Sistema Integrado de Autenticación para la Universidad Tecnológica de Bolívar- MiUTB

Fredy Mendoza Vargas

Universidad Tecnológica de Bolívar
Facultad de de Ingeniería , Programa de Ingeniería de Sistemas
Cartagena Bolívar, Colombia
2016

Sistema Integrado de Autenticación para la Universidad Tecnológica de Bolívar- MiUTB

Fredy Mendoza Vargas

Trabajo de grado presentado como requisito para optar al título de:
Ingeniero de Sistemas

Directores:
PhD Juan Carlos Martínez Santos
Msc Edwin Puertas Del Castillo

Universidad Tecnológica De Bolívar
Facultad de de Ingeniería , Programa de Ingeniería de Sistemas
Cartagena Bolívar, Colombia
2016

Declaración de autoría.

Yo, FREDY MENDOZA VARGAS, declaro que este trabajo de grado realizado, "Sistema Integrado de Autenticación para La Universidad Tecnológica De Bolívar- MiUTB", es de mi autoría. Confirmo que:

- Este trabajo fue realizado parcial o totalmente como estudiante de pregado de esta universidad.
- Donde se consulta el trabajo publicado de otros, se hace la respectiva atribución.
- Reconozco las principales fuentes de ayuda para este trabajo.
- Este trabajo fue realizado bajo la supervisión de PhD. Juan Carlos Martínez Santos y Msc. Edwin Puertas Del Castillo, quienes me guiaron y orientaron en el proceso.

Firma:

Fecha:

Agradecimientos

Agradezco primeramente a Dios por permitirme realizar y presentar este trabajo. Asimismo, expreso mi reconocimiento y gratitud a los tutores Juan Carlos Martínez Santos, Edwin Puertas del Castillo por su orientación incondicional en el proyecto. También agradezco Al estudiante de maestría Leonardo Enrique Castellanos Acuña que con su valiosa colaboración y contribución a que todo este proyecto se llevara a cabo exitosamente. Agradezco a mis padres, familiares y amigos por su apoyo incondicional a lo largo del desarrollo de este trabajo de grado.

Resumen

El presente trabajo de grado, tuvo como propósito implementar un prototipo de un sistema que permita a los estudiantes de la Universidad Tecnológica de Bolívar acceder a varias aplicaciones web de diferentes ambientes y tecnológias que se asemejan a los sistemas ofrecidos por la UTB, por medio de un sistema Single Sign-On (SSO). Un sistema SSO, permite a un usuario acceder a múltiples servicios, o sistemas de aplicación después de ser autenticado solo una vez. En este trabajo se describen primeramente todos aquellos elementos necesarios para comprender el funcionamiento de un sistema SSO y algunas de las implementaciones disponibles.

Para dar cumplimiento al objetivo, asimismo se describe el análisis y el diseño del prototipo usando los componentes y características del sistema Central Authentication Service (CAS), un proyecto que provee un sistema SSO web empresarial de código abierto. El núcleo de este proyecto fue desarrollado en Java, usando el framework Spring MVC, Spring Webflow y los componentes de Java.

Finalmente se presentan cada uno de los pasos que se realizaron para la implementación del prototipo con el servidor CAS, los resultados obtenidos de la investigación y algunas de las recomendaciones para una implementación real y completa en el contexto de la Universidad Tecnológica de Bolívar.

Palabras claves: *Autentication, Autoritation, Single Sign-On, Computer and Software Inventory, Central Authentication Services, Service-oriented architecture, Software as a service, Enterprise integration, High Performance Computing..*

Contenido

Lista de Figuras

Abreviaturas

Abreviatura	Término
ASP	Active Server Pages
CAS	Central Authentication Service
CSS	Cascading Style Sheets
DB	Data Base
ESB	Enterprise Service Bus
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JAR	Java ARchive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JSP	Java Server Pages
JWT	JSON Web Token
LDAP	Lightweight Directory Access Protocol
LMS	Learning Management System
MVC	Model-View-Controller
NMI	National Science Foundation Middleware Initiative
OAuth	Open Authorization
POM	Project Object Model
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SAVIO	Sistema de Aprendizaje Virtual Interactivo
SIRIUS	Sistema Integrado de Recursos de Información Universitaria para el Servicio
SPNEGO	Simple and Protected GSSAPI Negotiation Mechanism
SSO	Single Sign On
ST	Service Tickets
TGT	Ticket Granting Ticket
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTB	Universidad Tecnológica de Bolívar
XML	eXtensible Markup Language

1 Introducción

En los últimos años, múltiples empresas e instituciones educativas han adquiridos o han desarrollado varios productos computacionales para administrar y gestionar sus recursos en diferentes ámbitos, es decir, el conjunto de elementos disponibles de la empresa como los datos, información almacenada, etc. Muchas de estas herramientas de cómputo tienen diferentes ambientes y tecnologías y sus correspondientes sistemas de autenticación y validez de la información para dar acceso a cada una de ellas.

Esto ha generado problemas con la gestión de identidad y en la interoperabilidad de la información, esto es, la incapacidad de los sistemas o componentes dentro de una institución para intercambiar información entre sí y utilizar la información intercambiada [1]. También puede crear problemas para la población de usuarios perteneciente a la institución, en mantener las credenciales (por lo general una identificación del usuario y su contraseña) de cada uno de los sistemas al cual utiliza, dado que, cada vez que se desea usar el usuario alguno de los sistemas ofrecidos por la institución es necesario digitar las credenciales de autenticación para poder acceder a ellas y usarlos, incrementando así la posibilidad de olvidar o perder las contraseñas e imposibilitando además el ingreso a estos recursos.

Se ha vuelto una necesidad para los usuarios, que el acceso a sus diferentes cuentas sea rápida y de forma segura, por lo que muchas compañías han implementado el concepto de seguridad como es el inicio de sesión único, conocido por su siglas en ingles SSO, como una solución para el problema planteado anteriormente. El inicio de sesión único, conocido por las siglas en ingles SSO, es un sistema que permite a un usuario acceder a múltiples servicios, o sistemas de aplicación después de ser autenticado solo una vez [2]. SSO brinda una solución poderosa que ofrece reducción de costos y mayor flexibilidad para los administradores web y usuarios. Varias soluciones de SSO han sido propuestas dependiendo de la tecnología que se use.

En este trabajo de tesis se describe en que consiste conceptualmente un sistema SSO. De igual manera se analizan y se exponen algunas tecnologías, protocolos, investigaciones realizadas por otros autores para la implementación de un prototipo de un sistema SSO dentro del contexto de los sistemas que ofrece la Universidad Tecnológica de Bolívar (UTB). Por último se presenta el diseño e implementación del prototipo del sistema SSO y los resultados obtenidos.

1.1. Planteamiento del problema

La UTB en los últimos años han adquiridos varios sistemas computacionales para administrar y gestionar su información. Estas herramientas de computo tienen diferentes ambientes y tecnologías, y cada uno tienen sus sistemas de autenticación para acceder a cada una de ellas. Actualmente la UTB ofrece una serie de servicios y/o aplicaciones, que son usadas de forma constante por los estudiantes, al tener que ingresar obligatoriamente a las distintas plataformas para la verificación de la información a la cual desean acceder; para esto deben ingresar por medio de un formulario digitando su nombre de usuario y contraseña registradas en la universidad, a cada una de las aplicaciones con las mismas credenciales.

Esto ha generado problemas con la gestión de identidad y en la interoperativilidad de la información. Los principales aspectos identificados son:

- El mantener diversos sistemas de autenticación de identidad independientes para una misma población de usuarios, produce un incremento en la complejidad en el manejo y/o un riesgo de seguridad asociado a la tendencia del usuario a usar contraseñas débiles para contrarrestar dicha complejidad
- Mayor riesgo asociado con la confidencialidad por tener que duplicar esfuerzos en garantizar niveles mínimos de seguridad en todas las herramientas.

1.2. Justificación

Actualmente la Universidad Tecnológica de Bolívar posee diferentes sistemas de información implementados en diferentes tecnologías de la información, lo que ha hecho que la diversidad y heterogeneidad de los mismos traiga con si diferentes problemas al momento de interactuar entre ellos al tener que ingresar a través de un formulario con su usuario y contraseña, permitiendo la autenticación y validez de la información. A continuación se listan y se detallan los diferentes aplicaciones con quien interactúan los estudiantes de la institución.

- **SIRIUS I *(Sistema Integrado de Recursos de Información Universitaria para el Servicio)***

  El sistema de gestión académica de la Universidad Tecnológica de Bolívar, estó conformado por el sistema BANNER [1] (hoy este sistema utiliza la versión 7.2) provisto por Sungard, pero actualmente es soportado y propietario del sistema por **ellucian**[2]. Es conocido institucionalmente como SIRIUS con el fin de soportar los procesos de admisión, matrícula y registro académico de los estudiantes matriculados en los diferentes programas. Igualmente se habilitó para realizar procesos relacionados con los

[1] http://www.ellucian.com/Software/Banner-Student/
[2] http://www.ellucian.com/

estudiantes de posgrado y en la actualidad se trabaja para migrar toda la información de posgrados del antiguo sistema a SIRIUS I.

Este sistema se encuentra instalado sobre el sistema operativo Oracle[3] SOLARIS que dispone la versión 10. Y el sistema de gestión de base de datos que usa es Oracle versión 10g. SIRIUS I tiene para los usuarios 2 tipos de interfaces que son:

- *Nativa:* una interfaz que se maneja internamente en la universidad, manipulado por los directores de programas, registro académico y los profesionales de apoyo. Esta interfaz está desarrollada con Oracle Form de la versión de Oracle 10g para acceder a los datos por medio de un navegador.
- *Web:* Es la interfaz que es usada por los profesores y estudiantes. Esta interfaz web corre en un servidor Apache en la versión 2.0. Esta desarrollada con HTML para la estructura de la página web y por el lenguaje de programación JAVA para la lógica y comunicación de los datos y la información presentada.

En cuanto a la gestión de seguridad del sistema, relacionado con el manejo de los tipos de usuarios que interviene en el sistema, SIRIUS I tiene 4 tipos o roles de usuarios que son: **administrador** *(manipula todo el sistema)*, **administrativos** *(son los directores de programas y profesionales de apoyo, acceden a la interfaz nativa)*, **estudiantes y profesores** *(tienen acceso a la interfaz web).*

SIRIUS I se comunica con otros sistemas de la universidad a través de vistas por medio de la herramienta de conexión de oracle DB links, que son tablas para compartir la información necesaria de las múltiples tablas que hay en el sistema, con la autorización de simplemente ver la información mas no modificar los datos de las tablas. Los sistemas que se comunica son: SIRIUS II, PRIMO, SAVIO. Actualmente el sistema posee un inconveniente tecnológico en cuanto a los certificados de seguridad, por motivo de las versiones de los algoritmos de cifrado del sistema BANNER de la versión 7.2 se encuentran obsoletos, y por tal motivos el sistema no soporta las últimas actualizaciones de los navegadores de internet que están en el mercado.

- **SIRIUS II**

  Sistema de gestión administrativa conformado por el sistema ICEBERG provisto por la compañía Caseware[4], conocido institucionalmente como SIRIUS II, atiende todo lo relacionado con la gestión del talento humano, administrativa y la financiera, puesto en operación en el año 2013. Este sistema se encuentra totalmente articulado con el de gestión académica. Este sistema se encuentra instalado sobre el sistema operativo

[3]https://www.oracle.com
[4]http://www.casewaresa.com/

CentOS[5] que dispone la versión 5 actualizada a 7. Y el sistema de gestión de base de datos que usa es Oracle versión 11g. SIRIUS II tiene para los usuarios 2 tipos de interfaces que son:

- Nativa: una interfaz que se maneja internamente en la universidad, manipulado por los usuarios de contabilidad, financiera y administrativa de la universidad. Esta interfaz está desarrollada con Oracle Form de la versión de Oracle 11g para acceder a los datos por medio de un navegador.
- Web: tiene 2 portales web, uno es para los estudiantes y otro para proveedores. Estos dos portales web actualmente ya se implementó, pero no se han realizado todas las pruebas suficientes.

En cuanto a la gestión de seguridad del sistema, relacionado con el manejo de los tipos de usuarios que interviene en el, SIRIUS II tiene 4 tipos o roles de usuarios que son: *Administrador del sistema, usuarios administrativos (administrativo, financiera, etc.), estudiantes y proveedores.*. Este sistema también comunica con otros sistemas de la universidad a través de vistas que suple SIRIUS I. Estos sistemas son: SIRIUS I y el sistema de biblioteca con referentes a las multas de los estudiantes.

- **SAVIO *(Sistema de Aprendizaje Virtual Interactivo)***

  Sistema de Aprendizaje Virtual Interactivo - SAVIO, empleado para apoyar el proceso docente tanto en los programas a distancia, virtuales y presenciales mediante el uso de las tecnologías de la información. Se dispone de una versión 3.1 de MOODLE[6] , que facilita la interacción profesor-estudiante. SAVIO opera con todas las características de un sistema de gestión de aprendizaje *(LMS)*. Su objetivo es administrar, distribuir y controlar las actividades de formación de los estudiantes. Este sistema se encuentra instalado en 2 servidores en la nube en Rackspace[7], Un servidor de aplicación (web) y un servidor de bases de datos. Ambos servidores tienen instalado el sistema operativo CentOS y dispone la versión 6.5. Y el sistema de gestión de base de datos que usa es Mysql[8] versión 5.5.

  Internamente en SAVIO se encuentra un sistema propio de la UTB de plan de trabajo y de evaluación de docente realizado en el lenguaje de programación PHP[9] y el sistema de gestor de datos es MySQL. Además, internamente tiene un sistema de encuesta que fue desarrollado por la universidad también PHP y su Gestor de base de datos es MongoDB[10] versión 2.6.8.

[5] https://www.centos.org/
[6] https://moodle.org/
[7] https://www.rackspace.com/
[8] http://www.mysql.com/
[9] http://php.net/
[10] https://www.mongodb.com/

SAVIO tiene para todos los usuarios un solo tipo de interfaz, una interfaz web, que corre en un servidor Apache[11] y dispone de la versión 2.2.15. y fue desarrollado en PHP, en la versión 5.5.38. En cuanto a la gestión de seguridad del sistema, relacionado con el manejo de los tipos de usuarios que interviene en el, SAVIO tiene 6 tipos o roles de usuarios que son: *administrador, estudiantes, profesores, profesionales de apoyo, directores de programas y consejero*

Este sistema se comunica con SIRIUS I por medio unos scripts, que es un intermediario entre SIRIUS I y SAVIO para obtener la información que facilita el sistema SIRIUS, procesa la información para SAVIO. Y además, para la autenticación se comunica con un servidor LDAP por medio de la herramienta ApacheDS[12], el cual es el sistema que controla el directorio. Este directorio también es gestionado por los scripts, ya mencionados anteriormente, para guardar la información de los usuarios.

- **Sistema de Información de Biblioteca**

  Está conformado por el nuevo software bibliográfico ALEPH y el metabuscador PRIMO provistos por ExLibris[13], que permiten mejorar la gestión de los servicios bibliográficos facilitando una mayor autonomía e interacción con la plataforma tecnológica de forma local y remota. ALEPH, es el software bibliográfico que permite la gestión de los recursos de la biblioteca, como por ejemplo, libros, DVDs, vídeos, los libros electrónicos y las publicaciones periódicas, etc. El sistema de gestión de base de datos que usa es Oracle.

