Universitat Oberta de Catalunya

uoc.edu

Servicios de identidad federada en el ámbito empresarial

Rubén Ruiz Torres
Máster Interuniversitario en seguridad de las tecnologías de la información y de las comunicaciones.

Sistemas de autenticación y autorización

Victor Méndez Muñoz
Víctor García Font

02/06/20

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Servicios de identidad federada en el ámbito empresarial*
Nombre del autor:	*Rubén Ruiz Torres*
Nombre del consultor/a:	*Victor Méndez Muñoz*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Máster Interuniversitario en seguridad de las tecnologías de la información y de las comunicaciones.*
Área del Trabajo Final:	*Sistemas de autenticación y autorización*
Idioma del trabajo:	*Castellano*
Palabras clave:	*Identidad digital, identidad federada, sistemas distribuidos*

Resumen del Trabajo:

Los servicios de identidad federada permiten al usuario identificarse en distintos servicios haciendo uso de las mismas credenciales y del mismo proveedor de identidad. En el ámbito empresarial, donde los procesos de identificación y la autorización de los usuarios y de las aplicaciones que integran el ecosistema de cada empresa requieren de especial fiabilidad y seguridad, suelen emplearse servicios de identidad federada privados.

En el presente proyecto se analizan varias soluciones para la adopción de servicios de identidad federada privados en el ámbito empresarial, como son los estándares utilizados de proporción de servicios de identidad federada SAML, CAS y OpenID Connect, los proveedores de identidad Keycloak, Apereo CAS y OpenAM, además de los frameworks de programación para Java Spring, Play y Quarkus. Asimismo se ha desarrollado un prototipo con el que estudiar el funcionamiento de una de estas soluciones en cada uno de estos frameworks en el contexto simulado del ecosistema de aplicaciones de una empresa constituido por una aplicación de gestión de vacaciones, una aplicación de gestión de pedidos y un microservicio de generación de informes.

Abstract (in English):

Federated identity services allow users to identify themselves in different services using both the same credentials and the same identity provider. In a corporate environment, where the identification and authorization of users and of the applications included in a company's ecosystem must be especially reliable and secure, private federated identity services are typically used.

In this study we analyze several possible solutions regarding the implementation of private federated identity services in a corporate environment, including several standards used to provide federated identity services such as SAML, CAS and OpenID Connect and several identity providers such as Keycloak, Apereo CAS and OpenAM, as well as the Java programming frameworks Spring, Play and Quarkus. We have likewise developed a prototype intended to allow us to study how one of these solutions works in each of the aforementioned frameworks within the simulated context of a company's application ecosystem, which includes a holiday manager application, an order generating application and an order report generating microservice.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Los servicios de identidad federada [1] permiten al usuario identificarse en distintos servicios haciendo uso de las mismas credenciales y del mismo proveedor de identidad centralizado. Es frecuente que un proveedor de servicio delegue el proceso de identificación del usuario a un proveedor de identidad tales como Google, Facebook o Apple, lo que facilita el proceso de registro y proporciona al usuario gran control sobre su identidad: un ejemplo sería el de la aplicación de reproducción de música vía *streaming* Spotify, que permite a sus usuarios registrarse en la plataforma mediante su cuenta de Facebook o Google.

Sin embargo, en el ámbito empresarial este tipo de soluciones no suelen resultar prácticas, ya que la identificación y el registro de los empleados en el directorio de una empresa requieren de una mayor fiabilidad. Actualmente, las empresas utilizan ecosistemas que constan de múltiples aplicaciones en las que se integran todos los servicios necesarios para su funcionamiento. Ahora bien, la identificación y la autorización de los usuarios o de las distintas aplicaciones entre sí, de realizarse de forma aislada para cada aplicación, puede provocar tanto errores de sincronización entre las aplicaciones como a, en caso de no tenerse en consideración todas las posibles eventualidades, graves problemas de seguridad. De ahí que a menudo se empleen en este contexto servicios de identidad federada privados, que permiten la gestión interna de usuarios; más concretamente, en el ámbito empresarial el uso de servicios de identidad federada privados posibilita que distintos departamentos o áreas de una empresa compartan las identidades de los usuarios, así como el empleo de tecnologías que facilitan la interacción de los usuarios con distintas aplicaciones, como por ejemplo Single Sign-on. Este tipo de servicio se puede adquirir fundamentalmente en dos formatos: SAAS o «software as a service» (en cuyo caso se delega la identificación a un servicio online centralizado fuera de nuestro control) y «on premise» (en cuyo caso el servicio se instala en un centro de datos controlado exclusivamente por la empresa). En este proyecto se analizarán varias soluciones actualmente disponibles para la adopción de servicios de identidad federada privados en el ámbito empresarial, con énfasis en los servicios de proveedores de identidad tipo *on premise,* y se desarrollará un prototipo que permitirá estudiar el funcionamiento de una de estas soluciones en diferentes *frameworks* de programación dentro de un contexto empresarial simulado constituido por varios servicios internos de una empresa.

El prototipo contará con un servicio de identidad federada (escogido de entre los previamente analizados), una aplicación sencilla de gestión de vacaciones (que permitirá a los empleados solicitar, modificar y cancelar sus vacaciones y a la empresa autorizar o denegar dichas vacaciones), una aplicación sencilla de gestión de pedidos (que permitirá al usuario generar pedidos y a la empresa productos nuevos) y un microservicio de generación de informes (que proporcionará informes de pedidos en formato PDF) empleado por la aplicación anterior. Las distintas aplicaciones se desarrollarán en *frameworks* de desarrollo web diferentes con objeto de simular la heterogeneidad propia de los ecosistemas empresariales, donde distintos equipos de desarrollo utilizan distintas tecnologías, y por tanto de reproducir problemáticas similares a las que surgirían en una situación real.

1.2 Objetivos del Trabajo

El presente proyecto presenta por tanto dos objetivos generales que a su vez se subdividen en una serie de objetivos específicos, a saber:

1. Análisis de las soluciones para la adopción de servicios de identidad federada privados en el ámbito empresarial, con énfasis en los servicios tipo *on premise.*
2. Desarrollo de un prototipo para estudiar el funcionamiento de un servicio de identidad federada en diferentes frameworks dentro de un contexto empresarial simulado.

El primer objetivo general se subdivide en los siguientes objetivos específicos:

1.1. Análisis general de los protocolos utilizados para proporcionar servicios de identidad federada en el ámbito empresarial, tales como SAML [2], CAS [3] o OpenID Connect [4].

1.2. Análisis de los proveedores de identidad de tipo *on premise*, con énfasis en aquellos de código abierto y gratuito tales como Keycloak [5], Apereo CAS [6] u OpenAM [7]

1.3. Análisis de los tres *frameworks* para Java que se emplearán en el desarrollo del prototipo, a saber, Spring [8], Play [9] y Quarkus [10].

El segundo objetivo general se subdivide en los siguientes objetivos específicos:

- Desarrollar las distintas aplicaciones que constituyen el contexto empresarial simulado (aplicación de gestión de vacaciones, aplicación de gestión de pedidos y micro-servicio de generación de informes).
- Instalación y configuración del servicio de identidad federada.
- Conexión de cada una de las aplicaciones con el servicio de identidad federada.
- Creación de usuarios y roles, establecimiento del sistema de permisos de los mismos para cada de las aplicaciones.
- Comprobación del correcto funcionamiento de las aplicaciones y de su integración con el sistema de identidad federada.
- Evaluación de las herramientas utilizadas.

1.3 Enfoque y método seguido

Para llevar a cabo el primero de los objetivos generales del presente proyecto se realizará un inventario de las soluciones disponibles para la adopción de servicios de identidad federada privados, se realizará una descripción de cada una de las mismas y en función de sus características y del contexto que nos ocupa se determinará cuál de ellas se empleará en el desarrollo del prototipo.

A la hora de desarrollar las aplicaciones se empleará una metodología ágil de programación, con el objetivo de acotar el tiempo dedicado al desarrollo de cada aplicación y tener la posibilidad, a través de una serie de revisiones periódicas, de ajustar los objetivos del proyecto en función de cómo evolucione el desarrollo del mismo. Las tareas de identificación de usuarios e integración de las aplicaciones con el servicio de identidad federada tendrán carácter obligatoria y contarán con una fecha concreta de implementación. En todo momento se procurará utilizar tecnologías que favorezcan la estabilidad y reproducibilidad del entorno de desarrollo, tales como Docker [11] o Gradle [12].

1.4 Planificación del Trabajo

La planificación general del proyecto es la siguiente:

Table 1: Planificación general del proyecto

Definición	**Fecha de inicio**	**Fecha de fin**
Planificación del trabajo	24-02-2020	02-03-2020
Entrega 1	03-03-2020	03-03-2020
Desarrollo: App básica de gestión de vacaciones	04-03-2020	18-03-2020
Análisis de protocolos	04-03-2020	08-03-2020

Preparación del análisis	04-03-2020	04-03-2020
SAML	04-03-2020	05-03-2020
OpenID Connect	06-03-2020	07-03-2020
CAS	07-03-2020	08-03-2020
Análisis de proveedores de identidad	**09-03-2020**	**16-03-2020**
Preparación del análisis	09-03-2020	09-03-2020
CAS	09-03-2020	10-03-2020
Keycloak	11-03-2020	13-03-2020
OpenAM	14-03-2020	15-03-2020
Decidir y justificar qué proveedor de identidad utilizaremos	16-03-2020	16-03-2020
Análisis de los frameworks para Java	**18-03-2020**	**31-03-2020**
Play	18-03-2020	19-03-2020
Desarrollo: App básica de gestión de pedidos	**19-03-2020**	**30-03-2020**
Spring	19-03-2020	20-03-2020
Quarkus	21-03-2020	22-03-2020
Entrega 2	**31-03-2020**	**31-03-2020**
Desarrollo: Microservicio de generación de informes	**01-04-2020**	**08-04-2020**
Desarrollo: Instalación y configuración de servicio de identidad federada	**06-04-2020**	**11-04-2020**
Desarrollo: Conexión de las aplicaciones	**14-04-2020**	**21-04-2020**
Conexión de la aplicación de Pedidos	14-04-2020	16-04-2020
Conexión de la aplicación de Informes	16-04-2020	18-04-2020
Conexión de la aplicación de Vacaciones	19-04-2020	21-04-2020
Desarrollo: Usuarios / Roles y permisos	**21-04-2020**	**24-04-2020**
Configuración de control de acceso en aplicación de informes	21-04-2020	22-04-2020
Configuración de control de acceso en aplicación de vacaciones	22-04-2020	23-04-2020
Configuración de control de acceso en aplicación de pedidos	23-04-2020	24-04-2020
Desarrollo: Verificación y pruebas	**24-04-2020**	**27-04-2020**
Entrega 3	**28-04-2020**	**28-04-2020**
Conclusiones y escritura de la memoria	**29-04-2020**	**03-06-2020**
Entrega 4	**04-06-2020**	**04-06-2020**

Por lo que se refiere a las aplicaciones, su desarrollo se dividirá en una serie de tareas o user stories, las cuales se encuentran priorizadas en función de su importancia para el proyecto, de manera que su programación se abordará de manera ordenada en el tiempo. Dado que se contará con un tiempo limitado para la realización de las user stories de cada aplicación, de esta manera se garantiza que, en caso de tener que abandonar alguna de ellas por falta de tiempo, se trate de las que menor importancia revistan para el proyecto.

Table 2: User stories relacionadas con la aplicación de gestión de vacaciones

Id.	**User story**
1	Como un usuario quiero solicitar vacaciones
2	Como un administrador quiero aprobar vacaciones
3	Como un usuario quiero ver un listado con mis vacaciones junto con su estado
4	Como un administrador quiero denegar vacaciones

Table 3: User stories relacionadas con la aplicación de pedidos

Id.	**User story**
5	Como un usuario quiero realizar un pedido
6	Como un administrador quiero generar nuevos productos
7	Como un usuario quiero ver un listado de pedidos
8	Como un administrador quiero generar nuevos productos

Table 4: User stories relacionadas con el microservicio de informes

Id.	**User story**
9	Como un usuario quiero generar un informe con la información relacionada con mi pedido.

