Universitat Oberta de Catalunya uoc.edu

Trabajo Fin de Máster

Máster Universitario en Seguridad de las TIC y de las Comunicaciones

Implantación de un SSO

Alejandro Espinosa Sánchez

Consultor: Antoni González Ciria
Profesor responsable de la asignatura: Víctor García Font
Fecha de entrega: 04/06/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implantación de un SSO*
Nombre del autor:	*Alejandro Espinosa Sánchez*
Nombre del consultor/a:	*Antonio González Ciria*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2019
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, Autentificación, Autorización, Certificado digital, Moodle, Mediawiki*

Resumen del Trabajo (máximo 250 palabras):

El presente trabajo consiste en la implantación de un sistema centralizado de autenticación que permita a los usuarios acceder a una plataforma colaborativa y también a una aplicación web introduciendo sus credenciales o certificado digital una única vez, por medio de un sistema Single Sign-On (SSO).

Para llevar a cabo este objetivo, se utilizará el servidor Central Authentication Service (CAS) para unificar el proceso de autentificación en un punto centralizado. En este servidor, proceso de validación de los usuarios se llevará a cabo por medio del servidor LDAP que tendrá la información y las credenciales de los usuarios.

Adicionalmente, el proyecto implementará una aplicación web que gestionará la autorización a varios recursos protegidos, cada uno con ciertos permisos, para así limitar el acceso de los usuarios utilizando la librería Spring Security.

En conclusión, en este documento se recogen los procedimientos seguidos para implantar el sistema SSO y su posterior integración con las aplicaciones web, desde de la planificación, hasta la implementación de los objetivos planificados.

Abstract (in English, 250 words or less):

This work consists of the implementation of a centralized authentication system that allows users to access a collaborative platform and also a web application by entering their credentials or digital certificate only once, through a Single Sign-On system (SSO).

To accomplish this goal, the Central Authentication Service (CAS) server will be used to unify the authentication process at a centralized point. On this server, the user validation process will be carried out through the LDAP server that will have the information and user's credentials.

Additionally, the project will implement a web application that will manage the authorization of several protected resources, each with certain permissions, in order to limit the access of users using the Spring Security library.

In conclusion, this document includes the procedures followed to implement the SSO and its subsequent integration with web applications, from planning to the implementation of the planned objectives.

Tabla de contenido

Lista de figuras

Lista de códigos

1 introducción

La mayoría de usuarios que necesitan autenticarse en una aplicación recurren a crearse una cuenta en cada aplicación y cada vez son más organizaciones las que programan múltiples productos web y tienen la necesidad de centralizar en un punto la autentificación de sus productos.

Un ejemplo puede ser Google con las aplicaciones Docs, Gmail o Drive. Estas aplicaciones son independientes y cada una tiene su función, pero ambas comparten que la autenticación se realiza de forma centralizada en un único punto.

Este mecanismo de autentificación que utiliza Google se denomina Single Sign-On (SSO) y consiste en delegar la identificación de los usuarios generando una única identidad en un proveedor externo, con el objetivo de que esa identidad se utilice en ambas aplicaciones registradas.

La utilización de este sistema proporciona tanto para el usuario final como para el desarrollador de aplicaciones una serie de ventajas.

Por ejemplo, desde la perspectiva de la seguridad, la unificación del punto de autentificación aporta para las organizaciones una mayor seguridad ya que elimina la limitación de tener en cada aplicación su propio recurso para iniciar sesión.

Desde la perspectiva del usuario, se mejora su experiencia de usuario evitando la interrupción producida por tener que introducir sus credenciales cada vez y así evitando que el usuario tenga que registrarse en cada aplicación de la empresa y usar una identidad diferente.

Con este trabajo fin de máster se pretende proporcionar una guía en la que se implante un SSO conocido como WEB-SSO para generar una identidad única con el objetivo de autorizar el acceso a dos aplicaciones web: una plataforma educativa y una aplicación web.

1.1 Contexto y justificación del trabajo

La realización de este trabajo, en primer lugar, me permitirá aprender a realizar aplicaciones web donde la autenticación se realice de forma centralizada en un único punto.

Además, este trabajo tendrá la finalidad de aplicar a un proyecto práctico los conocimientos aprendidos en las asignaturas de seguridad en sistemas operativos y redes, mediante la implementación de una arquitectura DMZ y la posterior protección de la comunicación entre servidores y clientes.

1.2 Objetivos del Trabajo

El objetivo principal del proyecto es la implementación de una solución SSO que proporcione la autenticación de usuarios en un único punto en una aplicación web y en una plataforma educativa llamada Moodle.

Además del objetivo principal, se definen varios objetivos secundarios que se enumeran a continuación:

- Implementar la autenticación con usuario y contraseña o con certificado digital en el sistema CAS.

- Implementar la autorización de los usuarios utilizando Spring Security para gestionar los roles de los usuarios y limitando el acceso a los recursos.

- Implementar un balanceador de carga para distribuir el tráfico entre diferentes servidores CAS y APP.

- Crear una red desmilitarizada para separar una zona insegura que se ubica entre la red interna y la red externa.

- Proporcionar un entorno seguro encriptando las comunicaciones entre los servicios internos y entre el usuario final.

1.3 Enfoque y método seguido

Una vez fijados los objetivos que se pretenden conseguir en este trabajo se comentará la metodología que se utilizará para lograr los objetivos de la manera más óptima.

Para conseguir esos objetivos, se aplicará una metodología incremental con iteraciones quincenales de modo que se puedan tener versiones funcionales del proyecto que posteriormente se pondrán en común con el tutor para así obtener su retroalimentación y posibles mejoras.

Asimismo, se optará por la estrategia de utilizar soluciones específicas de proyectos de código libre ya creados tanto en la parte interna del servidor como en la capa de presentación.

1.4 Planificación del trabajo

La planificación temporal del trabajo ha venido acotada por las fechas de entrega fijadas en las diferentes entregas de la asignatura. De esta planificación, la fecha de inicio del proyecto fue el **20/02/2019** y se marcó la fecha de fin el día **04/06/2019**, con una dedicación diaria de 5 horas.

Como fechas clave y más importante, se tiene la entrega de la PEC3 donde se tendrá que tener los objetivos del trabajo finalizados y la PEC4 donde se redactará la memoria final del proyecto.

Entregables	Inicio	Entrega
PEC 1 - Plan de trabajo	20/02/2019	05/03/2019
PEC 2 - Preparación entorno y servidor de aplicación	06/03/2019	02/04/2019
PEC 3 - Integración con Moodle y mejoras en la arquitectura	03/04/2019	30/04/2019
PEC 4 - Memoria final	01/05/2019	04/06/2019
PEC 5 - Presentación en vídeo	05/06/2019	11/06/2019
Defensa del TFM	**17/06/2019**	**21/06/2019**

Cuadro 1 Planificación propuesta al inicio del proyecto.

A continuación, en el cuadro 2 se muestra con detalle la división del trabajo en subtareas. De esta división, cabe destacar que se organizaron siguiendo una lógica, es decir, se empezó definiendo un plan de trabajo, la preparación del entorno con sus componentes, la integración de CAS en los servidores y, por último, las mejoras para incrementar la seguridad y disponibilidad del servidor de autenticación.

ID	Nombre de la tarea	Tiempo estimado (días)	Fecha inicio	Fecha final
1	**PEC 1 - Plan de trabajo**	**13**	**20/02/2019**	**05/03/2019**
2	**PEC 2 - Preparación entorno y servidor de aplicación**	**27**	**06/03/2019**	**02/04/2019**
2.1	Diseño de la arquitectura de red	1	06/03/2019	07/03/2019
2.2	**Preparación del entorno**	**4**	**07/03/2019**	**11/03/2019**
2.2.1	Instalación del SO en las máquinas virtuales	1	07/03/2019	08/03/2019
2.2.2	Instalación del servidor de autenticación CAS	1	08/03/2019	09/03/2019
2.2.3	Instalación del servidor de aplicación	1	09/03/2019	10/03/2019
2.2.4	Instalación y configuración de MySQL y Apache	1	10/03/2019	11/03/2019
2.3	**Instalación de OpenLDAP**	**2**	**11/03/2019**	**13/03/2019**
2.3.1	Creación de las unidades organizativas	1	11/03/2019	12/03/2019
2.3.2	Introducción de usuarios de prueba	1	12/03/2019	13/03/2019
2.4	**Servidor de aplicación**	**19**	**13/03/2019**	**01/04/2019**
2.4.1	Diseño de la capa de presentación	2	13/03/2019	15/03/2019
2.4.2	Desarrollo de la lógica de la aplicación	6	15/03/2019	21/03/2019
2.4.3	Integración de la autentificación con CAS	9	21/03/2019	30/03/2019
2.4.4	Gestión de accesos a los recursos	2	30/03/2019	01/04/2019
3	**PEC 3 Integración con Moodle y Mediawiki y mejoras en la arquitectura**	**27**	**03/04/2019**	**30/04/2019**
3.1	Comunicaciones y servidores seguras	3	03/04/2019	06/04/2019
3.2	**Moodle**	**4**	**06/04/2019**	**10/04/2019**
3.2.1	Instalación y configuración	2	06/04/2019	08/04/2019
3.2.2	Integración de la autentificación con CAS	2	08/04/2019	10/04/2019
3.3	**Mejoras en el servidor CAS**	**20**	**10/04/2019**	**27/04/2019**
3.3.1	Acceso con certificado digital	10	10/04/2019	20/04/2019
3.3.2	Alta disponibilidad	7	20/04/2019	27/04/2019
4	**PEC 4 Memoria final**	**34**	**01/05/2019**	**04/06/2019**
5	**PEC 5 Presentación en vídeo**	**6**	**05/06/2019**	**11/06/2019**
5	**Defensa del TFM**	**4**	**17/06/2019**	**21/06/2019**

Cuadro 2 Estimación detallada del tiempo necesario para completar el proyecto.

1.5 Análisis de riesgos

Antes de empezar con el proyecto se identificaron posibles riesgos imprevistos que pudieran poner en peligro el proyecto y su contramedida para mitigarlos.

Cada riesgo está cuantificado siguiendo los siguientes valores:

- Riesgo muy elevado: 5
- Riesgo elevado: 4
- Riesgo medio: 3
- Riesgo bajo: 2
- Riesgo muy bajo: 1

A continuación, en el cuadro 1 se presentan los riesgos las consecuencias que pueden producir en el proyecto.

Identificación	Riesgo	Probabilidad	Impacto	Valor
Tiempo limitado	No completar todos los objetivos en el tiempo propuesto	Medio 3	Elevado 4	12
Curva de aprendizaje	Conocimiento limitado en la tecnología utilizada	Medio 3	Elevado 4	12
Daño en el ordenador	Existe un riesgo que el ordenador se averíe	Bajo 2	Muy elevado 5	10
Limitación de recursos	Falta de recursos en el equipo	Bajo 2	Elevado 4	8

Cuadro 3 Análisis de riesgos detectados en el proyecto.

Para los posibles riesgos mostrados en el cuadro 1, se plantean las posibles mitigaciones.

Identificación	Mitigación
Tiempo limitado	Realizar una correcta planificación de acuerdo al tiempo disponible
Conocimiento limitado en la tecnología utilizada	Investigar y buscar información en libros
Daño en el ordenador	Disponer de copias de seguridad
Limitación de recursos	Virtualizar externamente los servidores utilizando Microsoft Azure

Cuadro 4 Mitigaciones propuestas para los riesgos imprevistos.

1.6 Organización de la memoria

La memoria está dividida en los siguientes capítulos que se describirán brevemente a continuación.

1. **Introducción**. En el primer capítulo se introduce al lector en el objetivo de un SSO y el motivo por el que es importante en este proyecto. Seguidamente, se definen los objetivos junto con la planificación inicial y final y se concluye identificando los riesgos y sus posibles mitigaciones.
2. **Estado del arte.** En el segundo capítulo se proporciona una visión general del sistema de autenticación centralizada que utiliza algunas empresas importantes.
3. **Tecnología utilizada.** En el tercer capítulo se indican las tecnologías que se utilizan en el proyecto.
4. **Arquitectura del sistema.** En el cuarto capítulo se define la lógica y arquitectura del proyecto, indicando los servidores y sus correspondientes direcciones IP asignadas.
5. **Instalación y configuración de los servidores.** En el quinto capítulo se define el procedimiento para instalar y configurar los servidores para cumplir con los objetivos establecidos en el proyecto.
6. **Conclusiones.** En el sexto capítulo se define los objetivos cumplidos y las líneas de trabajo futuras.
7. **Bibliografía.** En este último capítulo se incluyen las referencias bibliografías consultadas.

2 Estado del arte

La implantación de un SSO es conocida por las organizaciones y cada vez más son conscientes de las ventajas que ofrecen. Es por eso, que el propósito de este capítulo será proporcionar en una visión general de los sistemas de autenticación centralizada que existen en la actualidad.

- **Web single sign-on (Web-SSO)**. Este tipo de solución opera solamente con aplicaciones y recursos que acceden a través de la web autentificando a los usuarios en varias aplicaciones en internet sin la necesidad de que lo hagan más de una vez.
  Los datos de acceso son interceptados a través de un proxy, de un componente en el servidor o en la porción de software que se ejecuta en el cliente. Los usuarios que no se han autenticado aún son redireccionados a un servicio de autenticación del que deben volver con un token o acceso exitoso.

- **Enterprise single sign-on (E-SSO)**. Opera como una autenticación primaria interceptando los requisitos de login que se requieren por las aplicaciones secundarias con el fin de completar los campos de usuario y contraseña.

- **Kerberos**. Es un protocolo de autenticación que se encarga de identificar a cada usuario a través de una contraseña. Este protocolo gestiona una estructura llamada Tickets que se entregan a los usuarios, el cual es usado por las aplicaciones cliente para obtener acceso. Kerberos es ampliamente utilizado en Active Directory. En esta plataforma Kerberos proporciona información de los roles de cada usuario autenticado, pero no verifica si esos privilegios son suficientes para acceder a sus recursos.

