Universitat Oberta de Catalunya

uoc.edu

Desarrollo de un SSO en una arquitectura basada en microservicios

Jordi Marimon-Illana
MISTIC-Máster universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Sistemas de autenticación y autorización

Antoni Gonzalez Ciria
Victor Garcia Font

2 de junio de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de un SSO en una arquitectura basada en microservicios*
Nombre del autor:	*Jordi Marimon Illana*
Nombre del consultor/a:	*Antoni Gonzalez Ciria*
Nombre del PRA:	*Victor Garcia Font*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Máster universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*Sistemas de autenticación y autorización*
Idioma del trabajo:	*Español*
Palabras clave	*Single-Sign-On, Microservicios, Python*

Resumen del Trabajo:

Los mecanismos de autorización y autenticación son clave en cualquier sistema que deba ser expuesto a Internet. El protocolo OpenID Connect ofrece autenticación encima del protocolo de autorización OAuth.

Con el desarrollo de este proyecto se pretende dar una solución de Single Sign-On que se adapte al conjunto de microservicios de la empresa Mitiga Solutions, añadiendo soporte al protocolo OpenID Connect.

Para el desarrollo se ha seguido una metodología en cascada, dividiendo el proyecto en fases específicas que permiten una mejor estimación del tiempo.

Con la realización de este trabajo se ha instalado y configurado el servidor de Single Sing-On Kyecloak el cual utiliza el servidor LDAP de la empresa para dar servicio a los usuarios internos. En la segunda parte del proyecto se ha modificado la librería de microservicios de Mitiga Solutions, así como dos de sus microservicios, para añadir compatibilidad con el protocolo OpenID Connect.

El sistema implantado permite proteger endpoints específicos de los microservicios y autorización mediante roles, así como la comunicación autenticada entre microservicios. El desarrollo ha permitido sentar las bases para extender el sistema al resto de microservicios disponibles.

El servidor de Single Sign On no sirve únicamente para los microservicios, sino que además pude ser utilizado para ofrecer un sistema de acceso centralizado para dar soporte a otros servicios disponibles en la empresa que sean compatibles con el protocolo utilizado.

Abstract:

Authorization and authentication mechanisms are a key component on any system exposed to Internet. The protocol OpenID Connect provides authentication on top of the authorization protocol OAuth.

With the development of this project, it is intended to provide a Single Sign-On solution for the set of microservices of the company Mitiga Solutions, adding support to the OpenID Connect protocol.

The development of this project has followed a cascade methodology, dividing the project into specific phases to allow a better time schedule.

With the completion of this project, the Sign Sing-On Kyecloak server has been installed and configured to work along with company's LDAP server in order to provide access to the internal users. As for the second part of the project, support to OpenID Connect protocol has been added to Mitiga's microservices library, as well as two of its microservices.

The implemented system allows to protect specific endpoints with authorization through roles, as well as authenticated communication between microservices.

The Single Sign On server is not only used for microservices but also can be used to provide a centralized access system to support other services available in the company that are compatible with the protocol OIDC.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

En Mitiga Solutions, una spin-off del Barcelona Supercomputing Center especializada en el análisis de riesgos atmosféricos se ha desarrollado un sistema operacional basado en microservicios que implementa la lógica del negocio.

En la actualidad se está planteando exponer a Internet algunos de los microservicios a través de sus APIs, así como una página web que se apoyara en el uso de dichas APIs para ofrecer el servicio.

Cabe destacar la necesidad de implementar un mecanismo de autenticación y autorización para hacer uso de los recursos que vayan a ser expuestos a Internet.

Dada la naturaleza descentralizada de los microservicios se requiere de un sistema que permita al usuario autenticarse de manera transparente y dotar al sistema de un aspecto uniforme al mismo tiempo que se garantiza la escalabilidad del sistema. Dadas las necesidades descritas anteriormente nace la necesidad de realizar este TFM, en el que se pretende implantar una solución de Single Sign-On en una arquitectura basada en microservicios.

Con esta solución se pretende dotar al sistema de autenticación centralizada, en la que un único proceso de autenticación sea necesario para acceder a cualquiera de los microservicios, utilizando estándares de la industria como OAuth2 [1] y OIDC [2].

En la arquitectura de microservicios de la empresa podemos diferenciar tres tipos de microservicios, esta diferenciación reside en el nivel de exposición del microservicio hacia el exterior.

- Adquisición de datos: Estos servicios nutren al sistema con los datos necesarios, realizando conexiones al exterior, pero sin admitir peticiones entrantes.
- Lógica de negocio: Estos servicios se encuentran únicamente conectados a la red interna, utilizan los datos obtenidos por el sistema y los transforman en información útil.
- Interfaz: Este último tipo de microservicio es el encargado de servir los datos al exterior e interactuar con el sistema, bien sea a usuarios del sistema o el público general. Los microservicios que se encuentran dentro de esta categoría son los que requieren de la interacción con el servicio de SSO.

Teniendo en cuenta el tipo de microservicios de los que se dispone, la solución implementada deberá centrarse únicamente en aquellos microservicios que admitan peticiones desde el exterior. Con esta aproximación se pretende reducir el volumen de peticiones al servicio de SSO.

Los usuarios del sistema se encuentran definidos en un servidor de LDAP, por tanto, un requisito del sistema de SSO es la compatibilidad con este protocolo para extraer la identidad de los usuarios.

Dadas las características de una empresa pequeña, la infraestructura de red de la que se dispone no puede garantizar la disponibilidad ni la calidad del servicio y su uso se limita a la conexión de usuarios y el hosting de servidores internos. Para garantizar un alto grado de disponibilidad del sistema, los distintos microservicios serán desplegados fuera de las instalaciones de la empresa a través de la contratación de un servicio de hosting, donde se instalará también el servicio SSO y una réplica de LDAP para garantizar la continuidad del sistema.

1.2 Objetivos del Trabajo

Los principales objetivos de este trabajo de fin de master se encuentran definidos en la siguiente tabla, donde se dividen los objetivos marcados entre los objetivos referentes al servició SSO y los microservicios.

Objetivo	**Prioridad**
Servicio SSO	
Autenticación mediante usuario y contraseña	Alta
Autenticación mediante certificados	Media
Compatibilidad con LDAP	Alta
Generación de tokens	Alta
Alta disponibilidad	Alta
Microservicios	
Aceptar tokens	Alta
Validar tokens	Alta
Autorizar y denegar peticiones	Alta

Tabla 1: Objetivos del proyecto

1.3 Enfoque y método seguido

Para la realización de este proyecto se utilizará una metodología en cascada. En este tipo de metodología las distintas fases del proyecto están bien definidas, permitiendo una planificación y seguimiento más sencillo, si bien realizar modificaciones en etapas avanzadas del desarrollo puede resultar costoso.

Se ha optado por utilizar esta metodología debido al corto periodo de tiempo disponible para la realización del proyecto y la facilidad de dividirlo en etapas bien diferenciadas.

La fase inicial en la metodología en cascada suele ser la captación de requerimientos, para este proyecto omitiremos esta primera fase, pues los requerimientos nos vienen impuestos de antemano.

Las fases en las que se dividirá el proyecto son las siguientes:

- Investigación: En esta primera etapa se buscarán y analizarán las distintas soluciones de SSO disponibles en el mercado y se seleccionará la solución más adecuada para la realización de este proyecto.
- Diseño: En esta etapa se definirá el comportamiento del sistema, tanto a nivel del servidor, como a nivel de los microservicios, definiendo las comunicaciones entre ambos, que elementos se deberán añadir en los microservicios para comunicarse con el servidor escogido, cuáles serán los mecanismos de autenticación y autorización, etc.
- Instalación y configuración: En esta fase procederá a realizar la instalación y configuración del servidor escogido en la primera etapa.
- Desarrollo: En esta etapa se procederá a implementar el diseño
- Test y ajustes: Se validará el funcionamiento del sistema junto al servidor implantado, y se realizaran los ajustes necesarios en el servidor o los microservicios.
- Despliegue del sistema: Finalmente, se procederá a desplegar el sistema, exponiendo a Internet los microservicios necesarios.

1.4 Planificación del Trabajo

En este apartado se define la planificación temporal para el desarrollo de este proyecto, incluyendo la dependencia entre cada una de las tareas, tal y como se muestra en la tabla 2. El tiempo estimado para las tareas incluyen la previsión temporal requerida para reuniones y elaboración de documentación.

ID	Nombre	Inicio	Fin
Fase 1	**Estudio**	**04/03/2020**	**09/03/2020**
1.1	Estudio de los distintos servicios de SSO	04/03/2020	05/03/2020
1.2	Realizar análisis de coste – beneficio	05/03/2020	09/03/2020
1.3	Elegir el servicio apropiado	09/03/2020	09/03/2020
Fase 2	**Diseño**	**10/03/2020**	**23/03/2020**
2.1	Estudio de la documentación del servicio SSO	10/03/2020	10/03/2020
2.2	Diseño del mecanismo de redirección	11/03/2020	11/03/2020
2.3	Diseño del mecanismo de autenticación	12/03/2020	12/03/2020
2.4	Diseño del mecanismo de autorización	13/03/2020	13/03/2020
2.5	Diseño del mecanismo de obtención de tokens	16/03/2020	16/03/2020
2.6	Diseño del mecanismo de validación de tokens	17/03/2020	17/03/2020
2.7	Diseño del mecanismo de renovación de tokens	18/03/2020	18/03/2020
2.8	Elaboración del diagrama de flujo para establecer la comunicación entre el servidor SSO y los microservicios	19/03/2020	20/03/2020
2.9	Validación de los diseños	19/03/2020	23/03/2020
Fase 3	**Implantación del servicio SSO**	**24/03/2020**	**31/03/2020**
3.1	Instalación del servicio SSO	24/03/2020	24/03/2020
3.2	Configuración del servicio SSO	24/03/2020	26/03/2020
3.3	Integrar el servicio SSO con LDAP	26/03/2020	30/03/2020
3.4	Verificación del funcionamiento del servicio	31/03/2020	31/03/2020
Entrega 2	**Incluye hasta la Fase 2**	**31/03/2020**	**31/03/2020**
Fase 4	**Desarrollo del sistema de autenticación y autorización**	**01/04/2020**	**20/04/2020**
4.1	Implementar la lógica de autenticación	01/04/2020	02/04/2020
4.2	Implementar el mecanismo de autenticación mediante usuario y contraseña	03/04/2020	06/04/2020
4.3	Implementar el mecanismo de autenticación mediante certificados	06/04/2020	09/04/2020
4.4	Implementar el mecanismo de obtención de tokens	09/04/2020	10/04/2020
4.5	Implementar el mecanismo de validación de tokens	13/04/2020	13/04/2020
4.6	Implementar el mecanismo de renovación de tokens	14/04/2020	16/04/2020
4.7	Implementar el mecanismo de redirección	17/04/2020	20/04/2020
Fase 5	**Validación del sistema**	21/04/2020	27/04/2020
5.1	Validar la generación de tokens	21/04/2020	21/04/2020
5.2	Validar la autenticación de tokens	22/04/2020	22/04/2020
5.3	Validar el mecanismo de redirección	23/04/2020	23/04/2020
5.4	Validar el mecanismo de renovación de tokens	24/04/2020	24/04/2020

5.5	Validar el funcionamiento global del sistema	27/04/2020	27/04/2020
Entrega 3		**28/04/2020**	**28/04/2020**
Fase 6	**Despliegue del sistema**	28/04/2020	04/05/2020
6.1	Despliegue del sistema en producción	28/04/2020	04/05/2020
6.2	Monitorización del sistema	28/04/2020	04/05/2020
Fase 7	**Memoria y presentación**	**05/05/2020**	
7.1	Conclusiones del trabajo	05/05/2020	08/05/2020
7.2	Elaboración de la memoria final	11/05/2020	22/05/2020
7.3	Elaboración del video de la presentación	25/05/2020	29/05/2020
Entrega final		**02/06/2020**	**02/06/2020**

Tabla 2: Planificación

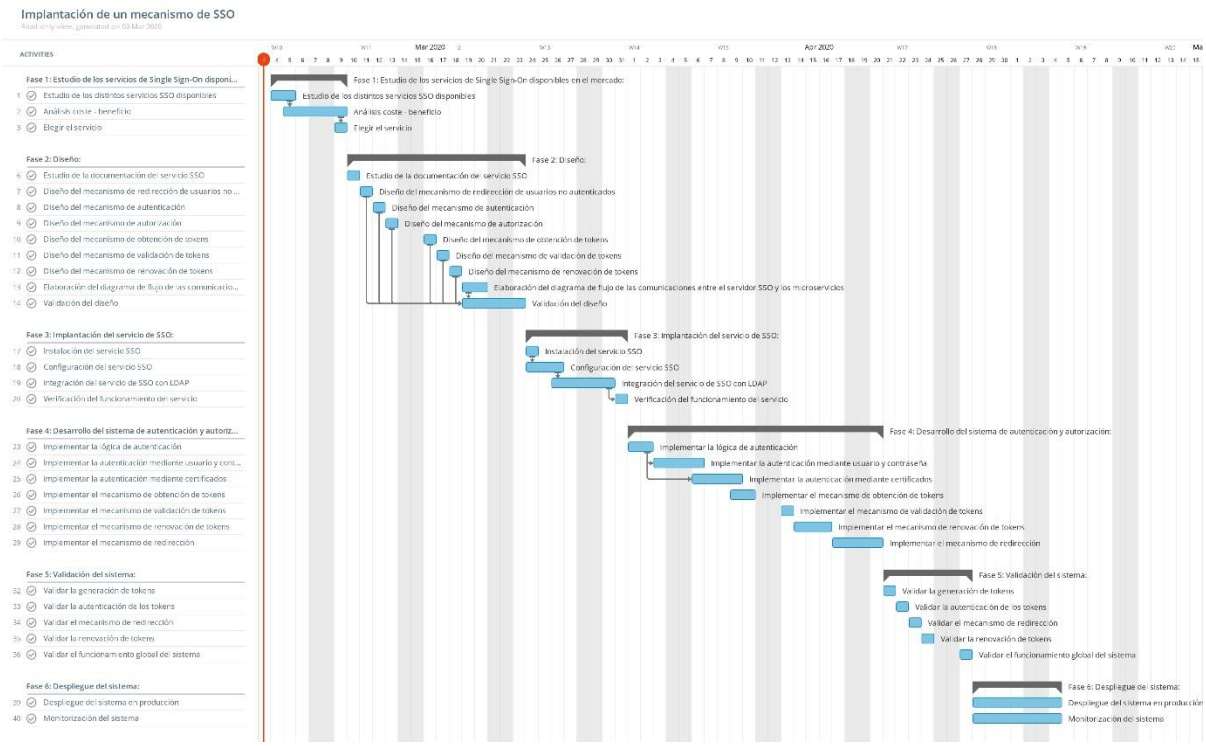


Figura 1: Diagrama de Gantt

1.5 Herramientas utilizadas

En este punto trataremos las herramientas, tecnologías, lenguajes y librerías utilizados para el desarrollo de este proyecto.

OIDC: Es una capa de autenticación construida encima del protocolo OAuth2.0. El uso de este protocolo permite a realizar la autenticación y autorización de acceso sin necesidad de que el usuario este constantemente introduciendo sus credenciales en el sistema.

Docker: Docker es una aplicación de virtualización que permite abstraer el código que se ejecuta en su interior del sistema operativo donde se está ejecutando.

KeyCloak: Es un servicio de IAM de código abierto, que implementa los estándares OIDC y OAuth2.0. Con KeyCloak se pretende dar un servicio de SSO de manera centralizada.
LDAP: El protocolo ligero de acceso a directorios permite la organización jerárquica de un conjunto de atributos. El uso de este protocolo nos permite definir los usuarios que existen en el sistema.

Python: Es el lenguaje de programación utilizado para la creación de los microservicios en MitigaSolutions. Es por ello que para la realización de este proyecto es necesario adaptar el código actual para añadir las capacidades de autenticación y autorización.

Flask: Es la librería utilizada para la creación de los microservicios que incluye las funcionalidades necesarias para crear un servidor web.

Flask-OIDC: Es la extensión de la librería en la que se implementan las funcionalidades relacionadas con OIDC.

Draw.io: Esta herramienta ha sido utilizada para la creación de los diagramas presentes en este documento.

1.6 Breve sumario de productos obtenidos

Con el desarrollo de este proyecto se ha logrado obtener la base para dotar al equipo de desarrollo de los microservicios las herramientas y el conocimiento necesario para añadir la capa de autenticación y autorización para cualquiera de los microservicios desarrollados en función de sus necesidades.

Se ha instalado y configurado el servidor de SSO Keycloak. En el proceso de configuración se ha conectado el servicio de SSO a un servicio LDAP para obtener los usuarios pertenecientes a la empresa y se han configurado los clientes necesarios para dar soporte a dos de los microservicios. Por último, se ha modificado el aspecto del servidor.

A nivel de los microservicios, se ha añadido soporte al protocolo OpenID Connect, mediante la modificación de los endpoints de los propios microservicios así como la modificación de la librería interna de Mitiga Solutions. El uso de este mecanismo es opcional, permitiendo el uso del sistema implementado en función de las necesidades de cada uno de los microservicios. Las funcionalidades que se han desarrollado son las siguientes:

- Aceptación de tokens
- Validación de tokens
- Actualización de tokens
- Autorización basada en roles
- Mecanismo de cierre de sesión

1.7 Breve descripción de los otros capítulos de la memoria

En el segundo apartado de este proyecto se introducen los sistemas SSO y como pueden usarse para habilitar los mecanismos de autenticación y autorización en una arquitectura basada en microservicios, seguido del diseño de los mecanismos necesarios a implementar en los microservicios. En la sección 4 se describe el proceso de instalación y configuración del servidor Keycloak y se crea un servidor simple en Python integrado con el servicio SSO a modo de PoC. En la sección 5 se desarrolla la implementación del código necesario para adaptar los microservicios ya existentes al uso del servicio de SSO, en este apartado se realiza la modificación de la librería de microservicios de MitigaSolutions, junto a dos de sus microservicios que nos permitirán cubrir los distintos casos de uso, seguido del apartado 6, donde se realiza la validación de los mecanismos habilitados con el uso de la herramienta postman. En el apartado 7 realiza el despliegue del sistema con los microservicios modificados y se realiza un conjunto de verificaciones del funcionamiento del sistema. Por último, en el apartado 8 se muestran las conclusiones del proyecto.

2. Introducción a los sistemas SSO

2.1 Servidores de Single Sign On

El single sign-on, o autenticación única, se basa en que el usuario debe autenticarse una única vez en uno de los sistemas federados y, a partir de ahí puede acceder al resto de sistemas sin autenticarse de nuevo.

Uno de los puntos clave de este mecanismo consiste en reducir la cantidad de credenciales que un usuario debe recordar, disminuyendo la fatiga de autenticación [3], logrando una mayor implicación por parte del usuario en establecer una contraseña segura.

El uso de este tipo se sistemas permite una gestión centralizada de los accesos, reduciendo los costes globales de mantenimiento.

2.2 Características del servicio SSO en una arquitectura basada en microservicios

A diferencia de una aplicación clásica, donde el conjunto del sistema tiene un único punto de acceso, en una arquitectura basada en microservicios, cada uno de los módulos con contacto con el exterior requieren de autenticación, tal y como se muestra en la figura 2

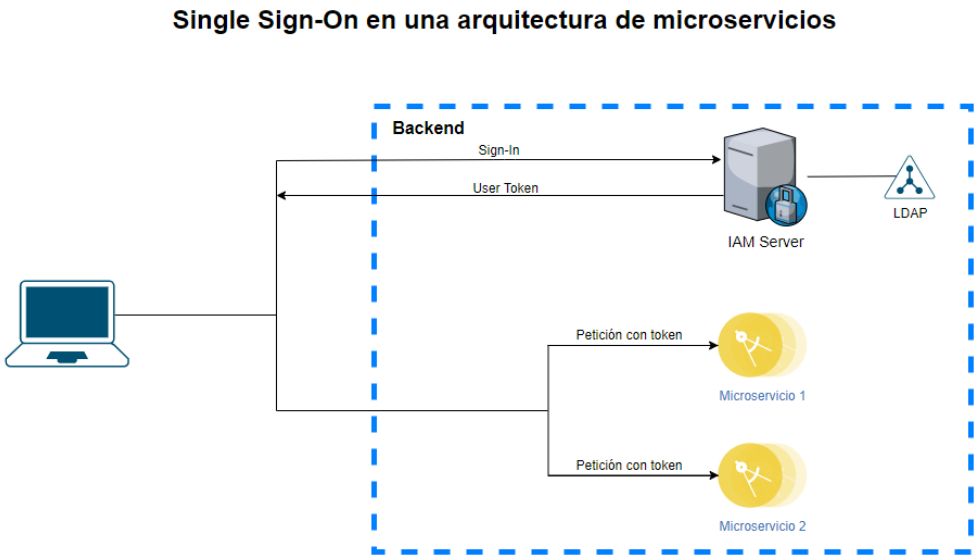


Figura 2: Autenticación en los microservicios

Se debe tener en cuenta cuales son las características inherentes a una arquitectura de microservicios, a grosso modo podemos destacar la escalabilidad, la disponibilidad y la tolerancia a fallos como características principales. Es por ello por lo que implantar un sistema de SSO en este tipo de arquitecturas supone un reto. Por un lado, se debe garantizar la disponibilidad del servicio, fijémonos pues, si para realizar

acciones sobre el sistema se requiere de autenticación y el servicio de SSO no está disponible dejaría inutilizado todo el sistema.

El siguiente punto que tratar es la escalabilidad, en este apartado debemos diferenciar dos factores que pueden afectar a la escalabilidad del sistema. Por un lado, se debe tener en cuenta el crecimiento de peticiones producido por un aumento en los usuarios del sistema, y por el otro lado, el sistema debe ser capaz de soportar un crecimiento en el número de microservicios que dependan de este.

Para dar solución a los problemas de disponibilidad y escalabilidad se pueden optar por técnicas de clustering, replicando el mismo servicio en múltiples instancias para adaptarse a la demanda del sistema. Si bien este tipo de técnicas permite al sistema adaptarse a la demanda introducen también los problemas clásicos de sincronización, integridad, balanceo de carga, etc.

Para la solución que implementará en el transcurso de este TFM se ha optado por utilizar una solución que utilice el protocolo OpenId Connect. Este protocolo es una extensión del protocolo OAuth2 que ofrece Autenticación por encima del protocolo de Autorización.
Este protocolo utiliza JSON WebTokens (JWT), que toman el rol de las credenciales utilizadas para obtener acceso a una API determinada. El token utilizado contiene la información necesaria del usuario para poder autorizar su acceso.

2.3 Sistemas SSO en el mercado

Si bien existen más soluciones disponibles en el mercado, debido al tiempo disponible para la realización del trabajo se ha optado por realizar una pequeña selección. Todas las opciones seleccionadas tienen en común las siguientes características: federación de usuarios mediante LDAP, uso de los protocolos OAuth2 y OIDC.
Se debe tener en cuenta que, para el tipo de aplicación que estamos tratando, el sistema de SSO seleccionado deberá dar solución a los puntos vistos en el apartado anterior, dando un mayor peso a la escalabilidad y disponibilidad del sistema.

2.3.1 Keycloak

Ofrece una solución de IAM de código abierto, enfocada a las aplicaciones y servicios modernos. Esta solución ofrece un alto nivel de personalización mediante plugins.
Las características más interesantes que ofrece esta solución son la escalabilidad y la disponibilidad mediante clustering, permitiendo la creación de réplicas del servicio para ofrecer la calidad de servicio necesaria en función de los recursos disponibles. Permite definir políticas de contraseñas y promete un alto rendimiento.

Un último punto a tener en cuenta el Identity Brokering, que ofrece la posibilidad de utilizar proveedores de identidad de terceros, como pueden ser las redes sociales.

Keycloak es una solución de SSO pensada para aplicaciones web y servicios RESTful, como son por ejemplo los microservicios que queremos securizar. Nos ofrece un conjunto de mecanismos, como los usuarios, roles, clientes, proveedores de identidad [4].