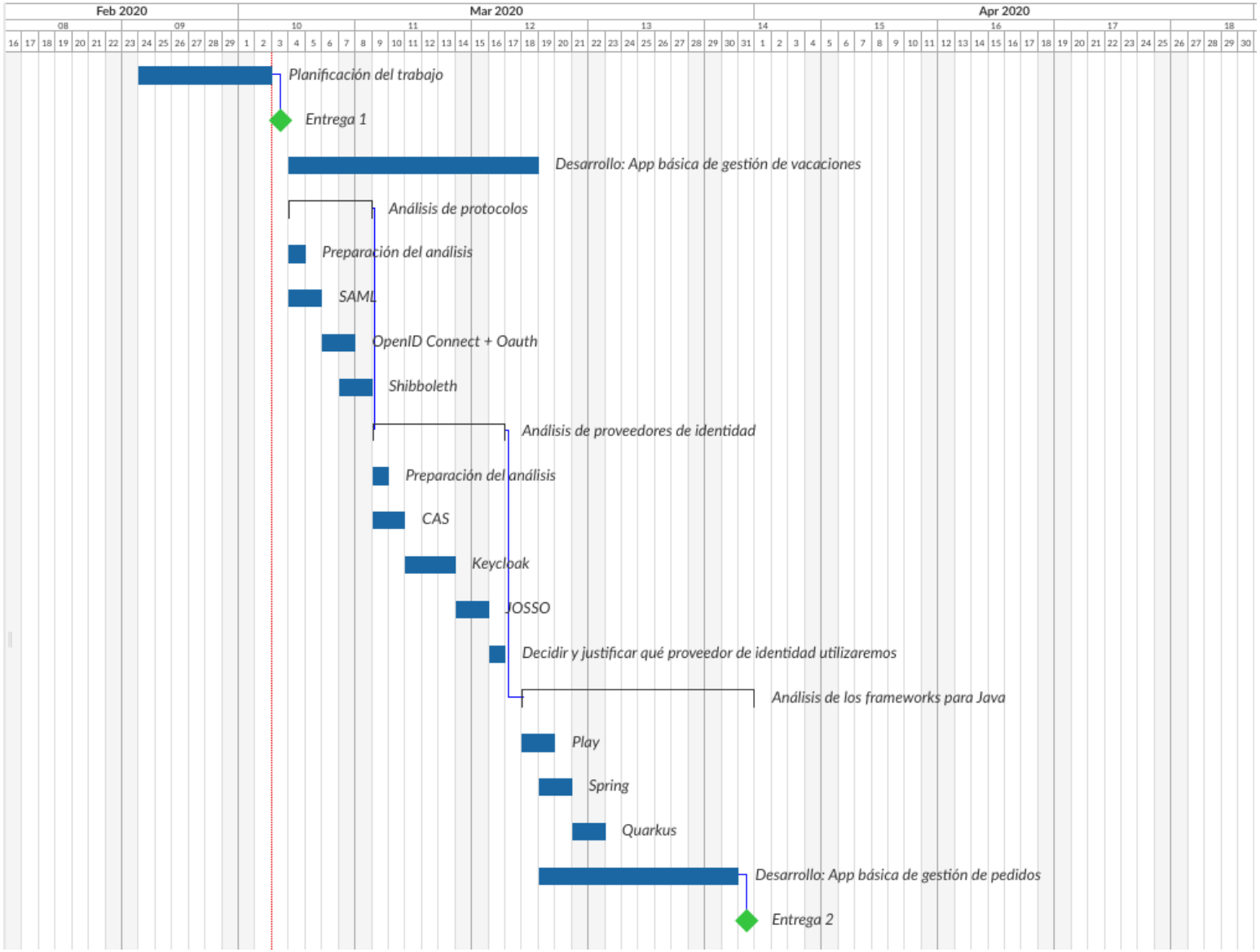


Figure 1: Diagrama de Gantt: Entregas 1 y 2

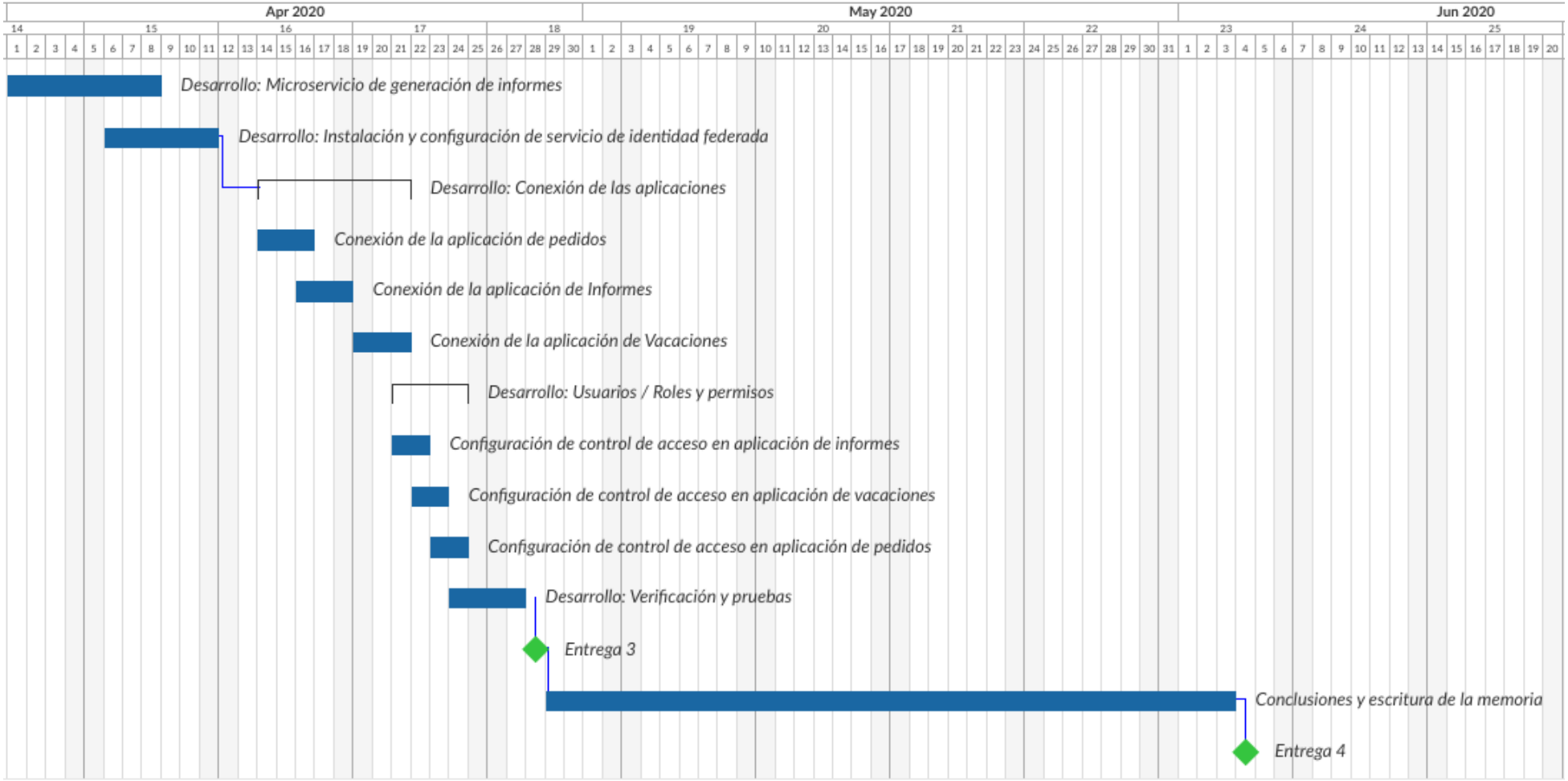


Figure 2: Diagrama de Gantt: Entregas 3 y 4

1.5 Breve sumario de productos obtenidos

En este trabajo se han desarrollado los siguientes productos:

- Estudio y análisis de una selección de tecnologías, librerías y estándares que posibilitan introducir un sistema de gestión de identidad federada en el ámbito empresarial, con énfasis en el procedimiento de autenticación Single Sign On (SSO)
- Una prueba de concepto o prototipo con tres aplicaciones web diferentes desarrolladas en Java, cada una de ellas basadas en un *framework* de desarrollo web distinto (a saber, Spring, Play 2 y Quarkus).
- Integración de las tres aplicaciones con el proveedor de identidad Keycloak mediante el estándar OpenID Connect, empleando fundamentalmente librerías integradas en el *framework* en cuestión o específicas para el mismo.
- Descripción y análisis de los pasos necesarios para implementar la integración de las aplicaciones con Keycloak mediante OpenID Connect.

2. Análisis

2.1 Estándares que proporcionan servicios de identidad federada en el ámbito empresarial

En este apartado se analizarán tres de los estándares que proporcionan servicios de identidad federada en el ámbito empresarial como son SAML, OpenId Connect y CAS. De entre las posibles aplicaciones de estos estándares nos centraremos en el procedimiento de autenticación conocido como Single Sign On (SSO), que permite al usuario conectarse a distintos sistemas a través de un solo acto de identificación.

2.1.1 SAML

SAML (Security Assertion Markup Language) es un estándar abierto utilizado en aplicaciones que necesitan SSO. Fue desarrollado por Oasis, un consorcio empresarial sin ánimo de lucro en el que se incluyen compañías como IBM, Dell, Microsoft, Oracle o Redhat y responsable, entre otros, de los estándares OpenDocument y XACML. La última versión de SAML, la 2.0, data de 2005[1].

Figura 1: Logotipo de SAML 2.0

En este estándar existen tres roles, a saber:

- Navegador web (HTTP Client): navegador web que solicita el recurso protegido y a través del cual se realiza el proceso de autenticación.
- Proveedor de servicio (SP/Service Provider): servicio web al que se solicita el recurso y que ha de verificar la identidad y autorización del usuario.
- Proveedor de identidad (IdP/Identity Provider): servicio web capaz de verificar la identidad del usuario y proporcionar información sobre este.

El estándar se basa en el intercambio de mensajes en formato XML entre las tres partes involucradas a través del protocolo HTTP y emplea herramientas de uso extendido en la en el ámbito empresarial tales como XML, XML Schema (XSD), XML encryption, o XML Signatures, lo que garantiza la interoperabilidad y facilita la implementación. Cuenta con especificaciones de carácter más general recogidas en el documento SAMLCore y con una serie de documentos en los que se detallan información y casos más específicos.

2.1.1.1 SAML Bindings:

SAML puede emplear distintos métodos a la hora de transferir mensajes XML a través del protocolo HTTP. Dos de los más sencillos son Redirect Binding y POST Binding, cuyo modus operandi consiste en enviar directamente la información codificada en parámetros HTTP a una URL concreta.

2.1.1.1.1 HTTP Redirect (GET) Binding

En este caso, los mensajes XML se envían a través una solicitud HTTP GET. Más concretamente, los mensajes se codifican en base64 y URL y se envían asociado al parámetro "SAMLRequest" o "SAMLResponse", existiendo la posibilidad de firmarlos antes de su codificación. Ahora bien, este método solo es factible para mensajes cortos (como por ejemplo “AuthnRequest”), dado que en las solicitudes HTTP GET los parámetros forman parte de su URL y no solo los navegadores sino muchos otros servicios web aplican restricciones en la longitud de esta.

---

1 Siempre que se haga alusión a las últimas o más recientes versiones de los distintos estándares, proveedores, *frameworks*, etc. mencionados en el presente trabajo, así como a funcionalidad de dichos estándares, proveedores, *frameworks*, etc. se tendrá como referencia el momento de redacción del mismo.

2.1.1.1.2 HTTP Post Binding

En este caso, bien el SP, bien el IdP responden al navegador utilizando un formulario XHTML en el que se incluyen diversos parámetros, tales como SAMLRequest, SAMLResponse, RelayToken, etc. El navegador habrá de enviar dicho formulario de forma manual (en el siguiente, el usuario habría de clicar en el botón «Submit») o mediante un script en Javascript que lo envíe automáticamente (según las especificaciones del estándar, cualquier solución, tanto manual como automática, sería válida. Se puede observar un ejemplo de este formulario en Ejemplo 8.

2.1.1.2 SAML Assertions

Se denomina assertions a los paquetes de información que contienen datos acerca de la identidad del usuario, su grado de autorización y el acceso a sus recursos que forman parte de la respuesta que el proveedor de identidad devuelve al SP. Existen tres tipos de assertions, a saber:

- Authentication assertion: indica que el usuario se ha autenticado en el IdP y proporciona información acerca de la autenticación; se define con la etiqueta <saml:AuthnStatement>.
- Attribute Assertion: proporciona información adicional acerca del usuario; se define con la etiqueta <saml:AttributeStatement>.
- Authorization Decision Assertion: da respuesta a la cuestión de si el usuario se encuentra autorizado para acceder a determinados recursos; se define con la etiqueta: <saml:AuthorizationDecisionStatement>.

2.1.1.3 SAML Protocols

Las especificaciones de SAML indican los distintos protocolos que el estándar puede emplear en función de sus aplicaciones. En el caso que nos ocupa, el de SSO, uno de los posibles protocolos a emplear es el Authentication Request Protocol, que permite la autenticación de usuarios solicitada por un SP (AuthnRequest) En este protocolo, el IdP es el encargado de proporcionar al usuario un formulario con el que autenticarse, de verificar su identidad, de buscar o generar sus atributos y de enviar la respuesta al SP (AuthnResponse). Otros protocolos incluidos en las especificaciones de SAML son el Assertion Query and Request Protocol, el Artifact Resolution Protocol, el Name Identifier Management Protocol, el Single Logout Protocol y el Name Identifier Mapping Protocol

Los ficheros presentes en el Ejemplo 9 y Ejemplo 10 ilustran las dos fases del protocolo, solicitud y respuesta. En la primera, el SP lleva a cabo una petición de autenticación al IdP mediante Post Binding y exigiendo que el nivel de seguridad sea como mínimo «PasswordProtectedTransport» (es decir, que la autenticación se produzca al menos mediante usuario y contraseña a través de un canal seguro); el mensaje no se encuentra firmado. En la segunda, en la que el IdP envía al SP la información que este le ha solicitado, destacan las siguientes elementos:

- Status: especifica la conclusión del proceso de autenticación.
- NameId: constituye un identificador único para el usuario autenticado.
- AuthnContextClassRef: expresa el nivel de seguridad garantizado por el IdP durante el proceso de autenticación.
- AttributeStatement: representa los atributos del usuario que el IdP devuelve al SP.

2.1.1.4 SAML Metadata

La configuración de la conexión entre los distintos roles que intervienen en SAML se lleva a cabo a través de archivos XML denominados metadata o metadatos. Estos archivos se encuentran encabezados por la etiqueta <md:entityDescriptor>, cuentan con un identificador único llamado «entityId» y contienen información sobre la organización correspondiente al rol en cuestión, SP o IdP.

En el caso que nos ocupa, la configuración emplearía además una de las siguientes etiquetas en función de si la configuración corresponde a un SP o a un IdP: <IDPSSODescriptor> y <SPSSODescriptor> respectivamente. La etiqueta <IDPSSODescriptor> define la información que necesitaría un SP para establecer una conexión, a saber: los detalles del binding (incluyendo el método y la URL a la que se enviarán los mensajes (SingleSignOnService) y la clave pública que permitirá al SP verificar la firma de las respuestas del IdP (KeyDescriptor).

La etiqueta <SPSSODescriptor> define la información empleada por el IdP para configurar su respuesta a un SP, a saber: la URL y el método de binding utilizados para enviar la respuesta (AssertionConsumingService) y los atributos requeridos por el SP (AttributeConsumingService).

Por ejemplo, un IdP con «entityId» idp.example.local tiene dos ficheros de metadatos entityDescriptor que definen la conexión con dos SP (entityIds: sp1.a-different-domain.local y sp2.another-example.local). Esto es, los ficheros de metadatos contendrán un SPSSODescriptor.

En el ejemplo, cada uno de los SP sólo necesitarán metadatos con la información del IdP, este único fichero de metadatos contiene un IDPSSODescriptor con entityId idp.example.local. La implementación SAML utilizada por el SP sólo necesitaría acceder a este fichero para saber qué medidas tomar para proceder con la autenticación de un usuario.

De cara al intercambio de metadatos entre las distintas partes implicadas en el proceso de autenticación, caben dos aproximaciones, el intercambio de metadatos dinámico y el estático. En el caso del intercambio estático, cada parte implicada dispone de manera local de los fichero de metadatos, que se cargan manualmente; es decir: el IdP dispone de los ficheros de metadatos de todos los SP que se conectan a él y dichos SP disponen del fichero de configuración del IdP. En el caso del intercambio dinámico, la gestión de los metadatos se deriva a un servicio externo que almacena los ficheros de las distintas partes implicadas y los pone a disposición de estas.

El intercambio de metadatos constituye una cuestión a tener en cuenta en el caso de SAML, puesto que los ficheros de metadatos incluyen información que ha de cambiar en el tiempo; la clave pública, por ejemplo, suele tener fecha de expiración. Por tanto, el programa encargado de interpretar el mensaje no solo ha de comprobar que esté firmado correctamente, sino también que la clave usada para hacerlo no haya expirado, de manera que la información de la que disponen todas las partes implicadas en el proceso de autenticación ha de encontrarse debidamente actualizada.

En el caso del intercambio estático, al aproximarse la fecha de expiración de la clave empleada por el IdP para firmar los mensajes XML habrá que configurar nuevo ficheros para todos los SP que se conectan con él, que en el caso de empresas de un tamaño importante pueden ser cientos. Además, el proceso de configuración de nuevos ficheros de metadatos para un SP cuenta con varios pasos en los que pueden surgir problemas y resulta delicado en lo que a seguridad se refiere. El intercambio dinámico disminuye la probabilidad de que surjan problemas al reducir el número de pasos implicados en el proceso, pues solo es necesario configurar un nuevo fichero de metadatos que el servicio externo pone a disposición de todos los SP.

2.1.2 OpenID Connect

OpenID Connect[2] es un estándar consistente en una especificación que establece una capa de autenticación sobre el estándar OAuth 2.0, el cual se orienta única y exclusivamente a procesos de autorización. OpenID Connect está desarrollado por la organización sin ánimo de lucro OpenID Foundation, apareció por primera vez en 2014 y permite a una aplicación delegar el proceso de verificación de un usuario a un proveedor de identidad y recabar información de dicho usuario tras obtener su consentimiento. La transferencia se realiza a través de una API de tipo REST y la información se codifica en forma de ficheros JSON (más concretamente mediante el estándar JSON Web Tokens (JWT), descrito en la especificación RFC 7519 [13]).

Figura 2: Logo de OpenID

La especificación OpenID Connect permite, entre otras cosas, encriptar los datos identificativos de los usuarios (a través de JWE, RFC 7516 [14]), encontrar proveedores de identidad OpenID y gestionar sesiones.

En esta especificación existen tres roles, a saber:

- OpenID provider (OP): proveedor de identidad en OpenID Connect; facilita el proceso de autenticación por SSO o mediante la intervención del usuario.
- Relying Party (RP): proveedor de servicio en OpenID Connect; verifica la identidad del usuario y extrae información de su perfil.
- Usuario final a través de un navegador web: proporciona sus credenciales de acceso y acepta o deniega la solicitud de consentimiento del OP.

2.1.2.1 OpenID Connect Endpoints

En OpenID Connect existen tres «endpoints» correspondientes a URL que el RP emplea para llevar a cabo los distintos pasos necesarios para la autenticación de un usuario, a saber:

- Authorization Endpoint (/authorize): dirige el navegador del usuario al formulario de autenticación; devuelve un código de autorización o un ID Token en función del flujo de autenticación escogido.
- Token Endpoint (/token): dirige al usuario a solicitar un ID Token en caso de que se haya escogido el flujo de autenticación «Authorization code flow».
- User Management Endpoint: proporciona información de perfil de un usuario ya autenticado.

Otros «endpoints» definidos en las especificaciones de OpenID Connect son WebFinger Endpoint (utiliza el protocolo protocolo WebFinger [15]) para encontrar OP basándose en información del usuario), Provider Metadata Endpoint (proporciona información acerca de las distintas URL y de la configuración del OP que el RP puede utilizar para configurar una conexión), Provider JWK Set Endpoint (contiene un documento en formato JSON con información sobre las claves públicas utilizadas para firmar los mensajes JWT), Client Registration Endpoint (permite registrar nuevos clientes o RP de forma automática) o Session Management Endpoint (permite a los RP comprobar si el usuario se encuentra todavía autenticado en el OP o no).

2.1.2.2 OpenID Connect Claims

En OpenID Connect se denomina «claims» a la información relativa a un usuario o entidad, pudiéndose solicitar dicha información de dos formas posibles: especificando en el parámetro correspondiente de la solicitud un grupo de «claims» o «scope»[3] (por ejemplo, «phone (scope)» incluye los «claims» «phone_number y «phone_number_verified»), o

2 No confundir con el protocolo OpenID, también desarrollado por la OpenID Foundation.

especificando los atributos concretos deseados mediante el parámetro opcional «claims» (por ejemplo, «phone_number (claim)» se corresponde con el número de teléfono del usuario). Ahora bien, durante el proceso de identificación en el OP el usuario ha de dar su consentimiento sobre la información solicitada, de manera que cabe la posibilidad de que el RP solo reciba parte de la misma.

2.1.2.3 Registro en OpenID Connect

De acuerdo con las especificaciones de OpenID Connect [18], todos los proveedores de servicio (RP) deben encontrarse registrados en los proveedores de identidad (OP). El proceso de registro puede llevarse a cabo de forma manual o automática, estando esta última definida en las especificaciones adjuntas a OpenID Connect «Discovery» [19] y «Dynamic registration» [20].

Durante el proceso de registro, ambas partes, RP y OP, intercambian una serie de datos. El RP le envía al OP un identificador de cliente que identifica única y exclusivamente al RP, las URL de redirección al RP y un mecanismo de autenticación y credenciales para el RP. El OP, por su parte, le envía al RP las URL de los distintos «endpoints» (Authorization, Token y User), un identificador del distribuidor que identifica única y exclusivamente al OP y un mecanismo para obtener las claves públicas del OP.

2.1.2.4 OpenID Connect Flows

Existen en Open ID Connect existen tres flujos que definen el proceso de autenticación desde que el RP solicita al OP que autentique un usuario y hasta que recibe la información de este: Authorization Code Flow, Implicit Flow y Hybrid Flow.

