Universitat Oberta de Catalunya

uoc.edu

Implantación de un SSO (Single Sign On)

Alfredo Franco Lara
Máster Universitario en Ciberseguridad y Privacidad
Seguridad Empresarial

Antoni González Ciria
Víctor García Font

Enero 2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implementación de un SSO*
Nombre del autor:	*Alfredo Franco Lara*
Nombre del consultor/a:	*Antonio González Ciria*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	*01/2023*
Titulación o programa:	Máster Universitario en Ciberseguridad y Privacidad
Área del Trabajo Final:	*Seguridad Empresarial*
Idioma del trabajo:	*castellano*
Palabras clave	*SSO, autenticación, autorización*

Resumen del Trabajo

Este trabajo fin de Máster consiste en la implementación de un sistema SSO (Single Sing-On) haciendo uso de herramientas Open Source. Con ello se pretende unificar el catálogo de aplicaciones disponibles en cualquier Administración o Empresa, permitiendo a los usuarios acceder a éstas simplemente autenticándose en el sistema una sola vez, y no recordando cada una de las credenciales de las aplicaciones que usa.

Para la implementación se estudian varias herramientas, valorando para cada una de ellas de forma determinante: la curva de aprendizaje, que esté basada en un lenguaje y entornos conocidos por el equipo de trabajo y que exista bastante documentación así como una comunidad que se encargue de ir actualizando la herramienta y corregir posibles vulnerabilidades reportadas.

La solución será implementada y probada en un equipo local, simulando las diferentes aplicaciones mediante máquinas virtuales, donde se mostrará cómo llevar a cabo la implementación y configuraciones necesarias para que el sistema funcione correctamente.

Abstract

This Master's thesis consists of the implementation of a SSO (Single Sing-On) system using Open Source tools. This is intended to unify the catalog of applications available in any Administration or Company, allowing users to access them by simply authenticating in the system only once, and not remembering each of the credentials of the applications used.

For the implementation several tools are studied, valuing for each of them in a determinant way: the learning curve, that it is based on a language and environments known by the work team and that there is enough documentation

as well as a community that is in charge of updating the tool and correcting possible vulnerabilities that are reported.

The solution will be implemented and tested in a local computer, simulating the different applications through virtual machines, where it will be shown how to carry out the implementation and configurations necessary for the system to work correctly.

Índice

Lista de figuras

1. Introducción

En la era de la información en la que vivimos, la mayoría de Empresas o Administraciones Públicas suelen poseer diferentes aplicaciones web que dan servicios a sus usuarios y empleados. Estas aplicaciones, incluso pueden estar desarrolladas en diferentes lenguajes.

En la mayoría de los casos, para poder acceder a estas aplicaciones, se suelen incorporar mecanismos de autenticación y autorización, donde habitualmente, la aplicación nos solicita unas credenciales (usuario y contraseña) como medida de seguridad. Son las propias aplicaciones las encargadas de gestionar la autenticación, normalmente validando las credenciales introducidas en una Base de Datos.

Todo esto se traduce a que un usuario debe tener tantas cuentas de autenticación como aplicaciones a las que tiene que acceder, teniendo éste que recordar cada una de las cuentas de acceso, con la posibilidad de que pueda olvidar alguna de ellas.

Una solución para este problema es el Single Sing On (SSO)[(3)]. Es una solución que permite a un usuario acceder a diferentes aplicaciones usando las mismas credenciales, de esta forma se puede compartir la información del usuario a través de diferentes dominios sin tener que obligar al usuario a identificarse cada vez que intenta acceder a éstos.

1.1. Contexto y justificación del Trabajo

El desarrollo del presente TFM se va a desarrollar en el contexto de una Administración Pública, en este caso, la Diputación Provincial de Cádiz. Administración en la cual me encuentro trabajando.

La Diputación está estructurada en dos áreas, el área de sistemas y el área de gestión. El área de gestión a su vez se divide en 5 áreas, *Administración Digital*, *Recaudación y Servicios Especiales*, *Recursos Humanos*, *Sistemas Públicos y Comunicación e Innovación*. Cada una de estas áreas tiene su propio equipo técnico, que se encarga del desarrollo y mantenimiento de cada una de las aplicaciones pertenecientes a cada área. Todas las aplicaciones para su acceso cuentan con un sistema de autenticación y autorización. Son estos equipos los encargados de implementar y mantener el sistema de autenticación y autorización de sus aplicaciones, sin haber un criterio común, desarrollándolo cada equipo de la forma que estima más adecuada, en cuanto a política de robustez en las contraseñas, sistema de recuperación, etc.

Actualmente la mayoría de empleados pertenecientes a la Diputación hacen uso de varias aplicaciones para desempeñar sus labores diarias, luego, cada uno de los empleados posee diferentes datos de autenticación para cada una de éstas aplicaciones, ya que cada una de estas credenciales se guardan en cada aplicación con las estructuras, condiciones, estándares y políticas que cada una definen. En ese caso se evidencian varios problemas:

1. Que los usuarios cuando llevan cierto tiempo sin acceder a una aplicación no se suelen acordar de las credenciales de acceso, lo que provoca que anoten éstas credenciales en sitios no seguros para poder

recordarlas con facilidad. Algo que va en contra del ENS (Esquema Nacional de Seguridad), el cual se está implantando.

2. Al tener cada una de las aplicaciones su propia implementación, es el equipo encargado de la misma el responsable de velar para que dicha implementación cumpla unos requisitos mínimos de robustez, política de cambio periódico de las contraseñas, etc. No habiendo un criterio unificado.
3. Mayor coste en horas de desarrollo al tener que implementar en cada una de las aplicaciones el sistema de login, diferentes equipos repiten mismas tareas para cada aplicación.
4. Mayor número de incidencias al CAU (Centro de Atención a Usuarios) para solicitar restablecer la contraseña. Muchas de las aplicaciones no tienen implementado un sistema de recuperación de contraseña, con lo cual los usuarios terminan abriendo incidencias cuando llevan cierto tiempo sin acceder por olvido de las credenciales.

A groso modo la infraestructura actual que dispone La Diputación es la siguiente

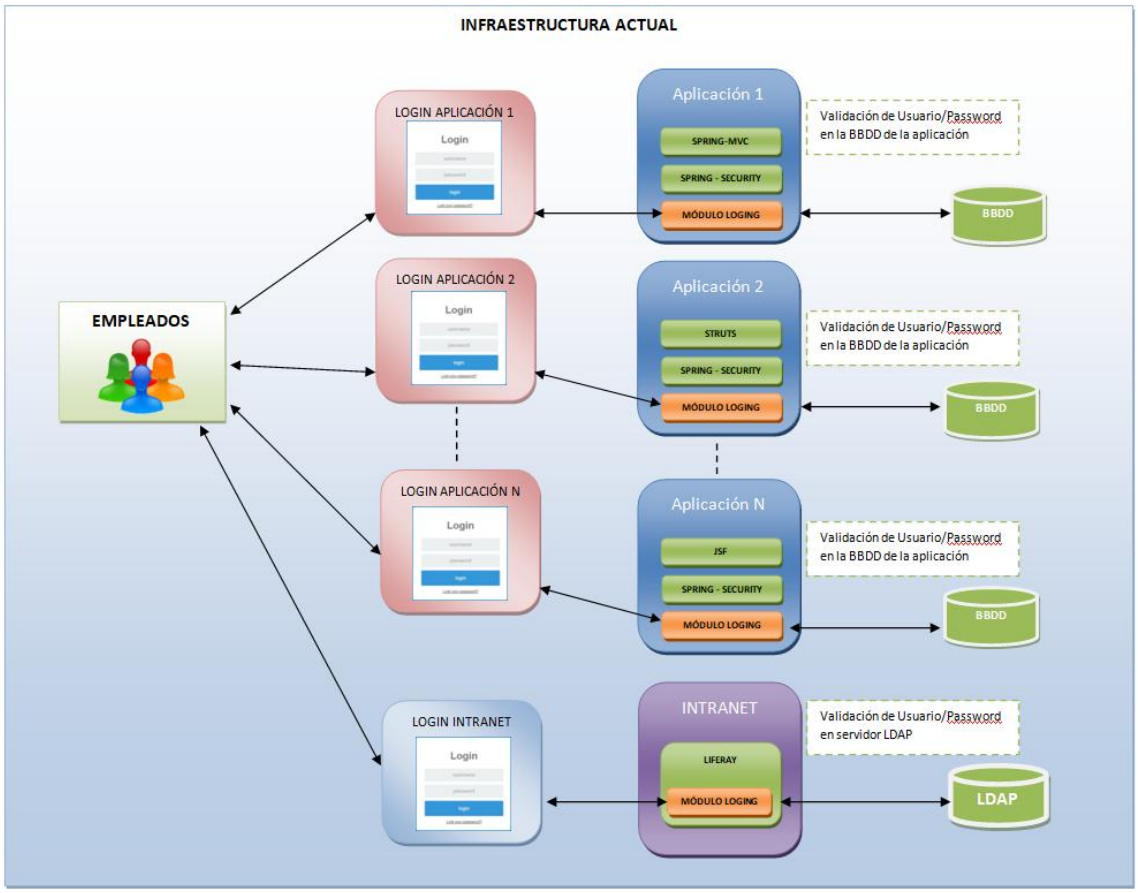


Figura 1: Arquitectura actual.

Como se puede observar en la figura, cada una de las aplicaciones suele usar frameworks distintos en la capa de presentación, aunque todas usan spring en la capa de negocio. Cada una de las aplicaciones implementa su propio formulario de login que suele usar por debajo spring-security para la autenticación, comprobando las credenciales en la BBDD que cada una tiene configurada. Todos los empleados actualmente se encuentran en un servidor

LDAP[4], siendo la intranet corporativa la única aplicación que emplea dicho servidor para autenticar a los usuarios que accede a ésta.

Para solventar estos problemas, la idea es unificar el sistema de login de todas las aplicaciones en un único lugar, para ello se desea hacer uso de un sistema de autenticación centralizado, es decir, un Single Sing-On (SSO)[3], basado en herramientas Open Source como CAS[6], JOSSO[8], OpenAM[9], etc.

Qué ventajas vamos a obtener con ello:

1. Delegar la implementación en un solo equipo de desarrollo, lo cual garantiza una mayor especialización en el sistema de seguridad implementado y un ahorro en costes al reutilizar éste para todas las aplicaciones.
2. Credenciales más seguras, ya que el SSO implementará mecanismos que obliguen a los usuarios a introducir contraseñas más seguras, y que les obligue cada cierto tiempo (por ejemplo 60 días) a modificarla.
3. Al tener una sola credencial para todas las aplicaciones los usuarios no tienen que memorizar/apuntar N contraseñas por cada aplicación que usan.
4. Cualquier actualización de la política de autenticación que se tenga que llevar a cabo sólo se desarrolla en un único lugar, ahorrando tiempo y unificando el criterio para todas las aplicaciones.
5. La mayoría de SSO provee mecanismos fuertes de autenticación, como puede ser el doble factor de autenticación o el acceso mediante certificado, con lo cual tendríamos un ahorro considerable en tiempo si se tuviera que aplicar alguno de estos criterios en todas las aplicaciones.
6. Ahorro considerable del número de incidencias que llegarían al CAU cuyo motivo sea restablecer la contraseña.

La empresa, en su día, estuvo analizando dos alternativas, CAS y JOSSO, decantándose en su momento por JOSSO, aunque por falta de recursos no se llegó a terminar la solución SSO. Por lo cual, para el desarrollo del presente TFM se ha decidido seguir el criterio que la empresa planteó en su día. Revisando la documentación de JOSSO observo que este componente ha sido adquirido por la empresa Atricore[1], con lo que pasa a ser de pago, ante esta situación, me he decantando por el uso de CAS (Central Authentication Service) ya que era el otro componente que la empresa estaba barajando, además encaja perfectamente para la solución que se desea implantar porque:

1. Es un producto Open Source.
2. Está basado en java y spring, lenguaje en el que está implementado todo el catálogo de aplicaciones web de la corporación, lo cual encaja perfectamente si es necesario extender o customizar el producto, reduciendo la curva de aprendizaje.
3. Existe bastante documentación en su sitio web y es un proyecto que parece estar bastante vivo, revisando su repositorio Git se puede observar que hay actualizaciones bastantes recientes. Es un punto a favor saber que hay una comunidad activa detrás velando por la

1 https://www.atricore.com/

evolución del producto y que en caso de vulnerabilidad sean capaces de sacar una actualización sin mucha dilación.

4. Soporta autenticación mediante LDAP y X.509, ya que la Diputación cuenta con un servidor LDAP donde se encuentran todos los empleados, además uno de los requisitos en la autenticación es el acceso mediante certificado digital o DNIe.
5. Soporta SAML(10) 1.1 y 2 (Security Assertion Markup Language), xml estándar para el intercambio de datos seguros de autenticación y autorización entre dominios. Requisito a tener en cuenta para posteriormente poderse integrar con Cl@ve[2]

A groso modo, el esquema de la infraestructura anterior podría quedar de forma resumida como se muestra en la siguiente figura

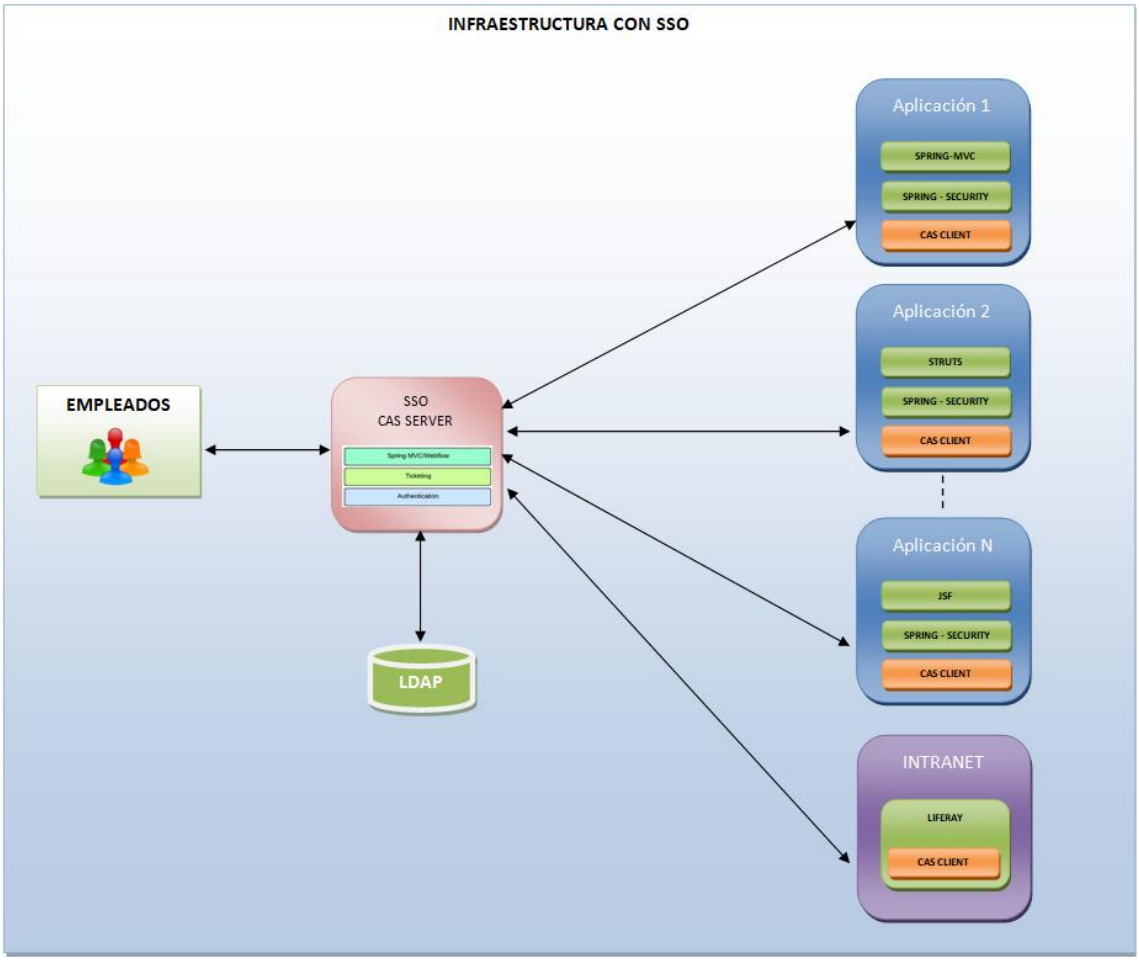


Figura 2: Arquitectura con SSO.

Podemos observar, cómo se ha simplificado todas las pantallas de login de cada una de las aplicaciones por el sistema SSO, CAS se encargará de orquestar las sesiones de autenticación mediante cookies que se grabarán en el browser del usuario. Estas cookies generadas por CAS son llamadas TGT (Ticket Granting Ticket) las cuales contienen un identificador único y un tiempo de expiración. Las cookies son destruidas una vez el usuario realiza un logout o cierra el browser.

2 https://administracionelectronica.gob.es/ctt/clave

1.1. Objetivos del Trabajo

- Implementar un sistema SSO haciendo uso de CAS (Central Authentication Service).
- Lograr que todas las aplicaciones web de la corporación se comuniquen con el servidor CAS, para ello se crearán dos aplicaciones piloto donde se compruebe que si nos logamos en una e intentamos acceder a otra aplicación, el servidor CAS, nos proporcionará un token que la aplicación debe validar dejándonos acceder sin necesidad de volver a introducir las credenciales.
- Configurar CAS para su integración con LDAP.
- Configurar CAS para poder autenticarse mediante certificados válidos de la FNMT y el DNIe, además de poder autenticarse mediante usuario/contraseña.
- Analizar un balanceador que proporcione alta disponibilidad (HA) para el SSO.
- Configurar el protocolo de comunicación para que éste sea seguro. Lo que implicaría que la comunicación vaya por https.
- Definir los posibles riesgos inherentes al sistema planteado y establecer el plan de mitigación para minimizarlos.

1.2. Impacto en sostenibilidad, ético-social y de diversidad

Según la competencia de compromiso ético y global (CCEG)[(2)], que define debemos actuar de manera honesta, ética, sostenible, socialmente responsable y respetuosa con los derechos humanos y la diversidad, tanto en la práctica académica como en la profesional, diseñando soluciones para mejorar estas prácticas, analizamos el impacto que tiene el presente TFM en cada una de las dimensiones que se deben considerar:

1. Sostenibilidad

   En esta dimensión podemos destacar el ahorro que se obtendría a nivel de recursos, ya que cada uno de los equipos de desarrollo encargados de las diferentes aplicaciones deben implementar en cada una de ellas el sistema de login, además de las posibles actualizaciones que se vayan realizando para adaptar dicho sistema a nuevas políticas de seguridad de la empresa. Esto se traduce, a que un solo equipo se encarga del desarrollo y mantenimiento del sistema de login mediante SSO, evitando tener diferentes personas realizando tareas similares, lo que supone un ahorro energético (iluminación de la oficina, consumo pc, consumo servidores de pruebas, etc.) que se podría estar empleando para el desarrollo de otras tareas.

2. Comportamiento ético y responsabilidad social

   En esta dimensión se tendría que analizar el conjunto de herramientas utilizadas para llevarlo a cabo. Por ejemplo, como herramienta SSO se ha seleccionado CAS, esta herramienta se distribuye bajo la licencia *Apache License Vesion 2.0*, luego a nivel ético, el producto resultante, si éste es modificado o personalizado debe tener en cuenta las limitaciones, permisos y condiciones que se imponen en ésta licencia.

3. Diversidad (género, entre otros) y derechos humanos

   En esta dimensión se destaca que con la implementación del SSO se garantiza que el sistema de login es totalmente accesible a todos los usuarios, haciendo que personas con cualquier discapacidad puedan logarse sin problemas, ya que al unificar la autenticación en un solo componente, nos aseguramos que si la pantalla de login es accesible, se aplica a todas las aplicaciones existentes y no se delega a cada aplicación que cumpla con dicha accesibilidad.

1.3. Enfoque y método seguido

El presente TFM parte desde cero, por lo que en este trabajo se deben abordar todas las tareas necesarias desde la elección y definición de la arquitectura a usar, hasta la implementación, pruebas e implantación de la misma.

Dentro de la puesta en producción de proyectos de desarrollo de software existen dos grandes grupos de metodologías para culminar con éxito el proyecto, las metodologías de desarrollo de software tradicionales y por otro lado, las ágiles, que son las más utilizadas a día de hoy.

Las metodologías ágiles se definen por su carácter ágil y flexible a los cambios, por lo que la posibilidad de fracasar u obtener unos malos resultados se reducirá considerablemente. En base a esto, elegimos ésta como metodología para abordar el presente TFM.

Ésta se basa en la metodología incremental, donde en cada ciclo de desarrollo se van agregando nuevas funcionalidades a la aplicación, con la particularidad de que los ciclos son mucho más cortos y rápidos, por lo que se deben ir agregando pequeñas funcionalidades en lugar de grandes cambios.

Las dos principales metodologías ágiles son:

- **Kanban[12]:** consiste en dividir las tareas en porciones mínimas y organizarlas en un tablero de trabajo dividido en tareas pendientes, en curso y finalizadas. De esta forma se crea un flujo de trabajo visual basado en las tareas prioritarias, incrementando el valor del producto.
- **Scrum[13]:** es una metodología incremental que divide los requisitos y tareas de forma similar a kanban. Normalmente se itera sobre tiempos cortos y fijos (entre dos y cuatro semanas) para conseguir un resultado completo de cada iteración. Las etapas son:
  - Planificación de la iteración (planning sprint)
  - Ejecución (sprint).
  - Reunión diaria (daily meeting).
  - Demostración de resultados (sprint review)

En este caso nos hemos decantado por usar **kanban**, ya que al ser más visual, de una simple mirada a nuestro tablero del TFM veremos el estado en el que estamos. Para el tablero vamos a usar la herramienta trello[3] donde se darán de alta cada una las tareas necesarias para culminar con éxito el TFM.

3 https://trello.com/

Para elaborar la planificación vamos a iterar cada sprint en función de las entregas planificadas en el campus virtual, con lo que tendremos los siguientes sprints:

- Sprint 1→ Entrega PEC 1: Plan de trabajo (11/10/2022).
- Sprint 2 → Entrega PEC 2: Entrega seguimiento 1 (08/11/2022).
- Sprint 3 → Entrega PEC 3: Entrega seguimiento 2 (06/12/2022).
- Sprint 4 → Entrega PEC 4: Memoria final (10/01/2023).
- Sprint 5 → Entrega PEC 5: Presentación en vídeo (17/01/2023).
- Sprint 6 → Entrega PEC 6: Preparación defensa TFM (27/01/2023).

1.4. Planificación del Trabajo

A continuación se muestra la planificación temporal teniendo presente las distintas entregas de las PECs incluidas en la planificación de la asignatura del TFM. Como se ha comentado anteriormente, la ejecución del proyecto se va a realizar siguiendo una metodología ágil, luego a la hora de definir las diferentes tareas que componen el proyecto, para poder identificarlas de forma visual, vamos a etiquetarlas con diferentes colores, estos colores son:

Figura 3: Etiquetas que representan los distintos Sprint.

De esta forma, visualmente podemos identificar a simple vista todas las tareas correspondientes a cada una de las entregas.