2.3.3 OpenAM Community Edition

Esta solución de código abierto que cuenta con más de 8 años de experiencia y un desarrollo constante. Si bien desde la versión community [5] no se indican las características del software, podemos encontrar que en su versión de pago en ForgeRock se asegura la alta disponibilidad y escalabilidad

2.3.4 Apereo CAS

En su versión de código abierto cuenta con una comunidad activa de desarrolladores, con soporte a una gran variedad de servicios de autenticación, como LDAP, RADIUS, JWT, MongoDb entre otros, incluye herramientas de monitorización de aplicaciones [6].
Finalmente cabe destacar que este software dispone de un conjunto de recomendaciones y guias para establecer una arquitectura que garantiza una alta disponibilidad y escalabilidad mediante clustering.

2.3.5 WSO2 Identity Server

Solución IAM de código abierto basado en estándares abiertos, y permite una alta extensibilidad y personalización, además de permitir ambas, federación de identidades, compatible con LDAP. Una característica interesante de esta aplicación es la gestión del consentimiento sobre los datos personales siguiendo la regulación GDRP [7].

2.3.6 Comparativa

Tras analizar brevemente el software de SSO disponible se ha confeccionado una tabla con el objetivo de poder elegir la opción apropiada para el proyecto.

Característica	Descripción	Key Cloak	Apereo CAS	Open AM	WSO2
Protocolos					
OIDC	Protocolo de autorización basado en OAuth2. En este apartado se valora que la aplicación este certificada [8]	✓	✗	✓	✓
OAuth2	Protocolo estándar de autorización que permite la delegación de autenticación de los clientes.	✓	✓	✓	✓
Federación de identidades					
LDAP	Protocolo ligero de	✓	✓	✓	✓

	acceso a directorios				
Identity Brokering	Provee la capacidad de identificar al usuario a partir de múltiples fuentes, como pueden ser Google o facebook	✓	✓	✓	✓
Soporte técnico					
Documentación	La documentación en aplicaciones de código abierto son una parte clave para lograr integrar el producto dentro del sistema.	✓✓	✓✓	✓	✓
Foro/Mailing list	Al no disponer de un soporte técnico, la actividad de los foros asociados a la aplicación puede resultar clave a la hora de abordar los problemas, en este apartado valoramos el tiempo de respuesta general y la actividad mostrada	Foro activo, Tiempo de respuesta entre 1-7 días	Mailing list activa, Tiempo de respuesta entre 1-3 semanas	Foro con poca actividad y, consultas sin respuesta.	Foro activo. Tiempo de respuesta entre 1-7 días
Escalabilidad	Que técnicas utilizan los distintos sistemas para ofrecer escalabilidad	Clustering	Clustering + propuestas de arquitectura	Clustering	Clustering
Otras funcionalidades					
OTP	Contraseñas de un único uso.	✓	×	✓	✓
Políticas de contraseñas	Permite definir políticas de contraseñas	✓	×	✓	✓
Políticas de acceso	Permite definir políticas de acceso a los usuarios.	Permite definir restricciones temporales de acceso			

Tabla 3: Comparativa de los distintos servidores de SSO OpenSource.

La aplicación por la que se optará para el desarrollo de este proyecto es KeyCloak. Si bien los demás servicios mostrados cumplen con los requisitos necesarios para nuestra arquitectura, KeyCloak ha destacado en 2 aspectos principales, dispone de la versión completa open source, además de una amplia documentación. Sin duda, disponer de una buena documentación asociada a la aplicación seleccionada nos da la confianza de poder llevar a cabo el proyecto de manera satisfactoria dentro de los términos establecidos. El último punto diferencial ha sido la actividad mostrada en los foros de la aplicación, así como los tiempos de respuesta.

3. Diseño

Previo al inicio del diseño de los mecanismos necesarios es importante tener presente una fotografía del sistema actual, sus capacidades y sus limitaciones. Tal y como podemos ver en la figura 3, en la actualidad la interacción con el sistema por parte del usuario se realiza a través de una aplicación web, la cual se comunica el microservicio encargado del backend y es este último el que se comunica con el resto de los microservicios de ser necesario.

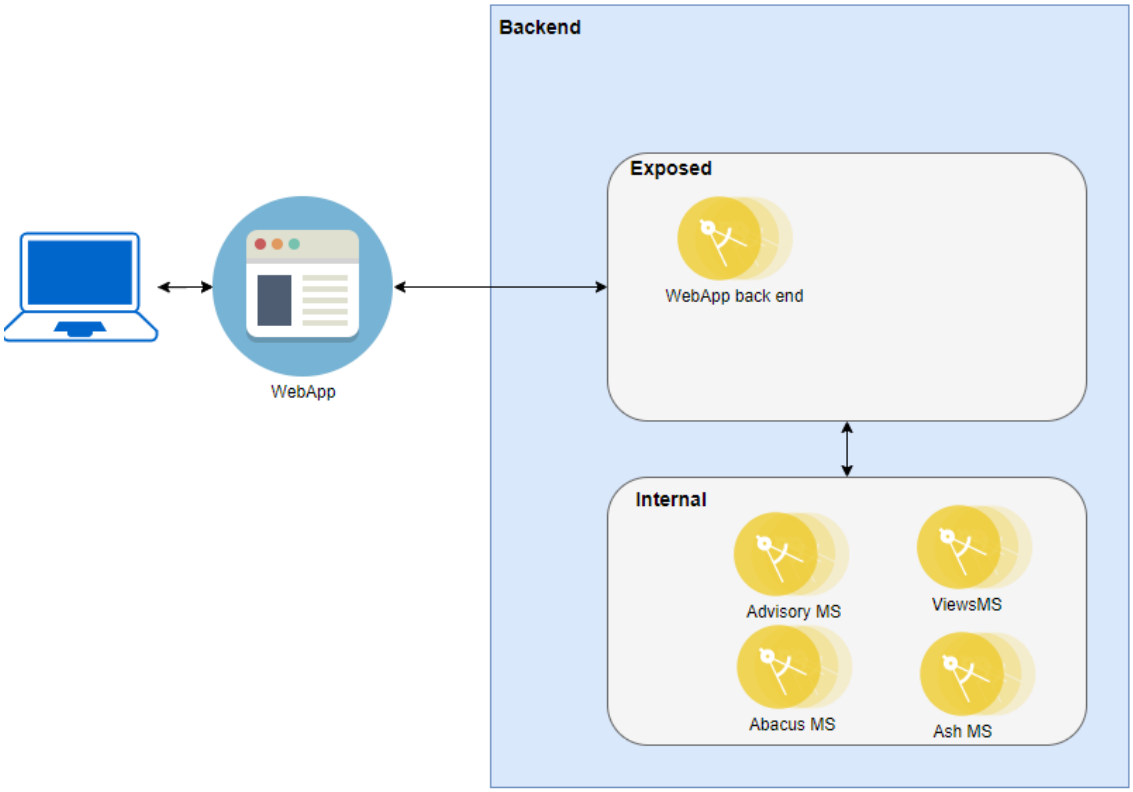


Figura 3: Arquitectura actual del sistema

Con esta aproximación únicamente nos debemos preocupar de un único punto de acceso al sistema, pero limita la capacidad que podemos ofrecer con el resto de los microservicios, por ejemplo, si queremos aplicar un modelo B2B en el cual se permita a los clientes atacar directamente a microservicios concretos se debe exponer el microservicio, y gestionar su seguridad desde el propio microservicio.

Lo que se pretende con el sistema que se desarrollara a lo largo de este TFM es la implantación del servicio de SSO y la adopción del protocolo OIDC en los microservicios. Tal y como se muestra en la figura 4, en este nuevo esquema disponemos del servidor SSO, que será el encargado de autenticar y autorizar al usuario. En este nuevo esquema, la exposición de un microservicio se respaldará del servidor de SSO para permitir únicamente las peticiones realizadas por un usuario autorizado, y en caso de permitir a un cliente el acceso a un recurso especifico este podrá hace uso directamente de la API REST definida en el microservicio.

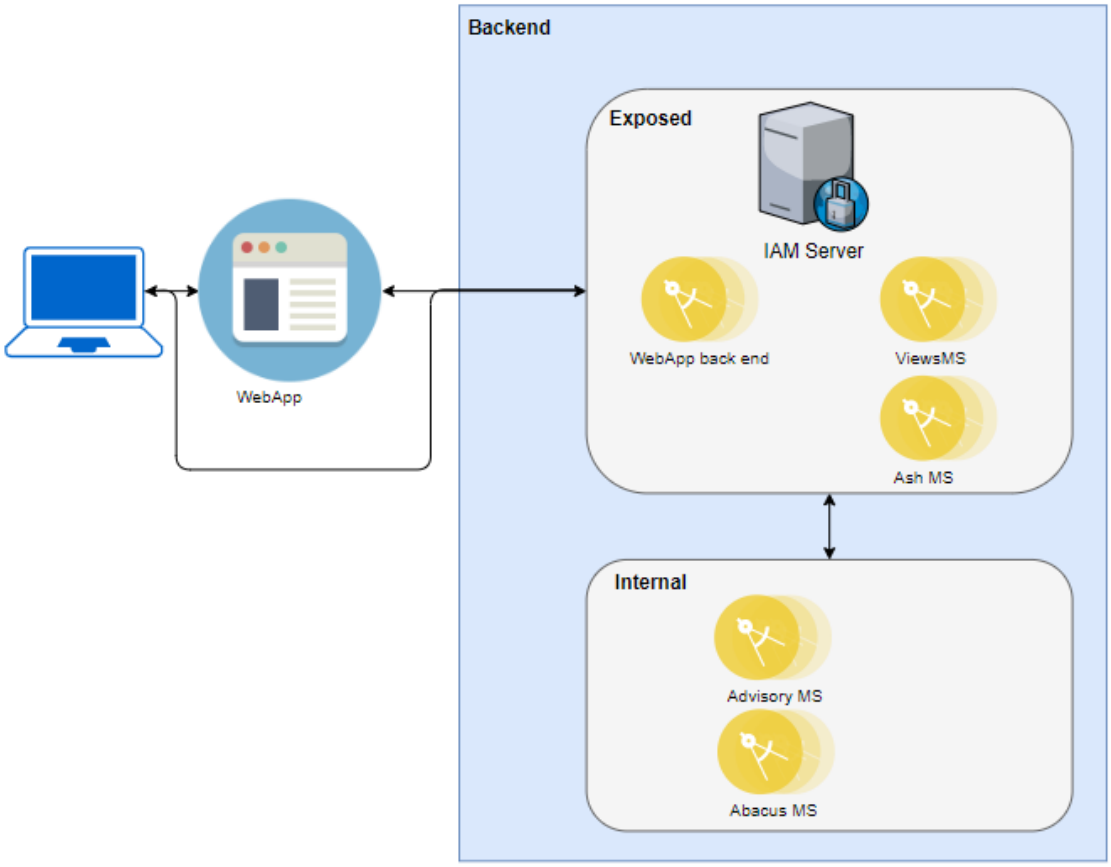


Figura 4: Arquitectura de microservicios con SSO

3.1 Diseño del mecanismo de redirección.

Este mecanismo se encargará de redirigir las peticiones http tanto por parte de los microservicios hacia el servidor de SSO como de vuelta del servidor SSO al endpoint del microservicio que ha invocado la llamada.

Cuando un microservicio reciba una petición **sin token** en un endpoint que requiera autenticación se redirigirá al usuario al servicio de autenticación, donde se le asignará un token que contendrá la información necesaria para identificar al usuario. Con este token el usuario podrá realizar peticiones autorizadas en el sistema, las cuales pueden ser denegadas en caso de no disponer de los permisos requeridos. El comportamiento esperado se muestra en la figura 5.

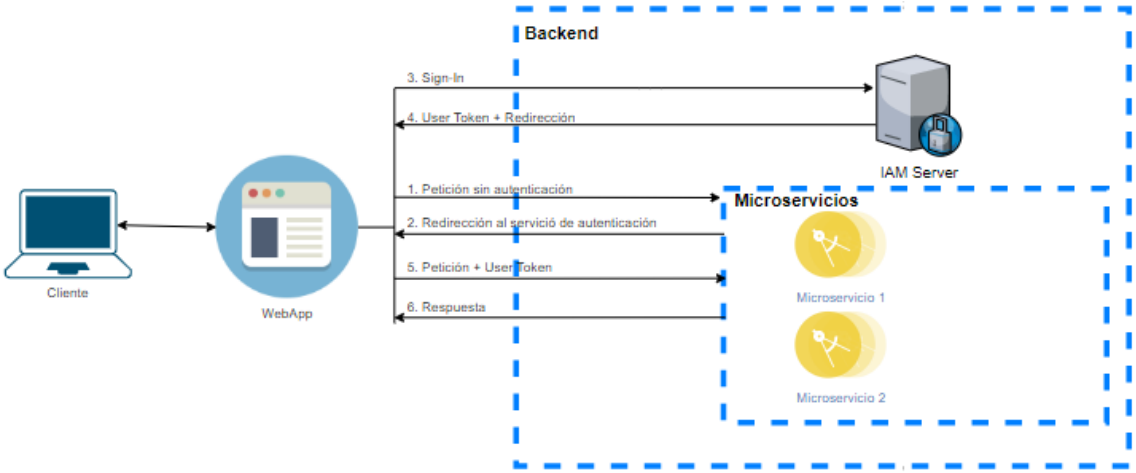


Figura 5: Mecanismo de redirección

Si bien la idea general del mecanismo queda reflejada con la figura anterior, entraremos más en detalle de este mecanismo con ayuda de la figura 6, donde se extienden las funcionalidades del mecanismo a los casos donde los procesos de autenticación/autorización fallan.

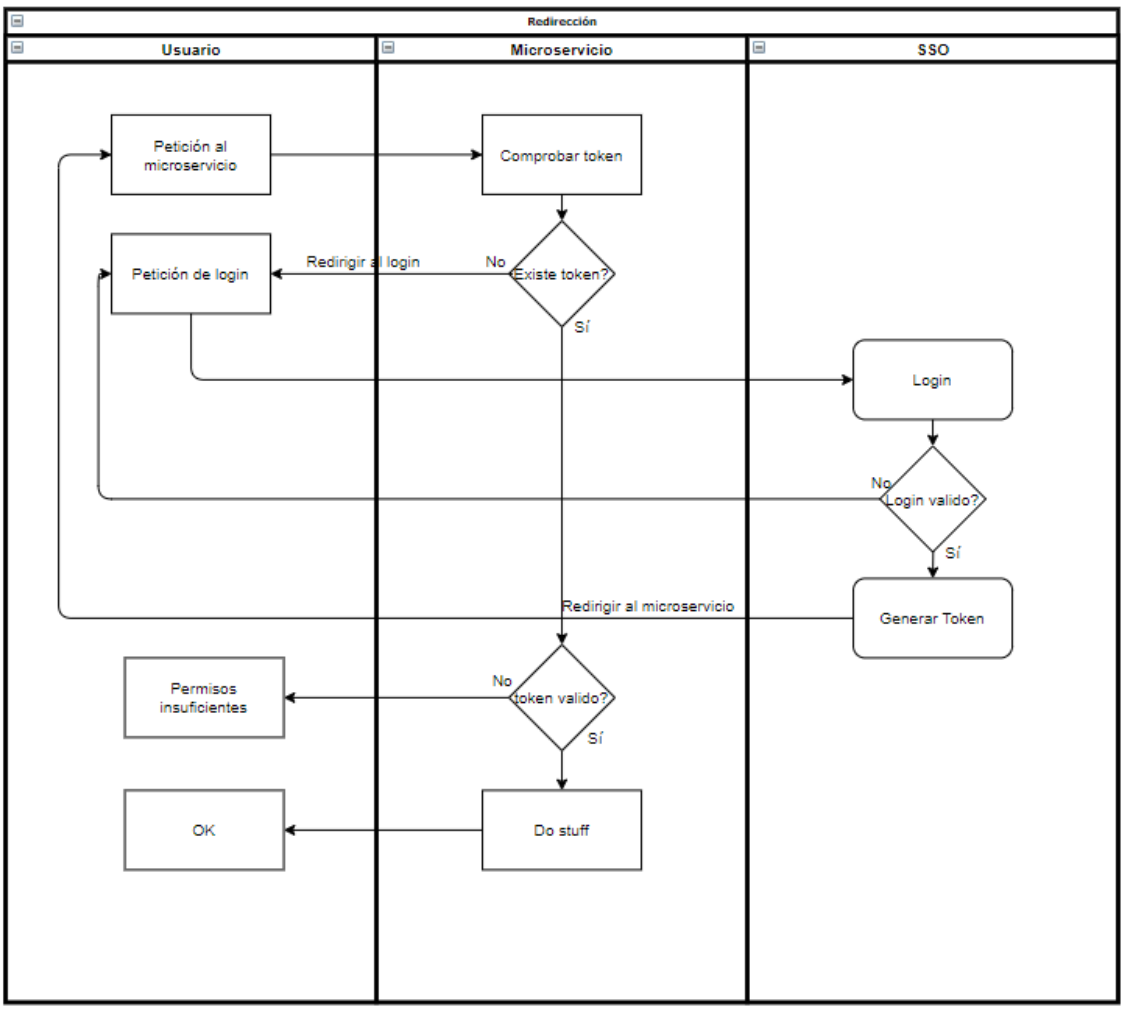


Figura 6: Autenticación - Diagrama de flujo

3.2 Diseño del mecanismo de autenticación.

El mecanismo de autenticación hará uso del protocolo OIDC, el cual utiliza un token JWT. Tal y como se muestra en la figura anterior, únicamente se aceptarán aquellas peticiones con un token asociado en caso de ser necesario.

Nótese que disponer de un token construido por el usuario no servirá para engañar al sistema y obtener acceso de manera fraudulenta. Para ello el token llevará asociada una firma generada por el servidor SSO mediante criptografía de clave pública. En la figura 7 se puede ver un esquema de los campos del token.

Con el uso de este token, los microservicios son capaces, no solo de autenticar la petición del usuario, puesto que este ha demostrado ser quien dice ser mediante el proceso de login en el servicio de SSO, sino que además nos provee del mecanismo necesario para realizar la autorización de la petición tal y como se verá en el siguiente punto.

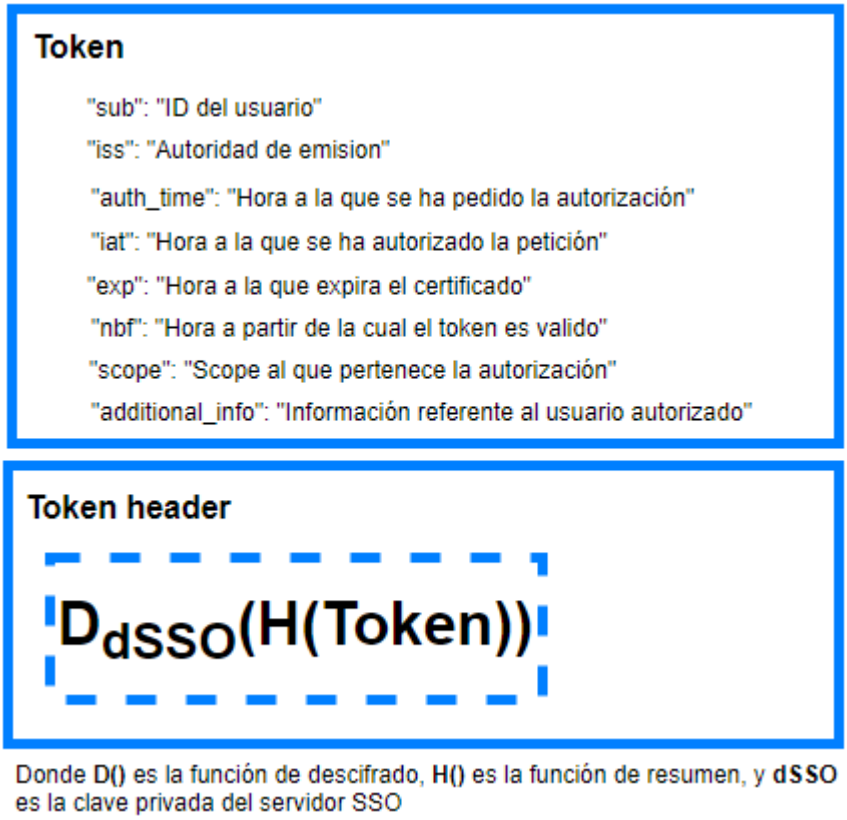


Figura 7: Contenido del Token

3.3 Diseño del mecanismo de autorización.

La validación de los permisos del usuario se puede realizar de dos maneras distintas. Por un lado, cuando el usuario realiza una petición a un microservicio este va a decidir si el usuario dispone de los permisos necesarios comprobando el contenido del token y tomando la decisión por su cuenta. Por otro lado, en lugar de ser el microservicio quien autorice la petición del usuario, otra opción es preguntarle al servidor

SSO si el usuario dispone o no de los privilegios necesarios para realizar una acción y actuar en función de la respuesta del servidor.

Dada la naturaleza descentralizada de los microservicios, la latencia introducida por la segunda opción podría llegar a repercutir en el rendimiento del sistema, sin contar la sobrecarga de peticiones en el servidor de SSO. Por estos motivos, la opción preferida para la autorización será la primera, cada microservicio será responsable de aceptar o denegar las peticiones recibidas.

La aproximación tomada pretende además hacer transparente el sistema de SSO a los desarrolladores, de manera que la modificación de un microservicio o la creación de uno nuevo no implique la modificación de los permisos definidos en el servidor SSO, sino que dependerán directamente de las necesidades del microservicio y, por ende, serán gestionados por el mismo.

Para lograr esta funcionalidad, los tokens expedidos por nuestro servidor de SSO deberán contener la información necesaria del usuario para permitir al microservicio tomar la decisión de autorizar o rechazar una determinada acción, tal y como se mostraba en la figura 6.
Nótese que el token expedido contiene información perteneciente al usuario y por ende dicha información debe estar protegida. Es por ello que el sistema únicamente permitirá peticiones sobre TLS de manera que la información se mantendrá oculta en su paso por redes hostiles como es Internet.

3.4 Obtención y validación de los tokens.

Los microservicios de los cuales se dispone actualmente están escritos en Python, utilizando la librería Flask.

Para añadir la nueva funcionalidad de autenticación y autorización utilizaremos la extensión de esta librería, Flask-OIDC [9]. El uso de esta librería es bastante simple, nos provee del método user_loggedin para determinar si el usuario se encuentra autenticado y en caso de estarlo se dispone de un conjunto de métodos para obtener la información disponible del usuario.

A su vez, para la validación del token se seguirán los siguientes pasos:

1. Obtener el token
2. Verificar la firma del token
3. Verificar la coherencia de las fechas del token
4. Verificar si el token dispone del scope necesario para acceder al recurso.

Puesto que para la expedición del token se utiliza criptografía de clave publica, para comprobar la autenticidad del token únicamente se debe verificar su firma, la cual se ha realizado con la clave privada del servidor SSO.

El scope del token nos permite conocer cuál es el alcance que se le ha concedido al usuario y por ende si dispone de los permisos necesarios para acceder a un recurso especifico.

3.5 Diseño del mecanismo de renovación de tokens.

El objetivo de este mecanismo no es otro que hacer el sistema más amigable para el usuario, de manera que mientras un usuario se mantenga activo durante la sesión se extenderá de manera automática su token, tal y como se muestra en la figura 8.

El funcionamiento es bastante sencillo. Para ello, cuando el sistema reciba una petición. Procederá a comprobar la validez del token. Si la fecha de expiración es anterior a la fecha actual, el microservicio descartará el token y redirigirá al usuario a la página de log in, para mayor detalle ver el punto 3.1. Si el token es válido, se procederá a comprobar cuanto tiempo de validez le queda al token, y en caso de encontrarse por debajo de un umbral se procederá a solicitar al servicio de SSO la extensión del token, para ello se hará una petición de *refresh_token.* Finalmente se procederá a validar el nuevo token para tomar la decisión de autorización pertinente. Para más detalle del proceso de autorización ver el punto 3.3.