- Authorization code flow: en este flujo el navegador no tiene acceso directamente a la información de perfil del usuario, pues el RP a requeriría del OP a través de una solicitud máquina a máquina o «backchannel».
- Implicit flow: en este flujo la información es enviada al navegador directamente tras la autentificación del usuario (este flujo está por tanto indicado para aplicaciones que se ejecutan dentro del navegador, como las aplicaciones de Javascript).
- Hybrid flow: en este flujo parte de la información es devuelta a través del Authorization endpoint y parte a través del Token Endpoint.

En el presente trabajo, nos centraremos en el primero de estos flujos, al ser el recomendado para el desarrollo de aplicaciones como las que nos ocupa en este caso. Para ilustrar su funcionamiento nos basaremos en el caso de un RP (client.example.com) que desea autenticar a un usuario en un determinado OP (openid.provider.local) [20].

3 Tanto la lista de «claims» [16] como la de «scopes» [17] se recogen en las especificaciones de OpenID Connect.

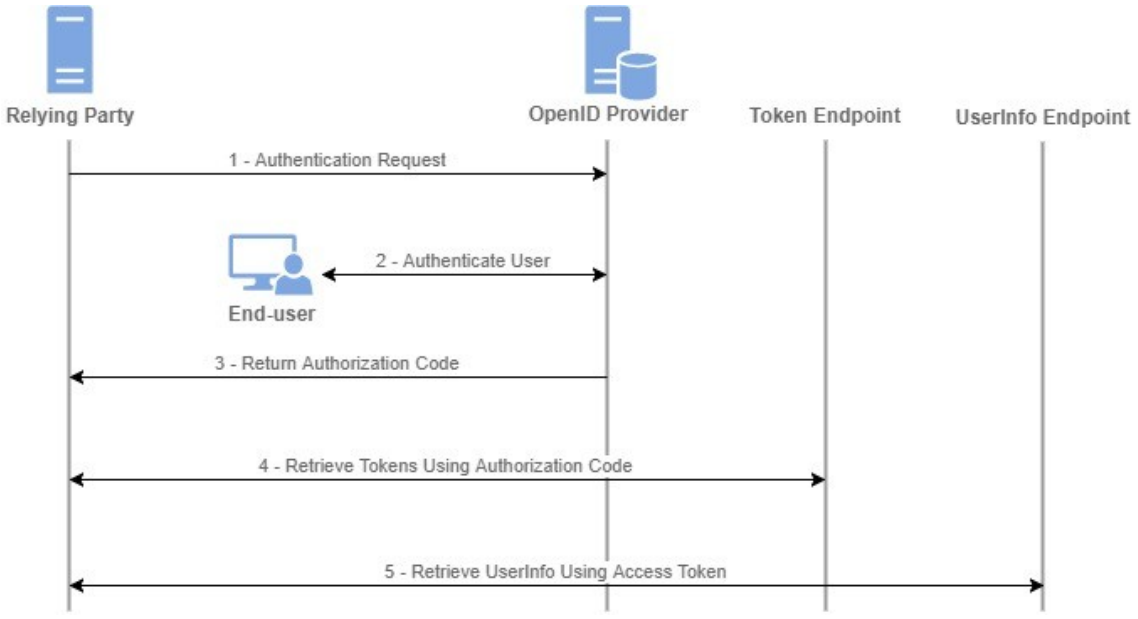


Figura 3: Authorization code flow en OpenID Connect [58]

El flujo se dividiría en dos pasos, la solicitud del código de autorización y la del ID Token. La solicitud del código de autorización contiene los siguientes parámetros:

- response_type: tipo de respuesta que el RP espera recibir del OP (en este caso, el código de autorización).
- client_id: identificador único del RP en el OP.
- redirect_uri: URL a la que redireccionar el navegador una vez finalizado el proceso de autenticación.
- state: información relacionada con la sesión del RP que el OP emplea para mantener el estado entre la solicitud y el «callback».
- scope: lista de los «scope» solicitados por el RP separados por espacios, en la que como mínimo debe figurar el «scope» «openid».

Al realizarse la solicitud del código de autorización, el RP redirige el navegador del usuario a la URL de autorización del OP. Si la sesión en la URL del OP carece de cookies o la sesión asociada a dichas cookies ha caducado, el OP muestra al usuario un formulario para que se autentique y, una vez que la autenticación se lleva a cabo con éxito, otro formulario para que el usuario acepte o rechace el envío de sus datos al RP. Finalmente, el usuario es redirigido al RP junto con su código de autorización. Se puede ver en detalle ambas redirecciones en Ejemplo 11 y Ejemplo 12

A continuación se lleva a cabo la solicitud del ID Token, en la que el RP solicita al OP el «ID Token» correspondiente al código de autorización que acaba de recibir mediante una solicitud «backchannel» en la que dicho código constituye uno de los parámetros Ejemplo 14. Otros parámetro incluidos en la solicitud son los siguientes:

- grant_type: indica al OP que se está enviando un código de autorización.
- code: código de autorización.
- redirect_uri: mismo valor que el utilizado en la solicitud del código de autorización.

El OP responde a la solicitud del RP con el ID Token correspondiente al código de autorización recibido en formato JWT. Este «id_token» contiene el siguiente contenido firmado:

{

"iss": "http://server.example.com",
"sub": "248289761001",
"aud": "s6BhdRkqt3",
"nonce": "n-0S6_WzA2Mj",
"exp": 1311281970,
"iat": 1311280970
}

El único «claim» incluido en esta respuesta que identifica al usuario es el correspondiente al atributo «sub» (Subject Identifier). Ahora bien, la respuesta también incluye asimismo un Access Token, que posteriormente puede utilizarse para interaccionar con otros endpoints como el user endpoint, de manera que el RP puede, de manera opcional, solicitar «claims» adicionales al OP a través de dicho endpoint. Se puede ver un ejemplo de una respuesta completa en Ejemplo 13.

Otros atributos incluidos en la respuesta son:

- iss («Issuerl Identifier»): identificador único del OP.
- Aud («Audience»): identificador único del RP.
- exp: fecha y hora de expliración del ID Token.
- iat: fecha y hora de emisión del JWT.

2.1.3 CAS

CAS (Central Authentication Service) es un estándar de autenticación basado en tickets. Su última versión, la 3.0.3 [3], data de diciembre de 2017, pero no se han introducido cambios mayores en sus especificaciones desde la publicaciones de la versión 3.0, en 2014. Tanto las especificaciones de este protocolo como su implementación de referencia (Apereo CAS) están desarrolladas por una fundación sin ánimo de lucro, «Apereo Foundation», bajo una licencia Apache 2.0 [22].

Figura 4: Logo de CAS

Originalmente CAS solo permitía definir un solo atributo identificativo del usuario en la respuesta del proveedor de identidad, lo que significa que para extraer información sobre este era necesario emplear servicios externos; sin embargo, desde la versión 3.0 es posible obtener otros atributos de usuario y, por tanto, información adicional sobre este.

En este protocolo existen tres roles, a saber:

- Navegador web / Cliente: navegador utilizado por el usuario para interaccionar con el servicio web y con el servidor CAS.
- Aplicación web / Servicio: servicio protegido por CAS y encargado de iniciar el proceso de autenticación redirigiendo al usuario al servidor CAS.
- Servidor CAS: servicio web encargado de gestionar el proceso de autenticación y de generar o proporcionar los atributos del usuario al servicio.

Cabe asimismo mencionar que en todos los proceso de autenticación gestionados mediante CAS participan tres elementos importantes:

- TGT (Ticket Granting Ticket): ticket que representa la sesión de SSO; se almacena en una cookie llamada «CASTGC» y asociada a la URL del servidor CAS en el navegador.
- ST (Service Ticket): ticket que el servidor CAS envía al servicio a través del navegador tras la autenticación y que este puede emplear para solicitar la información del usuario.
- CAS Response (XML): fichero XML que el servidor CAS envía al servicio y que contiene, bien la información del usuario autenticado, bien un mensaje de error.

2.1.3.1 CAS Endpoints

En CAS se definen tres «endpoints», a saber:

- **«/login»**: si no existe sesión SSO muestra un formulario de autenticación, recibe las credenciales del usuario, genera los tokens correspondientes (TGT y ST) y redirecciona el navegador del usuario al Servicio; si existe dicha sesión, lo redirecciona al servicio directamente incluyendo un «Service Token».
- **«/serviceValidate»**: recibe los nombres de servicio y los «service tokens», verifica la combinación de ambos y envía el fichero XML que contiene la información que identifica al usuario.
- **«/p3/serviceValidate»**: lleva a cabo las mismas operaciones que el anterior, pero además de la información que identifica al usuario también envía otros atributos, es decir, información adicional sobre este (solo encuentra disponible en servidores CAS con versión 3.0 o posterior).

2.1.3.2 CAS «Flows»

En CAS se definen dos flujos que definen el proceso desde que el servicio solicita la autenticación de un usuario al servidor CAS hasta que el servicio recibe la información del mismo. En el presente trabajo nos centraremos en el Web Authentication Flow y en el Proxy Web Flow.

2.1.3.2.1 Web Authentication Flow

Este flujo puede dividirse en tres pasos [23]: la solicitud inicial, la autenticación primaria y la validación del «Service Ticket». Para ilustrar el funcionamiento del flujo nos basaremos en el caso de un usuario sin sesión previa que trata de acceder a un servicio (https://app.example.com) registrado en el servidor CAS («https://cas.example.com»), el cual gestiona la autenticación y SSO de dicho servicio.

En la etapa de la solicitud inicial, el usuario intenta acceder al servicio, el cual comprueba que no hay ninguna sesión activa para ese usuario y redirige su navegador al Login Endpoint del servidor CAS, adjuntando como parámetro su nombre (el del servicio). Ver Ejemplo 15.

En la etapa de autenticación primaria, el servidor CAS proporciona al usuario un formulario en que introducir sus credenciales, las verifica, crea la cookie en que se almacena el TGT y redirige al usuario al servicio junto con un «Service Token». Ver Ejemplo 17.

En la etapa de validación del «Service Ticket», el servicio envía al servidor CAS una solicitud que incluye entre sus parámetros dicho «Service Token» y su (el del servicio), y el servidor le devuelve una respuesta que incluye otros atributos del usuario, es decir, información adicional sobre el mismo. Ver Ejemplo 16 y Ejemplo 18.

2.1.3.2.2 Proxy web flow

Este flujo se emplea específicamente para permitir que un servicio acceda a un segundo servicio protegido por CAS mediante una conexión máquina a máquina, es decir, sin que intervenga en el proceso el navegador del usuario. El servicio protegido actuaría por tanto como proxy para el primer servicio.

2.1.3.3 Configuración y registro en CAS

La especificaciones de CAS no recogen ningunas indicaciones por lo que se refiere a a la configuración y el registro de servicios en el servidor CAS. Por lo que el medio de registro de nuevos clientes queda abierto a los implementadores del proveedores de identidad.

A la hora de analizar las soluciones disponibles para la adopción de servicios de identidad federada privados en el ámbito empresarial, en el presente proyecto hemos decidido centrarnos en los servicios de proveedores de identidad de código abierto y gratuitos de tipo on premise, esto es, en los que el servicio se encuentra instalado en un centro de datos controlado exclusivamente por la empresa que emplea el servicio de identificación. Más concretamente, se examinarán los servicios Keycloak, OpenAM y Apereo CAS, proporcionando tanto una descripción general de los mismos como una constrastación de las características que los diferencian.

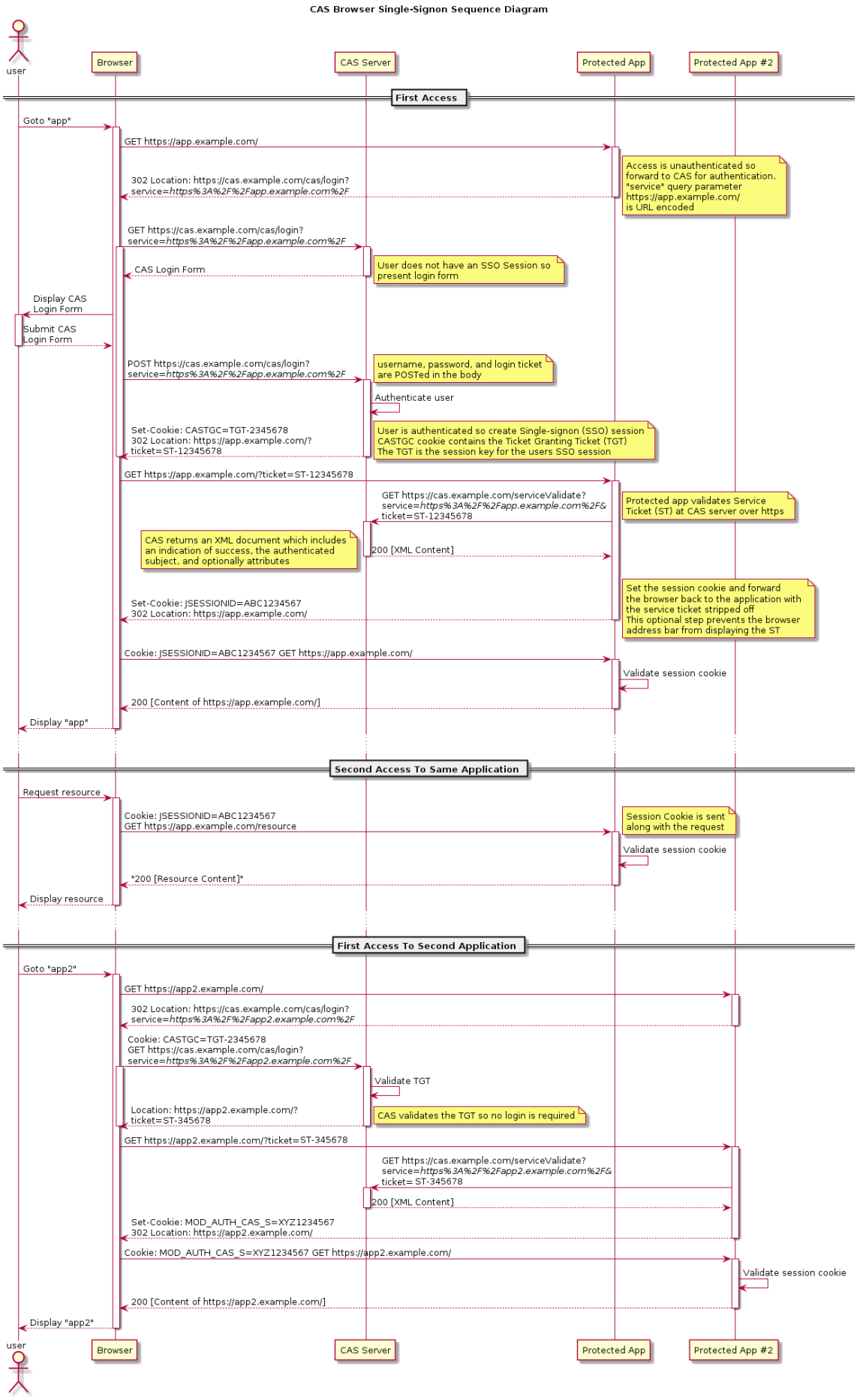


Diagrama 1: Diagrama del Web Authentication Flow en CAS extraido de la documentación ade Apereo [57]

2.2 Proveedores de identidad

2.2.1 Keycloak

Figura 5: Logo de Keycloak

Keycloak es una solución de gestión de identidad y control de acceso de código abierto escrita en el lenguaje de programación Java y publicada bajo licencia Apache 2.0 que permite gestionar la autenticación y autorización de usuarios de terceros servicios de manera sencilla y segura. Está desarrollada por la empresa JBoss y la comunidad Red Hat en general (Red Hat, además de financiar enconómicamente la solución, la emplea como base de su solución comercial SSO: «RH-SSO» [5]). La primera versión apta para producción fue publicada en septiembre de 2014 y su última actualización importante, la 9.0.0, data de febrero de 2020.

Entre las carácticas más relevantes [24] de Keycloak para el caso que nos ocupa pueden mencionarse los distintos protocolos con los que es compatible, entre otros, SAML, OpenID Connect, CAS [25], WS-Federation2 [26], Oauth 2.0 y Kerberos. Otras características reseñables son las siguientes:

- **Adaptador Java** [27]: librería que permite a cualquier aplicación Java interactuar con Keycloak de forma transparente, independientemente del protocolo (lo cual supone una gran ventaja a la hora de implantar sistemas con autenticación por Keycloak, pues solo hay que ocuparse configurar Keycloak y el adaptador).
- **SSO**: permite tanto «Single Sign On» como «Single Log Out» en aplicaciones web.
- **Federación de usuarios**: permite sincronizar usuarios desde LDAP o desde servidores de Active Directory.
- **Flujos de autenticación**: ofrece gran flexibilidad a la hora de definir flujos con funciones adicionales tales como recuperación de contraseñas, cambio periódico y obligatorio de contraseñas, etc.
- **Admin REST API**: proporciona una API para administrar el servidor.
- **Autenticación en dos pasos:** es compatible tanto con TOTP como HOTP
- **Kerberos Bridge**: permite que los usuarios ya autenticados por Kerberos iniciar una sesión SSO convencional (lo cual resulta especialmente ventajoso en redes corporativas donde Kerberos se usa con carácter general).

