ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Desarrollo y evaluación de una solución de gestión de identidad y acceso seguro multidispositivo para la protección de una plataforma de la Internet de las Cosas
(Development and evaluation of an identity and multidevice access management solution for securing an IoT platform)

Para acceder al Título de

Graduado en
Ingeniería de Tecnologías de Telecomunicación

Autor: Roberto Fernández Crespo

Octubre - 2022

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Roberto Fernández Crespo
Director del TFG:
Título: "Desarrollo y evaluación de una solución de gestión de identidad y acceso seguro multidispositivo para la protección de una plataforma de la Internet de las Cosas"
Title: "Development and evaluation of an identity and multidevice access management solution for securing an IoT platform"

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre):
Secretario (Apellidos, Nombre):
Vocal (Apellidos, Nombre):

Este Tribunal ha resuelto otorgar la calificación de: ....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

Este trabajo fin de grado se la dedico con todo mi amor y cariño a mis padres, los cuales siempre han estado ahí sacrificándose y esforzándose, dándome la oportunidad de formarme en lo que siempre quise y crecer profesionalmente. Sin vuestro apoyo no hubiera llegado hasta aquí.

A mi tía Tere, que siempre me ha apoyado igual que mis padres en todo lo que he hecho.

A Jorge Lanza, por involucrarse plenamente en realizar este trabajo. Sin ti no sería lo mismo.

A mis amigos, resaltando a Estefanía, Ehari, Jon, Mario y Victoria; por compartir tanto buenos como malos momentos, los cuales me han hecho crecer en lo personal.

Gracias a todos de corazón, por hacerme cumplir este sueño.

Resumen

El avance tecnológico que ha sufrido nuestra sociedad en las últimas décadas estás posibilitando el acceso a la información en cualquier lugar y momento desde todo tipo de dispositivos, desde el tradicional ordenador, pasando por el teléfono móvil, hasta los dispositivos embebidos portables (*wearables*).

Sin embargo, todas estas tecnologías traen consigo una serie de riesgos de seguridad tanto en el acceso y almacenamiento de la información, sino también en cuanto a la identificación de los usuarios que interaccionan.

La autenticación, la privacidad y la confianza se erigen, por tanto, son elementos que deben estar presentes y garantizados en cualquier ecosistema. La diversidad de protocolos y de mecanismos de seguridad vinculados es creciente. Así, por ejemplo, cada vez está más presente la autenticación basada en múltiples factores o la delegación de acceso a información personal a terceros, etc.

En este proyecto se plantea, por un lado, el estudio de diferentes alternativas para la autenticación y autorización, también conocidas como *Identity and Access Management* (IAM), y por otro su despliegue en entorno real orientado a la securización del acceso a recursos de una ciudad inteligente a través de toda clase de dispositivos de usuario.

Abstract

The technological advances in recent decades have allowed access to any information from anywhere using any devices: computers, an old but gold option; mobile phones; portable embedded devices (wearables).

Nevertheless, all these advances are dangerous when it comes to security. Not only when accessing or storing information, but also in terms of the identification of interacting users.

Authentication, privacy and trust are therefore elements that must be present and guaranteed in any ecosystem. The diversity of protocols and security mechanisms is growing. For instance, multi-factor authentication or the delegation of access to personal information to third parties, etc. are becoming increasingly common.

On the one hand, this project aims to analyse different alternatives for authentication and authorization, also known as Identity and Access Management (IAM). On the other hand, it seeks to analyse it in a real environment aimed at securing access to smart city resources through all kinds of user devices.

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

El avance tecnológico es una constante imparable, la cual, cada día, se va perfeccionando y mejorando en función de las necesidades humanas. La conectividad inalámbrica de muchos dispositivos, a la mayor velocidad y con la menor latencia posible, propicia la aparición de redes celulares 5G. El deseo de gran parte de la población de evitar el control bancario centralizado y disponer de una moneda descentralizada, sin banco central y sin administrador único, propició la aparición de Bitcoin y el consecuente desarrollo de la tecnología Blockchain. Estos son algunos del sinfín de ejemplos que podríamos enumerar.

Otro de estos casos es el de las ciudades inteligentes (*Smart Cities*), concebidas como ciudades en las que la tecnología permite mejorar los servicios ofrecidos a los ciudadanos. Utilizando una plétora de diferentes sensores, adquieren datos medioambientales, del estado de edificios, del tráfico, entre otros, y, a través de su análisis, son capaces de gestionar los recursos y servicios de la ciudad de manera eficiente, mejorando la calidad de vida de los ciudadanos. Todo esto se ha podido hacer realidad gracias a la eclosión de la Internet de las Cosas (IoT, *Internet of Things*), como red que conecta multitud de sensores y actuadores, y habilita el procesamiento de los datos adquiridos por los mismos. De esta forma, se constituye un nuevo ecosistema que permite el desarrollo de servicios y la mejora de los ya existentes. Por ejemplo, saber dónde hay plazas libres de aparcamiento ayuda a poder mejorar el tráfico, disminuir el tiempo de búsqueda, el consumo de combustible y por ende la contaminación, favoreciendo el medio ambiente.

Sin embargo, todas estas tecnologías traen consigo una serie de riesgos de seguridad. Cada día surgen nuevas brechas en todo tipo de sistemas, como por ejemplo el acceso a cámaras de vigila bebés [1] o clones de AirTag que eluden las funciones de protección de seguimiento de Apple [2]. Es por ello, que debemos de tener en cuenta la seguridad desde las primeras etapas del diseño de los servicios y sistemas, para que de forma holística hagan frente a las potenciales amenazas y garanticen la protección tanto del servicio como de los usuarios que lo utilicen.

La seguridad engloba varios conceptos, entre los que destacan la privacidad, la autenticación y el control en el acceso, y la confianza en el ecosistema proporcionado, usualmente por terceros.

En lo relativo a la privacidad, es indispensable que toda persona o entidad tenga control sobre sus datos y el uso que se hace de ellos. Un usuario no puede permitir que, al acceder a servicios, terceros obtengan su información de forma no autorizada. En este sentido, en la Unión Europea se han legislado numerosas medidas, siendo la más relevante la Directiva de Privacidad y Comunicaciones Electrónicas, por la cual los usuarios han de dar consentimiento expreso para que proveedores de servicios tengan acceso a información vinculada a ellos. Un ejemplo con el que nos encontramos en el día a día es la instalación y uso de *cookies* en los navegadores para la extracción de información relativa a la navegación [3].

Por otro lado, para tener un sistema seguro es vital la autenticación, es decir, que se proporcionen los medios para garantizar que el usuario sea quien dicen ser. Para ello, se puede utilizar diversos factores basándonos en algo que el usuario sepa (claves, contraseñas, PIN, etc.), algo que este posea (tarjeta inteligente, dispositivo, etc.) o algo que él sea (huella dactilar, iris, etc.). Hoy en día es muy común utilizar autenticación basada en 2 factores (2FA, *Two Factor Authentication*) o en múltiples (MFA, *Multiple Factor Authentication*). De esta forma se eleva el nivel de seguridad.

Otro aspecto a considerar en un entorno seguro es el control de acceso, por el cual se establece que no todos los usuarios pueden acceder a todos los recursos ofrecidos por un servicio. Un ejemplo ilustrativo lo podemos encontrar en el entorno educativo, donde los profesores pueden leer y escribir las calificaciones, pero los alumnos solo tienen permiso para leerlas. Sería impensable que los alumnos pudieran escribir sus notas en la plataforma. En este caso, se aplica un control de acceso basado en roles (RBAC, *Role-based access control*).

Finamente, la confianza también está presente en la seguridad. Simplemente cuando utilizamos *social login* podemos observar que la aplicación que estamos usando confía en terceros, ya usa los datos de estos últimos para autenticar al usuario.

1.1 Motivación

La motivación de este proyecto es comprender y profundizar en la gestión segura de servicios, es decir, en la autenticación y autorización que permita un mayor control de los usuarios que acceden a aplicaciones o recursos. Cada vez es mayor la variedad de dispositivos usados por los usuarios para acceder a los servicios y las herramientas disponibles para lograr securizarlos. Por tanto, se hace necesario no discriminar el medio de acceso en función de su naturaleza, y tratar de mantener el nivel de seguridad independientemente del mismo.

Adicionalmente, se debería aligerar el despliegue de las herramientas de seguridad. Para ello, nos vemos en la necesidad de utilizar protocolos estandarizados y seguros de autenticación y de autorización. A través de ellos se validará la identidad y las políticas de acceso, además de los dispositivos de los usuarios y la confianza en terceros.