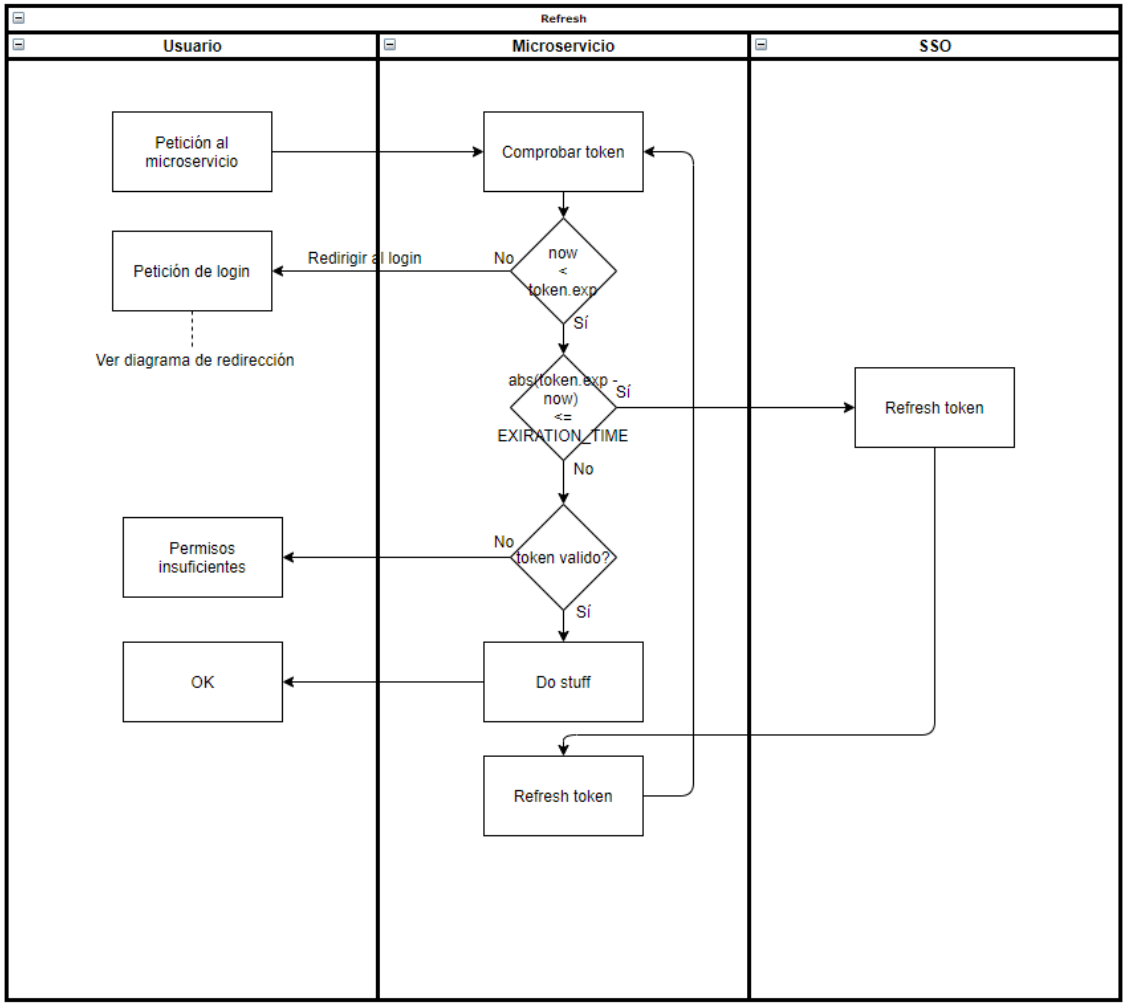


Figura 8: Renovación de tokens - Diagrama de flujo

4. Instalación del servicio

El contenido a partir de este punto se tratará en la tercera entrega. En caso de ser necesario el proceso de instalación se tratará en un anexo.

En este apartado veremos el proceso de instalación y configuración del servicio KeyCloak.

4.1 Instalación del servicio Keycloak

Siguiendo el manual de keycloak [10], el proceso de instalación se limita a la creación de un contenedor de Docker con una imagen creada por los desarrolladores.

```
docker run -p 80:8080 \
        -e KEYCLOAK_USER=admin \
        -e KEYCLOAK_PASSWORD=$KEYCLOAK_PWD \
        quay.io/keycloak
```

Los argumentos pasados a la llamada a Docker nos permiten definir las variables de entorno que utilizara para la creación del usuario administrador, en este caso, el nombre de usuario definido es “admin” y como contraseña utilizaremos la contraseña que hemos definido. Nótese que en este punto no se nos está exigiendo ningún tipo de complexidad a la hora de establecer la contraseña y será tarea del administrador establecer una contraseña segura.
El argumento -p nos permite definir el mapeo de puertos, en la imagen de docker, keycloak se ejecuta en el puerto 8080, el cual redirigiremos al puerto 80 del servidor.

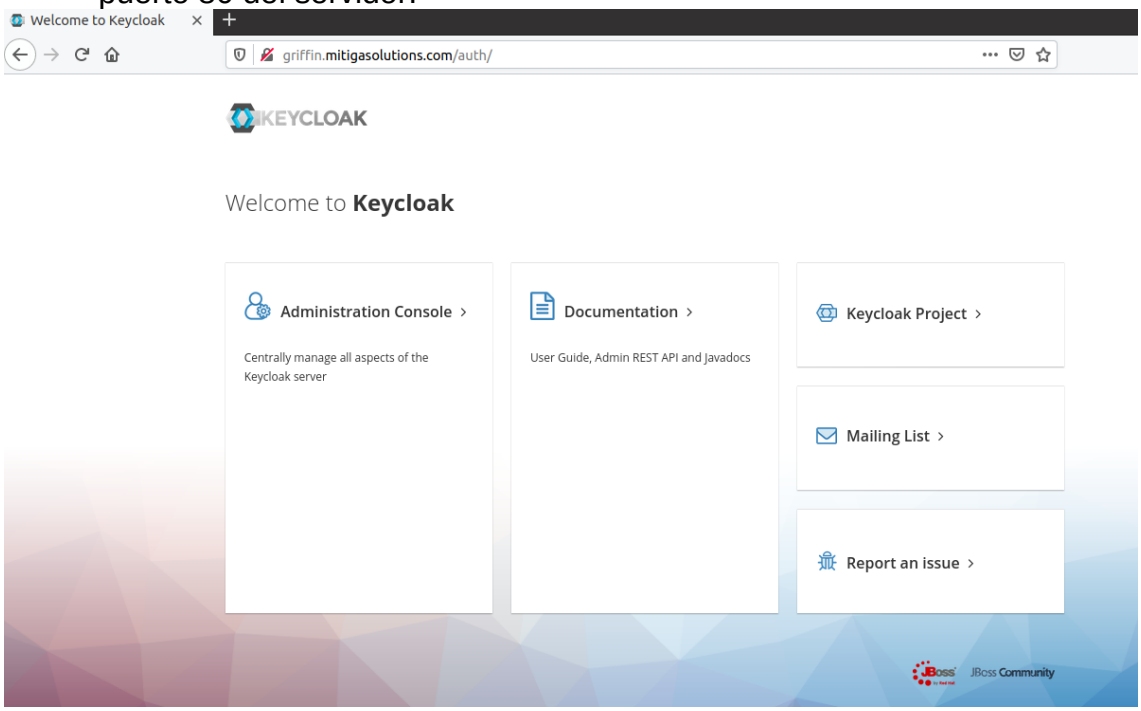


Figura 9: Pantalla de bienvenida de Keycloak

Con estos sencillos pasos ya tenemos el servidor operativo, tal y como podemos ver en la Figura 9, donde vemos la página de bienvenida al servidor. El siguiente paso consiste en entrar a la consola del administrador y configurar el servidor.

4.2 Creación del nuevo Realm.

Tras la instalación del servidor crearemos el Realm correspondiente a los usuarios de Mitiga Solutions, para ello accederemos al menú de creación de los reinos y añadiremos el nuevo real, tal y como se muestra en la Figura 10.

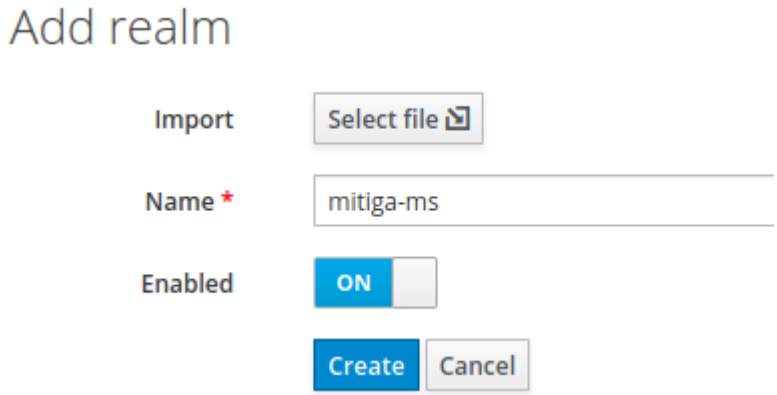


Figura 10: Creación de un nuevo reino

En el siguiente paso nos dirigiremos a la pestaña de login de la configuración del reino, donde habilitaremos la opción para permitir al usuario darse de alta, posteriormente, deberemos configurar este proceso para asignar los privilegios correspondientes a los usuarios externos del sistema, habilitaremos la opción de recordad contraseña, y habilitaremos la opción “Remember Me”, además de permitir al usuario acceder utilizando su dirección de correo en lugar del nombre de usuario, tal y como se puede ver en la Figura 12, de este modo permitiremos al usuario mantener su sesión activa hasta que expire, este mecanismo permitirá al usuario acceder a múltiples servicios sin necesidad de introducir nuevamente las credenciales de acceso.

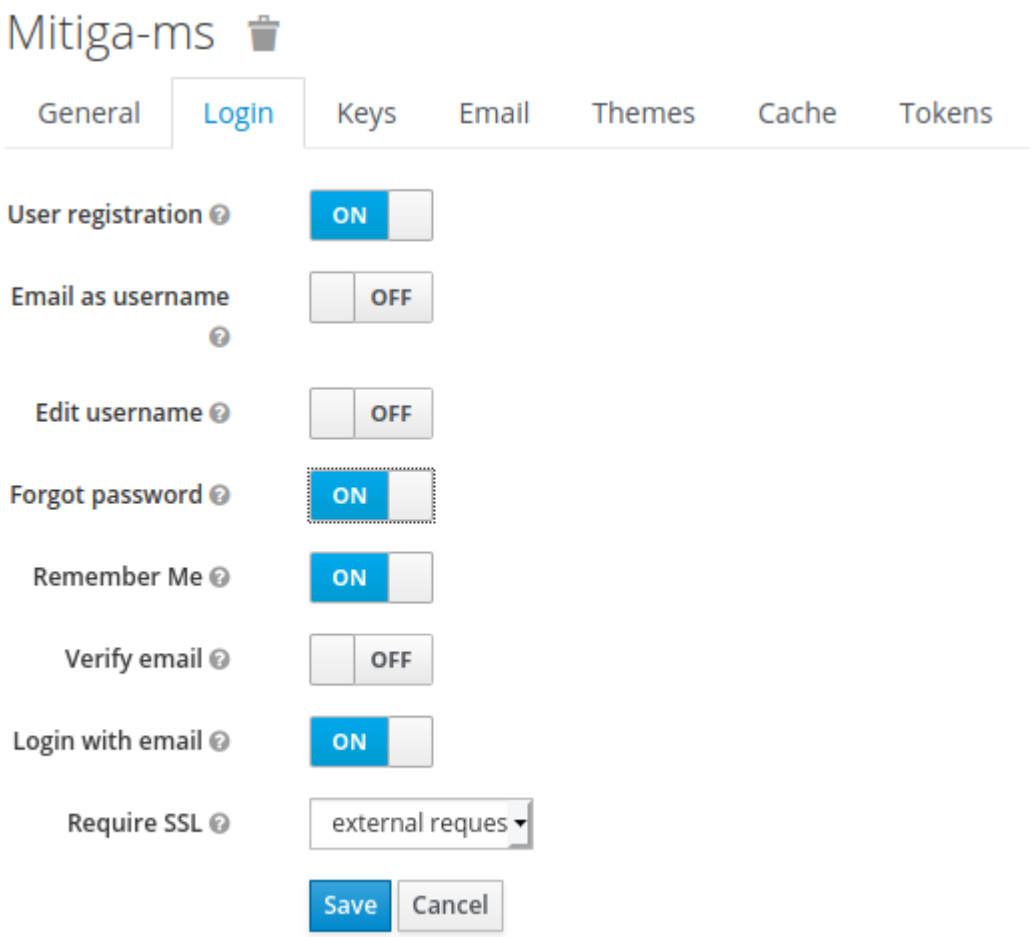


Figura 11: Definición de los parametros de acceso

4.3 Integración con LDAP.

Para el proceso de configuración del servidor utilizaremos la consola de administrador de keycloak, para ello accederemos al servidor con las credenciales generadas en el apartado anterior y una vez dentro nos situaremos en el reino creado en el apartado anterior.

El primer paso consistirá en integrar el servicio con LDAP, para ello configuraremos el mecanismo de federación de identidades para hacer uso del servidor ldap.

Los parámetros de configuración en este paso quedaran definidos tal y como se muestra en la Figura 12 y Figura 13. A continuación, detallaremos brevemente los diferentes parámetros de la configuración, siguiendo las recomendaciones del manual de integración de keycloak [10].

Establecemos el nivel de prioridad a 0, la prioridad máxima en keycloak, de esta manera nos aseguramos de que el servidor consulte los datos del servidor ldap. El siguiente atributo nos indica si queremos importar

los usuarios de ldap al servidor de keycloak, esta opción la dejaremos habilitada, de este modo podemos acceder a los datos de los usuarios directamente desde keycloak. En relación con la opción anterior, estableceremos el modo de edición a WRITEABLE y la opción Sync Registration deshabilitada. De esta manera logramos que las modificaciones realizadas sobre los usuarios de ldap se mantengan actualizadas en el servidor de LDAP, pero los usuarios externos a LDAP no se almacenen en él.

Los siguientes puntos hacen referencia a la configuración de la conexión con el servicio ldap y las consultas. En la guia se nos indica que debemos seleccionar “Other” en el proveedor de ldap si se trata de OpenLDAP, y de manera automática rellena parte de los campos a excepción de la conexión y la consulta. En estos campos establecemos la URL del servicio LDAP, y configuramos la conexión del usuario administrador.

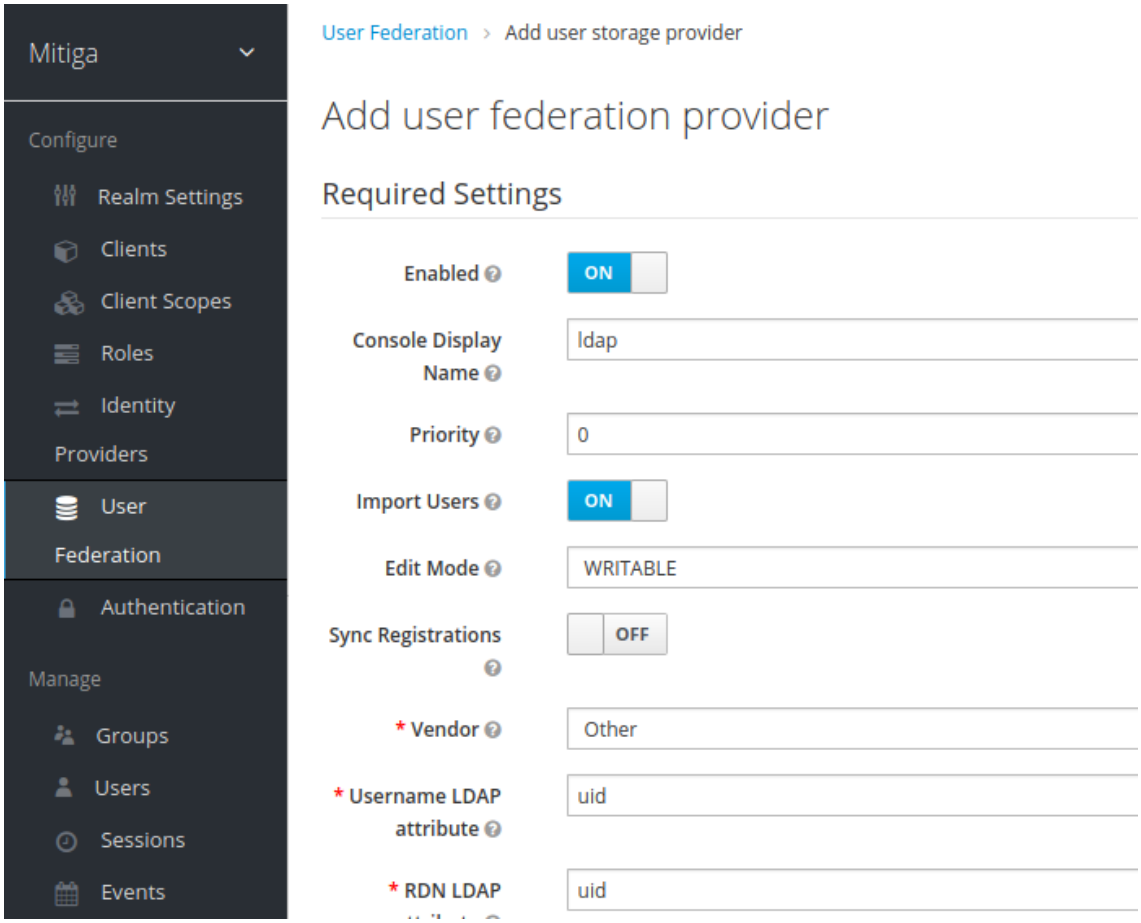


Figura 12: Configuración de ldap 1

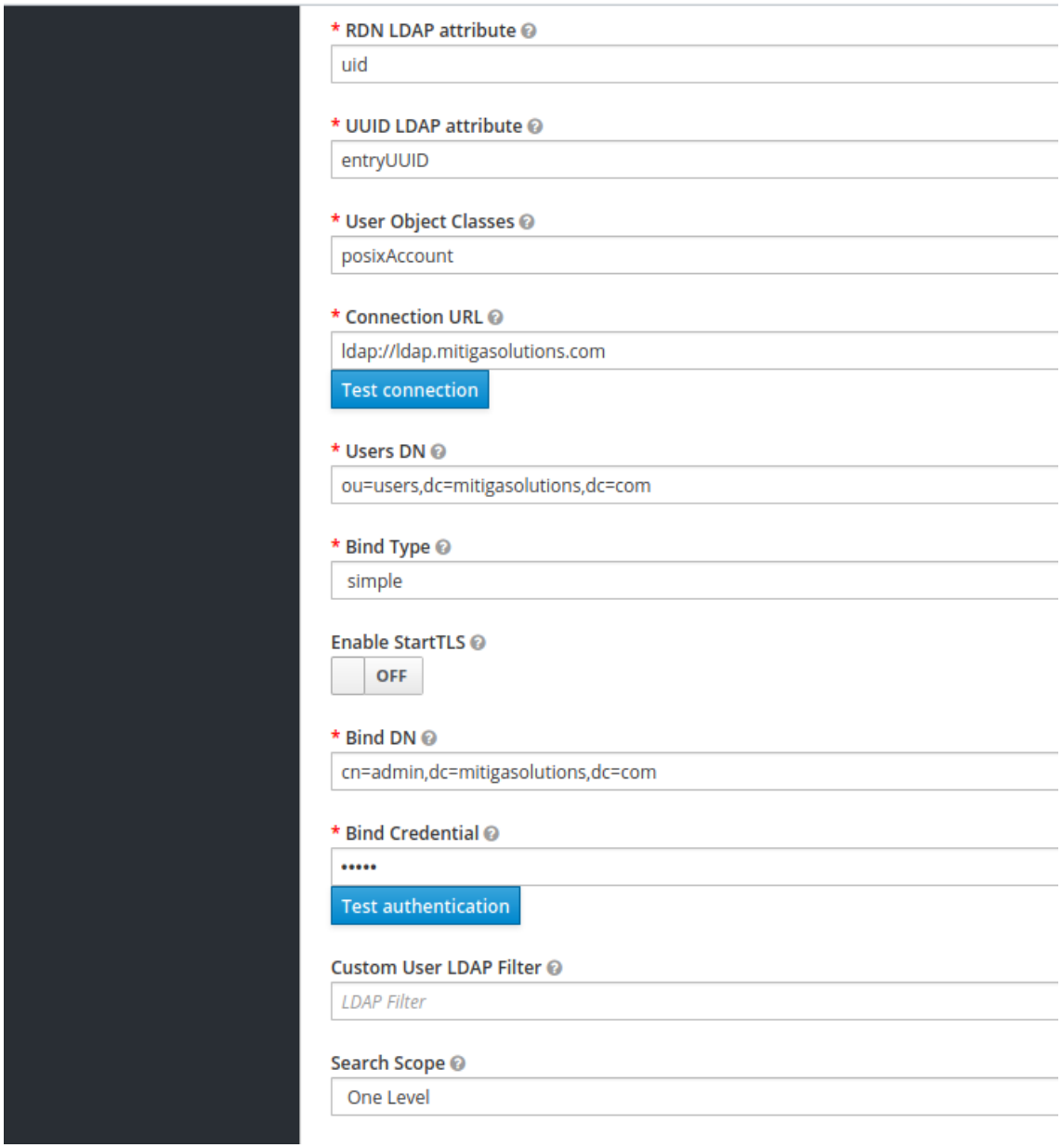


Figura 13: Configuración de ldap 2

4.4 Añadir clientes bajo demanda.

En este punto configuraremos el servidor de KeyCloak para permitir a los usuarios obtener un token OIDC bajo demanda, para ello debemos definir que es un cliente en oidc, para ello nos referiremos al contenido del rfc6749 [12], que define el cliente de la siguiente manera “Una aplicación que realiza peticiones en un recurso protegido en representación del dueño del recurso y su autorización”, por tanto, el cliente que vamos a configurar en este punto es un microservicio de prueba.

Para ello nos dirigiremos a la pestaña Clients y añadiremos un nuevo cliente de prueba al que llamaremos “dummy”.