- **Identidad federada**. Es una de las formas más nuevas de realizar tareas de SSO.
  Corresponde a una solución de gestión de identidad la cual permite usar las credenciales disponibles en un sistema de autenticación en otros, ya sea de una misma organización o incluso de otras empresas.

3 Tecnología utilizada

En este capítulo se detallan las tecnologías que se han utilizado en el proyecto, esto incluye una breve introducción y como se han adaptado y aprovechado sus características en beneficio del proyecto.

3.1 VirtualBox

VirtualBox es un programa multiplataforma de virtualización creado por la empresa alemana Innotek GmbH, pero mantenido actualmente por la empresa Oracle Corporation.

La utilización de este programa en el proyecto habilita la instalación de sistemas operativos virtuales conocidos como sistemas huésped en un ordenador físico, cada uno con sus propias interfaces de red y características definidas. Entre esas características, se destaca que permite configurar hasta cuatro adaptadores de red por la interfaz gráfica, lo que posibilita que se pueda crear una zona desmilitarizada. Finalmente, en la figura 1 se puede observar las máquinas virtuales que se ejecutan en la máquina host, cada una independiente y con una finalidad concreta.

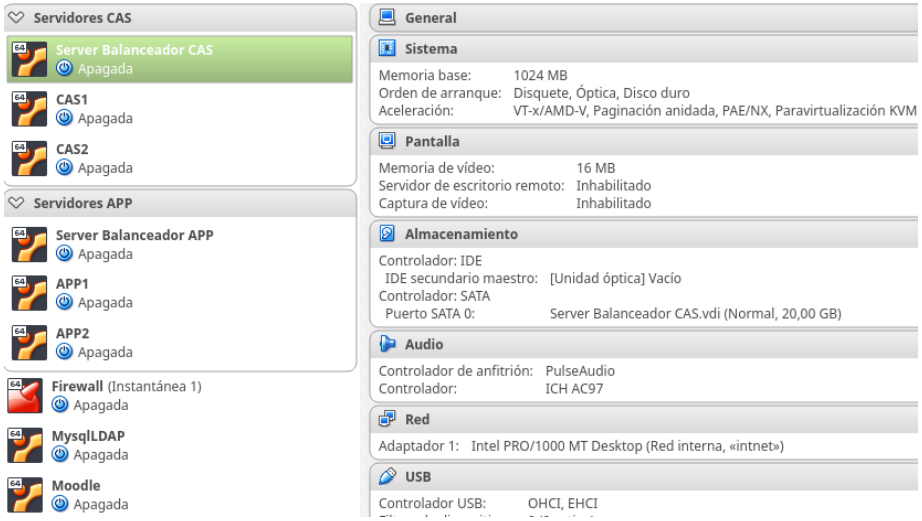

Figura 1 Servidores virtualizados con la herramienta VirtualBox

3.2 Apache Tomcat

Apache Tomcat es un contenedor web basado en el lenguaje Java que actúa como motor de servlets y JavaServer Pages (JSP) y fue desarrollado por Sun Microsystems. De esta forma, este contenedor web proporciona un entorno para desarrollar múltiples aplicaciones independientes en Java. En el proyecto se utiliza la versión **Apache Tomcat/9.0.12** para desplegar la aplicación CAS y APP, ambas en servidores Tomcat independientes.

3.3 Moodle

Es una plataforma educativa que tiene el acrónimo Modular Object-Oriented Dynamic Learning Environment y está dedicada a la enseñanza online. Esta plataforma educativa está desarrollada en PHP, utiliza una base de datos MySQL y se distribuye con una licencia pública GNU/GPL.

En el proyecto se utiliza esta plataforma porque tiene un módulo de autenticación CAS que hace posible que los usuarios puedan acceder sin introducir las credenciales de usuario utilizando para ello, la sesión del servidor CAS.

3.4 Haproxy

Haproxy *es un* servidor proxy de alta disponibilidad para protocolo TCP y servicio HTTP. Este servidor tiene como principal finalidad mejorar el rendimiento y la fiabilidad mediante la distribución de la carga de trabajo a través de múltiples servidores (por ejemplo, en aplicaciones de web, correo electrónico o base de datos).

3.5 Pfsense

Es una distribución basada en FreeBSD que está orientada para ser utilizada como cortafuegos y enrutador de paquetes. Se caracteriza por ser de código abierto y dispone de una interfaz web para su configuración basada en PHP. Además, este sistema destaca por tener una serie de funciones que permiten:

- **Port forwarding.** Permite a un usuario externo de la red tener acceso a un socket de red en una dirección IP privada desde el exterior.
- **Servidor DNS.** Es un servidor de nombres que tiene como objetivo resolver las peticiones de asignación de nombres, de forma que los nombres de host se traducen a direcciones IP.
- **Servidor DHCP.** Es un servidor que permite asignar parámetros de red a clientes que lo soliciten durante un tiempo limitado y sin interacción del usuario.
- **Extensible mediante módulos de terceros.** Extiende la funcionalidad para que se añadan nuevas funcionalidades adicionales como por ejemplo Snort para detectar intrusos en la red.

En el proyecto se utiliza este servidor para proporcionar un cortafuegos y Port forwarding. Con el cortafuegos, se proporciona un control exhaustivo del tráfico por medio de reglas para conceder o denegar el acceso. Por otro lado, con el port forwarding se facilita el acceso a los servidores internos balanceadores de carga desde el exterior.

4 Arquitectura del sistema

En este capítulo se argumenta la arquitectura lógica de los servidores. Como se puede observar en la figura 2, las peticiones del exterior se dirigen a la interfaz eth0 del enrutador que tiene dos interfaces de red, la primera interfaz *eth0* se utiliza para comunicaciones con el exterior y la segunda interfaz eth1 es asignada a la red interna de Virtualbox y permite acceder a los servidores internos.

El sistema estará formado por 9 nodos distribuidos de la siguiente manera:

- **FWA.** Es una máquina virtual que actúa de enrutador de paquetes y de cortafuegos, mediante la utilización del *Pfsense*. Dispone de una interfaz *eth0* con una dirección IP visible desde el exterior y utilizando *Port forwarding*, se redirige las solicitudes HTTPs del enrutador a los servidores balanceadores de carga.

- **server-app-bal** y **server-cas-bal.** Son servidores que integran *haproxy* para conseguir alta disponibilidad en los servidores APP y CAS. Además de las funciones propias del nodo maestro de un clúster, también permite distribuir la carga, repartiendo las peticiones HTTP y HTTPS recibidas entre los servidores definidos.

- **server-app-1** y **server-app-2**. Estos servidores tienen instalado un servidor Tomcat identifico que atiende las peticiones que les proporciona *server-app-bal.*

- **server-cas-1** y **server-cas-2**. Son servidores Tomcat que implantan el servidor CAS y atienden las peticiones del *server-cas-ba*l.

- **mysql_ldap.** Es una máquina virtual que centraliza la base de datos Mysql y el directorio de usuarios.

- **moodle**. Es una máquina virtual que tiene un servidor colaborativo que permite crear entornos virtuales de enseñanza.

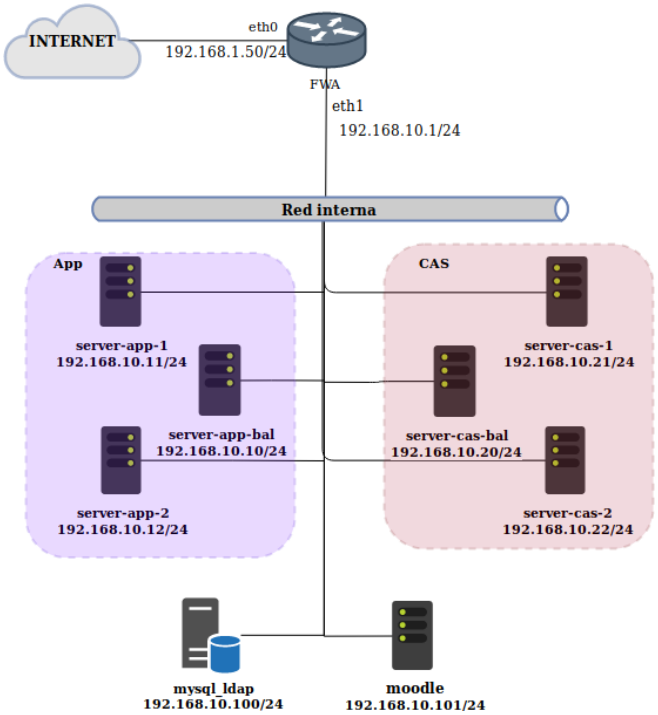

Figura 2 Arquitectura lógica de los servidores implantados y de la topología de red

5 Instalación y configuración de los servidores

Después de presentar la arquitectura del sistema, en la que se representa gráficamente los servidores, se procederá a instalar y configurar los servidores necesarios para el cumplimiento de los objetivos propuestos.

5.1 Servidor OpenLDAP

El servidor LDAP cumple un rol importante en el proyecto ya que almacena a los usuarios siguiendo una estructura definida y el servidor CAS realiza una búsqueda en el directorio para localizar al usuario y realizar la autentificación en el sistema.

5.1.1 Instalación del servidor LDAP

La instalación del servidor se realizó en el host **`mysql-ldap`** con la instrucción 5 y se introdujo una contraseña de administrador necesaria para realizar búsquedas en el directorio, por parte del servidor CAS.

```
sudo apt-get install slapd
```

Cuadro 5 Instrucción para descargar e instalar el servidor OpenLDAP.

5.1.2 Configuración del servidor LDAP

Para configurar el controlador de dominio se ejecutó la instrucción 6 en el servidor *mysql_ldap* y se configuró con el dominio **`misticEspinosa.com`**. de tal forma que en OpenLDAP se traducirá a **`“dc=misticEspinosa, dc=com”`**.

```
sudo dpkg-reconfigure slapd
```

Cuadro 6 Instrucción para configurar el dominio de empresa de OpenLDAP.

A continuación, en el código 7, se define la unidad organizativa **`usuarios`** que se utilizará para almacenar a los usuarios y posteriormente será donde se realicen las búsquedas.

```
# Unidad organizativa de Usuarios
dn: ou=usuarios, dc=misticEspinosa, dc=com
objectClass: organizationalUnit
ou: usuarios
```

Cuadro 7 Definición de la unidad organizativa para almacenar a los usuarios.

Por último, el objetivo del código 8 es crear un usuario que será de tipo **inetOrgPerson** que estará definido con el identificador **Alejandro**. Además, este usuario estará formado por otros atributos como la dirección de correo electrónico y el atributo **employeeType** que especifica los roles de ese usuario.

```
# Definición del usuario: Alejandro Espinosa
dn: uid=alejandro, ou=usuarios, dc=misticEspinosa, dc=com
objectClass: inetOrgPerson
cn: Alejandro Espinosa
sn: Alejandro
uid: alejandro
employeeType:ROLE_ADMIN,ROLE_MOODLE,ROLE_MEDIAWIKI
userPassword: {SSHA}mvAbmqPE8cWqqWFZjIqLXwtCL1289Doc
mail: aespinsa9@uoc.edu
```

Cuadro 8 Generación de un usuario en la unidad organizativa de usuarios.

Para generar las contraseñas de los usuarios, se utilizó la herramienta **slappasswd**. En esta herramienta, al no introducir ningún tipo de argumento de encriptación, se utiliza SSHA. En este comando se utilizó el argumento -s para indicar la contraseña en texto plano y en el código 9 se muestra cómo se utilizó y los parámetros utilizados para obtener los hashes de los usuarios.

```
$ slappasswd -s alejandro
{SSHA}mvAbmqPE8cWqqWFZjIqLXwtCL1289Doc
```

Cuadro 9 Generación de la contraseña encriptada del usuario alejandro.

5.2 Servidor CAS

En las siguientes subsecciones se detalla el proceso de instalación y configuración del servidor CAS, empezando primeramente por la instalación de CAS y finalizando con los procedimientos para configurarlos.

5.2.1 Instalación del servidor CAS.

Primeramente, se descargó el repositorio *cas-overlay-template* que es un repositorio con el código fuente de CAS preconfigurado. El objetivo de este repositorio es facilitar la personalización de CAS y gestión de dependencias de los componentes por medio de gradle.

```
git clone https://github.com/apereo/cas-overlay-template.git
```

Cuadro 10 Comando para clonar el repositorio cas-overlay-template.

El servidor CAS se empaquetará en un archivo WAR que tiene el acrónimo **W**eb application **AR**chive y su objetivo es distribuir aplicaciones web con todos los elementos comprimidos en un único contenedor que posteriormente se podrá ejecutar como si fuera una aplicación java.

El servidor CAS incluye Gradle que es una herramienta de automatización del proyecto para así gestionar de dependencias, integración continua y automatización del proceso empaquetado del proyecto. Al venir integrada en el proyecto significa que está configurada para realizar los pasos intermedios de gestión de dependencias y generar un archivo WAR.

```
./gradlew build
```

Cuadro 11 Compilación y generación del WAR del servidor CAS.

Una vez empaquetado, el proyecto ya está disponible para ser ejecutado. A continuación, se inició el servidor para verificar la instalación inicial ejecutando el código 12.

```
java -jar build/libs/cas.war
```

Cuadro 12 Comando para iniciar el servidor CAS.

Figura 3 Parámetros de configuración de CAS y de JAVA.

Como se observa en la figura 4, el despliegue del servidor CAS se realizó correctamente, sin embargo, todavía no se encriptaron las comunicaciones por lo que analizando el tráfico se podría obtener las credenciales del usuario. A continuación, se verificó el funcionamiento de CAS utilizando las credenciales **`casuser/Mellon`**.