1.2 Objetivos

El objetivo del proyecto no es otro que realizar un control de identidad y de acceso a un servicio, Smart Santander, sin modificarlo y permitiendo acceder a este independientemente del dispositivo cliente, ya sea un móvil, ordenador, o incluso dispositivos limitados o sin medios de interacción con el usuario.

Para lograr dicho objetivo, debemos de conseguir los siguientes hitos:

- Análisis de mercado y de protocolos relativos a gestores de identidad y acceso (IAM, *Identity and Access Management*).
- Análisis y testeo de comportamiento de un sistema IAM.
- Despliegue de sistema IAM en entorno real.
- Evaluación de la interacción de usuarios y dispositivos físicos limitados con el sistema IAM en un entorno de ciudad inteligente para garantizar el acceso y uso seguro de datos y servicios.

1.3 Organización del documento

Este documento se ha dividido en seis capítulos en los que se describen los trabajos realizados para alcanzar los objetivos finados, los cuales se enumeran a continuación.

En el capítulo 1, Introducción, se expone la situación en la que se desarrolla el proyecto, además de las motivaciones para llevarlo a cabo y los objetivos que se pretenden obtener.

Seguidamente en el capítulo 2, Estado del arte, se describen las diferentes soluciones IAM y los protocolos relativos a la autorización y autenticación. Resultado del mismo se muestra un resumen de las características de estos para facilitar su comparativa.

El capítulo 3, Análisis del problema, analiza el problema existente en profundidad. Junto con lo expuesto en el capítulo 2, permite elaborar una solución acorde a las necesidades a cubrir.

Tras esto, en el capítulo 4, Evaluación del sistema, se desarrolla el sistema en cuestión, explicando con detalle su configuración e instalación de manera sencilla. También se elaboran diferentes pruebas para asegurar su correcto funcionamiento.

En el capítulo 5, Implementación, se prueba el sistema en un entorno real, protegiendo Smart Santander y se detalla la elaboración de un dispositivo embebido, el cual será utilizado en una comprobación integra de la solución.

Finalmente, en el capítulo 6, Conclusiones y líneas futuras, se recogen las conclusiones del proyecto, comprobando si los resultados han satisfecho los objetivos iniciales. También se plantean las posibles evoluciones o tendencias que el proyecto es capaz de alcanzar.

2 Estado el arte

En este capítulo se detallará una serie de conceptos previos relativos a sistemas de control de acceso e identidad, seguido de diferentes soluciones IAM del mercado y se finalizará con los protocolos estándar de autenticación y autorización. Estos dos últimos contendrán una comparativa para facilitar la justificación de la solución seleccionada.

2.1 Conceptos previos

En nuestro día a día, estamos continuamente accediendo a aplicaciones donde debemos de identificarnos previamente. A este proceso de confirmación de la identidad de un usuario o de un dispositivo frente al otro u otros participantes de la comunicación se le conoce como autenticación [4].

Para ello, es necesario un factor de autenticación como prueba de la identidad. Existen diferentes ejemplos, aunque todos ellos se recogen algo que el usuario es (huella dactilar, identificación facial, retina…), algo que tiene (dispositivo móvil, tarjeta de identificación…), algo que sabe (contraseña, PIN, respuesta a ciertas preguntas…) o algo que hace (firma, voz…). Una práctica habitual es MFA (*Multi Factor Authentication*) [5], la cual combina algunos de estos factores, elevando los requisitos y la seguridad.

Sea cual sea el factor de autenticación, esta debe de ser fiable para que el sistema sea seguro. También debe de ser reconocida y aceptada por los usuarios, ya que, si estos al final son los consumidores del servicio, no tendría sentido que no pudieran identificarse y no disfrutaran de este.

Existen diferentes tipos de autenticación: directa, cuando en el proceso intervienen exclusivamente las partes interesadas; indirecta, cuando interviene una tercera parte confiable que actúa como autoridad o juez, avalando la identidad de las partes involucradas; unilateral, cuando solo se autentica una única parte; y mutua, cuando se autentican ambas.

Sin embargo, hay cientos de casos donde solamente la identidad no es suficiente. Por ejemplo, el uso de la banca a distancia. A la cuenta de Alice, solamente ella debe de ser capaz de acceder, mientras que cualquier otro usuario no puede acceder. Es aquí donde surge la autorización [6], que se define como el proceso que verifica a qué recursos pueden acceder las entidades (usuarios o dispositivos) y qué acciones pueden realizar, es decir, sus derechos de acceso.

Normalmente, todo esto se gestiona y configura con software IAM (*Identity and Access Management*) [7], garantizando que los usuarios adecuados tengan el acceso apropiado a los recursos tecnológicos. Suele tener numerosos parámetros configurables a medida del servicio que se quiere proteger, desde los diferentes recursos a proteger, los permisos, políticas, etc. Estos sistemas incluyen también facilidades de gestión de *login*, *emails* de verificación, métodos de autenticación extendidos, etc.

Para finalizar, un servicio básico de los IAM son los SSO (*Single Sign On*) [8]. Con ellos, un usuario es capaz de acceder a múltiples servicios o aplicaciones autenticándose una única vez. Un ejemplo lo encontramos en los servicios proporcionados por Google, a los cuales, con una única sesión, se puede acceder a las plataformas de Gmail, YouTube, Drive, etc.

Este servicio, además de mejorar la experiencia de usuario, bien planteado, aumenta la seguridad, ya que se reduce el número de credenciales en la red. También supone mayores beneficios para la plataforma, ya que solo ha de gestionar un único usuario y contraseña.

Para que un servicio SSO se desarrolle con normalidad, necesitamos a un proveedor de servicios (SP, *Service Provider*) [9], que es la entidad que proporciona o habilita servicios a los usuarios u a otros servicios. También la figura del proveedor de identidades (IdP, *Identity Provider*) [10] es un elemento de gran importancia en SSO. Al final, es el que gestiona la información de los usuarios, servicios, ordenadores, etc. De esta forma, las aplicaciones o servicios se pueden apoyar en este para autenticar o autorizar a clientes.

2.2 Soluciones IAM

En esta sección vamos a analizar las diferentes soluciones software IAM, tanto a nivel descriptivo como de sus características y sus diferentes distribuciones. Para finalizar, en la sección 2.2.4 observaremos las características de todas las herramientas.

2.2.1 Keycloak

Keycloak [11] es una herramienta de código abierto enfocada a la gestión de acceso e identidad, con carácter empresarial. WildFly (división de Red Hat) se centra en su desarrollo. Se encuentra escrito en Java.

El objetivo de esta herramienta es ser capaz de proteger aplicaciones y servicios sin alterar los mismos o de manera mínima. De esta manera, los desarrolladores se pueden centrar en la aplicación en sí y posteriormente Keycloak se encargará de proporcionar una capa de seguridad que proteja el acceso y uso de la misma.

Su interfaz se muestra en la Figura 2.1.