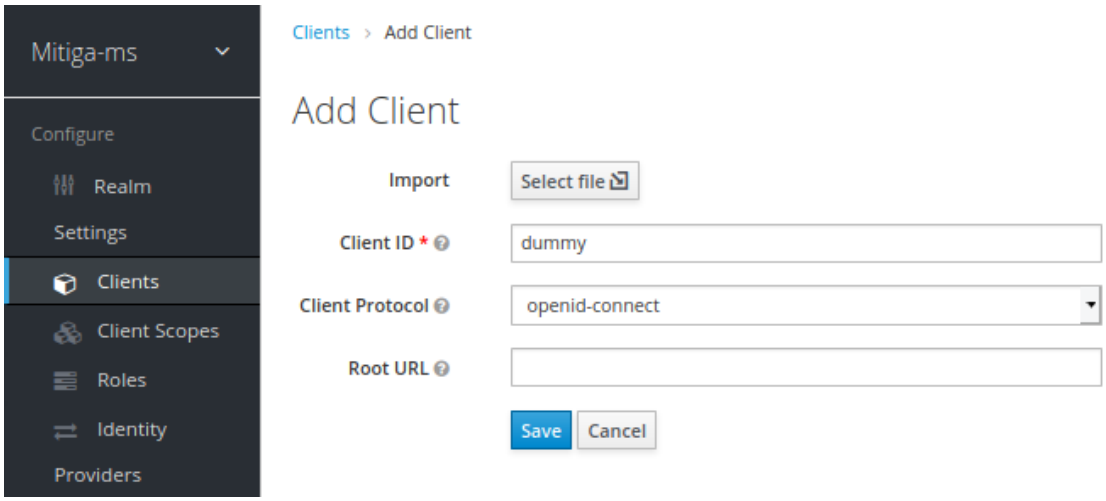


Figura 14: Creación de un nuevo cliente

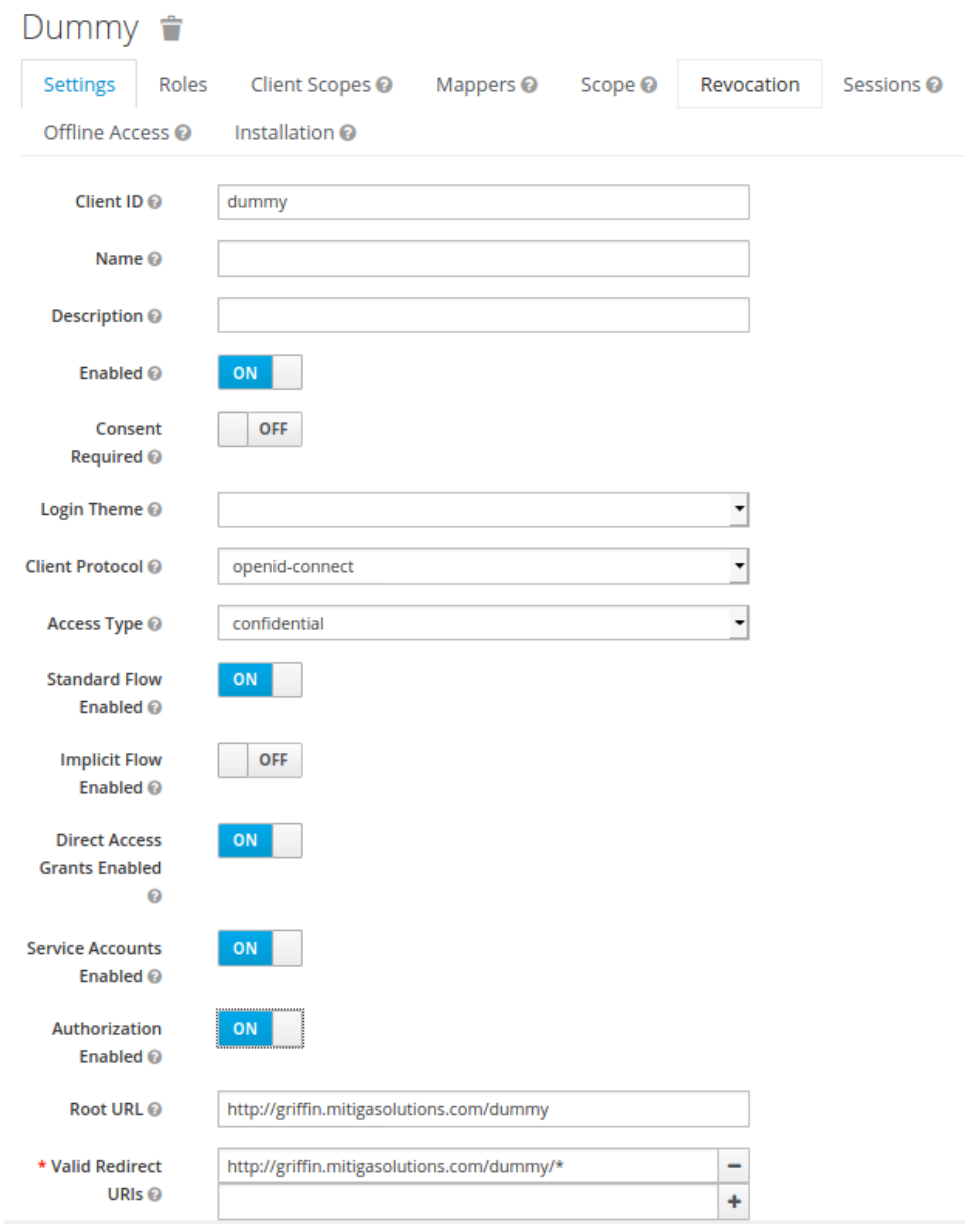


Figura 15: Configuración del cliente

Tras configurar el nuevo cliente nos desplazamos a la pestaña de Instalación donde seleccionaremos el formato “Keycloak OIDC JSON”

Al establecer el tipo de acceso a confidencial estamos permitiendo que a la hora de obtener el token se nos pidan las credenciales del usuario, este tipo de acceso nos permite la autenticación de usuarios pero no es el tipo de autneticacion que utilizaremos con los microservicios, para ellos utilizaremos el tipo “Bearer only”, el cual no realiza este conjunto de redirecciones sino que el cliente, en este caso el microservicio, se encarga de obtener su token.

Finalmente configuramos la URLs de redireccion, las cuales seran la url del servidor de KeyCloak https://griffin.mitigasolutions.com/oidc y https://griffin.mitigasolutions.com y procedemos a guardar la configuración. Para el resto de parametros dejamos las opciones por defecto.

4.5 Configuración de un microservicio con OIDC.

En este apartado crearemos un microservicio “dummy” para verificar el funcionamiendo del servidor. El funcionamiento de este microservicio es muy sencillo, y su unico proposito es el de validar la configuración.

```python
import json
import logging
import requests

from flask import Flask, g, redirect, url_for
from flask_oidc import OpenIDConnect
from oauth2client.client import OAuth2Credentials

logging.basicConfig(level=logging.DEBUG)

app = Flask(__name__)

app.config.update({
    'SECRET_KEY': 'SomeTestingKey',
    'TESTING': True,
    'DEBUG': True,
    'OIDC_CLIENT_SECRETS': 'client_secrets.json',
    'OIDC_ID_TOKEN_COOKIE_SECURE': False,
    'OIDC_REQUIRE_VERIFIED_EMAIL': False,
    'OIDC_USER_INFO_ENABLED': True,
    'OIDC_OPENID_REALM': 'mitiga-ms',
    'OIDC_SCOPES': ['openid', 'email', 'profile'],
    'OIDC_INTROSPECTION_AUTH_METHOD': 'client_secret_post'
})

oidc = OpenIDConnect(app)


@app.route('/')
def landing_page():
    if oidc.user_loggedin:
        return ('Hello, %s,</br> <a href="/private">See private</a> '
                '<a href="/logout">Log out</a>') % \
               oidc.user_getfield('preferred_username')
    else:
        return 'Welcome to Mitiga\'s Dummy MS, <a href="/private">Log in</a>'


```

Figura 16: Codigo del microservicio 1

```python
@app.route('/private')
@oidc.require_login
def private_page():
    """Ejemplo de endpoint protegido.

    Utilia el token de acceso para acceder a un servicio del backend.
    """
    info = oidc.user_getinfo(['preferred_username', 'name'])
    username = info.get('preferred_username')
    name = info.get('name')
    return ('Bienvenido %s (%s)'
            '<ul>'
            '<li><a href="/">Home</a></li>'
            '<li><a href="//griffin.mitigasolutions.com/auth/realms/'
            'mitiga-ms/account?referrer=dummy&'
            'referrer_uri=http://localhost:5000/private&'
            'redirect_uri=http://localhost:5000/logout&">Account</a></li>'
            '</ul>') % (username, name)


@app.route('/logout')
def logout():
    """Realiza un logout local eliminando la cookie de session"""
    oidc.logout()
    redirect_url = 'http://localhost:5000' + url_for('landing_page')
    return redirect(
        oidc.client_secrets.get('issuer') +
        '/protocol/openid-connect/logout?referrer=%s&redirect_uri=%s&' % \
        (oidc.client_secrets.get('token_id'),
         redirect_url)
    )


if __name__ == '__main__':
    app.run(port=5000)
```

Figura 17 Codigo del microservicio 2

1: client_secrets.json

```json
{
  "web": {
    "realm": "mitiga-ms",
    "issuer": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms",
    "auth_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/auth",
    "client_id": "dummy",
    "client_secret": "8ac76d9e-2901-407f-874b-6245687f4751",
    "redirect_uris": [
      "http://localhost:5000/"
    ],
    "userinfo_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/userinfo",
    "token_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/token",
    "token_introspection_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/token/introspect"
  }
}
```

Figura 18: Configuración del protocolo OIDC

Para validar su funcionamiento, accederemos a la pagina principal, donde deberia mostrarnos un mensaje generico, pues no conoce nuestra identidad en este punto, tal y como se puede observar en la Figura 19.

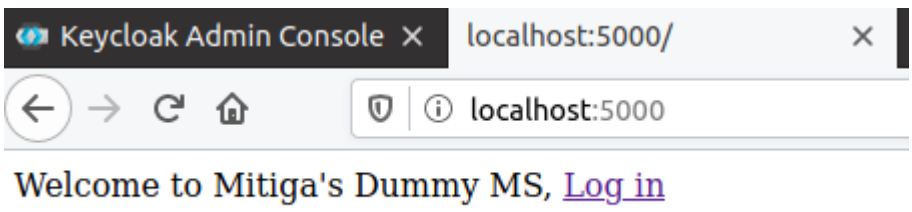


Figura 19: Pantalla de bienvenida sin autenticación

Seguidamente accederemos al endpoint “/private” donde se nos redirigira a la pagina de login del servidor de Keycloak, Figura 20, donde nos autenticaremos con nuestro usuario, que ha sido importado desde

ldap y se nos devolverá a la sección privada, tal y como podemos ver en la Figura 21.

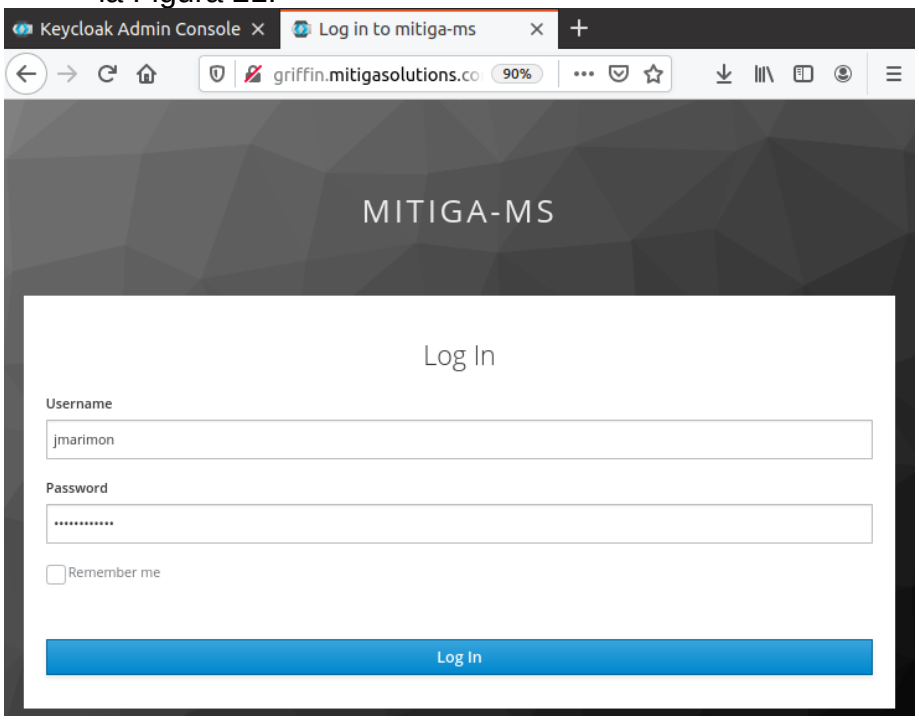


Figura 20: Formulario de acceso del proveedor de identidad

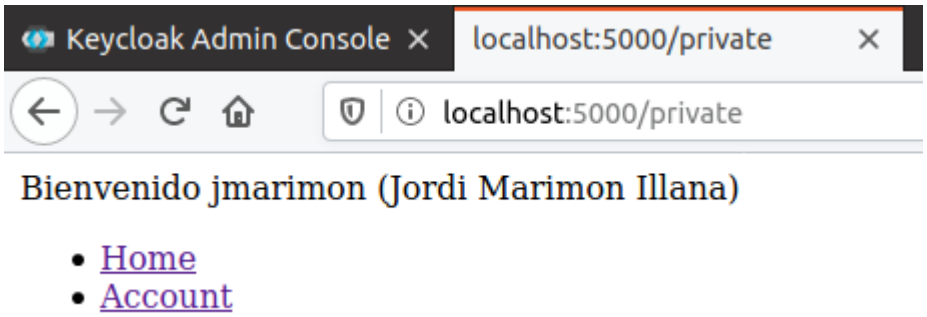


Figura 21: Pagina privada del usuario

Como ultimo paso de este proceso, validaremos el mecanismo de logout, para ello nos dirigiremos a la pagina “Home” y en ella clicaremos “log out”, Figura 22, al realizar esta acción, se estan activando una serie de mecanismos de redireccion, en el cual se lanza una peticion de logout al servidor de SSO para eliminar la session y destruir la cookie y despues se nos vuelve a redirigir a la pagina home, esta vez con la sesión cerrada, tal y como se muestra en la Figura 23.

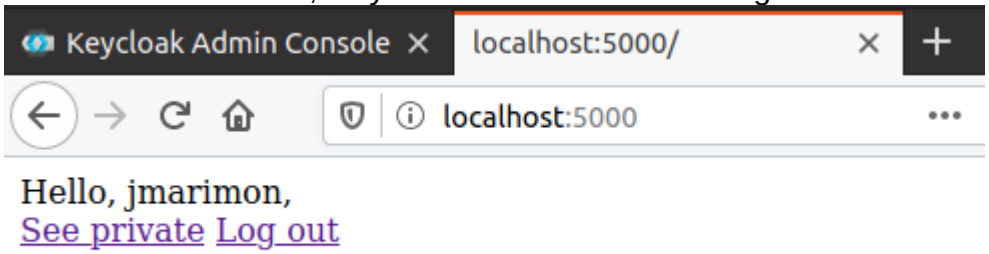


Figura 22: Pagina de bienvenida personalizada para el usuario

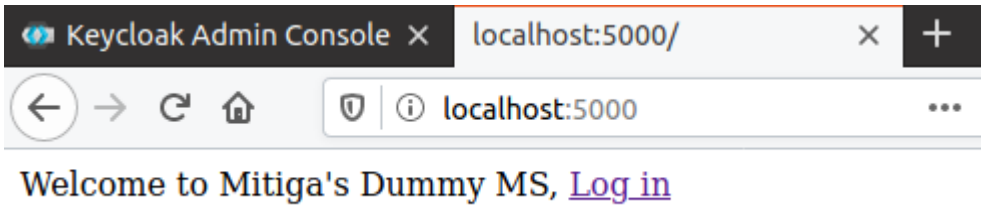


Figura 23: Pagina de bienvenida tras cerrar sesión

4.6 Modificación de la apariencia de KeyCloak.

Finalmente, el último paso a realizar es la modificación de la apariencia del servidor SSO. Para ello deberemos acceder al directorio donde se ubican los temas y realizar una copia del tema “keycloak” en el tema que queramos añadir, en este caso “mitiga” tal y como podemos ver en la Figura 24.

```
[root@fe8480bf2406 themes]# cp -r keycloak mitiga
[root@fe8480bf2406 themes]# ls
base  keycloak  keycloak-preview  mitiga  README.txt
[root@fe8480bf2406 themes]#
```

Figura 24: Creación de un nuevo tema

A continuación, accedemos al directorio del nuevo tema, donde nos encontramos el conjunto de componentes del sistema a los cuales podemos modificar el aspecto.

4.6.1 Modificación del login

Para modificar el aspecto de la página de login nos dirigiremos al directorio de login y añadiremos las imágenes que necesitemos para la creación del nuevo tema. Por último, modificamos el fichero css asociado al login para hacer uso de las nuevas imágenes y propiedades para nuestro login personalizado.

```
.login-pf body {
    background: url("../img/mitiga-bg.png") no-repeat center center fixed;
    background-size: cover;
    height: 100%;
}

#kc-logo-wrapper {
    background-image: url(../img/keycloak-logo-2.png);
    background-repeat: no-repeat;
    height: 63px;
    width: 300px;
    margin: 62px auto 0;
}

div.kc-logo-text {
    background-image: url(../img/mitiga-logo-text.png);
    background-repeat: no-repeat;
    height: 63px;
    width: 300px;
    margin: 0 auto;
}
```

Figura 25: Modificaciones del estilo del login

4.6.2 Modificación del panel de administración

Para modificar el aspecto del panel de administrador nos limitaremos a modificar las propiedades del fichero css asociado al panel de administración.

En primer lugar, modificaremos el color de la barra lateral y sus elementos para hacerlos diferenciables en el nuevo aspecto, cambiamos su color al color corporativo, y modificamos el color del texto en su interior a negro para que se pueda leer de manera fluida Figura X. y las modificaciones en la barra superior Figura Y

```
.sidebar-pf-left{
  background: #ffcc00;
}

.sidebar-pf .nav-pills > li a i, .sidebar-pf .nav-pills > li a span{
  color:  #72767b;
  display: inline-block;
  margin-right: 10px;
}
.sidebar-pf .nav-pills > li > a{
  color: #000;
  padding: 0px 20px 0 30px!important;
  line-height: 30px;
  border-left-width: 12px;
  border-left-style:  solid;
  border-left-color: #292e34;
  margin-left: -6px;
}

.sidebar-pf .nav-pills > li > a:hover{
  background: #ffeb99;
  border-color:#292e34;
  border-left-color: #393f44;
  color: #000;
}

.sidebar-pf .nav-pills > li > a:after{
  display: none!important;
}


.sidebar-pf .nav-pills > li.active > a {
  color: #000;
  background: #ffeb99!important;
  border-bottom: 1px solid #000!important;
  border-top: 1px solid #000!important;
  border-left-color: #39a5dc!important;
}
```

Figura 26: Modificación del estilo de la barra lateral

```
.navbar-pf{
  border-top: none!important;
  background-color: #e6b800!important;
}

.navbar-pf .navbar-brand {
  padding: 0;
  height: 56px;
  line-height: 56px;
  background-position: center center;
  background-image: url('../img/mitiga-logo-gscale.png');
  background-size: 148px 30px;
  background-repeat: no-repeat;
  width: 148px;
}

.navbar-pf .navbar-utility .dropdown-toggle {
    padding: 23px !important;
    color: #000 !important;
}
```

Figura 27: Modificación del estilo de la barra superior

```
.onoffswitch .onoffswitch-inner .onoffswitch-active {
    background-image: linear-gradient(top, #ffd633 0%, #ffcc00 100%);
    background-image: -o-linear-gradient(top, #ffd633 0%, #ffcc00 100%);
    background-image: -moz-linear-gradient(top, #ffd633 0%, #ffcc00 100%);
    background-image: -webkit-linear-gradient(top, #ffd633 0%, #ffcc00 100%);
    background-image: -ms-linear-gradient(top, #ffd633 0%, #ffcc00 100%);
    background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0, #ffd633),
color-stop(1, 0, #ffcc00));
    color: #000;
    padding-left: 10px;
}
```

Figura 28: Modificación del estilo de los botones

Finalmente, tras realizar las modificaciones mostradas, mostramos las páginas de login y de administración resultantes en las figuras 29 y 30.

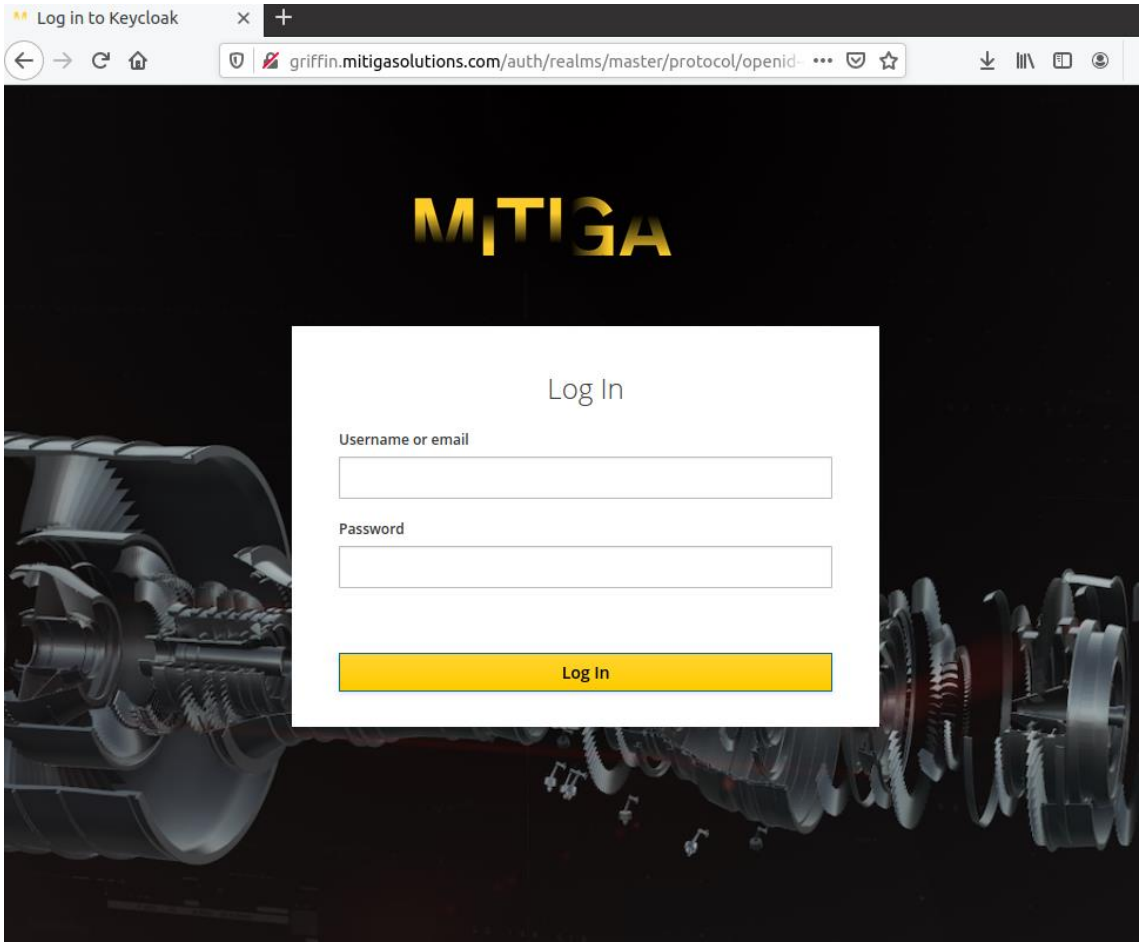


Figura 29: Página de login

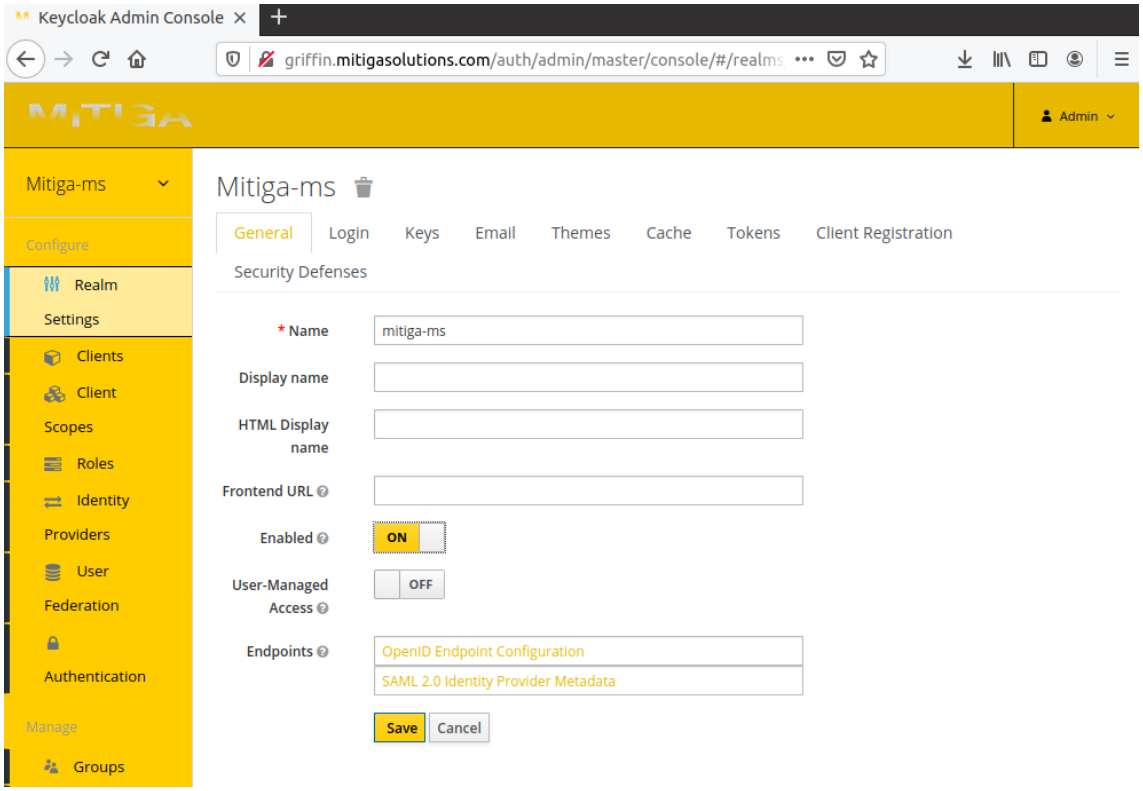


Figura 30: Panel de administrador

5. Implementación en los microservicios

En este apartado procederemos a modificar los microservicios de Mitiga Solutions. En este trabajo mostraremos únicamente 2 de los microservicios que se han modificado, el objetivo de esto es mostrar por un lado la interacción del usuario con los microservicios, así como la interacción entre microservicios. La selección de estos no ha sido arbitraria, sino que se ha buscado 2 microservicios que por sus características nos permitan mostrar los distintos mecanismos que se han comentado al largo de este documento. En primer lugar, tenemos el microservicio “Advisory”, que es el encargado de obtener los datos de eventos eruptivos a nivel global y enviar esta información al microservicio de “Ash” que se encarga de procesar estos datos y generar los resultados necesarios. El siguiente microservicio será “Ash”, el cual es el encargado de lanzar simulaciones de eventos eruptivos bajo demanda, esta demanda puede estar producida por otros microservicios, como los encargados de adquisición de datos oficiales que actúan como activador de la simulación, o bien de usuarios.