  PRIMO es un motor de búsqueda que permite recuperar recursos de muchas bases de datos al mismo tiempo, y asimismo incluye los recursos internos que están en la base de datos de la biblioteca comunicándose con ALEPH por medio de datos. Actualmente dispone la versión 4.2, se encuentra en un servidor externo de la universidad que dispone de un sistema operativo Linux y el sistema de gestión de base de datos es MySQL. El Proceso de autenticación de los estudiantes como usuario en primo, tienen que autenticar ingresando un nombre de usuario y contraseña válidos. Las credenciales son verificadas consultando a un servidor local de la universidad externo usando el Protocolo Ligero de Acceso a Directorios (LDAP).

  Este sistema tiene 2 tipo de interfaz: (1) Nativo: es la interfaz que proporciona el software ALEPH, para el personal de biblioteca. (2) Web: es la interfaz que proporciona el metabuscador PRIMO. En cuanto a la gestión de seguridad del sistema, relacionado con el manejo de los tipos de usuarios que interviene en el, PRIMO tiene 11 tipos o roles de usuarios que son: *Club amigos, docente de cátedra y tiempo completo, egresados, empleado, estudiante de posgrado, Estudiante de pregrado, instituciones con convenio, intercambio, usuario especial, estudiantes de educación permanente.*

[11] https://httpd.apache.org/
[12] http://directory.apache.org/
[13] http://www.exlibrisgroup.com/

La presentación de estos proyectos de adquisición e implementación de sistemas de información ante los órganos de decisión corresponde al simple cumplimiento de las funciones y responsabilidades establecidas en la normativa interna. Sin embargo, lo que se puede observar es la heterogeneidad de los sistemas, lo cual afecta para la administración de cada una de las aplicaciones y la posibilidad de poder integrar dichos sistemas para el beneficio administrativo y para los estudiantes al desear acceder a cualquier aplicación. También se puede ver que es una falta de cultura organizacional.

Por ejemplo, algunos de los sistemas de información adquiridos (SIRIUS I, SIRIUS II, PRIMO) fueron llevados al consejo académico y administrativo para su adquisición y posterior implementación. Sin embargo, otros fueron desarrollados por iniciativa propia en los diferentes departamentos y direcciones. Al momento de desarrollar algunos de los sistemas mencionados, no se tuvieron en cuenta estandarizaciones y arquitecturas de software necesarias para su desarrollo, debido que no existen en la institución normas, estándares, tecnologías u otros elementos necesarios para una adecuada integración de una aplicación con otra. Aun cuando es cierto que en su momento no existía documentación de ningún procedimiento de control que asegurara la integración, la causa real del problema no es esa. El problema surge debido a que el desarrollo de esas aplicaciones se ha efectuado autónomamente por parte de las dependencias con personal contratado para esos desarrollos y obviando el acompañamiento de la Dirección de Tecnologías, que permitiría que la aplicación naciera con un diseño integrado a los sistemas pre-existentes.

Como resultado del análisis de los sistemas de la Universidad y el estado actual de los mismos, permitió identificar la pregunta problema en el cual se enfocó esta investigación:

- *Cómo construir y administrar un sistema único de verificación de identidad de usuarios, que le permita con una sola identificación acceder a todos los recursos autorizados.*

1.3. Objetivos

El principal objetivo de este trabajo es proveer un prototipo para integrar SIRIUS, SAVIO, Primo e ICEBERG con un sistema único de autenticación para los estudiantes de la UTB.

Para lograr el anterior objetivo se proponen los siguientes objetivos específicos:

- Contar con un estado del arte de las diferentes formas de procedimiento de autenticación SSO. Comparar y definir la optima como apoyo a cumplir el objetivo principal.
- Obtener un modelo arquitectónico de los sistemas SIRUS, SAVIO, Primo e ICEBERG que nos permita obtener el estado actual de los sistemas para los estudiantes de la UTB
- Implementar un prototipo de un sistema SSO que permite a los estudiantes de la universidad tecnológica de bolívar acceder a varias aplicaciones web de diferentes ambientes y tecnologías que se asemejan a los sistemas ofrecidos por la UTB. Esto es con el fin de demostrar y cumplir el objetivo principal.

2 Marco Teórico

A continuación, se presentaran las definiciones de los diferentes términos que ayudaran a enmarcar el proyecto investigativo, esto ayudara a entender mucho más los temas que se tratan en él.

2.1. Autenticación y Autorización

Antes de continuar, es indispensable hacer una distinción y tener una definición clara entre dos tipos de servicios ofrecidos por las tecnologías a describir en este capítulo, que son la autenticación y la autorización:

- **Autenticación:** Es el proceso que contiene un sistema, con el objetivo de no solo de identificar a una persona, sino verificar que esa persona es quien dice ser realmente [2].

- **Autorización:** Es la operación en la que una vez que el usuario este autenticado, se establecen los permisos de acceso a los recursos. ¿Quién puede acceder a qué? [3].

La Mayoría de las aplicaciones o servicios implementan servicios de directorio donde se crean objetos tales como usuarios, equipos o grupos, con el objetivo de la autenticación y autorización en una red distribuida utilizando distintos protocolos, principalmente LDAP [1].

2.2. Single Sign-On (SSO)

Es un sistema que permite a un usuario acceder a múltiples servicios, o sistemas de aplicación después de ser autenticado solo una vez. El proceso de un SSO (ver la **figura 2.1**) requiere que el usuario inicie sesión por medio de un portal solo una vez al comienzo, y luego durante la sesión el sistema SSO le proporciona de modo transparente al usuario el acceso a los diferentes servicios, recurso o aplicaciones del sistema que se encuentran asignados [2].

[1] LDAP (Lightweight Directory Access Protocol): Como su nombre lo indica es un protocolo a nivel de aplicación ligero para acceder a servicios de directorios. Un directorio es una base de datos especializada optimizado para la lectura, navegación y búsqueda. Directorios tienden a contener información descriptiva, basada en atributos y apoyar las capacidades de filtrado sofisticados

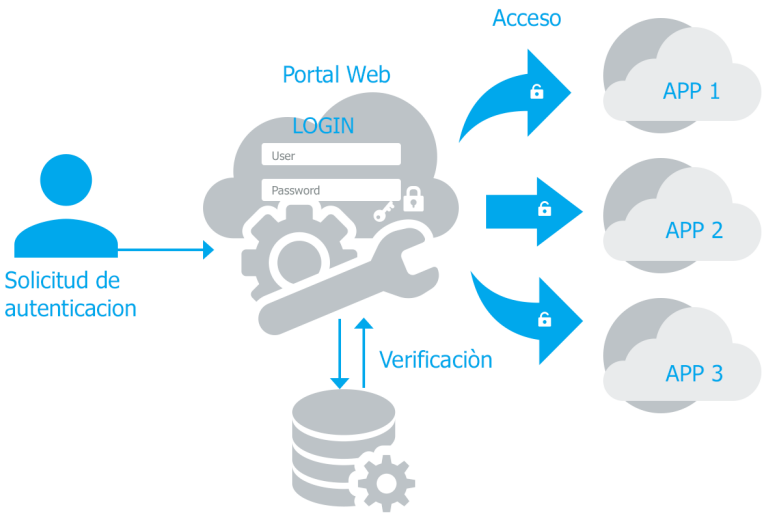


Figura 2.1: Proceso General de un sistema SSO.

SSO no se refiere a una sincronización de contraseñas, ya que en ese caso todas las aplicaciones y servicios funcionan con una misma contraseña no puede considerarse una implementación real, porque en lugar de fortalecer las características de seguridad del sistema, éstas se podrían considerar que están debilitándose dado que cuando todas las aplicaciones utilizan una misma contraseña, se corre el riesgo de que si un intruso logra conseguir la contraseña de una de las aplicaciones o servicios, inmediatamente tendrá acceso a todas ellas[5].

Un sistema SSO presenta grandes ventajas, por ejemplo: para los usuario, solo necesitan un único proceso de autenticación para acceder a cualquier aplicación reduciendo no solo el tiempo de autenticación, sino también en las incidencias relacionadas con el olvido de las diferentes contraseñas debido a que el usuario solo tiene que recordar sus credenciales (par de usuario/contraseña) una sola vez. También el área administrativa de una compañia también puede tener muchas ventajas en la gestión de los usuarios y recursos o servicios no va ser tan costoso, ya que al estar centralizados en un único punto reduce mucho los costos de seguridad para cada uno de los servicios.

2.3. Protocolos

Existen múltiples protocolos que han sido implementados para alcanzar los objetivos de la autenticación y autorización única, en este caso se explicarán brevemente los más relevantes:

2.3.1. Passport

Es un protocolo que permite a los usuarios hacer una única autenticación a diferentes aplicaciones, autenticándolos solo una vez en un servidor común. Passport es ampliamente desplegado por Microsoft. Cuando los usuarios inician sesión in el servidor Passport, este los redirige de vuelta al final del servidor. La información de autenticación es incluida en el mensaje redirigido en la cadena de consulta. Esta información es encriptada usando triple DES con una llave previamente establecida entre Passport y el servidor de la aplicación. Finalmente el servidor establece una cookie encriptada en el navegador del cliente [6].

2.3.2. Kerberos

Es un servicio de autenticación distribuido que permite a un proceso (cliente), ejecutarse en nombre de un mandante (usuario), para probar su identidad a un verificador (un servidor de aplicación, o solo un servidor) sin enviar información a través de la red que podría permitir a un atacante o al verificador hacerse pasar por el mandante. Kerberos opcionalmente provee integridad y confidencialidad para la información enviada entre el cliente y el servidor. Kerberos fue desarrollado a mediado de los 80s cómo parte del Proyecto Athena del MIT [7].

2.3.3. SAML 2.0

Es una versión de SAML, Security Assertion Markup Language. SAML es un estándar de mensaje abierto que codifica afirmaciones de seguridad y mensajes correspondientes al protocolo en formato XML. SAML 2.0 usa tokens de seguridad que contienen afirmaciones para pasar información acerca de un mandante entre una autoridad SAML (Proveedor de identidad) y un consumidor SAML (Proveedor de servicio) [8].

2.3.4. CAS

Es un protocolo basado en HTTP que requiere cada uno de suscomponentes para ser accedido a través de URIs específicas. CAS fue originalmente desarrollado por la Universidad de Yale para realizar autenticación única (SSO) [9]. CAS usa principalmente tickets incluyendo TGT (Ticket Granting Ticket), ST (Service Tickets) y demás, para verificar identidad de los usuarios. Un TGT indica al usuario que ha iniciado sesión exitosamente en el servidor CAS. Esta cookie mantiene el estado de la sesión del cliente, y mientras esta es válida el cliente puede presentarla en lugar de credenciales primarias. Un ST es un ticket enviado por CAS a un servicio para identificar ese servicio [10].

2.3.5. OAuth 2.0

OAuth es un protocolo que permite flujos simples de autorización para sitios web o aplicaciones. Este protocolo permite que un usuario que tiene determinados recursos en un servidor

(el proveedor de OAuth) pueda dar acceso a un tercero (el consumidor, usualmente un sitio web) a parte o todos esos recursos, sin necesidad de que ese tercero conozca su usuario y contraseña, ya que con esos datos tendría el control total de la cuenta. Actualmente existe una versión OAuth 2.0, una versión revisada y simplificada de OAuth. Oficialmente adoptada por grandes compañías como Facebook, Twitter, Google o Microsoft. Aventaja a la versión 1.0 en una mayor simplicidad de implementación, y en una arquitectura más robusta y que da soporte a mayor número de plataformas [11].

3 Estado del Arte

A continuación se realiza una revisión de algunos artículos, que por tener un fin similar a este trabajo pudieran servir como base de discusión para seleccionar el mejor modelo de implementación a desarrollar. De igual forma, se analiza también alguna de las herramientas o tecnologías existentes, ya sean pagas o gratuitas, que puedan también ayudar para lo que se desea implementar.

Teniendo en claro los conceptos fundamentales definidos en el capítulo anterior, se detallan diferentes modelos propuestos por los otros autores y los diferentes productos e implementaciones existentes en el mercado, referente al sistema de autenticación SSO:

Los Autores de [12] ilustran el diseño de una infraestructura de autenticación SSO Inter-Cloud, en la que es posible acceder a diferentes comunidades y recursos de cloud privado mediante SSO sin cambiar el diseño de los sistemas existentes. Para ello, utilizaron la técnica de mecanismos vinculados distribuidos y cuentas de Shibboleth[1]. Un repositorio de certificados de proxy se utiliza para asociar cada cuenta local en el sistema de nubes con una cuenta global para todos los sistemas interconectados. Específicamente Shibboleth se utilizó para la autenticación en el portal Web, soporte de gestión de usuarios distribuidos por las cuentas en la nube y la verificación y utilización de certificados. Además, implementaron un prototipo del sistema propuesto y confirmó su eficacia. En el experimento realizado, comprobaron que SSO estaba en funcionamiento entre dos lugares.

Los diseños propuestos por los autores [13, 14], proponen un esquema novedoso SSO basado en la federación de identidad. El ingreso, acceso, la comprobación de autoridad y el control de la autoridad se puede lograr mediante la adopción del "Ticket" basado en la estructura centralizada de Kerberos, sobre la base de verificación, declaración y afirmación mediante SAML. La Arquitectura propuesta específicamente por el autor [13], incluye tres módulos: el inicio de sesión, autenticación mediante Ticket y mediante una combinación de la idea del protocolo Kerberos en el módulo de autenticación y la gestión de verificación mediante el uso de una serie de lenguaje de toma de decisiones basada en la idea principal de SAML para garantizar el acceso de usuario válido en el servicio de seguridad. Su diseño SSO Se basó y se aplicó en un sistema integrado de radiodifusión y televisión, donde el sistema incluye los recurso de tres sistemas heterogéneos, con el objetivo principal de evitar la complejidad

[1] https://shibboleth.net/

de inicio de sesión varias veces y garantizar la seguridad de la información de los usuarios y la afirmación combinando la ventaja de Kerberos y SAML, garantizando la autenticación más la validación mediante la adopción de la autenticación mutua. Durante la autenticación SAML pudo proporcionarles mecanismos y estándares para la autenticación y autorización y proporcionar la seguridad y la interoperabilidad para la transmisión de información al igual que su autenticación.

Los autores [15] también plantean un esquema de SSO basado en la federación de identidad implementando un middleware orientado a servicios altamente distribuida mediante un bus empresarial (ESB) de código abierto, además de utilizar una arquitectura orientada a servicio SOA; utilizando una arquitectura de N-Capas.

3.1. Implementaciones de SSO Disponibles

A continuación, Se mencionan y se muestra una breve descripción de los más populares implementaciones, aplicaciones, tecnologías o herramientas SSO que se encuentran disponibles. Exponemos algunas de ellas:

3.1.1. Shibboleth

El sistema Shibboleth es un paquete software basado en estándares y de código abierto que permite establecer un SSO. Permite a los sitios para tomar decisiones informadas de autorización para el acceso individual de recursos en línea protegidas de una manera que preserva la privacidad. Este software implementa estándares de identidad federada ampliamente usados como SAML para ofrecer un servicio de SSO y un framework para el intercambio de atributos. Un usuario se autentica con sus credenciales de la organización, y la organización o el proveedor de identidad (componente encargado en autenticar al usuario) pasa la información de identidad mínima necesaria para el proveedor de servicios para permitir una decisión de autorización para acceder a los recursos [16].

3.1.2. CoSign Web SSO

Es un proyecto de código abierto diseñado originalmente para proveer a la universidad de **Michigan**[2] con un sistema seguro de autenticación web usando SSO. CoSing, actualmente hace parte de la *National Science Foundation Middleware Initiative (NMI)*[3]. Actualmente CoSign se encuentra en la versión 3.2.0 en el cual ya en esta versión no tienen los problemas de vulnerabilidad de sesión que fue descubierto por medio de ataques en todas las versiones anteriores de CoSign [17].

