultado de eventos ocurridos con el fin de poder realizar la trazabilidad del producto. Esto es explicado con mayor detalle en el Manual de Usuario.

VII- RESULTADOS

1. Producto Final

1.1. Política

Como se acordó en los objetivos específicos, se implementaron dos políticas de seguridad (JAR) que son: *Signature* y *Timestamp*. Estos son componentes encargados en aplicar y procesar (validar y retirar) etiquetas correspondientes a los métodos de aseguramiento mencionados, sobre mensajes SOAP 1.1.

1.2. Módulo de Administración de WSG-Client's Agent

Este módulo el poseedor de *WS-Guardian*, podrá administrar diferentes características que *WSG-Client's Agen*t requiere para su funcionamiento como lo son:

- **Agentes:** Estos han de ser previamente registrados, para que se les brinde la información del cliente para el cual están sirviendo. Toda esta comunicación se hace a través del protocolo HTTPS.
- **Servicios:** Donde se enriquece la información almacenada en *WS-Guadian* que es de utilidad para *WSG-Client's Agent*.
- **Políticas:** Con información sobre estas y el JAR firmado que la aplica y procesa.

1.2.1. Backend

Se encarga de proveer una API REST a la que se conecta *WSG-Client´s Agent* y la *Single Page Application,* además es el encargado de conectarse con la base de datos *MySQL* con el fin de realizar consultas en la información almacenada.

1.2.2. Frontend

Se encarga de conectarse al *Backend* y mostrar una interfaz gráfica para el cliente, donde pueda visualizar información de los recursos mencionados anteriormente e interaccionar con estos.

1.3. WSG-Client´s Agent

Es el módulo que está en contacto con la aplicación cliente, cuenta con un archivo de configuración que contiene parámetros de entrada para la su ejecución, una interfaz por donde hace la recepción de peticiones del cliente, para posteriormente aplicar políticas de seguri-

dad, en base a la configuración previa que descarga desde el módulo de administración explicado anteriormente. Adicionalmente, realiza una actualización periódicamente de las políticas (JAR) en base a la información que le brinde el "Modulo de Administración de *WSG-Client's Agent".*

1.4. Entregables

El equipo de desarrollo de *WSG-Client's Agent* entrega cuatro archivos con extensión .jar que son:

- *WSG-Client's Agent*
- Política de *Signature*
- Política *Timestamp*
- *Backend* del módulo de administración de WSG-Client´s Agent
- Script Base de Datos
- *Single Page Application* (*Frontend*) del módulo de administración de *WSG-Client´s Agent*

Junto a estos archivos se entrega un script de Base de Datos donde se crean las tabals requeridas por módulo de administración de *WSG-Client´s Agent* para persistir informacion necesaria para el funcionamiento de los agentes, un *Javadoc* con la documentación del código, un manual de usuario donde se le presenta al usuario las indicaciones necesarias para la configuración, ejecución y uso de los módulos, y por último un informe de pruebas que da constancia de que las historias de usuario y requerimientos expuestos anteriormente han sido satisfechas.

2. Casos de Pruebas

Durante el desarrollo del proyecto fueron realizadas diferentes pruebas. Las primeras pruebas que se realizaron fueron las pruebas de concepto. Estas pruebas se ejecutaron a lo largo de *sprints* que implicaban probar algún concepto clave para el desarrollo, como la funcionalidad de la firma digital y timestamp, las cuales implicaban probar el funcionamiento de una pequeña parte del desarrollo. Desde el *sprint 9* hasta al *sprint 12* se realizó el plan que incluía el diseño, ejecución y documentación de pruebas que se harían para todo el producto final. Se estableció que se harían pruebas manuales, se crearon los casos de prueba en la herramienta *SpiraTeam*, cada uno de estos casos fueron asociados a historias de usuario establecidas, las cuales se pueden observar en el *Anexo 1 – Historias de Usuario*.