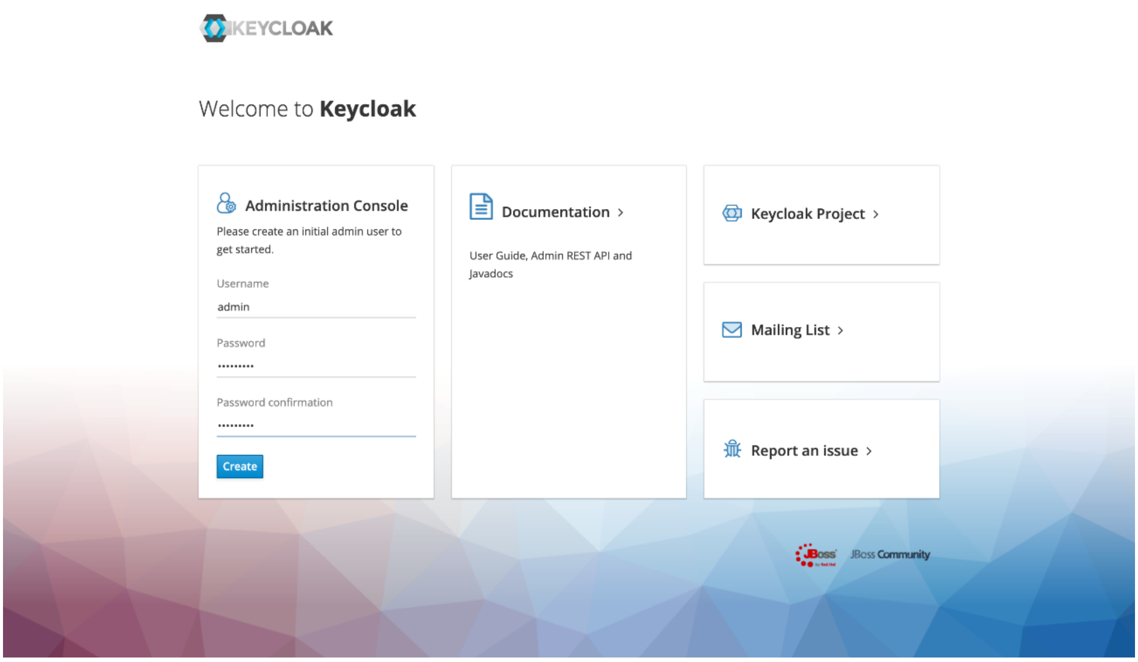


Figura 2.1 Interfaz de Keycloak

2.2.1.1 Características

Keycloak es una herramienta que permite desplegar un sistema de inicio de sesión único SSO, mediante el cual un usuario puede acceder a múltiples servicios empleando unas únicas credenciales. En este sentido, hablita la autenticación basada en múltiples factores, entre ellos haciendo uso de sistemas de contraseñas de un solo uso (OTP, *One Time Password*). Adicionalmente ofrece soporte para múltiples protocolos estándar de autorización y autenticación como son OIDC (*OpenID Connect*), OAuth2 (*Open Authorization*) o SAML (*Security Assertion Markup Language*), no solo de forma integral sino permitiendo también la integración e interacción con sistemas proveedores externos LDAP (*Lightweight Directory Access Protocol*)/*Active Directory* y *social login* con Google, Facebook, GitHub, etc.

De cara a la gestión, incorpora una consola centralizada, desde la cual, los administradores pueden gestionar toda la herramienta desde el panel Web. También existe una API (*Application Programming Interface*) para la administración. Desde dicha consola, se puede implementar formularios plenamente operativos de recuperación de contraseña, creación de usuarios, añadir dispositivo OTP, etc.

Por defecto, trae consigo un tema visual predeterminado. Sin embargo, se pueden añadir diferentes temas de inicio de sesión, recuperación de contraseña, etc. Estos pueden ser diferentes para cada aplicación.

Por último, Keycloak es flexible y personalizable. Existen muchas opciones para añadir a los proyectos, políticas y acciones programadas, tales como verificación de correo, contraseña temporal, cambio de contraseña, etc. También se puede emplear captchas, modificar los tiempos máximos de expiración de sesiones o tokens, agrupar usuarios, etc. De esta forma, se adapta perfectamente a la finalidad del servicio y a futuro no es limitante.

2.2.1.2 Distribuciones de Keycloak

Las principales formas de distribución [12] son servidor, docker y operador.

La versión server la podemos encontrar en la página web descargable en formato *.tar*, el cual contiene todo lo necesario para lanzar dicho programa. Solamente se requiere tener instalado Java Developer Kit. Es la clásica instalación de un programa.

Si optamos por la versión de docker, esta nos brinda tener las ventajas de Keycloak ligadas a las de los contenedores. Existen dos imágenes oficiales, la de DockerHub (versión 16.1.1) y la de Quay Container Registry (versión 18.2). Se recomienda emplear la de Quay Container Registry, ya que es donde se encuentran las versiones actualizadas.

Finalmente tenemos la operador. Esta distribución está basada en Operator SDK (*Software Development Kit*) para Kubernetes y OpenShift.

2.2.1.3 Adaptadores Keycloak

Para integrar Keycloak con las aplicaciones existen los adaptadores [13]. Se trata de librerías que ofrecen un API bajo el cual están desarrollados todos los procedimientos, protocolos e interacciones que deberían darse entre una aplicación cliente y un servidor basado en Keycloak. Para facilitar la integración, se ofrecen soluciones muy variadas, en diferentes lenguajes de programación. Así ejemplos de las plataformas soportadas se enumeran a continuación:

- OpenID Connect
    - Java (Spring Boot, JBoss EAP, WildFly, Jetty 9)
    - JavaScript
    - Node
    - Python
- SAML
    - Java (JBoss EAP, WildFly, Tomcat, Servlet filter, Jetty)
    - Apache HTTP Server

2.2.2 Aerobase IAM Server

Aerobase [14] es un IAM que surge de Keycloak y de otros proyectos de código abierto. Tanto es así, que la consola y los menús son similares a los de Keycloak. Pertenece a Titan Branding Wise, quien ofrece varios planes con diferentes precios en función de servicios extra. También existe la opción de código abierto que es gratuita y apoyada por la comunidad.

La interfaz se presenta en la Figura 2.2 y su documentación [15] es similar a la de Keycloak, pero más reducida.

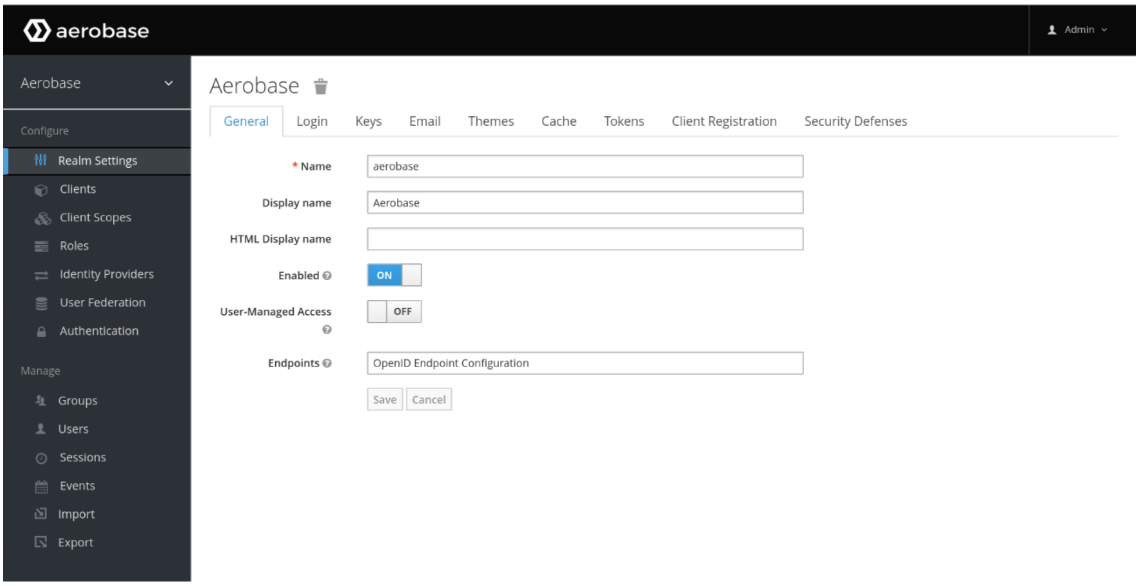


Figura 2.2 Interfaz de Aerobase

2.2.2.1 Características

Al derivarse de Keycloak, ha heredado ciertas características, como el SSO, múltiples factores de autenticación (OTP + User/Password), protocolos estándar como OIDC, OAuth2 y SAML; proveedores externos LDAP, *Active Directory*, *social login*, etc.

También contiene una consola centralizada donde los administradores pueden gestionar toda la herramienta desde el panel Web.

Este producto está disponible para Windows, CentOS, Redhat, Ubuntu, Debian y Fedora.

2.2.3 Gluu Server

Gluu Server [16] es una aplicación de software abierto, perteneciente a Gluu Organization, la cual ha agrupado diferentes herramientas, tanto propias como de código abierto, en un mismo producto.

Su interfaz se muestra en la Figura 2.3. La documentación de Gluu Server es completa [17], aunque no tiene una comunidad tan fuerte como era el caso de Keycloak.

2.2.3.1 Características

En cuanto a sus características, destaca SSO, permitiendo iniciar una sola vez y acceder a múltiples servicios. La parte de autenticación no solo se basa en las clásicas credenciales de usuario y contraseña, también podemos tener múltiples factores de autenticación como autenticación a través de reconocimiento facial (BioID, *Biometric IDentification*), notificaciones push al dispositivo del usuario, claves a través de SMS (*Short Message Service*), OTP, etc. En este sentido es muy completo.

En cuanto al *social login*, proveedores externos LDAP/Active Directory y los protocolos estándar también están integrados en la solución IAM, disponiendo tanto OAuth 2.0 y OpenID Connect como SAML 2.0.