2.2.2 OpenAM

Figura 6: Logo de OpenAM

OpenAM (Open Access Management) es un sistema de código abierto desarrollado en el lenguaje de programación Java y publicado bajo licencia CDDL que permite gestionar procesos de autenticación en servicios web. Surgió en febrero de 2010 como un fork de OpenSSO (sistema de control de acceso creado por Sun Microsystems, compañía adquirida posteriormente por Oracle) de la mano de la empresa ForgeRock, que lo mantuvo con carácter abierto y gratuito hasta 2016, cuando optaron por una versión de pago de código cerrado, ForgeRock Access Management. Siguen no obstante existiendo versiones de código abierto y gratuitas, siendo el más popular la desarrollada por Open Identity Platform [7], cuya versión más reciente, la 14.4.1, fue publicada en julio de 2019.

OpenAM es compatible con los protocolos SAML, OpenID Connect y OAuth 2.0, así como con un sistema modular en el que se integran veinte métodos diferentes de autenticación.

Otras características reseñables son las siguientes:

- **SSO**: permite «Single Sign On» en aplicaciones web.

- **Sistema modular de autenticación**: los flujos de autenticación están compuestos por módulos cuyo orden se puede configurar y que siguen el estándar abierto JAAS (Java Authentication and Authorization Service), lo que facilita su personalización.
- **Autenticación adaptativa en función del riesgo**: módulo que, en función de distintos factores de riesgo (rango de la IP de origen, número de intentos de acceso, carácter desconocido del dispositivo de origen, etc.), aplica unos u otros módulos de autenticación.
- **XACML** para configurar las políticas de autorización.

2.2.3 CAS

CAS es un sistema de código abierto desarrollado principalmente en el lenguaje de programación Java y publicado bajo licencia Apache 2.0 que permite gestionar la verificación de la identidad de los usuarios que se conectan a un servicio. Está desarrollado por Apereo Foundation y surgió de forma paralela al protocolo CAS, del que constituye la implementación de referencia. Tiene como base el conjunto de librerías Spring Framework (más concretamente los proyectos Spring Boot y Spring Cloud) [28] y presenta un gran número de características que facilitan su integración en sistemas complejos.

Figura 7: Logo de CAS

CAS es compatible con los protocolos CAS, SAML 2.0, OpenID, OpenID Connect, OAuth 2.0, WS-Federation y REST. Otras características reseñables son las siguientes:

- **SSO**: permite «Single Sign On» en aplicaciones web.
- **Múltiples mecanismos 2FA**: como OTP, U2F o Yukibey.
- **Múltiples adaptadores**: soporte oficial que permite la conexión sencilla al servidor mediante librerías compatibles con mútiples lenguajes y plataformas, tales como Java, Apache, PHP o .NET.
- **Integración nativa en multiples aplicaciones:** Un número de aplicaciones permiten realizar una conexión de forma trivial a CAS. Por ejemplo Moodle[29] o Tikiwiki [30]

2.3 Frameworks empleados en el desarrollo del prototipo

2.3.1 Spring Framework

Spring Framework [8] es un conjunto de librerías de código abierto orientado al desarrollo de aplicaciones en Java y publicado bajo licencia Apache 2.0. Está desarrollado por la empresa Pivotal Software, su primera versión estable data de 2004 y la más reciente, 5.2.5.RELEASE, fue publicada el 24 de marzo de 2020 [31].

Figura 8: Logo de Spring

El framework surgió con la llegada de los EJB (Enterprise Java Beans) a la plataforma Java EE y emplea una serie de conceptos novedosos para la época de su aparición como AOP (Aspect-Oriented Programming o AOP), POJO (Plain Old Java Object) o DI (inversión de dependencias) con objeto de simplificar el desarrollo de aplicaciones basadas en EJB [32].

Spring Framework es uno de los frameworks de programación Java más populares entre los desarrolladores de aplicaciones web [33]. Durante muchos años ha liderado la innovación en el ecosistema java y se ha transformado en un conjunto de librerías seguro, maduro y de referencia entre grandes y pequeñas empresas.

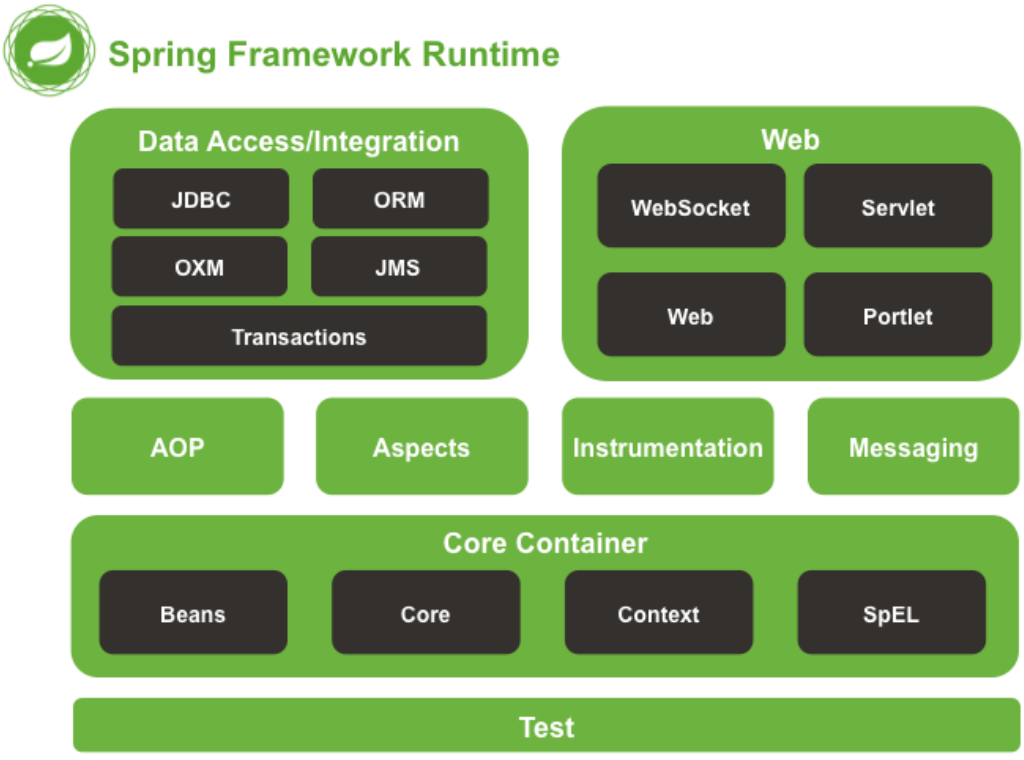


Figura 9: Arquitectura de Spring Framework [56]

2.3.1.1 Características principales de Spring Framework

Entre las características más relevantes de Spring Framework para el caso que nos ocupa pueden mencionarse las siguientes.

2.3.1.1.1 Contenedor de IoC

El «Inversion of control container» (IoC Container) [34] es un framework que permite realizar inyección de dependencias en las aplicaciones, posibilidad hoy en día muy común en la mayoría de los frameworks sobre lenguajes de programación orientados a objetos pero originalmente una de las novedades introducidas por Spring Framework.

2.3.1.1.2 Framework de acceso a datos

Este framework aporta una capa sobre JDBC o la librería ORM (Object relational mapping) Hiberante que facilita ciertos aspectos de la conexión a la base de datos de la aplicación y la persistencia de la información dentro de esta.

2.3.1.1.3 Spring MVC Framework

Este framework permite desarrollar la capa de vista de una aplicación web utilizando la llamada arquitectura MVC (Modelo Vista Controlador). Por defecto, las plantillas HTML se desarrollan empleando el framework Thymeleaf, aunque se pueden escoger otras opciones [35]

2.3.1.1.4 Spring Boot

Spring Boot es uno de los subproyectos disponibles de forma gratuita dentro del ecosistema Spring. El proyecto proporciona Spring framework en un único paquete y con una configuración por defecto que trata de dar respuesta a las expectativas del grueso de los desarrolladores [36] con objeto de simplificar el trabajo de configuración y la fase de ensayo y error tradicionalmente asociados al desarrollo en Spring. Asimismo incluye una serie de

dependencias que agilizan dicho desarrollo, como por ejemplo un servidor Jetty embebido que permite al desarrollador trabajar sin necesidad de instalar y mantener un entorno de desarrollo con un contenedor de aplicaciones independiente.

Otra característica reseñable de Spring Boot es que elimina la necesidad de generar el llamado código «boilerplate» o ficheros XML para su correcto funcionamiento. Como consecuencia, la mayor parte del código de un proyecto que usa Spring Boot es lógica de negocio o pequeñas clases de configuración para los distintos componentes. Además, Spring Boot incluye por defecto características que todo proyecto requiere al pasar a producción, como health checks, repositorios de configuración, «endpoints» con métricas de consumo de memoria, CPU, etc.

2.3.1.2 Autenticación y seguridad

El *framework* de seguridad que se emplea de manera estándar en las aplicaciones desarrolladas en Spring Framework se denomina Spring Security [37], el cual es compatible con diversos métodos de autenticación, como CAS, SAML 2.0, OpenID, Open ID Connect y otros métodos de autenticación no estándar basados en OAuth 2.0 como GitHub[4], autenticación mediante usuario y contraseña (para lo que emplea distintos repositorios de uno y de otra) o X509 Authentication.

En este último caso, puede configurarse el servidor web (o un balanceador de carga) para llevar a cabo «2-way SSL», de manera que el servidor verifique la validez de los certificados que los clientes le proporcionen. Tras la verificación, el certificado se envia a modo de parámetro y es recibido por un filtro que extrae la información relativa a la identidad del usuario.

El uso de Spring Security como framework de seguridad para Spring Framework presenta considerables ventajas, entre las que destacan las siguientes:

- El funcionamiento de los distintos elementos del framework de seguridad sigue los mismos patrones de diseño y paradigmas que los del framework empleado en el desarrollo, lo que aumenta la inteligibilidad del código.
- La configuración de ambos frameworks es similar (por ejemplo, ambos emplean el fichero «application.yml», los «Beans» de configuración, etc.).
- Los elementos que intervienen en las distintas capacidades de Spring Security se testan en un entorno y con unas herramientas similares a las que se usan en el desarrollo, lo que disminuye la probabilidad de incompatibilidades y problemas de dependencias.
- Es posible adoptar distintas estrategias a la hora de gestionar sesiones; por ejemplo, pueden almacenarse en distintos repositorios (bases de datos, memoria caché compartida entre distintos elementos de un cluster, etc.). Esta gestión es siempre segura, por defecto; así, por ejemplo, después de que un usuario se autentique con éxito el identificador de la sesión cambia, lo que lo protege frente a ataques de fijación de sesiones («Session fixation attacks»).
- La configuración de los diferentes medios de protección que suelen estar presentes hoy en día en las aplicaciones web (como CSRF, cabeceras CORS, etc.) se puede realizar de forma sencilla gracias a los distintos mecanismos de configuración y valores por defecto que el framework ofrece.

2.3.2 Play Framework (Play 2)

Play Framework es un framework desarrollado en el lenguaje de programación Scala, pero puede utilizarse con otros lenguajes basados en Java Virtual Machine (JVM), entre ellos, obviamente, Java. Nació como proyecto interno de la empresa francesa Zengularity S.A. en torno al año 2007, si bien en la actualidad su desarrollo corre a cargo fundamentalmente de la empresa Lightbend, que también le proporciona soporte comercial,

Figura 10: Logo de Play

4 Spring Security define este tipo de métodos de manera genérica como «OAuth Login».

y su primera versión data de mayo de 2008. La versión 2.0, de 2012, introdujo cambios importantes tales como el empleo del sistema de construcción de proyectos SBT en lugar de scripts. La versión más reciente es la 2.8, publicada en diciembre de 2019.

Play Framework se promociona como un framework de arquitectura ligera, stateless y web-friendly [38]y se encuentra claramente inspirado en otros frameworks basados en el paradigma de diseño «convention over configuration», tales como Ruby on Rails (para el lenguaje de programación Ruby) o Django (para Python).

2.3.2.1 Características principales de Play Framework

Entre las carácticas más relevantes de Play Framework para el caso que nos ocupa pueden mencionarse las siguientes.

2.3.2.1.1 Soporte nativo de Java y Scala

La versión de Play Framework Play 1.x se hallaba desarrollada en el lenguaje de programación Java y disponía de un módulo separado que lo hacía compatible con el lenguaje Scala. Sin embargo, la versión Play 2.x ya incorpora Scala dentro del propio framework. En la actualidad es posible utilizar Play Framework tanto con un lenguaje como con el otro (por ejemplo, la página web oficial del framework ofrece recursos para su desarrollo con ambos).

2.3.2.1.2 Programación Asíncrona

La arquitectura de la versión de Play Framework Play 2 está diseñada sobre la premisa de que cualquier solicitud puede dar lugar a una conexión persistente (por ejemplo mediante el uso de WebSockets). Play 2 es compatible de forma nativa con Akka [39], una implementación del «modelo de actor» para lenguajes basados en JVM que permite programar sistemas distribuidos.

2.3.2.1.3 SBT como herramienta de construcción de proyectos

La versión de Play Framework Play 2 emplea la herramienta de construcción de proyectos («build system tool») SBT [40] herramienta de referencia para la construcción de proyectos en el lenguaje de programación Scala. SBT es similar tanto en lo que respecta a su uso como en lo que se refiere a sus características a Gradle.

2.3.2.1.4 Naturaleza modular

El núcleo de Play Framework consiste simplemente en un servidor HTTP ligero equipado con un enrutador y una serie de mecanismos que permite gestionar solicitudes HTTP. Ahora bien, el framework cuenta asimismo con una amplia variedad de módulos creados por su comunidad de desarrollo que multiplican sus funcionalidades [41]; además, es posible desarrollar módulos personalizados que añadan funcionalidades relevantes concretamente para el proyecto en el que se está trabajando.

2.3.2.1.5 Recarga automática («hot-reloading»)

Play Framework dispone de un sistema de recarga automática de la aplicación en modo «desarrollo». Este sistema detecta cuándo se han realizado cambios en el código y vuelve a desplegar la aplicación de forma automática, lo que permite al desarrollador recibir *feedback* de forma mucho más sencilla y simplifica el entorno de desarrollo utilizado.

2.3.2.2 Autenticación y seguridad en Play Framework

La versión de Play Framework Play 2 es compatible de forma nativa con OpenID y con OAuth, pero existen además librerías de código abierto que amplían las funcionalidades del framework de tal manera que se posibilita su empleo con otros protocolos. En el presente trabajo analizaremos una de estas librerías, PAC4J, que es específica específica para la versión de Play Framework Play 2 [41]. PAC4J es un conjunto de librerías desarrollado por la empresa «CAS in the Cloud» [42] junto con voluntarios de la comunidad y compatible con distintos frameworks y herramientas basadas en el lenguaje de programación Java.

PAC4J es compatible, entre otros, con los siguientes métodos de autenticación: OAuth, SAML, CAS, OpenID Connect, HTTP, Google App Engine, Kerberos y autenticación mediante usuario y contraseña (para lo que emplea numerosos repositorios tales como SQL, MongoDB o LDAP). Asimismo permite gestionar una gran cantidad de mecanismos de protección, entre los que se encuentran CORS, protección contra CSRF («Cross-site request forgery») o mecanismos de autorización incluyendo gestión de permisos y roles.

2.3.3 Quarkus Framework

Quarkus Framework es un framework desarrollado en el lenguaje de programación Java y publicado bajo licencia Apache 2.0 que se orienta fundamentalmente en el desarrollo de aplicaciones en la nube o «cloud native», de microservicios y de aplicaciones «serverless». [43]. Está desarrollado por la comunidad de desarrolladores, si bien recibe apoyo corporativo fundamentalmente por parte de la empresa RedHat. Se trata del framework más reciente de los analizados en el presente trabajo, pues su primera versión, la 1.0.0.Final, apareció en noviembre de 2019; la más reciente, que data de abril de 2020, es la 1.4.0.Final.

Figura 11: Logo de Quarkus

En la práctica Quarkus Framework es un framework que envuelve un conjunto de librerías empleadas de manera frecuente por los desarrolladores en proyectos web y permite desplegar dichas librerías en contenedores de alto rendimiento tales como GraalVM o OpenJDK Hotspot. Además, el framework presenta un gran número de utilidades que facilitan el desarrollo de lógica de negocio en el mismo.

2.3.3.1 Características principales de Quarkus Framework

Entre las carácticas más relevantes de Quarkus Framework para el caso que nos ocupa pueden mencionarse las siguientes.

2.3.3.1.1 Experiencia de desarrollo

Quarkus Framework presenta toda una serie de características que facilitan de manera considerable el desarrollo de aplicaciones. Por ejemplo, es compatible con la función «live reload», que permite comprobar el efecto de los cambios realizados en la aplicación en tiempo real, sin necesidad de compilarla y despegarla de forma manual, que es el método habitual en el caso de aplicaciones desarrolladas en Java. Además, el framework aglutina la totalidad de la configuración de los distintos elementos que lo componen en un único fichero y permite generar un fichero ejecutable nativo directamente con un comando.