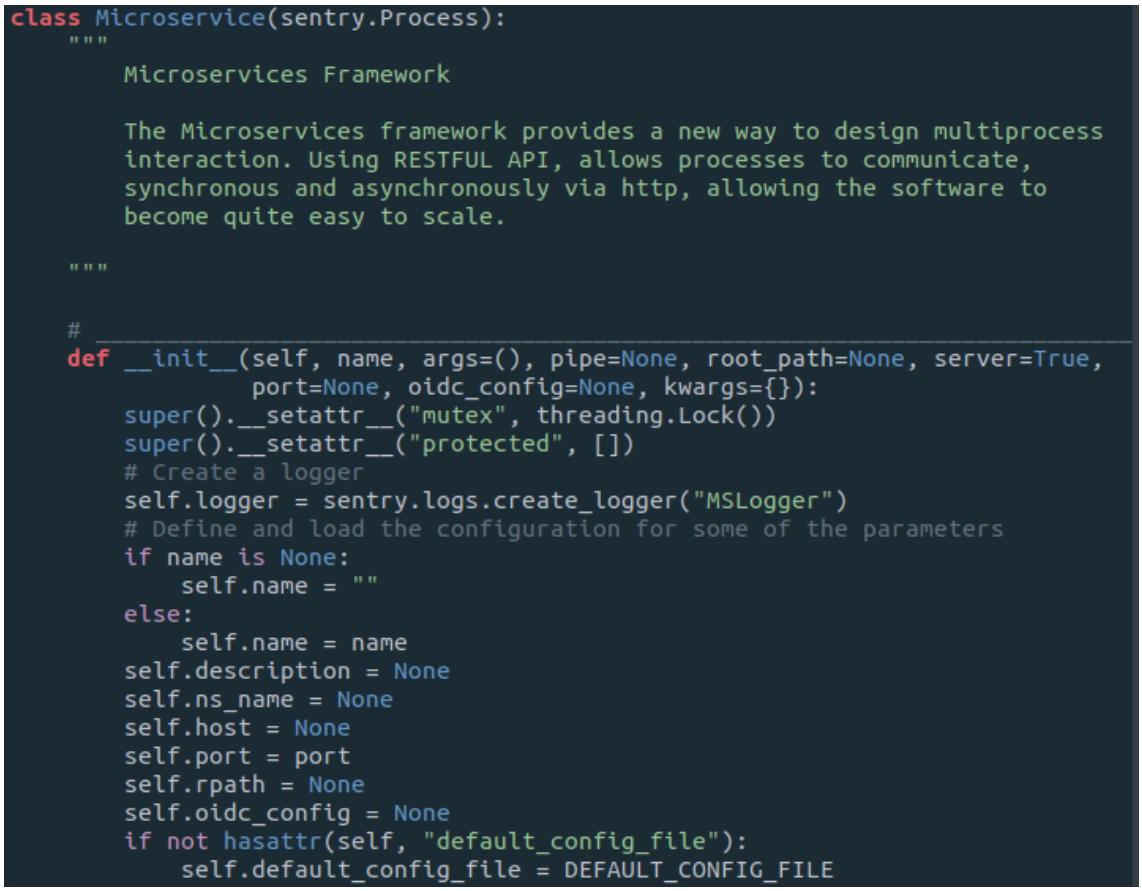
Para minimizar el código expuesto de la empresa Mitiga Solutions se utilizarán unas versiones adaptadas de dichos microservicios que nos servirán para mostrar las distintas funcionalidades disponibles con el uso de este protocolo.

En este apartado únicamente se muestra la configuración y el código implementado, pero no se visualizan los resultados evitar duplicar la información que se mostrará en el apartado 6 Validación del sistema. Para todo el código implementado en este apartado se ha utilizado la herramienta Postman [11] para la realización de las distintas peticiones a los endpoints disponibles, comprobando el uso de tokens y distintas variaciones como acceso con roles distintos.

5.1 Modificación de la librería de microservicios.

Todos los microservicios implementados se basan en una librería implementada internamente por Mitiga Solutions, la cual se basa en flask. En este apartado describiremos las modificaciones necesarias a la librería.

En primer lugar, deberemos importar el modulo flask_oidc para poder crear el objeto OpenIDConnect a partir de una aplicación flask. A continuación, añadiremos un nuevo parámetro opcional al método de inicialización de los microservicios que consistirá en el diccionario de configuración de la aplicación, que nos permitirá configurar los parámetros del objeto OpenIDConnect. En caso de no disponer de este parámetro no se realizará la creación de dicho objeto. En la Figura 31 se muestra la modificación del método “__init__” donde se le ha añadido el argumento adicional “oidc_config”, que por defecto es nulo. Este argumento será almacenado en una propiedad del objeto con el mismo nombre.



```
class Microservice(sentry.Process):
    """
        Microservices Framework

        The Microservices framework provides a new way to design multiprocess
        interaction. Using RESTFUL API, allows processes to communicate,
        synchronous and asynchronously via http, allowing the software to
        become quite easy to scale.

    """

    # ____________________________________________________________________
    def __init__(self, name, args=(), pipe=None, root_path=None, server=True,
                 port=None, oidc_config=None, kwargs={}):
        super().__setattr__("mutex", threading.Lock())
        super().__setattr__("protected", [])
        # Create a logger
        self.logger = sentry.logs.create_logger("MSLogger")
        # Define and load the configuration for some of the parameters
        if name is None:
            self.name = ""
        else:
            self.name = name
        self.description = None
        self.ns_name = None
        self.host = None
        self.port = port
        self.rpath = None
        self.oidc_config = None
        if not hasattr(self, "default_config_file"):
            self.default_config_file = DEFAULT_CONFIG_FILE
```

Figura 31: Inicialización del microservicio

El siguiente paso consiste en la creación del objeto OpenIDConnect, pero solo en caso de ser necesario, es decir, si se ha utilizado el argumento opcional “oidc_config”, para ello, tras la creación del objeto del tipo Flask, se llamará a la función encargada de habilitar este objeto, tal y como podemos ver en la Figura 32.

```
# ____________________________________________________________________
def _enable_oidc(self):
    """Enables OpenIDConnect protocol.

    Load the configuration for the OpenIDConnect protocol and
    creates a `flask_oidc.OpenIDConnect` Object

    Returns:
        (boolean): True if OpenIDConnect object is created, False otherways
    """
    enabled = False
    if self.oidc_config:
        # Load OpenIDConnect configuration.
        self.app.config.update(self.oidc_config)
        self.oidc = OpenIDConnect(self.app)
        enabled = True
    return enabled
```

Figura 32: Habilitar el protocolo OIDC

Para hacer más cómoda y reducir errores de escritura a la hora de definir los parámetros de configuración del protocolo crearemos una enumeración en la librería de los microservicios que contenga las claves del diccionario.

```python
class OIDC(Enum):
    """Define OpenIDConnect configuration keys."""
    SECRET_KEY = 'SECRET_KEY'
    TESTION = 'TESTING'
    DEBUG = 'DEBUG'
    OIDC_CLIENT_SECRETS = 'OIDC_CLIENT_SECRETS'
    OIDC_SCOPES = 'OIDC_SCOPES'
    OIDC_GOOGLE_APPS_DOMAIN = 'OIDC_GOOGLE_APPS_DOMAIN'
    OIDC_ID_TOKEN_COOKIE_NAME = 'OIDC_ID_TOKEN_COOKIE_NAME'
    OIDC_ID_TOKEN_COOKIE_PATH = 'OIDC_ID_TOKEN_COOKIE_PATH'
    OIDC_ID_TOKEN_COOKIE_TTL = 'OIDC_ID_TOKEN_COOKIE_TTL'
    OIDC_COOKIE_SECURE = 'OIDC_COOKIE_SECURE'
    OIDC_VALID_ISSUERS = 'OIDC_VALID_ISSUERS'
    OIDC_CLOCK_SKEW = 'OIDC_CLOCK_SKEW'
    OIDC_REQUIRE_VERIFIED_EMAIL = 'OIDC_REQUIRE_VERIFIED_EMAIL'
    OIDC_OPENID_REALM = 'OIDC_OPENID_REALM'
    OIDC_USER_INFO_ENABLED = 'OIDC_USER_INFO_ENABLED'
    OIDC_CALLBACK_ROUTE = 'OIDC_CALLBACK_ROUTE'
    OVERWRITE_REDIRECT_URI = 'OVERWRITE_REDIRECT_URI'
    OIDC_RESOURCE_SERVER_ONLY = 'OIDC_RESOURCE_SERVER_ONLY'
    OIDC_RESOURCE_CHECK_AUD = 'OIDC_RESOURCE_CHECK_AUD'
    OIDC_INTROSPECTION_AUTH_METHOD = 'OIDC_INTROSPECTION_AUTH_METHOD'
```

Figura 33: Enumeración de los parámetros disponibles.

Otro elemento muy interesante que podemos habilitar indistintamente para todos los microservicios que se pretendan implementar son los métodos necesarios para comprobar los permisos del usuario, para ello crearemos 2 métodos específicos, uno para validar los roles que tiene asignado un usuario dentro de un reino concreto y otro para acceder a los datos globales de un usuario concreto, dichos métodos se encuentran en la Figura 34.

```python
# ____________________________________________________________________
def user_in_realm_role(self, token_info, role):
    """Check if a user belongs to a realm role.

    Args:
        token_info (dict): Dictionary with OIDC token info.
        role (str): Role to be checked

    Returns:
        (bool): True if the user belongs to the role.
    """
    return \
        role.upper() in map(str.upper, token_info['realm_access']['roles'])

# ____________________________________________________________________
def user_in_resource_role(self, token_info, role):
    """Check if a user can access to a resource.

    Args:
        token_info (dict): Dictionary with OIDC token info.
        role (str): Role to be checked

    Returns:
        (bool): True if the user belongs to the role.
    """
    return role.upper() in map(
        str.upper, token_info['ressource_access']['account']['roles'])
```

Figura 34: Comprobación de roles

Puesto que será necesaria la comunicación autenticada entre distintos microservicios, deberemos habilitar también el mecanismo para la obtención de los tokens de los clientes a partir del secreto del cliente. Para ello se realizará una petición del tipo "grant_type: client_credentials", de manera que el servidor de SSO nos devolverá un token si el cliente es capaz de demostrar su identidad, es decir, si dispone de un secreto valido. En la Figura 35 se muestra como los microservicios construirán la petición.

```
#
def get_client_token(self):
    """Get the client token.

    Connect to the SSO server in order to retrieve a valid oidc token.
    """
    headers = {'Content-Type': 'application/x-www-form-urlencoded'}
    data = {
        "grant_type": "client_credentials",
        "client_id": self.oidc.client_secrets["client_id"],
        "client_secet": self.oidc.client_secrets["client_secret"],
    }
    r = requests.post(self.oidc.client_secrets["token_uri"],
                      headers=headers, data=data)
    if r.status_code != 200:
        return None
    return r.json()
```

Figura 35: Obtención del token del cliente

5.2 Modificación del microservicio "Advisory".

En este apartado veremos el proceso de configuración del microservicio de "Advisory" teniendo ya en cuenta las modificaciones que realizaremos también en Ash.

En primer lugar deberemos habilitar el uso del protocolo OIDC en este microservicio, para ello será necesario añadir el fichero "client_secrets.json" en el microservicio, y definir la configuración del servicio.

client_secrets.json

```
{
  "web": {
    "realm": "mitiga-ms",
    "issuer": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms",
    "auth_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/auth",
    "client_id": "advisory",
    "client_secret": "9b89c32f-4324-4cb1-bb2a-a5f7d6765d07",
    "redirect_uris": [
      "http://advisory.mitigasolutions.com:8000/"
    ],
    "userinfo_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/userinfo",
    "token_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/token",
    "token_introspection_uri": "http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/token/introspect"
  }
}
```

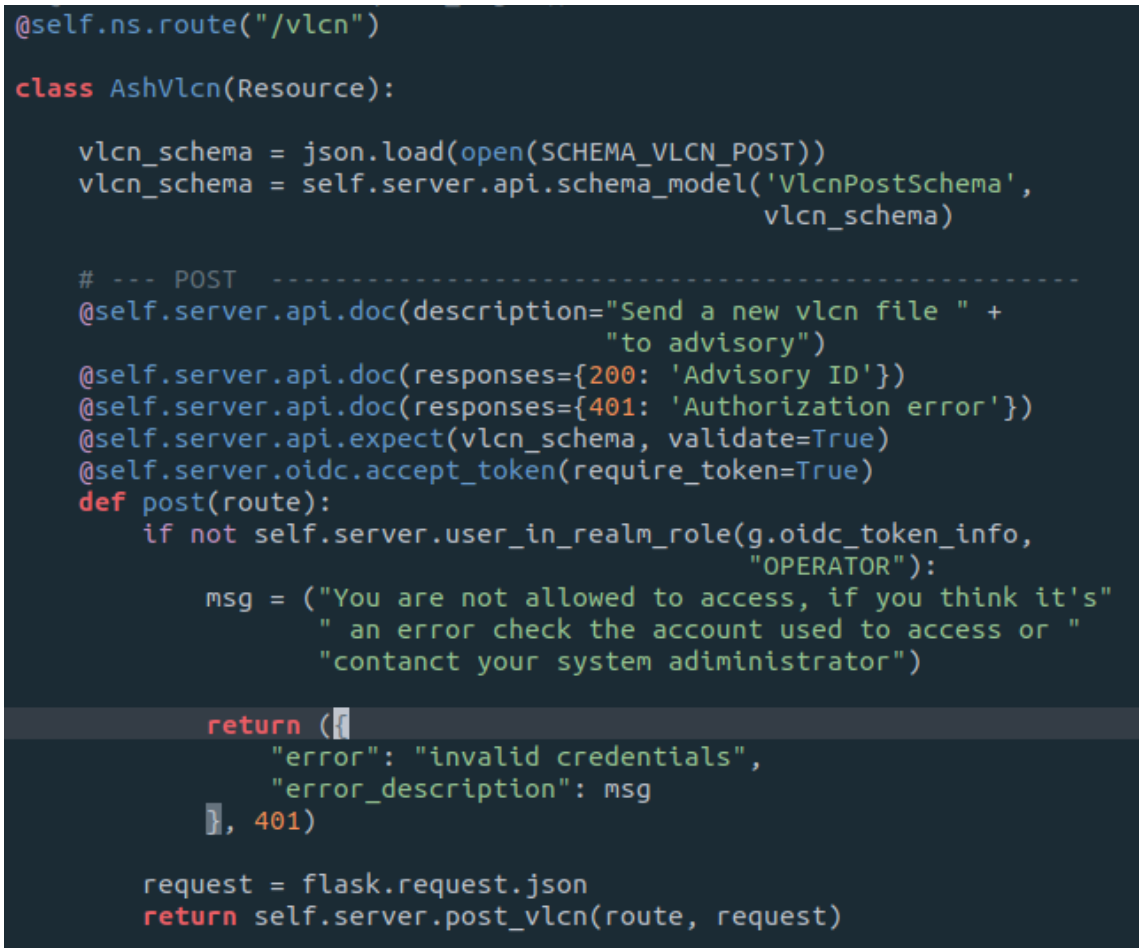
Figura 36: Fichero de configuración del protocolo OIDC

```
def __init__(self, name: str, args: typing.Tuple = (),
             pipe: None = None, root_path: None = None,
             server: bool = True, port: None = None,
             oidc_config: dict = {}, kwargs: dict = {}) -> None:
            ):
    """Constructor.

    Args:
        name(str): Microservice name
        args(tuple): List of arguments to the microservice
        pipe(pipe): Pipe to comunicate with the parent if any, default None
        root_path(str): root path for the ms to work (deprecated!)
        server(bool): True to start the server, default True
        port(int): Port number where to start the server (deprecated!)
        oidc_config(dict): OpenIDConnect configuration
        kwargs(dict): Key arguments to send to the ms
    """
    oidc_config = {
        microservices.OIDC.SECRET_KEY.value = 'smknlc!3324-1da12.',
        microservices.OIDC.TESTING.value = True,
        microservices.OIDC.DEBUG.value = True,
        microservices.OIDC.OIDC_CLIENT_SECRETS.value = \
            'config/client_secrets.json',
        microservices.OIDC.OIDC_CLIENT_SECRETS.value = False,
        microservices.OIDC.OIDC_REQUIRE_VERIFIED_EMAIL.value = False,
        microservices.OIDC.OIDC_USER_INFO_ENABLED.value = True,
        microservices.OIDC.OIDC_OPENID_REALM.value = 'mitiga-ms',
        microservices.OIDC.OIDC_SCOPES = ['openid', 'email', 'profile']
        microservices.OIDC.OIDC_INTROSPECTION_AUTH_METHOD.value = \
            'client_secret_post'
    }
    # Call the superclass constructor
    super().__init__(name, args, pipe, root_path, server, port,
                     oidc_config, kwargs)
```

Figura 37: Añadiendo la configuración de OIDC.

Con estos cambios el microservicio ya está listo para utilizar el protocolo OIDC, el siguiente paso es definir que endpoints deben estar protegidos, para ello es importante ver cuales son las funcionalidades principales de este microservicio, por un lado, obtiene los eventos de distintas fuentes de manera automática. Los endpoints públicos de este sistema permiten obtener información de los eventos que conoce el sistema, al ser un endpoint público, no es necesario utilizar el token para acceder a él, pero uno de los endpoints permite a un operador acceder introducir manualmente nueva información. Este endpoint es más crítico, puesto que no se puede permitir que cualquier usuario introduzca datos en el sistema, puesto que existe el riesgo de obtener información fraudulenta.

En este punto utilizaremos el decorador accept_token de la librería flask_oidc para hacer obligatorio el uso del token, y utilizaremos la función previamente definida para comprobar si el usuario pertenece al rol “OPERADOR”, de esta manera lograremos que un usuario básico, pero legitimo del sistema no pueda introducir información, pero, sin embargo, un operador, podrá utilizar su token para introducir datos.

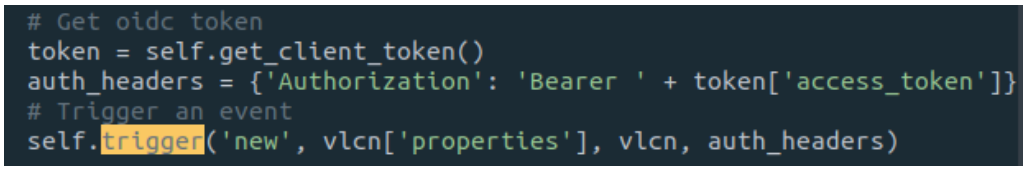


```
@self.ns.route("/vlcn")

class AshVlcn(Resource):

    vlcn_schema = json.load(open(SCHEMA_VLCN_POST))
    vlcn_schema = self.server.api.schema_model('VlcnPostSchema',
                                               vlcn_schema)

    # --- POST  ------------------------------------------------
    @self.server.api.doc(description="Send a new vlcn file " +
                                     "to advisory")
    @self.server.api.doc(responses={200: 'Advisory ID'})
    @self.server.api.doc(responses={401: 'Authorization error'})
    @self.server.api.expect(vlcn_schema, validate=True)
    @self.server.oidc.accept_token(require_token=True)
    def post(route):
        if not self.server.user_in_realm_role(g.oidc_token_info,
                                              "OPERATOR"):
            msg = ("You are not allowed to access, if you think it's"
                   " an error check the account used to access or "
                   "contanct your system administrator")

            return ({
                "error": "invalid credentials",
                "error_description": msg
            }, 401)

        request = flask.request.json
        return self.server.post_vlcn(route, request)
```

Figura 38: Securización de un endpoint mediante tokens

En el codigo que se muestra en la Figura 38 muestra el endpoint que permite introducir nueva información acerca eventos eruptivos en el sistema, donde unicamente se atienden las peticiones que contengan un token valido, que ademas pertecenza a un usuario con el rol de operador.

El siguiente punto a tratar es la comunicación entre microservicios, puesto que los datos que “Advisory” decide que información debe llegar a “Ash”, deberemos habilitar un mecanismo mediante el cual este pueda enviar su token junto con las peticiones al microservicio de “Ash”. Para ello “Advisory” generará un token con el mecanismo que hemos habilitado en la sección anterior. Todas las peticiones que advisory deba hacer a otro microservicio demostrando su identidad, las hara utilizando este token, como podemos ver en la Figura 39, advisory genera un evento mediante un trigger, añadiendo la cabezera de autenticación en el, en la Figura 40 podemos ver como esta implementado el mecanismo de triggering de la librería de microservicios.



```
# Get oidc token
token = self.get_client_token()
auth_headers = {'Authorization': 'Bearer ' + token['access_token']}
# Trigger an event
self.trigger('new', vlcn['properties'], vlcn, auth_headers)
```

Figura 39: Autorizando los triggers.

```
#
def trigger(self, event, trigger_filter, data={}, header=None):
    """
        Executes a trigger. This should be used whenever a long event
        finishes. It will send an HTTP message to the urls defined in the
        trigger.

        Args:
            event (str):
                Name of the event of which you want to trigger.

            trigger_filter(dict):
                Filter for the event that you want to trigger.

    """

    if self.triggers:
        for uid, trigger in self.triggers.items():
            if compare_dicts(trigger[FILTER], trigger_filter) and\
                    trigger[EVENT] == event:
                headers = {'Content-Type': 'application/json'}
                if header:
                    headers.update(header)
                for url in (trigger[URL]):
                    try:
                        r = requests.post(url, headers=headers,
                                          json=data, timeout=60)
                        if not r.ok:
                            self.logger.error(REQUEST_UNSUCCESSFUL,
                                              url)
                        else:
                            self.logger.info(
                                "Triggered url " + url + "for event: " +
                                event
                            )

                    # In case it fail the service should not crash
                    except Exception:
                        self.logger.error(REQUEST_ERROR)
                        self.logger.error(traceback.format_exc())
    else:
        self.logger.info(NO_TRIGGERS_AVAILABLE)
```

Figura 40: Mecanismo de Trigger.

Con estas modificaciones, “Advisory” es ahora capaz de combinar sus endpoints junto a Keycloak mediante el protocolo OIDC, limitando el acceso a los distintos usuarios en función del grupo al que pertenecen y es capaz de realizar peticiones autorizadas a otros microservicios en caso de ser necesario.

5.3 Modificación del microservicio “Ash”:

En este apartado veremos el proceso de configuración del microservicio de “Ash”

Tal y como hemos realizado con “Advisory”, en primer lugar, configuraremos el microservicio para utilizar el protocolo OIDC. Tras realzar las mismas modificaciones que las realizadas en “Advisory”, procederemos a analizar y decidir cuáles son los endpoints que deben

ser protegidos. Este microservicio es el encargado de convertir la información que proviene de advisoy en predicciones de la dispersión de la ceniza volcánica emitida durante una erupción a través del modelo FALL3D desarrollado en el Centro de Supercomputación de Barcelona. Este microservicio además permite a los usuarios realizar simulaciones con parámetros personalizados para generar estas simulaciones y por último, nos permite obtener datos históricos acerca de los distintos eventos eruptivos que se han producido en los últimos años y las simulaciones realizadas.

En este caso, todos los endpoints de este microservicio estarán protegidos, por un lado, tenemos la obtención de datos, que únicamente la podrán realizar algún tipo de usuario especifico, limitando el acceso de los usuarios básicos, por otro lado, los endpoints que se encargan de crear nuevas simulaciones estarán protegidos, para que únicamente un grupo específico de usuarios lo pueda utilizar. Además, este microservicio cuenta con una interfaz web, con un formulario limitado para la generación de simulaciones al cual cualquier usuario registrado debe poder acceder.