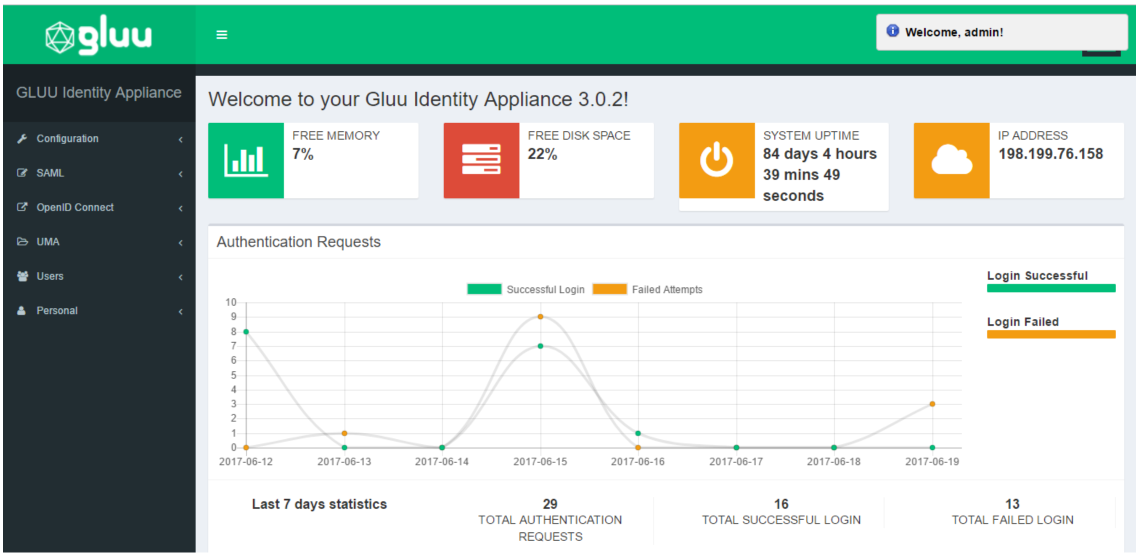


Figura 2.3 Interfaz de Gluu Server

La gestión una vez más es centralizada y además existe una API de gestión, desde la cual se puede configurar la gestión de identidad y el acceso personalizado (CIAM, *Customer Identity Access Management*).

2.2.3.2 Distribuciones de Gluu Server

Se puede instalar en Ubuntu, Docker, Kubernetes, algunas otras distribuciones de Linux, pero no Windows. Además, el proceso de instalación y configuración es relativamente tedioso y no tan sencillo como los otros IAM.

2.2.4 Comparativa IAM

A modo de resumen, en la Tabla 2.1 observamos las principales propiedades de los anteriores IAM, centrándonos en 3 agrupaciones; características, distribuciones y factores externos.

Tabla 2.1 Comparativa IAM

	Keycloak	*Aerobase IAM Server*	*Gluu Server*
Características			
SSO	Si	Si	Si
OpenID Connect	Si	Si	Si
OAuth 2.0	Si	Si	Si
SAML 2.0	Si	Si	Si
Social login	Si	Si	Si
OTP	Si	Si	Si
Consola centralizada	Si	Si	Si
IDP externo	Si	Si	Si
Temas	Si	Si	Si
Flexibilidad y personalización	Si	Si	Si
Distribuciones			
Tradicional	Si	Si	Si
Docker	Si	No	Si
Factores externos			
Documentación	Excelente	Pequeña	Excelente
Comunidad	Grande	Mediana	Pequeña
Precio	0 €	0 €	0 €
Licencia	Open source	Open source	Open source (requiere registro)

2.3 Protocolos estándar

2.3.1 OAuth 2.0

OAuth 2.0 [18] es un estándar cuya finalidad es permitir que aplicaciones o servicios web accedan a recursos de otras aplicaciones o servicios. Se trata de la segunda versión de este protocolo, la cual sustituye a la 1.0 (obsoleta). Su nombre significa autorización abierta (*Open Authorization*).

La principal característica es dar acceso a recursos a la vez que se restringen las acciones que las aplicaciones realizan sobre los recursos. Todo ello sin compartir las credenciales del usuario.

2.3.1.1 Fundamentos de OAuth 2.0

OAuth 2.0 es un protocolo de autorización. No debemos de confundirlo con autenticación. La documentación así lo remarca.

OAuth utiliza *tokens* de acceso. Un *token* incluye un conjunto de datos que representa la autorización para acceder a los recursos en nombre del usuario final. En sí OAuth no define como han de ser dichos *tokens*. En la práctica, se suele implementar *tokens* con formato *JSON* (*JavaScript Object Notation*) *Web Token* (*JWT*), los cuales contienen campos como fecha de expiración, emisor, nombre del usuario, correo electrónico, etc.

Define 4 roles: propietario del recurso, cliente, servidor de autorización y servidor de recursos. El propietario del recurso es el usuario o sistema que controla el acceso a los recursos.

El cliente es el sistema que requiere el acceso a los recursos protegidos. Para acceder a los recursos, debe tener el *token* de acceso correspondiente. No debemos confundirlo con el propietario.

Finalmente quedan los servidores. El servidor de autorización recibe las solicitudes de *tokens* de acceso por parte del cliente y los emite previa autenticación y consentimiento del propietario de los recursos. En cambio, el servidor de recursos protege los recursos del usuario y recibe las solicitudes de acceso del cliente. Acepta y valida un *token* de acceso del cliente y le devuelve los recursos adecuados.

2.3.1.2 Scopes

Los *scopes* especifican exactamente cuál es el motivo para la emisión del *token*. Típicos *scopes* serían roles del usuario, perfil del usuario y dirección de correo; como se muestra en la Figura 2.4. Otros ejemplos podrían ser fotos multimedia, mensajes del usuario, calendario del usuario etc.

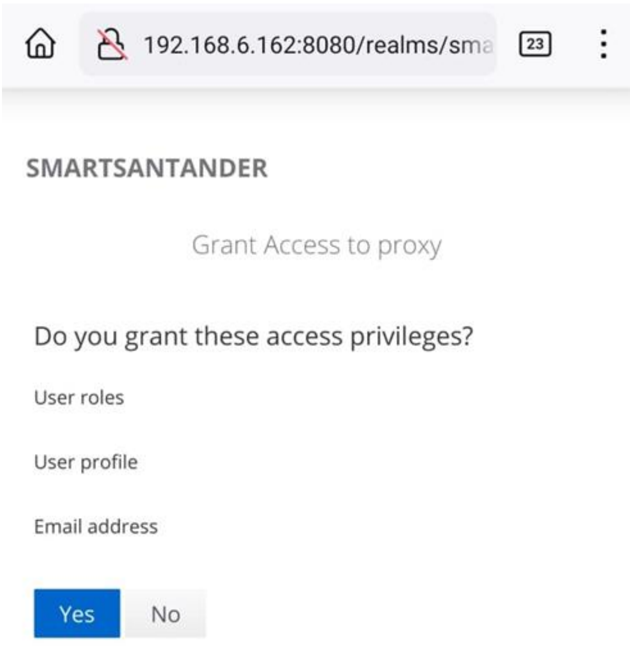


Figura 2.4 Ejemplo de OAuth2 *scopes*

2.3.1.3 Tokens e identificadores

OAuth maneja dos conceptos relativos a la identificación y dos *tokens*. El *Client ID* es un identificador único de la aplicación solicitante del recurso y el *Client Secret* es una clave secreta proveniente de la aplicación. De esta manera, el servidor de autorización autentica a la aplicación.

En cuanto a los *tokens*, encontramos los *tokens* de acceso (*Access Token*) y los *tokens* de refresco (*Refresh Token*). Los primeros son proporcionados por el servidor de autorización y usados para acceder a los recursos solicitados, como si de una entrada para un concierto se tratara. Tienen caducidad.

En cambio, los *tokens* de refresco, proporcionados también por el servidor de autorización, permiten solicitar un nuevo *token* de acceso cuando este caduca. Suelen tener un tiempo de expiración largo en comparación al A*ccess Token* y el cliente debe de almacenarlos de forma segura.

2.3.1.4 Grant types

Los *grants* o flujos de OAuth 2.0 son los pasos a seguir por todos los participantes para que el cliente acceda a los recursos de manera segura.

La elección de estos flujos depende del tipo de aplicación consumidora, grado de confianza en la misma y la interactuación por parte del usuario en el proceso.

2.3.1.4.1 Client Credentials Grant Type

Client Credentials [19] es el flujo más simple de OAuth2. Se usa cuando el cliente pertenece al usuario, por lo que no se requiere de acción de este. La aplicación se autentifica utilizando su identificador de cliente y su secreto. Se suele usar en microservicios, procesos automatizados, etc.

Figura 2.5 Client Credentials flow

En la Figura 2.5 podemos observar dicho flujo:

1) La aplicación se autentifica con el servidor de autorización utilizando su *Client ID* y su *Client Secret*.
2) El servidor de autorización valida el *Client ID* y el *Client Secret*.
3) El servidor de autorización responde con un *Access Token*.
4) La aplicación puede utilizar el *Access Token* para llamar a una API en su nombre.
5) La API responde con los datos solicitados.