Para la elaboración de la planificación se ha usado el plugin “planyway”[4] de trello que es libre. Se muestra a continuación la planificación para cada uno de los sprints:

1. Sprint 1: Entrega PEC 1

4 https://planyway.com/

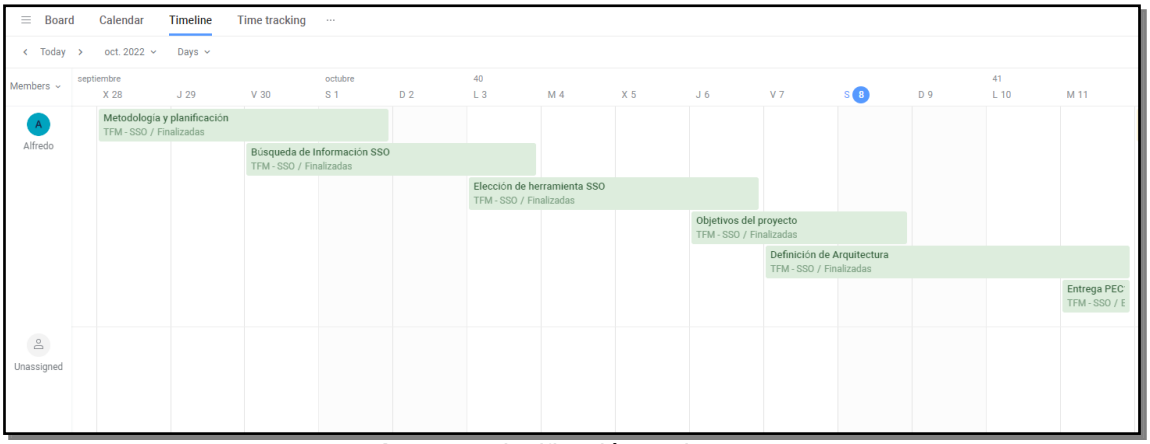


Figura 4: Planificación Sprint 1

2. Sprint 2: Entrega PEC 2

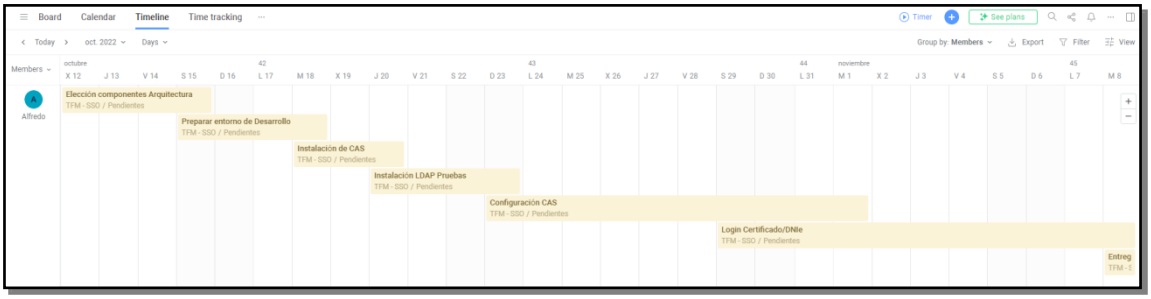


Figura 5: Planificación Sprint 2

3. Sprint 3: Entrega PEC 3

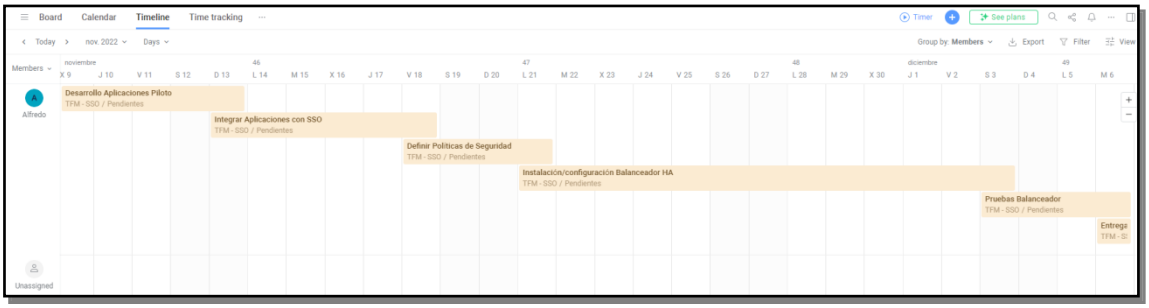


Figura 6: Planificación Sprint 3

4. Sprint 4: Entrega PEC 4

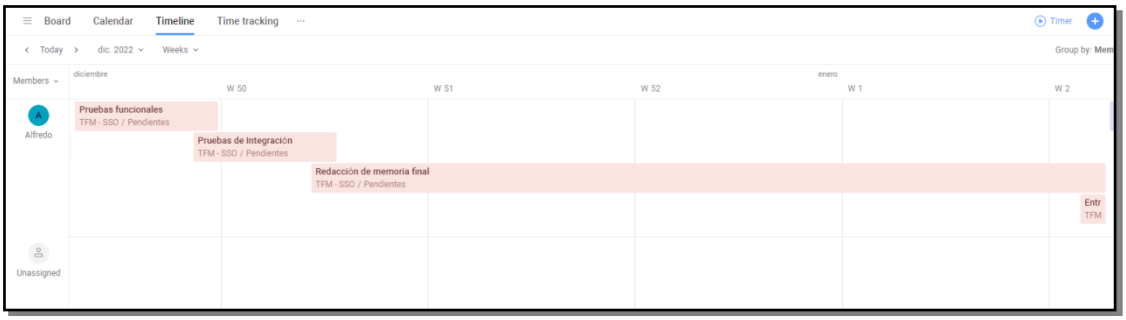


Figura 7: Planificación Sprint 4

5. Sprint 5: Entrega PEC 5

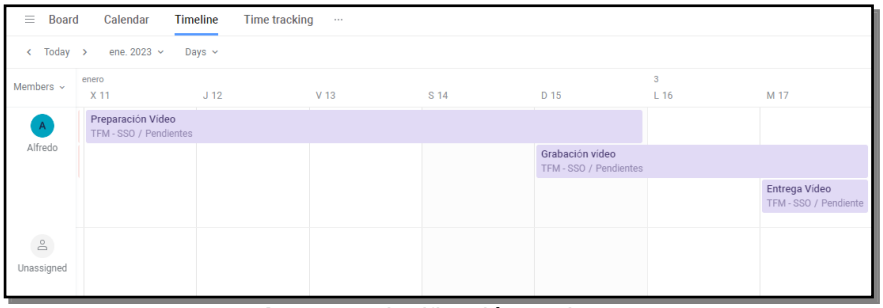


Figura 8: Planificación Sprint 5

6. Sprint 6: Entrega PEC 6

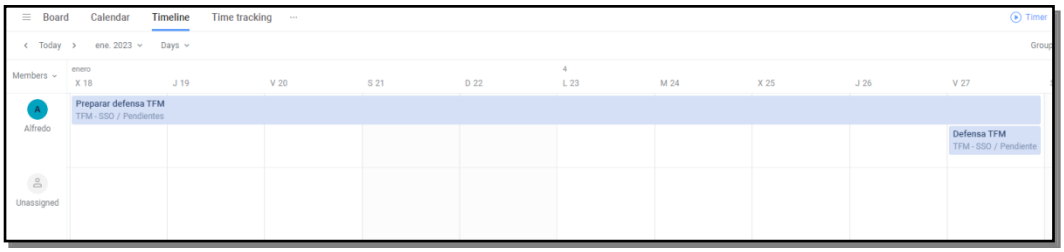


Figura 9: Planificación Sprint 6

A continuación se muestra una captura del tablero kanban en trello, el cual nos da una visión de las tareas que tenemos pendientes, las que tenemos en curso y las finalizadas

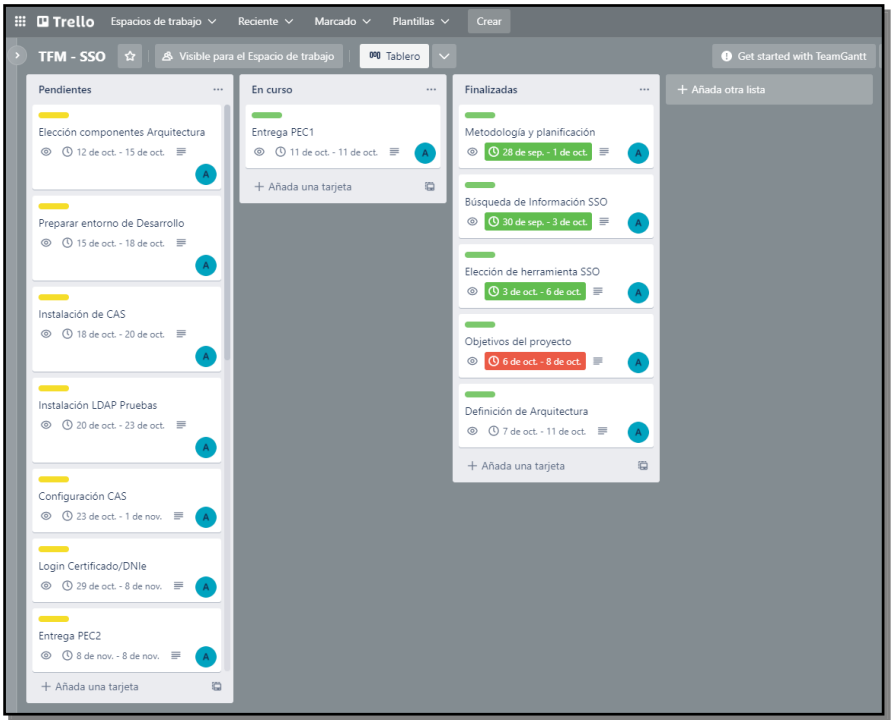


Figura 10: Tablero kanban

1.5. Breve sumario de productos obtenidos

A continuación se muestra un diagrama algo más técnico de la implementación final que se desea conseguir.

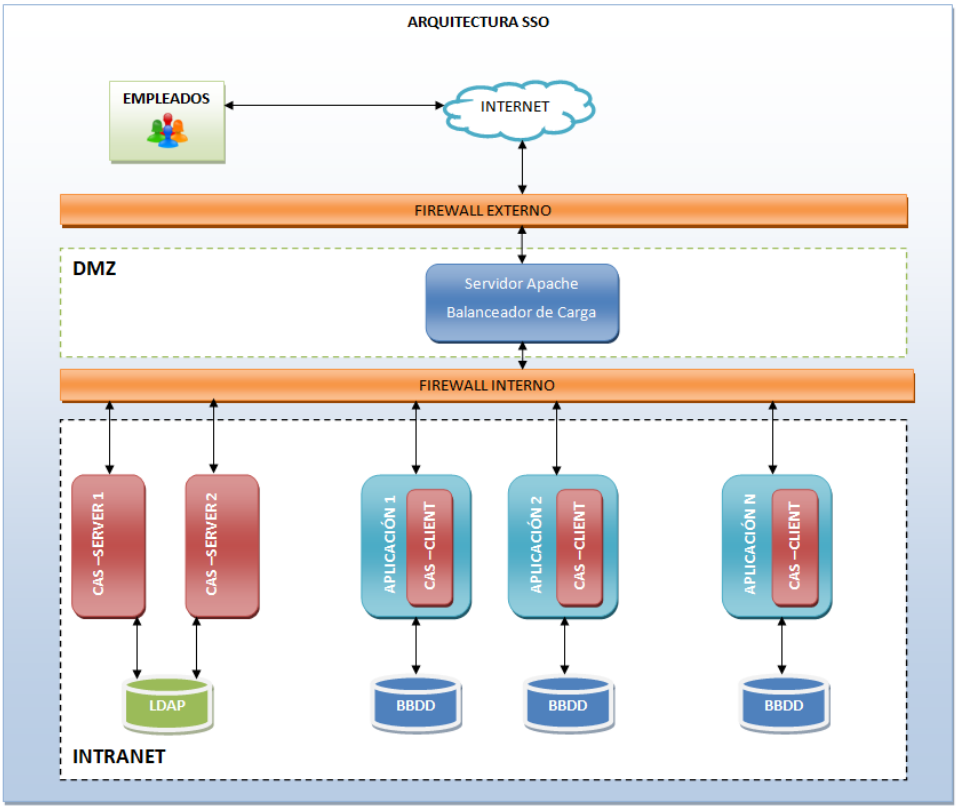


Figura 11: Diagrama con la arquitectura SSO

1.6. Breve descripción de los otros capítulos de la memoria

- *Capítulo 2. Análisis y Diseño del proyecto*. En este capítulo se realizará un análisis y diseño del sistema. Se analizarán y se seleccionarán los distintos componentes que formarán el sistema. Teniendo claro cada una de las herramientas, servidores, componentes de red, etc. que forman parte éste.
- *Capítulo 3. Implementación y Pruebas Unitarias*. En este capítulo se llevará a cabo las tareas de instalación, configuración, desarrollo y pruebas unitarias de cada uno de los componentes.
- *Capítulo 4. Pruebas*. En este capítulo se llevarán a cabo las pruebas de integración, donde se verificará que todos los componentes que forman el sistema funcionan correctamente.
- *Capítulo 5. Conclusiones y trabajos futuros.* En este capítulo se incluirán las conclusiones a las que se han llegado con éste TFM y los futuros trabajos que se deben realizar.
- *Capítulo 6. Glosario.* En este capítulo se incluirán todos los términos empleados que se consideren ser aclarados.
- *Capítulo 7. Bibliografía*. En este capítulo se incluirá todas las fuentes consultadas para la elaboración del presente TFM.
- *Capítulo 8. Anexos.* En este capítulo se incluirán los detalles de los apartados que son demasiados extensos para incluirlos en la memoria.

2. Análisis y Diseño del Sistema

2.1. Catálogo de Requisitos

Se detallan a continuación tanto los requisitos funcionales como los no funcionales que deben cumplir el sistema.

Como se ha comentado anteriormente La Diputación de Cádiz posee un catálogo de aplicaciones desarrolladas en Java, que se encuentran desplegadas sobre un contenedor web Apache Tomcat. Cada una de estas aplicaciones posee su propio sistema de Login, validando las credenciales del usuario (usuario/password) en base de datos. Todas estas aplicaciones poseen en BBDD una tabla que contiene las credenciales del usuario.

Con lo cual partimos de un catálogo de aplicaciones ya desarrolladas, en la que cada una de ellas implementa su propio sistema de autenticación y autorización, donde a cada usuario le tiene asignado una serie de permisos o roles que les permite acceder a ciertas opciones de menú o no según sus privilegios.

Partiendo de esta premisa, donde delegar el sistema de autorización al sistema SSO va a resultar mucho más costoso, ya que tendríamos que cargar todos los privilegios de los usuarios por cada aplicación al sistema LDAP, se opta por seguir delegando en cada una de las aplicaciones el sistema de autorización. En este aspecto el sistema SSO se usará sólo para autenticar al usuario y la aplicación una vez obtenga los datos del usuario autenticado procederá a comprobar en su BBDD si está autorizado para acceder a la aplicación.

Por ejemplo, imaginemos que el usuario “afrancol” intenta acceder a la aplicación 1, en ese caso la aplicación verificará que no está logado, redirigiendo la petición hacia el sistema SSO, el usuario se logará introduciendo sus credenciales y el sistema SSO pasará a la aplicación los datos del usuario logado, es aquí cuando la aplicación 1 debe validar si el usuario “afrancol” está autorizado para acceder a la aplicación, para ello, la aplicación buscará en BBDD si dicho usuario existe o no para autorizar el acceso. A continuación se muestra un diagrama de secuencia donde se puede observar el flujo.

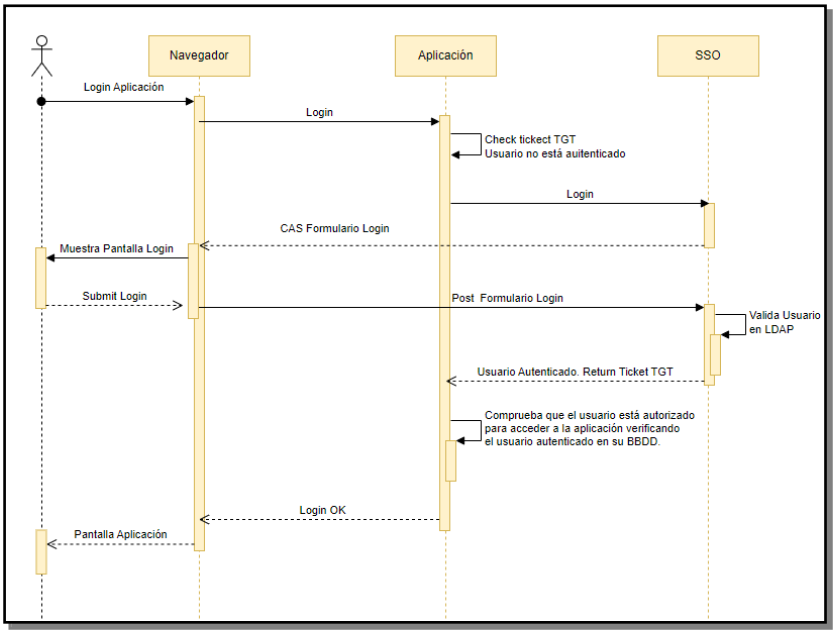


Figura 12: Diagrama de secuencia Login aplicación

Teniendo en cuenta estas premisas se detalla a continuación tanto los requisitos funcionales como los no funcionales.

2.1.1. Requisitos Funcionales

REQF-01: Acceso a aplicación mediante usuario/contraseña

Un usuario que previamente tenía acceso a una de las aplicaciones del catálogo debe seguir pudiendo acceder tras introducir sus credenciales en el sistema SSO.

REQF-02: Acceso a aplicación mediante Certificado Digital o DNIe

Un usuario que previamente tenía acceso a una de las aplicaciones del catálogo debe seguir pudiendo acceder tras logarse en el SSO con Certificado Digital o DNIe. Los certificados válidos para su acceso deben ser los emitidos por FNMT[5] o la Dirección General de la Policía para los DNIe.

REQF-03: Error de autenticación

Un usuario que no pertenezca a la Diputación no podrá acceder a ninguno de sus servicios. Para que el usuario pueda acceder debe estar dado de alta en el LDAP de la Corporación.

REQF-04: Error de autorización

Un usuario que no tenga acceso a una determinada aplicación no debe poder acceder. En este caso se debe mostrar un mensaje indicando que el usuario no posee permisos para acceder a la aplicación.

REQF-05: Implementación Aplicación 1

Se desarrollará una aplicación web de prueba que simule una de las aplicaciones del catálogo de la Diputación. Esta aplicación debe mostrar el usuario logado en la esquina superior y el mensaje “Bienvenido a la Aplicación 1”

5 Fábrica Nacional de Moneda y Timbre

REQF-06: Implementación Aplicación 2

Se desarrollará una aplicación web de prueba que simule una de las aplicaciones del catálogo de la Diputación. Esta aplicación debe mostrar el usuario logado en la esquina superior y el mensaje “Bienvenido a la Aplicación 2”

2.1.2. Requisitos No Funcionales

REQNF-01: Seguridad en las comunicaciones

La comunicación entre los distintos servidores debe ser segura, haciendo uso del protocolo https.

REQNF-02: Usuarios en LDAP

Los distintos usuarios y sus contraseñas encriptadas deben ser dados de alta en un servidor LDAP.

REQNF-03: Sistema de logging

Los sistemas debe registrar en un fichero de actividad todas las operaciones realizadas, usuario logado, IP desde la que accede, fecha y hora de acceso, etc.

REQNF-04: Implementación de Alta Disponibilidad

Se debe configurar el servidor CAS en alta disponibilidad en dos servidores.

2.2. Arquitectura del Proyecto

La arquitectura del proyecto es la mostrada en la Figura 11: **Diagrama con la arquitectura SSO**, donde se muestra los diferentes elementos que formarán parte de la solución. El sistema estará compuesto de:

- **Empleados**: Son los empleados de la Diputación, que accederán al catálogo de aplicaciones de ésta.
- **Firewall externo**: Se encargará de proteger las comunicaciones que llegan desde Internet hacia la DMZ.
- **Servidor Apache**[16]: Servidor HTTP Apache[17]. Este servidor hará de proxy inverso y de balanceador de carga, proporcionando un sistema de alta disponibilidad para el SSO.
- **CAS-Server 1**: Servidor donde se desplegará CAS[6] (Central Authentication Server), sistema SSO seleccionado.
- **CAS-Server 2**: Este servidor será una réplica del servidor anterior y su cometido es proporcionar el sistema de Alta Disponibilidad para el SSO.
- **Servidor LDAP**: En este servidor tendremos instalado OpenLDAP [18], servidor LDAP donde estarán dados de alta todos los empleados de la Corporación.
- **Aplicación 1**: Servidor de aplicaciones Tomcat [20] que contendrá la aplicación 1 de pruebas que hará uso del sistema SSO para su autenticación.
- **Aplicación 2**: Servidor de aplicaciones Tomcat [20] que contendrá la aplicación 2 de pruebas que hará uso del sistema SSO para su autenticación.

- **BBDD**: Servidor de Base de Datos que contendrá las bases de datos de las aplicaciones. En este caso se hará uso de HSQLDB.
- **Redes**: En el esquema se puede visualizar que existen dos redes diferenciadas, DMZ e Intranet, alojando en la zona DMZ los recursos que son accesibles desde el exterior y en la Intranet tendremos todos los recursos que no son accesibles desde el exterior.

2.3. Entorno

Para el desarrollo del presente TFM se usa un PC Portátil con las siguientes características:

- CPU Intel® Core™ i7-4500U 2.40 GHz
- RAM 8GB
- SO Windows 10 Home
- HDD SSD 480 GB

Para la simulación de los diferentes servidores se hará uso de máquinas virtuales, para ello usaremos la herramienta de virtualización **VMWARE WORKSTATION 16**[21]. De esta forma conseguimos poder simular nuestro entorno de desarrollo a un entorno de producción, ya que con la virtualización conseguimos aislar cada una de las máquinas respecto a las otras como si de máquinas físicas distintas se tratasen.

2.4. Componentes

A continuación se detallan cada uno de los componentes que se han seleccionado para implementar la solución.

2.4.1. Lenguajes de programación

Como se ha comentado en la introducción del presente TFM, la mayoría de aplicaciones que posee La Diputación están desarrolladas en **Java**, haciendo uso de los frameworks **Spring MVC** [22], **Spring-Security** [23] e **Hibernate** [24], con lo cual será éste el lenguaje seleccionado para la implementación de las aplicaciones pilotos que nos servirá de guía para la posterior configuración de las diferentes aplicaciones que constituyen el catálogo de aplicaciones de la Diputación. La idea es buscar una configuración estándar que sea común a todas las aplicaciones, por ejemplo, desarrollar una pequeña librería que nos facilite la configuración del los clientes CAS haciendo uso de web.xml[25] o anotaciones.

2.4.2. Servidor LDAP

LDAP [26] (Ligweight Directory Access Protocol) o “Protocolo Ligero de Acceso a Directorios” es un protocolo que permite el acceso a un servidor de directorio ordenado y distribuido para buscar información en un entorno de red. Por directorio nos referimos a una estructura jerárquica en forma de árbol donde se almacena la información.

Normalmente un servidor LDAP se encarga de almacenar información de autenticación del usuario (usuario y contraseña), pero también nos permite almacenar mucha más información, como datos de contacto, departamento al que pertenecen, ubicación, etc.

Los dos servicios LDAP más conocidos son *Active Directory de Windows* y *OpenLDAP*. En este caso nos decantamos por OpenLDAP[18] básicamente por ser open source y por ser uno de los servicios más populares del mercado.

2.4.3. Servidor HTTP Apache

Nos decantamos por Apache HTTP Server[17] como servidor de proxy inverso[27] y balanceador de carga[28] básicamente porque en la Corporación la mayoría de aplicaciones posee éste servidor por delante, para proteger la existencia y características del contenedor web donde se alojan las aplicaciones. En este caso el servidor recibirá las peticiones que van hacia las aplicaciones web y será éste el que según la url hacia la que va la petición decida hacia qué servidor de aplicaciones debe redirigir la petición, tomando la decisión de derivar la petición hacia el servidor que tenga menos carga para que la petición sea procesada lo más rápido posible.

2.4.4. Servidor CAS

Este será el servidor que proporcionará la funcionalidad Single Sign-On, centralizando la autenticación en este único punto. Está creado con Spring Framework y su principal función es autenticar a los usuarios y otorgar acceso a los servicios habilitados con el protocolo CAS[7], mediante la emisión y validación de tikects. A continuación se muestra un diagrama de flujo de una secuencia de autenticación para un primer acceso (cuando el usuario no dispone de sesión), como para un segundo acceso a la aplicación (cuando el usuario ya está autenticado en CAS Server).

Los conceptos claves en el protocolo CAS son:

- **TGT (Ticket Granting Ticket):** cookie TGC que se almacena, la cual representa la sesión SSO para el usuario.
- **ST (Service Ticket):** parámetro GET transmitido en las URLs, que representa el acceso otorgado por el servidor CAS a las aplicaciones CASsificadas para un usuario específico.

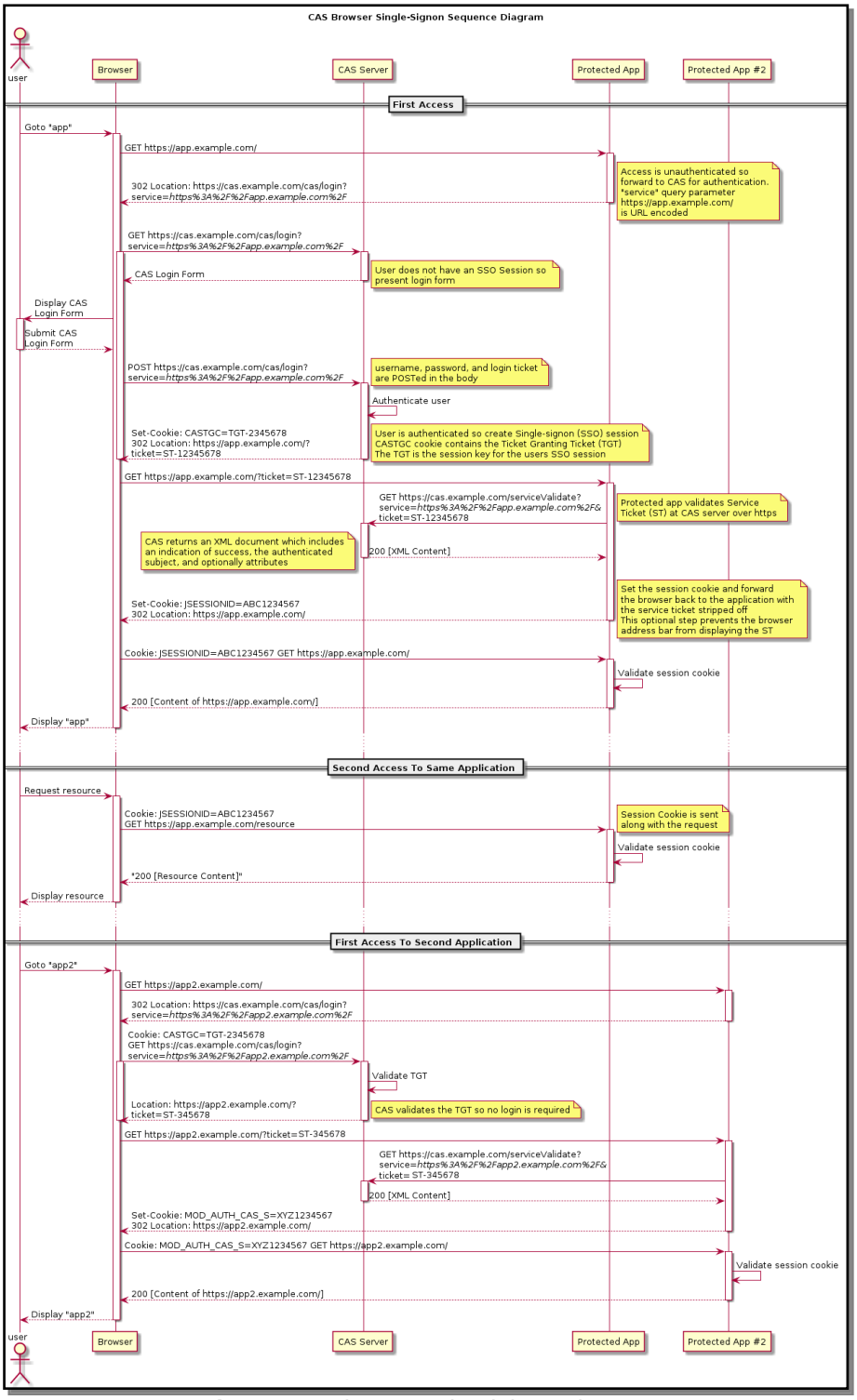


Figura 13: Diagrama de Flujo Login CAS

2.4.5. Servidor Aplicaciones

El servidor de aplicaciones será donde despleguemos las aplicaciones pilotos desarrolladas para probar la integración. En nuestro caso estos servidores serán **Apache Tomcat**(20), ya que en la Diputación todo el catálogo de aplicaciones se encuentran desplegados sobre éste servidor. Éste es un contenedor web con soporte para servlets[6] y JSP.