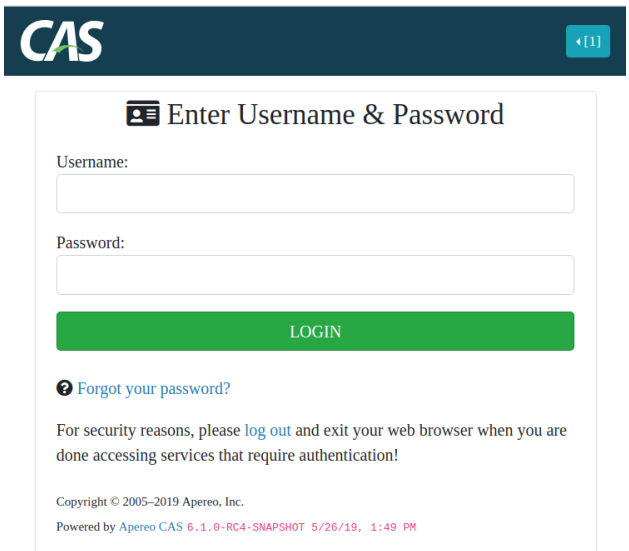

Figura 4 Portal CAS en el que se realiza la autentificación de los usuarios.

5.2.2 Configuración del servidor CAS

A continuación, en las sucesivas subsecciones se definirá el procedimiento para configurar el servidor CAS para que acepte conexiones seguras, autentifique a los usuarios mediante el servidor LDAP y permita la autentificación de usuarios con certificado digital X.509.

5.2.2.1 Conexión segura en el servidor

Un certificado SSL es el elemento principal del protocolo SSL, y se utiliza para cifrar las comunicaciones entre dos partes con la finalidad de preservar la confidencialidad e integridad de los datos. De esta forma, utilizando un certificado SSL permite que la comunicación no pueda ser alterada ni modificada por terceras personas.

El certificado está compuesto por varios archivos que sirven para implementar el protocolo SSL/TLS en un sitio web, y así establecer el protocolo HTTPS en la comunicación entre el cliente y el servidor.

Estos archivos están compuestos de tres partes principales:

- **Certificado SSL.** Es la parte pública que el servidor web envía al cliente que se conecta a través del protocolo HTTPS, y contiene el nombre del dominio e información que acredita. Cada certificado SSL tiene una clave pública que se utiliza para que el navegador web cifre la comunicación que sólo el servidor podrá desencriptar con su clave privada.

- **Clave privada.** Se utiliza para desencriptar los mensajes cifrados con la clave pública del certificado SSL. Lo almacena el servidor y se debe mantener protegido ya que se podría descifrar las comunicaciones.

- **Certificados intermedios**. Son terceras partes de confianza que expiden certificados y verifican que la persona u organización que solicita el certificado es propietaria del dominio. En este proyecto, se creará una autoridad certificadora que se emitirá será autofirmado y no estará verificado por una entidad de confianza.

Después de proporcionar un marco teórico de los tipos de certificados, se creará una autoridad certificadora para firmar utilizando el certificado raíz, los certificados de la aplicación cliente y del servidor CAS.

Para crear una autoridad certificadora es necesario ejecutar las instrucciones descritas en el código 13. Primero se crea una clave privada RSA de longitud 4096 bits. Seguidamente, se utiliza esa clave privada para generar un certificado de tipo X.509 a partir de la clave privada. Este certificado será el que se utilice para firmar las claves públicas del certificado de los servidores.

```
# Genera una clave privada de longitud 4096 bits
openssl genrsa -des3 -out rootCA.key 4096

# Genera un certificado autofirmado a partir de la clave privada rootCA.key
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 730 -out rootCA.crt -
subj "/C=ES/ST=Cataluña/L=Barcelona/O=Espinosa TFM Mistic/OU=Root
Certificado/CN=misticEspinosa.com"
```

Cuadro 13 Instrucciones para generar una autoridad certificadora.

Por defecto, el certificado generado en el código 13 no es de confianza ni está en la lista de autoridades. Esto da lugar a que cuando un usuario quiere acceder al servidor muestre un mensaje de advertencia indicando que el certificado no está emitido por una autoridad de confianza. Para solucionar el problema, es necesario añadir el certificado generado en el código 13 al almacén de certificados del navegador (véase figura 5).

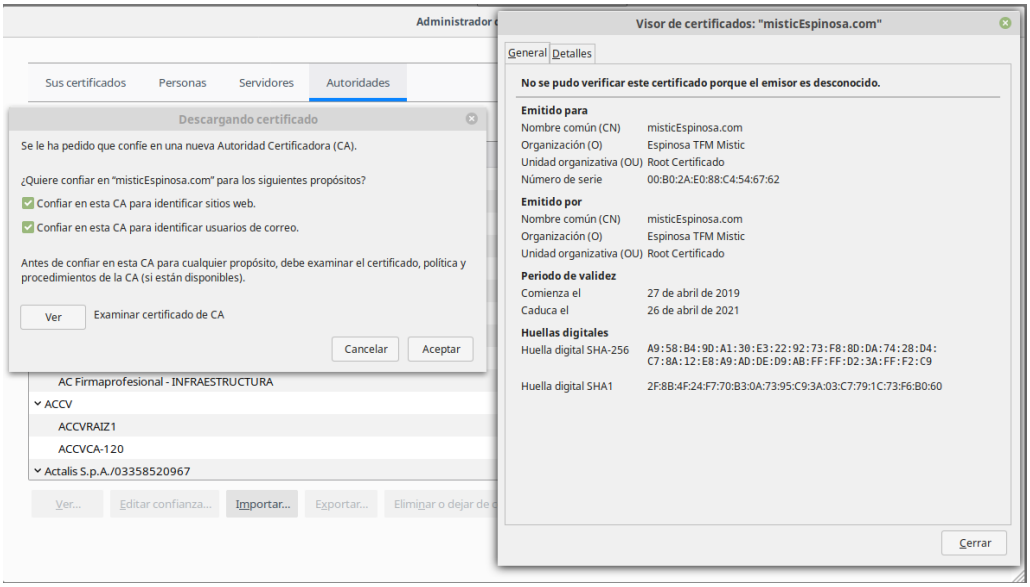

Figura 5 Visor del certificado de la autoridad certificadora.

El siguiente paso consistirá en generar el certificado para el servidor CAS que estará formado por una clave privada y una petición de firma de certificado. Esta petición de firma de certificado será un fichero que almacenará información codificada en base64 con la clave pública e información del dominio y la autoridad certificadora se encargará de firmarlo.

```
# Clave privada de longitud 4096 para el certificado del servidor CAS
openssl genrsa -out clavePrivadaCAS.key 4096

# Petición de firma de certificado (CSR)
openssl req -new -sha256 -key clavePrivadaCAS.key -subj
"/C=ES/ST=Cataluña/L=Barcelona/O=Espinosa TFM Mistic/OU=Servidor
CAS/CN=misticEspinosa.com" -out clavePrivadaCAS.csr

# Firma del CSR por la autoridad de certificación
openssl x509 -req -in clavePrivadaCAS.csr -CA rootCA.crt -CAkey rootCA.key -
CAcreateserial -out clavePublicaCAS.crt -days 730 -sha256
```

Cuadro 14 Instrucciones para crear el certificado del servidor CAS.

```
openssl pkcs12 -name="CASp12" -export -inkey clavePrivadaCAS.key -in
clavePublicaCAS.crt -out server.p12

keytool -importkeystore -srckeystore server.p12 -srcstoretype pkcs12 -srcalias
"CASp12" -destkeystore servidorCAS.keystore -deststoretype jks -deststorepass
espinosa -destalias tomcat
```

Cuadro 15 Exportación de la clave pública y privada al formato p12 y exportación al almacén de claves del servidor CA.

Después de generar el certificado, el siguiente paso fue mover el almacén de certificados **`servidorCAS`** al directorio **`conf`** de configuración del servidor. A continuación, se modificó el archivo **`cas.properties`** añadiendo el fragmento de código 16.

```
cas.server.name: https://misticEspinosa.com:8553
cas.server.prefix: https://misticEspinosa.com:8553/cas

# Configuración del puerto de escucha
server.port=8553
# Activar SSL
server.ssl.enabled=true
#Localización del almacén de certificados
server.ssl.key-store=etc/cas/servidorCAS
# Contraseña para acceder al almacén de certificados
server.ssl.key-store-password=espinosa
# Contraseña del certificado
server.ssl.key-password=espinosa
```

Cuadro 16 Conector para establecer conexiones HTTPs en el servidor CAS.

Al reiniciar y acceder al servidor CAS aparece un error de certificado autofirmado. Este error se produce porque el certificado no ha sido firmado por una autoridad certificadora reconocida por el explorador web y se soluciona añadiendo la clave pública del certificado generado del código 16 al almacén de certificados de confianza del explorador web.

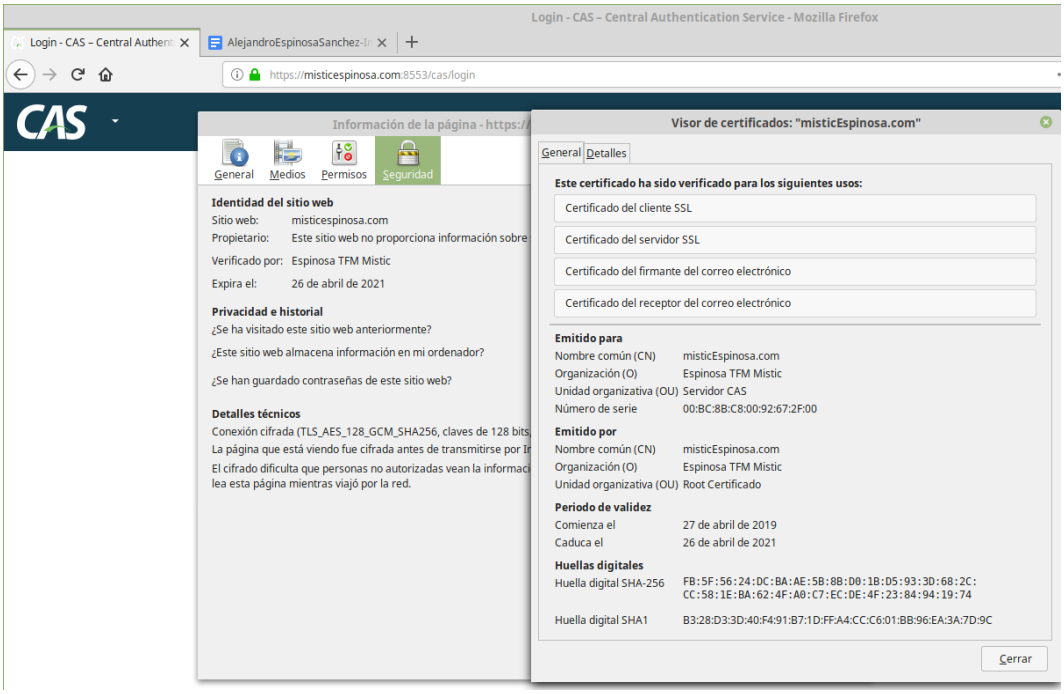

Figura 6 Visualización del certificado digital del servidor CAS.

5.2.2.2 Configuración de la interaccion con OpenLDAP

En esta sección se configurará el servidor CAS para realizar la autentificación de los usuarios mediante la utilización de LDAP. Para autentificar a un usuario, el servidor CAS proporciona varias arquitecturas, pero la utiliza en este proyecto es **Direct Bind**. Esta arquitectura se utiliza para buscar a los usuarios que estén en una unidad organizativa y en este caso, los usuarios están en **ou=usuarios,dc=misticEspinosa,dc=com.**

Para configurar la autenticación de los usuarios con la arquitectura **Direct**, como paso previo, fue necesario especificar a gradle la dependencia que permite la interacción con LDAP.

```
compile "org.apereo.cas:cas-server-support-ldap:${project.'cas.version'}"
```

Cuadro 17 Dependencia para añadir la autentificación de los usuarios utilizando LDAP.

Seguidamente, se añadió en el archivo de configuración **application.properties** la configuración para autentificar a los usuarios y buscarlos en el directorio LDAP. Una propiedad interesante se encuentra en la última línea del código 18 donde se define aquellos atributos del usuario que el servidor pone a disposición a las aplicaciones autorizadas.

```
# Especifica que los usuarios serán autentificados
cas.authn.ldap[0].type=DIRECT

# Especifica que los usuarios serán autentificados
cas.authn.ldap[0].type=AUTHENTICATED

# Ruta del servidor LDAP
cas.authn.ldap[0].ldapUrl=ldap://localhost

cas.authn.ldap[0].useSsl=false

# Unidad organizativa donde se almacenan los usuarios
cas.authn.ldap[0].baseDn=ou=usuarios,dc=espinosa,dc=com

# El filtro de búsqueda de usuarios es por el identificador de usuario
cas.authn.ldap[0].searchFilter=uid={user}

# Usuario para realizar búsquedas en la unidad organizativa
cas.authn.ldap[0].bindDn=cn=admin,dc=misticEspinosa,dc=com

# Credenciales del usuario
cas.authn.ldap[0].bindCredential=admin

# Lista de atributos que el servidor pone a disposición de las aplicaciones
autorizadas
cas.authn.ldap[0].principalAttributeList=cn:NombreApellidos,givenName:Nombre,uid:us
ername,employeeType:Roles
```

Cuadro 18 Configuración de CAS para realizar la autentificación de los usuarios con LDAP.

En el servidor CAS se utilizó el código 18 que tiene el objetivo de configurar las búsquedas en la unidad organizativa de los usuarios. La búsqueda se realiza utilizando el identificador del usuario y los atributos que pone a disposición del usuario son el **Nombre completo**, el **identificador de usuario** y sus **roles**. Para verificar que el usuario inicia sesión en el sistema y que se obtienen los atributos del usuario, se puede autentificar al usuario tal y como se hace en la figura 7.