Cabe destacar, que, a raíz del vínculo entre el usuario y la aplicación, y la no participación del primero en el intercambio, no existe el intercambio de usuario y contraseña como en el resto de flujos.

2.3.1.4.2 Resource Owner Password Grant Type

Resource Owner Password [20] es un flujo en el que el usuario interactúa proporcionando la dupla usuario y contraseña a la aplicación cliente. Esto en principio, esto no es compatible con la filosofía de OAuth2, pero lo que se pretende es aprovechar las ventajas del uso de *Access Tokens* con una vida útil limitada.

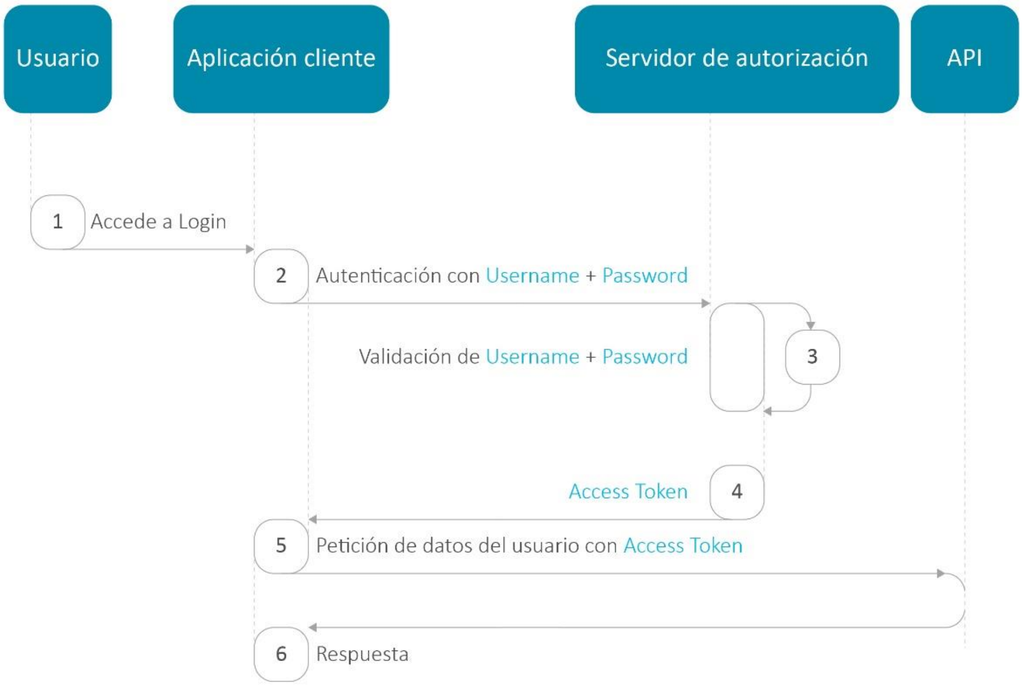


Figura 2.6 Resource Owner Password flow

En la Figura 2.6 se describe dicho flujo:

1) El usuario hace clic en *Login* dentro de la aplicación e introduce sus credenciales.
2) La aplicación reenvía las credenciales del usuario al servidor de autorización.
3) El servidor de autorización valida las credenciales.
4) El servidor de autorización responde con un *Access Token* (y opcionalmente, un *Refresh Token*).
5) La aplicación puede utilizar el A*ccess Token* para llamar a una API y acceder a la información sobre el usuario.
6) La API responde con los datos solicitados.

2.3.1.4.3 Authorization Code Grant Type

Authorization Code [21] es un flujo en el que los usuarios autorizan a los clientes de forma explícita. Esto es gracias al uso de un código, el cual es emitido por el servidor de autenticación y luego es usado por la aplicación para obtener el *Access Token*.

En la Figura 2.7 podemos observar dicho flujo. A modo de aclaración, realmente la autenticación y autorización son dos fases separadas, ya que primero el usuario ha de identificarse, con los factores pertinentes. Y tras esto, es cuando decide consentir a la aplicación que puede acceder.

También deberíamos de encontrar el papel de propietario del recurso. Sin embargo, puesto que en la mayoría de casos el propietario del recurso es el propio usuario, se ha optado por suprimirlo en la figura.

Figura 2.7 Authorization Code flow

El intercambio resultante es:

1) El usuario hace clic en *Login* dentro de la aplicación web normal.
2) La aplicación redirige al usuario al servidor de autorización
3) El servidor de autorización redirige al usuario a la solicitud de inicio de sesión y autorización.
4) El usuario se autentica y puede ver una página de consentimiento que enumera los permisos que le dará a la aplicación.
5) El servidor de autorización redirige al usuario de vuelta a la aplicación con un código de autorización (un solo uso).
6) La aplicación envía este código al servidor de autorización junto con el *Client ID* y el *Client Secret* de la aplicación.
7) El servidor de autorización verifica el código, el *Client ID* y el *Client Secret*.
8) El servidor de autorización responde con un *Access Token* y, opcionalmente, un *Refresh Token*.
9) La aplicación puede utilizar el *Access Token* para llamar a una API y acceder a información sobre el usuario.

10) La API responde con los datos solicitados.

2.3.1.4.4 Implicit Grant Type

Implicit Grant Type [22] es un flujo que simplifica al de *Authorization Code*. Los usuarios continúan autorizando a los clientes de forma explícita. Sin embargo, no se hace uso del código de autorización, sino que el servidor de autorización emite ya directamente el *Access Token*.

Figura 2.8 Implicit flow

En la Figura 2.8 podemos observar dicho flujo.

1) El usuario hace clic en *Login* en la aplicación.
2) La aplicación redirige al usuario al servidor de autorización.
3) El servidor de autorización redirige al usuario a la solicitud de inicio de sesión y autorización.
4) El usuario se autentica y puede ver una página de consentimiento que enumera los permisos le dará a la aplicación.
5) El servidor de autorización redirige al usuario de vuelta a la aplicación con un *Access Token*.
6) La aplicación puede utilizar el *Access Token* para llamar a una API y acceder a información sobre el usuario.
7) La API responde con los datos solicitados.

2.3.1.4.5 OAuth 2.0 Device Authorization Grant

OAuth 2.0 Device Authorization [23] es un flujo posterior a la publicación del estándar OAuth 2.0. Este tiene la finalidad de dar acceso a clientes conectados a Internet que carecen de navegador web o están limitados en cuanto a entrada de datos por parte del usuario. De esta forma, clientes OAuth2 en dispositivos (televisiones, consolas multimedia, marcos digitales de fotos, impresoras…) pueden acceder a los recursos protegidos apoyándose en otro dispositivo del usuario que no tenga las anteriores limitaciones.

2.3.1.4.5.1 Terminología

Antes de ver el flujo en sí, vamos a exponer los siguientes términos:

- *device_code*: Código del dispositivo que está ligado al dispositivo en sí, permitiendo su reconocimiento por parte del servidor de autenticación.
- *user_code*: Código de usuario, el cual ha de ser presentado al usuario, ya sea embebido en la *verification_uri_complete*, mostrado tal cual, en un QR, etc. El usuario ha de identificarse junto con este. Gracias a ello, el servidor de autenticación es capaz de reconocer a la pareja usuario dispositivo.
- *verification_uri*: Dirección web donde el usuario ha de introducir el *user_code*.
- *verification_uri_complete*: Se trata de la *verification_uri* con el *user_code* como parámetro. Útil para mejorar la experiencia de usuario.
- *expires_in*: Tiempo que durará la validez de *device_code* y *user_code*.
- *interval*: Tiempo que el dispositivo puede hacer sondeo. Si este se supera, el servidor de autenticación, pedirá al dispositivo que disminuya la frecuencia de sondeos.

2.3.1.4.5.2 Flujo

En la Figura 2.9 podemos observar dicho flujo. Lo primero que se observa, es la existencia de dos subflujos, el *Browser flow* y el *Device flow*.

El *Browser flow* básicamente habilita la emisión del *token* al dispositivo, al *device code*. Para ello, de alguna forma, el dispositivo le tiene que enviar el *user code* y la URL (*Uniform Resource Locator*) al usuario donde ha de autenticarse y autorizarle, mediante un dispositivo de apoyo (móvil, *tablet*, pc…). Una vez se autoriza, el *token* es emitido a este, el cual ya tiene permitido solicitar recursos.

El *Device flow* por su parte, tiene una serie de fases. La primera es obtención de códigos, envío de estos al usuario y comienzo de sondeo del *token*. La segunda es la autorización del dispositivo por parte del usuario. La tercera es la obtención del *token*, debido a los sondeos en paralelo al B*rowser flow*. Y finalmente, las solicitudes de recursos con el *Access Token*.