6 https://es.wikipedia.org/wiki/Java_Servlet

2.4.6. Servidor de Base de Datos

Este es un servidor de datos donde se crearan las BBDD de las que se nutren las aplicaciones web. En este caso se hará uso de MySQL[29] como motor de base de datos, donde se creará un esquema para cada una de las aplicaciones pilotos creadas, en este caso hago uso de una instalación de MySQL que poseo en mi PC Windows, donde sólo será necesario crear los esquemas con las tablas de pruebas. En Diputación todas las BBDD de las aplicaciones están alojadas en Oracle, pero haremos uso de MySQL en las fases de desarrollo y pruebas por temas de licencias.

2.4.7. Firewalls

Los firewall[30] son unos dispositivos de seguridad de la red que monitorea el tráfico entrante y saliente, decidiendo si permite o bloquea el tráfico en función de un conjunto de reglas definidas. En el presente TFM no vamos a entrar en la valoración y configuración de ellos, ya que se nos escapa del ámbito que nos hemos planteado, centrándonos en la implantación de CAS como sistema SSO y su configuración.

3. Construcción e Implementación

En esta fase se detalla la construcción e implementación del sistema.

3.1. Instalación y configuración del servidor CAS

Para las personalizaciones y configuraciones que podamos ir realizando del servidor CAS se ha creado una máquina virtual con una imagen de Ubuntu 22.04.1 TSL[7] donde se realizarán las pruebas pertinentes.

Los requerimientos mínimos de la máquina para la instalación de CAS server son:

- Java 1.7. JDK
- Contenedor Servlet. Apache Tomcat es el más utilizado por la comunidad
- Herramienta de construcción WAR Overlays[8], la cual nos proporcionará una forma sencilla y flexible de sobrescribir la configuración o personalizar el servidor.

Como se ha mencionado anteriormente, CAS provee WAR Overlay (Superposición de WAR) para combatir la repetición de código y recursos. Además de proporcionarnos todo el código base nos permite descargar un servidor CAS pre-construido al cual le podemos insertar/sobrescribir un comportamiento específico.

La mayoría de los aspectos de CAS pueden ser controlados añadiendo, eliminando o modificando ficheros en la superposición. También es posible y de hecho muy común personalizar el comportamiento de CAS agregando componentes de terceros que implementan las APIs de CAS como ficheros Java o referencias de dependencia.

El proceso para trabajar con overlay se puede resumir en los siguientes pasos:

1. Se comienza con la construcción básica que nos proporcionan.
2. Se identifican los artefactos de la construcción básica que necesitan cambios. Estos artefactos son generados normalmente por la compilación en el directorio ***build*** de gradle[9]. Posteriormente podemos ejecutar la tarea gradle ***unzip*** para descomprimir el war generado y poder examinarlo.
3. Se copian los artefactos identificados para ser modificados en el directorio ***/src/main/resources***
    a. Si el directorio ***/src/main/resources*** no existe procedemos a crearlo.
    b. Las rutas y nombre de los ficheros copiados deben ser exactamente iguales a las rutas y ficheros que se desean sobrescribir, ya que de lo contrario la superposición no surtirá efecto.

7 https://ubuntu.com/download/desktop

8 https://apereo.github.io/cas/development/installation/WAR-Overlay-Installation.html

9 https://es.wikipedia.org/wiki/Gradle

4. Después de realizar los cambios se procede a reconstruir, donde se generará un nuevo war que contendrá los cambios realizados, pudiendo repetir éste proceso tantas veces como se desee.

Para una primera toma de contacto con CAS procedemos a instalar tanto Java como Git en la máquina virtual y posteriormente procedemos a descargarnos CAS Overlay Template 6.6.1. El detalle de la instalación se puede seguir en el anexo Instalación CAS Overlay Template

3.2. Instalación y carga de datos en OpenLDAP

La instalación del servidor OpenLDAP la llevamos a cabo en una máquina virtual destinada a servidor de datos, en este caso el sistema operativo que corre en la máquina es un Ubuntu Server 22.04.1 LTS. El proceso de instalación se puede seguir en el anexo Instalación OpenLDAP.

Una vez instalado el servidor LDAP procedemos a cargar algunos datos de empleados para realizar las pruebas. Para ello lo primero que hacemos es definir la estructura de directorios que se pretende dar de alta, la cual será similar a la siguiente imagen

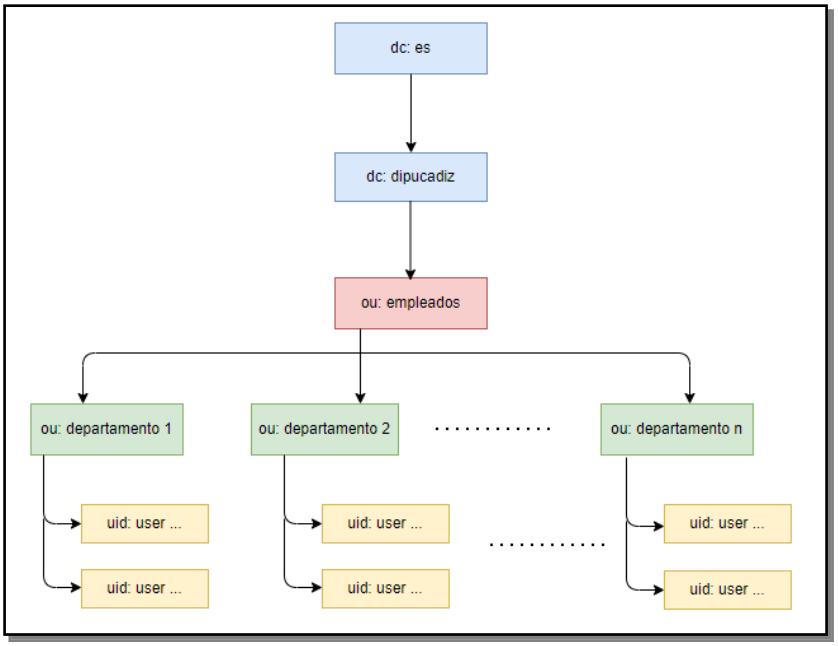


Figura 14: Estructura Directorios LDAP

El proceso de creación de la estructura correspondiente y la carga de los usuarios se puede ver en detalle en el anexo Creación de estructura y carga de datos en LDAP

3.3. Integración de CAS y OpenLDAP

Una vez tenemos nuestro servidor CAS y LDAP instalado y configurado con algunos usuarios de pruebas, es necesario configurar CAS para establecer LDAP como método de autenticación. De esta forma CAS es capaz de autenticar el usuario/contraseña introducido contra el servidor OpenLDAP. Los pasos a seguir para su integración son[31]:

- Lo primero es incluir la dependencia *cas-server-support* en el fichero *build.gradle* del proyecto *cas-overlay-template*, tal como se muestra en la siguiente captura.

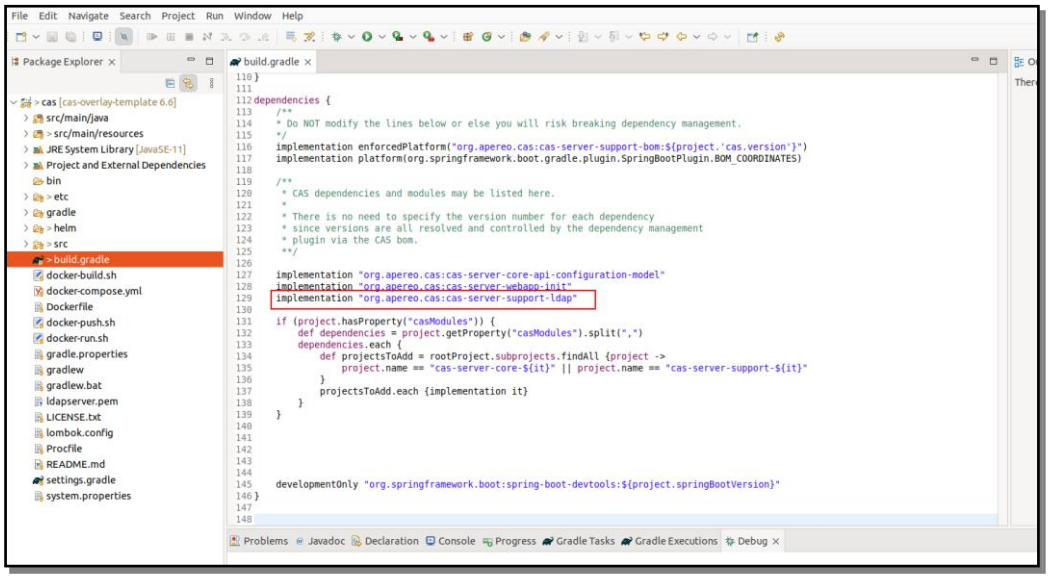


Figura 15: Integración LDAP. Añadir dependencia en el fichero build.gradle

- En el fichero *application.yml* se debe configurar las propiedades de acceso al LDAP, las propiedades establecidas en este caso son:
    - *cas.authn.ldap[0].type*: Tipo de autenticación, en nuestro caso establecemos el valor AUTENTICATED, ya que de esta forma los datos introducidos en el formulario de login serán buscados en LDAP para verificar que dicho usuario existe.
    - *cas.authn.ldap[0].ldap-url*: La url del servidor LDAP.
    - *cas.authn.ldap[0].base-dn*: Aquí se introduce el nombre distinguido que contiene la ruta completa del objeto que se desea buscar
    - *cas.authn.ldap[0].bind-dn*: El dn de enlace que se usará al conectarse al LDAP.
    - *cas.authn.ldap[0].search-filter*: Es el filtro a usar para las búsquedas.

A continuación se muestra una captura de las propiedades incluidas en el fichero *application.yml*

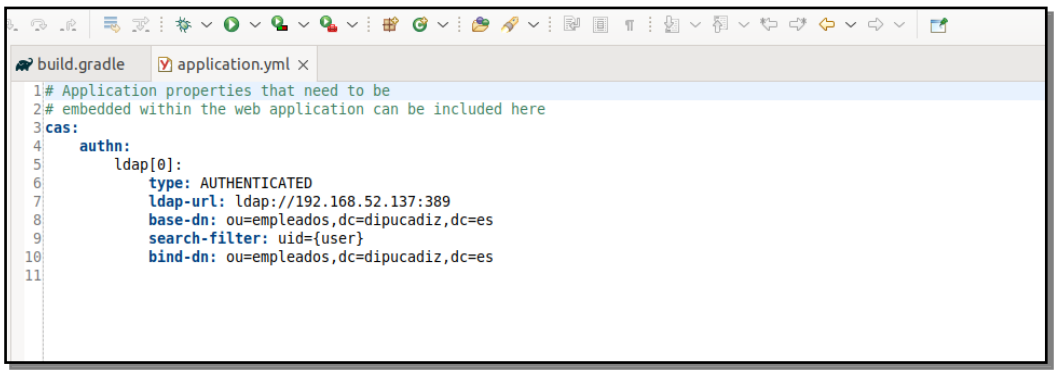


Figura 16: Integración LDAP. Fichero de configuración application.yml

Seguimos los consejos de CAS para la solución de problemas y habilitamos el logging adicional del conector *ldaptive*, modificando el nivel de trazas para LDAP a “debug”, tal como se muestra en la siguiente figura

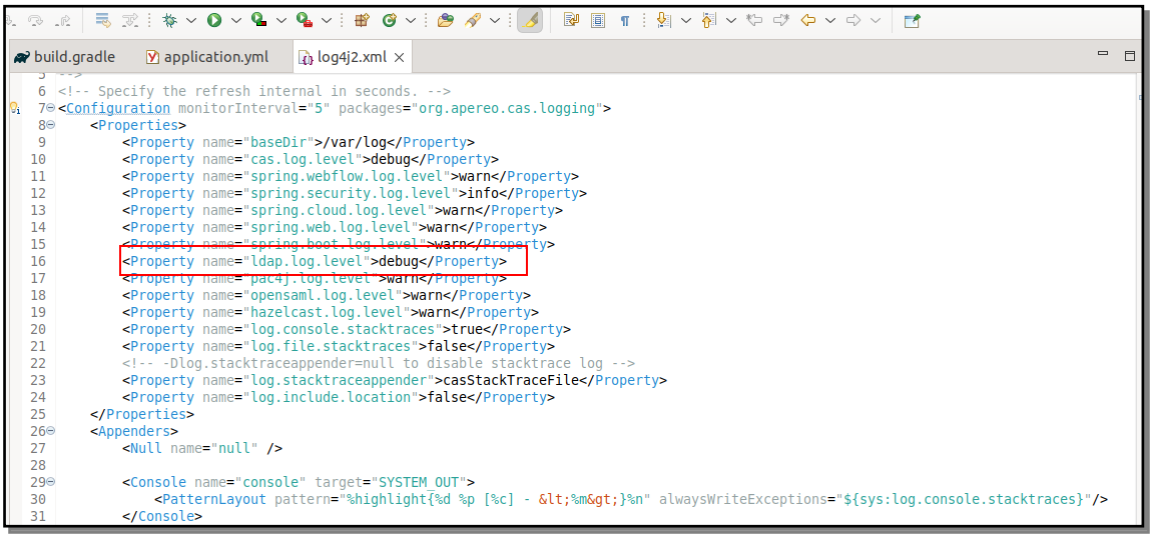


Figura 17: Integración LDAP. Fichero de configuración log4j2.xml

Para verificar que toda la configuración es correcta probamos que el servidor arranca sin problemas e introducimos un usuario/password de los que hemos dado de alta en LDAP. Se comprueba que el servidor arranca correctamente y tras introducir un usuario éste se loga satisfactoriamente. A continuación se puede observar en las propiedades de autenticación cómo el manejador de autenticación es *LdapAuthenticacionHandler*

Figura 18: Integración LDAP. Datos de inicio de sesión.

Si revisamos el log del servidor también podemos ver cómo el adaptador de conexión al LDAP ha encontrado al usuario introducido.

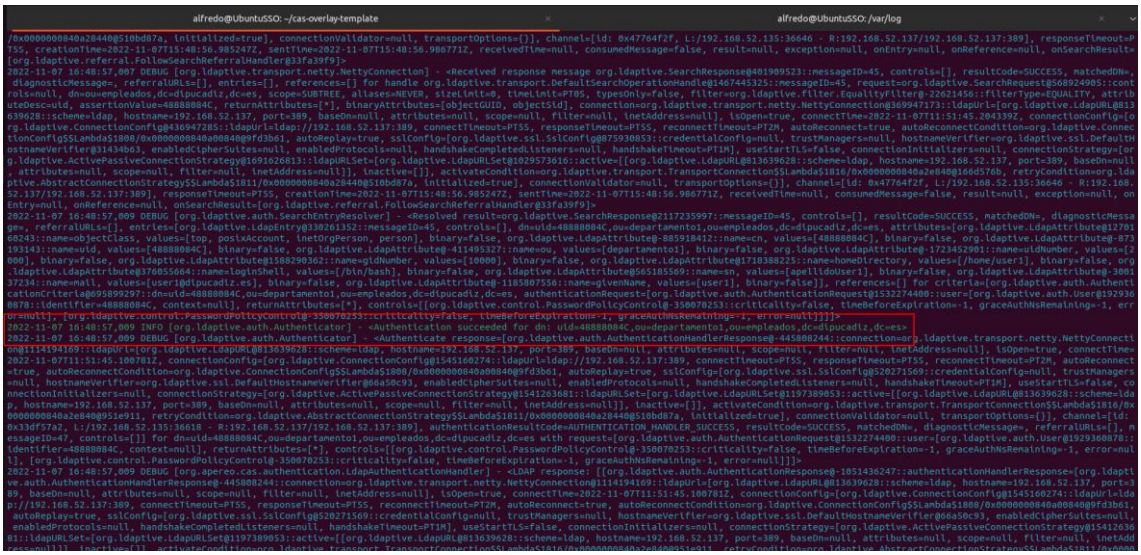


Figura 19: Integración LDAP. Log tras autenticación.

3.4. Autenticación mediante certificado digital y DNIe.

En este apartado vamos a realizar las configuraciones necesarias para que CAS permita el uso de certificados digitales o DNIe para la autenticación. En este caso debemos configurar el servidor embebido que posee CAS para indicarle al navegador que solicite uno de los certificados que éste tenga instalados, a groso modo el procedimiento sería:

1. Accedemos a la pantalla de login.
2. El servidor le solicita un certificado digital emitido por algunas de las Autoridades Certificadoras (CA) en las que el servidor confía.
3. El navegador comprueba si tiene algún certificado digital de alguna de las CA, en caso de poseer más de uno muestra un diálogo para que el usuario seleccione el certificado con el que desea interactuar.
4. El usuario selecciona el certificado y es cuando los datos del certificado llegan a la capa donde reside la aplicación CAS, ya que toda la negociación anterior va a nivel de la capa SSL/TLS del servidor.

A continuación se muestra cómo podemos configurar CAS para conseguir esto.

Lo primero que debemos hacer es habilitar el soporte X.509[32] incluyendo la dependencia ***cas-server-support-x509-webflow***, en el WAR overlay. Incluimos la dependencia en el fichero *build.gradle* al igual que hicimos anteriormente para la dependencia de LDAP.

El siguiente paso será configurar el servidor embebido Apache Tomcat que posee CAS para lograr dicha autenticación. Tendremos que crear nuestro almacén de confianza para que acepte los certificados emitidos por la FNMT y DNIe. Para ello nos debemos descargar los certificados raíz de la FNMT desde aquí y los certificados raíz del DNIe desde aquí. Una vez descargados se almacenan todos en un fichero JKS, bien mediante la herramienta *keytool* de java o mediante alguna aplicación de escritorio, en mi caso he usado *KeyStore Explorer*[10] donde tenemos una interfaz más visual del contenido del fichero que

10 https://keystore-explorer.org/

estamos creando. Una vez incorporado todos los certificados descargados tenemos que nuestro almacén contiene lo siguiente

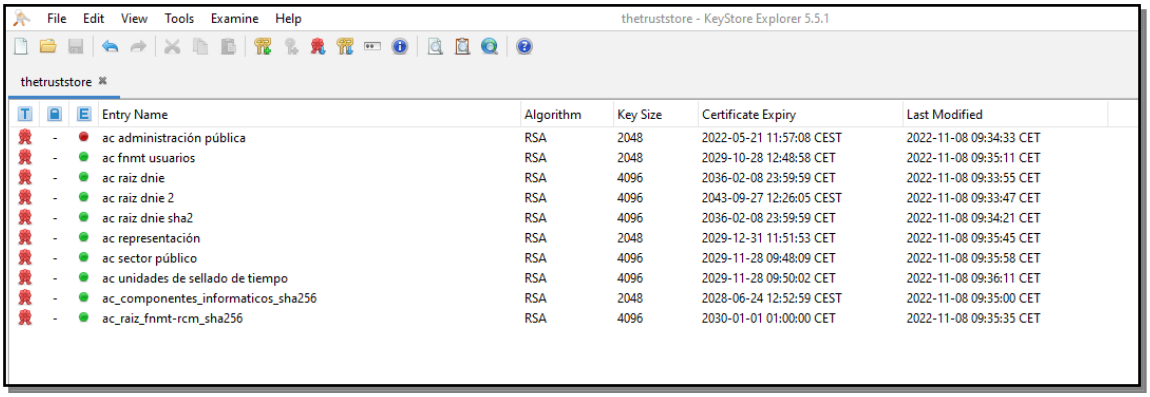


Figura 20: Configuración X.509. Certificados del almacén de confianza.

Posteriormente configuramos las propiedades necesarias en CAS para habilitar dicha autenticación. Para ello basta con indicar las siguientes propiedades[31]:

- *server.ssl.trust-store:* en esta propiedad establecemos la ruta del almacén de confianza que hemos generado con los certificados raíz.
- *server.ssl.trust-store-password:* contraseña del almacén de confianza.
- *server.ssl.client-auth:* aquí indicamos el modo de autenticación, en nuestro caso indicamos ***want***, que indica que la autenticación mediante certificado X.509 es requerida pero no obligatoria, de esta forma si no seleccionamos ningún certificado para logarnos, CAS nos mostrará la pantalla de login donde podemos introducir nuestras credenciales de forma alternativa.

A continuación se muestra nuestro fichero de configuración *application.yml* una vez incluida dichas propiedades.

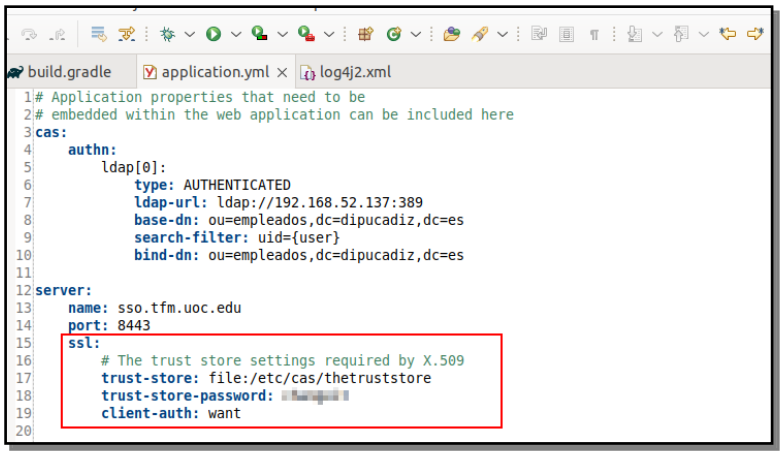


Figura 21: Configuración X.509. Fichero de configuración

Realizamos una prueba para verificar que la configuración es correcta, donde podemos comprobar que nada más acceder el navegador nos muestra diálogo solicitando el certificado con el que nos deseemos autenticar.

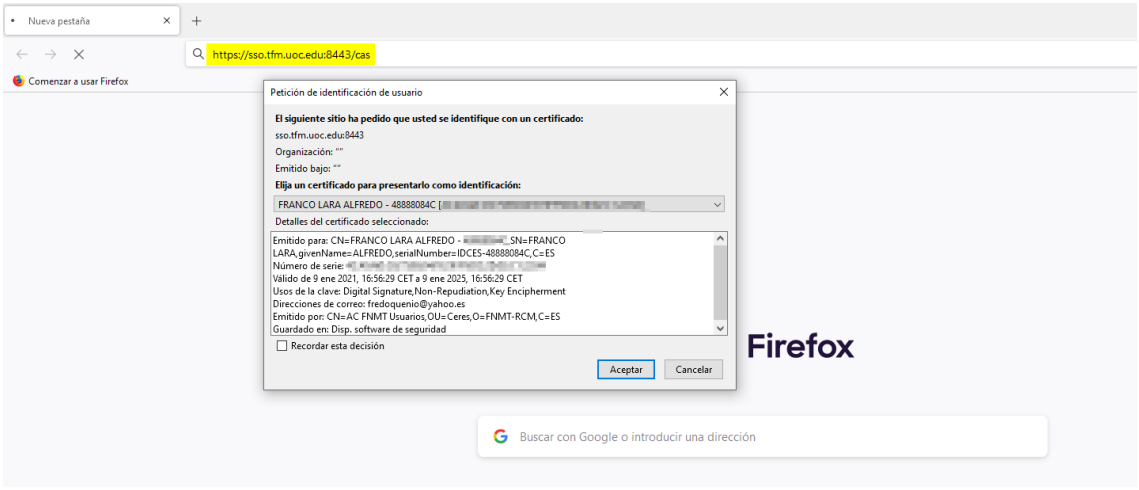


Figura 22: Configuración X.509. Solicitud certificado.