Puesto que ya disponemos de un ejemplo de la securización de los endpoints mediante el uso del decorador “accept_token”, en el cual realizábamos una comprobación del grupo al cual pertenecía el usuario, es este apartado describiremos únicamente la parte en la que se protege el acceso a la interfaz web mediante el decorador “require_login”. La función de este decorador es la de redirigir al usuario hasta la página de acceso del proveedor de identidad en caso de no disponer de un token y una vez se ha autenticado se redirige nuevamente a la página que se pretendía acceder.

```
@self.app.route('/web/<path:path>')
@self.oidc.require_login()
def send_js(path):
    print(WEB_ROOT_PATH + path)
    return flask.send_from_directory(
        WEB_ROOT_PATH, path)
```

Figura 41: Inicio de sesión obligatorio.

5.4 Implementación del mecanismo de renovación de tokens.

El desarrollo del mecanismo de renovación de tokens se ha descartado tras analizar el funcionamiento de los microservicios. Se ha observado que la propia librería Flask-OIDC está gestionando de manera automática la actualización de los tokens, obteniendo un nuevo token cada vez que el token ha expirado siempre y cuando se encuentre dentro de la ventana del token de actualización.

5.5 Implementación del mecanismo de cierre de sesión

El este apartado se añadirá el método de logout en la librería de microservicios. Este mecanismo se encargará de eliminar los datos

asociados a la sesión del usuario y realizará el cierre de sesión en el servidor de SSO.

Tal y como se muestra en la Figura 42, el proceso de cierre de sesión requiere de una dirección a la que redirigir al usuario tras cerrar sesión en el servidor de SSO.

```
# _____________________________________________________________________
def logout(self, redirect_url):
    """Close user session.

    Logs out from the oidc library, clears the session data and performs
    a logout from the SSO server.

    Args:
        redirect_url (str): Url to get redirected to after logout.
    """
    if not self.oidc_config:
        return False

    self.oidc.logout()
    session.clear()
    return redirect(
        self.oidc.client_secrets.get('issuer') +
        'protocol/openid-connect/logut?referrer=%sredirect_uri=%s' % \
        (self.oidc.client_secrets.get('token_id'), redirect_url)
    )
```

Figura 42: Mecanismo de cierre de sesión.

6. Validación

En este sexto apartado del proyecto veremos la validación de los distintos mecanismos implementados en el transcurso del proyecto. Para ello levantaremos los microservicios de “Advisoy” y “Ash” para realizar un conjunto de pruebas en algunos de los distintos endpoints. La realización de estas pruebas se realizará mediante la herramienta Postman, una aplicación que permite definir peticiones http, permitiendo definir todos los parámetros. Un punto para destacar de esta aplicación es la posibilidad de obtener los tokens OIDC de manera muy sencilla, abriéndose en una ventana la página de inicio de sesión del sistema de SSO.

Prueba 1:
En esta primera prueba nos centraremos en 3 peticiones distintas al microservicio de advisory, cubriendo las necesidades del endpoint POST /ash/vlcn, encargado de recibir los datos introducidos por un operador.

En el primer caso, crearemos una nueva petición en postman, en la cual no introduciremos ningún token, esperando que, por tanto, la aplicación nos devuelva un mensaje indicando el error de Autorización.

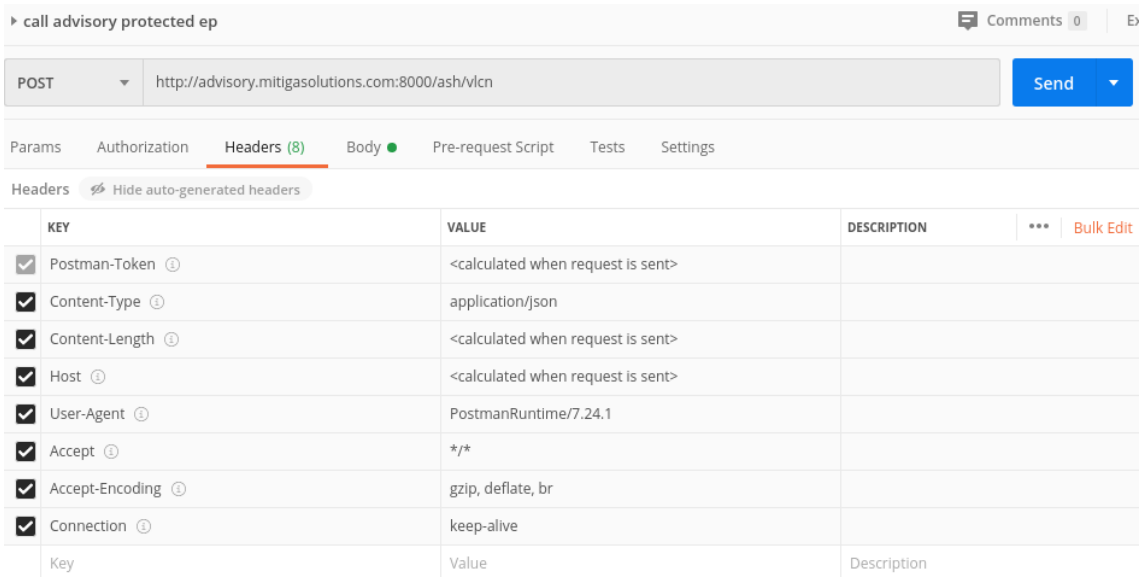


Figura 43: Cabecera de la petición

```
{
  "type": "Feature",
  "geometry": {
    "coordinates": [],
    "type": "MultiPolygon"
  },
  "properties": {
    "ch_correction_factor": 0.8,
    "eruption_color_code": "ORANGE",
    "eruption_column_height": 2000,
    "eruption_end_t": "20191030124715",
    "eruption_finished": false,
    "eruption_granulometry": "string",
    "eruption_start_t": "20191030150000",
    "event_t": "20191030150000",
    "polyfon_concentration": [],
    "polygon_fl": [
      50,
      100
    ],
    "polygon_t": [
      "20191030150000",
      "20191030150000"
    ],
    "source_id": "Manual",
    "source_img": "",
    "source_type": "vaac",
    "source_url": "",
    "volcano_altitude": 3000,
    "volcano_area": [
      "darwin"
    ],
    "volcano_id": "268010",
    "volcano_name": "Dukono",
    "volcano_position": [
      127.88,
      1.68
    ]
  }
}
```


Figura 44: Cuerpo de la petición

Tras enviar la petición, recibimos el siguiente error de autenticación, en la Figura 45, lo cual nos indica que, a priori el sistema está funcionando correctamente.

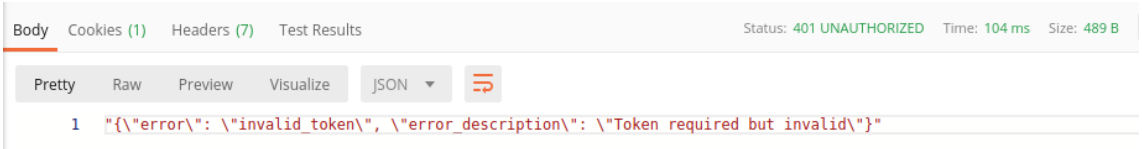


Figura 45: Petición rechazada

La siguiente prueba, para comprobar si el funcionamiento es correcto, será realizar la misma petición, pero esta vez utilizaremos un token de usuario, correspondiente a un usuario que no pertenezca al grupo “OPERADOR”, recordemos que, tal y como se ha visto en el apartado 5, únicamente los operadores deben poder acceder de manera satisfactoria a este endpoint. En la Figura 466 se muestra cómo obtener el token de acceso en Postman, para ello se debe acceder a “Get New Access Token” y configurarlo tal y como se muestra en la Figura 4747.

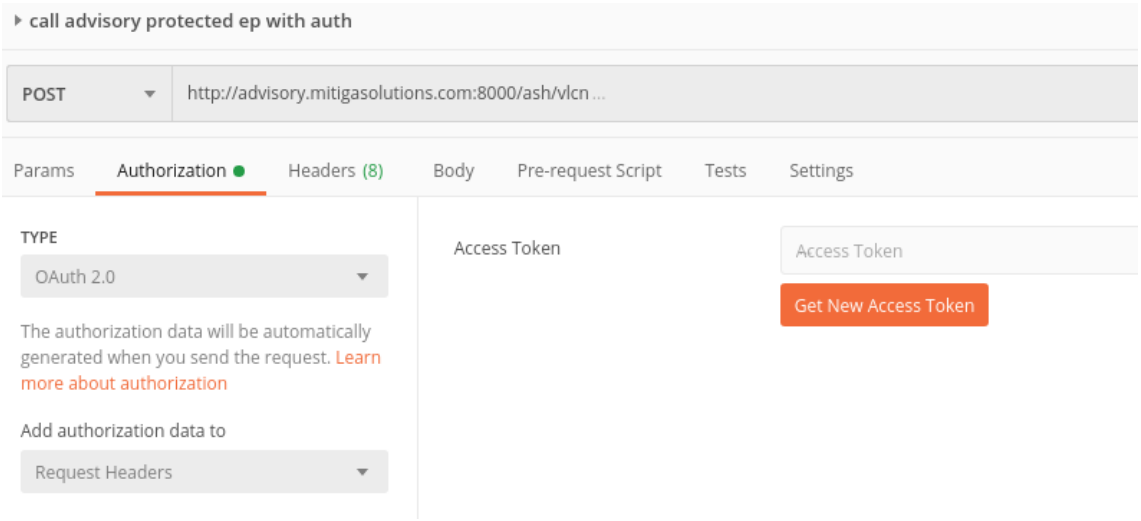


Figura 46: Configuración del token de usuario.

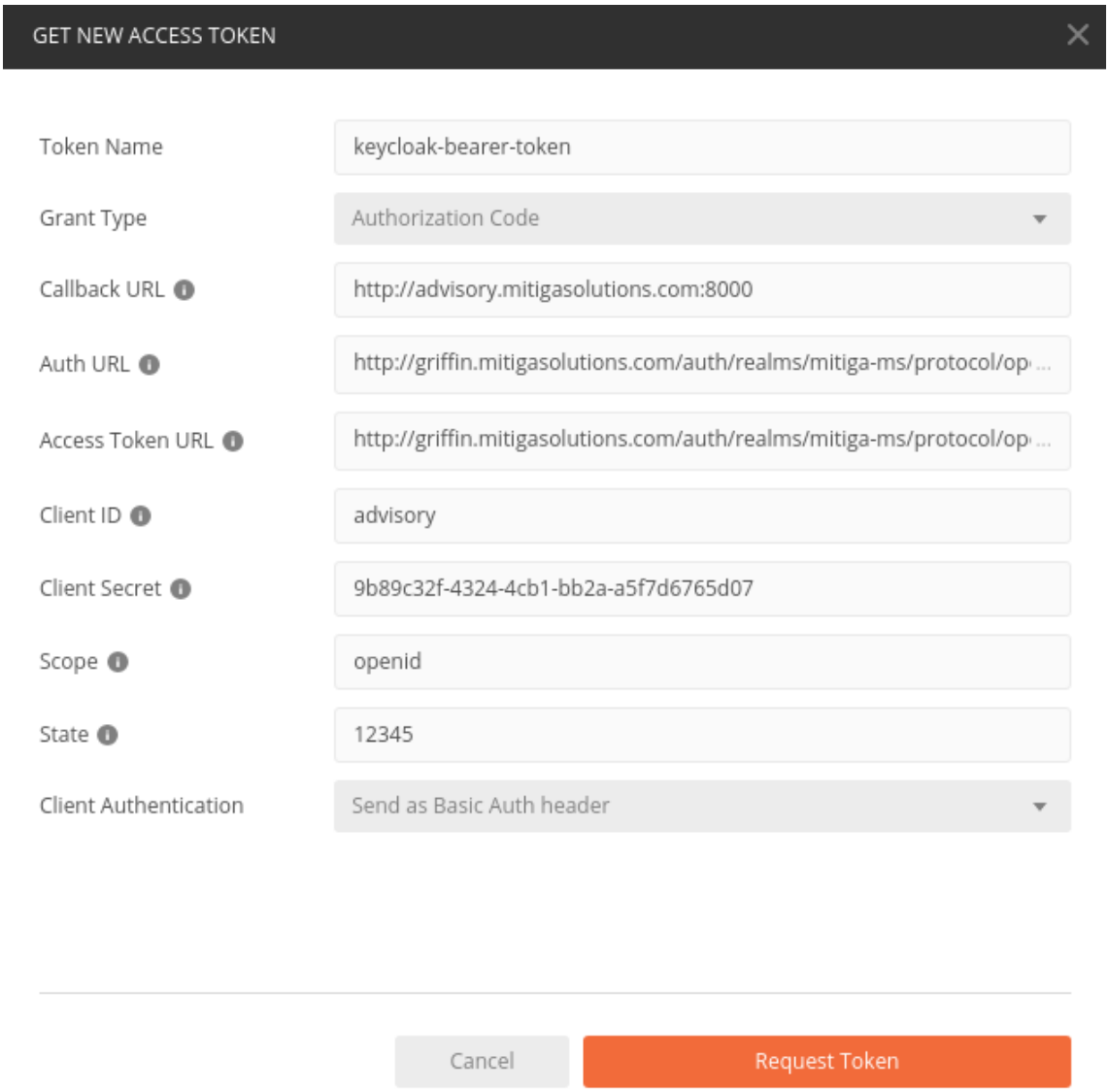


Figura 47: Configuración de la petición de token

Una vez le demos a “request token” se nos redirige al servicio de login de keycloak, donde introduciremos las credenciales, tal y como se muestra en la Figura 488 y tras acceder obtenemos el token que se muestra en la Figura 499.

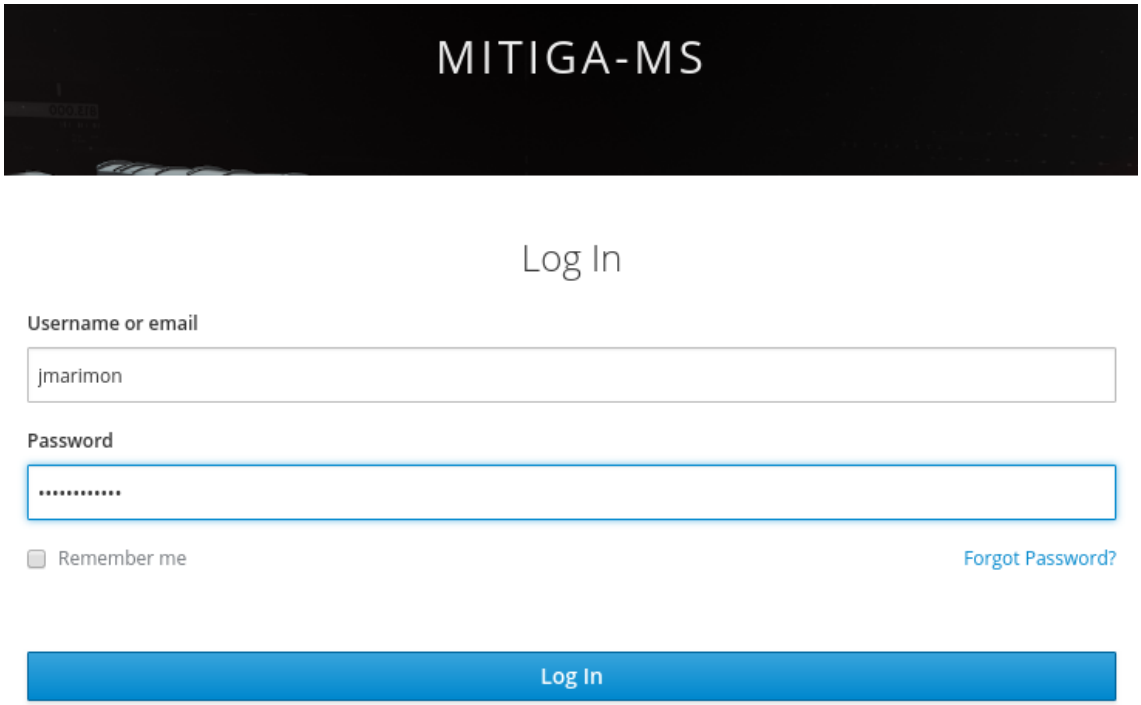


Figura 48: Login como usuario basico

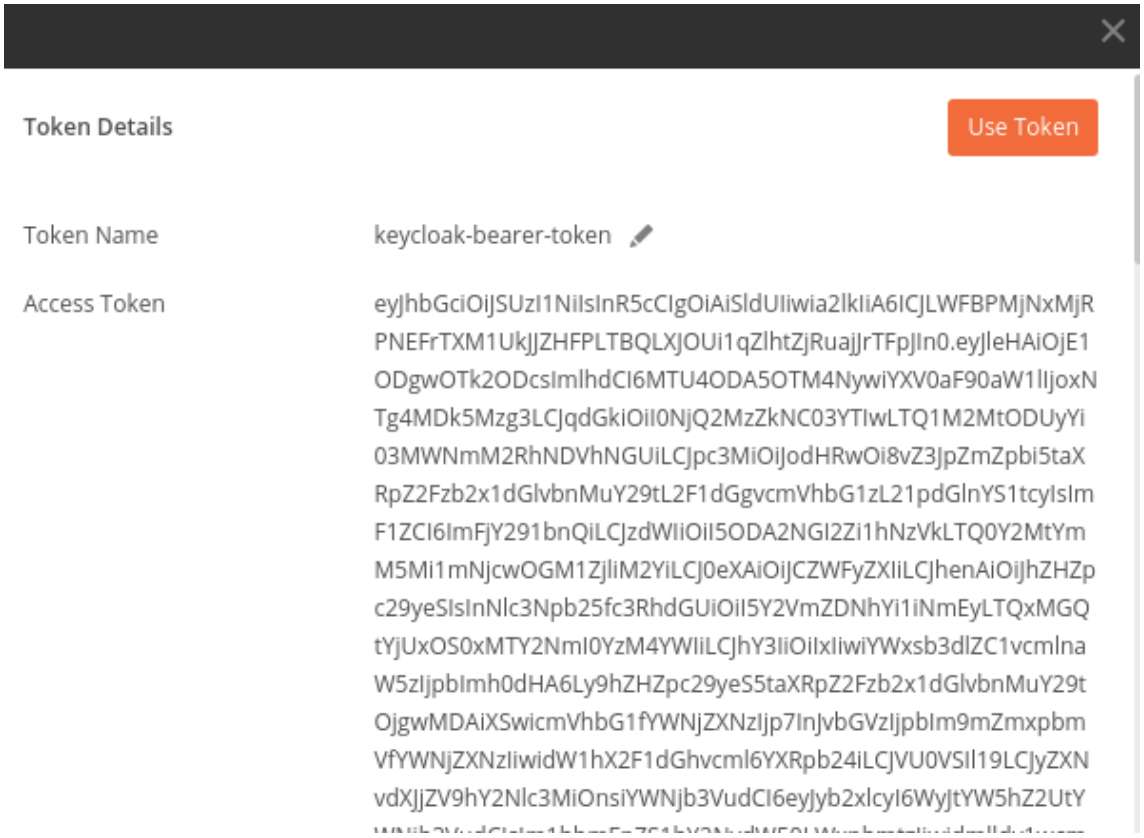


Figura 49: Token de usuario.

Con la nueva petición ya construida realizamos la petición a “Advisory” y obtenemos, nuevamente, y tal como esperábamos un mensaje de error de autorización, pero esta vez se corresponde al mensaje personalizado que habíamos introducido.

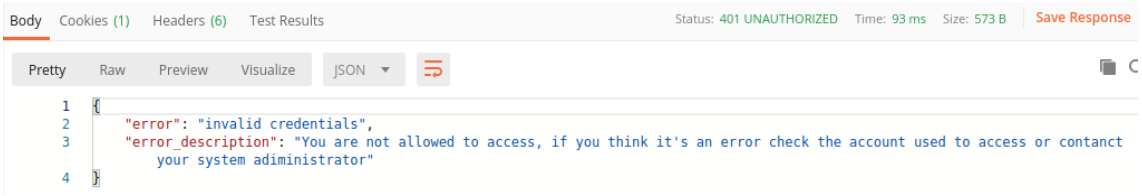


Figura 50: Respuesta de petición no autorizada

Con esto, el último paso consistirá en acceder como operador, y verificar que, es este último caso sí que se realiza la petición, para ello cambiaremos el token, de la misma manera que hemos generado el anterior.

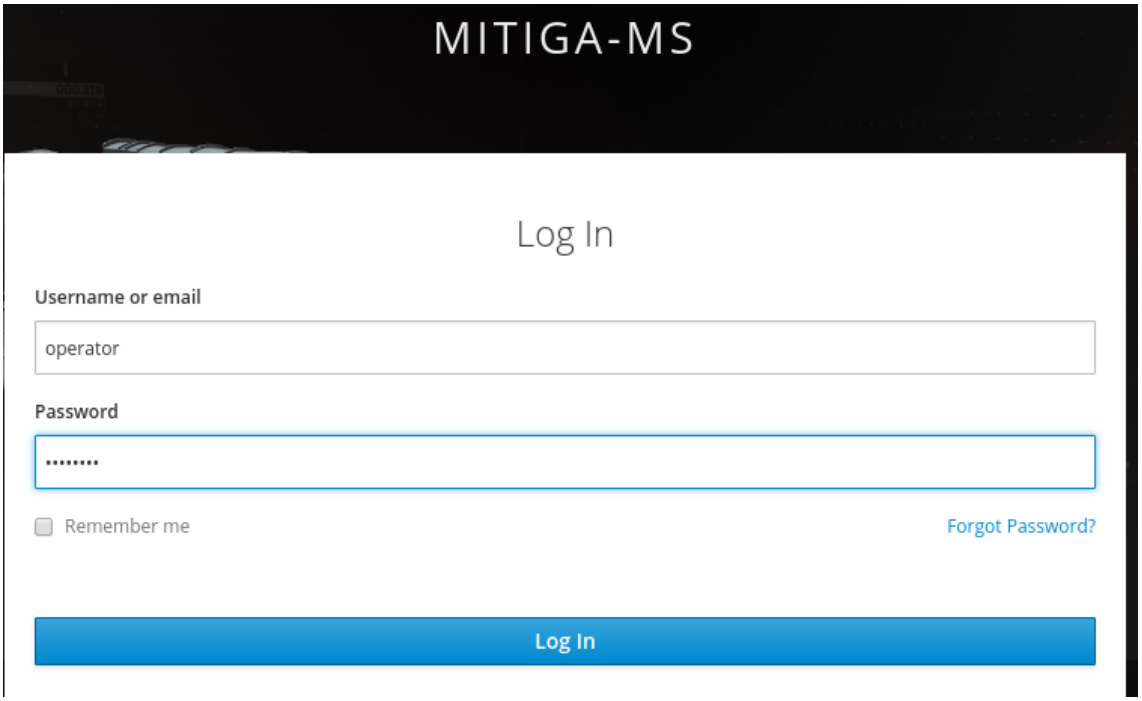


Figura 51: Login in como operador.

Tras acceder como operador, volvemos a realizar la petición y esta vez, obtenemos una respuesta satisfactoria.

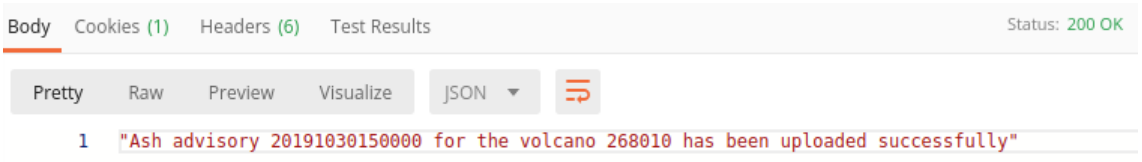


Figura 52: Petición satisfactoria a /ash/vlcn

Con este conjunto de pruebas, damos por concluida la primera prueba, la cual nos valida de manera satisfactoria la implantación del mecanismo de aceptación de tokens, así como el mecanismo implementado para la verificación de los grupos de usuarios.

Prueba 2:
Esta segunda prueba consiste en validar el funcionamiento del mecanismo de redirección del microservicio “Ash” para su página de generación de simulaciones a partir de una plantilla web.