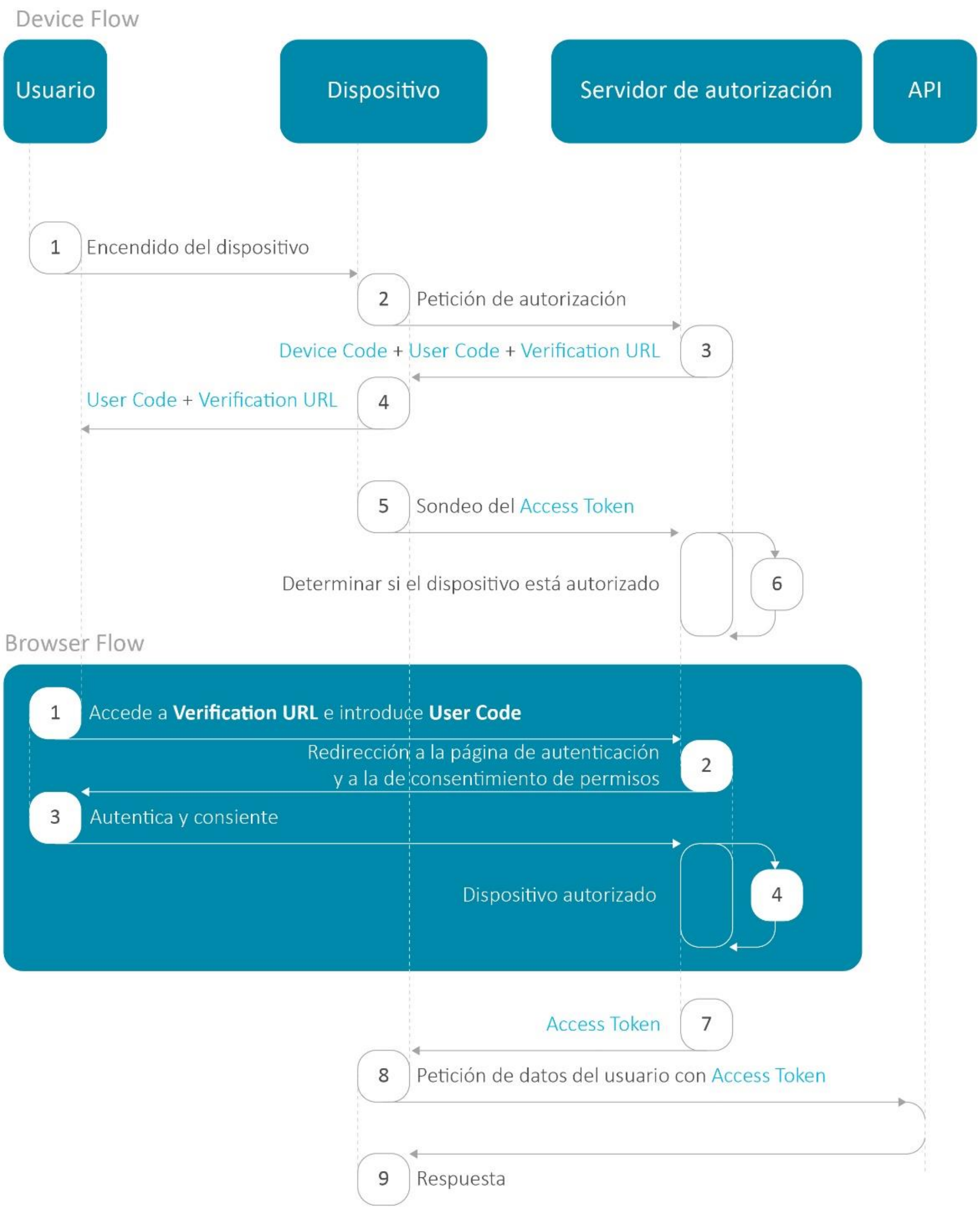


Figura 2.9 Device Authorization flow

El detalle de ambos flujos, tal como se muestra en la Figura 2.9, se describe a continuación:

Device flow:

1) El usuario inicia la aplicación en el dispositivo.
2) La aplicación del dispositivo solicita autorización al servidor de autorización utilizando su Client ID.
3) El servidor de autorización responde con un *device_code*, *user_code*, *verification_uri*, *verification_uri_complete*, *expires_in* (tiempo de vida en segundos para *device_code* y *user_code*), y el *interval*.

4) La aplicación del dispositivo pide al usuario que se active utilizando su ordenador o su *smartphone*. La aplicación puede conseguirlo de las siguientes maneras
    a. pidiendo al usuario que visite la *verification_uri* e introduzca el *user_code* después de mostrar estos valores en pantalla
    b. pidiendo al usuario que interactúe con un código QR (*Quick Response*) o una URL acortada con un código de usuario incrustado generado a partir de la *verification_uri_complete*
    c. navegar directamente a la página de verificación con el código de usuario incrustado utilizando *verification_uri_complete*, si se ejecuta de forma nativa en un dispositivo basado en navegador
5) La aplicación del dispositivo comienza a sondear su servidor de autorización en busca de un *Access Token*. Este continúa hasta que el usuario completa la ruta de flujo del navegador o el código de usuario expira.
6) Cuando el usuario completa con éxito el *Browser flow*, el servidor de autorización responde con un *Access Token* (y, opcionalmente, un R*efresh Token*). El *device_code* expirará.
7) La aplicación del dispositivo puede utilizar el *token* de acceso para llamar a una API y acceder a información sobre el usuario.
8) La API responde con los datos solicitados.

Browser flow:

1) El usuario visita la *verification_uri* en su navegador, introduce el *user_code* y confirma que el dispositivo que se está activando muestra el *user_code*. Si el usuario visita la *verification_uri_complete* por cualquier otro mecanismo (como escanear un código QR), sólo será necesaria la confirmación del dispositivo.
2) El servidor de autorización redirige al usuario a la solicitud de inicio de sesión y consentimiento, si es necesario.
3) El usuario se autentifica utilizando una de las opciones de inicio de sesión configuradas y puede ver una página de consentimiento solicitando autorizar la aplicación del dispositivo.
4) La aplicación de dispositivo está autorizada para acceder a la API.

A modo de resumen, este flujo está diseñado para aplicaciones en dispositivos que tengan impedimentos en la introducción de credenciales o navegación web. El usuario da consentimiento explícito a través del navegador de su móvil, ordenador o cualquier otro equipo bajo su control. La confianza en el dispositivo es de tercero no confiable. La comunicación es de cliente servidor.

2.3.1.5 Decisión de grants

En la Tabla 2.2, recogemos las variables en las que nos debemos de fijar a la hora de elegir el flujo que más se adecua a nuestro caso.

Tabla 2.2 Tabla de decisión

	Client Credentials	*Resource Owner Password*	*Authorization Code*	*Implicit*
Tipo de aplicación consumidora	Aplicaciones internas	Aplicaciones móviles, escritorio, web, si son confiables, no terceros	Aplicaciones móviles, escritorio, web…	Aplicación de una sola página
Interactuación por parte del usuario	Ninguna	Usuario y contraseña	Consentimiento explícito a través de un navegador.	Consentimiento explícito a través del navegador
Grado de confianza de la aplicación	Elevado	Tercero confiable o la propia organización	Un tercero no confiable	Casos muy específicos
Tipo de comunicación	Entre servidores	Cliente-servidor	Cliente-servidor	Solo cliente, sin servidor

2.3.2 OpenId Connect

OpenID Connect (OIDC) [24] es un protocolo de identidad que usa los mecanismos de autenticación y autorización de OAuth 2.0.

Extiende a OAuth en cuanto a autenticación utilizando *tokens* JWT (*JSON Web Tokens*) [25], añadiendo un tipo de *token* extra y un *scope* con valor “openid”.

Los *tokens* JWT son objetos JSON (*JavaScript Object Notation*) con una estructura básica predeterminada y codificados en Base64. En la Figura 2.10 se observa dicha estructura, donde en la cabecera se incluye la información relativa al algoritmo empleado en la firma y metainformación. El *payload* incluye la información que contiene el *token* y cuyo formato es libre. Así, por ejemplo, para el caso que nos conlleva, incluye datos del usuario como correo electrónico o nombre, también la fecha de expedición, el emisor del *token* o para quien va dirigido, entre otros. La última parte se trata de la firma de todo el contenido, de manera que se asegura la integridad del *token* además de la identidad del emisor.