Figura 7 Autentificación en el sistema CAS mostrando los atributos del usuario autenticado.

5.2.2.3 Definición de las aplicaciones autorizadas

Por defecto, el servidor CAS no permite la autenticación de aplicaciones externas (véase figura 8), por lo que las aplicaciones Moodle, Wiki y la aplicación CAS no podrán utilizar el servidor CAS y será necesario añadirlas como aplicaciones autorizadas.

Figura 8 Acceso de una aplicación no autorizada la autentificación de CAS.

El servidor CAS almacena las aplicaciones autorizadas en un registro de servicio y normalmente las aplicaciones se definen en formato JavaScript Object Notation (JSON). Para añadir las aplicaciones, en primer lugar, se descargó el complemento que permite definir los servicios de las aplicaciones.

```
compile "org.apereo.cas:cas-server-support-json-service-registry:${casServerVersion}"
```

Cuadro 19 Dependencia para registrar servicios en el servidor CAS.

En segundo lugar, en el archivo **`application.properties`** se añadió la localización de los archivos JSON de los servicios permitidos y en código 20 se observa esa configuración.

```
cas.serviceRegistry.initFromJson=true
cas.serviceRegistry.json.location=classpath:/services
```

Cuadro 20 Definición de la ruta del directorio de servicios.

En último lugar, se definió la estrategia de acceso de las aplicaciones que tendrán permitido autenticarse con CAS. Esta estrategia de acceso proporciona un control detallado de las reglas de autorización del servicio y describe si el servicio puede usar el servidor CAS. Además, puede configurarse para requerir un determinado conjunto de atributos que deben existir antes de que se pueda otorgar acceso al servicio.

Un caso concreto se muestra en el código 21 que define la aplicación APP. Esta aplicación está definida como servicio, es de tipo **RegexRegisteredService** y significa que se pueden definir expresiones regulares para abarcar múltiples servicios. El archivo de definición del servicio tiene los siguientes atributos:

- **@class**: Define el tipo de servicio registrado. En este caso, el servicio es de tipo **RegexRegisteredService** lo que significa que se utilizan expresiones regulares para hacer las posibles correspondencias de los servicios.
- **serviceId:** Especifica las rutas de los servicios.
- **name**: Describe el nombre del recurso.
- **id**: Campo numérico para identificar el servicio.
- **description**: Describe brevemente la finalidad del servicio.
- **evaluationOrder**. Es un campo numérico que define el orden de ejecución si múltiples.
- **attributeReleasePolicy**. Decide la política de resolución de los atributos para un servicio determinado. En cada política se puede aplicar un filtro de resolución de atributos.

  Estos filtros son:

- **Return All**. Proporciona todos los atributos.
- **Deny All.** No devuelve ningún atributo.
- **Return Allowed.** Solamente proporciona aquellos atributos que se indiquen explícitamente en la configuración del servicio.
- **Return Mapped.** Es similar a la política anterior, pero en esta política se devuelve una colección que puede ser renombrada con otro nombre.

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^(https)://misticespinosa\\.com.*",
  "name" : "APP",
  "id" : 1010,
  "description" : "Acceso a la aplicación APP",
  "evaluationOrder" : 10000,
"attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy",
    "principalAttributesRepository" : {
      "@class" :
"org.apereo.cas.authentication.principal.DefaultPrincipalAttributesRepository"
    }
  }
}
```

Cuadro 21 Definición de la aplicación APP.

Mientras que la política de resolución de atributos define cómo se proporcionan los nombres en los servicios, la política de acceso se encarga de proporcionar un control de los atributos y valores necesarios para permitir la autenticación a un servicio. En la práctica, este control se utiliza para restringir el acceso de un usuario a un recurso si dicho usuario no dispone de los roles necesarios.

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "^(https|http)://misticespinosa\\.com:5050.*",
  "name" : "Moodle",
  "id" : 1011,
  "description" : "Moodle HTTPs",
  "evaluationOrder" : 1000,
  "attributeReleasePolicy": {
     "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy",
     "principalAttributesRepository" : {
      "@class": "org.apereo.cas.authentication.principal.DefaultPrincipalAttributesRepository"
      }
  },
  "accessStrategy" : {
      "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
      "requiredAttributes" : {
             "@class" : "java.util.HashMap",
             "Roles" : [ "java.util.HashSet", [ "ROLE_MOODLE" ] ]
      }
    }
  }
```

Cuadro 22 Definición de la aplicación Moodle.

Para visualizar gráficamente el control de acceso de la aplicación Moodle se autenticó a un usuario que no disponía del rol **ROLE_MOODLE**. Al realizar la autenticación satisfactoria, el servidor deniega el acceso al servicio por no tener el rol necesario(véase figura 9).

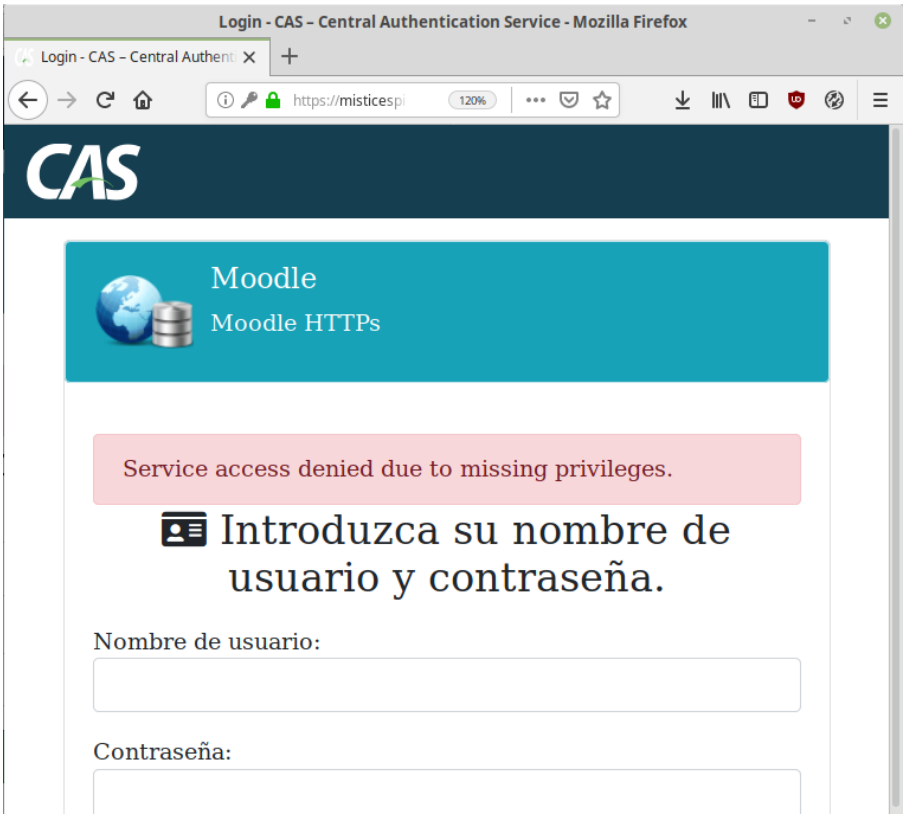

Figura 9 Acceso denegado por no disponer del rol ROLE_Moodle necesario

5.2.2.4 Autentificación con certificado de clave pública

Un certificado electrónico es un tipo de documento firmado por una autoridad de certificación que asocia una clave pública a la identidad de una persona física que posee la clave privada asociada a dicha clave pública. De esta forma, cuando se usa la clave privada se puede asegurar, atendiendo a la autoridad de la Autoridad de certificación, que ha sido usada por la entidad asociada.

Para crear un certificado de clave pública, como se ha comentado anteriormente, es necesario crear una clave privada y una clave pública. Para ello, en primer lugar, se creó una clave privada de tipo RSA del cliente alejandro utilizando una longitud de clave de 4096 bits.

```
openssl genrsa -des3 -out alejandro.key 4096
```

Cuadro 23 Generación de la clave privada del certificado del cliente.

A continuación, utilizando la clave privada se generó la solicitud de firma de certificado con la información del usuario. Esta solicitud de certificado es un bloque de texto cifrado que almacena la información que será incluida finalmente en el certificado SSL, como por ejemplo el nombre o common name (dominio para el que es generado el certificado), además de estos datos también incluirá una clave pública que será incluida también en el certificado.

```
openssl req -new -key alejandro.key -out alejandro.csr
```

Cuadro 24 Creación del CSR de la clave privada del usuario alejandro.

Como penúltimo paso, se actuó como autoridad de certificación para firmar la solicitud de firma de certificado con la clave privada y pública del certificado ROOT.

```
openssl x509 -req -days 365 -in alejandro.csr -CA ca.crt -CAkey ca.key -CAcreateserial
-out alejandro.crt
```

Cuadro 25 Comando para firmar el CSR del usuario alejandro.

Como último paso, los navegadores web no gestionan claves públicas y privadas y se deben unificar en un archivo .p12. La extensión .p12 contiene la clave pública y privada en formato binario y el acceso está protegido en el archivo PFX con una contraseña.

```
openssl pkcs12 -export -clcerts -in alejandro.crt -inkey alejandro.key -out
alejandro.p12
```

Cuadro 26 Comando para exportar a formato .p12 la clave pública y privada.

El certificado generado en el código 26 es necesario que se añada a la lista de certificados del navegador web para que se presente al servidor en el handshake SSL.

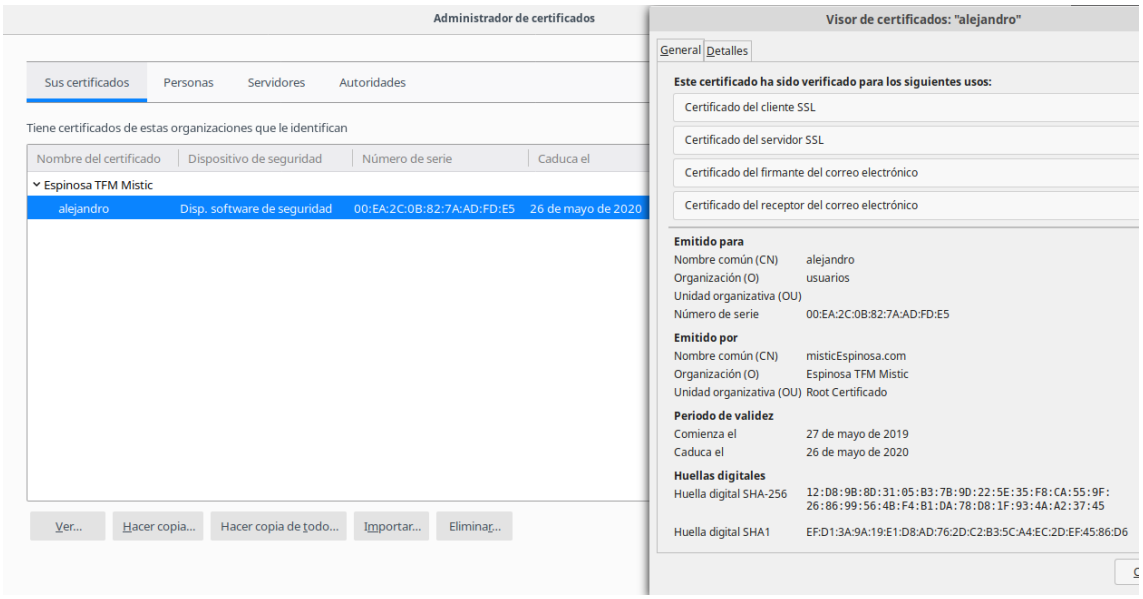

Figura 10 Certificado de persona física importado en el navegador.

Para añadir la funcionalidad de autenticación con certificado digital fue necesario añadir la dependencia que la gestiona en el archivo gradle.

```
compile "org.apereo.cas:cas-server-support-x509-webflow:${project.'cas.version'}"
```

Cuadro 27 Dependencia para soportar la autentificación con certificado digital.

Después de importar el certificado de usuario en el navegador web, se configuró el servidor CAS para que gestiona certificados digitales.

Para configurarlo, se utilizó el código que se muestra a continuación. En particular para restringir el acceso en la primera línea se comprueba que el formato del certificado con el que se firmó siga el patrón establecido. En la segunda línea no se aplica ningún tipo de restricción al formato de los certificados y se aceptan todos. En la práctica significa que se aceptarán como válidos todos aquellos certificados que sigan el formato del firmante definido en la línea 1.

```
# Expresión regular de un certificado permitido
cas.authn.x509.regExTrustedIssuerDnPattern=CN=misticEspinosa.com.+
cas.authn.x509.regExSubjectDnPattern=.+

cas.authn.x509.principalType=SUBJECT_DN
# Ruta de acceso al servidor LDAP
cas.authn.x509.ldap.ldapUrl=ldap://192.168.10.100
#
cas.authn.x509.ldap.useSsl=false

# Unidad organizativa donde se encuentran los usuarios
cas.authn.x509.ldap.baseDn=ou=usuarios,dc=misticEspinosa,dc=com
# Filtro de búsqueda de usuarios
cas.authn.x509.ldap.searchFilter=(uid={user})
# Usuario que tiene el acceso concedido
cas.authn.x509.ldap.bindDn=cn=admin,dc=misticEspinosa,dc=com
# Credenciales del usuario
cas.authn.x509.ldap.bindCredential=admin
```

Cuadro 28 Configuración de CAS para gestionar certificados.

Con la configuración anterior, el sistema CAS aceptará como válidos todos aquellos certificados que sigan el formato del firmante definido comience por **`CN=misticEspinosa.com.`** En la figura 11, se puede observar cómo se efectúa la autenticación satisfactoriamente. Sin embargo, no se logró efectuar búsquedas en LDAP y no fue posible obtener los roles del usuario.