Test #	Name
0	Root
TC14909	Actualizar las politicas exitosa
TC14910	Actualizar las politicas fallida
TC14906	Almacenar IP y mac de WSG-Client's Agent en WS-Guardian exitoso
TC14908	Almacenar IP y mac de WSG-Client's Agent en WS-Guardian fallido
TC14907	Almacenar IP y mac de WSG-Client's Agent en WS-Guardian fallido por conexión
TC14857	Aplicación de políticas en el WSG-Client´s Agent exitosa
TC14858	Aplicación de políticas en el WSG-Client´s Agent fallido
TC15106	Cargar Información y Jar de politica a WS-Guardian
TC14781	Cargar Jars de manera correcta
TC14779	Descarga de jar URL no responde
TC14778	Descarga y validación de JAR exitosa
TC15057	Descarga y validación de JAR invalida
TC14179	Despliegue interfaz para registro
TC14911	Guardar IP y MAC del cliente exitosa
TC14912	Guardar IP y MAC del cliente fallido - IP
TC14913	Guardar IP y MAC del cliente fallido - MAC
TC14859	Guardar Jars en Disco
TC14782	Guardar servicios en repositorio
TC14314	Listar servicios de un Agente
TC14860	Rechazo de solicitud de petición desde externo exitosa
TC14180	Registrar Agente exitosamente
TC15058	Registrar servicios en WS-Guardian
TC14284	Registro agente con IP erronea
TC14285	Registro Agente con MAC erronea
TC14616	Registro y descarga de datos exitoso
TC14617	Registro y descarga de datos fallido ip guardian no responde
TC14618	Registro y descarga de datos incorrecto

Figura 27: Casos de pruebas

En la *figura 27: Casos de prueba*, puede ver los 27 casos de prueba que fueron creados. Aquí se detalla el nombre de cada uno de los casos de prueba que se ejecutó. estos cubren las historias de usuario (explicado en las secciones anteriores). En el informe de pruebas se puede ver información y resultados cada caso de prueba ejecutado (descripción, entradas, salidas y resultados).

2.1. Casos de Pruebas Principales - Funcionales

A continuación, se muestran los resultados de los casos de pruebas principales a través del informe generado por la plataforma *Spira Team*. Los detalles de estos casos son explicados en el informe de pruebas.

Módulo de Administración de WSG-Client's Agent

- **Cargar agentes:** En este caso de prueba se verificó que el módulo pudiera registrar un agente en el módulo de *WS-Guardian*. En la *figura 28: Casos de prueba – Cargar agente* se pueden observar las entradas, salidas y adicionalmente el estatus de la prueba. Estos se ejecutaron con éxito cada paso del caso de prueba.

Test TC:14180-Registrar Agente exitosamente

Se van a ingresar campos validos para el correcto registro del Agente

Execution Status:	Passed	**Priority:**	4 - Low
Type:	Functional	**Status:**	Draft
Author:	Juan Gama	**Creation Date:**	17-Sep-2019
Owner:		**Last Execution:**	06-Nov-2019
Automation Engine:		**Last Updated:**	06-Nov-2019
Component(s):			
Automatizado:	N		

Step	Description	Expected Result	Sample Data	Last Status
1	Seleccionar nombre del cliente, ingresar MAC del agente, adicionalemente dejar el estado en check		Cliente seleccionado: Agente maquina1 Davivienda	Passed

			MAC: 34-F6-4B-6B-9B-B7	
2	Oprimir boton de registro	Agente guardado con Exito!		Passed

Figura 28: Casos de prueba – Cargar agente

- **Cargar Políticas:** En este caso de prueba se verificó que el módulo de administración desarrollado para *WS-Guardian* permita cargar los JAR que aplican las políticas, y asociarlos a la política adecuada. En la *figura 29: Casos de prueba – Cargar política* se puede ver los pasos y su descripción.

Test TC:15106-Cargar Información y Jar de politica a WS-Guardian

Para este escenario se probara que el modulo de WS-GUARDIAN pueda guardar archivos y relacionarlos a las políticas

Execution Status:	Passed	**Priority:**	
Type:	Functional	**Status:**	Draft
Author:	Juan Gama	**Creation Date:**	06-Nov-2019
Owner:		**Last Execution:**	06-Nov-2019
Automation Engine:		**Last Updated:**	10-Nov-2019
Component(s):			
Automatizado:	N		