Otro ejemplo de la atención que presta Quarkus Framework a la experiencia de los desarrolladores que lo emplean es la existencia de extensiones que permiten su compatibilidad con otros frameworks de programación. Así, la extensión «spring-di» permite el uso de anotaciones de tipo Spring (como por ejemplo la anotación «@Autowired», que permite la inyección de una dependencia en una clase) en lugar de las anotaciones propias del framework de inyección de dependencias que se emplea por defecto en Quarkus, que es CDI (como por ejemplo «@Inject»); además, la conversión de un tipo de anotaciones al otro se lleva a cabo de manera eficiente, esto es, manteniendo el consumo de memoria y de potencia de procesamiento al mínimo.

2.3.3.1.2 Alto rendimiento y reducido consumo de recursos [44]

Quarkus Framework emplea una serie de estrategias que optimizan su rendimiento al reducir el consumo de memoria que requieren para su funcionamiento y minimizar el tiempo de arranque de la aplicación desarrollada sobre su framework, a saber:

- **Prioridad a GraalVM/Hotspot como máquinas virtuales Java.**
  Quarkus Framework emplea preferentemente como máquinas virtuales Java las máquinas GraalVM/Hotspot, que incluyen distintas técnicas enfocadas a mejorar el rendimiento y el tiempo de iniciación de las aplicaciones desarrolladas en Java, tales como la compilación «just-in-time» [45] o la optimización adaptativa [46].

- **Preprocesamiento de metadatos en el momento de compilación**
  Quarkus Framework trata de reducir lo máximo posible el número de clases cargadas en la máquina virtual en cada momento, de manera que lo estén única y exclusivamente aquellas necesarias en tiempo de ejecución. Por ejemplo, en los *frameworks* convencionales las clases necesarias únicamente para el despliegue de la aplicación permanecen cargadas en la máquina virtual en muchos casos, mientras que en Quarkus Framework esto no ocurre, lo que disminuye el uso de memoria y aumenta la velocidad de arranque de la aplicación.
- **Reducción del uso de reflexión.**
  La reflexión es una técnica que permite invocar clases en tiempo de ejecución. Es una técnica que se emplea de manera frecuente en la programación en Java y que puede provocar problemas de rendimiento, puesto que implica que todas las clases susceptibles de ser invocadas en tiempo de ejecución deben estar presentes en el empaquetado, lo que aumenta tanto el tamaño de la aplicación como su consumo de memoria. Quarkus Framework hace un uso limitado de esta técnica, reduciendo así los desventajas que conlleva.

2.3.3.2 Autenticación y seguridad en Quarkus Framework

Debido a lo reciente de su aparición, Quarkus Framework solo dispone en la actualidad de extensiones que permiten su compatibilidad con mecanismos que proporcionan servicios de identidad populares en el caso de los microservicios, a cuyo desarrollo se encuentra fundamentalmente orientado [47]. Son los siguientes: OpenID Connect, Hashicorp Vault, JDBC (autenticación a través de credenciales almacenadas en la base de datos y autenticación mediante ficheros de propiedades (mecanismo de autenticación básico en el que las credenciales de acceso se incluyen en uno de estos ficheros).

A la hora de facilitar su nombre de usuario y contraseña, el usuario puede sencillamente usar el método de autenticación «Basic authentication» para proporcionar las credenciales mediante una cabecera en la solicitud, o bien llevar a cabo un proceso de autenticación similar al de cualquier aplicación web, ya sea mostrando su propio formulario de autenticación o delegando esta a un proveedor de identidad de OpenID Connect.

2.4 Conclusiones parciales (análisis)

2.4.1 Estándares

Del análisis de los tres estándares de proporción de servicios de identidad considerados en el presente trabajo se desprende que el primero de ellos, SAML (el estándar más antiguo de los analizados, y más concretamente su última versión, la 2.0 de 2005), es un estándar complejo pero notablemente asentado, dotado de madurez y caracterizado por un nivel de seguridad considerablemente elevado. Cabe asimismo señalar que son numerosas las empresas que a día de hoy continúan empleando programas antiguos únicamente compatibles con SAML, de manera que también siguen utilizando este estándar, que constituye una opción conservadora y fiable. Ahora bien, la mencionada complejidad de SAML 2.0 implica que tanto su correcta configuración como el posterior mantenimiento de los certificados digitales necesarios para su funcionamiento puede suponer tanto un esfuerzo como un desafío. Los protocolos de SAML 2.0 definen además mensajes muy pesados y requieren por tanto la transmisión de ficheros XML de gran tamaño, lo que puede suponer un problema en los casos en los que el ancho de banda de la red a través de la que se produce el intercambio sea limitado.

El segundo de los estándares analizados, OpenID Connect, es un estándar relativamente joven (su primera versión data de hace solo seis años) cuyas principales ventajas son el uso que hace de tecnologías modernas como JSON-OAuth2 y lo sencillo que resulta su integración con aplicaciones móviles. Este estándar ofrece un nivel de seguridad bastante elevado, no requiere (gracias al empleo del intercambio de metadatos de tipo dinámico frente al estático) un mantenimiento excesivamente laborioso y permite autenticar proveedores de identidad mediante un método más sencillo que el utilizado por el anterior estándar SAML (firma de ficheros XML), a saber, el uso de claves simétricas (denominadas «client secret» en las especificaciones de OpenID Connect). Además, OpenID Connect es, de todos los estándares analizados en este trabajo, el único compatible de forma nativa con más de uno de

los *frameworks* que se han empleado en el desarrollo del prototipo utilizado para ilustrar el funcionamiento de uno de estos estándares: Spring y Quarkus. No obstante, es probable que, en el ámbito empresarial, el estándar de uso preferente siga siendo SAML hasta que dejen de utilizarse aplicaciones únicamente compatibles con este estándar.

Por lo que se refiere a CAS, se trata de un estándar de proporción de servicios de identidad que destaca por lo fácil de su configuración y por su gran sencillez de funcionamiento, uso y administración. El nivel de seguridad que proporciona es limitado: así, por ejemplo, la seguridad de los mensajes involucrados en el proceso de autenticación está principalmente basada en la encriptación del canal de transmisión mediante el protocolo TLS y únicamente permite la encriptación de aquellos atributos del mensaje de naturaleza confidencial, como pueden ser las credenciales de acceso del usuario [48] (por el contrario, los mecanismos de encriptación usados por los dos primeros estándares, SAML 2.0 y OpenID Connect, permiten la encriptación de todos los atributos incluidos en el mensaje). Además, el análisis de los proveedores de identidad considerados en este trabajo (Keycloak, OpenAM y Apereo CAS) ha revelado que solo es compatible de forma nativamente con este último (mientras que SAML 2.0 y OpenID lo son con los tres), lo que podría dificultar su adopción como estándar de proporción de servicios de identidad dentro del ámbito empresarial.

2.4.2 Proveedores de identidad

Por lo que se refiere a los proveedores de identidad, cabe destacar la amplia variedad de opciones disponibles a la hora de elegir un servicio de cara a la gestión de identidad federada dentro del ámbito empresarial y capaces de responder a las necesidades más frecuentes en el uso de tales servicios, como es el caso de la interacción de los usuarios con distintas aplicaciones mediante Single Sign On. En este caso, cualquiera de los tres sistemas evaluados podrían haber cubierto las expectativas para el desarrollo del prototipo.

El primero de los proveedores de identidad analizados ha sido Keycloak, un servicio que cuenta con el respaldo de una corporación como Red Hat, una de las mayores empresas multinacionales proveedoras de servicios informáticos y uno de los exponentes más importantes del mundo del software libre, circunstancia relevante, al menos a priori, de cara a la valoración de aspectos de indudable interés a la hora de elegir un servicio que implementar dentro de la infraestructura de una empresa, tales como la calidad del producto en sí, el soporte técnico con el que dicho producto cuenta o el futuro mantenimiento del servicio una vez implementado, pues supone cierta garantía respecto a todos ellos. Por lo que se refiere a las funcionalidades de Keycloak en sí mismo, cabe destacar la opción «User Federation», que permite integrar el proveedor con varios directorios de usuarios de uso común en el ámbito empresarial como son el Lightweight Directory Access Protocol (LDAP) o el Active Directory de Microsoft.

En segundo lugar se analizó el proveedor de identidad Apereo CAS, el más antiguo de los servicios considerados en el presente trabajo, que sin embargo cuenta con una gran comunidad de usuarios y de desarrolladores, así como con un considerable ecosistema de empresas que contribuyen a su soporte, todo lo cual supone cierta garantía de cara al futuro mantenimiento del servicio, además de entrañar la existencia de una gran cantidad de documentación e información sobre el mismo disponible en la red. elegir Por lo que se refiere al proveedor en sí, una de las características más relevantes de Apereo CAS en relación con su empleo en la gestión de identidad federada en el ámbito empresarial es el hecho de que se encuentra integrado de forma nativa con múltiples servicios, como por ejemplo la herramienta de gestión de aprendizaje virtual Moodle o el sistema de gestión de contenidos colaborativo Tikiwiki, de manera que la elección de Apereo como proveedor de identidad en empresas que utilicen alguno de estos servicios supondría un importante ahorro de tiempo respecto a otros proveedores que carezcan de esta integración nativa.

Por último se ha analizado el proveedor de identidad OpenAM, un servicio que, si bien sería capaz de responder a las necesidades que plantea la gestión de la identidad federada de los usuarios de una empresa en general y de hecho cubre las funcionalidades necesarias para la creación del prototipo desarrollado en el presente trabajo, presenta algunos problemas que se derivan de los limitado de su actual implantación (se trata de un servicio relativamente joven) y de su comunidad de usuarios y desarrolladores: no ha sido posible, por ejemplo, localizar ninguna empresa que preste soporte comercial al producto y el reducido número de desarrolladores involucrados activamente en el proyecto en la actualidad supone cierta

incertidumbre respecto al desarrollo y el mantenimiento del servicio en el futuro. No obstante, el hecho de que OpenAM ofrezca la posibilidad de adaptar el proceso de autenticación de los usuarios en función de distintos factores de riesgo involucrados en dicho proceso constituye una funcionalidad muy atractiva en el ámbito que nos ocupa, donde la seguridad de las aplicaciones utilizadas puede resultar vital para la supervivencia de una empresa.

Aunque en el caso que nos ocupa cualquiera de los tres proveedores examinados podría haberse empleado en el desarrollo del prototipo que permitirá estudiar su funcionamiento en diferentes frameworks, finalmente se escogió el segundo de los proveedores de servicios de identidad federada analizados, esto es, Keycloak, por tratarse de una opción muy sólida, ya que es compatible con buen número de protocolos, cuenta con el apoyo económico y técnico de una gran empresa y la literatura sobre su configuración, uso y mantenimiento es no solo abundante sino también actualizada.

2.4.3 Frameworks

Por lo que se refiere al primero de los frameworks analizados en el presente trabajo, Spring Framework es uno de los frameworks de programación en Java más populares para el desarrollo de aplicaciones web [33]. Además, ha liderado durante años la innovación dentro de dicho lenguaje y con el tiempo se ha convertido en un conjunto de librerías seguro, maduro y de referencia tanto para grandes como para pequeñas empresas.

Entre las mayores ventajas del segundo framework analizado, Play Framework, se encuentran su aproximación asíncrona por defecto y el elevado número de características que presenta que facilitan el desarrollo de proyectos empleando Play Framework.

En cambio, el framework no es compatible de forma nativa con estándares de autenticación modernos como OpenID Connect, lo que implica que para trabajar con Play Framework y este tipo de estándares es necesario depender de librerías de código de terceros que permitan su compatibilidad. También son incompatibles las versiones de Play Framework Play 1 y Play 2, lo que puede suponer una desventaja a la hora de emplear el framework en un ámbito corporativo, pues las empresas suelen tender a favorecer las soluciones que garantizan cierta estabilidad y un fácil mantenimiento sobre aquellas que ofrecen nuevas características. Cabe asimismo mencionar que, si bien Play Framework es efectivamente compatible con el lenguaje de programación Java, la documentación del framework deja traslucir que está orientado fundamentalmente al lenguaje Scala; de hecho, ciertas secciones del framework se encuentran disponibles única y exclusivamente en dicho lenguaje, como ocurre por ejemplo con el sistema de plantillas Twirl [49].

En cualquier caso, a pesar de que en los últimos años ha perdido popularidad, Play Framework continúa siendo el principal framework para el desarrollo de aplicaciones en el lenguaje de programación Scala.

Por último, Quarkus es un framework de muy reciente creación que, si bien según su documentación ya es apto para producción, todavía se encuentra en desarrollo activo, lo que podría suponer una desventaja a la hora de emplearlo en un ámbito corporativo, pues podría inducir a algunas empresas a descartar su adopción.

Ahora bien, hay también que señalar que las librerías de código que constituyen la base de este framework son librerías muy maduras que se emplean en la actualidad en la mayoría de los proyectos web (además, Quarkus Framework se orienta fundamentalmente al desarrollo de microservicios y, en el caso de que en el futuro surgieran problemas relacionados con su framework, reemplazar este tipo de servicios es siempre relativamente sencillo). Por otra parte, la experiencia de desarrollo que ofrece Quarkus Framework es prácticamente impecable y el empleo del framework incrementa de forma considerable la velocidad de desarrollo, pues Quarkus emplea tecnologías que todo desarrollador con experiencia en el ecosistema del lenguaje Java conoce, lo que reduce sustancialmente la curva de aprendizaje.

3. Desarrollo del prototipo

El objetivo de este prototipo es ilustrar, a través de tres sencillas aplicaciones que simulan servicios internos de una empresa, el funcionamiento de una de las posibles soluciones para la adopción de servicios de identidad federada privados en el ámbito empresarial en tres *frameworks* diferentes. Por tanto, el desarrollo de este prototipo incluirá tanto la programación de las tres aplicaciones como la integración de la solución seleccionada (más concretamente el combo Keycloak + OpenID Connect) con cada uno de esos *frameworks*.

Como estándar de proporción de servicios de identidad federada hemos decidido emplear OpenID Connect debido a que es compatible con la mayor parte de los *frameworks* de desarrollo web modernos y a lo sencillo que resulta configurar una conexión entre la aplicación y el proveedor de identidad gracias al protocolo Discovery. Como ya comentamos en el apartado anterior, el hecho de que OpenID Connect sea un estándar relativamente joven puede dar lugar a problemas a la hora de integrarlo con programas antiguos que solo sean compatibles con otros estándares (por ejemplo SAML), pero consideramos que la funcionalidad «Identity Brokering» [50] del proveedor de identidad seleccionado disminuye la gravedad de este posible problema, pues permite interconectar fácilmente aplicaciones con OpenID Connect y SAML 2.0

Por lo que se refiere al proveedor de identidad, de entre los analizados hemos optado por Keycloak, pues, a pesar de que todos los proveedores examinados habrían permitido desarrollar las funcionalidades del prototipo, Keycloak cuenta con soporte comercial por parte de una empresa muy conocida como es Redhat, es muy sencillo de configurar y pone a disposición del desarrollador una gran cantidad de información actualizada sobre su uso y funcionamiento.

Para desarrollar el *frontend* de las aplicaciones se han utilizado los motores de plantillas recomendados para cada *framework*, esto es, Thymeleaf en el caso de Spring y Twirl en el caso de Play 2 (Quarkus, al ser un *framework* orientado fundamentalmente al desarrollo de microservicios, no requiere *frontend*).

3.1 Aplicación de gestión vacaciones

La aplicación de gestión de vacaciones permite a un usuario con el rol «user» solicitar vacaciones con una fecha inicial y una fecha final y a un usuario con el rol «admin» acceder a dicha solicitud y realizar una de dos posibles acciones sobre la misma, a saber, aprobarla o denegarla. Esta aplicación se ha desarrollado con Spring Framework.

En primer lugar se ha configurado un cliente de OpenID Connect en Keycloak tal y como se describe en el Anexo 1. por otra, se implementó una versión propia de «userAuthoritiesMapper» encargada de extraer el «claim» llamado «roles» y añadirlos como «authorities» para la aplicación. Esto último ha sido necesario ya que Spring Security relaciona de forma automática los «scopes» devueltos por el OP con las «authorities» que se usan para limitar el acceso del usuario a ciertas funcionalidades de la aplicación y en este caso queríamos, en aras de la homogeneidad, utilizar los roles asignados a los usuarios en Keycloak.

La conexión de Spring Boot a OpenID Connect es extremadamente sencilla, ya que solo es necesario incluir en el fichero «application.yml» la información presente en el Ejemplo 19

Hemos escogido configurar una clave simétrica (client secret) para el OP y la aplicación que se utilizará a la hora de autenticar la aplicación en el OP y hemos proporcionado una URL que la aplicación utilizará para acceder a la configuración del OP, tal y como se define en la especificación OpenID Connect Issuer Discovery [19].

Para proteger los distintos recursos de la aplicación se han empleado tres estrategias fundamentalmente:

- La integración de Thymeleaf y Spring Security, que permite fácilmente mostrar u ocultar información en función del rol del usuario autenticado, así como extraer información acerca este (como por ejemplo su nombre)

- El empleo de anotaciones @PreAuthorize o @PostAuthorize para limitar el acceso del usuario a los distintos «endpoint» en función de su rol (por ejemplo, solo un usuario con el rol «admin» puede acceder al «endpoint» de aprobación o denegación de solicitud).