Figura 11 Inicio de sesión utilizando el certificado electrónico de alejandro.

5.2.2.5 Gestión distribuida de tickets

Para implementar la alta disponibilidad fue necesario que todos los servidores CAS compartan los tickets generados para que cualquier servidor reconozca como propios los tickets de otros servidores miembros. El servidor CAS utiliza el mecanismo de *Tickets Storage* que se comentará posteriormente. Actualmente existen dos formas de implementar la alta disponibilidad en CAS:

- **Un solo nodo** en un entorno virtualizado. Este sistema tiene la desventaja de que si falla la máquina virtual es transferida a otro host físico. Es la forma más simple y más utilizada en entornos empresariales.

- **Varios nodos**. En esta arquitectura, se enlazan varios nodos a un balanceador de carga que se encarga de distribuir los tickets entre los servidores CAS miembros.

En la arquitectura de varios nodos se puede considerar que existen dos modos de funcionamiento:

- **Activo/Pasivo**. En este modo de funcionamiento uno de los nodos recibe una petición y se utiliza un balanceo donde solo un nodo CAS proporciona un ticket. Este modo de funcionamiento tiene la desventaja de que si se produce un fallo obliga al usuario a volver a iniciar sesión.

- **Activo/Activo**. Todos los nodos reciben la petición, pero sólo un nodo responde. Es necesario utilizar un almacén de tickets para que los nodos puedan compartir los tickets. En esta arquitectura si un nodo falla, el otro nodo puede responder a peticiones y gestionar los tickets por lo que es el modo de funcionamiento que se utilizará en este proyecto al permitir la alta disponibilidad.

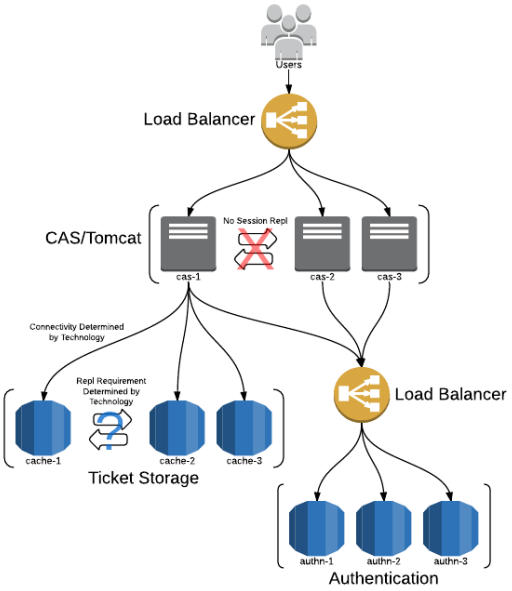

Figura 12 Arquitectura de alta disponibilidad recomendada por Apereo.

Para configurar el almacenamiento de tickets existen varias formas de realizarlo:

- **Predeterminado**: Utiliza un mapa interno en memoria para el almacenamiento y recuperación de tickets. Este componente no conserva el estado del ticket en los reinicios y no es una solución adecuada para los entornos de clúster CAS que se implementan en el modo activo / activo.

- **Basados en caché**. Proporcionan una solución de alto rendimiento para el almacenamiento de tickets en implementaciones de alta disponibilidad. Los siguientes componentes están soportados:

    - **Default**. Es un sistema de gestión de tickets basado en memoria y no mantiene los tickets entre nodos. Este sistema, para el objetivo del proyecto, no proporciona una redundancia de los tickets por lo que una caída provocaría perder las sesiones establecidas y no sería una solución.

    - **Hazelcast.** Es un sistema de gestión de tickets distribuido basado en compartir distintas estructuras de datos de forma distribuida con una escalabilidad flexible y cómoda. En esas estructuras, se localiza los tickets que los distribuye a todos los miembros para mantener la consistencia y no perderlos en caso de que un nodo falle.

    - **Ehcache.** Es un sistema de gestión de tickets similar a Hazelcast, pero con la excepción que divide la caché en diversas *regiones* donde se ubican los objetos. Estas regiones son configurables a través del XML de configuración.

        - **Memcached.** Es un sistema distribuido de propósito general. Su funcionamiento se basa en una tabla hash distribuida a lo largo de varios equipos que permiten el almacenamiento de cualquier tipo de estructura.

        - **Infinispan.** Es una base de datos distribuida sucesora de JBoss Cache de tipo NoSQL que almacena los datos siguiendo el formato clave-valor.

- **Basados en BBDD**. En este tipo de sistema, los tickets se almacenan en tablas y se accede a los tickets utilizando la capa intermedia conocida como JPA.

- **Basados en BBDD NoSQL**. Se puede utilizar una base de datos no relacional para almacenar los tickets de los usuarios.
  Concretamente, se pueden utilizar:
    - **Redis.** Es motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes. La principal diferencia entre Redis y otros sistemas basados en caché es que los valores no están limitados a ser cadenas de texto y soportan colecciones de elementos desordenados no repetidos (sets), hashes y listas.

    - **MongoDB** Es un sistema de base de datos NoSQL orientado a documentos de código abierto que tiene como singularidad que, en vez de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico haciendo que la integración de los tickets de usuarios se realice de forma más eficiente.

    - **Amazon DynamoDB.** Es una base de datos NoSQL propietaria de claves-valor y documentos que ofrece escalabilidad y disponibilidad para **garantizar** el acceso rápido a los datos. Las tablas de DynamoDB no se ciñen a un esquema fijo, y cada elemento puede tener un número diferente de atributos, por lo que la gestión de tickets utilizando esta base de datos sería una opción interesante. si se quiere un sistema rápido y totalmente gestionado que permite almacenar y recuperar de manera fácil y económica cualquier cantidad de datos, así como atender cualquier nivel de tráfico de solicitudes.

La solución escogida por su baja latencia y por su granularidad es **Hazelcast.** Para configurar la aplicación, se añadieron los atributos que se muestran a continuación en el archivo **application.properties.**

```
# Política de reemplazo de caché LRU(Least Recently Used)
cas.ticket.registry.hazelcast.cluster.evictionPolicy=LRU
# Propiedad que indica el número de copias en los servidores CAS
cas.ticket.registry.hazelcast.cluster.backupCount=2
# Número de copias de seguridad asíncronas
cas.ticket.registry.hazelcast.cluster.asyncBackupCount=2
# Servidores CAS miembros que compartirán los tickets
cas.ticket.registry.hazelcast.cluster.members=192.168.10.21,192.168.10.22
```

Cuadro 29 Configuración de CAS para gestionar tickets de forma distribuida.

5.3 Configuración de Haproxy

La gestión de tickets implementada en la sección anterior utilizando Hazelcast cambia el paradigma de gestión de tickets a una gestión de forma distribuida. Esto permite que, si se balancea el tráfico, todos los nodos almacenan la misma base de datos con los tickets mantenimiento la consistencia de tickets.

En este proyecto se utilizó un balanceador para el servidor CAS y otro balanceador para la aplicación cliente CAS. Cada balanceador de carga gestiona sus propios servidores y en ambos de utiliza una política de planificación Round Robin.

A continuación, se explicará el archivo de configuración de Haproxy para balancear la carga del servidor CAS. Este archivo está compuesto de directivas y concretamente son:

- **Global**. En esta directiva se especifican las opciones del balanceador de carga, tal como el usuario que lo ejecutará, el fichero de PID y el archivo de almacenamiento del registro.
- **Defaults**. Esta directiva define los parámetros propios del mecanismo de balanceo de carga y que serán válidos para todos los balanceadores, como por ejemplo los timeouts y el modo de trabajo por defecto.
- **Frontend**. Es el punto de entrada de las conexiones de usuario. Especifica cómo entrarán los usuarios definiendo el puerto de enlace y cómo reenviará estas solicitudes hacia el backend que es donde realmente se produce el balanceo.
- **Backend**: En esta directiva se especifican los servidores/puerto sobre los que se balancean las sesiones de usuario, modo de balanceo y de qué manera se gestionaran las sesiones.

En la directiva **Backend**, se definieron un conjunto de parámetros para especificar el comportamiento del balanceo de la carga para los servidores CAS.
Estos parámetros son:

- **cookie**. Parámetro que activa la persistencia de las sesiones basadas en cookies.
- **insert**. Indica que la cookie persistente se inserta en las respuestas del servidor.
- **nocache**. No permite que la cookie sea cacheada.
- **balance roundrobin**. Esto es balanceo equilibrado de una conexión a cada servidor alternativamente.
- **server**. Describe el servicio sobre el que se balancea la conexión. Habrá tantos como servidores. Cada servidor tiene un identificador interno de servicio balanceado y un socket de red donde se enviarán las solicitudes.
- **cookie**. Indica todos los parámetros insertados por HAProxy que tendrán las cookies que serán enviadas a cada uno de los servidores.

```
#---------------------------------------------------------------------
# Frontend ServidorCAS
#---------------------------------------------------------------------
frontend servidorcas
   bind *:443
   option tcplog
   mode tcp
   default_backend     app

#---------------------------------------------------------------------
# Planificación RoundRobin para balancear el tráfico entre los servidores
#---------------------------------------------------------------------
backend app
       mode  tcp
       balance roundrobin
       cookie SERVERID insert indirect nocache
       server CAS1 192.168.10.21:8553 check
       server CAS2 192.168.10.22:8553 check
```

Cuadro 30 Archivo de configuración del balanceador de carga de los servidores CAS.

5.4 Aplicación cliente

La aplicación cliente se diseñó utilizando Spring para la parte lógica y para la parte visual Bootstrap. La implementación de la parte lógica, se implementó utilizando el patrón de arquitectura de software Modelo Vista Controlador (MVC), de tal forma que utilizando tres componentes permite separar la lógica de la aplicación de la vista.

Cada componente tiene su propia función en la arquitectura. En particular, el modelo se encarga de los datos, generalmente consultando la base de datos para efectuar actualizaciones, consultas o búsquedas.

Por otro lado, el controlador tiene como objetivo controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicarles dichas órdenes a la vista.

Por último, la vista representa visualmente los datos, es decir, la representación gráfica y ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

Con la base teórica de la arquitectura en la que está basada la aplicación cliente, se implementará la interacción con el servidor CAS para la autentificación de los usuarios.

5.4.1 Configuración de la aplicación para interactuar con CAS

La configuración de Spring para interactuar con CAS se realizó definiendo una serie de métodos para definir las acciones a realizar en el diagrama de flujo de SSO. Estas acciones se enumeran a continuación:

1. El usuario intenta acceder a un recurso protegido.

2. Se intercepta la solicitud de acceso a un recurso protegido y se activa el método *AuthenticationEntryPoint* para redireccionar al usuario a la página de autentificación de CAS.

3. A continuación, si la autentificación se realiza de forma satisfactoria, el servidor redirecciona al usuario al recurso *SERVICE* con la información del ticket añadida como parámetro.

4. Seguidamente, se activa *CasAuthenticationFilter* para procesar los datos del ticket incluyendo los roles del usuario.

5. Finalmente, si el ticket es correcto, es decir, tiene los roles y la información requerida para acceder al recurso compartido, el usuario será redireccionado al recurso solicitado.

En el código 31 se define el mecanismo de autentificación con las acciones que realizará el sistema cuando intercepte una solicitud de acceso a un recurso que necesite permisos. Cada sistema de autenticación tendrá su **AuthenticationEntryPoint** propio, que realiza acciones como enviar avisos para la autenticación.

```
@Bean
@Primary
public AuthenticationEntryPoint authenticationEntryPoint(ServiceProperties
serviceProperties) {
       CasAuthenticationEntryPoint entryPoint = new  CasAuthenticationEntryPoint();
       entryPoint.setLoginUrl(LOGIN_URL);
       entryPoint.setServiceProperties(serviceProperties);
       return entryPoint;
    }
```

Cuadro 31 Definición del mecanismo de autentificación del cliente CAS.

El código 32 define la configuración ***ServiceProperties*** local de interacción del cliente con el servidor CAS. Un método que se utiliza es ***setSendRenew*** que se activa y solicita las credenciales de usuario, aunque previamente exista una sesión de usuario.

```
@Bean
    public ServiceProperties serviceProperties() {
        ServiceProperties serviceProperties = new ServiceProperties();
        serviceProperties.setService(SERVICE);
        serviceProperties.setSendRenew(true);
        return serviceProperties;
    }
```

Cuadro 32 Definición de las propiedades locales de interacción del servidor CAS.

El código 33 instancia un objetivo de tipo **TicketValidator** para definir validar un ticket en posesión del usuario. La validación de los tickets se realiza utilizando la última versión del protocolo CAS 3.0 y esta versión tiene una nueva funcionalidad para devolver los atributos del usuario mediante el uso del recurso **/p3/serviceValidate**.

```
@Bean
public TicketValidator cas30ServiceTicketValidator() {
    return new Cas30ServiceTicketValidator(TICKET);
}
```

Cuadro 33 Implementación del método de gestión de la respuesta del servidor CAS.

El código 34 se utiliza para procesar el ticket y obtener la información del usuario que se autentica utilizando **UserDetailsService**. En la información que devuelve el ticket, se encuentran los ***Authorities***, qué son los roles que tendrá el usuario. Por lo tanto, en el momento de recuperación y creación de ese objeto, es necesario también indicar los roles correctos.

```
@Bean
public AuthenticationUserDetailsService<CasAssertionAuthenticationToken>
authenticationUserDetailsService() {
    return new UserDetailsService();
}
```

Cuadro 34 Método que procesa los atributos recibidos del servidor para crear el objeto usuario.