Step	Description	Expected Result	Sample Data	Last Status
1	Ingresar a la url de WS-GUARDIAN https://localhost:4200/policies	Interfaz que muestra la lista de politicas que estan cargadas en WS-GUARDIAN	N/A	Passed
2	Dar click en "Nuevo"	Interfaz que permita cargar archivo, relacionar politicas de entra y salida, y nombre del la con la que se va a llamar el Jar		Passed
3	Diligenciar los datos	n/a	Tipo Política: Signature Política Entrada: Firma de agente Política Salida: Validación firma agente Expiración: 11/11/2019 Nombre Clase a Cargar: SOAPSignature Versión: 2.0 Archivo Jar	Passed
4	Click en Guardar	Política guardad con exito		Passed

Figura 29: Casos de prueba – Cargar política

WSG-Client's Agent

- **Cargar configuración:** En el siguiente caso de prueba se puede ver el caso de prueba asociado a la descarga de datos que hace el *WSG-Client's Agent* a *WS-Guardian*. En la *figura 30: Casos de prueba – Cargar configuración* se puede observar el informe generado por la plataforma *Spira Team*.

Test TC:14616-Registro y descarga de datos exitoso

En este caso de prueba se probara que el WS-Agente descargará los datos de WS-Guardian mediante un servicio POST, este es el escenario ideal para el cual no se tendrá ningun inconveniente.

Execution Status:	Passed	**Priority:**	
Type:	Functional	**Status:**	Draft
Author:	Juan Gama	**Creation Date:**	06-Oct-2019
Owner:		**Last Execution:**	06-Nov-2019
Automation Engine:		**Last Updated:**	10-Nov-2019
Component(s):			
Automatizado:	N		

Step	Description	Expected Result	Sample Data	Last Status
1	Obtener la IP y puerto de WS-Guardian mediante la clase UTILS	Clase agente con los atributos,ipGuardian; portGuardian; ipAgent; portAgent; MacAgent; cargados	N/A	Passed
2	Envio de petición POST desde el WS-Agent para enrolar con WS-Guardian	Se recibe un JSON que contiene los servicios, operaciones y politicas.	se envia un request POST mediante HTTPS con la IP y mac del Agente, para enrolar	Passed

3	Mapeo a la clase ResponseConfiguration	Clase ResponseConfiguration con la información mapeada proveniente de la respuesta	Mediante el uso de la libreria Gson de Google, se envia string con la respuesta y el nombre de la clase a la que se va a mapear(ResponseConfiguration)	Passed
4	Verificación de que la clase ResponseConfiguration tenga los datos.	politicas y servicios con su respectiva informacion	pedir mapa de politicas y lista de servicios.	Passed

Figura 30: Casos de prueba – Cargar configuración

- **Aplicar políticas:** En este caso de prueba se observan las peticiones y logs asociados al aplicar las políticas de seguridad que fueron descargadas de *WS-Guardian*. En la *figura 31: Casos de prueba – Aplicar políticas* se pueden ver los pasos ejecutados y la descripción de cada uno.

Test TC:14857-Aplicación de políticas en el WSG-Client´s Agent exitosa

Cargar diferentes políticas de seguridad haciendo uso del mismo método.

Execution Status:	Passed	**Priority:**	
Type:	Functional	**Status:**	Draft
Author:	Michael Vargas	**Creation Date:**	16-Oct-2019
Owner:		**Last Execution:**	09-Nov-2019
Automation Engine:		**Last Updated:**	10-Nov-2019
Component(s):			
Automatizado:	N		

Step	Description	Expected Result	Sample Data	Last Status
1	Buscar servicio de la petición	Servicios	Petición	Passed
2	Buscar Política políticas a aplicar	Políticas	Petición y nombre/operación de servicio.	Passed
3	Aplicar cada política asociada a la petición	Petición política aplicada	Petición	Passed
4	Enviar petición a WS-Guardian con las políticas	Boolean	Petición modificada	Passed

Figura 31: Casos de prueba – Aplicar políticas

2.2. Casos de Pruebas Principales – No Funcionales

A continuación, los casos de prueba más relevantes del sistema. Los detalles de estos casos son explicados con detalle el Informe de Pruebas - No Funcionales.

- **(NF1)** Como usuario, quiero ejecutar el *WSG-Client's Agent* en plataformas con sistema operativo *Windows* y *Linux*, para que pueda ser usado por varios clientes.