Si seleccionamos dicho certificado podemos comprobar cómo nos hemos logado correctamente.

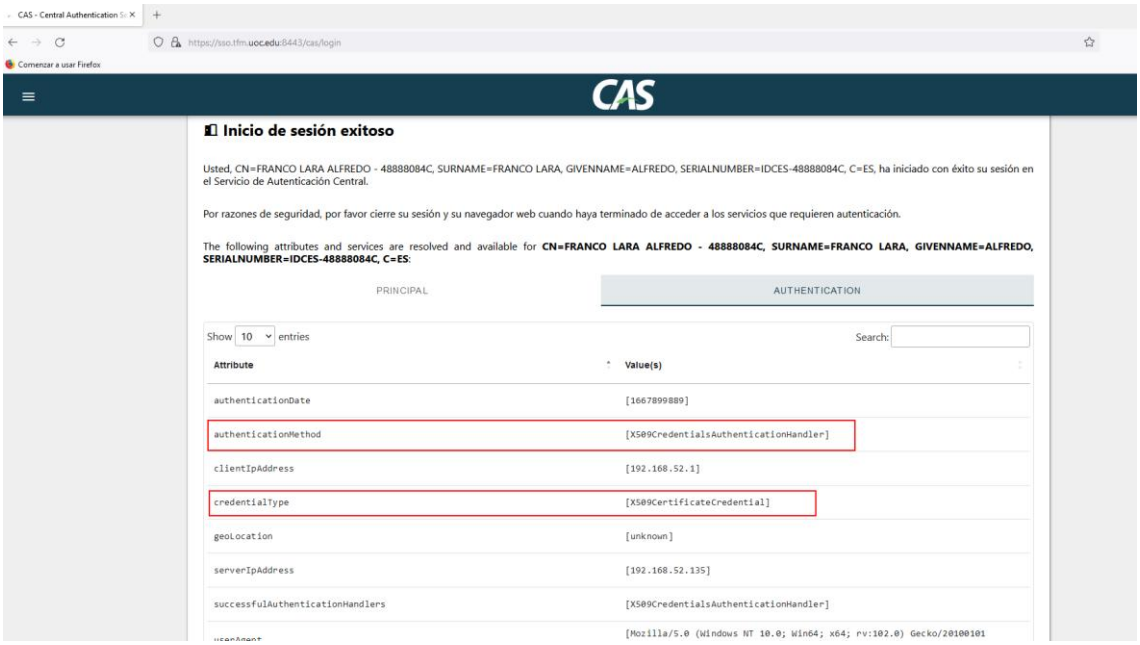


Figura 23: Configuración X.509. Datos login.

En este caso se ha logrado poder autenticarnos mediante certificado digital, a través de un certificado emitido por la FNMT o DNIe, pero debemos ir un paso más allá, ya que todo el que tenga un certificado de alguna de las CA no debe poder autenticarse, sino que sólo deben poder acceder los que están dados de alta en nuestro LDAP, para ello se debe verificar que el usuario cuyo certificado se ha seleccionado se encuentra en nuestro LDAP. CAS debe recuperar los datos del certificado y a partir de ellos buscar el usuario en LDAP para poder autenticarse.

CAS nos provee una integración de X509 con LDAP, para ello debemos configurar las siguientes propiedades:

- cas.authn.x509.ldap.ldap-url: url del servidor ldap
- cas.authn.x509.ldap.base-dn: base dn del ldap

- cas.authn.x509.ldap.bind-dn: el enlace dn que se usará al conectarse a LDAP
- cas.authn.x509.ldap.search-filter: filtro de usuario a usar para buscar.

A continuación se muestra el fichero de configuración. De esta forma le indicamos a CAS que una vez seleccionado un certificado debe buscar a éste en LDAP.

```
cas:
  authn:
      ldap[0]:
          type: AUTHENTICATED
          ldap-url: ldap://192.168.52.137:389
          base-dn: ou=empleados,dc=dipucadiz,dc=es
          search-filter: uid={user}
          bind-dn: ou=empleados,dc=dipucadiz,dc=es
      x509:
        ldap:   #configuramos la autenticación del certificado mediante LDAP,de esta forma se extraerá los datos del certificado
          ldap-url: ldap://192.168.52.137:389
          base-dn: ou=empleados,dc=dipucadiz,dc=es
          bind-dn: ou=empleados,dc=dipucadiz,dc=es
          search-filter: uid={user}
          bind-credential:
```

Figura 24: Configuración X.509. Integración con LDAP

3.5. Construcción de aplicaciones piloto

En este apartado vamos a crear dos aplicaciones piloto que nos servirán de guía sobre la configuración necesaria que hay que establecer sobre las diferentes aplicaciones que forman el catálogo de la Corporación. Para que las aplicaciones pilotos sean lo más parecidas a las aplicaciones de La Diputación, vamos a partir para su creación de un arquetipo maven[11] que desarrollé hace unos años para las nuevas aplicaciones, el cual nos proporciona un proyecto base que es similar a la mayoría de aplicaciones existentes.

Para su creación lanzamos el siguiente comando para cada una de las aplicaciones que deseamos generar.

mvn archetype:generate -DarchetypeGroupId=es.dipucadiz.epicsa.arquetipo -DarchetypeArtifactId=arquetipo_spring-mvc -DarchetypeVersion=1.0.1-SNAPSHOT -DgroupId=uoc.tfm.sso -DartifactId=aplicacion1

```
C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo>mvn archetype:generate -DarchetypeGroupId=es.dipucadiz.epicsa.arquetipo -DarchetypeArtifactId=arquetipo_spring-mvc -DarchetypeVersion=1.0.1-SNAPSHOT -DgroupId=uoc.tfm.
sso -DartifactId=aplicacion1
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.2.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.2.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.2.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[WARNING] Archetype not found in any catalog. Falling back to central repository.
[WARNING] Add a repository with id 'archetype' in your settings.xml if archetype's repository is elsewhere.
[INFO] Using property: groupId = uoc.tfm.sso
[INFO] Using property: version = 1.0.0-SNAPSHOT
[INFO] Using property: package = es.dipucadiz.epicsa
[INFO] Using property: artifactId = aplicacion1
Confirm properties configuration:
groupId: uoc.tfm.sso
version: 1.0.0-SNAPSHOT
package: es.dipucadiz.epicsa
artifactId: aplicacion1
 Y: : N
[INFO] Using property: groupId = uoc.tfm.sso
Define value for property 'version' 1.0.0-SNAPSHOT: :
Define value for property 'package' es.dipucadiz.epicsa: : uoc.tfm.sso
[INFO] Using property: artifactId = aplicacion1
Confirm properties configuration:
groupId: uoc.tfm.sso
version: 1.0.0-SNAPSHOT
package: uoc.tfm.sso
artifactId: aplicacion1
 Y: :
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: arquetipo_spring-mvc:1.0.1-SNAPSHOT
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: uoc.tfm.sso
[INFO] Parameter: artifactId, Value: aplicacion1
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: package, Value: uoc.tfm.sso
[INFO] Parameter: packageInPathFormat, Value: uoc/tfm/sso
[INFO] Parameter: package, Value: uoc.tfm.sso
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: groupId, Value: uoc.tfm.sso
[INFO] Parameter: artifactId, Value: aplicacion1
[INFO] Parent element not overwritten in C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-client\pom.xml
[WARNING] Don't override file C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-client\pom.xml
[INFO] Parent element not overwritten in C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-core\pom.xml
[WARNING] Don't override file C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-core\pom.xml
[INFO] Parent element not overwritten in C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-web\pom.xml
[WARNING] Don't override file C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-web\pom.xml
[INFO] Parent element not overwritten in C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-ws\pom.xml
[WARNING] Don't override file C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1\aplicacion1-ws\pom.xml
[INFO] Project created from Archetype in dir: C:\Users\afrancol\PROYECTOS\Recaudacion\workspace_v3\prueba_arquetipo\aplicacion1
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 20.194 s
```

11 https://maven.apache.org/guides/introduction/introduction-to-archetypes.html

Figura 25: Aplicación Piloto. Creación mediante arquetipo.

Una vez creadas ambas aplicaciones, (Aplicacion1 y Aplicacion2) haciendo uso de maven, se procede a importar el proyecto a nuestro IDE[12], en este caso Eclipse. Se puede observar cómo la aplicación es un proyecto que consta de 4 módulos

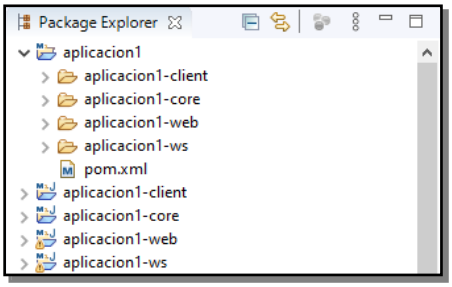


Figura 26: Aplicación Piloto. Estructura aplicación

Los módulos son:

- Módulo web: Contiene la aplicación web.
- Módulo ws: Contiene una API REST con los servicios que la aplicación expone a terceros.
- Módulo core: Este módulo es de tipo jar y contiene toda la lógica de negocio de la aplicación, el cual es compartido por el módulo ws y web.
- Módulo client: Este módulo provee una implementación del cliente REST para las aplicaciones de terceros.

Una vez importado el proyecto, realizamos unos cambios de estilos y cambiamos el logo por el de la UOC, desplegamos el módulo web sobre un contenedor Apache Tomcat para verificar su funcionamiento. Comprobamos que la aplicación arranca correctamente. A continuación se muestra la pantalla de inicio de la aplicación 2.

Figura 27: Aplicación Piloto. Pantalla inicio

Si pulsamos sobre el icono de login, la aplicación nos lleva al formulario de login que trae incorporado, donde las credenciales introducidas las valida contra una BBDD hslqdb que trae incorporada con dos usuarios por defecto,

12 https://www.eclipse.org/ide/

uno con rol USER y otro con rol ADMIN, tal como se puede apreciar en la siguiente captura

```
29 @Service
30 @Scope(proxyMode = ScopedProxyMode.TARGET_CLASS)
31 public class UsuarioService extends GenericServiceImpl<Usuario, Long> implements UserDetailsService {
32
33     private static final long serialVersionUID = 4373361015683602197L;
34
35     @Autowired
36     private UsuarioRepository usuarioRepository;
37
38     @Autowired
39     private PasswordEncoder passwordEncoder;
40
41     @PostConstruct
42     protected void initialize() {
43         //Inicializamos la tabla Usuarios con dos usuarios por defecto con dos roles diferentes.
44         save(new Usuario("user@dipucadiz.es", "user", "ROLE_USER"));
45         save(new Usuario("admin@dipucadiz.es", "admin", "ROLE_ADMIN"));
46     }
47
```

Figura 28: Aplicación Piloto. Carga de usuarios por defecto.

Se procede a introducir las claves de cada uno de los usuarios en la pantalla de login

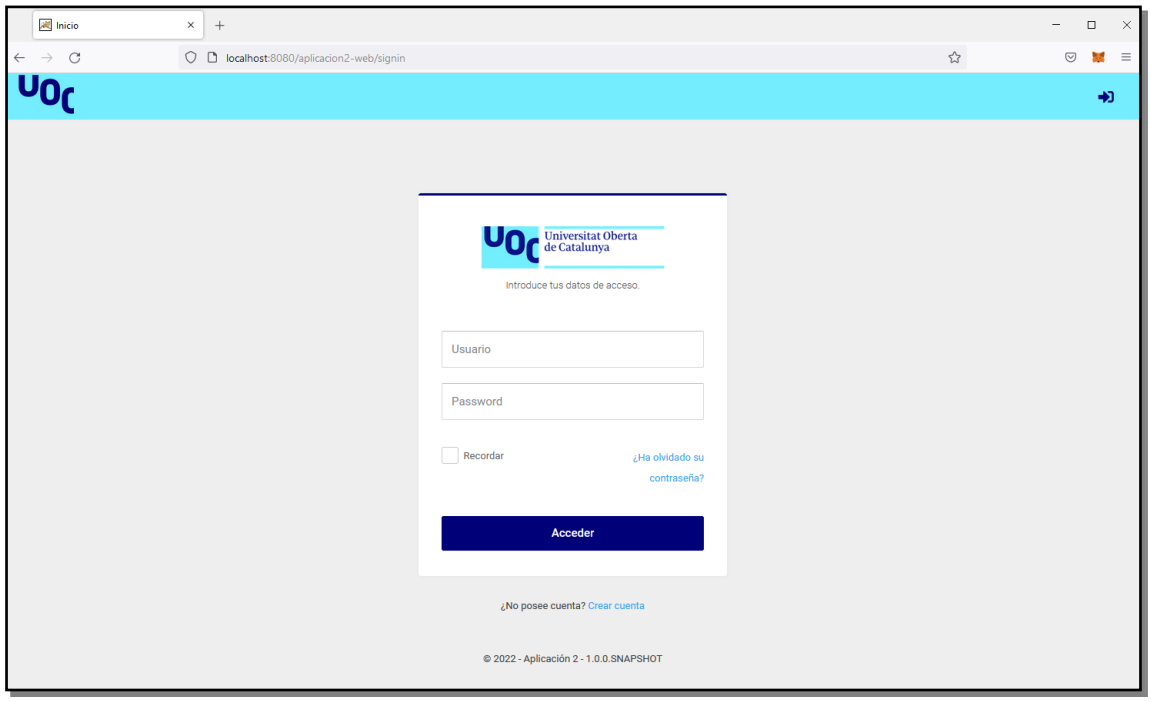


Figura 29: Aplicación Piloto. Pantalla de Login.

Si nos logamos con el usuario user@dipucadiz.es observamos cómo tras introducir las credenciales la aplicación nos muestra los datos del usuario logado, tanto en la esquina superior derecha como la pantalla principal.

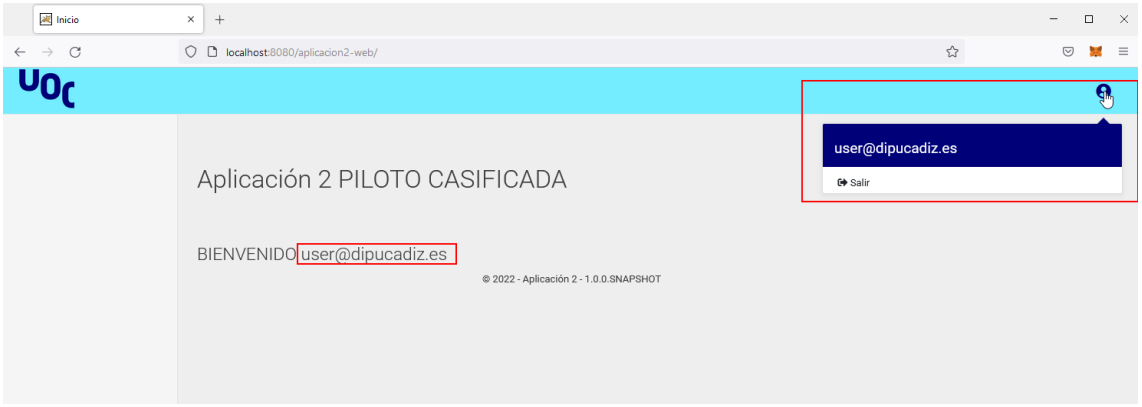


Figura 30: Aplicación Piloto. Pantalla inicio tras login.

Si ahora nos logamos con el usuario admin@dipucadiz.es verificamos que al tener rol de administrador también nos habilita una nueva opción para acceder al área de administración

Figura 31: Aplicación Piloto. Login usuario rol administrador.

Tal como se ha comentado anteriormente la idea es cambiar esta pantalla de login para que sea sustituida por la autenticación que nos proporciona CAS. Como todas nuestras aplicaciones hacen uso de spring-security nos vamos a centrar en éste para su integración con CAS[33]. Los pasos a seguir se pueden resumir en lo siguiente:

1. Se debe definir un *CasAuthenticationEntryPoint* que se encargará de redirigir hacia el servidor CAS los accesos hacia URLs que requiera autenticación. En ésta URL se le indica el parámetro *service* donde se establece la URL de retorno una vez el usuario se haya autenticado en el servidor CAS.
2. Se define el filtro *CasAuthenticacionFilter* que siempre estarán escuchando las peticiones hacia */login/cas* (aunque se puede configurar ésta). Ese filtro construye un *UsernamePasswordAuthenticationToken* que representa el ticket de servicio.
3. Se define un proveedor de autenticación, en este caso *CasAuthenticationProvider* que se encargará de validar que el ticket de servicio recibido desde el servidor CAS es correcto, obteniendo el nombre de usuario autenticado.
4. Este nombre de usuario le llegará a *AuthenticationUserDetailsService* que se encargará de verificar si el usuario autenticado posee los permisos necesarios en nuestra aplicación, es aquí donde la aplicación debe buscar en su BBDD el usuario autenticado en CAS para decidir si autoriza su acceso.
5. Si todo ha ido bien, vuelve el control al filtro *CasAuthenticationFilter* que se encarga de colocar el token de autenticación en el contexto de seguridad de Spring, redirigiendo al usuario a la página de acceso definida o a la de error.

Para simplificar la configuración de las aplicaciones se decide crear un pequeño cliente parametrizable que nos facilite esta integración con un mínimo esfuerzo, de esta forma, para futuras actualizaciones basta con actualizar este cliente y se replicará a todas las aplicaciones de forma transparente.

Una vez generado el cliente cuyo detalle se puede observar en el anexo Construcción librería CAS Client Spring Security se procede a configurar la aplicación con el cliente generado, los pasos para el correcto funcionamiento son:

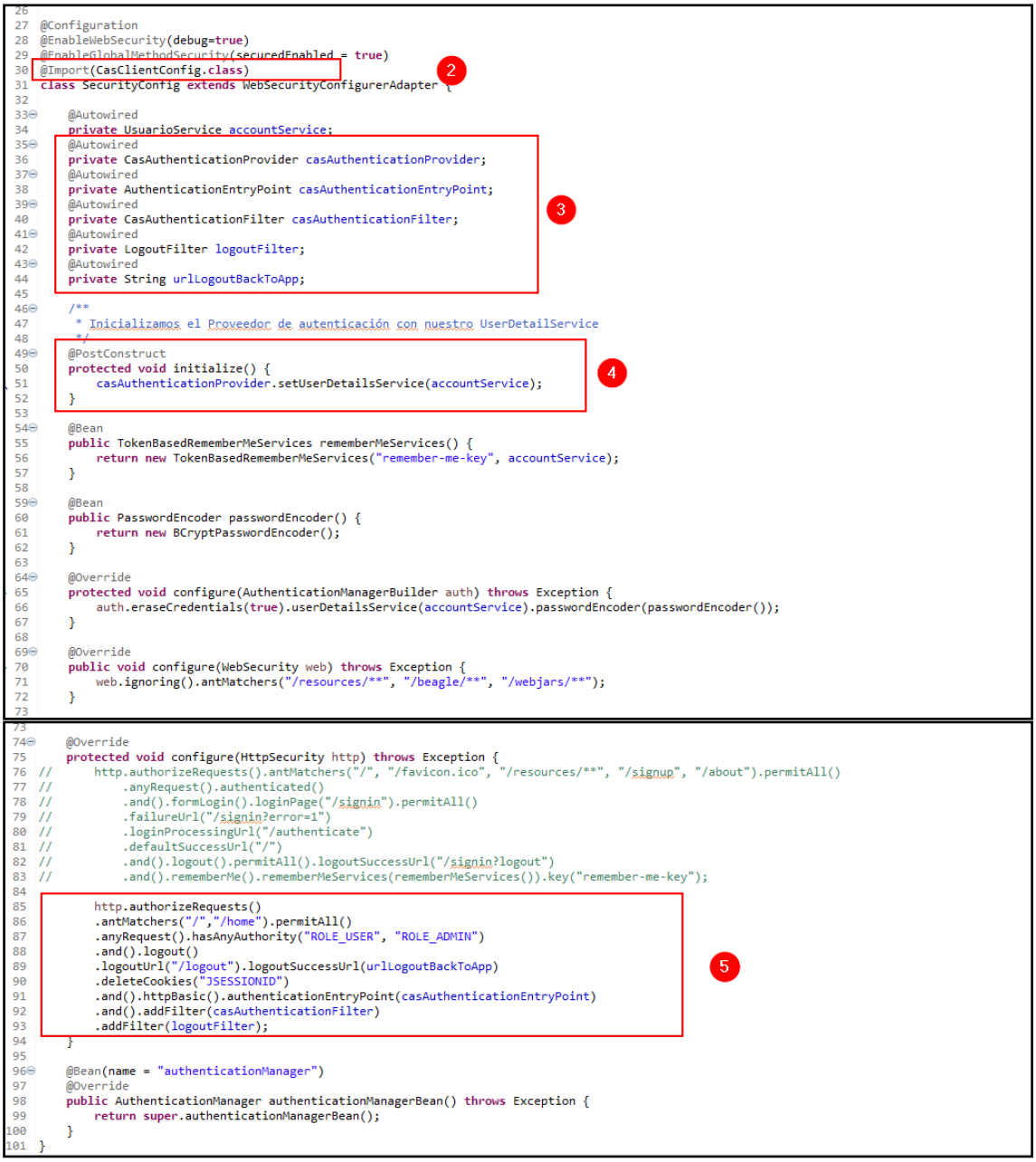
1. Incluir en la aplicación cliente que deseamos casificar una dependencia al nuevo cliente. Para ello basta con incluir en el fichero *pom.xml* de la aplicación lo siguiente

```
<dependency>
     <groupId>uoc.tfm.sso</groupId>
     <artifactId>cas-client-spring-security</artifactId>
     <version>0.0.1-SNAPSHOT</version>
</dependency>
```

2. En nuestro fichero de configuración debemos importar la clase de configuración *CasClientConfig*
3. Inyectamos los beans *casAuthenticacionProvider, casAuthenticationEntryPoint, casAuthenticationFilter, logoutFilter* y *urlLogoutBackToApp*
4. Se establece nuestra implementación de *UserDetailsService* al proveedor de autenticación.
5. Cambiamos la configuración de httpSecurity para que contemple los filtros y proveedor de autenticación definido.
6. En el fichero properties de la aplicación definimos las propiedades necesarias que necesita el cliente.

Se muestra a continuación la configuración establecida en la aplicación 2.

Figura 32: Aplicación Piloto. Inclusión dependencia en pom.xml.



```
@Configuration
@EnableWebSecurity(debug=true)
@EnableGlobalMethodSecurity(securedEnabled = true)
@Import(CasClientConfig.class)
class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UsuarioService accountService;
    @Autowired
    private CasAuthenticationProvider casAuthenticationProvider;
    @Autowired
    private AuthenticationEntryPoint casAuthenticationEntryPoint;
    @Autowired
    private CasAuthenticationFilter casAuthenticationFilter;
    @Autowired
    private LogoutFilter logoutFilter;
    @Autowired
    private String urlLogoutBackToApp;

    /**
     * Inicializamos el Proveedor de autenticación con nuestro UserDetailService
     */
    @PostConstruct
    protected void initialize() {
        casAuthenticationProvider.setUserDetailsService(accountService);
    }

    @Bean
    public TokenBasedRememberMeServices rememberMeServices() {
        return new TokenBasedRememberMeServices("remember-me-key", accountService);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.eraseCredentials(true).userDetailsService(accountService).passwordEncoder(passwordEncoder());
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/resources/**", "/beagle/**", "/webjars/**");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
//      http.authorizeRequests().antMatchers("/", "/favicon.ico", "/resources/**", "/signup", "/about").permitAll()
//          .anyRequest().authenticated()
//          .and().formLogin().loginPage("/signin").permitAll()
//          .failureUrl("/signin?error=1")
//          .loginProcessingUrl("/authenticate")
//          .defaultSuccessUrl("/")
//          .and().logout().permitAll().logoutSuccessUrl("/signin?logout")
//          .and().rememberMe().rememberMeServices(rememberMeServices()).key("remember-me-key");

        http.authorizeRequests()
        .antMatchers("/","/home").permitAll()
        .anyRequest().hasAnyAuthority("ROLE_USER", "ROLE_ADMIN")
        .and().logout()
        .logoutUrl("/logout").logoutSuccessUrl(urlLogoutBackToApp)
        .deleteCookies("JSESSIONID")
        .and().httpBasic().authenticationEntryPoint(casAuthenticationEntryPoint)
        .and().addFilter(casAuthenticationFilter)
        .addFilter(logoutFilter);
    }

    @Bean(name = "authenticationManager")
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

Figura 33: Aplicación Piloto. Configuración cliente CAS.