Finalmente, el método más importante se define en el bean **AuthenticationProvider** que se activa cuando el usuario accede a un recurso protegido y es dirigido a la página de autentificación en CAS. A continuación, una vez introduzcan las credenciales y se envíe el formulario a la página, el servidor autentifica al usuario y crea un ticket que se añade a la dirección URL de tal forma que lo obtenga el cliente y pueda validarlo. Para validarlo, se invoca al método **cas30ServiceTicketValidator** y se obtiene la información con el método **authenticationUserDetailsService** con los roles del usuario.

```
@Bean
public CasAuthenticationProvider casAuthenticationProvider(ServiceProperties
serviceProperties) {
    CasAuthenticationProvider casAuthenticationProvider = new
CasAuthenticationProvider();
    casAuthenticationProvider.setServiceProperties(serviceProperties);
    casAuthenticationProvider.setTicketValidator(cas30ServiceTicketValidator());
casAuthenticationProvider.setAuthenticationUserDetailsService(authenticationUserDet
ailsService());
    casAuthenticationProvider.setKey("&MiClaveEspinosaMistic$");
    return casAuthenticationProvider;
}
```

Cuadro 35 Definición del bean AuthenticationProvider.

5.4.2 Configuración de la aplicación cliente para obtener los roles del usuario

Como se definió en el código 36, la clase **`UserDetailsService`** extiende de la clase abstracta **`AbstractCasAssertionUserDetailsService`** para proporcionar un mecanismo de creación de un usuario de tipo ***`UserDetails`*** a partir de la respuesta del servidor CAS. Este mecanismo hace posible la obtención de los datos de usuario para agregar información de sus roles y así poder gestionar la autorización a los recursos.

```
public final class UserDetailsService extends
AbstractCasAssertionUserDetailsService {
 @Override
 protected UserDetails loadUserDetails(final Assertion assertion) {

 AttributePrincipal principal= assertion.getPrincipal();
 String strRoles = (String) principal.getAttributes().get("Roles");
 String username = (String) principal.getAttributes().get("username");

 List<GrantedAuthority> roles
=AuthorityUtils.commaSeparatedStringToAuthorityList(strRoles);
 return new User(username, "",roles);
 }
}
```

Cuadro 36 Procesamiento de los datos de usuario y de los roles del usuario.

Internamente cuando se genera un ticket, el servidor CAS muestra por consola información del proceso de autentificación entre la que se destaca los atributos que facilita al servicio, los atributos requeridos para autentificarse y la dirección IP origen y del servidor.

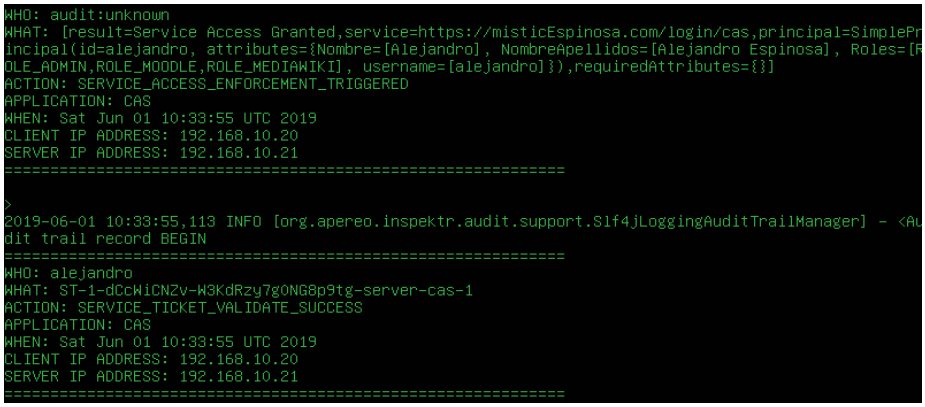

Figura 13 Información interna de la autentificación del usuario en el servidor de aplicación.

5.4.3 Conexión segura en el servidor

El procedimiento de implementación de la conexión TLS en el servidor de aplicación es equivalente al seguido en el servidor CAS, con la excepción de que no es necesario crear una nueva autoridad certificadora que firme el certificado. Esto se debe a que se utilizará el certificado raíz para firmar el certificado de esta aplicación.

```
# Clave privada de longitud 4096 para el certificado del servidor CAS
openssl genrsa -out clavePrivadaAPP.key 4096

# Petición de firma de certificado (CSR)
openssl req -new -sha256 -key clavePrivadaAPP.key -subj
"/C=ES/ST=Cataluña/L=Barcelona/O=Espinosa TFM Mistic/OU=Servidor
CAS/CN=misticEspinosa.com" -out clavePrivadaAPP.csr

# Firma del CSR por la autoridad de certificación
openssl x509 -req -in clavePrivadaAPP.csr -CA rootCA.crt -CAkey rootCA.key -
CAcreateserial -out clavePublicaAPP.crt -days 730 -sha256
```

Cuadro 37. Creación del certificado del servidor de aplicación.

```
openssl pkcs12 -name="APPp12" -export -inkey clavePrivadaAPP.key -in
clavePublicaAPP.crt -out client.p12

keytool -importkeystore -srckeystore client.p12 -srcstoretype pkcs12 -srcalias
"APPp12" -destkeystore servidorAPP.keystore -deststoretype jks -deststorepass
espinosa -destalias tomcat
```

Cuadro 38 Exportación del certificado del servidor de aplicación al almacén de llaves.

Por defecto en Spring está activado el puerto de conexión no seguras. Este puerto se puede modificar añadiendo unos atributos en el archivo ***application.properties*** como se muestra en el código 39. Este archivo establece los parámetros de configuración de los componentes de la aplicación de Spring y hacen referencia a atributos de clase que tienen la anotación **@Value** para inicializar las propiedades.

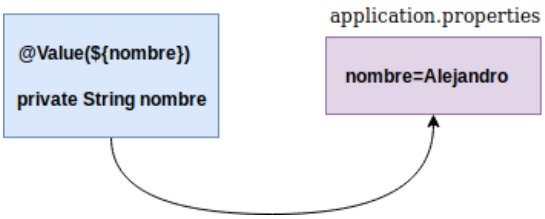

Figura 14 Correspondencia de un atributo definido en el archivo application.properties a un atributo de clase.

```
# Puerto de escucha de peticiones HTTPS
server.port=8443
# Activar SSL
server.ssl.enabled=true
# Ruta del almacén de claves y contraseña
server.ssl.key-store=src/main/resources/certificado/client-keystore.jks
server.ssl.key-store-password=espinosa
```

Cuadro 39 Configuración del servidor de aplicación para utilizar el protocolo SSL

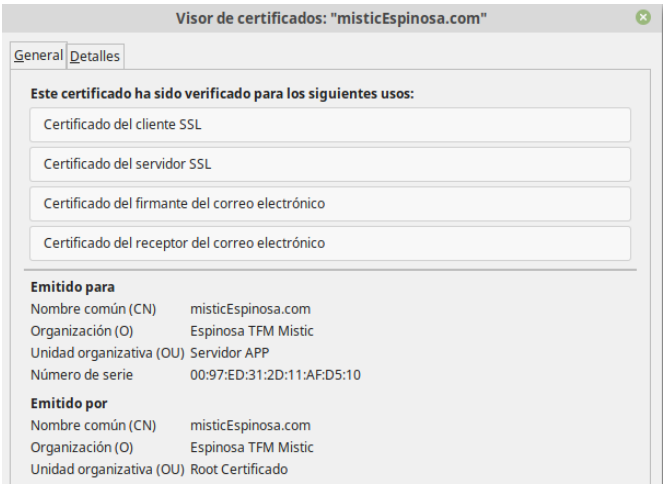

Figura 15 Visualización del certificado del servidor de aplicaciones.

5.4.4 Gestión de la autorización de los recursos

Para la autorización de los recursos, se utilizó SpringSecurity que es un componente esencial del proyecto de Spring que permite proteger aplicaciones de terceras personas que no dispongan de los roles requeridos. Básicamente resuelve dos problemas:

- **Autenticación**: Proceso por el cual un usuario valida unas credenciales contra el sistema y adquiere unos roles.
- **Autorización**: Proceso por el cual se le da permiso a un usuario para acceder a un recurso. Esto dependerá de los roles asignados.

En este proyecto, hay rutas que se tienen que proteger con Spring Security y otras que no pueden ser gestionadas directamente por Spring Security ya que no dependen de la aplicación. En ese sentido, se definen los recursos junto con los roles necesarios que se protegerán:

- **/index.jsp**. Es un recurso público al que se puede acceder sin realizar el proceso de autenticación.
- **/secured/menu.jsp**. Es un recurso protegido al que únicamente pueden acceder los usuarios autenticados.
- **/secured/recursoProtegido.jsp**. Es un recurso protegido y sólo pueden acceder los usuarios con rol **ROLE_ADMIN.**

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.httpBasic().authenticationEntryPoint(authenticationEntryPoint);

    http.authorizeRequests().antMatchers("/").permitAll();
    http.authorizeRequests().antMatchers("/logout.jsp").authenticated();
    http.authorizeRequests().antMatchers("/secured/**").authenticated();