[2]http://umich.edu/
[3]http://www.nmi-edit.org/

Este proyecto tiene las siguientes características: Su funcionamiento en cuanto a las contraseñas si son usadas solo trabaja mediante un ambiente seguro (es decir mediante SSL), Soporta la autenticación por SPNEGO[4], factor de múltiple autenticación también es soportado, utiliza el protocolo Kerberos para solicitar las credenciales al servidor central para la autenticación (por ejemplo: LDAP, Oracle, etc).

3.1.3. adAS

Un proyecto de software que fue desarrollado por una empresa española llamada PRiSE[5] y se entrega a la comunidad como software libre bajo la licencia Apache 2.0. Es un servidor de autenticación avanzado que realiza funciones de proveedor de identidad, facilita el despliegue de un servicio SSO. Este sistema es capaz de obtener la información de los usuarios (id o usuario, password, etc) independientemente en qué fuente de dato se encuentre, puede trabajar con: *Oracle Database, PostgreSQL[6], MySQL, LDAP, Microsoft Active Directory, Microsoft SQL Server, etc.* Además permite integrar en un gran número de aplicaciones de muy distinta clase. La integración se realiza independientemente de las tecnologías de implementación que tengan y del entorno de despliegue al que pertenezcan, ya sean pagos o libres. Algunas de las aplicaciones compatibles con adAs [18]:

- *Drupal, Wordpress, moodle, Sakai, Mahara, Blackboard, entre otras. Además de estas aplicaciones, adAS es compatible con cualquier recurso que esté desarrollado bajo cualquiera de las siguientes tecnologías: Apache, php, Microsoft .NET, Java.*

3.1.4. KeyCloack

Es una solución SSO para aplicaciones web, servicios web REST y móviles, es de código abierto y libre y está licenciado bajo Apache License 2.0. Es un servidor de autenticación donde los usuarios pueden además de iniciar y cerrar sesión, también pueden administrar sus cuentas de usuario. Algunas características de este proyecto [19]:

- Puede ser usado para realizar conexiones sociales a través de red social favorita del usuario es decir, Google, Facebook, Twitter, etc.
- Soporta Integración de Active Directory y LDAP como servidor de directorio central para el registro y autenticación de los usuarios.
- El administrador puede ver las sesiones de usuario y qué aplicaciones / clientes tienen un token de acceso. Las sesiones pueden ser invalidadas por dominio o por usuario. También soporta el registro de usuarios.

[4]http://spnego.sourceforge.net/
[5]https://www.prise.es
[6]https://www.postgresql.org/

- Protocolos o las distintas formas para la autenticación y verificación que soporta son los siguientes: SAML 2.0, JSON Web Token (JWT)[7], OAuth 2.0, conexión OpenID.

- En cuanto a las aplicaciones tenemos que tiene soporte y se adapta: Tomcat, Jetty, y aplicaciones de Javascript/HTML5. Planes para soportar Node.js, RAILS, etc.

3.1.5. Central Authentication Service (CAS)

Este proyecto fue creado por la Universidad de Yale con el fin de proporcionar una aplicación para autenticar un usuario de manera que sea confiable. Desde el año 2004 CAS estaba siendo mantenido por Jasig, pero actualmente es soportado por la institución llamada **apereo**[8]. Este proyecto provee un sistema SSO web empresarial de código abierto. El núcleo de este proyecto fue desarrollado en Java, usando el framework Spring MVC, Spring Webflow y los componentes de Java [20].

La arquitectura de este sistema se comprende de dos componentes físicos que son el CAS Clients y CAS Server (**figura 3.1**). El **CAS Clients** tiene dos significados distintos en su uso común: (1) es cualquier aplicación habilitada para el CAS que pueden comunicarse con el servidor mediante un protocolo soportado o compatible. (2) Es un paquete de software que puede ser integrado con diversas plataformas y aplicaciones con el fin de comunicarse con el CAS Server mediante de algunos protocolos de autenticación que son soportados por CAS server. En cambio, El **CAS Server** su principal responsabilidad es autenticar a los usuarios y permitir el acceso a los servicios habilitados para CAS, comúnmente llamados CAS Clients, mediante la emisión y validación de tickets. Una sesión SSO se crea o es creada cuando el servidor emite un ticket de acceso (TGT) al usuario a través de un login exitoso. Un Ticket de Servicio (ST) es emitido para un servicio a petición del usuario mediante un navegador que redirecciona utilizando el TGT como un token. [21]

Como características tenemos que:

- Soporte para la autenticación de los usuarios una o múltiples conexión a LDAP, Bases de datos (Ej: Mysql, Postgresql, etc.), X.509, SPNEGO, etc.

- Soporte para múltiples protocolos como CAS (Protocolo implementado que es propio de este sistema), SAML, OAuth, OpenID, Protocolo REST, etc. Soporte de clientes MultiPlataformas (Java, .Net, PHP, Perl, Apache, etc). Se integra con uPortal, Liferay, BlueSocket, Moodle y Google Apps para nombrar unos pocos. También puede integrarse con una plataforma de SSO federada con Shibboleth.

[7] https://jwt.io/
[8] https://www.apereo.org/

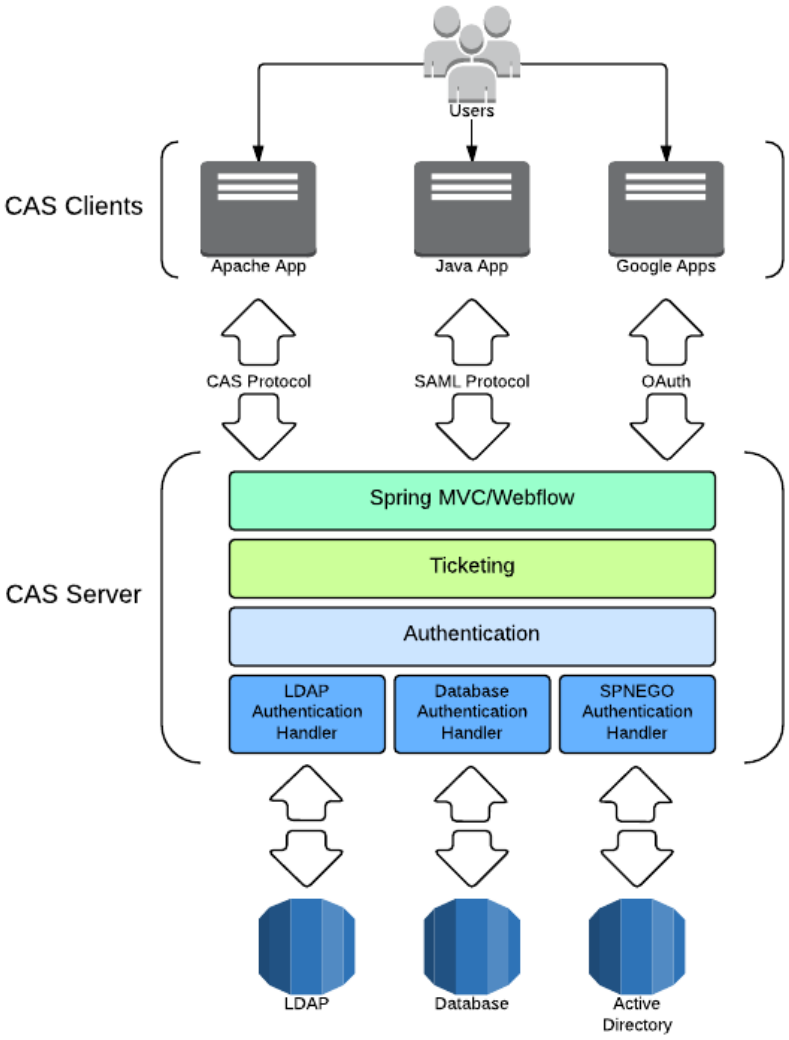


Figura 3.1: Arquitectura de CAS (tomado de la documentacion oficial de CAS [21]).

- Además tiene una aplicación web aparte para la administración de registros de los servicios que se desean integrar con este sistema llamado CAS Services Management

Como se puede observar, existen una gran cantidad de trabajos, proyectos e implementaciones de un sistema SSO, Desarrollado en múltiples tecnología y diversas tipos de licencias (ver **Tabla 3.1**).

Tabla 3.1: Implementaciones de sistemas SSO

Tipo de Licencia		**Nombre del Producto**
Libre	**Open Source**	CoSign Single sign on, Distributed Access Control Syste (DACS), Enterprise Sign On Engine, CAS, Barebones SSO, Shibboleth
	Apache 2.0	Keycloack, Thinktectur e Identity Server, WSO2 Identity Server, adAS, ZXID.
	GNU General Public License	Accounts & SSO, FreeIPA, JBoss SSO, JOSSO
Propietarios		Facebook Connect, Google Apps, Ubuntu SSO, SecureLogin, SafeID, Ping Dock, OneLogin, Numina Application Framework, NetIQ Access Manager, myOneLogin, Microsoft accoutn, Loginradius, Janrain Federate SSO, ILANTUS Sign-on Express, Imprivata OneSign, IBM Tivoli Access Manager, LTAP, HP IceWall SSO, Gigya SSO, Forefront identity Manager, Evidian, ComponentSpace SAML Suite, Clearlogin, CA Single Sign-On, Bitium, AuthAnvil Single Sign-on, Authen2cate, Auth0, Athens access and identity management, Active Directory Federation Services.

4 Análisis y Diseño del Prototipo

Al identificar la necesidad que actualmente la UTB presenta frente al acceso e identificación constante que deben tener los estudiantes para el uso continuo de los servicios, también el haber descrito las características y beneficios que presenta el uso de un sistema SSO y los modelos o implementaciones existentes, se tomó la decisión de implementar un prototipo de un sistema SSO que permita a los estudiantes acceder a las distintas plataformas a través de un solo portal web, utilizando como proveedor del SSO el sistema CAS.

4.1. ¿Por qué CAS?

Algunas de las principales razones que se consideraron para escoger CAS como proveedor de SSO fueron las siguientes:

- Permite integrar múltiples servidores de base de datos o de directorios activos como MySQL y LDAP. Esto da una gran ventaja para adaptar CAS en el contexto de la universidad, dado que algunos sistemas que ofrece a los estudiantes la UTB no se encuentran en un solo servicio de directorio o base datos. Por ejemplo, SAVIO tiene dos tipos de estudiantes que son: estudiantes de pregrado y maestría; Entonces las credenciales de los estudiantes de pregrados son provistos por un servidor de directorios LDAP, pero las credenciales de los estudiantes de maestría son provisto por una base de datos en MySQL. Debido a esto, CAS se pudo ajustar perfectamente.

- Admite integrar múltiples aplicaciones en distintas tecnologías o ambientes de desarrollo. Lo cual se consideró porque las aplicaciones o sistemas que ofrecen la UTB para los estudiantes, están o fueron construidas y desarrolladas en diferentes tecnologías que se puede integrar fácilmente con CAS.

- Proporciona una agradable comunidad de código fuente abierto que apoya y contribuye activamente al proyecto. Esta razón es la que diferencia de las otras implementaciones presentadas anteriormente.

- La familiaridad y la facilidad en la forma de instalación, configuración y extensión de los componentes de CAS. También su documentación es mucho más clara y está constantemente actualizada por su comunidad.

4.2. Análisis de Requerimientos

En esta sección, se expone el análisis de los requerimientos del sistema a proponer. Los requerimientos de un sistema son descripciones de cómo el sistema debe comportarse con el fin de tener una especificación de lo que debería ser implementado[27]. Por lo tanto, los requerimientos obtenidos del prototipo son:

- *Administración de aplicaciones:* El sistema debe permitir registrar o integrar, actualizar y eliminar una aplicación o recurso que facilita la universidad, Por ende el sistema debe tener una interfaz o dashboard para la configuración y administración de los aplicativos.
- *Gestión de acceso:* El sistema debe permitir acceder a los aplicativos registrados y permitidos para los estudiantes. Por lo tanto el sistema también debe tener una interfaz web para que los estudiantes puedan acceder directamente a las diferentes aplicaciones integradas al sistema SSO
- *Monitoreo y Estadísticas:* el sistema debe permitir que el administrador pueda tener una interfaz donde se proporcione informaciones estadísticas referentes de las sesiones de los usuarios al sistema, monitoreo de la configuración del servidor e información del servidor, etc.

4.3. Diseño del prototipo

A continuación la figura **4.1** representa las relaciones entre actores y los diferentes casos de uso basados en los requerimientos mencionados anteriormente. Los diagramas de casos de usos sirven para especificar la funcionalidad y el comportamiento del sistema deseado, por medio de una secuencia de interacciones entre el sistema y sus actores. Los actores de un diagrama de casos de usos son entidades externas al sistema modelado y qué pueden interactuar con él, pueden ser no solamente personas, sino también otros sistemas de hardware y software [29]. Las relaciones existentes entre casos de usos y actores pueden ser la interacción de un actor con un caso de uso, la extensión y consumo de caso de uso a otro.

Los actores considerados en el escenario propuesto y con los que interactúan el sistema, son los siguientes:

- ***Usuario no autorizado:*** Actor que representa un usuario no autorizado por el sistema. Este tipo de usuario interactúa con el sistema a través de un navegador web donde se muestra el interfaz del sistema hacia el usuario. Este actor solamente intentará ingresar y autenticarse en el sistema.
- ***Estudiantes:*** Usuario autorizado a utilizar el sistema. Sólo los estudiantes que se han identificado ante el sistema podrán acceder a los servicios ofertados por el mismo.

Interactúan con el sistema a través de un navegador web donde se muestra el interfaz del sistema.

- ***Administrador:*** Es el usuario autorizado (hereda) que tiene permisos especiales dentro del sistema. Se considera que el administrador es el primer usuario del sistema y que posee una clave especial. Interactúa con el sistema a través de un navegador web donde se muestra el interfaz del sistema.

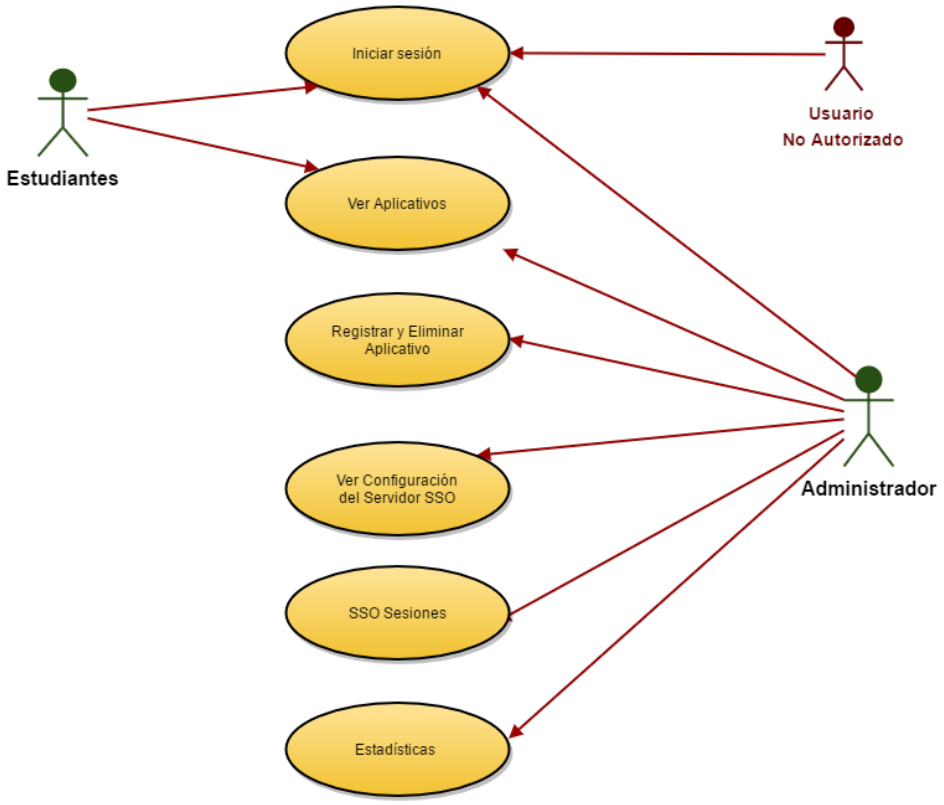


Figura 4.1: Caso de uso del prototipo.