- La extracción de información desde el IDToken dentro de la lógica de negocio, de manera que un usuario con el rol «admin» pueda visualizar todas las solicitudes de vacaciones pero un usuario con el rol «user» solo pueda visualizar las propias.

Finalmente, se pueden observar ejemplos de la interfaz de la aplicación en Ejemplo 20 y Ejemplo 21.

3.2 Aplicación de gestión de pedidos

La aplicación de gestión de pedidos permite a un usuario con el rol «user» crear pedidos y visualizarlos; el usuario con el rol «admin», además de poder realizar las acciones anteriores, puede visualizar pedidos de otros usuarios y crear nuevos productos. Esta aplicación ha sido desarrollada con Play 2 en conjunción con la librería de seguridad PAC4J en su variante para Play Framework (play-pac4j [50]).

En el desarrollo de la aplicación se utilizó como base una aplicación de ejemplo proporcionada por la empresa Lightbend y consistente en una serie de tutoriales y el envío de un formulario, la cual se amplió y adaptó de tal forma que fuera capaz de llevar a cabo las funcionalidades establecidas al inicio del presente trabajo, a saber, permitir a los usuarios realizar pedidos y permitirle a la empresa generar nuevos productos. A continuación se configuró la conexión entre la aplicación y el proveedor de identidad Keycloak y se definieron los usuarios y roles de la misma siguiendo los pasos detallados en el define en el Anexo 1, y se añadió a la aplicación la dependencia a la librería play-pac4j.

Por lo que se refiere al desarrollo y la configuración del control del acceso de los usuarios a la aplicación y del control del grado autorización de los mismos dentro de ella, al tratarse PAC4J de una librería independiente al framework utilizado fue necesario implementar un módulo de seguridad. En Play Framework los módulos permiten definir extensiones del framework [52] y así añadirle a este nuevas funcionalidades, en este caso, funcionalidades orientadas al control del acceso y el grado de autorización de los usuarios en la aplicación.

Más concretamente, en este módulo, que se configuró en la clase SecurityModule, se definieron los parámetros de la conexión entre la aplicación y el proveedor de identidad a través del objeto OidcConfiguration; se configuraron el identificador de la aplicación («client-id»), la clave simétrica («client-secret») y el «Discovery URL»; se mapeó desde el «claim» roles a las autorizaciones al definir un AuthorizationGenerator (Ejemplo 24). y se crearon dos controles de autorización o «Authorizers» que definen el grado de acceso de los usuarios, uno aplicable a los dos roles disponibles, el de usuario y el de administrado («any-role»), y otro únicamente a los administradores («admin») (el uso de estos «Authorizers» en la protección de los «endpoints» mediante la anotación @Secure se ilustra en el Ejemplo 22.

De acuerdo con las instrucciones de PAC4J, además de este módulo se implementó un un filtro de seguridad («SecurityFilter») con objeto de cribar todas las conexiones entrantes y permitir que la aplicación tome decisiones sobre el grado de acceso de un usuario a sus recursos ofrecidos en función de factores como si tiene o no permiso para acceder a una determinada página web o si se encuentra o no autenticado.

Para limitar el acceso de los usuarios a los recursos ofrecidos por la aplicación se empleó el concepto de «profile», que dentro de PAC4J consiste en un contenedor en el que se almacena la información del usuario extraída de un proveedor de identidad [53] en el contexto de PAC4J es un contenedor de información del usuario extraída de un IdP. Este «profile» se inyectó tanto en las vistas (donde permite, entre otras acciones, mostrar el nombre de usuario (ver Ejemplo 23) u ocultar o no la opción de creación de nuevos productos en función del rol del usuario autenticado) como en la lógica de negocio (donde permite extraer el identificador del usuario para después almacenarlo con el pedido o filtrar la lista general de pedidos para mostrar únicamente los pedidos de un usuario concreto).

3.3 Aplicación de generación de informes

La aplicación de gestión de generación de informes permite a un usuario con el rol «user» generar resúmenes en formato PDF de los pedidos realizados por dicho usuarios y a un usuario con el rol «admin» generar resúmenes de cualquier pedido (ver Ejemplo 27); en ambos casos, el usuario inicia la solicitud de generación del informe mediante un enlace desde la aplicación de pedidos, lo que provoca que el navegador realice una solicitud HTTP, con el identificador único del pedido como parámetro a la aplicación de informes. Esta aplicación ha sido desarrollada con Quarkus.

El primer paso en desarrollo de esta aplicación consistió en configurar la conexión entre la aplicación y el proveedor de identidad Keycloak siguiendo un proceso similar al descrito en el Anexo 1. A continuación se añadió la extensión que da soporte a OpenID Connect (oidc) y se ha completado la configuración imprescindible para realizar la conexión. En cuanto al mapeo del «claim» especial roles que hemos configurado dentro del ID Token como contenedor de los roles de usuario, se puede realizar de forma muy sencilla mediante la configuración de la propiedad: «quarkus.oidc.roles.role-claim-path», donde le hemos indicado la situación dentro del token JWT del listado de roles. Se pueden observar todos los detalles de configuración en el Ejemplo 25.

Por lo que se refiere a la toma de decisiones respecto a la aceptación o rechazo de la solicitud por parte de un usuario de generar un informe, esta se basa en la extracción de la información relativa a los usuarios del ID Token y en la valoración de sus roles («user» o «admin») y por tanto de si se encuentran o no autorizados para generar el informe de un determinado pedido. Para ello solo ha sido necesario inyectar en la lógica de negocios de la aplicación el objeto «SecurityIdentity», en el que se encuentra almacenada dicha información, mediante los métodos «getRoles()» y «getPrincipal()» (ver Ejemplo 26).

Por último y debido a que el identificador único escogido para conectar la aplicación de gestión de pedidos y la de generación de informes (esto es, el claim «sub») no se encuentra disponible en el objeto «SecurityIdentity», ha sido necesario inyectar el objeto «JSONWebToken» en la lógica de negocio de la aplicación de informes y a extraer el identificador único de dicho objeto.

3.4 Conclusiones parciales (prototipo)

El prototipo cuyo desarrollo se describe en este tercer apartado del presente trabajo tenía como objetivo estudiar el funcionamiento de la solución para la adopción de servicios de identidad federada seleccionada de entre todas las consideradas (en este caso el estándar OpenID Connect en conjunción con el proveedor Keycloak) en diferentes frameworks. En este subapartado se evalúa la experiencia de desarrollo de dicho prototipo y se detallan las dificultades que han surgido a la hora de crear las distintas aplicaciones (cada una de las cuales se ha desarrollado utilizando un framework diferente), haciendo especial hincapié en los aspectos relacionados con el proceso de autenticación y con la integración del proveedor de identidad con las tres aplicaciones mediante OpenID Connect.

En primer lugar cabe mencionar que, al desarrollar las tres aplicaciones, estas se han mantenido agnósticas respecto al proveedor de identidad, es decir, se han conectado con dicho proveedor, Keycloak, mediante opciones válidas para cualquier proveedor compatible con OpenID Connect (en el de caso Play mediante la librería PAC4J y en el de Spring y Quarkus a través del propio framework) en lugar de emplear un adaptador específico para este proveedor. Esta opción ha implicado que la conexión entre aplicación y proveedor resultara algo más compleja, pues fue necesario mapear el claim «roles» en cada uno de los frameworks para que estos pudieran identificar dichos roles, si bien uno de ellos, Quarkus, dispone de una propiedad de configuración que permite realizar el mapeo de forma automática. Sin embargo, se optó por este curso de acción para así dejar abierta la posibilidad de sustituir Keycloak de forma sencilla por cualquier otro proveedor de identidad compatible con OpenID Connect; además, se evitaba así la necesidad de configurar el adaptador en los tres frameworks.

Es también reseñable que el hecho de que uno de los frameworks (más concretamente Play 2) no sea compatible de forma nativa con el estándar de trabajo que se decida emplear (en este caso OpenID Connect) implica la necesidad de utilizar un elemento adicional, a saber, la librería PAC4J, la cual cuenta con una versión específica para Play 2 y permite utilizarlo con varios protocolos con los que no es compatible de forma nativa, como OpenID Connect. Ahora bien, esta librería ha supuesto un grato descubrimiento, puesto que dispone de una excelente documentación y cuenta con numerosos ejemplos de uso, de manera que, a pesar de que la conexión entre la aplicación (específicamente la de gestión de pedidos) y el proveedor de identidad ha supuesto un desarrollo adicional, el proceso ha sido sencillo y rápido.

Otro detalle de interés ha sido el comprobar que en ocasiones los tres frameworks utilizados en el desarrollo del prototipo abordan de manera diferente un mismo cometido: así, no todos ellos siguen la misma lógica a la hora de adjudicarle identificadores únicos a los usuarios de las aplicaciones, sino que cada uno adopta estrategias diferentes: Quarkus (que utiliza la librería Smallrye para leer los tokens JWT) utiliza como identificador, en función de su disponibilidad, primero «upn», a continuación «preferred_username» y por el último «sub»; mientras que Spring y Play (a través de PAC4J) solo emplean «sub». En este ejemplo, esta circunstancia nos ha hecho decantarnos por «sub» como «claim» en el que almacenar el identificador único de los usuarios, puesto que no contemplamos conectar las aplicaciones con proveedores de identidad adicionales a Keycloak (hay que tener cuenta que en este proveedor el claim «sub» es un «UUID» único generado por él mismo durante la creación del usuario).

En cambio, otras operaciones se llevan a cabo de manera similar en los tres frameworks, como es el caso a la hora de extraer información del perfil de un usuario en la interfaz de usuario o de bloquear mediante anotaciones el acceso de ciertos usuarios a determinados recursos dentro de la aplicación: de hecho, en ocasiones la única diferencia entre los tres es el tipo de anotaciones empleadas y, en el caso de Quarkus, existen extensiones que permiten utilizar en este framework anotaciones de otro como es Spring.

4. Conclusiones

El primer objetivo del presente trabajo consistía en llevar a cabo un estudio del estado de la cuestión en lo que respecta a las tecnologías utilizadas en el ámbito empresarial para la implementación de sistemas de gestión de identidad federada, con énfasis en los procedimientos de autenticación de tipo Single Sign On, que permite la autenticación de usuarios en distintas aplicaciones. Más concretamente, se han analizados tres opciones distintas tanto en lo que a estándares de proporción de servicios de identidad federada (SAML, CAS, OpenID Connect) como a lo que proveedores de identidad (Keycloak, CAS, OpenAM) se refiere. Asimismo se han analizado los tres frameworks (Spring, Play, Quarkus) posteriormente empleados en el prototipo desarrollado para estudiar el funcionamiento de las solución para la adopción de servicios de identidad federada escogida a tal efecto.

El análisis de los estándares de proporción de servicios de identidad federada ha revelado que SAML es un estándar maduro y seguro, pero también complejo y con algunas problemáticas añadidas (como el tamaño de sus ficheros XML); mientras que CAS es un estándar muy sencillo, pero menos seguro y únicamente compatible de forma nativa con uno de los proveedores de identidad considerados. OpenID Connect, por su parte, es un estándar tanto sencillo como seguro, además de compatible con un mayor número de los frameworks más modernos.

De este estudio comparativo se desprende que la opción más conveniente para el caso que nos ocupa es OpenID Connect, pues se trata de un estándar al mismo tiempo sencillo, moderno y seguro. Además, incluso si se diera la eventualidad de que alguna de las aplicaciones a conectar con el sistema de gestión de identidad fuera únicamente compatible con otro estándar como SAML, el empleo del proveedor Keycloak permitiría que la conexión de esa aplicación en concreto se realizara mediante dicho estándar.

El análisis de los proveedores de identidad ha puesto de manifiesto la existencia de una amplia gama de opciones capaces de cubrir las necesidades de uso de los servicios de gestión de identidad federada más frecuentes, incluido la autenticación mediante Single Sign On, así como la posibilidad de emplear cualquiera de los proveedores considerados en el desarrollo del prototipo propuesto y el hecho de que todos ellos cuentan con funcionalidades interesantes para el caso que nos ocupa, tales como la federación de usuarios, la compatibilidad nativa con servicios externos o la adaptación del proceso de autenticación en función del riesgo.

No obstante, este estudio comparativo ha permitido elegir como proveedor de identidad a usar en el desarrollo del prototipo Keycloak, ya que por una parte está mucho más asentado que OpenAM y, por otra, dispone de soporte técnico directo y de documentación adaptada a frameworks más modernos, dos características de las que CAS carece.

El análisis de los frameworks empleados en el desarrollo del prototipo ha evidenciado que Spring es un framework seguro, maduro y muy popular, mientras que Play es un framework muy utilizado en el desarrollo de aplicaciones en Scala pero incompatible con estándares de autenticación modernos, y Quarkus es un framework notablemente reciente que aún se encuentra en desarrollo, aunque teóricamente ya es apto para producción.

En segundo objetivo de este proyecto consistía en desarrollar el mencionado prototipo para el estudio del funcionamiento de la solución para la adopción de servicios de identidad federada (servicio de proporción de servicios de identidad + proveedor de identidad) escogida en función de los resultados obtenidos en la etapa de análisis. Dicho prototipo, una vez desarrollado, incluye tres aplicaciones (cada una de ellas con un framework distinto) que comparten un procedimiento de autenticación de usuarios de tipo Single Sign On centralizado en un proveedor de identidad.

El desarrollo de este prototipo ha puesto de relieve tanto las similitudes y las diferencias existentes entre el funcionamiento de la solución seleccionada en cada uno de los frameworks empleados como las dificultades encontradas en el proceso. Así, se ha comprobado que en algunos casos los tres frameworks abordan una misma operación de maneras diferentes, mientras que en otros emplean aproximaciones prácticamente idénticas entre sí. Esto puede afectar a la toma de decisiones a la hora de configurar la solución, como ha sido el caso en la elección de referente para el identificador único de los usuarios.

Asimismo se han puesto de manifiesto varias casuísticas que requerirían de operaciones adicionales de cara a la configuración de sistema de gestión de identidad tal y como se ha planteado en el presente trabajo, tales como la necesidad de mapear el claim «roles» en cada uno de los frameworks para mantener las aplicaciones agnósticas respecto al proveedor de identidad o la necesidad de emplear elementos adicionales para suplir posibles incompatibilidades de los frameworks utilizados con el estándar de proporción de servicios de identidad empleado.

Cabe asimismo mencionar que en el desarrollo de este proyecto ha sido necesario en varios momentos ajustar la planificación original a raíz tanto de la aparición de dificultades técnicas inesperadas durante el desarrollo del prototipo como de imperfecciones en las previsiones iniciales acerca del tiempo y la dedicación necesarios para la realización de cada una de las tareas y subtareas que lo componen.

Como fruto adicional del presente proyecto, derivado tanto del trabajo de análisis realizado como de la prueba de concepto desarrollada, esperamos haber contribuido a desmitificar la complejidad de los sistemas de gestión de identidad, los cuales pueden proporcionar un servicio de gran valor dentro del ámbito empresarial, puesto que no solo contribuyen a aumentar la seguridad de las interacciones tanto de clientes como de empleados sino que también ayudan a disminuir el coste administrativo derivado de la gestión no automatizada de este tipo de procesos. Igualmente se ha evidenciado que ya existen frameworks que permiten trabajar con sistemas de gestión de identidad federada sin necesidad de desarrollar extensas y costosas implementaciones.

Este trabajo puede, por último, servir de punto de partida futuro para varios proyectos que permitan ampliar y profundizar sus objetivos y conclusiones, entre los cuales cabe citar los siguientes:

- Estudiar el funcionamiento en los distintos frameworks de soluciones de gestión de identidad que incluyan estándares diferentes a OpenID Connect.

- Estudiar el funcionamiento en los distintos frameworks de soluciones de gestión de identidad que incluyan proveedores de identidad en la nube.

- Analizar procedimientos de autenticación diferentes a Single Sing On, como Single Log Out o la autenticación iniciada por el proveedor de identidad.

- Analizar las herramientas disponibles para cada frameworks de cara a la realización de test unitarios y de integración relacionados con los procedimientos de autenticación.

- Analizar la funcionalidad de «Identity Brokering» de los proveedor de identidad, por ejemplo utilizando repositorio de usuarios diferente al repositorio interno del proveedor, como LDAP, Microsoft Active Directory o Google Suite, etc.

5. Bibliografía

[1] D. W. Chadwick, «Federated Identity Management», p. 26.

[2] «OASIS Security Services (SAML) TC | OASIS». https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#overview (accedido mar. 03, 2020).

[3] «CAS - CAS Protocol Specification». https://apereo.github.io/cas/6.0.x/protocol/CAS-Protocol-Specification.html (accedido abr. 18, 2020).

[4] «OpenID Connect | OpenID». https://openid.net/connect/ (accedido mar. 03, 2020).

[5] «Keycloak». https://www.keycloak.org/ (accedido mar. 03, 2020).

[6] «CAS | Apereo». https://www.apereo.org/projects/cas (accedido mar. 03, 2020).

[7] «OpenAM - Open Access Manager · Open Identity Platform», *Open Identity Platform - Open Source Solutions for Access Management, Identity Management, Directory Services*. https://www.openidentityplatform.org/openam (accedido abr. 19, 2020).

[8] «Spring», *Spring*. https://www.spring.io (accedido mar. 03, 2020).