Para el cumplimiento del anterior caso de prueba, el *WSG-Client's Agent* fue diseñado para ejecutarse en múltiples plataformas. Esto con el fin de no quedar restringido a un solo sistema operativo. Al ejecutarse en la máquina virtual de java se asegura la portabilidad del sistema. Debido a que los sistemas operativos más comunes son *Windows* y *Linux* se validó el *WSG-Client's Agent* en estos ambientes [17].

- **(NF3)** Como desarrollador, quiero que la arquitectura sea Cliente / Servidor

La arquitectura esta soportada por comunicaciones cliente/servidor evidenciadas tanto en paso de mensajes entre WSG-*Client's Agent* y el módulo de administrador de *WSG-Client's Agen*t, como entre WSG-*Client's Agent* y *WS-Guardian*. Las comunicaciones mencionadas anteriormente se basan en una comunicación request-response [17].

- **(NF6)** Como usuario, quiero que *WSG-Client´s Agent* no se demore más de 1 segundo aplicando una política de seguridad, para tener una baja latencia.

En el caso de prueba anterior, al ser un *WSG-Client's Agent el* puente entre el cliente y el proveedor del servicio, la comunicación requiere un tiempo extra ya que WSG-*Client's Agent* aplica seguridad a la petición y procesa la respuesta para validar la misma. Por esta razón se garantiza un tiempo de latencia entre 0.4 y 1 un segundo. Este requerimiento se cumplió satisfactoriamente incluso en equipos de trabajo personal (bajo desempeño). Con esto concluimos que, en equipos de alto desempeño como servidores, tomará menos tiempo en el proceso completo.

3. Análisis del resultado

Basados en los resultados de las pruebas realizadas para verificar el cumplimiento de lo propuesto, se encontró que el producto final y el diseño propuesto para la solución fueron exitosos, debido a que a través de los resultados de los casos de prueba se muestra el cumplimiento de las historias de usuario y objetivos específicos establecidos.

El producto permite descargar un archivo que proviene del módulo de administración a través de REST/HTTPS. Una vez obtuvo la información de configuración, este permitió cargar el componente de la política de seguridad, cargando del archivo el código a ser ejecutado (JAR) y los datos de los servicios SOAP expuestos con diferentes políticas, de esta manera al momento de llegar una petición al *WSG-Client's Agent* este tuvo la capacidad de aplicar la política.

El producto *WSG-Client's Agent*, al llamar los métodos genéricos que deben estar contenidos en los archivos JAR, permite a cualquier política de seguridad que se desee agregar a un mensaje SOAP, se pueda agregar a través de una configuración y un archivo con la implementación

de la política. Como se ha establecido en los objetivos específicos, el producto fue implementado únicamente con dos políticas de seguridad: *signatura* y *timestamp*. Al observar los resultados de los casos de prueba se muestra que es posible implementar otras políticas de seguridad que se encierren en el marco de las librerías WSS4J de Apache.

El producto fue probado en las plataformas *Windows* y *Linux*, aprobando los procesos de pruebas en ambos sistemas operativos, aplicados tanto a requerimientos funcionales como no funcionales.

VIII- CONCLUSIONES

1. Análisis de Impacto del Proyecto

La propuesta del proyecto tuvo un gran impacto en el ámbito de los servicios web *SOAP*, permitiendo la automatización al consumir servicios web protegidos por políticas de seguridad del estándar de *WS-Security;* al obtener de manera personalizada las políticas de seguridad necesarias para el cliente, las cuales serán aplicadas y procesadas por *WSG-Client´s Agent*. De esta forma se hace más trasparente para el cliente el consumo de servicios protegidos. Por esta razón, el producto es liviano lo cual es un punto importante para la perspectiva del cliente, y para el administrador del producto.

Como se ha hablado a lo largo del documento, *WSG-Client* permite la aplicación de las políticas de seguridad de manera automática sin la necesidad de tener conocimiento de estas.

En el mundo real puede ser usado en las máquinas de los clientes que deben enviar y/u obtener mensajes SOAP, contenidos con datos sensibles y que deben cumplir con restricciones de seguridad. Esto ahorra los costos de tiempo y dinero que se deben invertir en la implementación de políticas de seguridad.