4.3.1. Arquitectura del Prototipo

La figura **4.2** muestra un diseño de la arquitectura propuesta. Esta arquitectura tiene como objetivo crear una autenticación única para acceder a distintas aplicaciones, que en nuestro caso son: una aplicación de Moodle, una aplicación Java web , una en PHP, otra con ASP .NET y una aplicación con Wordpress por medio del servidor CAS, por consiguiente, para la gestión autenticación de los usuarios se usón como servicio de directorios un servidor LDAP y una base de datos en MySQL. Cada una de estas aplicaciones mencionadas representan a las aplicaciones existentes que ofrece la UTB.

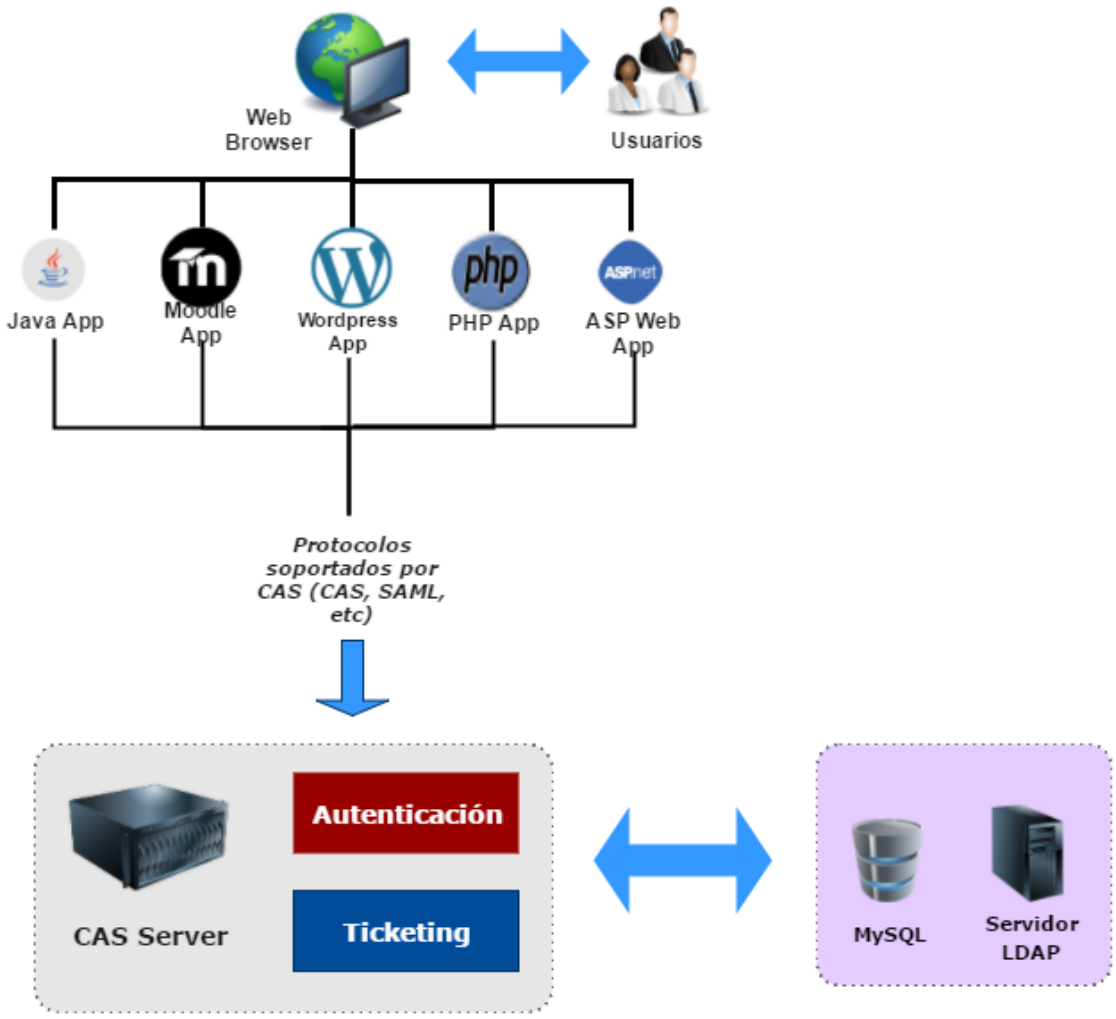


Figura 4.2: Arquitectura del prototipo.

4.3.2. Flujo de autenticación básica

En esta sección se indica el flujo básico de autenticación del SSO propuesto con CAS. Desde el flujo obtenemos la idea básica de SSO de CAS (ver **figura 4.3**).

1. El cliente usuario final **A** solicita el URI de una aplicación web. La aplicación web aprovecha un servidor CAS para proporcionar el servicio de autenticación. Bajo estos supuestos, el cliente trata de acceder a un recurso web específica en el servidor web, por ejemplo, http://web-app/about.me

2. Si la aplicación web comprueba la solicitud y no encuentra Ticket de acceso CAS junto con la solicitud, se redirigirá la petición a una URI del login de CAS. La URI de solicitud suele ser validado por un agente cliente CAS (por ejemplo, un filtro de Spring Security) que está desplegada en el mismo contexto que la aplicación web. El

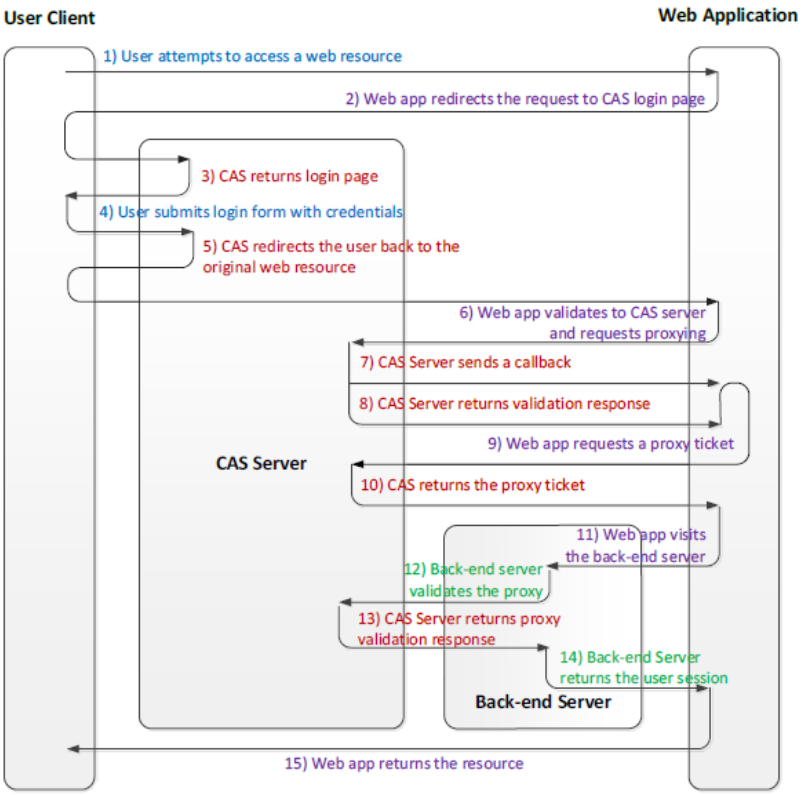


Figura 4.3: Flujo de la autenticación con CAS [28]

agente de cliente CAS redirige al usuario final a la solicitud de inicio de sesión en el servidor CAS. Esta redirección CAS también se produce cuando un ticket de la solicitud no es válido. También la URI de redirección debe contener un parámetro de "serviceçuyo valor es la URI original del recurso web. La URI sigue el siguiente patrón: http://cas-server/login?service=http://web-app/index

3. El servidor CAS retorna una página HTML que contiene un formulario de acceso.

4. El usuario cliente envía el ID de usuario y la contraseña para el servidor CAS

5. A continuación, el servidor CAS debe validar la credencial de usuario en un servidor de directorio de usuarios, que para nuestro caso, una base de datos MySQL y el servidor LDAP. El servidor CAS también debe comprobar si el servicio URI dentro del parámetro "service"(por ejemplo, http://web-app/index) ha sido registrado como un servicio CAS. Si no es así, la autenticación falla, incluso si la credencial de usuario es correcta. Después de la validación, el servidor CAS redirige a la URI del servicio original.

6. La aplicación web valida la petición. En concreto, se obtiene el parámetro "ticket"de la URI de la solicitud y lo valida con el servidor CAS. Igual al paso 2), esto se hace generalmente por un agente de cliente CAS en la aplicación web.

7. El servidor CAS devuelve la respuesta de éxito de la validación. Además, el servidor CAS puede devolver algunos atributos de usuario después de la validación para que la aplicación web puede hacer control de acceso para los recursos de la web basada en los atributos de usuario.

8. Aplicación web devuelve el recurso web para el cliente del usuario.

5 Implementación del prototipo

5.1. Consideraciones

Para la instalacón, configuración e implementación y extensión del prototipo SSO con CAS, se instalaron las siguientes herramientas de desarrollo de software en el equipo:

- Java Development Kit (JDK), que es un software que provee herramientas de desarrollo para la creación de programas en Java en el ordenador [22].
- Un contenedor de servlets que se utiliza en la Referencia oficial de la implementación para Java Servlet y Java Server Pages (JSP). En este caso se instaló **Apache Tomcat**[1].
- **Apache Maven**[2], una herramienta open source para administrar proyectos de software. Se utilizó para la administración y gestión de dependencia del proyecto y también para la instalación y la gestión de las dependencias de CAS Server.
- Mysql y MongoDB como gestores de base de datos para el prototipo. En MySQL para la autenticación de usuarios por base de datos y MongoDB para el registro de servicios o aplicaciones que se desean integrar a CAS, llamados CAS Client, según su arquitectura.
- Eclipse[3] como entorno de desarrollo integrado para el desarrollo y configuración del proyecto.

Se utilizó también el servidor LDAP de la UTB para la autenticación. Ya que en este servicio de directorio están las credenciales de los estudiantes de la Universidad.

5.2. Centro de autenticación del servidor CAS

5.2.1. Ejecución.

Para comenzar con la instalación y ejecución del servidor CAS y alguno de sus componentes, se utilizó el método recomendado ***Maven WAR Overlay*** para organizar las personalizaciones, como la configuración de los componentes y el diseño de interfaz de usuario. Este

[1] http://tomcat.apache.org/
[2] https://maven.apache.org/
[3] https://eclipse.org/

metodo Overlay se utiliza para compartir recursos comunes en varias aplicaciones web [23]. Para ello se abrió Eclipse y se creó un nuevo proyecto Maven en: *file, new project, Maven Project*. Al momento de crear el proyecto Maven, se seleccionó la casilla *Çreate a simple Project (skip archetype selection)"* (Ver **figura 5.1**). Esto es para crear el proyecto principal en Maven conocido como proyecto POM (Project Object Model), es la unidad fundamental de trabajo en Maven; es un archivo XML que contiene información sobre los detalles del proyecto y de configuración utilizados por Maven para construir el proyecto [24].

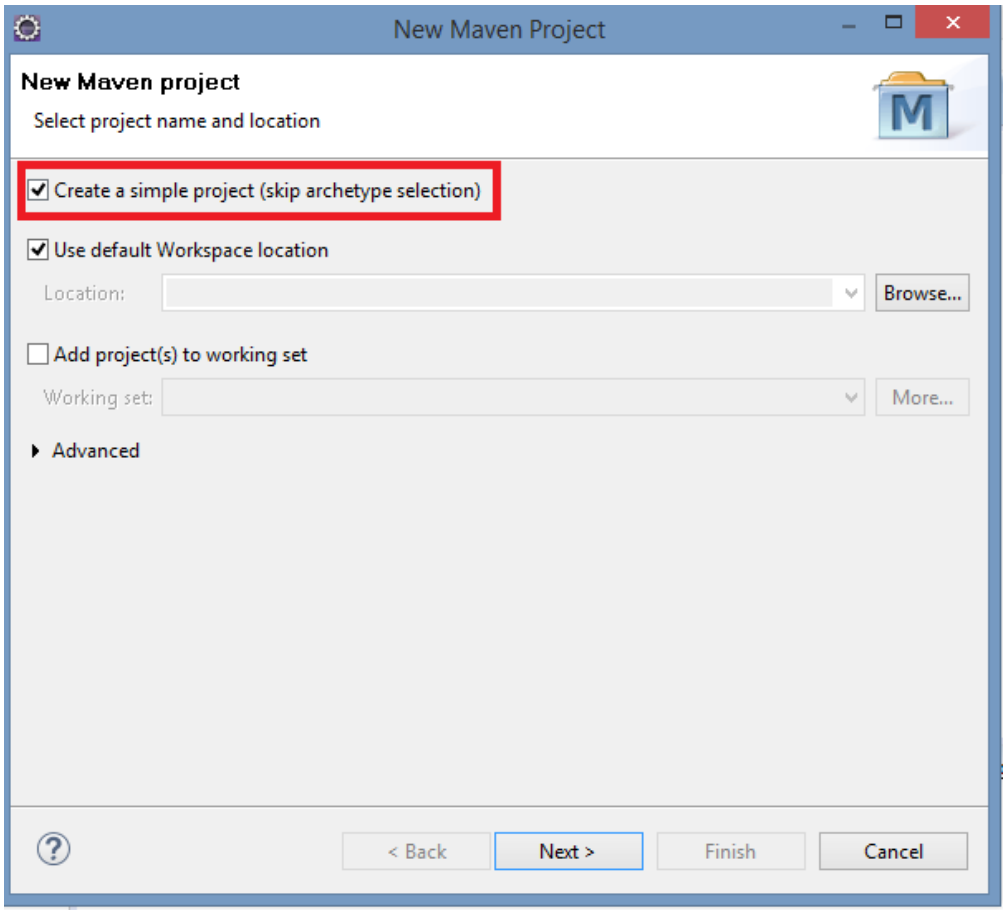


Figura 5.1: Creación del proyecto Maven en Eclipse

Posteriormente, se colocó cada uno de los datos de configuración del proyecto POM (ver **figura 5.2**), como **artefact id** (nombre del proyecto), **group id** (El paquete del proyecto), descripción, etc. Luego se oprimió clic en finalizar y se creó el proyecto con un archivo **pom.xml**, archivo que gestionó las dependencias y los módulos del proyecto.

Entonces se agregaron las dependencias y configuraciones en el archivo *pom.xml* del proyecto y Maven descargó las dependencias al proyecto.