[9] «Play Framework - Build Modern & Scalable Web Apps with Java and Scala». https://www.playframework.com/ (accedido mar. 03, 2020).

[10] «Quarkus - Supersonic Subatomic Java». https://quarkus.io/ (accedido mar. 03, 2020).

[11] «Empowering App Development for Developers | Docker». https://www.docker.com/ (accedido mar. 03, 2020).

[12] «Gradle Build Tool», *Gradle*. https://gradle.org/ (accedido mar. 03, 2020).

[13] J. Bradley, N. Sakimura, y M. B. Jones, «JSON Web Token (JWT)». https://tools.ietf.org/html/rfc7519 (accedido abr. 13, 2020).

[14] J. Hildebrand y M. Jones, «JSON Web Encryption (JWE)». https://tools.ietf.org/html/rfc7516 (accedido abr. 13, 2020).

[15] «WebFinger». https://webfinger.net/ (accedido abr. 13, 2020).

[16] «Final: OpenID Connect Core 1.0 incorporating errata set 1. Standard Claims». https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims (accedido abr. 13, 2020).

[17] «Final: OpenID Connect Core 1.0 incorporating errata set 1. Scope Claims». https://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims (accedido abr. 13, 2020).

[18] «The Authorization Code Flow in Detail: Registration», *openid-connect-documentation*. https://rograce.github.io/openid-connect-documentation/explore_auth_code_flow#registration (accedido abr. 13, 2020).

[19] «Final: OpenID Connect Discovery 1.0 incorporating errata set 1». https://openid.net/specs/openid-connect-discovery-1_0.html (accedido abr. 13, 2020).

[20] «Final: OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1». https://openid.net/specs/openid-connect-registration-1_0.html (accedido abr. 13, 2020).

[21] «OpenID Connect explained | Connect2id». https://connect2id.com/learn/openid-connect (accedido abr. 13, 2020).

[22] «About | Apereo». https://www.apereo.org/content/about (accedido abr. 18, 2020).
[23] «The CAS Protocol for Application Owners | Identity & Access Management». https://iam.uconn.edu/the-cas-protocol-for-application-owners/# (accedido abr. 18, 2020).
[24] «Keycloak - Documentation». https://www.keycloak.org/documentation (accedido abr. 19, 2020).
[25] Doccrazy, *Doccrazy/keycloak-protocol-cas*. 2020.
[26] *cloudtrust/keycloak-wsfed*. Cloudtrust, 2020.
[27] «Keycloak - Securing Applications and Services Guide - Java Adapters». https://www.keycloak.org/docs/latest/securing_apps/#java-adapters (accedido abr. 19, 2020).
[28] *Github repository: apereo/cas*. Apereo Foundation, 2020.
[29] «CAS server (SSO) authentication - MoodleDocs». https://docs.moodle.org/30/en/CAS_server_(SSO)_authentication (accedido may 17, 2020).
[30] T. Community, «Tikiwiki Documentation - CAS Authentication», *Documentation for Tiki Wiki CMS Groupware*. http://doc.tiki.org/CAS-Authentication (accedido may 17, 2020).
[31] «Release notes - Spring Framework v5.2.5.RELEASE - spring-projects/spring-framework», *GitHub*. https://github.com/spring-projects/spring-framework (accedido abr. 24, 2020).
[32] «Introduction to Spring Framework», *GeeksforGeeks*, ene. 16, 2019. https://www.geeksforgeeks.org/introduction-to-spring-framework/ (accedido abr. 24, 2020).
[33] «Stack Overflow Developer Survey 2019», *Stack Overflow*. https://insights.stackoverflow.com/survey/2019/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2019 (accedido abr. 24, 2020).
[34] M. Fowler, «Inversion of Control», *martinfowler.com*. https://martinfowler.com/bliki/InversionOfControl.html (accedido abr. 24, 2020).
[35] «Thymeleaf». https://www.thymeleaf.org/ (accedido abr. 24, 2020).
[36] *It's a Kind of Magic: Under the Covers of Spring Boot - Brian Clozel, Stéphane Nicoll*. .
[37] «Spring Security Project», *Spring*. https://spring.io/projects/spring-security (accedido abr. 24, 2020).
[38] «What is Play? - Play Framework Documentation - 2.8.x». https://www.playframework.com/documentation/2.8.x/Introduction (accedido abr. 25, 2020).
[39] «Akka: build concurrent, distributed, and resilient message-driven applications for Java and Scala». https://akka.io/ (accedido abr. 25, 2020).
[40] «sbt - The interactive build tool». https://www.scala-sbt.org/ (accedido abr. 25, 2020).
[41] «Play Framework - Documentation - Module Directory - 2.8.x». https://www.playframework.com/documentation/2.8.x/ModuleDirectory (accedido abr. 25, 2020).
[42] «CAS in the cloud - Official webpage». https://www.casinthecloud.com/ (accedido abr. 25, 2020).
[43] *Quarkus why, how and what by Emmanuel Bernard*. .

[44] «Quarkus - Vision - Container First». https://quarkus.io/vision/container-first (accedido abr. 26, 2020).
[45] «Wikipedia - Just-in-time compilation», *Wikipedia*. mar. 29, 2020, Accedido: abr. 26, 2020. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Just-in-time_compilation&oldid=948050395.
[46] «Wikipedia - Adaptive optimization», *Wikipedia*. abr. 19, 2020, Accedido: abr. 26, 2020. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Adaptive_optimization&oldid=951840182.
[47] *Secure your Quarkus Applications by Sebastien Blanc*. .
[48] «CAS - ClearPass». https://apereo.github.io/cas/6.0.x/integration/ClearPass.html (accedido abr. 29, 2020).
[49] «Play Framework - Documentation - Java Templates - 2.8.x». https://www.playframework.com/documentation/2.8.x/JavaTemplates (accedido abr. 25, 2020).
[50] «Keycloak - About». https://www.keycloak.org/about (accedido may 02, 2020).
[51] *Github - pac4j/play-pac4j*. PAC4J, 2020.
[52] «Java Play Modules - 2.8.x». https://www.playframework.com/documentation/2.8.x/JavaPlayModules#Writing-Play-Modules (accedido may 10, 2020).
[53] «pac4j: User Profile - security for Java». https://www.pac4j.org/docs/user-profile.html (accedido may 10, 2020).
[54] «Keycloak - Guide - Keycloak on Docker». https://www.keycloak.org/getting-started/getting-started-docker (accedido may 02, 2020).
[55] «Keycloak - Guide - Keycloak on OpenJDK». https://www.keycloak.org/getting-started/getting-started-zip (accedido may 02, 2020).
[56] «Spring Framework Documentation». https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html (accedido abr. 24, 2020).
[57] «CAS - CAS Protocol». https://apereo.github.io/cas//6.1.x/protocol/CAS-Protocol.html (accedido abr. 18, 2020).
[58] «The Authorization Code Flow in Detail», *openid-connect-documentation*. https://rograce.github.io/openid-connect-documentation/explore_auth_code_flow (accedido abr. 13, 2020).

6. Anexos

Anexo 1. Configuración general de Keycloak

En este anexo se ejemplifican los pasos a seguir para configurar una conexión entre el proveedor de identidad Keycloak y una aplicación o cliente mediante el estándar OpenID Connect, así como para definir los usuarios de dicha aplicación y los roles de estos (en este caso «user» y «admin»). Estos pasos se ilustran empleando la aplicación de solicitud de vacaciones desarrollada como parte del proyecto («vacaciones-app»), pero son similares en las otras dos aplicaciones desarrolladas («pedidos-app» e («informes-app»).

Tras arrancar Keycloak en nuestra máquina[5], nos autenticamos con un usuario con funciones de administrador («admin») y creamos un realm, esto es, dispondrá de su propio grupo de clientes configurados, su propio grupo de usuario, roles, etc. (en este caso, hemos identificado el realm con el nombre de la empresa ficticia a la que pertenecen las aplicaciones, «EmpresaSL»).

Posteriormente, en el apartado «Roles» definimos los roles disponibles en el realm, que asignaremos a los usuarios y que serán compartidos por las aplicaciones.

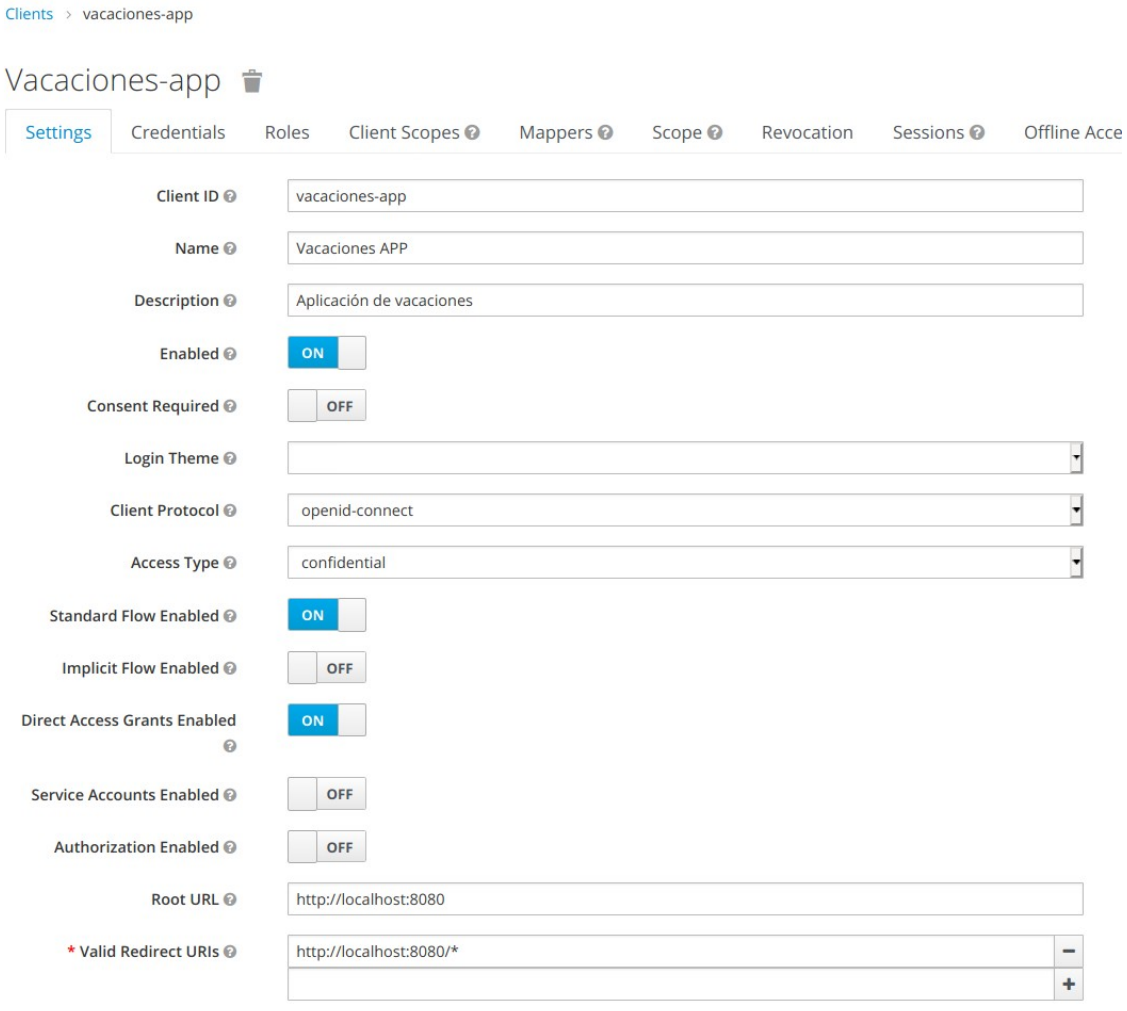
A continuación creamos los usuarios («Users → New User») y les damos nombre («username»). En la pestaña «Credentials» les asignamos una contraseña y en la pestaña «Role Mappings», un rol.

A continuación definimos una conexión entre Keycloak y la aplicación («Clients → Create»). En la pestaña «Settings» definimos el identificador de la aplicación o «client-id» («vacaciones-app») y el estándar que se va a utilizar utilizar («openid-connect»), introducimos la URL base de la aplicación y seleccionamos el tipo de acceso (en este caso, confidencial). Keycloak por su parte define de forma automática las URL de redireccionamiento autorizadas. En la pestaña «Credentials» seleccionamos el método de autenticación entre la aplicación y el OP; en este caso hemos configurado en el apartado anterior el identificador único («client id») y la clave simétrica es generada por Keycloak («secret»).

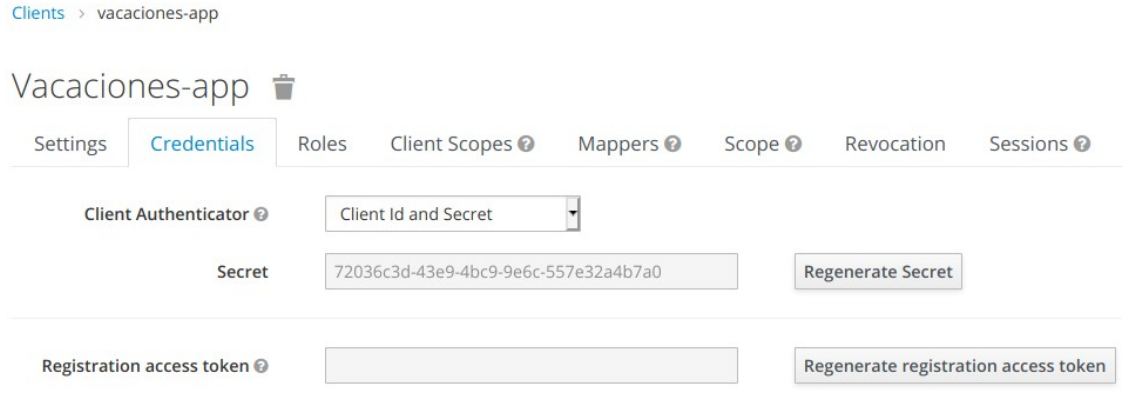
Por último, se realiza un mapeo para crear un «claim» personalizado llamado «roles» en el que se incluyen los roles de usuarios de la aplicación («user», «admin»), para ello, se definió un «mapper» asociado a la aplicación encargado de leer todos los roles asociados al usuario autenticado y de situarlos en un «claim» que se devuelve en el ID token.

Ejemplo 1: Creación del realm «EmpresaSL»