En este caso, esperamos que al tratar de acceder a la página con el formulario de simulación se nos redirija a la página del proveedor de identidad y una vez accedamos con las credenciales se nos devuelva a la página de login como la de la Figura 48, y si nos fijamos en la url,

http://griffin.mitigasolutions.com/auth/realms/mitiga-ms/protocol/openid-connect/auth?client_id=ash&redirect_uri=http%3A%2F%2Fash%2Emitigasolutions%2Ecom%3A8001%2Foidc_callback&scope=openid+email&access_type=offline&response_type=code&state=eyJjc3JmX3Rva2VuIjogInJwZDFkMGJEUnJwc01EQ0JTUVpsZEdrZGJYYm1tRndhIiwgImRlc3RpbmF0aW9uIjogImV5SmhiR2NpT2lKSVV6VXhNaUo5LkltaDBkSEE2THk5c2IyTmhiR2h2YzNRNk9EQXdNUzkzWldJdmFXNWtaWGd1YUhSdGJDSS5uXzJRd3dNVjdpOGpzaG05MkJES1BVc2dKbnEtNFp4Yk1JaHRvamxMZmtBeGoxbmRXYjFyWHhnc0dhVUVXWGsyb080RlFlU3c1ck1tY0dTbm1IZEtSdyJ9&openid.realm=mitiga-ms

Si analizamos ligeramente la url podemos ver los siguientes campos
Client_id → Ash
Redirect_uri -> http://ash.mitigasolutions.com:8001/oidc_callback
Scope → [oidc, email]
Realm → mitiga-ms

Finalmente, tras acceder como un usuario, se nos devuelve a la página con el formulario de la simulación, dando por concluida esta segunda prueba de manera satisfactoria.

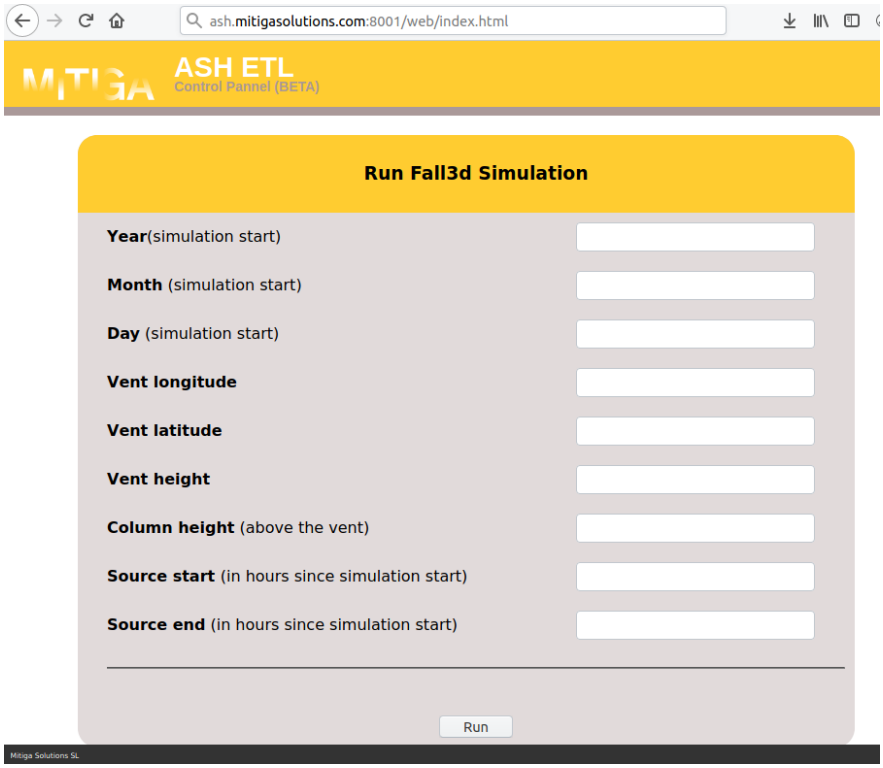


Figura 53: Formulario de simulación

Prueba 3:
En esta tercera prueba trataremos de obtener un token asociado al micorservicio “Advisory” para tratar de enviar un evento eruptivo al microservicio “Ash”, de manera este la acepte como una petición valida. Para complementar esta prueba, trataremos de enviar la misma petición utilizando un token de usuario, el cual debería ser rechazado, puesto que no queremos que un usuario cualquiera sea capaz de introducir ruido en el sistema de predicción en tiempo real.

En esta prueba intentaremos pues acceder sin ningún tipo de token, Figura 5454, con un token correspondiente a un usuario genérico, Figura 55 y con un token de un operador Figura 566.

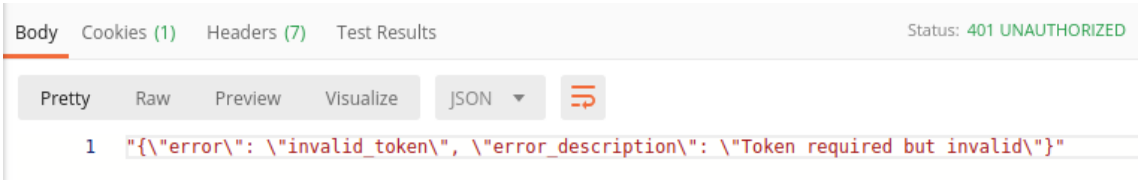


Figura 54: Intento de acceso sin token

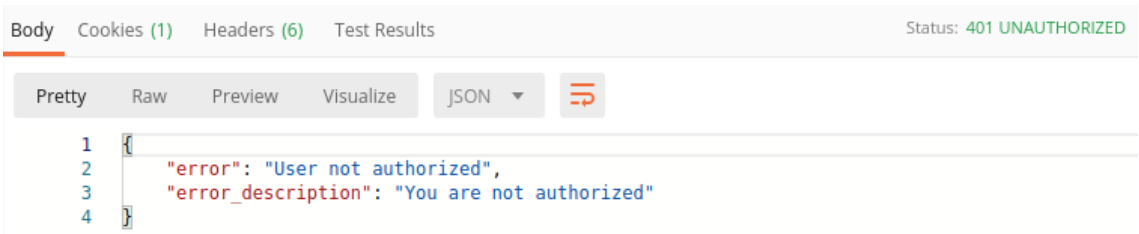


Figura 55: Intento de acceso con un usuario genérico

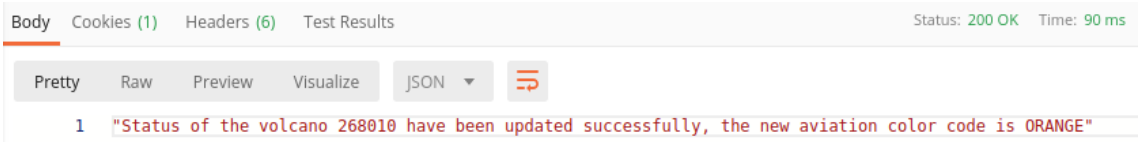


Figura 56: Intento de acceso con un operador

El funcionamiento para usuarios basado en roles funciona correctamente. El ultimo punto a tratar es esta prueba es la autorización de un cliente contra el microservicio de “Ash”. Para ello, a continuación, realizaremos una petición emulando ser el microservicio “Advisory” para obtener un token valido asociado al cliente tal y como se ve en la Figura 577, y lo utilizamos para acceder al microservicio “Ash”, obteniendo la respuesta de la Figura 58.

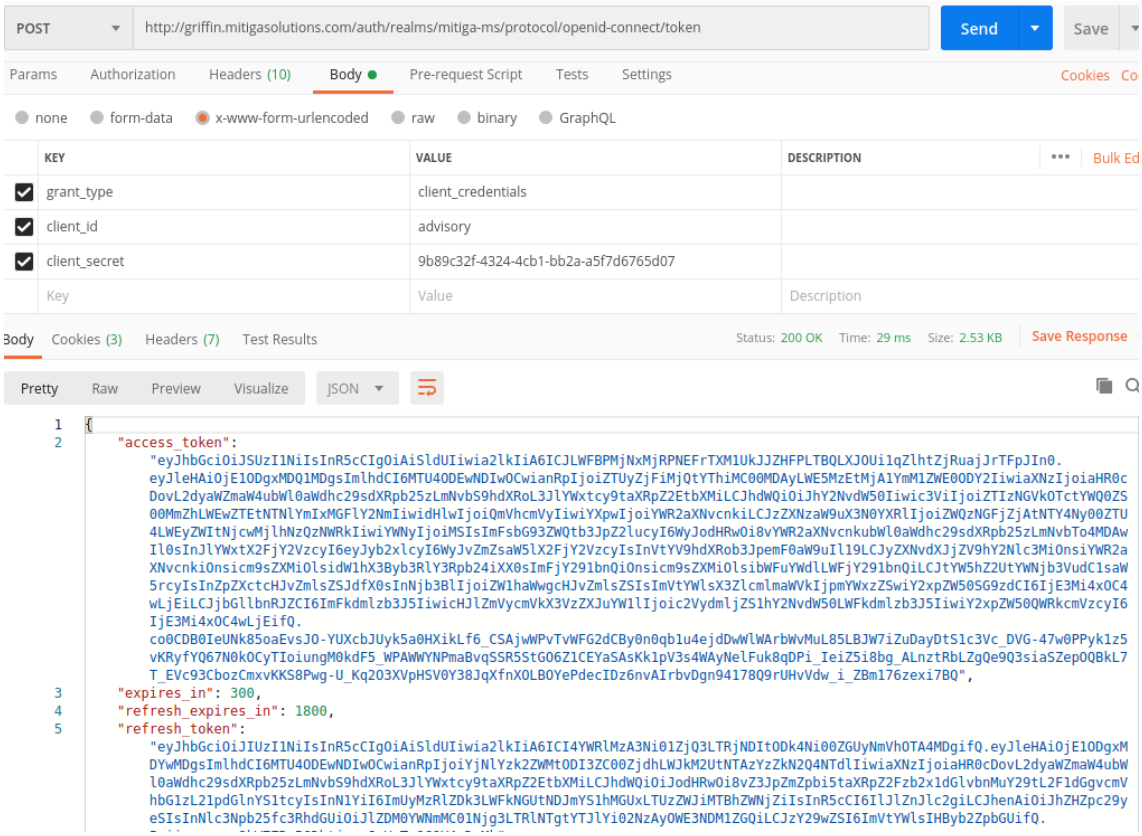


Figura 57: Petición del token del cliente

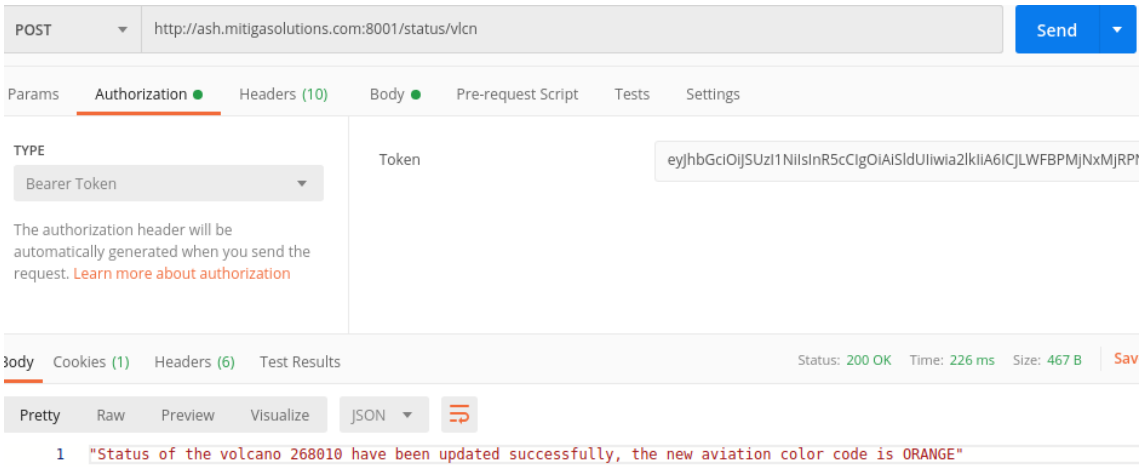


Figura 58: Petición del cliente Advisory autorizada.

7. Despliegue del sistema

En este apartado pondremos en marcha los dos microservicios modificados para verificar que el sistema es funcional. Los puntos clave a comprobar en este apartado son la interacción por parte de un usuario con el microservicio “Ash”. Y la comunicación autenticada entre microservicios, concretamente el reporte de nuevos eventos volcánicos por parte de “Advisory”.

Para levantar el sistema únicamente debemos dirigirnos al directorio de cada uno de los microservicios y ejecutar el comando “python3 main.py” o bien utilizar contenedores de docker para levantar cada uno de ellos. Tras levantar el sistema pasaremos a comprobar que ambos microservicios se estén ejecutando correctamente.

Antes de levantar los servicios deberemos modificar la configuración para hacer uso del protocolo HTTPS tal y como se muestra en la Figura 59. De esta manera nos aseguramos de que el token sea enviado de forma segura.

Figura 59: Configuración mediante HTTPS.

Una vez con el sistema en marcha vamos a realizar un conjunto de comprobaciones, por un lado, vamos a verificar el correcto funcionamiento del servidor de SSO y por el otro el funcionamiento de los distintos microservicios y su interacción entre ellos y con el sistema de login.

Las comprobaciones que haremos para validar el correcto funcionamiento del sistema serán las siguientes:

- Los datos introducidos en el microservicio “Advisory” generan una petición valida al microservicio “Ash”.
- Tratar de introducir los mismos datos directamente en el microservicio de “Ash” sin ningún token no debe ser permitido.
- Tratar de introducir los mismos datos con un usuario autorizado debe ser permitido.
- Acceder a la página web del microservicio “Ash” debe redirigirnos al login.
- Comprobar que el mecanismo de cierre de sesión funciona adecuadamente.

Con ambos microservicios en funcionamiento nos dirigiremos a la su API para ver las opciones que ofrece cada uno de ellos, tal y como se ve en la Figura 60.

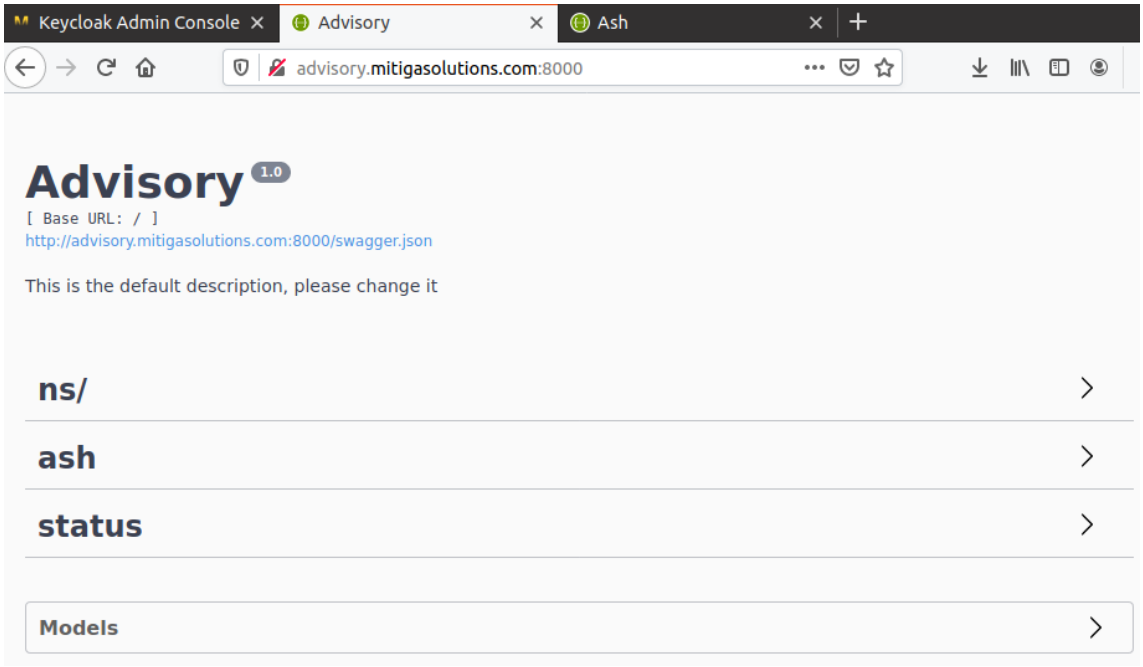


Figura 60: API de advisory

Para la primera prueba consultaremos el estado del volcán Dukono, dado que acabamos de arrancar el microservicio y la base de datos utilizada se encuentra vacia, nos devolverá un mensaje indicando que no se ha encontrado el estado del volcán, tal y como se muestra en la Figura 61, el siguiente paso será utilizar advisory para introducir nuevos datos acerca de una erupción. Estos nuevos datos generarán el envio de un mensaje desde el Advisory hacia Ash, el cual llevará el token de advisory, y por tanto será aceptado como una petición valida, tal y como se ve en la Figura 63. Para ello utilizaremos la misma herramienta que hemos utilizado para realizar los tests, Postman. Si volvemos a realizar la misma consulta a Ash, veremos que el estado del volcán se ha actualizado Figura 64.

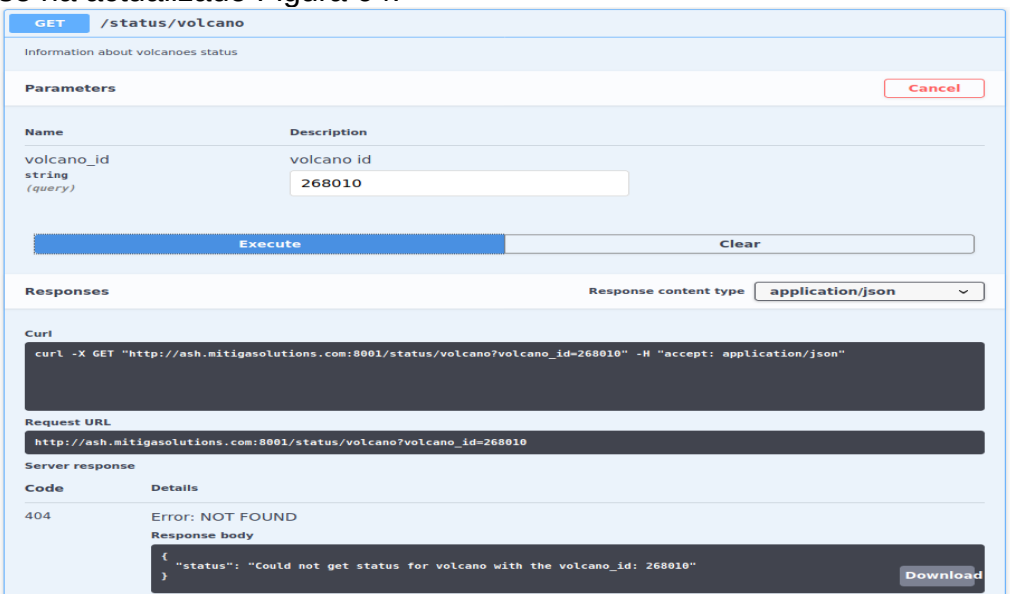


Figura 61: Consulta del estado del volcán.

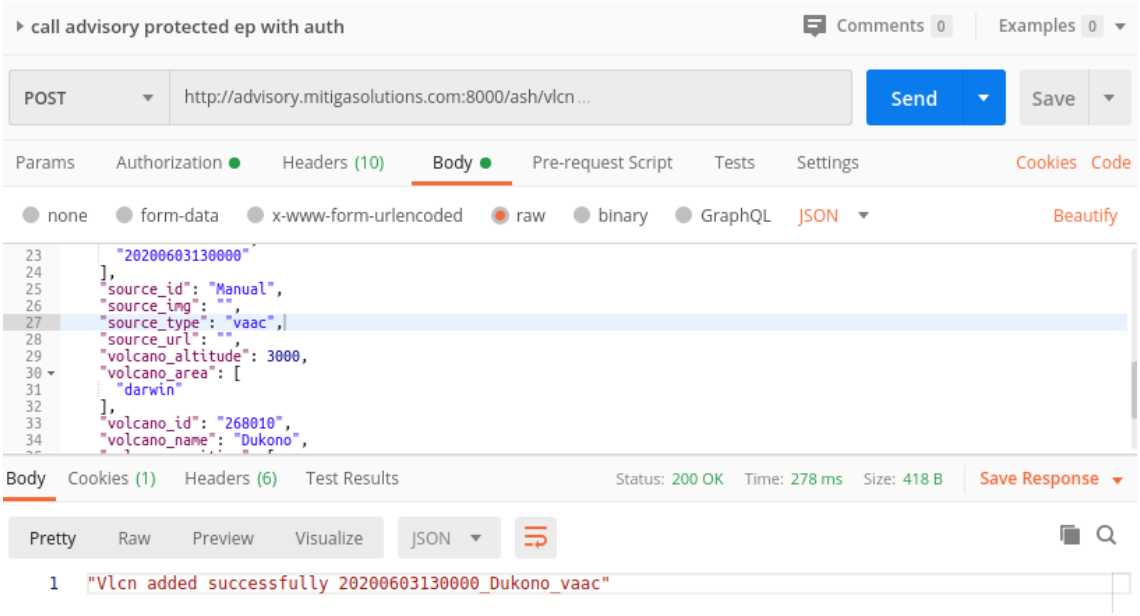


Figura 62: Petición autorizada a Advisory

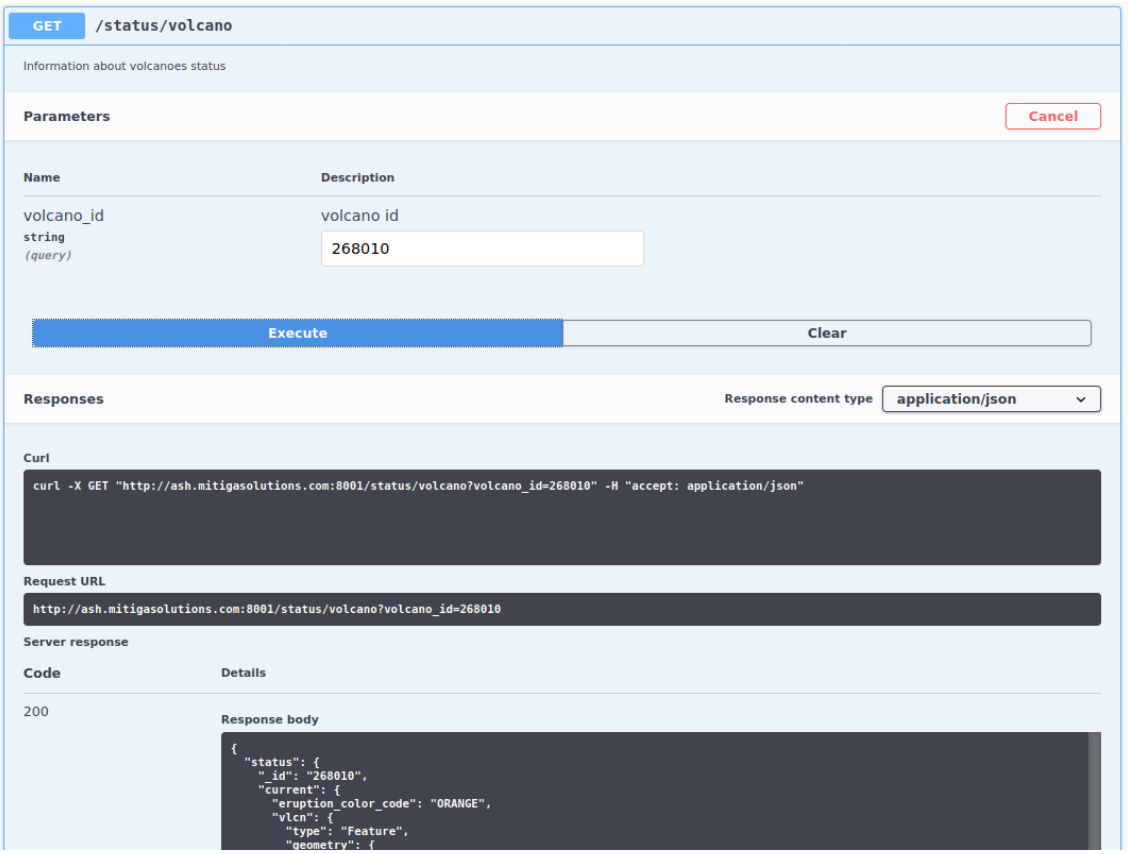


Figura 63: Consulta del estado del volcán

A continuación, vamos a ver como efectivamente, si tratamos de actualizar manualmente el estado del volcán desde el microservicio Ash se nos va a denegar la petición como podemos ver en la Figura