```
app.name=Aplicación 2
app.version=1.0.0.SNAPSHOT

#configuración CAS
sso.appClientUrl=http://localhost:8080/aplicacion2-web
sso.appClientFailLogin=http://localhost:8080/aplicacion2-web/home?error=2

```

Figura 34: Aplicación Piloto. Propiedades de configuración.

Una vez realizados los cambios anteriores volvemos a arrancar la aplicación para verificar que todo funciona correctamente. Tras arrancar pulsamos sobre el botón de login y verificamos que en este caso la aplicación nos redirige hacia el servidor CAS, y cómo la url de login, posee el parámetro “service” con la url que se ha configurado en la propiedad *sso.appClientUrl*, tal como se puede comprobar en la siguiente captura.

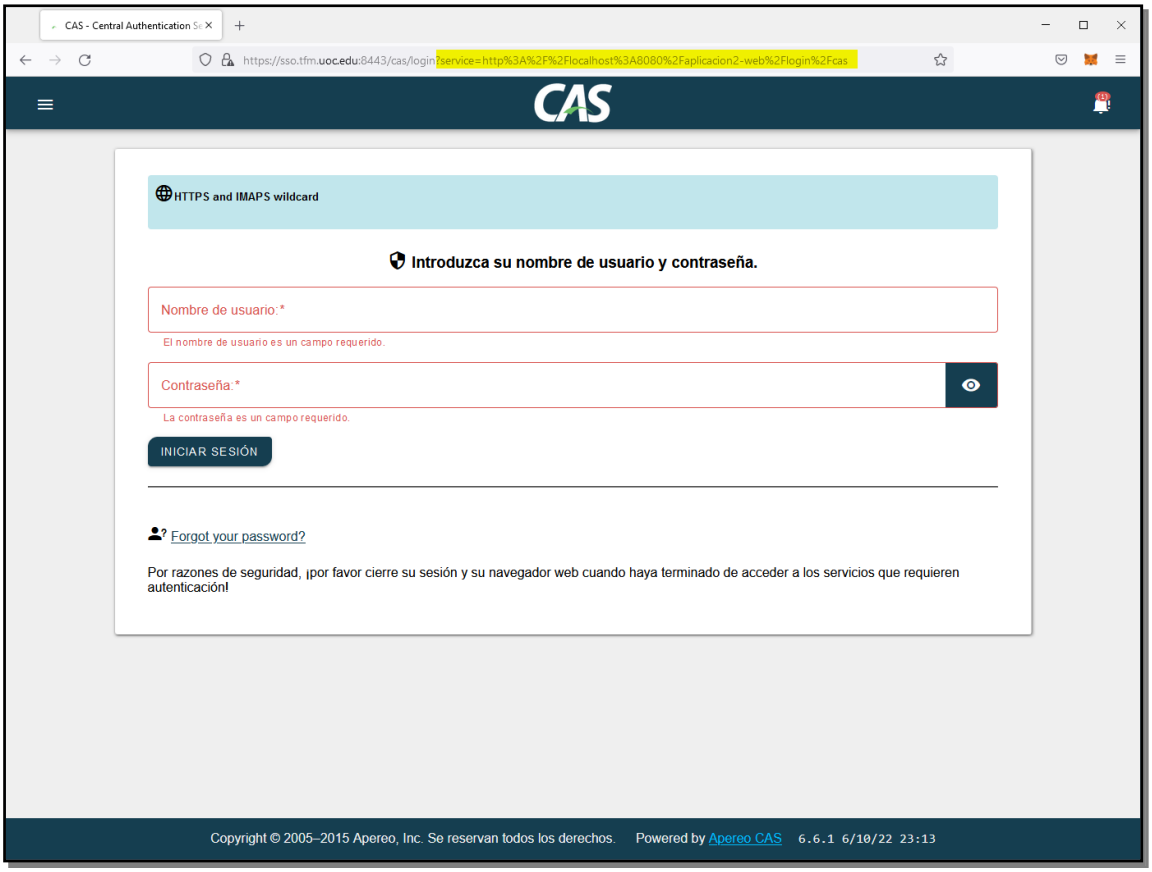


Figura 35: Aplicación Piloto. Pantalla login CAS.

Una vez logado en el servidor CAS éste redirige hacia la aplicación cliente, donde se ejecuta el filtro correspondiente para verificar el ticket ST recibido, tal como se puede apreciar en el log de la aplicación

Figura 36: Aplicación Piloto. Log de la validación del ticket ST

Se observa cómo a nuestra aplicación, una vez validado el ticket ST, nos llega el usuario “48888084C” y es éste usuario el que procede a buscar nuestra aplicación en la tabla usuario. En nuestro caso los datos de prueba que contiene la aplicación creada, para realizar el login busca en la tabla usuario un registro cuyo email es “48888084C” que es el dato que retorna el servidor CAS,

tal como se aprecia en el log que lanza el servidor, mostrando la consulta realizada para obtener al usuario y redirigiendo a la página de error.

Figura 37: Aplicación Piloto. Log del proceso de Autorización.

Donde nos indica que el usuario no está autorizado para acceder a la aplicación.

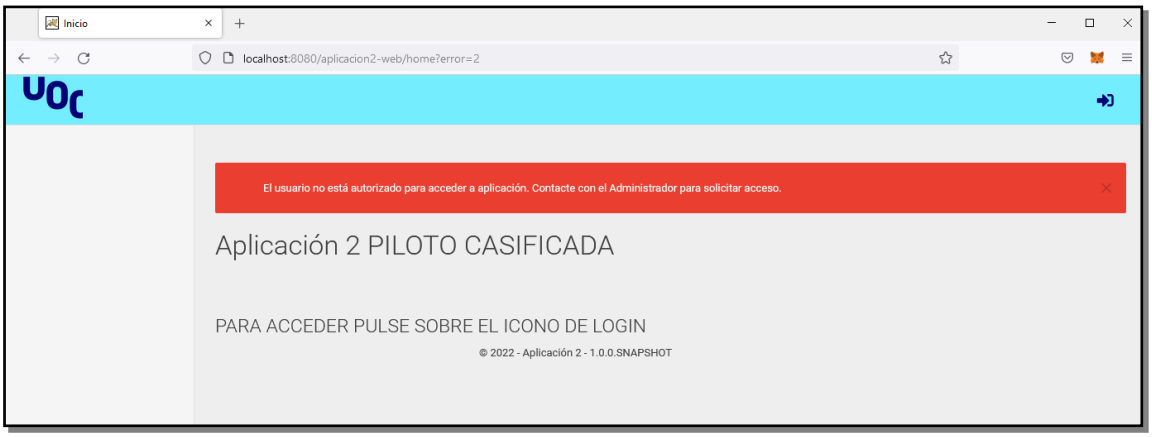


Figura 38: Aplicación Piloto. Pantalla acceso no autorizado.

Si en la inicialización de los datos de la tabla de usuario modificamos uno de los registros y sustituimos el email por el identificador “48888084C”, tal como se muestra a continuación

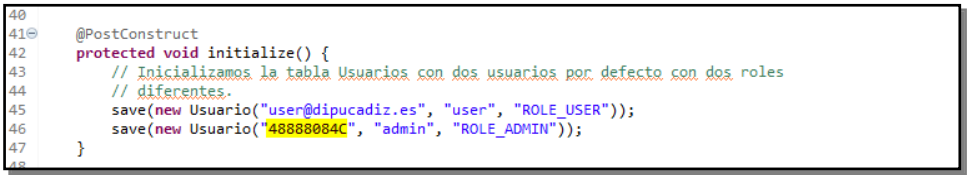


Figura 39: Aplicación Piloto. Modificación carga inicial usuarios.

Y volvemos a arrancar la aplicación, veremos como en este caso la aplicación cliente sí va a encontrar en BBDD el usuario logado en CAS y autorizará su acceso. Como se la ha indicado a la aplicación que el usuario “48888084C” posee el rol admin, vemos en la siguiente captura, cómo el usuario se ha logado correctamente y muestra las opciones correspondientes a los privilegios otorgados.

Figura 40: Aplicación Piloto. Pantalla inicio tras login mediante CAS

3.6. Alta Disponibilidad (Hight Availability) del Servidor CAS

Como se ha comentado en apartados anteriores, el servidor CAS, al ser un recurso crítico para todas las aplicaciones, debemos asegurar su disponibilidad, y para conseguirlo debemos tener varias instancias de CAS Server corriendo, lo que nos garantiza que si uno de los servidores no se encuentra disponible el resto seguirá procesando peticiones y el servicio no se verá interrumpido. Para conseguir esto, CAS recomienda la siguiente arquitectura

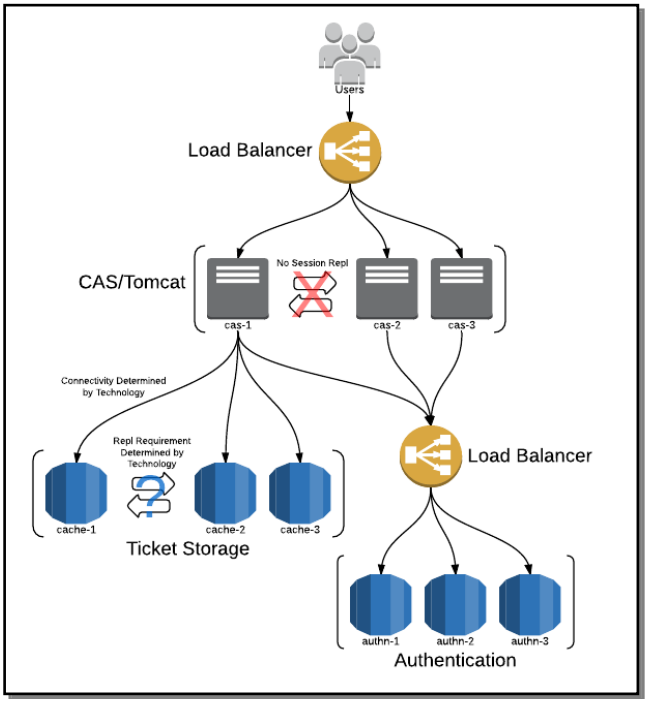


Figura 41: Alta Disponibilidad. Arquitectura recomendada de CAS

Luego la implementación de alta disponibilidad la podemos resumir en lo siguiente:

3.6.1. Compartir el registro de tickets entre todos los nodos CAS

Debemos disponer de un mecanismo de ticket compartido, donde todos los servidores CAS compartan la misma información de autenticación. Ya que al tener varios nodos y un balanceador delante, no sabemos si el nodo que va a procesar nuestra petición va a ser el mismo o no. Para ello vamos a usar JPA

Ticket Registry[34], donde se almacenará los tickets en una base de datos relacional MariaDB. El proceso de instalación de la BBDD se puede observar en el anexo Instalación de MariaDB. Una vez tenemos nuestra BBDD debemos configurar CAS para que almacene la información de los tickets en ella.

Lo primero es añadir el módulo *cas-server-support-jpa-ticket-registry* en el fichero *build.gradle*, tal como se muestra a continuación

```
dependencies {
    /**
    * Do NOT modify the lines below or else you will risk breaking dependency management.
    */
    implementation enforcedPlatform("org.apereo.cas:cas-server-support-bom:${project.'cas.version'}")
    implementation platform(org.springframework.boot.gradle.plugin.SpringBootPlugin.BOM_COORDINATES)

    /**
     * CAS dependencies and modules may be listed here.
     *
     * There is no need to specify the version number for each dependency
     * since versions are all resolved and controlled by the dependency management
     * plugin via the CAS bom.
     **/

    implementation "org.apereo.cas:cas-server-core-api-configuration-model"
    implementation "org.apereo.cas:cas-server-webapp-init"
    implementation "org.apereo.cas:cas-server-support-ldap"
    implementation "org.apereo.cas:cas-server-support-x509-webflow"
    implementation "org.apereo.cas:cas-server-support-jpa-ticket-registry"

    if (project.hasProperty("casModules")) {
        def dependencies = project.getProperty("casModules").split(",")
        dependencies.each {
            def projectsToAdd = rootProject.subprojects.findAll {project ->
                project.name == "cas-server-core-${it}" || project.name == "cas-server-support-${it}"
            }
            projectsToAdd.each {implementation it}
        }
    }
```

Figura 42: Alta Disponibilidad. Importación módulo JPA.

Y posteriormente configurar las propiedades necesarias para conectar con la BBDD en el fichero *application.yml*, indicando el usuario que hemos creado para ello.

```
#CONFIGURACIÓN DEL REGISTRO DE TICKETS EN BBDD
ticket:
   registry:
      jpa:
         user: appcasserver
         password: [redacted]
         driver-class: org.mariadb.jdbc.Driver
         url: jdbc:mariadb://192.168.52.137:3306/cas_sso
         dialect: org.hibernate.dialect.MariaDBDialect
```

Figura 43: Alta Disponibilidad. Configuración Base de Datos

De esta forma ya tenemos nuestro servidor CAS configurado para poder compartir la información de los tickets mediante BBDD.

Se verifica que una vez realizado los cambios el servidor almacena los datos de los tickets en BBDD, tal como se puede apreciar en la siguiente consulta realizada sobre la tabla. Se ha realizado un login con certificado y otro mediante user/password, donde se observa cómo el campo *"principal_id"* contiene los datos del certificado como los introducidos en el formulario de login.

```
select * from Cas_Tickets ct;
```

id	body	creation_Time	parent_Id	principal_Id	type
TGT-2-	{"@class":"org.apereo.cas.ticket.TicketGrantingTicketImpl","@id":1,"id":"TGT-2-	2022-11-30 10:52:13	[NULL]	CN=LARA JIMENI	org.apereo.cas.ticket.TicketGrantingTicketImpl
TGT-3-XzOVL-QoV6PyOw3QfOCPZclkbwVqqJXhJJe3EzfOF1QMtySuZhO0eP6Ybu3Y1rjBj	{"@class":"org.apereo.cas.ticket.TicketGrantingTicketImpl","@id":1,"id":"TGT-3-	2022-11-30 11:03:08	[NULL]	48888084C	org.apereo.cas.ticket.TicketGrantingTicketImpl

3.6.2. Crear varios nodos de CAS Server

En este caso, como nuestros servidores CAS vienen con el servidor Tomcat embebido basta con generar el war correspondiente desde *cas-overlay-template* para posteriormente arrancarlo con el comando *java –jar cas.war*. En esta prueba de concepto, al no disponer de más espacio en disco para poder crear otra máquina virtual, se decide arrancar los dos nodos en la misma máquina en un puerto diferente. Tendremos el servidor CAS1 en el puerto 8443 y el servidor CAS2 en el puerto 9443. Para ello basta con cambiar el puerto en el fichero *application.yml* y generar un war para cada uno de los puertos.

```
48 server:
49     name: sso.tfm.uoc.edu
50     port: 9443
```

3.6.3. Configurar el Balanceador de Carga

Se procede a continuación a la configuración del balanceador de carga que será el encargado de redirigir las peticiones de los usuarios hacia uno de los servidores CAS según su disponibilidad. Para ello comenzamos instalando el servidor Apache HTTP, cuyo detalle se puede observar en el anexo Instalación de Apache HTTP.

Una vez instalado realizamos la configuración necesaria para que el servidor sea capaz de balancear la carga. Todo el tráfico del balanceador debe ir en https, luego configuramos Apache para que todo el tráfico recibido por el puerto 80 mediante http lo redirija hacia el puerto 443 por https, esto lo logramos editando el fichero *000-default.conf* y añadiendo la opción *Redirect*

```
alfredo@ssoproxy: ~
  GNU nano 6.2                                  /etc/apache2/sites-enabled/000-default.conf
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName ssoproxy

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf

        Redirect / https://proxy.tfm.uoc.edu
        #ProxyPass / https://ssol.tfm.uoc.edu:8443/cas
        #ProxyPassReverse / https://ssol.tfm.uoc.edu:8443/cas
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 44: Alta Disponibilidad. Fichero configuración 000-default.conf

Posteriormente debemos configurar apache en el puerto 443 para que vaya por https, para ello debemos generar los certificados correspondientes para la comunicación https. Para la generación vamos a hacer uso de la herramienta OpenSSL(36), donde crearemos nuestros certificados para poder realizar las pruebas necesarias (estos certificados en los entornos de Producción deben ser emitidos por una Autoridad Certificadora). Para ello, lo primero que hacemos es crearnos nuestra propia CA (Certification Authority) para poder crearnos nuestros certificados, luego como primer paso lanzamos el siguiente comando

sudo openssl –out rootCA.key

Para generar la clave privada de CA, posteriormente a partir de la clave privada generada procedemos a generar la clave pública y la almacenamos en el fichero *rootCA.pem* con el siguiente comando

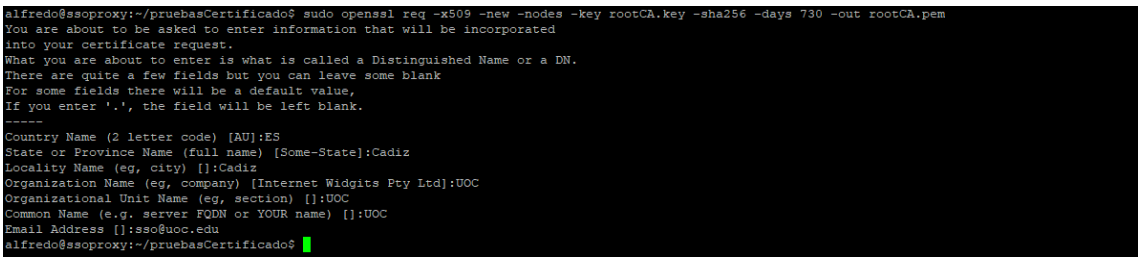


Figura 45: Alta Disponibilidad. Creación clave pública a partir de clave privada.

Pasamos ahora a generar el certificado del servidor web, para ello generamos la clave privada igual que anteriormente y posteriormente un fichero con extensión *.csr,* que es el fichero Certificate Signing Request (Solicitud de Firma de Certificado). A continuación se muestra los comandos lanzados para su generación

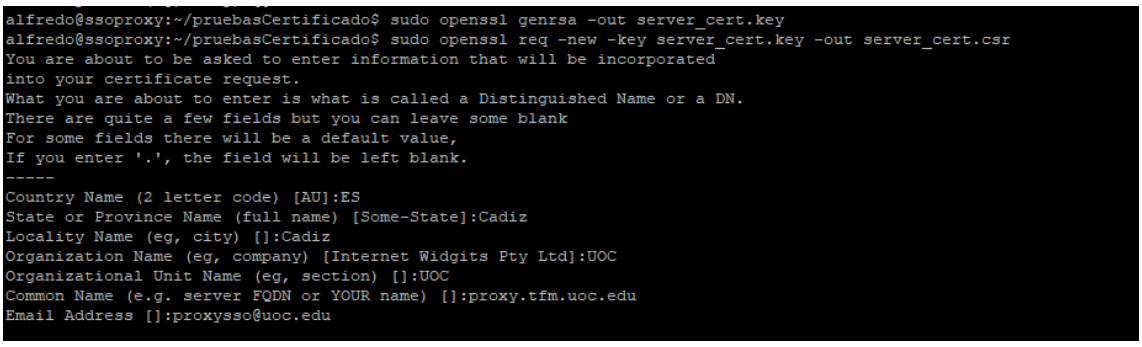


Figura 46: Alta Disponibilidad. Certificado Solicitud de Firma.

A partir de los certificados generados anteriormente generamos el certificado del servidor con el siguiente comando

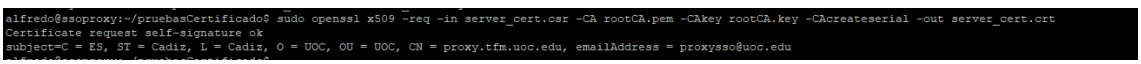


Figura 47: Alta Disponibilidad. Certificado de servidor

Una vez generado los certificados los copiamos al directorio */etc/apache2/certificates* los ficheros *server_cert.crt* y *servert_cert.key* (clave privada y solicitud de firma del servidor y configuramos el fichero *default-ssl.conf* para que se nutra de los certificados generados

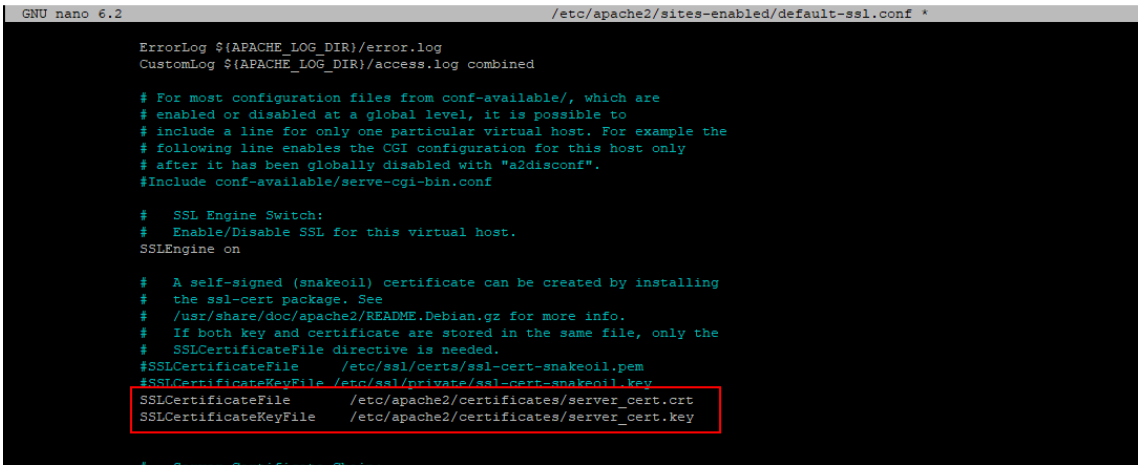


Figura 48: Alta Disponibilidad. Configuración ssl

Para habilitar tanto el módulo ssl como el de proxy debemos añadir dichos módulos a nuestra instalación, lo hacemos lanzando el comando

sudo a2enmod ssl, proxy_http

Pasamos ahora a la configuración del balanceador de carga, para ello volvemos a editar el fichero *default-ssl.conf* y añadimos las siguientes configuraciones(38), donde definimos un balanceador, con nombre *balanceadorsso,* que contiene dos servidores que están en la misma máquina en diferentes puertos como se había comentado. Posteriormente indicamos que todas las peticiones cuya url sea */cas* será re-direccionada hacia uno de los servidores definidos en el balanceador.

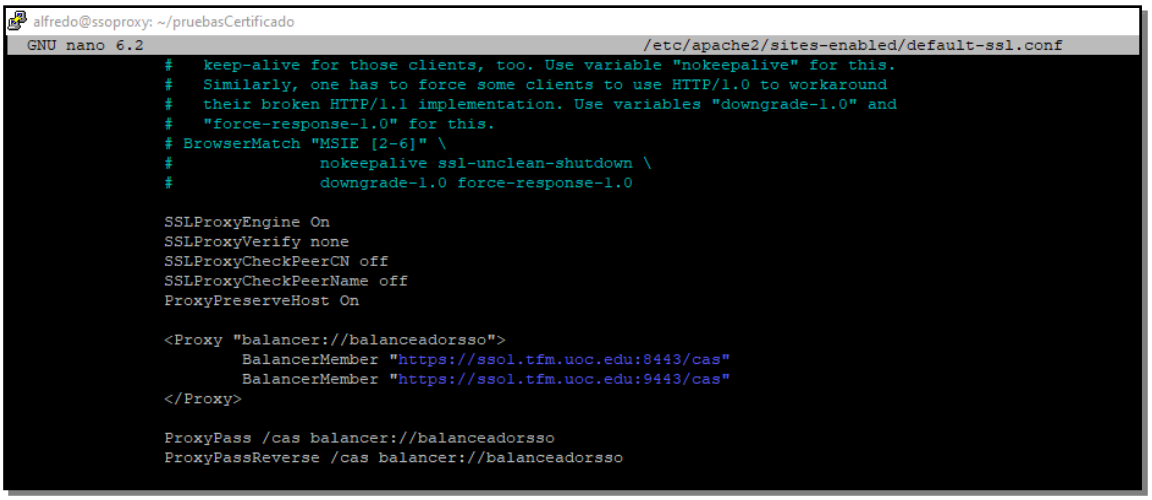


Figura 49: Alta Disponibilidad. Configuración módulo proxy.

Ahora debemos configurar la autenticación mediante certificado digital o DNIe, ya que será nuestro proxy el encargado de solicitar al usuario el certificado a través del navegador para posteriormente remitírselo al servidor CAS. Para ello le debemos indicar a nuestro Apache qué certificados debe mostrar cuando nos lo solicita. Como los certificados admitidos son los de la FNMT y DNIe, debemos copiar los certificados raíz de la FNMT y DNIe en */etc/ssl/certs/* y en el fichero de configuración indicamos dicha ruta a la propiedad *SSLCACertificatePath*, indicamos el tipo de autenticación, en la propiedad *SSLVerifyClient* y la establecemos como *optional*, ya que nuestro servidor CAS permite logarnos mediante certificado o introduciendo nuestras credenciales.

Para que el certificado llegue al servidor CAS, debemos definir la propiedad *SSLOptions* para que Apache incluya el certificado en la cabecera *SSL_CLIENT_CERT* y CAS pueda recuperarlo. Y por último, en nuestro fichero de configuración indicamos que si el módulo ssl está presente establecemos las cabeceras correspondientes[13]. A continuación se muestra las configuraciones establecidas en el fichero.