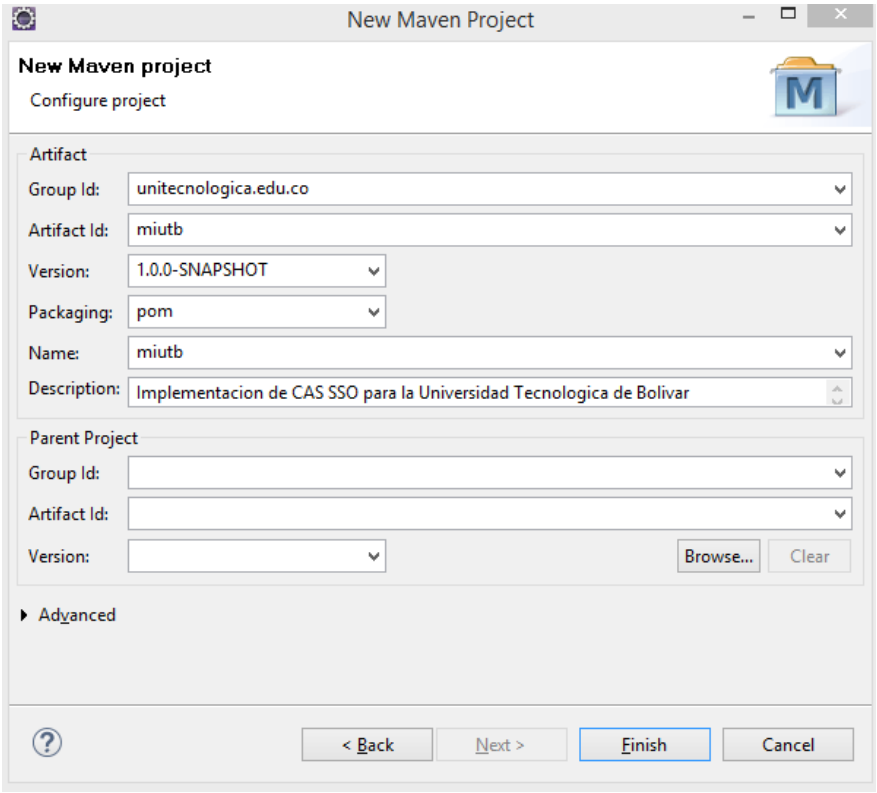


Figura 5.2: Configuración del proyecto Maven.

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jasig.cas</groupId>
                <artifactId>cas-server-webapp</artifactId>
                <version>${cas.version}</version>
                <type>war</type>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.jasig.cas</groupId>
                <artifactId>cas-management-webapp</artifactId>
                <version>${cas.version}</version>
                <type>war</type>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.jasig.cas</groupId>
                <artifactId>cas-server-core</artifactId>
                <type>test-jar</type>
                <version>${cas.version}</version>
            </dependency>
```

```
            <dependency>
                <groupId>org.jasig.cas</groupId>
                <artifactId>cas-server-integration-mongo</artifactId>
                <version>${cas.version}</version>
            </dependency>
            <dependency>
                <groupId>org.jasig.cas</groupId>
                <artifactId>cas-server-support-jdbc</artifactId>
                <version>${cas.version}</version>
            </dependency>
            <dependency>
                <groupId>org.jasig.cas</groupId>
                <artifactId>cas-server-support-ldap</artifactId>
                <version>${cas.version}</version>
            </dependency>
        </dependencies>

    </dependencyManagement>
    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.3</version>
                    <configuration>
                        <source>1.7</source>
                        <target>1.7</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-eclipse-plugin</artifactId>
                    <version>2.10</version>
                    <configuration>
                        <wtpversion>3.6</wtpversion>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

    <properties>
        <cas.version>4.2.5</cas.version>
    </properties>

    <!-- Repositorios -->
    <repositories>
```

```
        <repository>
            <id>sonatype-releases</id>
            <url>http://oss.sonatype.org/content/repositories/releases/</url>
        </repository>
        <repository>
            <id>sonatype-snapshots</id>
            <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
        </repository>
        <repository>
            <id>shibboleth-releases</id>
            <url>https://build.shibboleth.net/nexus/content/repositories/releases</url>
        </repository>
    </repositories>
```

Despues de esto, se agregaron 3 módulos maven del proyecto. Para agregar un módulo al proyecto nos vamos a *File, New Project, Maven Module*, teniendo en cuenta que en el campo *"parent"* aparezca el nombre del proyecto padre que se creó anteriormente (Ver **figure 5.3**).

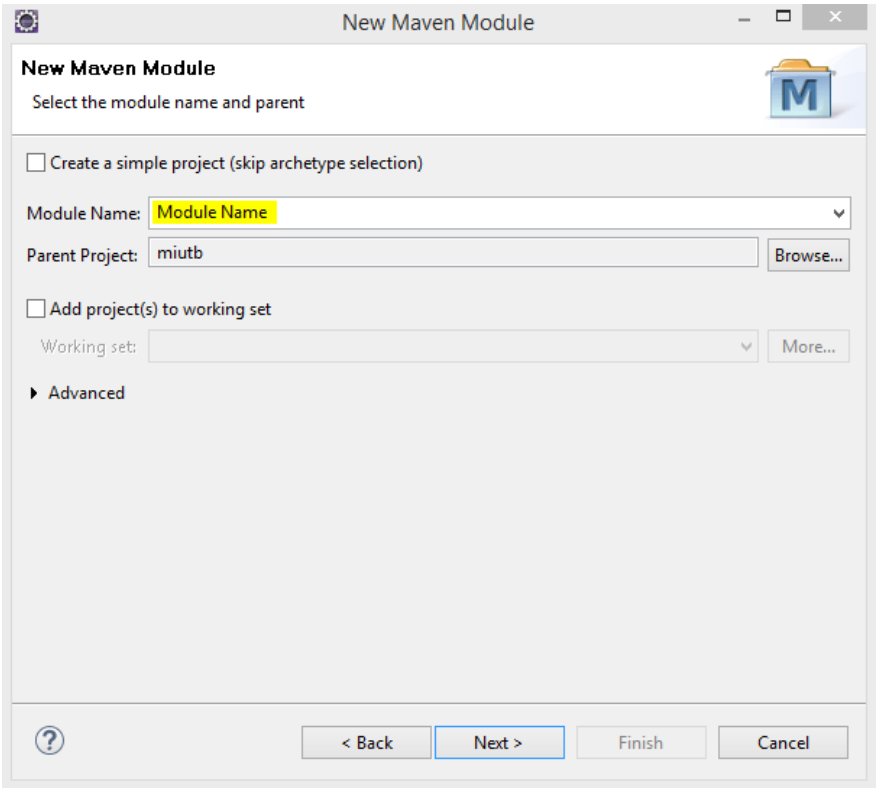


Figura 5.3: Creación de un Módulo Maven al proyecto.

Se creó los módulos y cada uno de ellos creó un archivo *pom.xml* para la adición y configuración de las dependencias. Los 3 módulos Maven que se crearon fueron:

- **Módulo *cas-miutb-mongo-integration:*** Es el módulo de tipo JAR, es decir una librería que configuró la conexión a una base de datos de MongoDB para el registro de servicios. Este módulo fue usado para la gestión del registro y de acceso de las aplicaciones al integrarse en nuestro SSO.

  Para la creación de este modelo, se seleccionó la casilla *"Create a simple Project (skip archetype selection)"* y luego proporcionó los datos de configuración del módulo, teniendo en cuenta que el tipo de paquete de este módulo será JAR (ver **figura 5.4**).

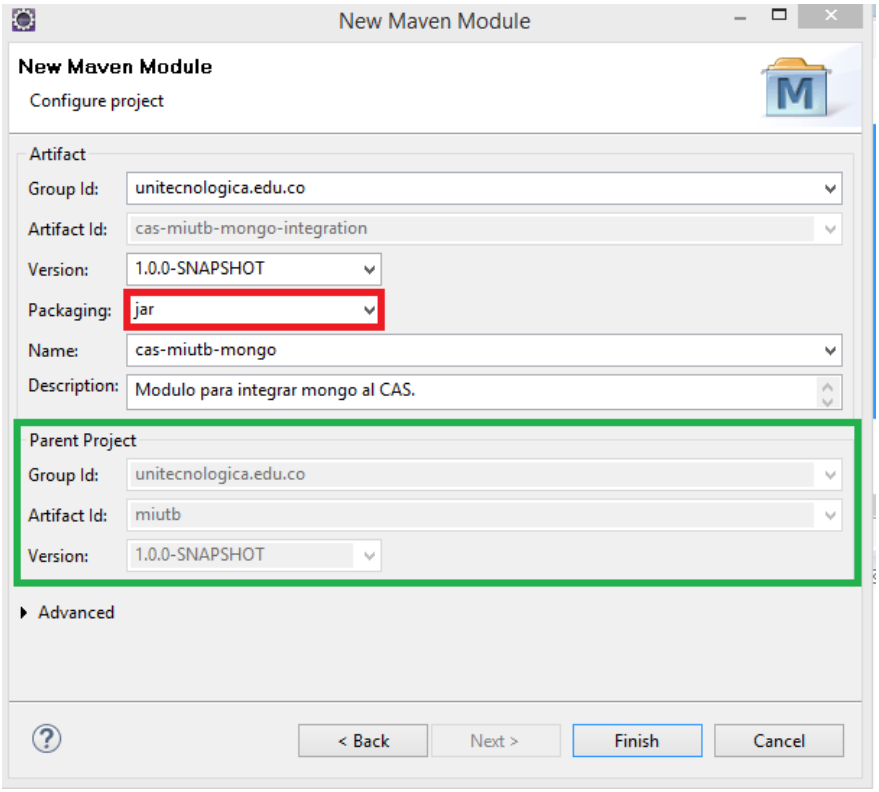


Figura 5.4: Configuración de un Módulo JAR

Por último, se agregó la siguiente dependencia en el archivo *pom.xml*.

```
<dependencies>
        <dependency>
            <groupId>org.jasig.cas</groupId>
            <artifactId>cas-server-integration-mongo</artifactId>
            <version>${cas.version}</version>
        </dependency>
    </dependencies>
```

- **Módulo *cas*:** Es el módulo tipo web donde se implementó el CAS Server. Al ser un módulo de tipo web, se seleccionó el tipo arquetipo *maven-archetype-webapp* (ver **figura 5.5**) para que maven creará una estructura por defecto de un proyecto web.

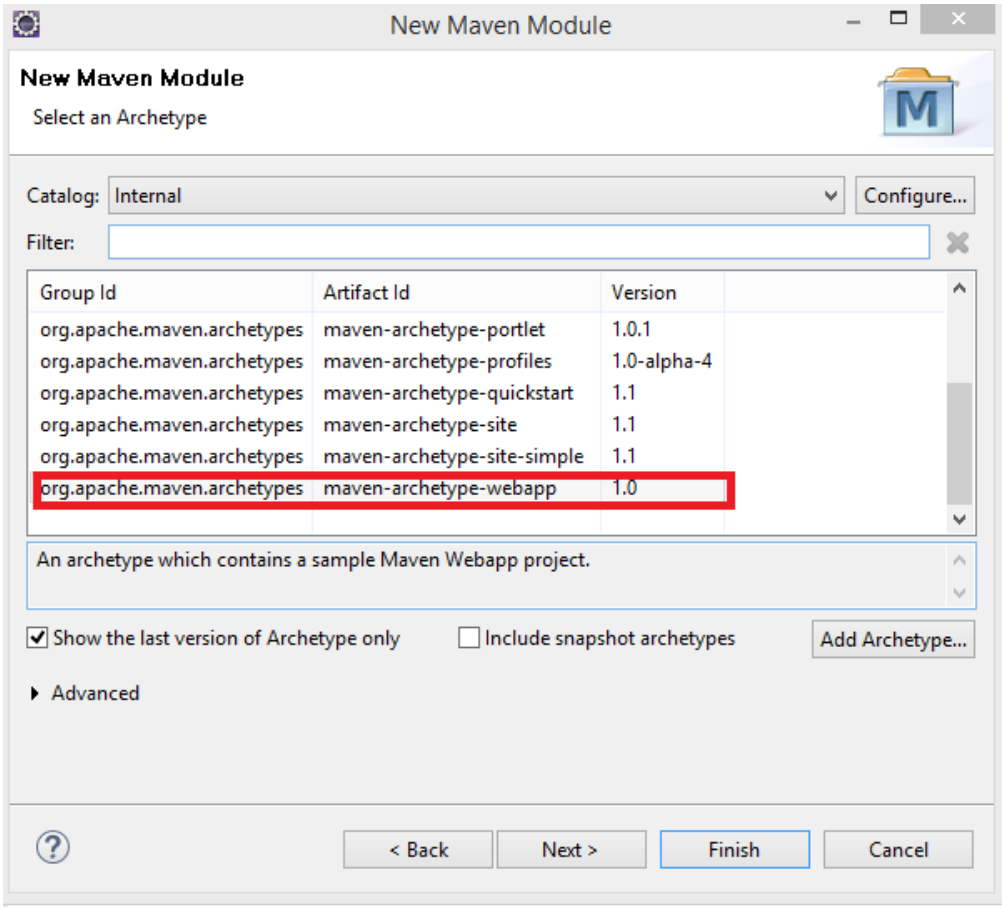


Figura 5.5: Creación de un Módulo de tipo web.

Luego al crear el módulo, se agregó las dependencias necesaria en el archivo pom.xml de este módulo:

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.jasig.cas</groupId>
            <artifactId>cas-server-webapp</artifactId>
            <version>${cas.version}</version>
            <type>war</type>
            <scope>runtime</scope>
        </dependency>
        <dependency>
```

```
            <groupId>unitecnologica.edu.co</groupId>
            <artifactId>cas-miutb-mongo-integration</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.jasig.cas</groupId>
            <artifactId>cas-server-support-jdbc</artifactId>
            <version>${cas.version}</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.38</version>
        </dependency>
        <dependency>
            <groupId>org.jasig.cas</groupId>
            <artifactId>cas-server-support-ldap</artifactId>
            <version>${cas.version}</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>cas</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.6</version>
                <configuration>\textit{\textbf{}}
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

Al finalizar la creación del módulo, se eliminó el archivo ***index.jsp*** y ***web.xml*** para que al momento de que maven descargue las dependencias y construya lo indicado en el pom.xml, obtendremos en una carpeta **target** los archivos por defecto de CAS que se necesitaron para su configuración.

- **Módulo *cas-services*:** El módulo de tipo web donde se implementó y se configuró una aplicación web que provee CAS llamado ***Services Management Webapp***[4]. La instalación del Services Management de CAS permite a los administradores del servidor CAS para declarar y configurar los servicios o aplicaciones que pueden hacer uso de CAS.

[4]El *services Management Webapp* ya no es parte del servidor CAS y es una aplicación web independiente y se utiliza para añadir, editar, borrar todos los servicios CAS

Al ser un módulo de tipo web, se seleccionó el tipo arquetipo *maven-archetype-webapp* (como el módulo anterior) para que Maven creará una estructura de proyecto web, y por las mismas razones como en el módulo anterior, se eliminó los archivos *index.jsp* y *web.xml*.. Se agregó las dependencias necesaria en el archivo pom.xml de este módulo de la siguiente forma:

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.jasig.cas</groupId>
            <artifactId>cas-management-webapp</artifactId>
            <type>war</type>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>unitecnologica.edu.co</groupId>
            <artifactId>cas-miutb-mongo-integration</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>cas-services</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.6</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

Al finalizar y comprobar que todo funcionaba correctamente, se ubicó en el directorio donde se encontraba el proyecto *miutb* en la línea de comando del equipo y se usó el siguiente comando:

```
mvn clean package
```

Al ejecutarse este comando, maven va al archivo pom.xml del proyecto y descargara todas las dependencias tanto en el proyecto padre como en cada uno de sus módulos y se obtuvó el siguiente resultado (ver **figura 5.6**).

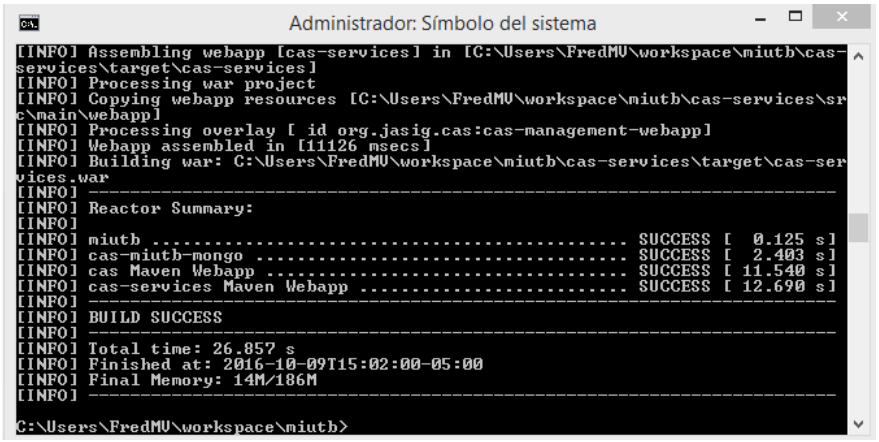


Figura 5.6: Resultados de la instalación y ejecución de CAS

Y pues como podemos ver todo ha salido perfectamente. Si ubicamos en la carpeta de cada uno de los módulos vamos a tener un directorio llamado **target** y vamos a encontrar un archivo .war para los módulos de tipo web y JAR para el módulo de tipo JAR (Estos archivos son los que vamos a desplegar en el servidor Tomcat para correr la aplicación).

5.2.2. Configuración.

Configuración del Módulo cas:

Primero que todo, se tuvo que copiar algunos archivos de configuración a nuestro modulo, para eso se copiaron los archivos **cas.properties** y **deployerConfigContext.xml** original de la carpeta generada por maven llamada **target**, es decir, se copiaron los archivos mencionados anteriormente de la carpeta *\miutb\cas\target\cas\WEB-INF* a la carpeta *\miutb\cas\src\main\webapp\WEB-INF* (ver **figura 5.7**).

¿Por qué esto? porque primeramente cada vez que se realizaba algún cambio o se volvía a construir la aplicación con *mvn clean package*, Maven borraba los archivos de la carpeta *target* y los actualizaba. Debido a esto, se tuvo que copiar los archivos al directorio del proyecto utilizando la misma estructura. **¿Por qué esos archivos?** Porque esos archivos son los que más se utilizaron para realizar cada una de las siguientes configuraciones:

1. **Configuración General:**

   Primero se configuró la URL del sistema SSO, en este caso la dirección URL que es `https://labsoftware04.unitecnologica.edu.co:8443/`, donde se desplegó el servidor Tomcat. Por eso hubo que decirle a CAS cual es nuestra URL donde va a estar desplegado el proyecto y la localización en el archivo *cas.properties*:

   ```
   server.name=https:// labsoftware04.unitecnologica.edu.co:8443
   ```

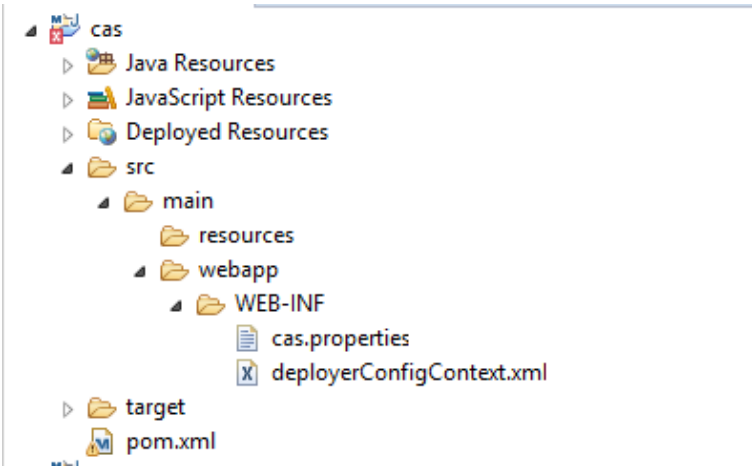


Figura 5.7: Árbol de archivos del módulo cas

```
server.prefix=${server.name}/cas

##
# CAS Internationalization
#
locale.default=es
# locale.param.name=locale
message.bundle.encoding=UTF-8
```

2. **Configuración para el registro de servicio:**

   Para esto se utilizó el módulo *cas-miutb-mongo-integration*. Se tuvó que hacer lo siguiente en cada archivo:

   - En el archivo cas.properties: En este archivo se suministró los datos de conexión de mongo:

     ```
     mongodb.host=localhost
     mongodb.port=27017
     mongodb.userId=ssouser
     mongodb.userPassword=123qwe
     cas.service.registry.mongo.db=admin
     mongodb.timeout=5000
     ```

   - En el archivo deployerConfigContext.xml: se realizaron 2 cosas: (1) se comentó la siguiente línea de código:

     ```
     <!--<alias name="jsonServiceRegistryDao" alias="serviceRegistryDao" /> -->
     ```

     Y se agregó el siguiente código:

     ```
     <alias name="mongoServiceRegistryDao" alias="serviceRegistryDao" />
     ```

3. **Conexión con JDBC (MySQL) y LDAP:**

   Para la conexión con LDAP se colocó los siguientes datos en el archivo cas.properties:

```
##LDAP
#=======================================
# General properties
#=======================================
ldap.url=ldap://23.253.34.120:389

# Start TLS for SSL connections
ldap.useStartTLS=false

# Directory root DN
ldap.rootDn=dc=utbvirtual,dc=edu,dc=co

# Base DN of users to be authenticated
ldap.baseDn=ou=all,ou=People,dc=utbvirtual,dc=edu,dc=co

# LDAP connection timeout in milliseconds
ldap.connectTimeout=3000

# Manager credential DN
ldap.managerDn=ou=all,ou=People,dc=utbvirtual,dc=edu,dc=co

# Manager credential password
ldap.managerPassword=nonsense

#=======================================
# LDAP connection pool configuration
#=======================================
ldap.pool.minSize=3
ldap.pool.maxSize=10
ldap.pool.validateOnCheckout=false
ldap.pool.validatePeriodically=true

# Amount of time in milliseconds to block on pool exhausted condition
# before giving up.
ldap.pool.blockWaitTime=3000

# Frequency of connection validation in seconds
# Only applies if validatePeriodically=true
ldap.pool.validatePeriod=300

# Attempt to prune connections every N seconds
ldap.pool.prunePeriod=300

# Maximum amount of time an idle connection is allowed to be in
# pool before it is liable to be removed/destroyed
```

```
ldap.pool.idleTime=600

#=======================================
# Authentication
#=======================================
ldap.authn.searchFilter=(uid={user})

# Ldap domain used to resolve dn
ldap.domain=utbvirtual.edu.co

# Should LDAP Password Policy be enabled?
ldap.usePpolicy=false

# Allow multiple DNs during authentication?
ldap.allowMultipleDns=false
```

Y en el archivo deployerConfigContext.xml se agregó lo siguiente:

```
  <!-- JDBC Authentication -->
      <!-- Contruccion de los beans Manualmente -->

      <!-- Encoding Password JDBC -->
      <bean id="passwordEncoderJDBC"
          class="org.jasig.cas.authentication.handler.DefaultPasswordEncoder"
          c:encodingAlgorithm="MD5" p:characterEncoding="UTF-8" />

      <!-- Contruccion del bean AuthenticationHandler JDBC MySQL -->
      <bean id="secondAuthenticationHandler"
          class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler"
          p:dataSource-ref="dataSource" p:passwordEncoder-ref="passwordEncoderJDBC"
          p:sql="select password from test_sso.user where username=? " />


      <bean id="dataSource"
          class="com.mchange.v2.c3p0.ComboPooledDataSource"
          p:driverClass="${database.driverClass}"
          p:jdbcUrl="${database.url}"
          p:user="${database.user}"
          p:password="${database.password}"
          p:initialPoolSize="${database.pool.minSize}"
          p:minPoolSize="${database.pool.minSize}"
          p:maxPoolSize="${database.pool.maxSize}"
          p:maxIdleTimeExcessConnections="${database.pool.maxIdleTime}"
          p:checkoutTimeout="${database.pool.maxWait}"
          p:acquireIncrement="${database.pool.acquireIncrement}"
          p:acquireRetryAttempts="${database.pool.acquireRetryAttempts}"
          p:acquireRetryDelay="${database.pool.acquireRetryDelay}"
```

```
        p:idleConnectionTestPeriod="${database.pool.idleConnectionTestPeriod}"
        p:preferredTestQuery="${database.pool.connectionHealthQuery}" />

    <!-- LDAP Authentication Configuration -->
    <bean id="ldapAuthenticationHandler"
          class="org.jasig.cas.authentication.LdapAuthenticationHandler"
          p:principalIdAttribute="uid"
          c:authenticator-ref="authenticator">
          <property name="principalAttributeList">
              <list>
                  <!-- <value>displayName</value>  -->
              </list>
          </property>
    </bean>

    <ldaptive:anonymous-search-authenticator
        id="authenticator"
        ldapUrl="${ldap.url}"
        connectTimeout="${ldap.connectTimeout}"
        validateOnCheckOut="${ldap.pool.validateOnCheckout}"
        failFastInitialize="true"
        blockWaitTime="${ldap.pool.blockWaitTime}"
        idleTime="${ldap.pool.idleTime}"
        maxPoolSize="${ldap.pool.maxSize}"
        minPoolSize="${ldap.pool.minSize}"
        validatePeriodically="${ldap.pool.validatePeriodically}"
        validatePeriod="${ldap.pool.validatePeriod}"
        prunePeriod="${ldap.pool.prunePeriod}"
        useSSL="${ldap.use.ssl:false}"
        useStartTLS="${ldap.useStartTLS}"
        allowMultipleDns="${ldap.allowMultipleDns:false}"
        baseDn="${ldap.baseDn}"
        subtreeSearch="${ldap.subtree.search:true}"
        userFilter="${ldap.authn.searchFilter}" />
```

Y también se añadió los controladores de autenticación:

```
  <util:map id="authenticationHandlersResolvers">
        <entry key-ref="proxyAuthenticationHandler" value-ref="proxyPrincipalResolver" />
        <entry key-ref="primaryAuthenticationHandler" value-ref="primaryPrincipalResolver" />
        <entry key-ref="secondAuthenticationHandler" value="#{null}" />
        <entry key-ref="ldapAuthenticationHandler" value="#{null}" />
    </util:map>
```

Y por último se agregó el esquema de LDAP en el beans:

```
xmlns:ldaptive=http://www.ldaptive.org/schema/spring-ext
```

En la sección *xsi:schemaLocation* se agregó:

```
http://www.ldaptive.org/schema/spring-ext
http://www.ldaptive.org/schema/spring-ext.xsd
```

4. **Configuración de Monitoreo:**

   Para permitir que el usuario administrador pueda ver la configuración del sistema, las estadísticas y las sesiones activas, los tickets de sesión, etc. Se tuvó que colocar la IP del servidor en el archivo *cas.properties*:

   ```
   # security configuration based on IP address to access the /status and /statistics pages
   cas.securityContext.adminpages.ip=127\.0\.0\.1|0:0:0:0:0:0:0:1
   ```

   Cabe recalcar que estas vistas de configuración son para usarlas dentro del equipo servidor, para no exponer la información de la configuración del sistema SSO. Pero también existe la posibilidad de que se pueda acceder desde cualquier equipo colocando en vez de una IP, colocamos lo siguiente **(pero por seguridad no se debe realizar de esta forma)**:

   ```
   # security configuration based on IP address to access the /status and /statistics pages
   cas.securityContext.adminpages.ip=.+
   ```

Configuración del Módulo cas-services:

De igual forma que en la configuración del módulo cas, se copiaron algunos archivos de configuración a nuestro módulo. Los archivos fueron: *cas-management-servlet.xml, managementConfigContext.xml, cas-management.properties.* Estos archivos se ubicaron en la carpeta *\miutb\cas-services\src\main\webapp\WEB-INF*. También se realizó una copia del archivo *user-details.properties* ubicado en *\miutb\cas-services\target\cas-services\WEB-INF\classes*, y se copió en la carpeta *\miutb\cas-services\src\resource* (ver **figura 5.8**).

1. **Configuración de Acceso y Autorización:**

   El acceso a la aplicación web de gestión es controlada a través de **Spring-Security**[5]. Para configurar que usuario va usar el Services management se añadió en el archivo **user-details.properties** lo siguiente (para este caso el usuario es **admin**):

   ```
   # Example:
   #casuser=notused,ROLE_ADMIN
   admin=admin,ROLE_ADMIN
   ```

2. **Configuración del almacenamiento de los Servicios:**

   El almacenamiento de persistencia debe ser la misma configuración que ya se utiliza en su servidor CAS en el archivo *deployerConfigContext.xml*. Se utilizó MongoDB, por eso se realizó la siguiente configuración:

[5] http://projects.spring.io/spring-security/

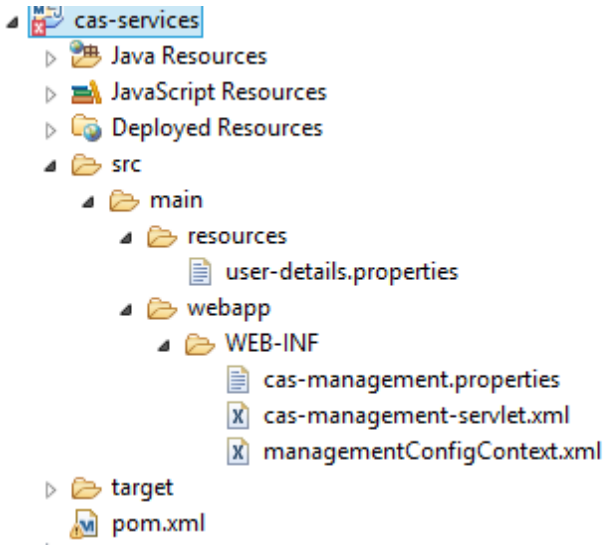


Figura 5.8: Árbol de archivos del módulo cas-services

- En el archivo cas.properties: se facilitó los datos de conexión de Mongo:

```
mongodb.host=localhost
mongodb.port=27017
mongodb.userId=ssouser
mongodb.userPassword=123qwe
cas.service.registry.mongo.db=admin
mongodb.timeout=5000
```

- En el archivo deployerConfigContext.xml: se realizaron 2 cosas: se comentó la siguiente línea de código:

```
<!--<alias name="jsonServiceRegistryDao" alias="serviceRegistryDao" /> -->
```

Y se agregó el siguiente código:

```
<alias name="mongoServiceRegistryDao" alias="serviceRegistryDao" />
```

3. **Configuración de URLS:**

La configuración de la aplicación cas-services se realizó agregando las direcciones URL donde fue desplegado el archivo .war en el archivo cas-management.properties como se muestra a continuación:

```
# CAS
cas.host=https://labsoftware04.unitecnologica.edu.co:8443
cas.prefix=${cas.host}/cas-miutb-webapp
cas.securityContext.casProcessingFilterEntryPoint.loginUrl=${cas.prefix}/login