De la misma manera, *WSG-Client´s Agent* facilita el aseguramiento de servicios web y motiva a las empresas y desarrolladores a implementar mecanismos de seguridad en sus sistemas, y a no ignorarlas por temas económicos ni de desarrollo.

2. Trabajo a Futuro

Corto Plazo

Es necesario la revisión de *CleanCode* y realizar la verificación del código haciendo uso de un estándar de seguridad como OWASP. Esto para realizar un buen manejo del código y garantizar la calidad del mismo, ya que no debe tener vulnerabilidades de codificación ni ser fuente te ataques por algoritmos maliciosos.

Para el módulo de administración de *WSG-Client´s Agent*, es necesario realizar la integración de este con *WS-Guardian* en la parte de *Backend*, *Frontend* y datos, ya que este módulo complementa las entidades con las que ya cuenta *WS-Guardian.*

Además de esta integración, se puede hacer más flexible la configuración de algunos parámetros de entrada que requiere *WSG-Client´s Agent,* ya que en la actualidad estos valores deben ser ingresados de forma manual. Con esto se incrementa el nivel de automatización ofrecido al cliente.

Mediano Plazo

Definir nuevos requerimientos no funcionales para el manejo de concurrencia y paralelismo, para verificar la robustez de la aplicación como por ejemplo: realizar pruebas de estrés. Esto también es importante para observar y verificar cómo la aplicación actúa ante ataques como denegación de servicios (DDOS).

También es importante asegurar no solo la comunicación entre *WSG-Client´s Agent* y *WSG* como está actualmente, sino también asegurar *WSG-Client´s Agent* como tal y evitar que este no sea la fuente de ataque entre la máquina del cliente y *WSG*.

Largo Plazo

Se pueden implementar nuevas políticas de seguridad ya que *WS-Guardian* maneja actualmente nueve de ellas. Esto permitirá que *WSG-Client's Agent* cubra todas estas políticas y lograr que más clientes de la empresa *ITAC S.A.* usen el producto.

WSG-Client´s Agent puede ser ajustado para no solo aplicar seguridad a mensajes SOAP sino adicionar seguridad al consumo de servicios REST. Además, se podría pensar en eliminar la dependencia con *WS-Guardian,* para que pueda ser comercializado como un producto completamente independiente, que cuente con un repositorio de políticas de seguridad (librarías) del estándar de *WS-Security,* de donde descargará los componentes que requiera para su proceso.

3. Conclusiones

WSG-Client´s Agent logró cumplir los principios de seguridad básicos sobre a los mensajes SOAP, con la implementación de dos técnicas de seguridad: Firma digital y Timestamp. La primera asegura *autorización, autenticación, no repudio e integridad,* y la segunda evita ataques de repetición y permite determinar cuándo el mensaje fue creado y cuál es su vigencia. Además, el agente cuenta con la capacidad de enviar peticiones REST por https para descargar sus archivos de configuración, asegurando confidencialidad.

WSG-Client´s Agent fue diseñado para ser ejecutado en cualquier plataforma, siendo independiente al sistema operativo que se escoja para su ejecución. Esto se da gracias al lenguaje de desarrollo *Java (lenguaje de implementación)* que posee la característica de ser portable, lo que permite ejecutarse en diferentes plataformas. Esto se validó mediante pruebas de aceptación, en donde se probó la ejecución en los sistemas operativos: *Linux* y *Windows.*

Tanto los requerimientos funcionales y no funcionales fueron validados en *WSG-Client´s Agent,* mediante la ejecución de los casos de prueba diseñados en la plataforma *Spira Team.*

*E*stos casos de prueba fueron estructurados para cubrir cada requerimiento (historias de usuario). *A*demás, se generó un informe de resultados y análisis de las pruebas, donde se evidencia el funcionamiento y cumplimiento de los requerimientos.

IX- REFERENCIAS

[1] "SOA aporta a la Administración Pública un nuevo paradigma para mejorar la interoperabilidad entre sus aplicaciones. Su implantación requiere la aceptación de estándares y normas comunes," *ComputerWorld*, 16-Feb-2007. [Online]. Available: https://www.computerworld.es/archive/soa-aporta-a-la-administracion-publica-un-nuevo-paradigma-para-mejorar-la-interoperabilidad-entre-sus-aplicaciones-su-implantacion-requiere-la-aceptacion-de-estandares-y-normas-comunes. [Accessed: 30-Mar-2019].