13 https://www.ferrol.gal:8443/docs/config/valve.html#SSL_Valve

```
                SSLProxyEngine On
                SSLProxyVerify none
                SSLProxyCheckPeerCN off
                SSLProxyCheckPeerName off
                ProxyPreserveHost On

                <Proxy "balancer://balanceadorsso">
                        BalancerMember "https://ssol.tfm.uoc.edu:8443/cas"
                        BalancerMember "https://ssol.tfm.uoc.edu:9443/cas"
                </Proxy>

                ProxyPass /cas balancer://balanceadorsso
                ProxyPassReverse /cas balancer://balanceadorsso

                ProxyRequests off

                <IfModule log_forensic_module>
                        ForensicLog ${APACHE_LOG_DIR}/forensic.log
                </IfModule>

                <IfModule ssl_module>
                        RequestHeader set SSL_CLIENT_CERT "%{SSL_CLIENT_CERT}s"
                        RequestHeader set SSL_CIPHER "%{SSL_CIPHER}s"
                        RequestHeader set SSL_SESSION_ID "%{SSL_SESSION_ID}s"
                        RequestHeader set SSL_CIPHER_USEKEYSIZE "%{SSL_CIPHER_USEKEYSIZE}s"
                </IfModule>

        </VirtualHost>
</IfModule>
```

Figura 50: Alta Disponibilidad. Fichero de configuración default-ssl.conf

Para aplicar todos los cambios realizados se reinicia el servidor con el comando ***sudo systemctl restart apache2***

Verificamos que si arrancamos los dos nodos CAS e introducimos la url del proxy en el navegador y nos logamos introduciendo usuario/password la petición llega a uno de los servidores. Si tras ese inicio de sesión detenemos el servidor que ha procesado nuestra solicitud y volvemos, ahora sólo tenemos levantado un nodo CAS (el que no procesó nuestra petición de login), y volvemos a la url del proxy, vemos cómo ahora nuestro balanceador redirige la petición hacia el otro nodo, el cual procesa la petición verificando que ya estamos logado mediante el ticket que el otro nodo había grabado en la BBDD que configuramos anteriormente para compartir la información, mostrando directamente la pantalla de inicio de sesión exitoso.

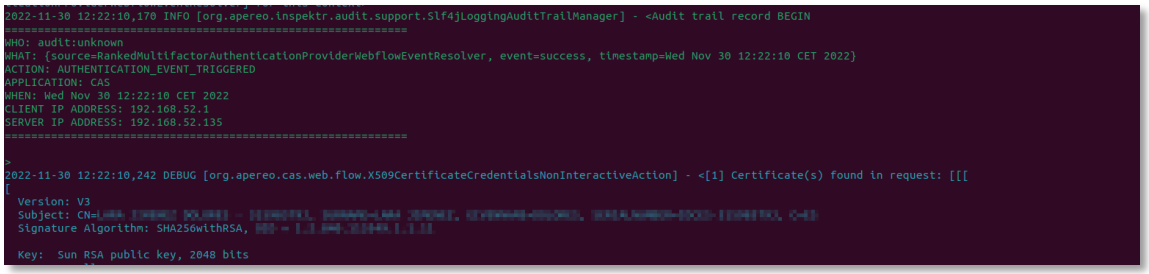
3.6.4. Configurar CAS para autenticar con certificado recibido desde proxy

Debemos configurar CAS para que pueda extraer un certificado X509 de las cabeceras de las peticiones http. Esta cabecera será generada por nuestro proxy con la configuración que hemos realizado anteriormente. En nuestro servidor CAS configuraremos la propiedad ***cas.authn.x509.extract-cert*** a true para indicar la extracción del certificado desde la cabecera. En este caso nuestro fichero de configuración contendrá las siguientes propiedades que se muestran en la siguiente figura.

```
cas:
  authn:
      ldap[0]:
          type: AUTHENTICATED
          ldap-url: ldap://192.168.52.137:389
          base-dn: ou=empleados,dc=dipucadiz,dc=es
          search-filter: uid={user}
          bind-dn: ou=empleados,dc=dipucadiz,dc=es
      x509:
         principal-type: SUBJECT_DN
         extract-cert: true #habilitamos la extracción del certificado de la cabecera, ya que nos llegará desde el proxy
         ssl-header-name: ssl_client_cert #nombre de la cabecera donde en la que nos llegará el certificado.
```

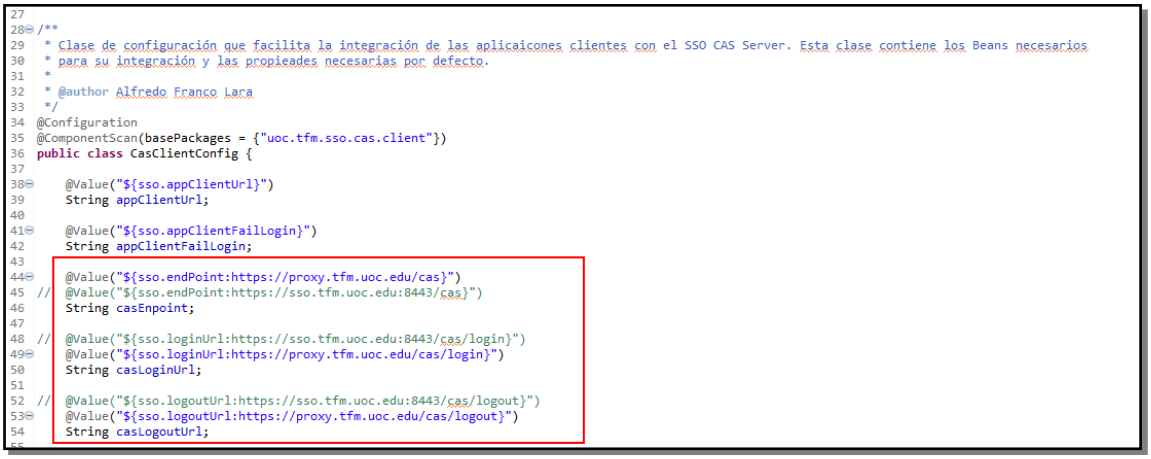
Figura 51: Alta Disponibilidad. CAS integración X509 con LDAP

Tras realizar la configuración se verifica que el nodo CAS extrae el certificado que recibe del servidor proxy, tal como se aprecia en la siguiente captura del log.



3.6.5. Modificar la configuración de las aplicaciones pilotos

Se debe cambiar la configuración de las aplicaciones pilotos para que éstas a la hora de realizar el login apunten al proxy definido. Aquí podemos optar por dos opciones, definimos en cada de una las aplicaciones en el fichero de configuración las propiedades *sso.endPoint*, *sso.loginUrl* y *sso.logoutUrl* para que apunte al proxy, o modificamos el cliente que habíamos generado para facilitar la integración de las aplicaciones, modificando el valor por defecto de estas propiedades para que apunte al nuevo proxy. En este caso optamos por modificar el cliente, sustituyendo los valores por defecto que habíamos indicado, quedando como se muestra a continuación



```
/**
 * Clase de configuración que facilita la integración de las aplicaiones clientes con el SSO CAS Server. Esta clase contiene los Beans necesarios
 * para su integración y las propieades necesarias por defecto.
 *
 * @author Alfredo Franco Lara
 */
@Configuration
@ComponentScan(basePackages = {"uoc.tfm.sso.cas.client"})
public class CasClientConfig {

    @Value("${sso.appClientUrl}")
    String appClientUrl;

    @Value("${sso.appClientFailLogin}")
    String appClientFailLogin;

    @Value("${sso.endPoint:https://proxy.tfm.uoc.edu/cas}")
//  @Value("${sso.endPoint:https://sso.tfm.uoc.edu:8443/cas}")
    String casEnpoint;

//  @Value("${sso.loginUrl:https://sso.tfm.uoc.edu:8443/cas/login}")
    @Value("${sso.loginUrl:https://proxy.tfm.uoc.edu/cas/login}")
    String casLoginUrl;

//  @Value("${sso.logoutUrl:https://sso.tfm.uoc.edu:8443/cas/logout}")
    @Value("${sso.logoutUrl:https://proxy.tfm.uoc.edu/cas/logout}")
    String casLogoutUrl;
```

Figura 52: Alta Disponibilidad. Modificar urls clientes hacia proxy

He aquí una de las ventajas de usar un cliente común, al realizar la modificación y subirla al servidor maven las aplicaciones podrán actualizarse sin esfuerzo alguno siendo estos cambios transparentes para ellas.

3.7. Arquitectura Pruebas Laboratorio

A continuación se muestra un pequeño esquema de la arquitectura de laboratorio que se ha montado para el desarrollo, implementación y pruebas del sistema SSO. Todo se ha desarrollado en un único PC, donde se han instalado 3 máquinas virtuales a través de VMWARE. En una de ellas se ha instalado dos servidores CAS, en los puertos 8443 y 9443. En otra máquina hemos instalado el servidor LDAP y una BBDD MariaDB para que los nodos CAS puedan compartir la información de los tickets y en la última máquina se ha instalado un servidor Apache HTTP que actúa de proxy inverso y balanceador de carga. Por falta de espacio y recursos en el PC, las aplicaciones piloto se

han desarrollado sobre éste, sin hacer uso de máquinas virtuales y se despliegan en un servidor tomcat instalado en el mismo PC, donde las BBDD que tienen configuradas ambas aplicaciones son HSQLDB[14].

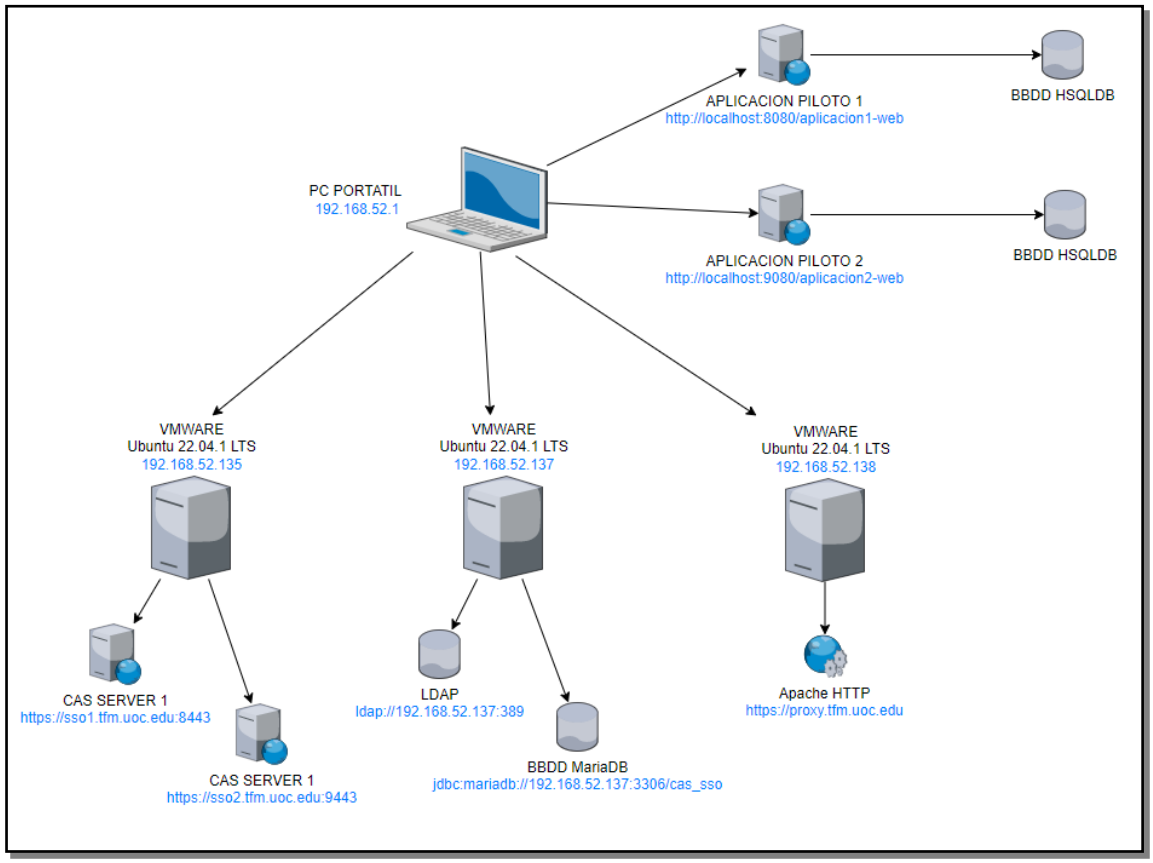
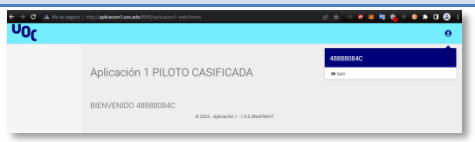


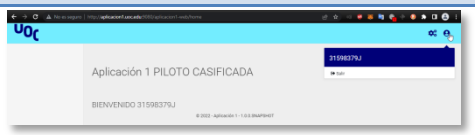
Figura 53: Arquitectura de Laboratorio.

14 http://hsqldb.org/

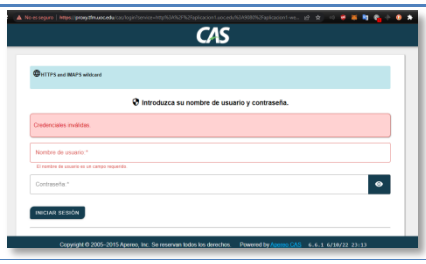
4. Pruebas

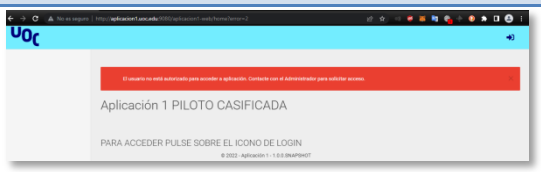
En este capítulo se llevarán a cabo las pruebas de integración que nos permite verificar que se cumplen con todos los requisitos que se han definido.

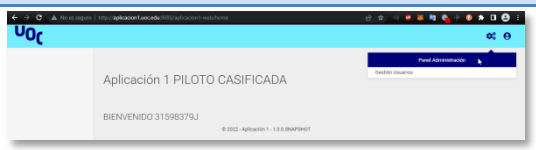
CP-01	**Acceso a la Aplicación 1 mediante usuario/contraseña**
Descripción	Accedemos a una aplicación con un usuario dado de alta en la Corporación y tiene permiso de acceso a la aplicación 1 introduciendo su usuario y contraseña.
Pasos	1. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login. 2. El navegador nos solicita un certificado para autenticarnos. 3. Cancelamos solicitud de certificado. 4. Nos muestra pantalla login de CAS, donde se introduce un usuario y contraseña correcta. 5. CAS nos redirige hacia la página principal de la aplicación 1, donde se muestra los datos del usuario logado.
Resultado	OK
Evidencias	

CP-02	**Acceso a la Aplicación 1 mediante Certificado Digital o DNIe**
Descripción	Accedemos a una aplicación con un usuario que está dado de alta en la Corporación y tiene permiso de acceso a la aplicación 1 mediante certificado digital
Pasos	1. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login. 2. El navegador nos solicita un certificado para autenticarnos. 3. Seleccionamos un certificado de un usuario que se encuentra dado de alta en LDAP. 4. CAS valida el certificado seleccionado y nos redirige hacia la página principal de la aplicación 1, donde se muestra los datos del usuario logado.
Resultado	OK
Evidencias	

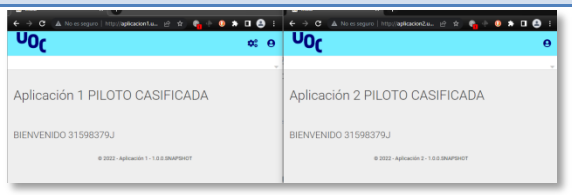
CP-03	**Error de autenticación**
Descripción	Intentamos acceder a una de las aplicaciones con un usuario que no pertenece a la Corporación.
Pasos	1. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login. 2. El navegador nos solicita un certificado para autenticarnos. 3. Cancelamos solicitud de certificado. 4. Nos muestra pantalla login de CAS, donde se introducen un usuario y contraseña que no se encuentra dado de alta en LDAP. 5. CAS nos muestra mensaje de error indicando que las credenciales son inválidas.
Resultado	OK

Evidencias	

CP-04	Error de autorización
Descripción	Nos autenticamos en CAS con un usuario dado de alta en LDAP pero éste no tiene acceso a una determinada aplicación.
Pasos	1. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login. 2. El navegador nos solicita un certificado para autenticarnos. 3. Cancelamos solicitud de certificado. 4. Nos muestra pantalla login de CAS, donde se introducen las credenciales de un usuario que se encuentra en LDAP pero no en la BBDD de usuarios de la aplicación 1. 5. CAS nos autentica correctamente y nos redirige hacia la página principal de la aplicación 1, donde se muestra el mensaje de error “Usuario no autorizado”.
Resultado	OK
Evidencias	

CP-05	Acceso a la aplicación 1 con Rol Administrador
Descripción	Se accede a la aplicación 1 con un usuario que posee el Rol de Administrador en ésta.
Pasos	1. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login. 2. El navegador nos solicita un certificado para autenticarnos. 3. Cancelamos solicitud de certificado. 4. Nos muestra pantalla login de CAS, donde se introducen las credenciales de un usuario que se encuentra en LDAP y en la BBDD de la aplicación 1 tiene el Rol Administrador. 5. CAS nos autentica correctamente y nos redirige hacia la página principal de la aplicación 1, donde se muestra el panel de administración para los usuarios con rol administrador.
Resultado	OK
Evidencias	

CP-06	SSO entre aplicaciones
Descripción	Partimos de un usuario que tiene acceso tanto a aplicación 1 como aplicación 2. Si el usuario accede a la aplicación 1 autenticándose mediante CAS y posteriormente accede a la aplicación 2, debe acceder sin que se le solicite las credenciales de nuevo.
Pasos	1. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login. 2. El navegador nos solicita un certificado para autenticarnos.

	3. Seleccionamos un certificado de usuario que está autorizado para acceder tanto a la aplicación 1 como a la aplicación 2. 4. CAS nos redirige hacia la página principal de la aplicación 1, donde se muestra los datos del usuario logado. 5. Accedemos a la url http://aplicacion2.uoc.edu:8080/aplicacion2-web/ y pulsamos sobre el botón de login. 6. Accedemos directamente con el usuario con el cual nos habíamos logado en la aplicación 1, sin necesidad de volver a seleccionar de nuevo el certificado ni de introducir nuevas credenciales.
Resultado	OK
Evidencias	

CP-07	**Balanceo de carga**
Descripción	Se debe verificar que el proxy inverso que actúa como balanceador está funcionando correctamente.
Pasos	1. Ambos nodos CAS se encuentran ejecutándose. 2. Accedemos a la url http://aplicacion1.uoc.edu:9080/aplicacion1-web/ y pulsamos sobre el botón de login y nos logamos correctamente. 3. Paramos uno de los nodos y accedemos a la aplicación 2 mediante la url http://aplicacion2.uoc.edu:8080/aplicacion2-web/ 4. Pulsamos sobre el botón de login y nos logamos correctamente. 5. Se verifica que las peticiones son redirigidas al único nodo activo. 6. El sistema sigue funcionando correctamente aunque paremos uno de los nodos.
Resultado	OK
Evidencias	

5. Conclusiones y trabajos futuros

En el siguiente capítulo se realiza una pequeña descripción de las conclusiones que se han obtenido del presente TFM y los posibles trabajos futuros que se deberían abordar.

5.1. Conclusiones del TFM

En este trabajo se ha comprobado las ventajas que ofrece un sistema Single Sing-On (SSO) para una empresa en cuanto a robustez y seguridad, ya que el producto seleccionado (CAS), como la mayoría de los existentes, nos ofrece infinidad de alternativas a la hora de autenticarnos, facilitando la labor de implementación de estas integraciones en un solo sistema, y además, suelen tener una comunidad detrás que vela porque el producto sea lo más seguro posible.

Además para los empleados también supone una ventaja el evitar memorizar las distintas credenciales de cada una de las aplicaciones a las que accede, ya que al autenticarse una sola vez puede acceder a todo el catálogo de aplicaciones sin tener que volver a repetir el proceso de login.

5.2. Consecución de objetivos

De los objetivos marcados al inicio del TFM se han podido conseguir prácticamente todos. Se ha implantado y configurado el sistema SSO con CAS Server en alta disponibilidad. Se han creado dos aplicaciones piloto que simulan a las existentes, centrándonos en la búsqueda de una integración lo más fácil y cómoda posible, donde se ha creado un pequeño cliente parametrizable que hace que la configuración de las aplicaciones sea casi automática, con sólo incluir una dependencia a este cliente y configurar varias propiedades en un properties tendríamos la aplicación integrada con nuestro sistema SSO.

Se ha configurado CAS para poder autenticarnos tanto con Certificado Digital, DNIe o introduciendo las credenciales y se ha protegido las comunicaciones mediante protocolo seguro SSL/TLS.

5.3. Seguimiento de la planificación y metodología

En cuanto a la planificación se refiere, sí es cierto que las primeras entregas han estado más apretadas, por el hecho de tener que realizar bastante búsqueda de información y comparación con poco margen. He de destacar también que la curva de aprendizaje sobre la forma de personalizar CAS con cas-overlay-template ha sido algo alta, sobre todo a la hora de parametrizar el servidor embebido Apache Tomcat que venía con CAS y la poca experiencia con gradle al comenzar el trabajo.

Aunque al principio sí que me ha costado un poco más de tiempo, luego se ha visto compensado a la hora de realizar la instalación de varios nodos CAS, ya que para ello no era necesario tener que volver a repetir todo el proceso en otro servidor, sino que bastaba con disponer de una instalación de java para su ejecución, y con sólo cambiar el puerto de ejecución y volver a empaquetar el proyecto ha sido suficiente para tener operativo otro nodo.

En cuanto a la planificación inicial, sí es cierto que las tareas de Integración de LDAP con CAS y el login con Certificado Digital o DNIe me han tomado más tiempo del esperado, aunque luego han sido compensando en otras tareas que sí han requerido menos tiempo del planificado.

He de mencionar también las limitaciones que me he encontrado a la hora de poder definir toda la arquitectura lo más semejante a como se podría poner en un entorno de Producción, ya que muchos de estos recursos al ser parte de arquitectura de sistemas, como pueden ser DNS, firewalls, varias máquinas físicas, etc. no los he podido replicar correctamente. Aún así estoy bastante satisfecho con el resultado, teniendo en cuenta que el PC donde se ha llevado a cabo el presente TFM cuenta con una antigüedad de 7 años y en la fase final donde tenía que arrancar todas las máquinas virtuales para realizar las pruebas oportunas el sistema se volvía bastante lento.

En cuanto a la metodología ágil seguida para la consecución del proyecto ha sido bastante acertada, ya que me ha permitido tener una visión exacta de la evolución de éste mediante el panel Kanban.

5.4. Líneas de trabajo futuros

Como posibles trabajos futuros que no han podido ser abordados en el presente TFM pueden ser:

- Personalización de la pantalla de login de CAS. Aunque CAS provee una pantalla de login, es necesario que ésta sea personalizada acorde a los estilos de la Corporación. En este caso sólo es necesario personalizar la jsp de la pantalla de login, cambiando los estilos.
- Definir política de seguridad en cuanto a las contraseñas. Se debe definir una política de seguridad que cumpla con el ENS (Esquema Nacional de Seguridad) como la renovación de ésta cada 60 días, caracteres admitidos, etc. e implantar dicha política sobre el servidor CAS para que se apliquen a todas las aplicaciones.
- Definir el acceso con certificado como alternativo. Para mejorar la experiencia de usuario, cuando se acceda a CAS siempre nos debe mostrar el formulario de introducir usuario/contraseña y un botón alternativo de autenticación con certificado. De esta forma el usuario selecciona si desea autenticarse con certificado digital o no y no mostrando la ventana de elección de certificado al intentar acceder al SSO, lo que puede provocar confusión a los usuarios, al pensar que es necesario un certificado para poder autenticarse.

6. Glosario

A continuación se incluye la definición de los términos y acrónimos más relevantes utilizados en esta Memoria.

- **SSO**: Single Sing-On. Método que permite a los usuarios acceder a varias aplicaciones con un único nombre de usuario y contraseña.
- **ENS**: Esquema Nacional de Seguridad.
- **Autenticación**: La autenticación es el proceso de verificar la identidad de un usuario.
- **Autorización**: La autorización es el proceso de verificar si un usuario posee permiso para realizar una acción específica o acceder a un determinado recurso.
- **Login**: Es el proceso de iniciar sesión en un sistema informático o en una aplicación web.
- **Logout**: Es el proceso de cerrar la sesión en un sistema informático o en una aplicación web.
- **Servidor**: Es un dispositivo que se utiliza para procesar y gestionar las solicitudes de los clientes y dar una respuesta a éstas.
- **Máquina virtual**: Es un software que simula la existencia de una computadora completa, con su propio sistema operativo y aplicaciones.
- **IDE**: Es el acrónimo de Entorno de Desarrollo Integrado y es una aplicación que proporciona a los desarrolladores un conjunto completo de herramientas para crear, deputar y probar software.
- **Software**: Conjunto de programas que permiten a la computadora realizar determinadas tareas.
- **Firewall**: Es un software que se encarga de proteger una red de dispositivos informáticos de ataques externos no deseados.
- **DMZ**: Acrónimo de Zona Desmilitarizada, es una zona de la red que se utiliza para alojar servicios y aplicaciones que están disponibles para el acceso externo.
- **Intranet**: Red interna de una organización.
- **Alta Disponibilidad**: Se refiere a la capacidad de un sistema o servicio de estar disponible y funcionar de forma correcta sin interrupciones.
- **Maven**: Es una herramienta de gestión de proyectos, utilizada principalmente para gestionar las dependencias, realizar compilaciones y pruebas y generar informes de documentación.
- **LDAP**: Es el acrónimo de Lighweight Access Protocol, es un protocolo de red utilizado para acceder y gestionar información almacenada en un directorio.
- **CAS**: Es el acrónimo de Central Authentication Service, es un sistema de autenticación de usuarios diseñado para proporcionar acceso seguro a

varias aplicaciones y sistemas a través de un único punto de inicio de sesión.