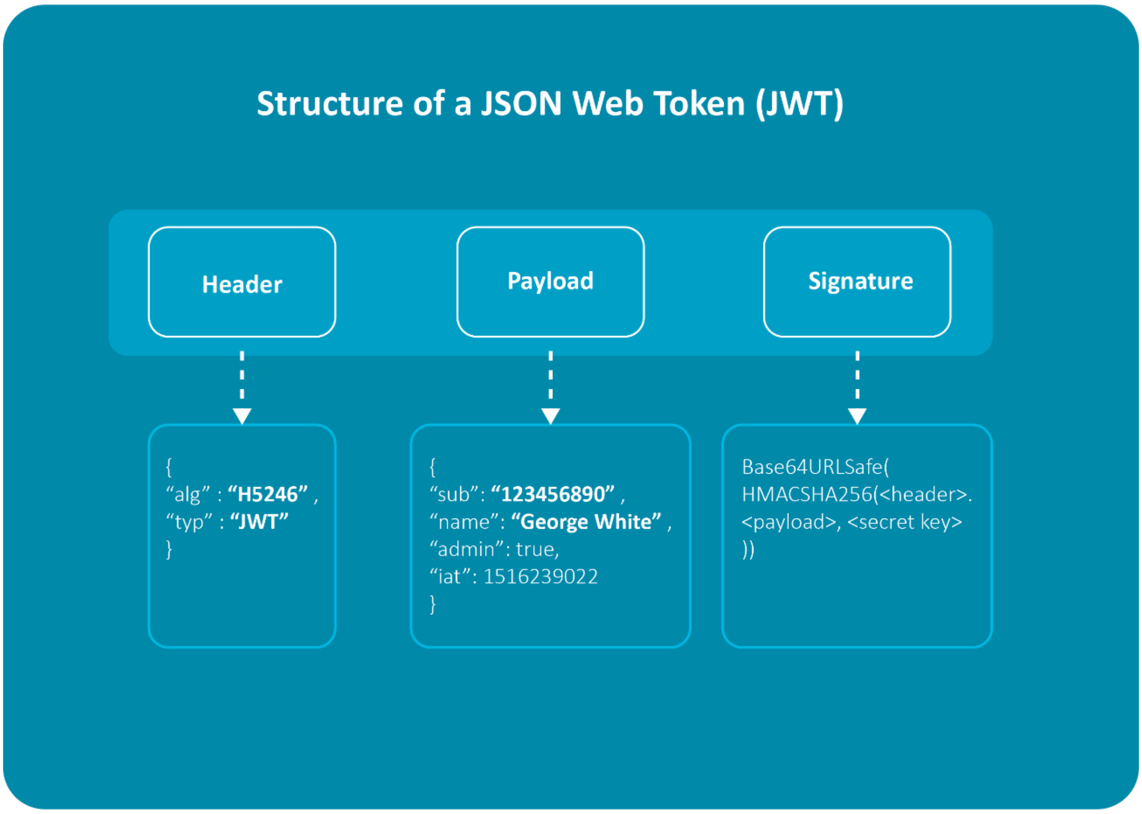


Figura 2.10 Estructura de un token JWT

2.3.2.1 Terminología

En el estándar de OIDC se definen los términos de OIDC *Provider* y *Relying party*. El primero simplemente es como llama OIDC al IdP. Sus tareas son autenticar a usuarios, alojar y validar los consentimientos asociados al usuario y la emisión de *tokens*.

El RP (*Relying Party*) se corresponde con el cliente o servicio que solicita la identidad de un usuario.

2.3.2.2 Tipos de token

Como OICD contiene al protocolo OAuth 2.0, también tiene integrado sus *Access* y *Refresh Tokens*, cuya descripción ha sido realizada anteriormente. Sin embargo, OIDC añade un *token* más, el *ID Token*. Este tiene la finalidad de proporcionar información sobre el resultado de la autenticación. En su interior, podemos encontrar datos de identidad del usuario, también llamados *claims*. Ejemplos de *claims* son nombre de usuario, correo electrónico, ID persistente o cualquier parámetro que considere la *Relying Party*.

2.3.2.3 Flujos

Los flujos de OIDC son heredados de OAuth 2.0. Por lo tanto, nos encontramos con *Authorization Code flow*, Implicit flow, etc. Sin embargo, OIDC no se queda solo con estos, también añade uno, el *Hybrid flow* [26], que se puede considerar un punto intermedio entre *Authorization Code flow* e *Implicit flow*.

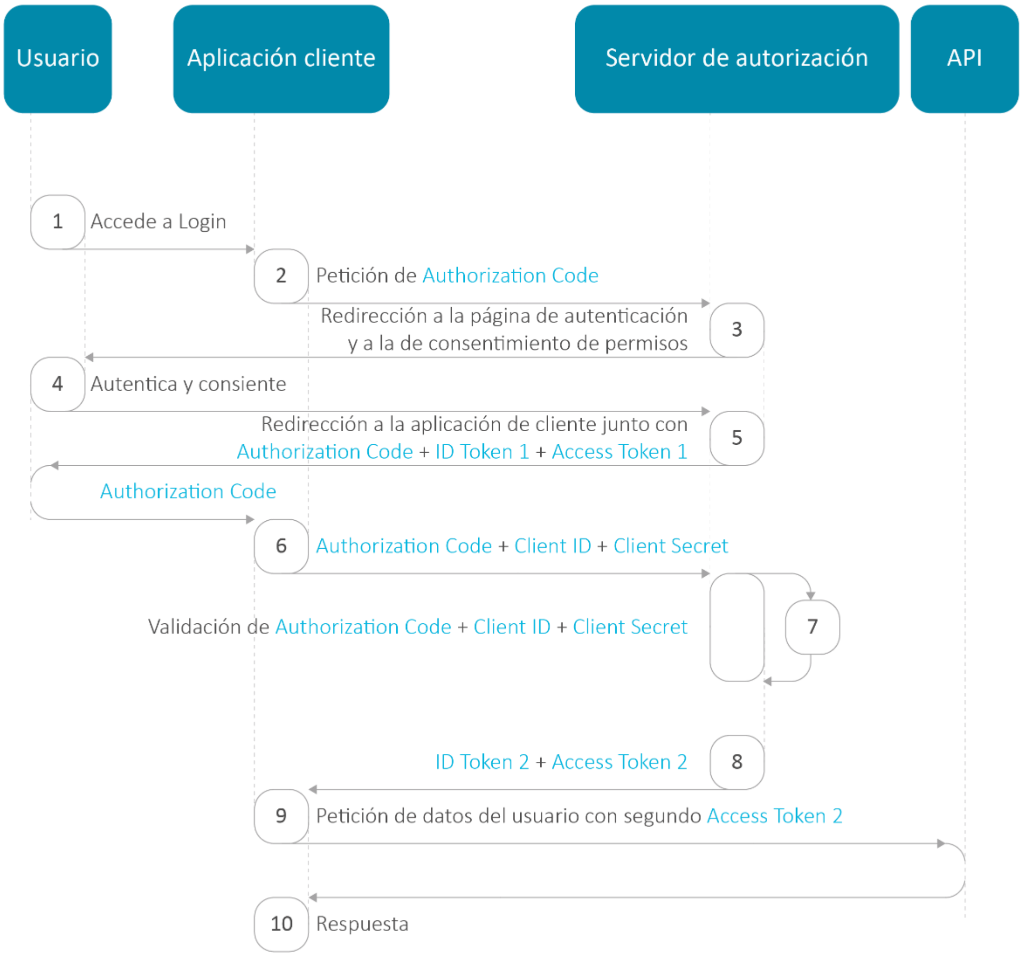


Figura 2.11 Hybrid flow

En la Figura 2.11 podemos observar dicho flujo:

1) El usuario hace clic en *Login* dentro de la aplicación.
2) La aplicación redirige al usuario al servidor de autorización.
3) El servidor de autorización redirige al usuario a la solicitud de inicio de sesión y autorización.
4) El usuario se autentica usando una de las opciones de inicio de sesión configuradas y puede ver una página de consentimiento que enumera los permisos que le dará a la aplicación.
5) El servidor de autorización redirige al usuario de vuelta a la aplicación con un código de autorización (un solo uso), más un *ID token*, *Access Token*, o ambos, dependiendo del tipo de respuesta solicitada.
6) La aplicación envía el código al servidor de autorización junto con el *Client ID* y el *Client Secret* de la aplicación.
7) El servidor de autorización verifica el código, el *Client ID* y el *Client Secret*.
8) El servidor de autorización responde con un segundo *ID Token* y un *Access Token* (y, opcionalmente, un *Refresh Token*).

9) La aplicación puede utilizar el segundo *Access Token* para llamar a una API y acceder a información sobre el usuario.
10) La API responde con los datos solicitados.