http.authorizeRequests().antMatchers("/secured/recursoProtegido.jsp").access("hasRo
le('ROLE_ADMIN')");
}
```

Cuadro 40 Configuración de las rutas que se protegidas y del método de autentificación del cliente CAS.

En el método *configure,* se establece la autorización para cada recurso y configura el mecanismo de autentificación que se debe utilizar en caso de requerir autenticación por parte de un usuario que acceda a un recurso protegido.
Para realizar la protección de los recursos se pueden utilizar los siguientes métodos:

- **permitAll().** Un recurso definido con este método indica que será público y no necesitará autentificación.
- **authenticated().** Define que un usuario tiene que estar autenticado para acceder al recurso.
- **access("hasRole('ROLE_ADMIN')")**. Define que un usuario debe tener el rol ADMIN para acceder al recurso.

A continuación, la figura 16 se muestra la aplicación cliente una vez el usuario se ha autentificado en el sistema. Al haberse autentificado con el usuario alejandro que tiene todos los roles, el usuario puede acceder a todos recursos.

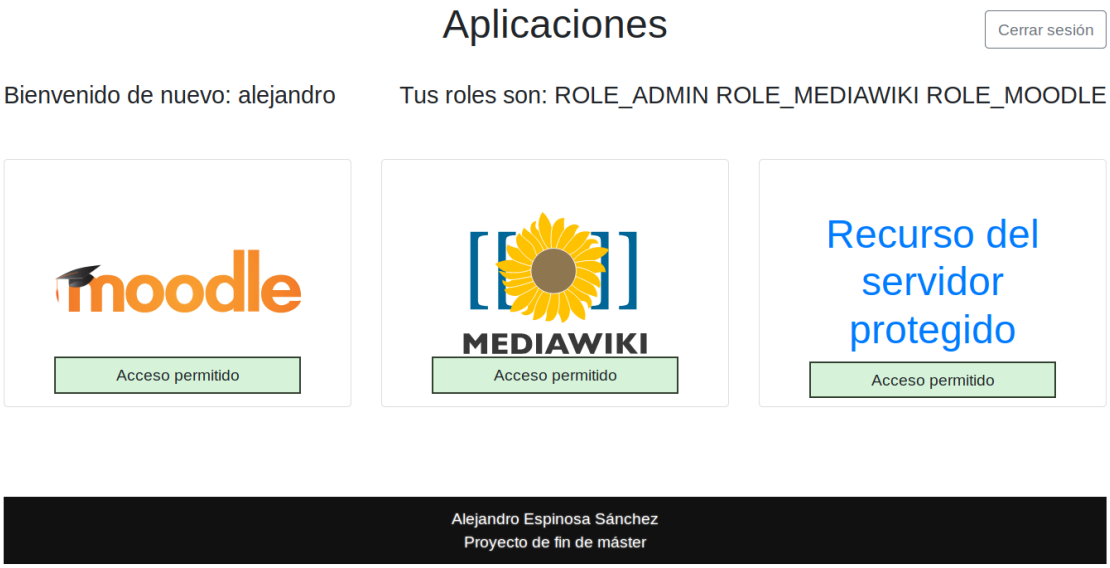

Figura 16 Visualización de las aplicaciones que tiene accesibles el usuario alejandro.

Sin embargo, cuando el usuario **emilia** se autentifique como tiene el rol **ROLE_MEDIAWIKI** únicamente podrá acceder al recurso MediaWiki.

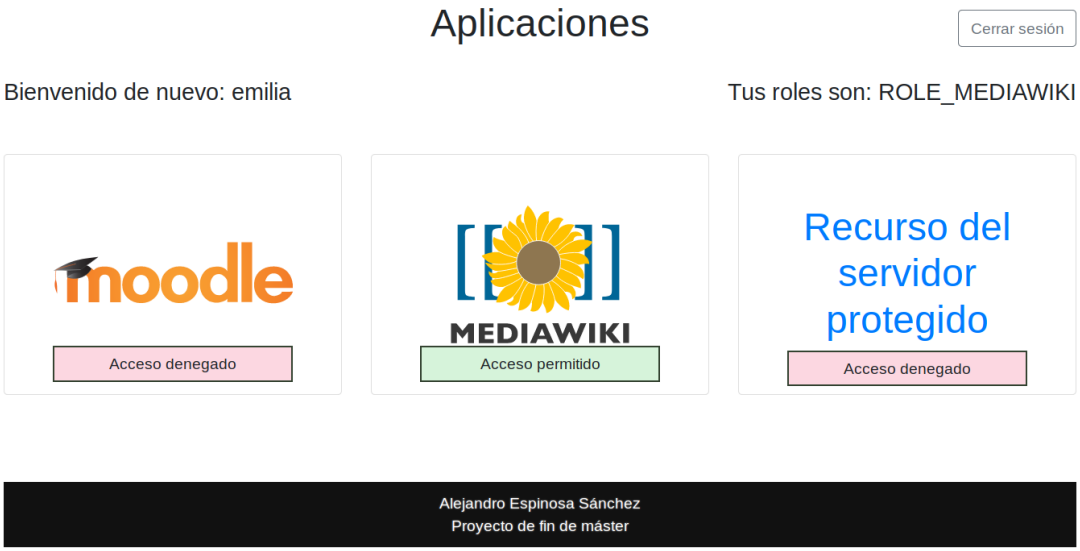

Figura 17 Visualización de las aplicaciones que tiene accesibles el usuario emilia.

5.5 Moodle

En esta sección se definen los pasos para configurar Moodle para que se autentifique con CAS partiendo de la base de que el servidor está funcionando con un usuario administrador local. Para activar la autenticación CAS, se debe:

1. Navegar a *Administración del sitio > Extensiones > Identificación > Gestión Identificación y pulsar en el botón de habilitar CAS.*

Figura 18 Métodos de autentificación en la plataforma Moodle.

2. En esa pestaña, para acceder al recurso que permite gestionar la configuración es necesario pulsar en el botón **Configuración**.

   Al pulsar en ese botón en esa pestaña, aparecen varias cajas de texto con parámetros configurables: servidor CAS, servidor LDAP y como se gestiona la asociación de los atributos obtenidos con el usuario autentificado.

   En la figura 19, se muestra la configuración para conectarse con el servidor CAS, la ubicación del recurso CAS y la versión del protocolo que se utilizará el servidor CAS.

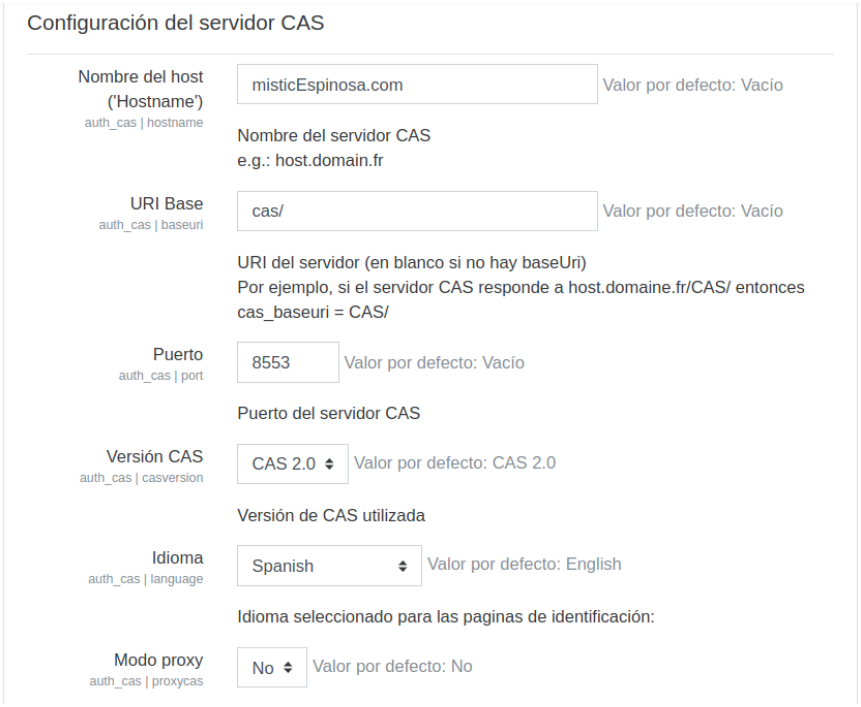

Figura 19 Parámetros de configuración de Moodle para autentificarse con CAS.

En la configuración de Moodle para interactuar con CAS (véase figura 20), se configuró para que efectuara consultas en LDAP con el objetivo de obtener información del usuario a autentificar.

Figura 20 Parámetros de configuración de Moodle para efectuar consultas en LDAP.

Siguiendo en la configuración de CAS para interactuar con LDAP, en la figura 21 se especifica la unidad organizativa donde se ubican los usuarios y el atributo de usuario que tiene la finalidad de identificar inequívocamente al usuario.

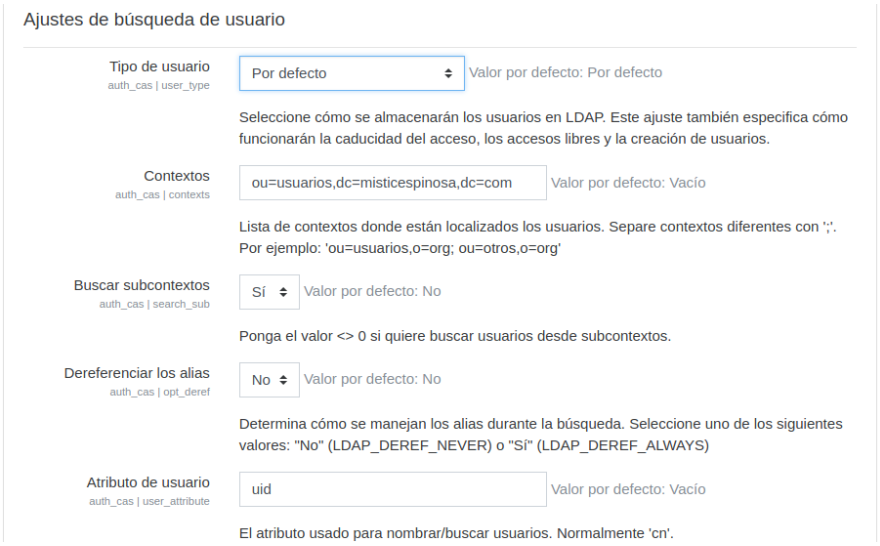

Figura 21 Configuración de Moodle para efectuar búsquedas en el directorio organizativo de los usuarios.

Como último paso en el proceso de configuración de Moodle con CAS fue definir la correspondencia de los atributos del propio modelo Usuario con los obtenidos del servidor CAS. Para ello, en la figura 22 se muestra que se asoció el nombre, los apellidos y el correo electrónico de cada usuario con los atributos del usuario de LDAP.

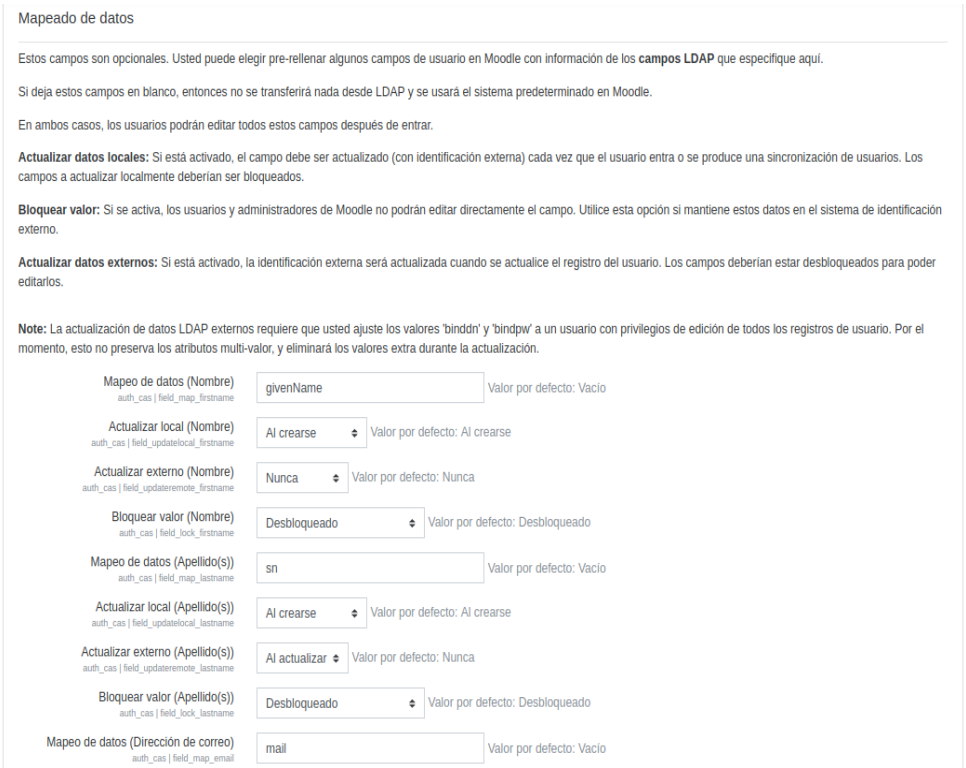

Figura 22 Asociación de los atributos obtenidos del servidor CAS con los atributos del modelo de usuario de moodle.

Finalmente, se efectuó una verificación de que Moodle autentificaba correctamente a los usuarios en el sistema. En la figura 23, se muestra que el usuario se autenticó correctamente, se realizó la asociación de atributos correctamente y el ticket generado del servidor CAS al autenticar al usuario en Moodle.

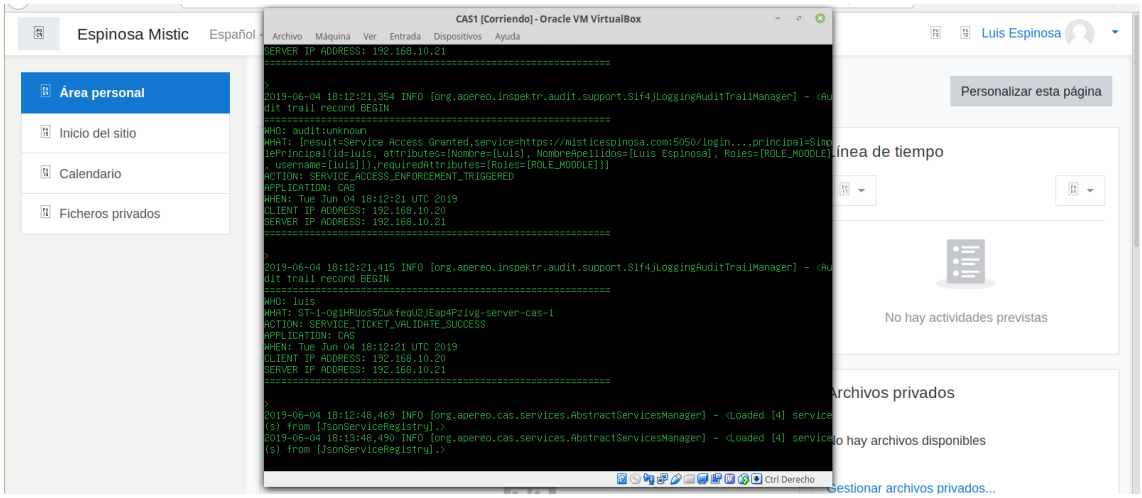

Figura 23 Verificación de la autenticación del usuario luis en Moodle utilizando CAS.

Finalmente, se verificó que un usuario sin el rol **ROLE_MOODLE** no pudiera autenticarse en CAS para acceder a Moodle. Esta restricción se definió en el archivo de servicio especificando que como argumento necesario era tener el rol **ROLE_MOODLE**.

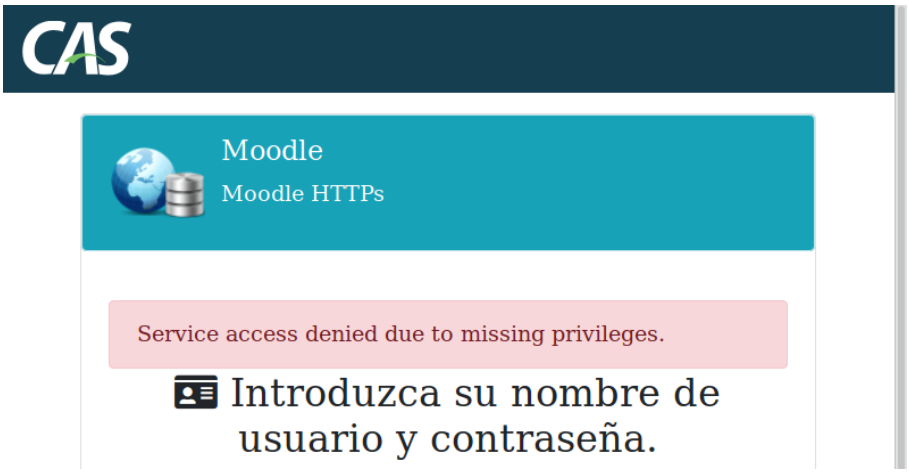

Figura 24 Comprobación que un usuario sin el rol ROLE_MOODLE no puede autenticarse en Moodle.

5.6 MediaWiki

En esta sección se detalla el proceso de configuración de Mediawiki para poder autentificar de forma centralizada a los usuarios utilizando CAS. Se parte de la base de que Mediawiki está funcionando sin autentificar a los usuarios por medio de CAS y seguidamente se especifican las acciones para configurarlo ya que es una extensión no oficial. Para configurarlo, se debe:

1. Crear una carpeta en el directorio donde se guardan las extensiones con nombre CASAuth/

2. Ejecutar la instrucción para clonar el repositorio CASAuth en esa carpeta. Este repositorio contiene la extensión que permite a MediaWiki gestionar la autentificación con CAS.

```
git clone https://github.com/CWRUChielLab/CASAuth.git
```

Cuadro 41 Instrucción para clonar el repositorio de la extensión CAS que permite autenticar usuarios en MediaWiki.

3. Descargar la extensión PhpCAS que define la lógica de interacción con CAS en MediaWiki y luego extraer los archivos en el directorio extensions/CASAuth/CAS/.

```
https://github.com/apereo/phpCAS.git
```

Cuadro 42 Instrucción para clonar el repositorio que implementa la interacción con CAS

4. Importar el archivo en el que se especifican las propiedades de CAS en el archivo de LocalSettings.php que proporciona las opciones de configuración básicas de Mediawiki.