# Management
```

```
cas-management.host=https://labsoftware04.unitecnologica.edu.co:8443
cas-management.prefix=${cas-management.host}/cas-miutb-management
cas-management.securityContext.serviceProperties.service=
${cas-management.prefix}/callback
```

Después de todo este proceso de configuración, en la línea de comando del ordenador, se utilizó nuevamente el comando: *mvn clean package.*

Es necesario mencionar que cuando se autentica en un servidor CAS, la aplicación Services Management será procesado como un servicio regular de CAS y, por tanto, debe definirse en el registro de los servicios del servidor CAS. Por tal motivo, se registró el servicio en nuestra base de datos de MongoDB añadiendo el siguiente documento en la colección **RegisteredService**[6] (ver la **figura 5.9**):

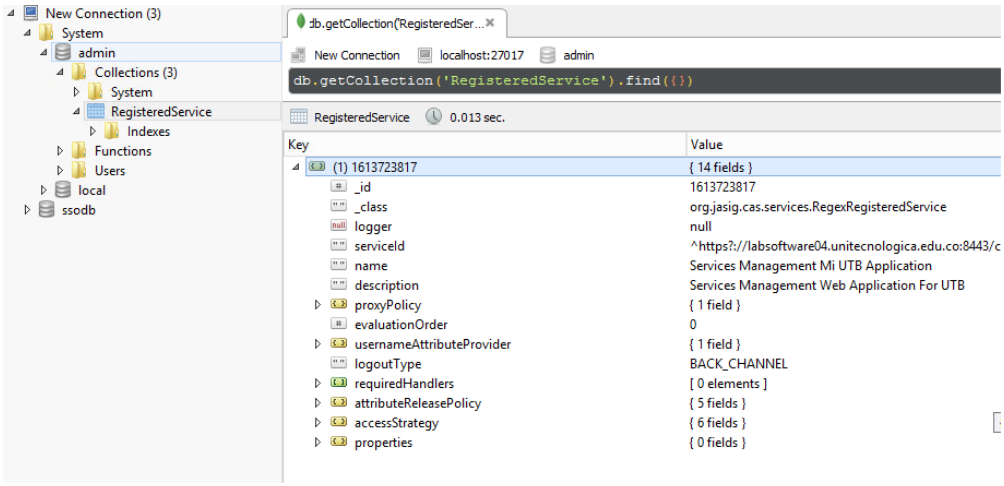


Figura 5.9: Registro del cliente CAS Services Management en MongoDB

```
{
    "_id" : NumberLong(1613723817),
    "_class" : "org.jasig.cas.services.RegexRegisteredService",
    "logger" : null,
    "serviceId" : "^https?://labsoftware04.unitecnologica.edu.co:8443/cas-services/.*",
    "name" : "Services Management Mi UTB Application",
    "description" : "Services Management Web Application For UTB",
    "proxyPolicy" : {
        "_class" : "org.jasig.cas.services.RefuseRegisteredServiceProxyPolicy"
    },
    "evaluationOrder" : 0,
    "usernameAttributeProvider" : {
        "_class" : "org.jasig.cas.services.DefaultRegisteredServiceUsernameProvider"
    },
```

[6]La colección RegisteredService"de mongo, es creada al momento de subir los archivos y desplegar las 2 aplicaciones web: *cas y cas-services*

```
    "logoutType" : "BACK_CHANNEL",
    "requiredHandlers" : [],
    "attributeReleasePolicy" : {
        "_class" : "org.jasig.cas.services.ReturnAllowedAttributeReleasePolicy",
        "allowedAttributes" : [],
        "principalAttributesRepository" : {
            "_class" : "org.jasig.cas.authentication.principal.DefaultPrincipalAttributesRepository",
            "expiration" : NumberLong(2),
            "timeUnit" : "HOURS"
        },
        "authorizedToReleaseCredentialPassword" : false,
        "authorizedToReleaseProxyGrantingTicket" : false
    },
    "accessStrategy" : {
        "_class" : "org.jasig.cas.services.TimeBasedRegisteredServiceAccessStrategy",
        "enabled" : true,
        "ssoEnabled" : true,
        "requireAllAttributes" : true,
        "requiredAttributes" : {},
        "caseInsensitive" : false
    },
    "properties" : {}
}
```


5.3. Configuración de los clientes CAS

Es necesario recordar que el termino de CAS Client, es cualquier aplicación habilitada para el CAS que pueden comunicarse con el servidor mediante un protocolo soportado (Ej: CAS, SAML, OAtuh). En este proyecto usaremos un Moodle, Wordpress y aplicaciones en los lenguajes Java, PHP, ASP .Net como clientes.

Configuración de Moodle

CAS funciona básicamente mediante la configuración de un sitio Moodle para no realizar la autenticación en sí, sino a los usuarios no autenticados en lugar hacia adelante a un servidor CAS que a su vez devolverá un token de autenticación en el sitio Moodle. Moodle puede extraer el nombre de usuario de la ficha y luego utilizar sus mecanismos internos de autorización (roles, los registros capacidades) y los atributos de usuario (nombre, imagen, etc.). La ventaja es que el sitio Moodle no tiene que manejar y contraseñas que los usuarios, una vez que han autenticados primera vez, pueden pasar sin problemas a otra aplicación web sin tener que presentar sus credenciales de nuevo [25].

1. Como administrador de Moodle, se habilitó la autenticación de CAS Server de la siguiente manera: fuimos a *Administración del sitio, Plugins, Autenticación, Gestionar autenticación* y se hizo clic en el icono del ojo opuesto de usar un servidor CAS (SSO) (Figura **5.10**).

2. Se hizo clic en el enlace configuración, se configuró lo necesario, posteriormente, se hizo clic en el botón "Guardar cambios"(Figure **5.11**).

Figura 5.10: Habilitación de CAS Server en Moodle

Figura 5.11: Configuración de CAS en Moodle

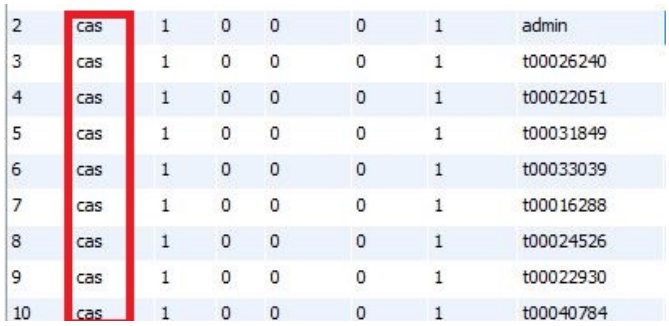
3. Se cambió el método de autenticación de los usuarios en la tabla **mdl_user** de la base de datos de Moodle, cambiando el valor de manual o LDAP, a CAS(Figura **5.12**) .

4. Por último, para que Moodle tenga los permisos de usar el servidor SSO, se registró la aplicación como un cliente por medio de la aplicación web Services Management, que en nuestro caso es el modulo *cas-services*.

Configuración de una Aplicación Java

A continuación se muestran los pasos a pasos realizados para configurar la integración de una aplicación web de Java (JSP) con nuestro servidor SSO:

1. Se añadió la siguiente dependencia en el archivo pom.xml de la aplicación de Java:



2	cas	1	0	0	0	1	admin
3	cas	1	0	0	0	1	t00026240
4	cas	1	0	0	0	1	t00022051
5	cas	1	0	0	0	1	t00031849
6	cas	1	0	0	0	1	t00033039
7	cas	1	0	0	0	1	t00016288
8	cas	1	0	0	0	1	t00024526
9	cas	1	0	0	0	1	t00022930
10	cas	1	0	0	0	1	t00040784

Figura 5.12: Modificación de la tabla de usuarios en Moodle

```
<dependency>
    <groupId>org.jasig.cas</groupId>
    <artifactId>cas-client-core</artifactId>
    <version>3.1-RC1</version>
</dependency>
```

2. Se añadió el filtro de autenticación en el archivo web.xml:

```
<filter>
  <filter-name>CAS Authentication Filter</filter-name>
  <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>
  <init-param>
    <param-name>casServerLoginUrl</param-name>
    <param-value>https://labsoftware04.unitecnologica.edu.co:8443/cas/login</param-value>
  </init-param>
  <init-param>
    <param-name>serverName</param-name>
    <param-value>https://labsoftware04.unitecnologica.edu.co:8443/</param-value>
  </init-param>
</filter>
    <filter-mapping>
    <filter-name>CAS Authentication Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Se añadió el filtro de validación en el archivo web.xml:

```
<filter>
  <filter-name>CAS Validation Filter</filter-name>
  <filter-class>
       org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter
  </filter-class>
  <init-param>
    <param-name>casServerUrlPrefix</param-name>
```

```
    <param-value>https://labsoftware04.unitecnologica.edu.co:8443/cas</param-value>
  </init-param>
  <init-param>
    <param-name>serverName</param-name>
    <param-value>https://labsoftware04.unitecnologica.edu.co:8443/cas-client/</param-value>
  </init-param>
  <init-param>
    <param-name>redirectAfterValidation</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
    <filter-mapping>
    <filter-name>CAS Validation Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

4. Por último, para que esta aplicación tenga los permisos de usar el servidor SSO, se registró como un cliente por medio de la aplicación web Services Management, que en este trabajo es el modulo *cas-services*.

Configuración de una Aplicación ASP .NET

El cliente CAS de .NET proporciona integración para la plataforma Microsoft Windows a través de la plataforma .NET. A continuación se muestran los pasos a pasos realizados para configurar la integración de una aplicación web de ASP .NET (aspx) con el servidor SSO:

1. **Instalación:** Se descargó la librería del CAS Client para .NET por medio del manejador de paquetes que permite instalar y actualizar librerías y herramientas en *Visual Studio*[7], llamado **NuGet**[8]. Paquetes NuGet para el cliente .NET están disponibles en `https://www.nuget.org/packages/DotNetCasClient`

2. **Configuración:** se añadió en el archivo web.config de la aplicación .NET lo siguiente:

   a) Se definió la sección de configuración del *casClientConfig*:

   ```
   <configSections>
       <section name="casClientConfig"
                type="DotNetCasClient.Configuration.CasClientConfiguration, DotNetCasClient"/>
       <section name="log4net"
                type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"/>
     </configSections>
   ```

   b) Se añadió un elemento de configuración <*casClientConfig*> directamente bajo la raíz el elemento <*configuration*>. La posición del elemento <*casClientConfig*> en el archivo web.config es intrascendente.

[7] https://www.visualstudio.com/
[8] https://www.nuget.org/

```
<casClientConfig
      casServerLoginUrl="https://labsoftware04.unitecnologica.edu.co:8443/cas/login"
      casServerUrlPrefix="https://labsoftware04.unitecnologica.edu.co:8443/cas/"
      serverName="http://labsoftware04.unitecnologica.edu.co/webappasp"
      notAuthorizedUrl="~/NotAuthorized.aspx"
      cookiesRequiredUrl="~/CookiesRequired.aspx"
      redirectAfterValidation="true"
      gateway="false" renew="false"
      singleSignOut="true"
      ticketTimeTolerance="5000"
      ticketValidatorName="Cas10"
      proxyTicketManager="CacheProxyTicketManager"
      serviceTicketManager="CacheServiceTicketManager"
      gatewayStatusCookieName="CasGatewayStatus"/>
```

c) Se agregó la configuración de donde se guardaría los logs del cliente .NET para depuración de la aplicación:

```
<log4net debug="true">
    <appender name="RollingFile"
              type="log4net.Appender.RollingFileAppender">
      <file value="C:\sampleCAS\dotnetcasclient.log"/>
      <appendToFile value="true"/>
      <maximumFileSize value="1000KB"/>
      <maxSizeRollBackups value="2"/>
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern
          value="%date [%thread] %-5level %logger %ndc - %message%newline"/>
      </layout>
    </appender>
    <root>
      <level value="TRACE"/>
      <appender-ref ref="RollingFile"/>
    </root>
  </log4net>
```

d) Se registró en la sección **httpModules** el registro del módulo CasAuthenticationModule y ademas se agregó la configuración de diagnostico y seguridad con el cliente CAS para .NET:

```
<system.webServer>
    <modules>
      <remove name="DotNetCasClient"/>
      <add name="DotNetCasClient"
           type="DotNetCasClient.CasAuthenticationModule,DotNetCasClient"/>
    </modules>
  </system.webServer>
  <system.diagnostics>
    <trace autoflush="true" useGlobalLock="false"/>
```

```
  <sharedListeners>
    <add name="TraceFile" type="System.Diagnostics.TextWriterTraceListener"
         initializeData="C:\sampleCAS\castrace.log"
         traceOutputOptions="DateTime"/>
  </sharedListeners>
  <sources>
    <!-- Provides diagnostic information on module configuration parameters. -->
    <source name="DotNetCasClient.Config"
            switchName="Config"
            switchType="System.Diagnostics.SourceSwitch">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>

    <source name="DotNetCasClient.HttpModule"
            switchName="HttpModule"
            switchType="System.Diagnostics.SourceSwitch">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>

    <source name="DotNetCasClient.Protocol"
            switchName="Protocol"
            switchType="System.Diagnostics.SourceSwitch">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>

    <source name="DotNetCasClient.Security"
            switchName="Security"
            switchType="System.Diagnostics.SourceSwitch">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>
  </sources>
  <switches>
    <add name="Config" value="Information"/>
    <add name="HttpModule" value="Verbose"/>
    <add name="Protocol" value="Information"/>
    <add name="Security" value="Information"/>
  </switches>
</system.diagnostics>
```

e) Por ultimo, se configuró la sección del formulario de autenticación y la autorización

<form>, de manera que apunte a la URL de inicio de sesión del servidor CAS definido en el atributo *casServerLoginUrl* de la sección *casClientConfig*. Es de vital importancia que la URL de inicio de sesión CAS es el mismo en ambos lugares:

```
<authentication mode="Forms">
      <forms
        loginUrl="https://labsoftware04.unitecnologica.edu.co:8443/cas/login"
        timeout="90" defaultUrl="~/Default.aspx"
        cookieless="UseCookies"
        slidingExpiration="true"/>
    </authentication>

<authorization>
      <allow users="*"/>
    </authorization>
    <pages
      controlRenderingCompatibilityVersion="3.5"
      clientIDMode="AutoID"/>
    <roleManager
      enabled="true"
      defaultProvider="AspNetReadOnlyXmlRoleProvider">
      <providers>
        <add name="AspNetReadOnlyXmlRoleProvider"
             type="DotNetCasClient.Security.ReadOnlyXmlRoleProvider"
             xmlFileName="~/App_Data/UserRoles.xml"/>
      </providers>
    </roleManager>
```

3. **Registro en el Services Management:** Para que esta aplicación tenga los permisos de usar el servidor SSO, también se registró como un cliente por medio de la aplicación web Services Management.

Configuración de una Aplicación PHP

A continuación se muestran los pasos a pasos realizados para configurar la integración de una aplicación de PHP con el servidor SSO:

1. Se Descargó el PHP cliente en el siguiente link: `https://developer.jasig.org/cas-clients/php/current/`, y se colocó la carpeta descargada en la carpeta principal de la aplicación web.

2. Se creó un archivo de configuración *config.php* para configurar la comunicación con el servidor SSO, el archivo de prueba fue el siguiente:

```
<?php
```

```
$phpcas_path = 'phpCAS/';

$cas_host = 'labsoftware04.unitecnologica.edu.co';
// Context of the CAS Server
$cas_context = '/cas';
// Port of your CAS server. Normally for a https server it's 443
$cas_port = 8443;
// Path to the ca chain that issued the cas server certificate
$cas_server_ca_cert_path = '/path/to/cachain.pem';

$cas_real_hosts = array('cas-real-1.example.com', 'cas-real-2.example.com');
// Client config for cookie hardening
$client_domain = '127.0.0.1';
$client_path = 'phpcas';
$client_secure = true;
$client_httpOnly = true;
$client_lifetime = 0;
// Database config for PGT Storage
$db = 'pgsql:host=localhost;dbname=phpcas';
//$db = 'mysql:host=localhost;dbname=phpcas';
$db_user = 'phpcasuser';
$db_password = 'mysupersecretpass';
$db_table = 'phpcastabel';
$driver_options = '';