- **Proxy**: Un proxy es un servidor que actúa como intermediario entre un cliente y otro dispositivo o servidor.
- **Proxy inverso**: Es un tipo especial de proxy que se utiliza para enrutar las solicitudes de red desde Internet hacia un servidor o grupo de servidores detrás de firewall.
- **Spring**: Es un marco de aplicaciones Java de código abierto que se usa para desarrollar aplicaciones.
- **Java**: Es un lenguaje de programación orientado a objetos.
- **SSL:** Es el acrónimo de Secure Sockets Layer y es un protocolo de red que se utiliza para establecer una conexión segura entre dos dispositivos a través de una red.
- **TSL**: Es el acrónimo de Trasnport Layer Security y es un protocolo de red que proporciona seguridad para las comunicaciones a través de red, es una versión actualizada y mejorada de SSL.
- **X509:** Es un estándar de la industria para el formato de certificado digital.
- **Certificado Digital:** Es un archivo digital que contiene la información sobre una persona o empresa que se utiliza para establecer la identidad en línea.
- **DNIe:** DNI electrónico.
- **Balanceador de carga:** Es un dispositivo que distribuye el tráfico de red entrante en varios servidores o dispositivos para equilibrar el número de peticiones que cada uno de ellos recibe.

7. Bibliografía

1. **Iria da Cunha**, “El trabajo fin de grado y de máster: Redacción, defensa y publicación”, Primera Edición FUOC, 2015
2. Guía transversal para el estudiantado de TF de los Estudios de Informática, Multimedia y Telecomunicación. Septiembre 2022
3. ¿Qué es el SSO? https://www.cloudflare.com/es-es/learning/access-management/what-is-sso/. Consultado en Octubre de 2022.
4. LDAP. Protocolo ligero de acceso a directorios. https://es.wikipedia.org/wiki/Protocolo_ligero_de_acceso_a_directorios. Consultado en Octubre de 2022.
5. CAS Enterprise Single Sign-On for All https://apereo.github.io/cas/6.6.x/index.html. Consultado en Octubre de 2022
6. Arquitectura CAS. https://apereo.github.io/cas/6.6.x/planning/Architecture.html. Consultado en Octubre de 2022
7. Protocolo CAS. https://apereo.github.io/cas/6.6.x/protocol/CAS-Protocol.html. Consultado en Octubre de 2022.
8. JOSSO EE. https://josso.atricore.com/. Consultado en Octubre de 2022.
9. OpenAM Community Edition. https://github.com/OpenIdentityPlatform/OpenAM. Consultado en Octubre de 2022.
10. Qué es SAML y cómo funciona https://ciberseguridad.com/herramientas/saml/. Consultado en Octubre de 2022
11. Repositorio Git CAS (Central Authentication Service) https://github.com/apereo/cas. Consultado en Octubre de 2022.
12. ¿Qué es kanban? Explicación para principiantes. https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban. Consultado en Octubre de 2022.
13. ¿Qué es SCRUM? https://proyectosagiles.org/que-es-scrum/. Consultado en Octubre de 2022.
14. Ministerio de Política Territorial y Función Pública. Plataforma Cl@ve 2 “Manual de Integración para proveedores de Servicios”. Revisión 2.8. Septiembre de 2021.
15. Centro de transformación tecnológica. “Cl@ve identificación”. https://administracionelectronica.gob.es/ctt/clave. Consultado en Octubre de 2022.
16. Servidor HTTP Apache. https://es.wikipedia.org/wiki/Servidor_HTTP_Apache. Consultado en Noviembre de 2022.

17. HTTP SERVER PROJECT. https://httpd.apache.org/. Consultado en Noviembre de 2022.
18. OpenLDAP. https://es.wikipedia.org/wiki/OpenLDAP. Consultado en Noviembre de 2022.
19. Sitio web OpenLDAP. https://www.openldap.org/. Consultado en Noviembre de 2022.
20. Apache Tomcat. https://tomcat.apache.org/. Consultado en noviembre de 2022.
21. Sitio Oficial de VMWARE. https://www.vmware.com/es/products/workstation-pro.html. Consultado en Octubre de 2022.
22. Web MVC Framework. https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html Consultado en Noviembre de 2022.
23. Spring Security. https://docs.spring.io/spring-security/reference/index.html. Consultado en Noviembre de 2022.
24. Hibernate ORM Documentation. https://hibernate.org/orm/documentation/6.1/ Consultado en Noviembre de 2022.
25. Java Apereo CAS Client. https://github.com/apereo/java-cas-client. Consultado en Octubre de 2022.
26. Para qué sirve el protocolo LDAP y cómo funciona. https://www.redeszone.net/tutoriales/servidores/que-es-ldap-funcionamiento/#538404-que-es-ldap-y-para-que-sirve. Consultado en Octubre de 2022.
27. Proxy Inverso. https://es.wikipedia.org/wiki/Proxy_inverso. Consultado en Octubre 2022.
28. Balanceador de Carga. https://es.wikipedia.org/wiki/Equilibrador_de_carga. Consultado en Octubre de 2022
29. Manual Referencia MySQL. https://dev.mysql.com/doc/refman/8.0/en/. Consultado en Octubre 2022.
30. ¿Qué es un firewall? https://www.cisco.com/c/es_mx/products/security/firewalls/what-is-a-firewall.html. Consultado en Octubre 2022
31. Integración de CAS con LDAP. https://apereo.github.io/cas/6.6.x/authentication/X509-Authentication.html. Consultado en Noviembre de 2022.
32. CAS Autenticación X.509. https://apereo.github.io/cas/6.6.x/authentication/X509-Authentication.html. Consultado en Noviembre de 2022.

33. Spring-Security. CAS Authentication. https://docs.spring.io/spring-security/site/docs/5.2.11.RELEASE/reference/html/jc-authentication.html#cas. Consultado en Noviembre de 2022.
34. Registro de Tickets mediante JPA. https://apereo.github.io/cas/6.6.x/ticketing/JPA-Ticket-Registry.html. Consultado en Noviembre de 2022.
35. Creación de BBDD y usuarios en MariaDB. https://minds-lab.com/crear-un-usuario-y-otorgarle-permisos-sobre-una-base-de-datos-mysql-mariadb/. Consultado en Noviembre de 2022
36. OpenSSL. “Cryptography and SSL/TLS Toolkit”. https://www.openssl.org/. Consultado en Noviembre de 2022.
37. How to Install and Configure Apache Reverse Proxy Server With SSL/TLS Encryption. https://www.cherryservers.com/blog/how-to-install-and-configure-apache-reverse-proxy-server-with-ssl-encryption. Consultado en Noviembre de 2022.
38. Apache Module mod_proxy_balancer. https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html. Consultado en Noviembre de 2022.

8. Anexos

9.1. Instalación CAS Overlay Template

En este apartado se procederá a la descarga del proyecto CAS Overlay Template para proceder a construir el WAR por defecto que nos proporciona CAS y poder configurar y personalizar CAS para adaptarlo a nuestras necesidades.

Comenzamos instalando la JDK de Java con el comando

sudo apt install default-jdk.

Tras la instalación de la JDK procedemos a la instalación de *git*[15] con el comando

sudo apt install git

Una vez instalado procedemos a descargarnos la rama 6.6 de cas-overlay-template del repositorio git, para ello ejecutamos el siguiente comando en nuestro directorio de trabajo

git clone --branch 6.6 https://github.com/apereo/cas-overlay-template

Una vez clonado el repositorio tendremos en nuestro directorio una estructura como la siguiente

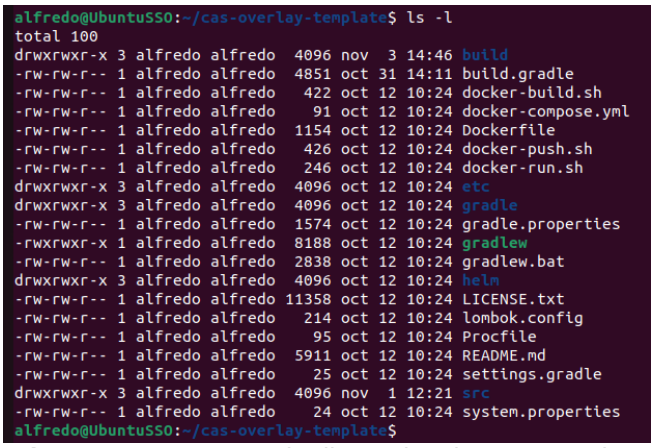


Figura 54: Estructura de directorios de CAS Overlay

Si seguimos las instrucciones que se indican en el repositorio Git[16] vemos que nos indica que la primera tarea que debemos ejecutar para construir el proyecto es

./gradlew clean build

La ejecución se ha completado satisfactoriamente, como se puede apreciar en la siguiente imagen

Figura 55: Ejecución del comando ./gradew clean build

15 https://git-scm.com/

16 https://github.com/apereo/cas-overlay-template/tree/6.6

Si nos vamos al directorio *build/libs* podemos observar cómo se ha generado el fichero **cas.war**, el cual podemos ejecutar.

Para que el servidor se pueda ejecutar correctamente es necesario crear un almacén de claves, ya que el fichero cas.war generado viene configurado como una aplicación de Spring Boot que contiene un servidor Apache Tomcat embebido para su ejecución. Éste servidor por defecto viene configurado para escuchar en el puerto 8443 mediante el protocolo TSL, de ahí que sea necesario generar el almacén de claves. Para ello se puede hacer uso de la herramienta *keytool* de la JDK o mediante la tarea gradle *createKeyStore*. En este caso por simplicidad hacemos uso de la tarea gradle lanzando el siguiente comando **./gradlew createKeystore --info**

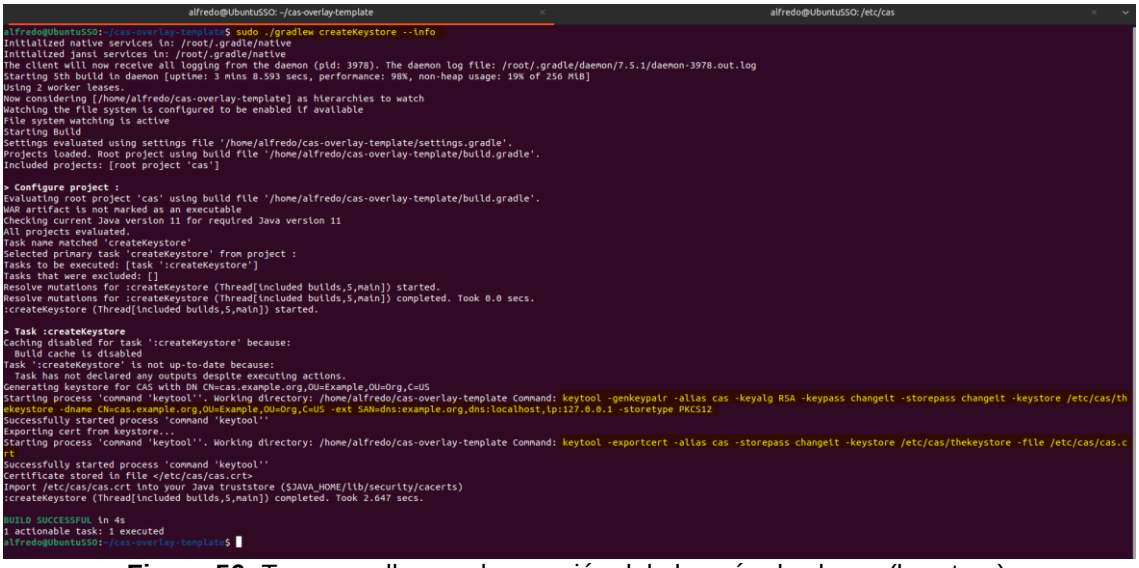


Figura 56: Tarea gradle para la creación del almacén de claves (keystore)

Aquí se puede observar cómo la tarea gradle realmente lo que hace es lanzar el comando keytool de la JDK, donde como primer paso crea el certificado **cas.crt** con clave RSA y posteriormente crea el almacén de claves **thekeystore** donde añade éste certificado. Estos ficheros son generados en la ruta ***/etc/cas*** que es la ruta que tiene configurado el servidor CAS por defecto para buscar dicho almacén.

Una vez tenemos configurado el almacén de claves podemos probar a ejecutar el servidor CAS lanzando la tarea *run* de gradle. Tras lanzar la tarea se observa que el servidor ha arrancado sin errores, tal como se muestra a continuación.

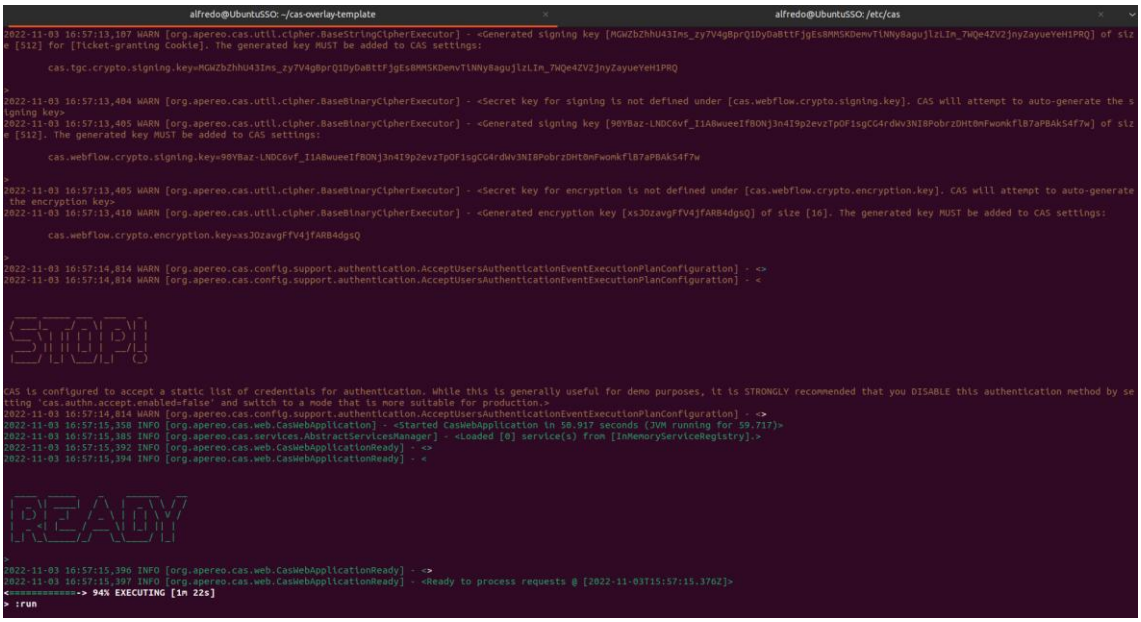


Figura 57: Tarea gradle para arrancar el servidor CAS

Si accedemos a la url donde se encuentra desplegado el servidor CAS, en nuestro caso https://192.168.52.135:8443/cas comprobamos cómo podemos acceder a la pantalla de login de CAS.

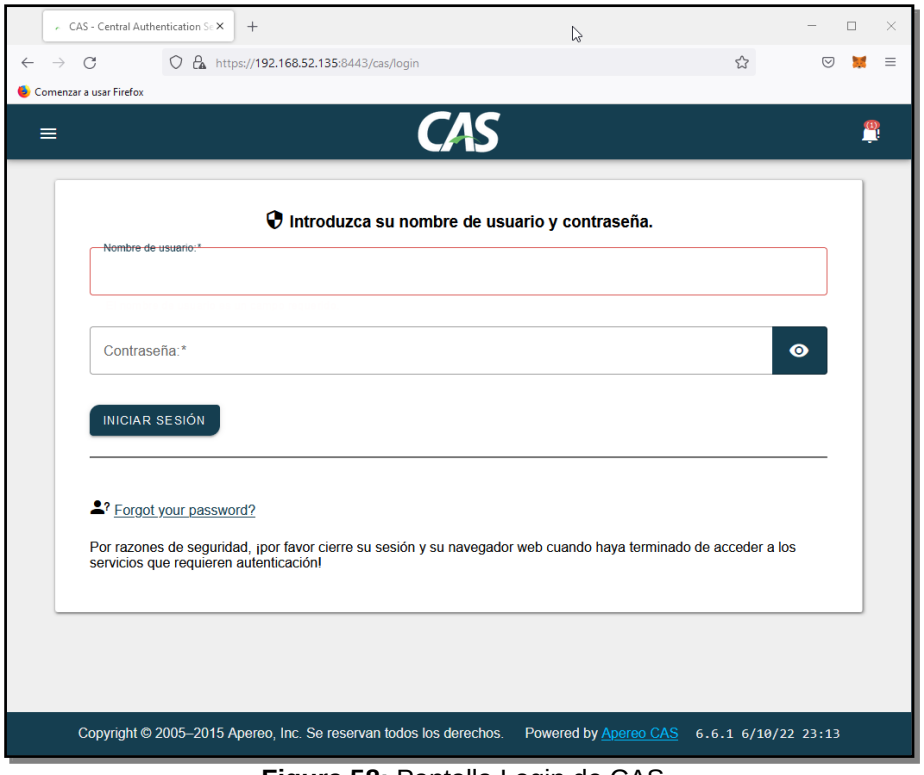


Figura 58: Pantalla Login de CAS

CAS viene por defecto con un usuario y password configurado en memoria (casuser/Mellon) que si lo introducimos podemos verificar que el sistema de login está funcionando correctamente. En la siguiente captura se puede observar cómo tras logarnos el sistema nos muestra ciertos atributos.

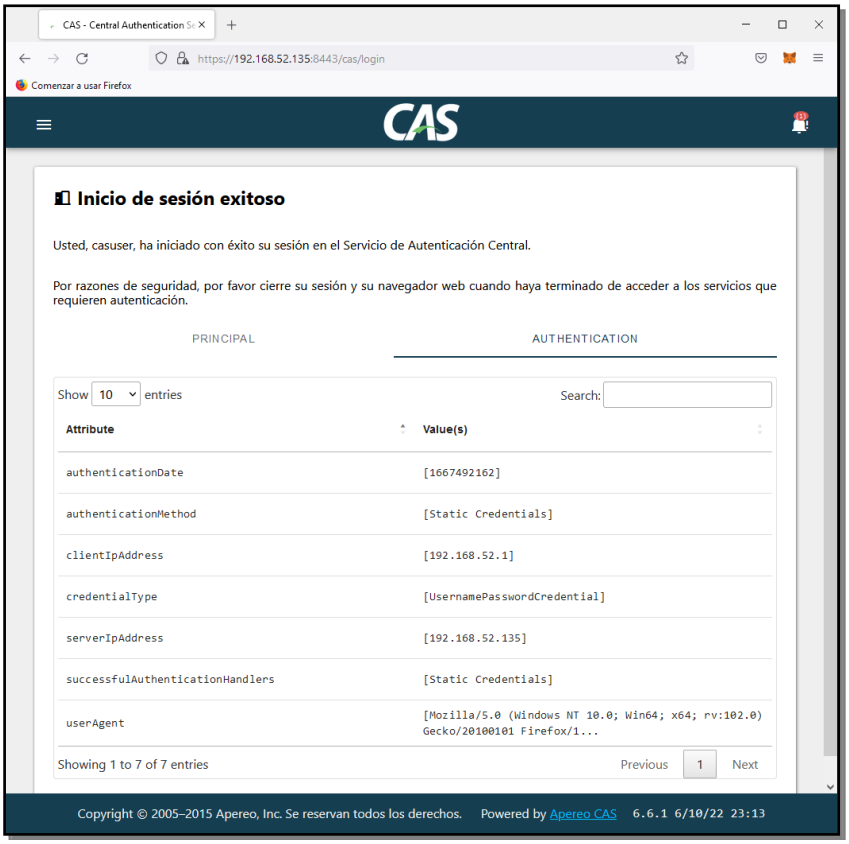


Figura 59: Pantalla información Login CAS

Si observamos la consola del servidor CAS se puede observar cómo va registrando cada uno de los eventos que se han producido en el servidor, en este caso la acción de autenticación y posteriormente el ticket TGT generado, mostrando tanto la fecha y hora de la creación como la IP desde donde se ha realizado la petición.

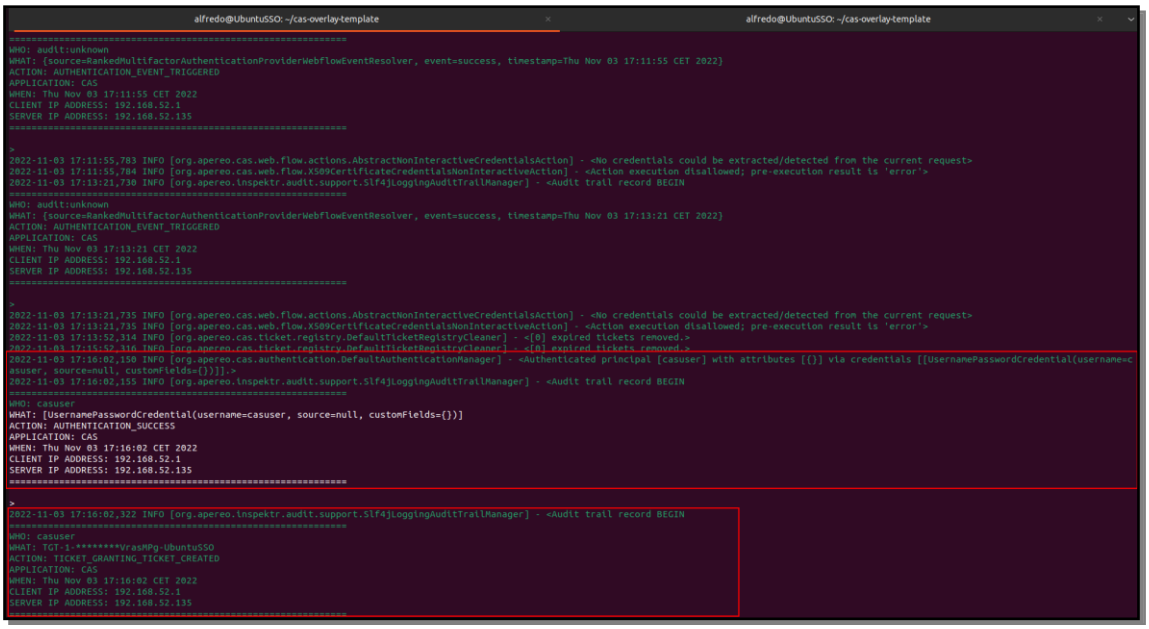


Figura 60: Consola de log del servidor CAS

De esta forma damos por finalizada la instalación inicial.

9.2. Instalación OpenLDAP

Para la instalación[17] de OpenLDAP en Ubuntu tiramos de los repositorios de éste. Para ello lanzamos el comando **sudo apt install slpad ldap-utils**, en este caso durante el proceso de instalación nos solicitará que introduzcamos una contraseña para el administrador a través de una pantalla como la siguiente

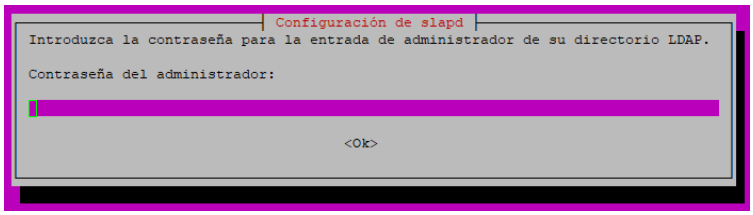


Figura 61: Instalación OpenLDAP. Solicitud Contraseña

Una vez introducida la contraseña y completada la instalación debemos pasar a la configuración de OpenLDAP. Para ello debemos ejecutar el comando **sudo dpkg-reconfigure slapd**. Durante el proceso de configuración lo primero que nos solicitará es que introduzcamos un nombre dominio DNS sobre el que se construirá la base DN del directorio LDAP. En nuestro caso incluiremos el domino “dipucadiz.es”

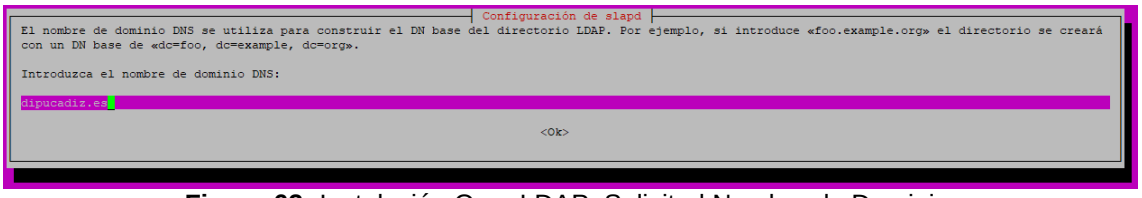


Figura 62: Instalación OpenLDAP. Solicitud Nombre de Dominio

Posteriormente nos solicitará el nombre de la organización, que en nuestro caso será “dipucadiz”

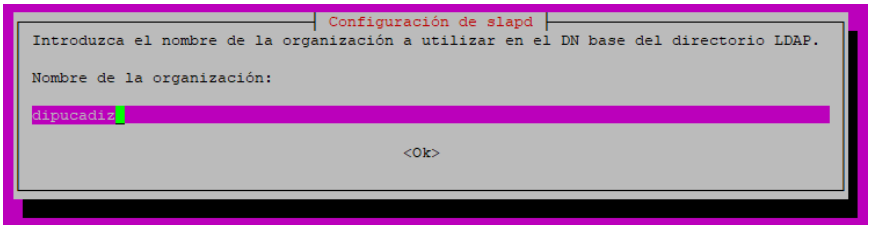


Figura 63: Instalación OpenLDAP. Solicitud Organización

Para comprobar la instalación podemos lanzar el comando **sudo slapcat**, el cual nos retornará información de la base de datos LDAP. Con esta comprobación damos por finalizada la instalación del servidor LDAP.