5 En el desarrollo de este proyecto se han utilizado las versiones de Keycloak en Docker [54] y en OpenJDK [55].

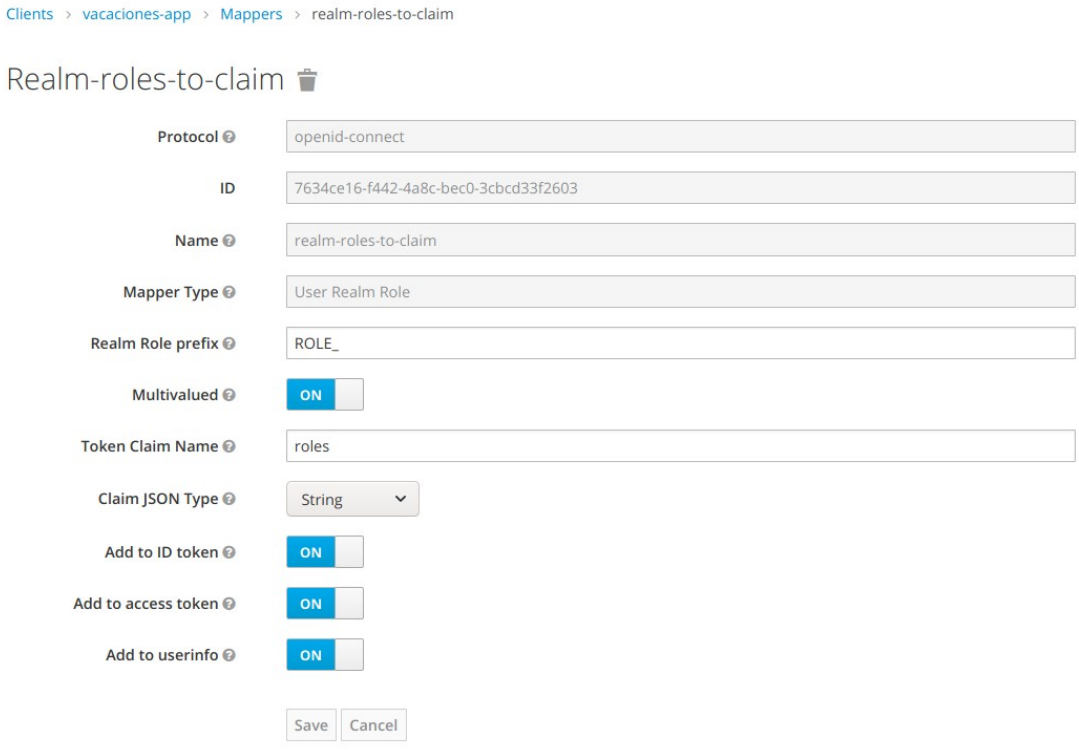


Ejemplo 2: Configuración del cliente

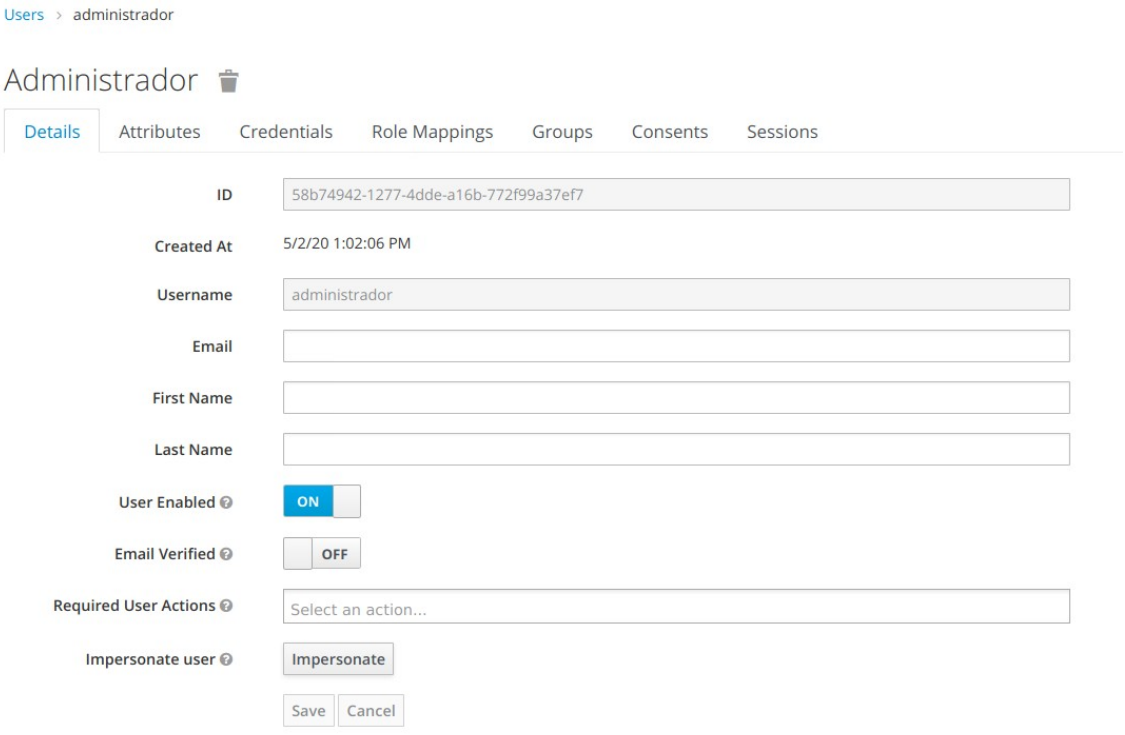


Ejemplo 3: Configuración del cliente (Credenciales)

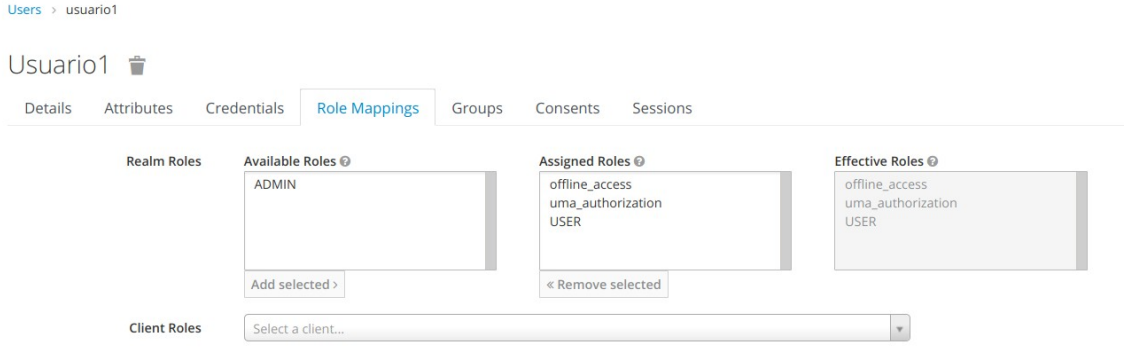
Ejemplo 4: Configuración del realm (listado de roles)



Ejemplo 5: Definición del «mapper» que sitúa los roles de usuario en un «claim» llamado «roles»



Ejemplo 6: Creación de usuarios



Ejemplo 7: Asignación de roles a un usuario

Anexo 2: Ejemplos con el análisis de estándares

```
<form method="post" action="https://idp.example.org/SAML2/SSO/POST" ...>
  <input type="hidden" name="SAMLRequest" value="base64(resquest)" />

  <input type="submit" value="Submit" />
</form>
```

Ejemplo 8: Formulario devuelto por el SP para comenzar el proceso de login en un IdP mediante Post binding

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="urn:oasis:
names:tc:SAML:2.0:assertion" ID="ONELOGIN_809707f0030a5d00620c9d9df97f627afe9dcc24"
Version="2.0" ProviderName="SP test" IssueInstant="2014-07-16T23:52:45Z" Destination="http://
idp.example.com/SSOService.php" ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST" AssertionConsumerServiceURL="http://sp.example.com/demo1/index.php?acs">
  <saml:Issuer>http://sp.example.com/demo1/metadata.php</saml:Issuer>
  <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:1.1:nameid-format:transient"
  AllowCreate="true"/>
  <samlp:RequestedAuthnContext Comparison="exact">
    <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:
    PasswordProtectedTransport</saml:AuthnContextClassRef>
  </samlp:RequestedAuthnContext>
</samlp:AuthnRequest>
```

Ejemplo 9: AuthnRequest en el que el SP (sp.example.com) realiza una petición de autenticación al IdP (idp.exampyle.com) mediante POST binding

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="urn:oasis:names:tc:
SAML:2.0:assertion" ID="_8e8dc5f69a98cc4c1ff3427e5ce34606fd672f91e6" Version="2.0" IssueInstant="2014-
07-17T01:01:48Z" Destination="http://sp.example.com/demo1/index.php?acs"
InResponseTo="ONELOGIN_4fee3b046395c4e751011e97f8900b5273d56685">
  <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
  <saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/
  2001/XMLSchema" ID="_d71a3a8e9fcc45c9e9d248ef7049393fc8f04e5f75" Version="2.0" IssueInstant="2014-
  07-17T01:01:48Z">
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
    <saml:Subject>
      <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php" Format="urn:oasis:names:
      tc:SAML:2.0:nameid-format:transient">_ce3d2948b4cf20146dee0a0b3dd6f69b6cf86f62d7</saml:NameID>
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipient="http://sp.
        example.com/demo1/index.php?acs"
        InResponseTo="ONELOGIN_4fee3b046395c4e751011e97f8900b5273d56685"/>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2014-07-17T01:01:18Z" NotOnOrAfter="2024-01-18T06:21:48Z">
      <saml:AudienceRestriction>
        <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
    <saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z" SessionNotOnOrAfter="2024-07-17T09:01:
    48Z" SessionIndex="_be9967abd904ddcae3c0eb4189adbe3f71e327cf93">
      <saml:AuthnContext>
        <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</
        saml:AuthnContextClassRef>
      </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
      <saml:Attribute Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test@example.com</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="eduPersonAffiliation" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
      format:basic">
        <saml:AttributeValue xsi:type="xs:string">users</saml:AttributeValue>
        <saml:AttributeValue xsi:type="xs:string">examplerole1</saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

Ejemplo 10: SAML Response en el que el IdP devuelve la información solicitada por el SP

```
HTTP/1.1 302 Found
Location: https://openid.provider.local/authorize?
          response_type=code
          &scope=openid
          &client_id=random_id
          &state=uijejls31
          &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Ejemplo 11: El RP redirige el navegador del usuario a la URL de autorización del OP

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
          code=SplxlOBeZQQYbYS6WxSbIA
          &state=uijejls31
```

Ejemplo 12: El navegador del usuario es redirigido a la URL definida por redirect_uri

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
    yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
    NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
    fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
    AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
    Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
    NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
    QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
    K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
    XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "expires_in": 3600,
}
```

Ejemplo 13: Token endpoint devuelve un ID Token junto con otros parámetros

```
POST /token HTTP/1.1
Host: openid.provider.local
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code
 &code=SplxlOBeZQQYbYS6WxSbIA
 &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Ejemplo 14: Solicitud al Token Endpoint en la que se emplea como parámetro el código de autorización

```
302 Location: https://cas.example.com/cas/login?service-htts%3A%2F%2Fapp.example.com%2F
```

Ejemplo 15: El servicio redirecciona el navegador del usuario al Login Endpoint del servidor CAS

```
GET https://cas.example.com/p3/serviceValidate
    ?service=https%3A%2F%2Fapp.example.com%2F&ticket=ST-12345678
```

Ejemplo 16: El servicio envía una solicitud al «endpoint» del servidor CAS "p3/serviceValidator" con su nombre y el ST como parámetros

```
Set-Cookie: CASTGC=TGT-2345678
302 Location:https://app.example.com/?ticket=ST-12345678
```

Ejemplo 17: El servidor CAS crea la cookie con el TGT y redirige al usuario al servicio con un Service token como parámetro

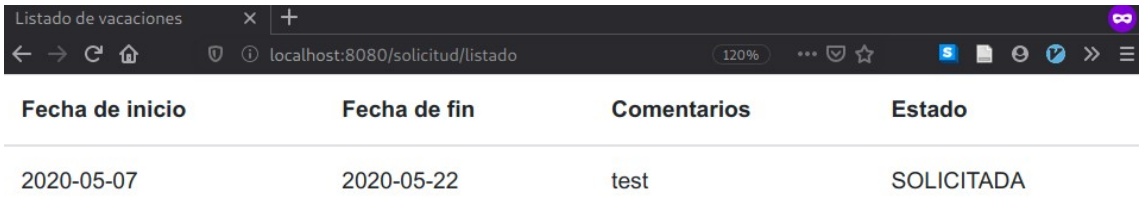
```
<cas:serviceResponse xmlns:cas="http://www.yale.edu/tp/cas">
  <cas:authenticationSuccess>
    <cas:user>username</cas:user>
    <cas:attributes>
      <cas:firstname>John</cas:firstname>
      <cas:lastname>Doe</cas:lastname>
      <cas:title>Mr.</cas:title>
      <cas:email>jdoe@example.org</cas:email>
      <cas:affiliation>staff</cas:affiliation>
      <cas:affiliation>faculty</cas:affiliation>
    </cas:attributes>
    <cas:proxyGrantingTicket>PGTIOU-84678-8a9d...</cas:proxyGrantingTicket>
  </cas:authenticationSuccess>
</cas:serviceResponse>
```

Ejemplo 18: Respuesta del servidor CAS al servicio en la que se incluyen atributos adicionales del usuario

Anexo 3: Ejemplos relacionados con el desarrollo del prototipo

```yaml
security:
  oauth2:
    client:
      registration:
        keycloak:
          client-id: vacaciones-app
          client-secret: 72036c3d-43e9-4bc9-9e6c-557e32a4b7a0
      provider:
        keycloak:
          issuer-uri: http://localhost:8180/auth/realms/EmpresaSL
```

Ejemplo 19: Sección del fichero «application.yml» en que se especifica la configuración de la conexión entre la aplicación y Keycloak mediante OpenID Connect



Fecha de inicio	Fecha de fin	Comentarios	Estado
2020-05-07	2020-05-22	test	SOLICITADA

Ejemplo 20: Solicitudes de vacaciones visualizables para «usuario1», usuario con rol «user»

Usuario	Fecha de inicio	Fecha de fin	Comentarios	Estado	Autorizado por:	Acciones
175e1e7b-e087-4bd6-90f4-0b133c418d68	2020-05-10	2020-05-28	test	APROBADA	58b74942-1277-4dde-a16b-772f99a37ef7	Aprobar Denegar
49f2b87a-e11c-4e01-9424-4c0dc1db0875	2020-05-07	2020-05-22	test	SOLICITADA		Aprobar Denegar

Ejemplo 21: Solicitudes de vacaciones visualizables para «administrador», usuario con rol «admin», junto las posibles acciones a realizar: aprobar o denegar

```java
@Secure(clients = "OidcClient", authorizers = "any-role")
public CompletionStage<Result> listarPedidos(Http.Request request) {
    CommonProfile profile = profileUtil.getProfile(request);
    if (profile.getRoles().contains(ProfileUtil.ROL_ADMIN)) {
        return pedidoService.listaTodosPedidos().thenApplyAsync(pedidos -> ok(lista.render(pedidos, profile)),
                executionContext.current());
    } else {
        return pedidoService.listaPedidos(profile.getId()).thenApplyAsync(pedidos -> ok(lista.render(pedidos, profile)),
                executionContext.current());
    }
}
```

Ejemplo 22: @Secure permite solo el acceso a "authorizers" de tipo any-role ae este método. Filtrado de la lista en función del rol del usuario extraido del "profile"

```
@import org.pac4j.core.profile.CommonProfile
@(profile: CommonProfile)
    @main("Aplicacion de pedidos", profile) {

        <section id="content">
            <div class="wrapper doc">
                <article>
                    <h2>Aplicación de pedidos</h2>
                    <p>Bienvenido <b>@profile.getUsername</b></p>
                    <p>Lista de roles: <b>@profile.getRoles</b></p>
                    <p>Seleccione la opción deseada en el menú para continuar</p>
                </article>
            </div>
        </section>
    }
```

Ejemplo 23: Inyección del «profile» en la plantilla de índice de la aplicación y extracción de nombre de usuario y roles del mismo.

```
@Provides
protected OidcClient provideOidcClient() {
    final OidcConfiguration oidcConfiguration = new OidcConfiguration();
    oidcConfiguration.setClientId("pedidos-app");
    oidcConfiguration.setSecret("9dfb47d5-1603-4342-8148-be3ef03486e1");
    oidcConfiguration.setDiscoveryURI("http://localhost:8180/auth/realms/EmpresaSL/.well-known/openid-configuration");
    final OidcClient oidcClient = new OidcClient(oidcConfiguration);

    oidcClient.addAuthorizationGenerator((ctx, profile) -> {
        if (profile.getAttribute("roles") != null) {
            profile.addRoles((List) profile.getAttribute("roles"));
        }
        return Optional.of(profile);
    });

    return oidcClient;
}
```

Ejemplo 24: Extracto del «SecurityModule» donde se configuran los parámetros de conexión así como el mapeo de roles.

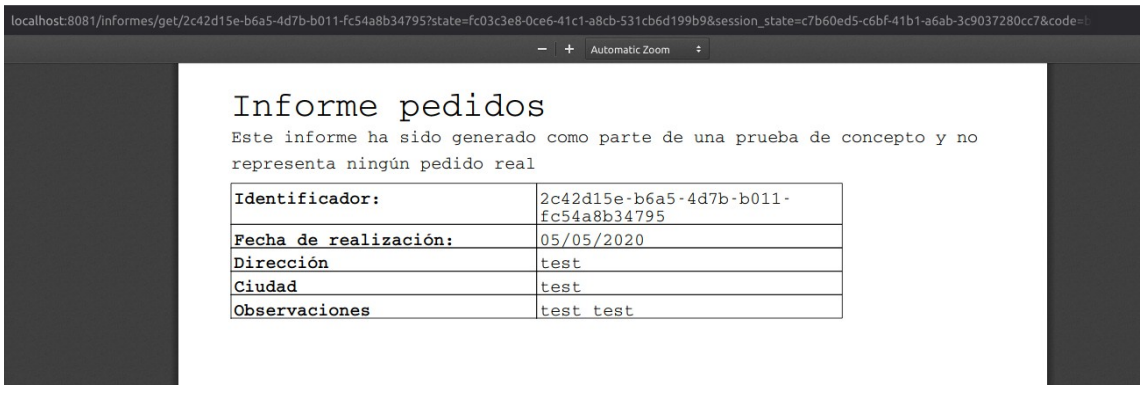
```
quarkus.oidc.auth-server-url=http://localhost:8180/auth/realms/EmpresaSL
quarkus.oidc.client-id=informes-app
quarkus.oidc.credentials.secret=dc269ddb-33cd-4367-8fbb-c36254b3da67
quarkus.oidc.application-type=web-app
quarkus.http.auth.permission.authenticated.paths=/*
quarkus.http.auth.permission.authenticated.policy=authenticated
quarkus.oidc.roles.role-claim-path=roles
```

Ejemplo 25: Sección del fichero "application.properties" donde se especifican los detalles de conexión de la informes-app a Keycloak por OpenID Connect

```java
@Inject
SecurityIdentity securityIdentity;

@GET
@Produces("application/pdf")
@Path("/get/{uuid}")
public byte[] generarInforme(@PathParam(value = "uuid") UUID identificador) {
    if (securityIdentity.getRoles().contains("ADMIN")) {
        logger.debug("Intentando extraer informe con usuario Admin");
        return informesService.getInformePorIdentificador(identificador);
    } else if (securityIdentity.getRoles().contains("USER")) {
        logger.debug("Intentando extraer informe con usuario User");
        return informesService.getInformePorIdentificadorYUserId(identificador,
                securityIdentity.getPrincipal().getName());
    } else {
        throw new RoleException("El usuario no dispone de ninguno de los roles esperados");
    }
}
```

Ejemplo 26: Código que realiza la solicitud de informe en función del rol y id del usuario



Ejemplo 27: Informe de pedido generado a partir de un pedido guardado en pedidos-app