[2] Packt, "SOA: Implementing Message-Level Security," *Packt Hub*, 30-Apr-2010. [Online]. Available: https://hub.packtpub.com/soa-implementing-message-level-security/. [Accessed: 30-Mar-2019].

[3] "Home - wsguardian.co." [Online]. Available: http://www.wsguardian.co/. [Accessed: 15-May-2019].

[4] "Mecanismos de WS-Security," 24-Oct-2014. [Online]. Available: undefined. [Accessed: 31-Mar-2019].

[5] "Reflection & Introspection: Objects Exposed." [Online]. Available: http://www2.sys-con.com/itsg/virtualcd/Java/archives/0305/sagar2/index.html. [Accessed: 21-Apr-2019].

[6] "IEEE 1058.1-1987 - Estándar IEEE para planes de gestión de proyectos de software." [Online]. Available: https://standards.ieee.org/standard/1058_1-1987.html#Additional. [Accessed: 06-Apr-2019].

[7] "Acerca de la especificación del lenguaje de modelado unificado versión 2.5.1." [Online]. Available: https://www.omg.org/spec/UML/About-UML/. [Accessed: 15-Apr-2019].

[8] "BPMN Specification - Business Process Model and Notation." [Online]. Available: http://www.bpmn.org/. [Accessed: 15-Apr-2019].

[9] "IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications." [Online]. Available: https://standards.ieee.org/standard/830-1998.html. [Accessed: 06-Apr-2019].

[10] "IEEE 1016-2009 - IEEE Standard for Information Technology--Systems Design--Software Design Descriptions." [Online]. Available: https://standards.ieee.org/standard/1016-2009.html. [Accessed: 06-Apr-2019].

[11] T. Preston-Werner, "Versionado Semántico 2.0.0-rc.2," *Semantic Versioning*. [Online]. Available: https://semver.org/lang/es/. [Accessed: 17-Apr-2019].

[12] S.-M. Lee, O.-S. Kwon, J.-H. Lee, C.-J. Oh, and S.-H. Ko, "TY*SecureWS: An Integrated Web Service Security Solution Based on Java," in *E-Commerce and Web Technologies*, 2003, pp. 186–195.

[13] "Oracle® Fusion Middleware Understanding Oracle Web Services Manager." [Online]. Available: https://docs.oracle.com/middleware/1212/owsm/OWSMC/owsm-intro.htm#OWSMC110. [Accessed: 21-Apr-2019].

[14] "Cloud Data Protection Solution | Digital Guardian." [Online]. Available: https://digitalguardian.com/products/cloud-data-protection. [Accessed: 16-Apr-2019].

[15] "Network Data Loss Prevention | Digital Guardian." [Online]. Available: https://digitalguardian.com/products/network-dlp. [Accessed: 16-Apr-2019].
[16] "SignSOAPRequest / Code / [r5] /SignSOAPRequest." [Online]. Available: https://sourceforge.net/p/signsoaprequest/code/HEAD/tree/SignSOAPRequest/. [Accessed: 09-Nov-2019].

X- APÉNDICES

1. Anexos

Anexo 1 – Historias de Usuario.xls
Anexo 2 – Diagrama de despliegue.pdf
Anexo 3 – Diagrama de clases_WSG-Client´s Agent.pdf
Anexo 4 - Diagrama de clases _administración del WSG-Client´s Agent.pdf
Anexo 5 - Diagrama de secuencia _Configuración.pdf
Anexo 6 - Diagrama de secuencia _Aplicar/Procear política.pdf
Anexo 7 - Diagrama de secuencia _Actualización.pdf

2. Listas de Tablas

3. Listas de Figuras

4. Documentos Relacionados

VFP (Propuesta del Proyecto)
PMP (Software Project Management Plan)
SRS (Software Requirements Specification)
SDD (Software Design Description)
Informe de Pruebas - Funcionales
Informe de Pruebas – No Funcionales
Manual de Usuario
Reporte Postmortem