9.3. Creación de estructura y carga de datos en LDAP

Para poder añadir información a LDAP haremos uso de la los ficheros con extensión *.ldif* (LDAP Data Interchange Format), que básicamente son ficheros de texto plano con un formato específico, el cual podemos usar para la carga de datos.

Como primer paso, para lograr el esquema mostrado en la Figura 14, debemos crear las unidades organizativas correspondientes, en este caso “Empleados”,

17 http://somebooks.es/ldap-parte-1-instalar-openldap-en-ubuntu-20-04-lts/

“Departamento 1” y “Departamento 2”. Para ello creamos el fichero *ou-_empleados.ldif* con el siguiente contenido

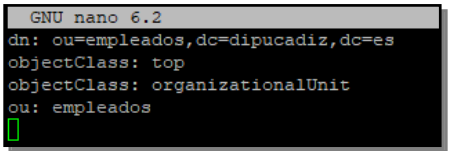


Figura 64: Carga de datos LDAP. Fichero ldif empleados

En este caso estamos creando la unidad organizativa empleados que cuelga de *es→dipucadiz→empleados*, para cargar el fichero lanzamos el comando

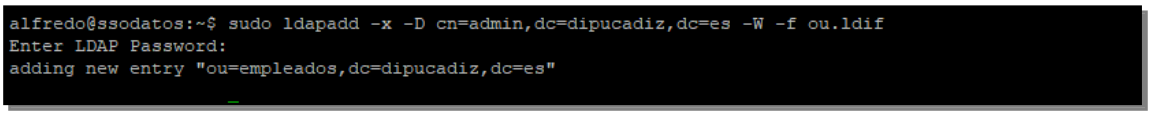


Figura 65: Carga de datos LDAP. Comando carga fichero ldif empleados

Donde se puede observar la nueva entrada añadida.

Ahora procedemos a crear las dos unidades organizativas que cuelgan de “empleados”, en este caso “departamento 1” y “departamento 2”, para ello creamos el fichero *ou_departamentos.ldif* con el siguiente contenido

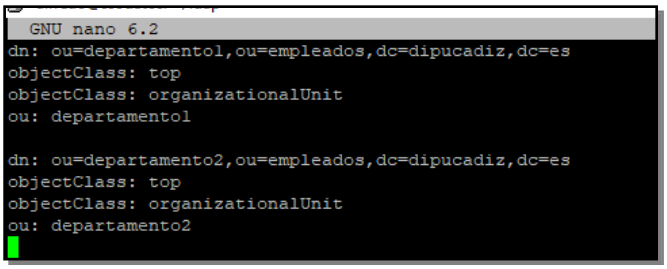


Figura 66: Carga de datos LDAP. Fichero ldif departamentos

Guardamos el fichero y lanzamos el comando para cargar el fichero ldif

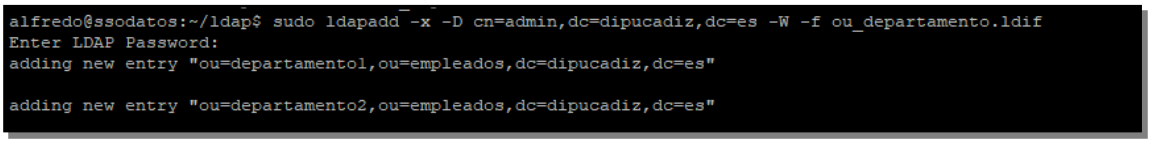


Figura 67: Carga de datos LDAP. Comando carga fichero ldif departamentos

Para comprobar que se han creado las unidades organizativas podemos lanzar el comando **sudo slapcat** donde podemos observar los diferentes elementos que se han creado.

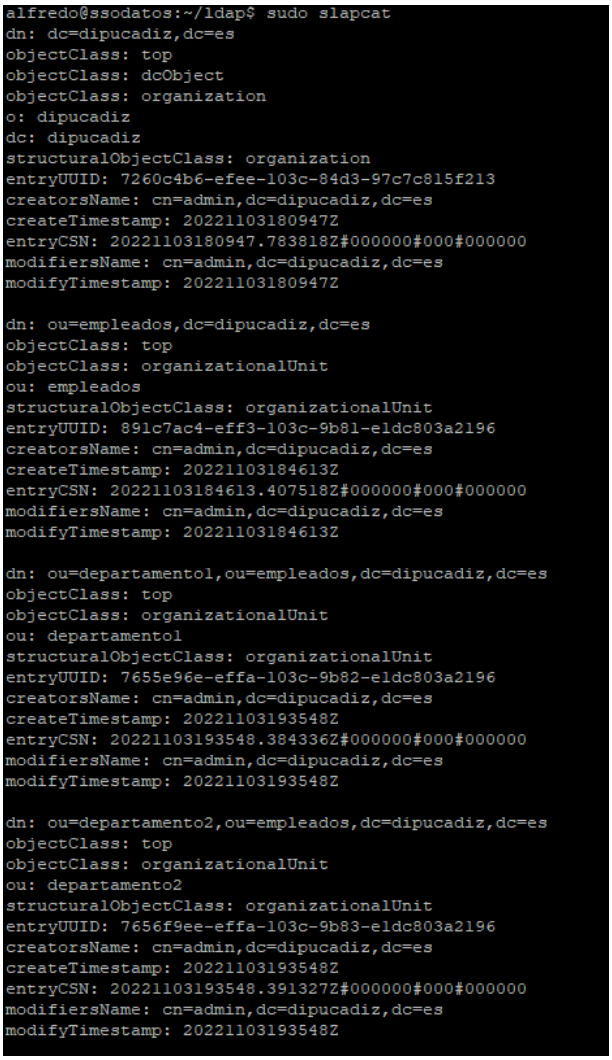


Figura 68: Carga de datos LDAP. Salida comando slapcat

Ahora procedemos a cargar algunos usuarios para poder realizar nuestras pruebas. Para añadir los usuarios seguimos el mismo procedimiento, creamos un fichero ldif con la información de éste, en este caso creamos el fichero *usuarios.ldif,* donde incluiremos nuestros usuarios de pruebas. Aquí debemos tener en cuenta que las contraseñas de los usuarios deben ir cifradas, para ello, podemos hacer uso del comando **slappasswd** el cual nos solicita una contraseña inicial y utilizando un algoritmo *SHA-1* nos retorna el hash de la contraseña introducida, siendo éste hash el que tenemos que incluir en el fichero ldif. A continuación se muestra la ejecución de éste comando, el cual nos retorna la cadena a incluir en el fichero

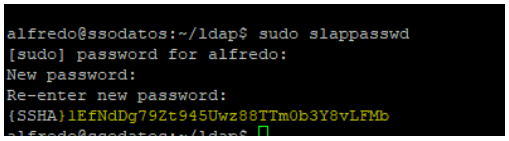


Figura 69: Carga de datos LDAP. Salida comando slappasswd

Posteriormente creamos el contenido del fichero *usuarios.ldif*, donde añadimos un usuario a cada uno de los departamentos de pruebas creados.

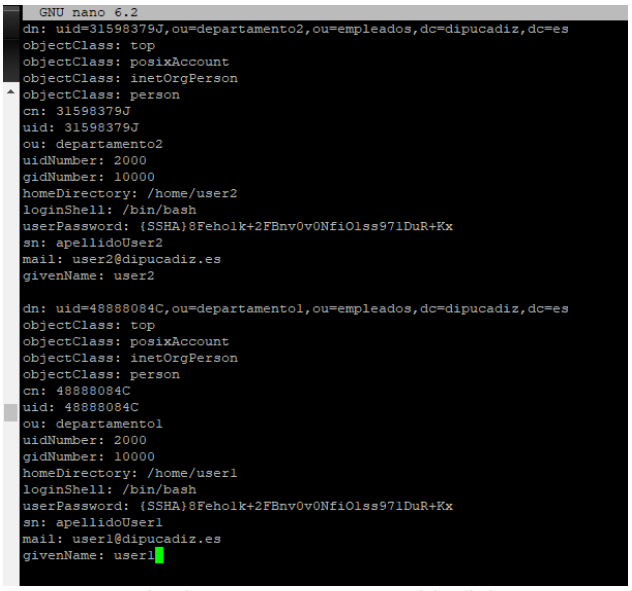


Figura 70: Carga de datos LDAP. Contenido fichero ususarios.ldif

Posteriormente cargamos el fichero usando el comando

sudo ldapadd -x -D cn=admin,dc=dipucadiz,dc=es -W -f usuarios.ldif

Si lanzamos el comando **sudo slapcat** podemos ver en la salida los dos usuarios creados

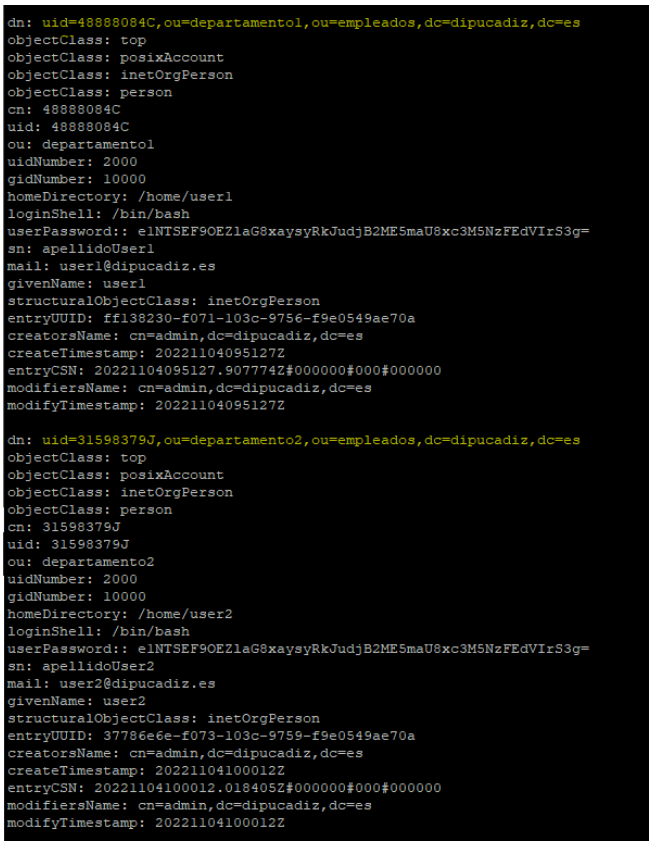


Figura 71: Carga de datos LDAP. Visualización usuarios creados

9.4. Construcción librería CAS Client Spring Security

Esta librería será el nexo de unión entre las aplicaciones cliente que se desean integrar con CAS y el servidor SSO.

Lo primero que debemos hacer es crear un proyecto maven, para ello desde Eclipse vamos a *File > New > Project* y seleccionamos *Maven Project*. En el diálogo que se nos abre marcamos el check *"Create a simple Project (skip archetype selecction)"* para crear un proyecto simple sin hacer uso de ningún arquetipo, tal como se muestra en la siguiente captura

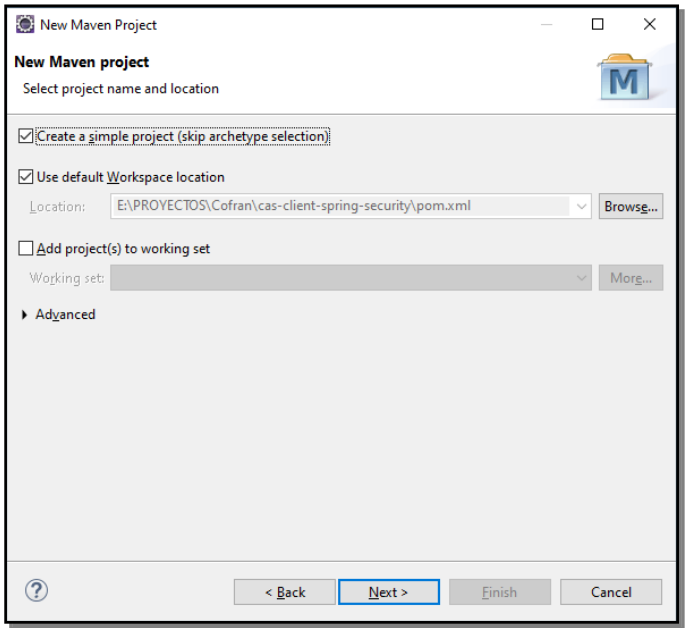


Figura 72: Construcción Cliente CAS. Creación nuevo proyecto.

En la siguiente pantalla incluimos el GroupId, ArtifactId y Versión, en nuestro caso incluimos los datos que se muestran a continuación

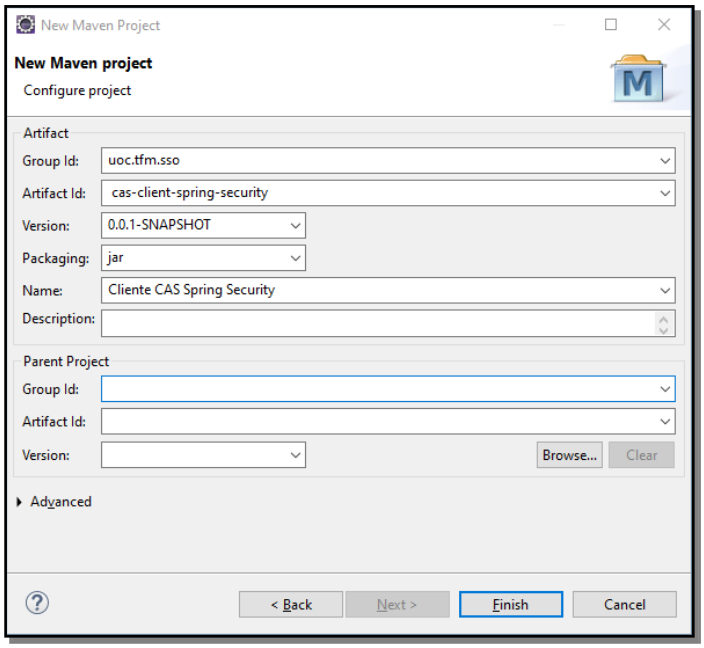


Figura 73: Construcción Cliente CAS. Selección GroupId y ArtifactId

Pulsamos sobre Finish y ya tenemos nuestra librería creada. Ahora debemos crear los componentes necesarios, para ello creamos la clase *CasClientConfig* y *CasUserDetails*.

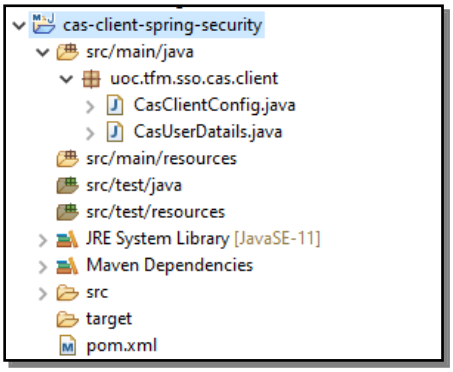


Figura 74: Construcción Cliente CAS. Estructura del proyecto

Veamos el contenido de estas dos clases. La clase *CasClienteConfig.java* es la clase que contiene todos los Beans de configuración que son necesarios, la parametrización de las URLs del servidor CAS y la URL de login y error de la aplicación cliente. A continuación se muestra el contenido de la clase

```java
/**
 * Clase de configuración que facilita la integración de las aplicaicones clientes con el SSO CAS Server. Esta clase contiene los Beans necesarios
 * para su integración y las propieades necesarias por defecto.
 *
 * @author Alfredo Franco Lara
 */
@Configuration
@ComponentScan(basePackages = {"uoc.tfm.sso.cas.client"})
public class CasClientConfig {

    @Value("${sso.appClientUrl}")
    String appClientUrl;

    @Value("${sso.appClientFailLogin}")
    String appClientFailLogin;

    @Value("${sso.endPoint:https://sso.tfm.uoc.edu:8443/cas}")
    String casEnpoint;

    @Value("${sso.loginUrl:https://sso.tfm.uoc.edu:8443/cas/login}")
    String casLoginUrl;

    @Value("${sso.logoutUrl:https://sso.tfm.uoc.edu:8443/cas/logout}")
    String casLogoutUrl;

    @Autowired
    private CasUserDatails casUserDatails;

    @Bean
    public ServiceProperties serviceProperties() {
        ServiceProperties serviceProperties = new ServiceProperties();
        serviceProperties.setService(appClientUrl + "/login/cas");
        serviceProperties.setSendRenew(false);
        return serviceProperties;
    }

    @Bean
    @Primary
    public AuthenticationEntryPoint casAuthenticationEntryPoint(ServiceProperties sP) {
        CasAuthenticationEntryPoint entryPoint = new CasAuthenticationEntryPoint();
        entryPoint.setLoginUrl(casLoginUrl);
        entryPoint.setServiceProperties(sP);
        return entryPoint;
    }

    @Bean
    public CasAuthenticationProvider casAuthenticationProvider() {

     CasAuthenticationProvider provider = new CasAuthenticationProvider();
     provider.setServiceProperties(serviceProperties());
     provider.setTicketValidator(ticketValidator());
     provider.setUserDetailsService(casUserDatails);
     provider.setKey("CAS_PROVIDER_LOCALHOST_9000");
     return provider;
    }

    @Bean
    public TicketValidator ticketValidator() {
        return new Cas30ServiceTicketValidator(casEnpoint);
    }

    @Bean
    public CasAuthenticationFilter casAuthenticationFilter(ServiceProperties sP, CasAuthenticationProvider authenticationProvider) throws Exception {
        CasAuthenticationFilter filter = new CasAuthenticationFilter();
        filter.setServiceProperties(sP);
        filter.setAuthenticationManager(new ProviderManager(Arrays.asList(authenticationProvider)));
        filter.setAuthenticationFailureHandler(authenticationFailureHandler());
        return filter;
    }

    @Bean
    public SecurityContextLogoutHandler securityContextLogoutHandler() {
        return new SecurityContextLogoutHandler();
    }

    @Bean
    public AuthenticationFailureHandler authenticationFailureHandler() {
        AuthenticationFailureHandler afh = new SimpleUrlAuthenticationFailureHandler(appClientFailLogin);
        return afh;
    }

    @Bean
    public LogoutFilter logoutFilter() {
        LogoutFilter logoutFilter = new LogoutFilter(urlLogoutBackToApp(), securityContextLogoutHandler());
        logoutFilter.setFilterProcessesUrl("/logout/cas");
        return logoutFilter;
    }

    @EventListener
    public SingleSignOutHttpSessionListener singleSignOutHttpSessionListener( HttpSessionEvent event) {
        return new SingleSignOutHttpSessionListener();
    }

    @Bean
    public String urlLogoutBackToApp() {
        return casLogoutUrl + "?cas_back_to=" + appClientUrl;
    }
}
```

Figura 75: Construcción Cliente CAS. Clase de configuración

En esta clase definimos los siguientes Beans:

- *serviceProperties:* Almacena las propiedades relacionadas con el servidor CAS, aquí principalmente se define la url de la aplicación cliente que será la encargada de procesar los tickets ST de CAS.
- *casAuthenticationEntryPoint:* Es el encargado de comenzar con el esquema de autenticación, aquí se define la URL de login del servidor CAS y se establecen las propiedades definidas anteriormente.

- *casAuthenticationProvider:* Es la implementación del proveedor de autenticación que se integra con el servidor CAS, en el proveedor se establece las propiedades definidas anteriormente, el validador de tickects y la implementación de UserDetailsService.
- *ticketValidator:* Se define el validador de tickets, que en nuestro caso usamos la implementación *Cas30ServiceTicketValidator* que usa el protocolo CAS versión 3.
- *casAuthenticacionFilter:* Filtro que se encarga de procesar todas las peticiones hacia */login/cas* que llegan a la aplicación una vez el usuario se ha logado en el servidor CAS, el cual se encarga de verificar que el ticket que presenta la URL es un tickect válido para proceder a autenticar al usuario en la aplicación.
- *secutiryContextLogoutHandler:* Se encarga de realizar un cierre de la sesión del usuario en la aplicación e invalida la sesión.
- *authenticationFailureHandler:* Aquí definimos la estrategia que se usará para manejar los intentos fallidos de autenticación. En este caso simplemente redirigimos hacia la url de error.
- *logoutFilter:* Se encarga de cerrar la sesión del usuario en el servidor CAS, aquí se define la url de vuelta para que el servidor CAS redirija una vez cerrada la sesión el control hacia la aplicación cliente. El filtro se ejecutará para las peticiones que se hagan hacia */logout/cas*
- *singleSignOutHttpSessionListener:* Listener usado para detectar cuando se destruye una sesión HTTP para eliminarla del mapa de sesiones administrativas.
- *urlLogoutBackToApp:* Este Bean compone un String que contiene la url de logout del servidor CAS incluyendo el parámetro configurado en CAS para redirigir hacia la aplicación cliente una vez completado el logout.

Por otro lado tenemos la clase *CasUserDetails.java* que implementa la interfaz *UserDetailsService* utilizada para la carga específica de datos de usuario en la aplicación cliente. Las aplicaciones clientes deben poseer un bean *UserDetalService* que será el que se inyecte en esta clase y se encargará de lanzar el método *loadUserByUsername* implementado en la aplicación cliente. Aquí llegará el usuario logado en el servidor CAS y será donde se proceda a verificar que el usuario se encuentra en la BBDD de la aplicación para poder cagar sus datos y otorgar autorización para poder acceder a la aplicación. A continuación se muestra el contenido de la clase CasUserDetails

```
@Service
public class CasUserDatails implements UserDetailsService {

    @Autowired
    private UserDetailsService userDetailService;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return userDetailService.loadUserByUsername(username);
    }


}
```

Figura 76: Construcción Cliente CAS. Implementación de UserDetailsService

9.5. Instalación de MariaDB

Procedemos a la instalación del servidor de base de datos MariaDB en la misma máquina donde tenemos instalado el servidor LDAP. Para ello lanzamos el comando ***sudo apt install mariadb-server***, una vez instalado el servidor procedemos a crear la BBDD *"cas_sso"*, una vez creada, añadimos un nuevo usuario, que sólo podrá acceder desde nuestra red y le damos permiso sobre la BBDD, en la siguiente captura se muestran los comandos ejecutados.

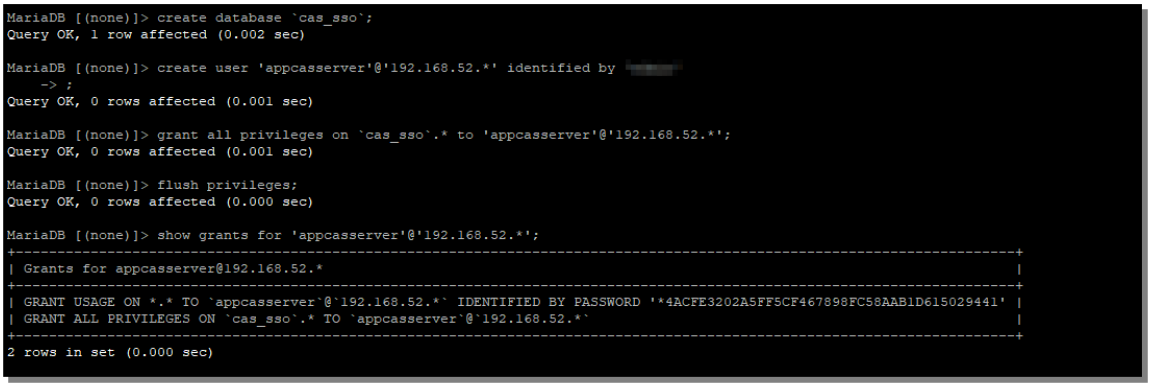


Figura 77: Instalación MariaDB.

Una vez creada la BBDD y el usuario, probamos a conectar desde el cliente DBeaver[18] para probar que el usuario puede conectar sin problemas y se procede a crear la tabla necesaria para que CAS almacene los tickets. Para ello lanzamos la siguiente sentencia

```
create table Cas_Tickets (
        id varchar(768) not null,
         body text not null,
         creation_Time datetime not null,
         parent_Id varchar(1024),
         principal_Id varchar(1024),
         type varchar(1024) not null,
         primary key (id)
    )
```

Figura 78: Instalación MariaDB. Creación Tabla Cas_Tickets

De esta forma ya tenemos nuestro servidor de BBDD operativo para poder configurar los servidores CAS.

9.6. Instalación de Apache HTTP

Procedemos a la instalación de Apache HTTP en una nueva máquina virtual Ubuntu Server. Para su instalación ejecutamos el comando

sudo apt install apache2

Tras su instalación el servidor Apache se inicia automáticamente, el cual lo podemos comprobar con el comando ***sudo systemctl status apache2***

18 https://dbeaver.io/

Figura 79: Instalación Apache HTTP.