// Generating the URLS for the local cas example services for proxy testing
if (isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] == 'on') {
    $curbase = 'https://' . $_SERVER['SERVER_NAME'];
} else {
    $curbase = 'http://' . $_SERVER['SERVER_NAME'];
}
if ($_SERVER['SERVER_PORT'] != 80 && $_SERVER['SERVER_PORT'] != 443) {
    $curbase .= ':' . $_SERVER['SERVER_PORT'];
}
$curdir = dirname($_SERVER['REQUEST_URI']) . "/";
// CAS client nodes for rebroadcasting pgtIou/pgtId and logoutRequest
$rebroadcast_node_1 = 'http://cas-client-1.example.com';
$rebroadcast_node_2 = 'http://cas-client-2.example.com';
// access to a single service
$serviceUrl = $curbase . $curdir . 'example_service.php';
// access to a second service
$serviceUrl2 = $curbase . $curdir . 'example_service_that_proxies.php';
$pgtBase = preg_quote(preg_replace('/^http:/', 'https:', $curbase . $curdir), '/');
$pgtUrlRegexp = '/^' . $pgtBase . '.*$/';
$cas_url = 'https://' . $cas_host;
if ($cas_port != '443') {
    $cas_url = $cas_url . ':' . $cas_port;
}
```

```
$cas_url = $cas_url . $cas_context;
// Set the session-name to be unique to the current script so that the client script
// doesn't share its session with a proxied script.
// This is just useful when running the example code, but not normally.
session_name(
    'session_for:'
    . preg_replace('/[^a-z0-9-]/i', '_', basename($_SERVER['SCRIPT_NAME']))
);
// Set an UTF-8 encoding header for internation characters (User attributes)
header('Content-Type: text/html; charset=utf-8');
?>
```

3. Para finalizar, se creó un archivo index.php, se utilizaron las funciones de CAS Client PHP para la autenticación con el servidor SSO, un ejemplo sencillo fue:

```
<?php
/**
 *   Example for a simple cas 2.0 client
 *
*/
// Load the settings from the central config file
require_once 'config.php';
// Load the CAS lib
require_once $phpcas_path . '/CAS.php';
// Enable debugging
phpCAS::setDebug();
// Enable verbose error messages. Disable in production!
//phpCAS::setVerbose(true);
// Initialize phpCAS
phpCAS::client(CAS_VERSION_2_0, $cas_host, $cas_port, $cas_context);
phpCAS::setNoCasServerValidation();
if (isset($_REQUEST['logout'])) {
    phpCAS::logout();
}
if (isset($_REQUEST['login'])) {
    phpCAS::forceAuthentication();
}
// check CAS authentication
$auth = phpCAS::checkAuthentication();
?>
<html>
  <head>
    <title>phpCAS simple client</title>
  </head>
  <body>
<?php
if ($auth) {
```

```
    // for this test, simply print that the authentication was successfull
  phpCAS::forceAuthentication();
        ?>
    <h1>Successfull Authentication!</h1>
    <p>the user's login is <b><?php echo phpCAS::getUser(); ?></b>.</p>
    <p><a href="?logout=">Logout</a></p><?php
} else {
                                        ?>
    <h1>Guest mode</h1>
    <p><a href="?login=">Login</a></p><?php
}
                                       ?>
    <p>phpCAS version is <b><?php echo phpCAS::getVersion(); ?></b>.</p>
  </body>
</html>
```

Configuración de una Aplicación Wordpress

Wordpress, un software de gestor de contenido (CMS) que puedes utilizar para crear fantásticas webs, blogs o aplicaciones, con una comunidad grande; Dado al crecimiento de su comunidad en los últimos tiempos se han creado muchos plugins para extender este sistema. Para la configuración de una aplicación de wordpress con el servidor SSO, se instaló un plugin llamado **CAS Maestro**[9]. En la cual se proporció los datos al momento de configurar la URL del servidor CAS, algunas otras configuraciones de la forma de autenticarse los usuarios de wordpress con el SSO. También se selecció la opción de registrarse los usuarios automáticamente si no existe en la base de datos de wordpress (ver **figura 5.13**).

[9]https://es.wordpress.org/plugins/cas-maestro/installation/

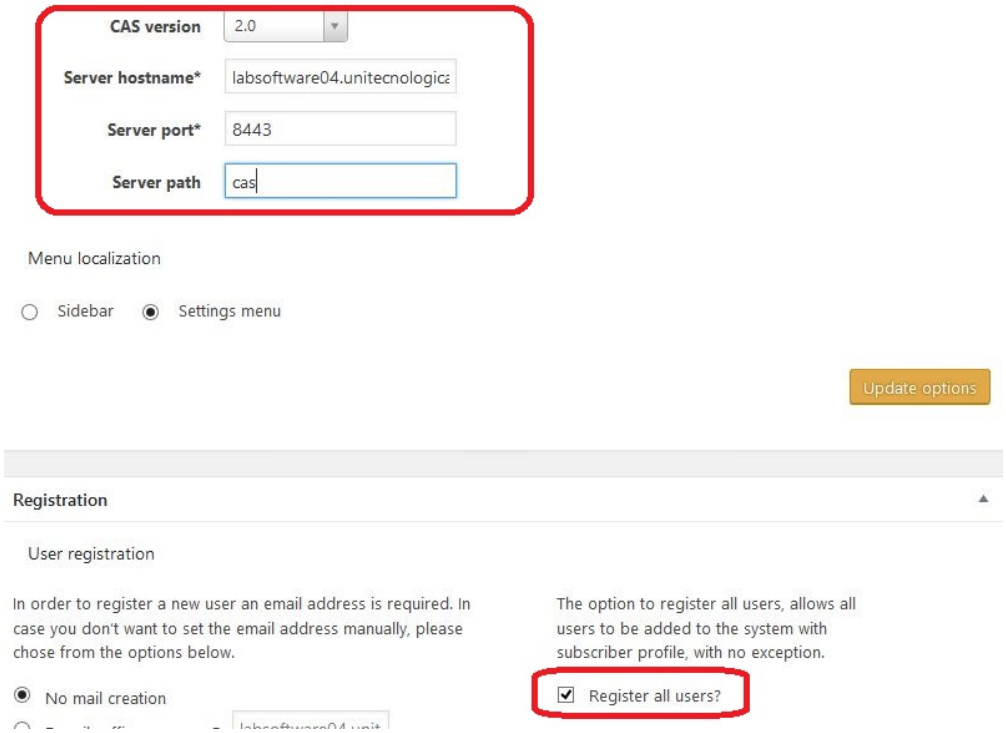


Figura 5.13: Configuración de Autenticación con CAS en Wordpress

6 Conclusiones y recomendaciones

6.1. Conclusiones

Este trabajo se describe el análisis, diseño y la guía de implementación de un prototipo de un sistema SSO que permite a los estudiantes de la Universidad Tecnológica de Bolívar acceder a varias aplicaciones web de diferentes ambientes y tecnologías que se asemejan a los sistemas ofrecidos por la UTB. Para esto se expuso primeramente las características y beneficios de seguridad del uso del sistema SSO. También se ha descrito como implementar y configurar el servidor CAS como proveedor de SSO, ya que por sus características, flexibilidad y arquitectura podía adaptarse como solución a el contexto de los problemas de integrar por medio de un único punto de entrada los sistemas que ofrece la UTB a los estudiantes.
Finalmente se ha demostrado configuración de los clientes, en este caso Moodle y una aplicación de Java que simula respectivamente SAVIO y SIRIUS; también se demostró la integración como clientes una aplicación ASP, PHP y Wordpress.

6.2. Recomendaciones

Al desarrollar este trabajo, se encontraron diferentes obstáculos a la hora de implementar *100 %* CAS como proveedor del SSO en la universidad. Entre ellas hay configuraciones como por ejemplo de seguridad, disponibilidad, y entre otras que se deben tener en cuenta como trabajos a futuro:

- *Transporte Seguro (https)*: Toda la comunicación con el servidor CAS debe ocurrir en un canal seguro. Puesto que la revelación de los datos permite ataques de suplantación, es de vital importancia garantizar el canal de comunicación entre los clientes de CAS y el servidor CAS esté asegurada. Prácticamente, esto significa que todas las URLs CAS deben utilizar HTTPS, pero también significa que todas las conexiones desde el CAS Server a cada una de las aplicaciones debe hacerse mediante HTTPS.
- Tener un único directorio central para la información de los usuarios, es decir, que la información de los usuarios (en nuestro caso los estudiantes), como sus credenciales de acceso, estén centralizados en un único punto (por ejemplo en un servidor LDAP o Active Directory). Con el objetivo de no tener duplicidad de información de los usuarios al estar en diferentes directorios o bases de datos.

- *Registro de tickets basado en caché:* se recomienda la implementación de los componentes de almacenamiento de Tickets basado en caché, para proporcionar el mejor equilibrio entre la facilidad de uso, escalabilidad y tolerancia a fallos. Algunos de los componentes pueden ser: Hazelcast[1], Ehcache[2], Memcached[3].

- Se recomienda una configuración de alta disponibilidad del servidor SSO (en nuestro caso, CAS), asegurando que exista una redundancia adecuada para que el servicio sea robusto frente a: (1) las fallas de los componentes en un momento determinado, (2) rutinas de mantenimientos con el fin de que se pueda realizar sin tiempo de inactividad del servicio y (3) la sobrecargas de toda la población de usuarios que desean acceder al servicio. Esto puede lograrse colocando varios nodos, y en menor grado, con el CAS de un solo nodo con capacidades avanzadas de la máquina virtual (ver **figura 6.1**).

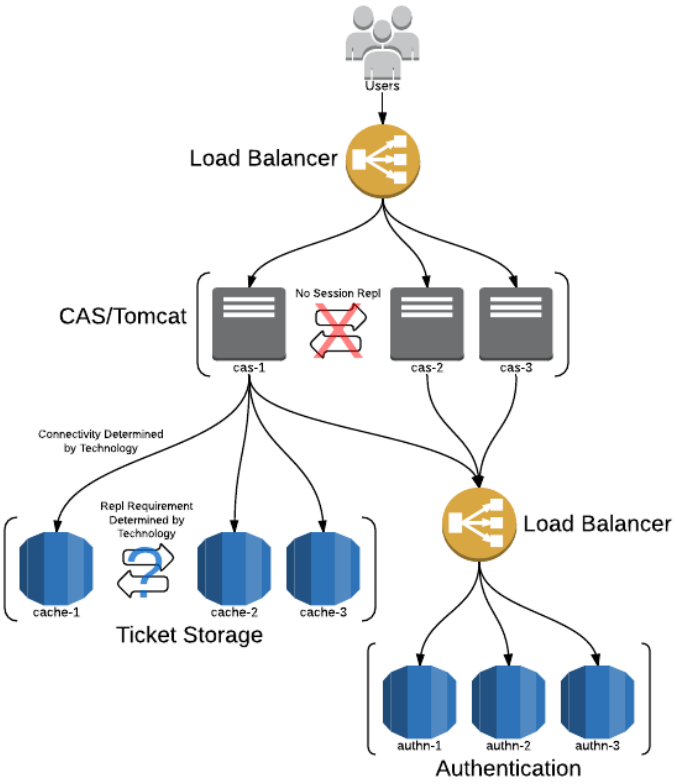


Figura 6.1: Arquitectura recomendada para alta disponibilidad por CAS [30].

[1]https://hazelcast.org/
[2]http://www.ehcache.org/
[3]http://memcached.org/

Bibliografía

[1] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[2] Shirey, R. W. (2007). Internet security glossary, Version 2.

[3] Samuel Muñoz Hidalgo. (2010). Ïntegración de un metasistema de identidad en la arquitectura edoroam para proporcionar un servicio de inicio de sesión único unificado".

[4] Zeilenga, K. OpenLDAP Software 2.3 Administrator's Guide. http://www.openldap.org/doc/admin23/. Accessed: 02-03-2015

[5] Rivera, N. B. (2009). Modelo de single sing-on para herramientas del Grupo Qualdev (Doctoral dissertation, Uniandes).

[6] Avisrubun Passport Protocol. http://avirubin.com/passport.html. Accessed: 2016-06-01.

[7] B. Clifford Neuman and Theodore Ts'o. "Kerberos: An authentication service for computer networks". In: Communications Magazine IEEE (Jan. 1994).

[8] Thomas GroÃŸ. "Security analysis of the SAML single sign-on browser/artifact profile". In: (Dec. 2003).

[9] EMC. DOCUMENTUM CONTENT SERVER CENTRAL AUTHENTICATION SERVICE (CAS) SSO. Tech. rep. 2013.

[10] Fang Yinglan, Jin Hao, and Han Bing. "Single Sign-On Research and Expan- sion Based On CAS". In: The Open Cybernetics And Systemics Journal (Aug. 2014).

[11] Seddigh, U. (2014). Evaluation of Single Sign-On Frameworks, as a Flexible Authorization Solution: OAuth 2.0 Authorization Framework.

[12] Powell, C.; Aizawa, T.; Munetomo, M., "Design of an SSO authentication infrastructure for heterogeneous inter-cloud environments,Çloud Networking (CloudNet), 2014 IEEE 3rd International Conference on , vol., no., pp.102,107, 8-10 Oct. 2014 doi: 10.1109/CloudNet.2014.6968976

[13] Yebin Chen; Baozhu Wu; Bing Xia; Lianghong Shi, "Design of web service single sign-on based on ticket and assertion,.Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on , vol., no., pp.297,300, 8-10 Aug. 2011 doi: 10.1109/AIMSEC.2011.6010391

[14] Revar, A.G.; Bhavsar, M.D., "Securing user authentication using single sign-on in Cloud Computing,.Engineering (NUiCONE), 2011 Nirma University International Conference on , vol., no., pp.1,4, 8-10 Dec. 2011 doi: 10.1109/NUiConE.2011.6153227

[15] Sliman, Layth, et al. "Single sign-on integration in a distributed enterprise service bus."Network and Service Security, 2009. N2S'09. International Conference on. IEEE, 2009.

[16] About Shibboleth, "What's Shibboleth?". https://shibboleth.net/about/. Accessed: 11-02-2016

[17] The University of Michigan, ÇoSign: Secure, Intra-Institutional Web Authentication,". http://www.weblogin.org/. Accessed: 11-02-2016

[18] About Shibboleth, "What's Shibboleth?". https://shibboleth.net/about/. Accessed: 11-02-2016

[19] RedHat Project, "Keycloak: SSO Solutions [Overview]". http://www.keycloak.org/. Accessed: 15-02-2016.

[20] Official Documentation, .Enterprise Single Sign-On". https://apereo.github.io/cas/4.2.x/index.html. Accessed: 12-02-2016

[21] Official Documentation, "Architecture". https://apereo.github.io/cas/4.2.x/planning/Architecture. Accessed: 12-02-2016

[22] Java SE — Oracle Technology Network — Oracle "Java SE Override". http://www.oracle.com/technetwork/java/javase/overview/index.html. Accessed 11-07-2016.

[23] Maven Overlays Configuration. http://maven.apache.org/plugins/maven-war-plugin/overlays.html. Accessed 05-07-2016

[24] Apache Maven Project Ïntroduction to the POM". https://maven.apache.org/guides/introduction/introduction-to-the-pom.html. Accessed: 05-07-2016

[25] Moodle CAS Server SSO. https://docs.moodle.org/30/en/CAS_server_(SSO)_authentication Accessed: 2016-06-01.

[26] GitHub "Apereo .NET CAS Client" https://github.com/apereo/dotnet-cas-client. Accessed: 18-07-2016.

[27] Wiegers, K., & Beatty, J. (2013). Software requirements. Pearson Education.

[28] EMC. CENTRAL AUTHENTICATION SERVICE (CAS) SSO FOR EMC DOCUMENTUM REST SERVICES. January, 2014.

[29] . Ceria, S. (1998). Casos de Uso un método practico para explorar requerimientos. Universidad de Buenos Aires.

[30] Official Documentation CAS, "High Availability Guide (HA/Clustering)". https://apereo.github.io/cas/4.2.x/planning/High-Availability-Guide.html. Accessed: 12-02-2016