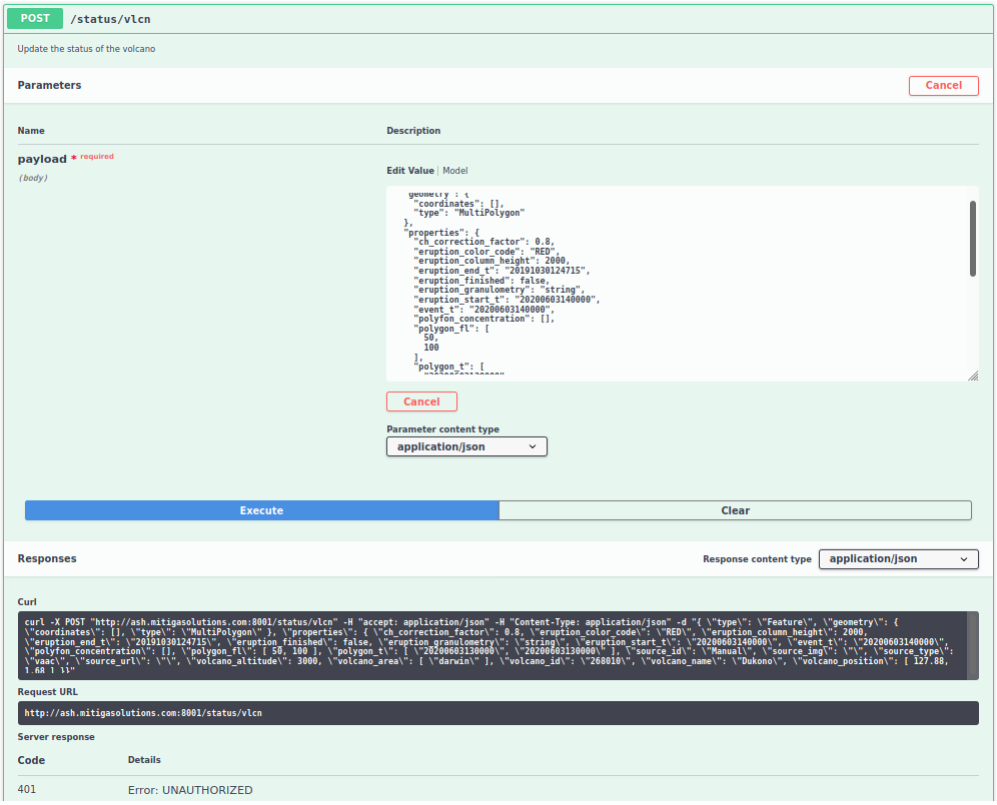


Figura 64: Petición no autorizada a Ash

Por último, mediante Postman volveremos a actualizar el estado del volcán, para ello obtendremos un token para el operador y realizaremos la petición, la cual debería ser aceptada, tal y como se muestra en la Figura 65 y en la Figura 66 podemos ver como el estado del volcán ha cambiado de “ORANGE” a “RED”, tal y como se indica en la petición

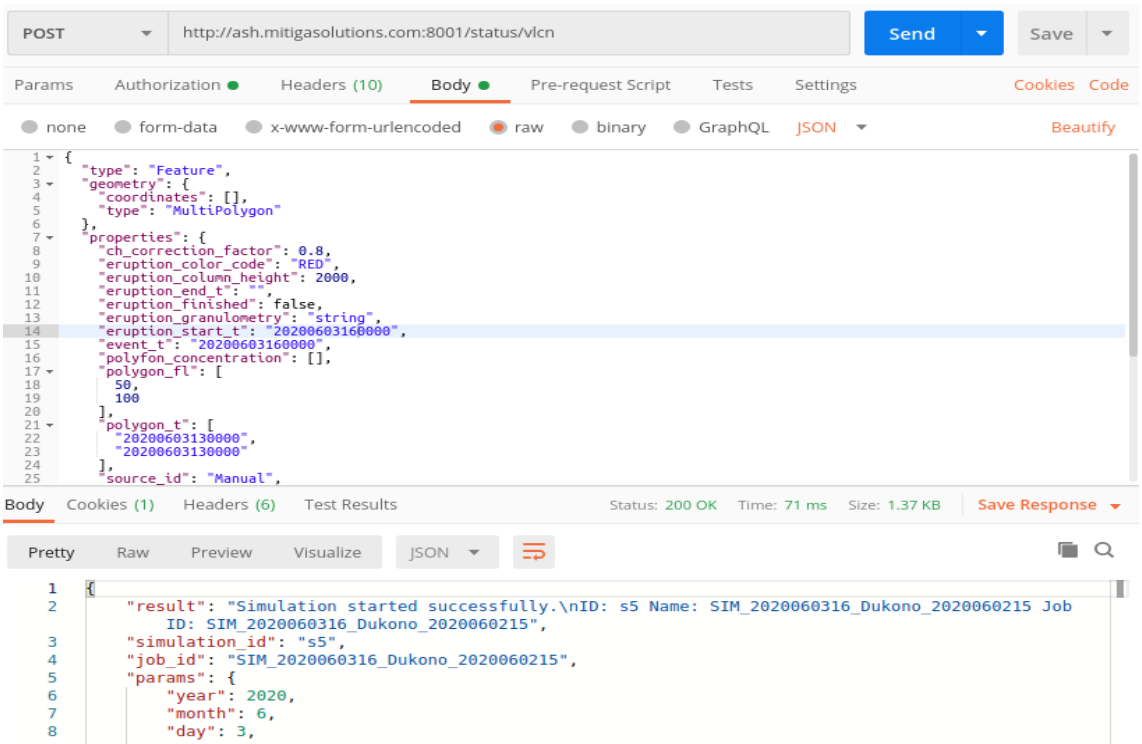


Figura 65: Petición por parte de un usuario válido

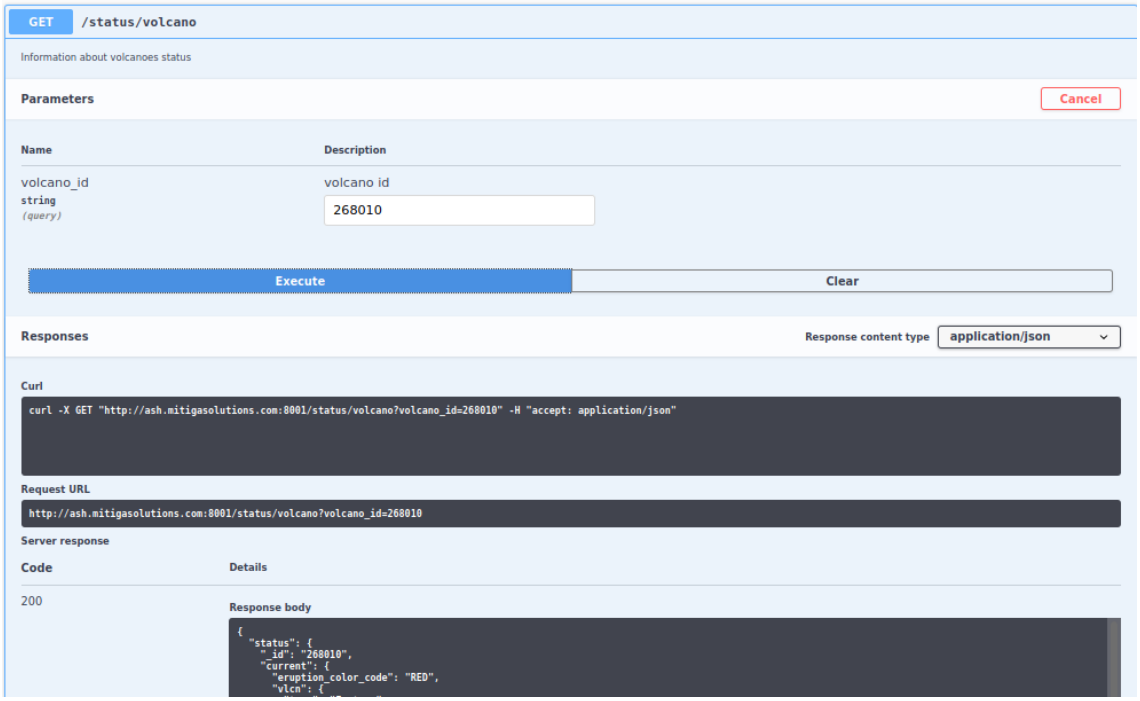


Figura 66: Actualización del estado del volcán

La ultima comprobación que nos queda por hacer es el acceso a la interfaz web del operador para poder lanzar una nueva simulación. Al tratar de acceder a la web se nos redirige automaticamente al servidor SSO, donde introduciremos las credenciales de usuario para lograr acceso, tal y como se muestra en las Figuras 67, 68.

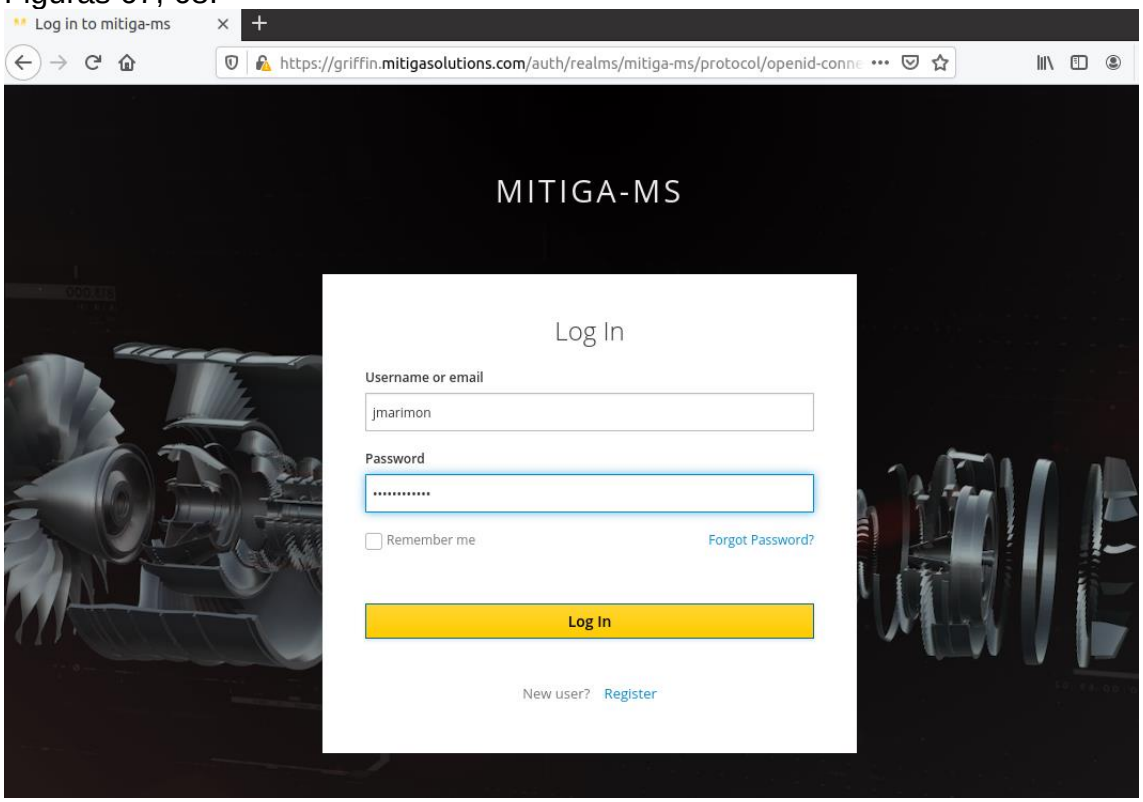


Figura 67: Página de inicio de sesión

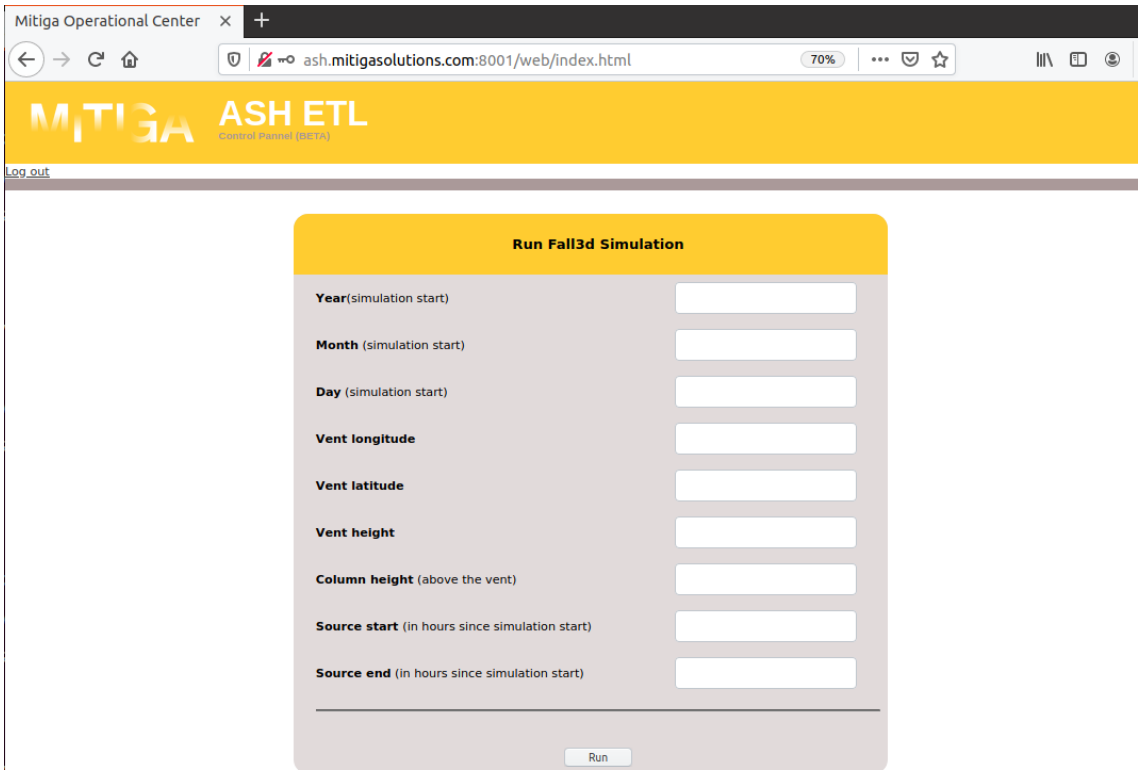


Figura 68: Panel de simulaciones.

Finalmente, tras realizar hacer clic en “Log out” se nos va a redirigir nuevamente a esta misma página, que tras el log out, nos redirige nuevamente a la pantalla de login.

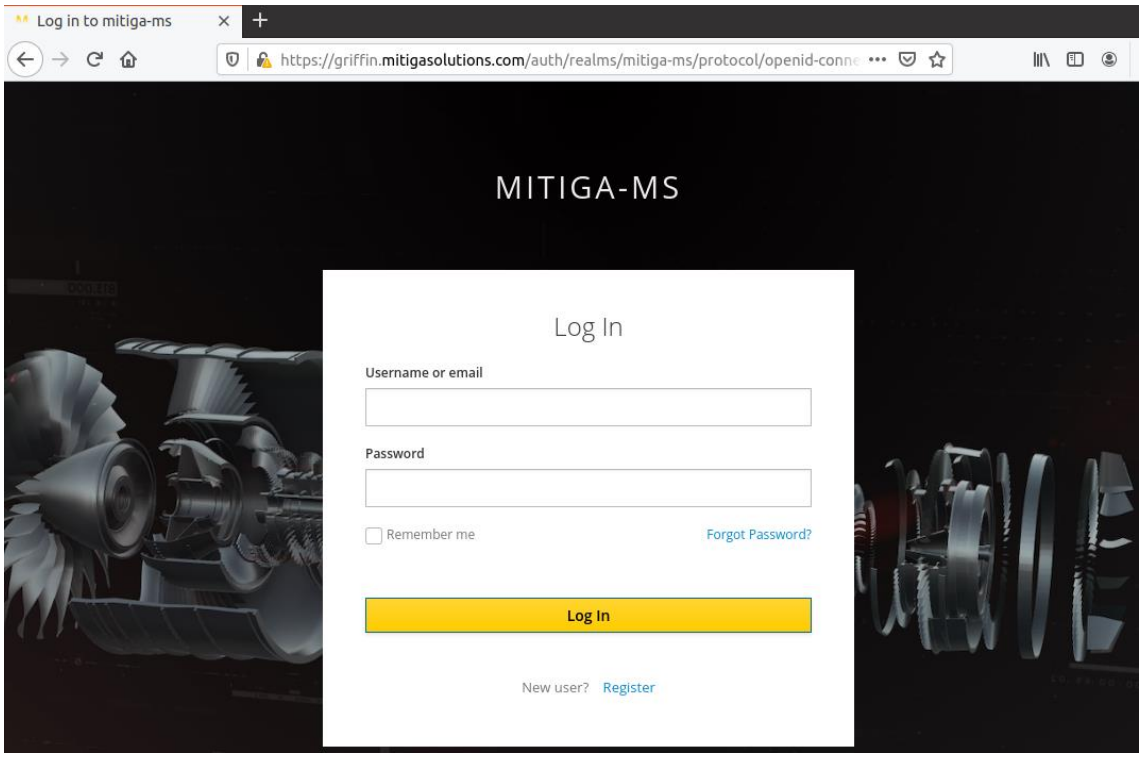


Figura 69: Página de login

Podemos concluir que el sistema funciona de la manera esperada y está listo para ser implantado en los demás microservicios de Mitiga Solutions.

8. Conclusiones

Con la realización de este trabajo se ha logrado instalar y configurar de manera satisfactoria un servidor SSO y se han establecido los mecanismos necesarios para dar soporte al protocolo OIDC a los microservicios desarrollados por MitigaSolutions.

8.1 Objetivos

En la siguiente tabla encontramos el estado en el que han quedado los distintos objetivos de este TFM. Se han logrado cumplir la mayoría de los objetivos establecidos para la elaboración de este trabajo, dejando pendiente la autenticación de los microservicios mediante certificados.
En el trascurso del trabajo se ha optado por dejar el sistema de autenticación por defecto para los servicios de keycloak los cuales son gestionados mediante una clave compartida entre el servidor de SSO y el microservicio.

A nivel de interacción con el usuario, los objetivos se han cumplido, permitiendo la autenticación mediante el par usuario y contraseña, y los mecanismos de renovación de tokens implementado en la librería Flask-OIDC que logra dar al usuario la sensación de fluidez, evitando que el usuario deba introducir nuevamente las credenciales cada cinco minutos durante la misma sesión.

Por último, se han establecido los mecanismos necesarios para autorizar el acceso a los distintos recursos en función del rol del usuario, permitiendo establecer mecanismos específicos que únicamente puedan ser usados por parte de otro de los microservicios, evitando que un usuario mal intencionado introduzca ruido en el sistema.

Objetivo	**Prioridad**	**Estado**
Servicio SSO		
Autenticación mediante usuario y contraseña	Alta	Realizado
Autenticación mediante certificados	Media	Descartado
Compatibilidad con LDAP	Alta	Realizado
Generación de tokens	Alta	Realizado
Alta disponibilidad	Alta	Realizado
Microservicios		
Aceptar tokens	Alta	Realizado
Validar tokens	Alta	Realizado

Autorizar y denegar peticiones	Alta	Realizado

Tabla 4: Estado final de los objetivos

El no cumplimiento del objetivo relacionado con la autenticación de los microservicios mediante certificados se debe principalmente a dos motivos. El primero de ellos ha sido el desplazamiento de algunas fases del proyecto debido a la situación actual debido al COVID-19. El segundo motivo, y el cual ha sido decisivo para tomar la decisión de descartar el objetivo ha sido que el mecanismo por defecto para la autenticación de microservicios que nos ofrece Keycloak era suficiente para poder terminar el proyecto.

8.2 Planificación

La planificación temporal de este proyecto era ambiciosa, pretendiendo llevar a cabo el diseño y el desarrollo del sistema completo de microservicios además de la selección, instalación y configuración del servidor de SSO.

En las primeras fases, se cumplió de manera satisfactoria con la planificación inicial del proyecto, hasta la fase 2 del proyecto, en la cual se realizaba el diseño de los mecanismos a implementar. El motivo por el cual se desfasaron los tiempos de ejecución del proyecto se debe a la dificultad para realizar la validación de los diseños debido al inicio del COVID-19. La opción tomada para tratar de mitigar el retraso en la finalización de la fase 2, fue tratar de solapar la validación del diseño con el proceso de instalación y configuración del servicio SSO, aumentando el volumen de trabajo en ese periodo.

La fase de desarrollo se alargó más de lo esperado en la planificación inicial debido a una estimación demasiado optimista de el tiempo necesario para investigar los distintos mecanismos a implementar y comprobar su correcto funcionamiento.

Finalmente, tal y como se ha mencionado anteriormente, se optó por realizar la demostración del funcionamiento del sistema con un número limitado de microservicios debido a que el tiempo necesario para adaptar todo el sistema era demasiado grande.

8.3 Líneas de trabajo futuras.

Tras la realización de este proyecto podemos encontrar múltiples líneas de continuación, siendo la más inmediatas la adopción del nuevo sistema por parte de todos los microservicios de MitigaSolutions que requieran ser expuestos a Internet, de manera que se pueda gestionar el acceso autorizado a los recursos protegidos.

Otra opción interesante que se podría implantar es la integración de diversos servicios que se encuentra corriendo en los servidores internos de la empresa, como Jenkins y nextcloud entre otros, y que ofrecen la

posibilidad de autenticar y autorizar a los usuarios mediante los protocolos OIDC y OAuth2 de manera que todos los servicios internos quedarían recogidos bajo el mismo sistema de autenticación, dando un aspecto más uniforme a todos los elementos corporativos, a la vez que se reduce la cantidad de contraseñas que deben recordad los empleados.

Por último, y pese a tratarse de una mejora estética, el equipo de diseño de la empresa será el encargado de dar la apariencia final del servidor de Single Sign On.

9. Glosario

OAuth (Open Authorization): Protocolo de autorización basado en tokens para el acceso autorizado a recursos sin exponer las credenciales del usuario.
OIDC (Open ID Connect): Protocolo de autenticación y autorización, ofrece la capa de autenticación por encima del protocolo OAuth que provee de la autorización.
SSO (Single Sign-On): Esquema de autenticación que permite al usuario acceder a múltiples servicios mediante un único identificador y contraseña.
Cliente: En OIDC, identificador del cliente, comúnmente aplicaciones o servicios que implementan el protocolo OIDC.
Usuario: En OIDC, los usuarios del sistema.
Realm: Define la configuración de un conjunto de clientes y usuarios
JWT (JSON Web Tokens): Definido en el RFC 7519, es un estándar para la representación de datos transferidos entre dos partes.
API (Application programming interface): Define las peticiones validas y su formato.

10. Bibliografía

[1] «OAuth Community Site». https://oauth.net/ (accedido mar. 29, 2020).

[2] «OpenID Connect | OpenID». https://openid.net/connect/ (accedido mar. 31, 2020).

[3] M. A. Sasse, M. Steves, K. Krol, y D. Chisnell, «The Great Authentication Fatigue – And How to Overcome It», en *Cross-Cultural Design*, Cham, 2014, pp. 228-239, doi: 10.1007/978-3-319-07308-8_23.

[4] «Keycloak - Documentation». https://www.keycloak.org/documentation.html (accedido mar. 30, 2020).

[5] «OpenAM 13.5 > Getting Started With OpenAM». https://backstage.forgerock.com/docs/openam/13.5/getting-started/ (accedido mar. 31, 2020).

[6] «CAS - Home». https://apereo.github.io/cas/6.1.x/index.html (accedido mar. 30, 2020).

[7] «Identity & Access Management - Features». https://wso2.com/identity-and-access-management/features/ (accedido mar. 31, 2020).

[8] «OpenID Certification | OpenID». https://openid.net/certification/ (accedido mar. 31, 2020).

[9] «Flask-OIDC — Flask-OIDC 1.1 documentation». https://flask-oidc.readthedocs.io/en/latest/ (accedido mar. 29, 2020).

[10] «Server Installation and Configuration Guide». https://www.keycloak.org/docs/latest/server_installation/index.html (accedido abr. 28, 2020).

[11] «Postman | The Collaboration Platform for API Development», *Postman*. https://www.postman.com (accedido abr. 28, 2020).

[12] D. Hardt <dick.hardt@gmail.com>, «The OAuth 2.0 Authorization Framework». https://tools.ietf.org/html/rfc6749#section-1.1 (accedido abr. 28, 2020).