```
require_once(“extensions/CASAuth/CASAuth.php" );
```

Cuadro 43 Instrucción para importar el archivo de configuración de CAS en MediaWiki

```
# Definición del servidor CAS
$CASAuth["Server"]="misticespinosa.com";

# Puerto de comunicación con el servidor CAS
$CASAuth["Port"]=8553;

# Recurso CAS
$CASAuth["Url"]="/cas/";

# Versión de CAS que se utilizará. Por defecto es la 2.0, pero se puede utilizar la
1.0.
$CASAuth["Version"]="2.0";

# Define que se creará una cuenta cuando un usuario se autentifique con CAS y no
tenga una cuenta previa.
$CASAuth["CreateAccounts"]=true;
```

Cuadro 44 Archivo de configuración de la extensión CAS para permitir la autenticación de MediaWiki

5.5 Cortafuegos

En esta sección se definirán los pasos para configurar el cortafuegos para crear y configurar las interfaces de red de tal forma que la infraestructura separe la zona pública de la interna para crear un equipo bastón de tipo ***Dual-homed***.

5.5.1 Instalación del cortafuegos

La instalación del cortafuegos se realizó en una máquina virtual, en donde se instaló la imagen **pfsense**. Al ser un software gratuito, se puede descargar desde la página del proyecto y seleccionar la arquitectura del sistema que se ajuste al equipo donde se va a instalar. En este proyecto, se usará la imagen de 64 bits en formato ISO.

Una vez descargado la imagen del cortafuegos, se crearon las interfaces de red en VirtualBox para separar la zona externa de la zona interna. Como se muestra en la figura 25, se creó un equipo bastión de tipo ***Dual-homed*** que se utiliza como filtro de paquetes. La ventaja de usar esta arquitectura es crear una separación entre la red externa e interna, lo que admite todo el tráfico de entrada y salida pase por el cortafuegos. De esta forma, este host evita que los atacantes intenten acceder a un dispositivo interno.

Figura 25 Configuración de los adaptadores de red del cortafuegos.

5.5.2 Configuración del cortafuegos

A continuación, se configuraron las interfaces de red con la opción 1 (Assign Interfaces). El objetivo de esta configuración fue especificar qué interfaz gestionaría la red pública y la red privada. En pfSense se puede configurar una interfaz de red de tres formas:

- **WAN**. Con esta asignación se indica que la red recibirá peticiones del exterior siendo la dirección IP que verán los usuarios que quieran acceder a los servidores internos.
- **LAN.** Con este nombre se especifica que estará ubicada en la zona interna de la red de área local de los servidores.
- **OPT.** Con esta nomenclatura se indica que la interfaz podrá ser de tipo WAN o LAN con las mismas propiedades de las redes anteriores.

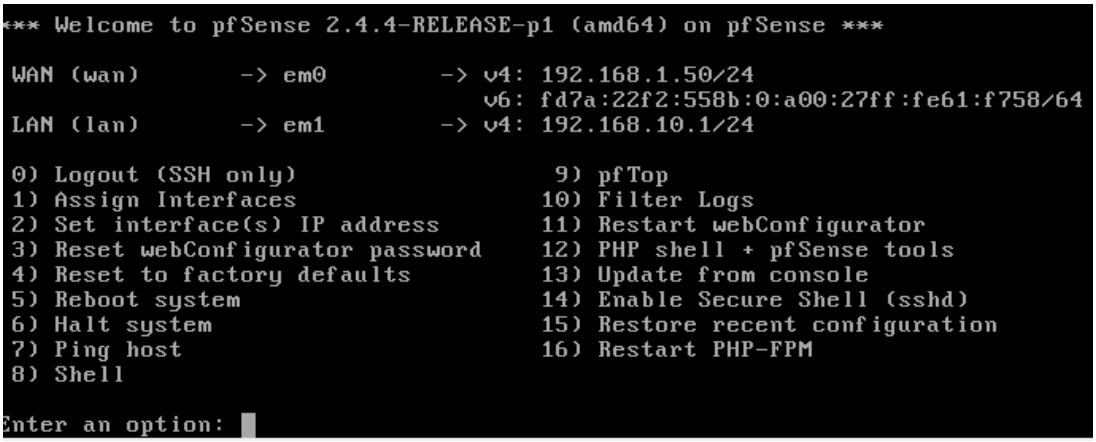

Figura 26 Asignación de las interfaces de red para la red pública y red de área local.

En este caso, se configuró una interfaz WAN y una LAN con la dirección IP correspondiente basándose en el esquema lógico definido en la arquitectura del proyecto (véase figura 26).

Interfaz	Tipo configuración	Dirección IP	Máscara de red	Gateway
WAN	em0	192.168.1.50	255.255.255.0	192.168.1.1
LAN	em1	192.168.10.1	255.255.255.0	-

Cuadro 45 Visualización de la dirección IP de la interfaz de red pública y local.

5.5.3 Configuración de redireccionamiento de puertos

La redirección de puertos es la acción de redirigir un puerto de red de un nodo de red a otro. Esta técnica habilita a que un usuario externo tenga acceso a un puerto en una dirección IP privada (dentro de una LAN) desde el exterior vía un enrutador con NAT activado. De esta forma, se permite que ordenadores remotos (por ejemplo, máquinas públicas en Internet) se conecten a un ordenador en concreto dentro de una LAN privada.

Para acceder a la configuración del cortafuegos, es necesario a través de un navegador web ubicado en la red local **`192.168.10.1`** ya que por defecto se deniega los paquetes del exterior para así evitar un acceso a los servidores internos.

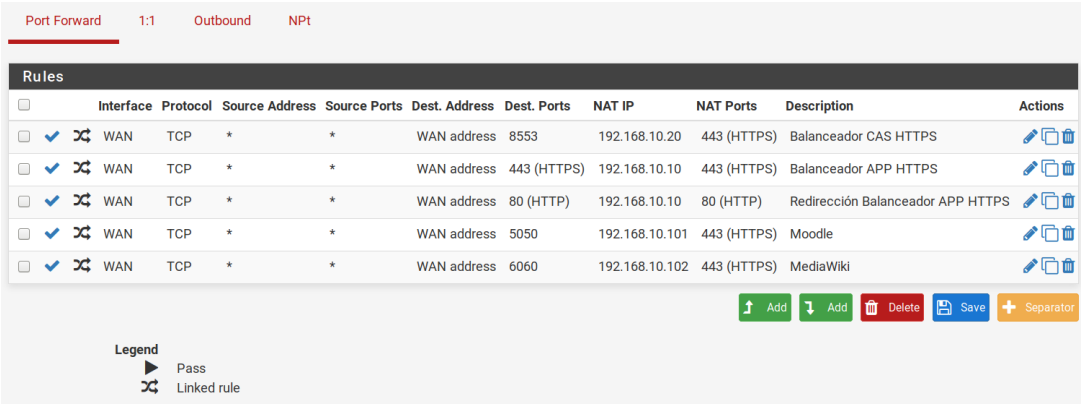

Figura 27 Reglas NAT creadas en el cortafuegos.

Para crear una regla que permita el redireccionamiento de puertos, se debe configurar los siguientes parámetros:

- **Disabled**: Atributo que permite activar o desactivar la regla.
- **Interface**: La regla se aplicará cuando lleguen paquetes por la WAN.
- **Protocol**: Define el tipo de protocolo (TCP o UDP) que activará la regla. En este caso, el transporte de la información se realiza por TCP.
- **Source**: Es un desplegable que permite configurar parámetros más avanzados de la dirección IP.
- **Destination**: Este atributo se utiliza para especificar la interfaz o red destino.
- **Destination Port Range**: Especifica el puerto del tráfico destino.
- **Redirect target IP**: Define la dirección IP destino.
- **Redirect Target Port**: Se utiliza para especificar el puerto donde se recibirá el tráfico en la dirección IP destino.
- **Description**: Este parámetro se utiliza para definir un texto autocontenido del objetivo de la regla.

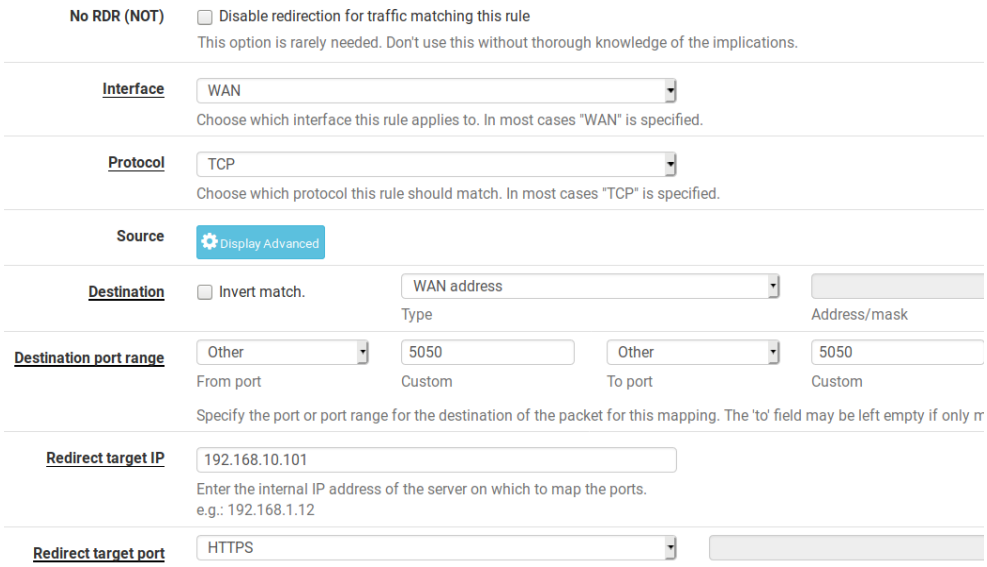

Figura 28 Asistente de configuración de una regla de redirección de tráfico en el cortafuegos.

En la figura 29 se muestra las reglas definidas en la tabla de reglas NAT para la interfaz WAN.

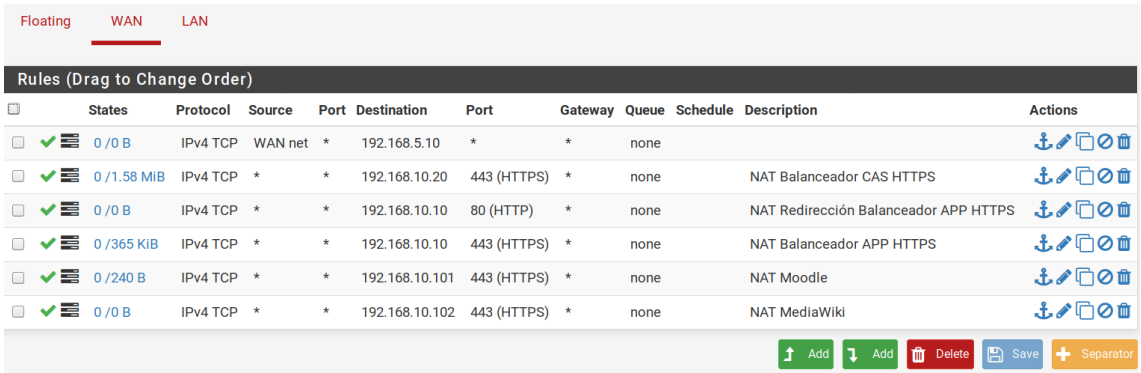

Figura 29 Reglas de redirección de puertos especificadas en el cortafuegos.

6 Conclusiones

En este último capítulo se hará una recapitulación de los objetivos propuestos inicialmente y los resultados obtenidos. El objetivo principal de este proyecto se cumplió satisfactoriamente, los usuarios se autentifican en un único punto que es centralizado para las tres aplicaciones. Además, se aplica una política de acceso basada en roles antes de autorizar el acceso a las aplicaciones en el servidor CAS para así tener un control total basado en atributos para habilitar el acceso.

Los objetivos secundarios no se completaron todos y otros se completaron parcialmente. El objetivo de permitir el acceso con un certificado digital se completó parcialmente, el servidor CAS solicita el certificado digital y autentifica al usuario, pero no sé logró obtener los roles del usuario del servidor LDAP. En este sentido, el no obtener los roles del usuario significa que no es posible efectuar un control de acceso por servicio y por tanto no es posible acceder a otras aplicaciones.

La metodología incremental permitió gestionar el tiempo para completar los objetivos planificados. En esa planificación, se introdujeron modificaciones en el tiempo estimado para garantizar el éxito del trabajo. En particular, no se tuvo en cuenta que la alta disponibilidad en el servidor CAS implicaba configurar un clúster en el que todos los nodos compartieran los tickets y ese punto fue el que más tiempo requirió ya que se tuvo muchos problemas de dependencias que finalmente se pudieron solucionar compilando previamente el servidor con un gestor de dependencias.

Las líneas de trabajo futuro pueden ser:

- Implementar el factor de doble autenticación para incrementar la seguridad. Por ejemplo, se podría utilizar la herramienta Google Authentification para permitir el acceso solamente si el usuario introduce un código enviado a través del teléfono móvil.

- Una posible implementación interesante, sería explotar los mecanismos de federación con entornos foráneos (Dropbox, Google, Facebook).

- Mejorar la seguridad del servidor CAS implementando un mecanismo que detecte múltiples accesos al sistema no satisfactorios y muestre un captcha con el objetivo de determinar cuándo el usuario es o no humano.

7 Bibliografía

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/222

https://www.jorgehernandezramirez.com/2017/03/08/spring-boot-spring-security-1er-part/

https://en.wikipedia.org/wiki/Infinispan

https://es.wikipedia.org/wiki/MongoDB

https://aws.amazon.com/es/elasticache/what-is-redis/

https://apereo.github.io/cas/6.0.x/configuration/Configuration-Properties.html#ldap-authentication

https://apereo.github.io/cas/6.0.x/ticketing/Hazelcast-Ticket-Registry.html

https://apereo.github.io/cas/6.0.x/installation/X509-Authentication.html

https://apereo.github.io/cas/6.0.x/services/AutoInitialization-Service-Management.html#service-registry-initialization

https://apereo.github.io/cas/6.0.x/configuration/Configuration-Properties.html#service-registry