A modo de nota, este flujo es idéntico al *Authorization Code flow*, solo que el usuario también obtiene un *token* diferente al de la aplicación (trama 5 de ambos flujos).

2.3.3 SAML

Security Assertion Markup Language (SAML) [27] es un estándar abierto que permite el inicio de sesión único (SSO). Se intercambian, mediante esquemas XML (*Extensible Markup Language*) llamados SAML *Assertions*, datos relativos a la autorización y autenticación entre los proveedores de servicios y los proveedores de identidad.

2.3.3.1 Roles

La especificación SAML define los roles principales, proveedor de identidad y proveedor de servicio. El principal es aquel que solicita un servicio a un proveedor de servicios.

En cuanto a los proveedores no debemos de confundirlos. El proveedor de servicio (SP, *Service Provider*), como su propio nombre indica, es el encargado de conceder servicios al principal. Mientras que el proveedor de identidad (IdP, *Identity Provider*), es el encargado de emitir la información relevante a la autenticación y autorización del usuario que quiere acceder a los recursos.

2.3.3.2 SAML Assertion

No vamos a profundizar en exceso como son estas, pero de manera general vamos a ver que tipos existen y un ejemplo de esta.

Existen 3 tipos. El primero de ellos es *Authentication Assertions*, las cuales prueban la identificación del usuario y proporcionan la hora en que el usuario se conectó y el método de autenticación que utilizó. Luego están las *Attribution Assertions*. Estas pasan los atributos SAML al proveedor de servicios. Los atributos SAML son piezas específicas de datos que proporcionan información sobre el usuario. Por último, las *Authorization Decision Assertions* dicen si el usuario está autorizado a utilizar el servicio o si el proveedor de identificación denegó su solicitud debido a un fallo de la contraseña o a la falta de derechos sobre el servicio.

En la Figura 2.12 observamos un extracto de una *Assertion*.

```xml
<?xml version="1.0"?>
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="u
  <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer><ds:Signature xm
  <ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#pfx44576a12-7401-6704-0bf5-77de5b7e4653"><ds:Transforms><d
<ds:KeyInfo><ds:X509Data><ds:X509Certificate>MIICajCCAdOgAwIBAgIBADANBgkqhkiG9w0
  <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
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer><ds:Signature
  <ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#pfx48e9ce1f-1080-7322-abde-d2ed446da748"><ds:Transforms><d
<ds:KeyInfo><ds:X509Data><ds:X509Certificate>MIICajCCAdOgAwIBAgIBADANBgkqhkiG9w0
    <saml:Subject>
      <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php" Fo
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipi
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2014-07-17T01:01:18Z" NotOnOrAfter="2024-01-18T0
      <saml:AudienceRestriction>
        <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
      </saml:AudienceRestriction>
```


Figura 2.12 Ejemplo de respuesta SAML firmada con Assertion

2.3.3.3 Intercambio SAML

En la Figura 2.13 se muestra el intercambio SAML habitual. Alice (principal) quiere acceder a la consola de administración de un servicio de gestión de recursos (SP). Al intentar acceder al servicio, el SP solicita la información de identidad de Alice. Si la confirmación del IdP es exitosa, este le envía una confirmación de dicha identidad. Ahora es el SP quien con la respuesta decide si Alice puede acceder o no a dicha consola. Finalmente, si todo es correcto, se le concede el acceso.

2.3.3.4 Resumen

Por una parte, SAML supone una simplificación con respecto a la gestión de todos los inicios de sesión por separado. Es por ello, que se reducen costes y los usuarios ganan en accesibilidad, ya que pueden acceder a diferentes SP iniciando sesión una sola vez.

Por otra parte, la seguridad aumenta gracias a la eliminación de credenciales adicionales.

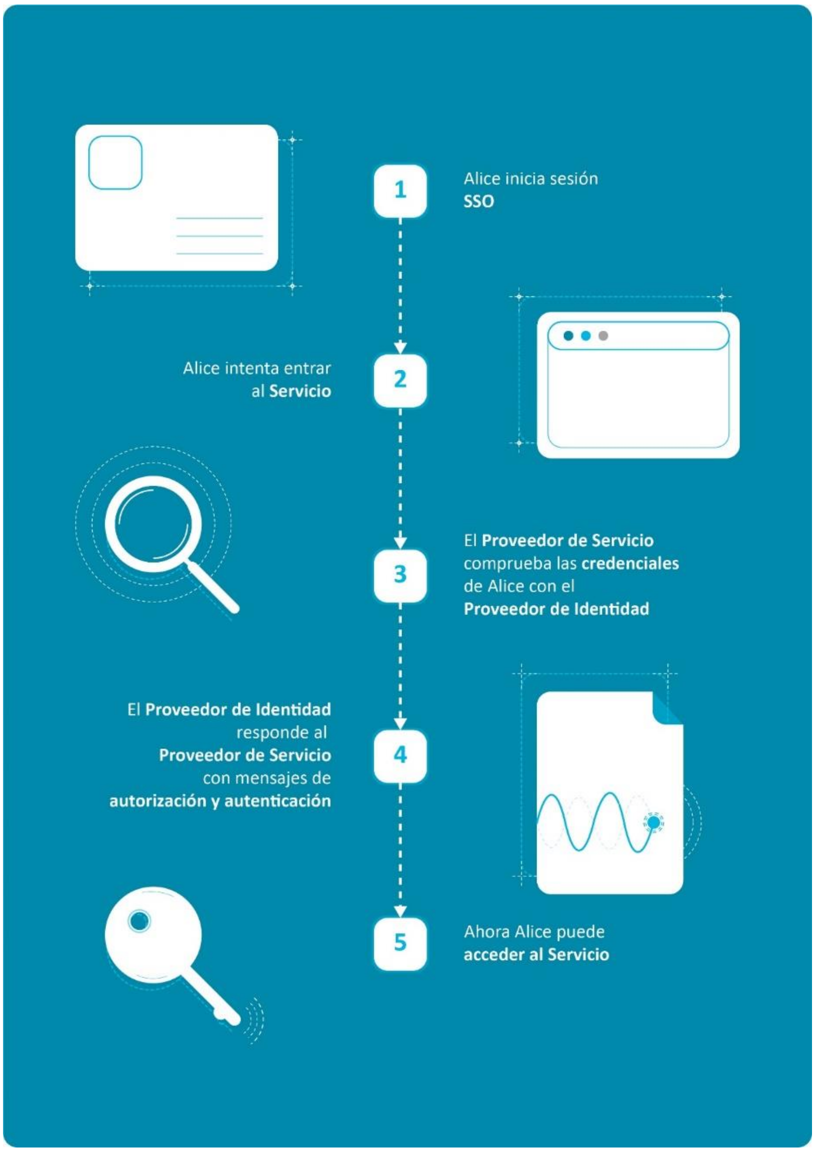


Figura 2.13 Ejemplo de SAML

2.3.4 Comparativa protocolos de autenticación y autorización

En la Tabla 2.3 podemos observar las principales diferencias entre los protocolos vistos anteriormente.

Tabla 2.3 Comparativa Protocolos

	OAuth 2.0	*OIDC*	*SAML*
Autorización	Si	No	Si
Autenticación	No	Si	Si
Confianza	Alta	Media	Alta
Características	Media	Media	Alta
Requisitos básicos	Atractivo (más fácil)	Atractivo (más fácil)	Manipulación Pesada
Codificación de datos	JWT u otros	JWT	XML
Ejemplos de aplicaciones	Dispositivos móviles, aplicaciones web, móviles…	Aplicaciones móviles/web	Servicios estatales, autenticación empresarial
Procesado de datos	Ligero	Ligero	Pesado
SSO	No	Si	Si

3 Diseño

Una vez que tenemos una visión general de soluciones IAM y protocolos disponibles, vamos a analizar la problemática que se pretende solventar, optar por una solución justificada de todas las posibles y describirla en detalle.

3.1 Descripción del problema

El principal problema que se ha de solucionar en este proyecto es la protección de la plataforma de acceso a la información de los sensores disponibles en Smart Santander, es decir, debemos de ser capaces de controlar el acceso a los recursos sin modificar nada en Smart Santander. En concreto, debemos de ser capaces de, inicialmente, diferenciar dos tipos de usuarios, usuarios básicos y administradores o usuarios avanzados. Los usuarios tienen que ser capaces de acceder únicamente a ciertas métricas de los sensores distribuidos. Sin embargo, los administradores tienen que ser capaces de acceder a cualquier métrica, entre ellas las avanzadas como los niveles de batería, además de poder realizar operaciones adicionales sobre los sensores o la creación de subscripciones, etc.

La solución debe ser capaz de integrar también los procedimientos de creación y recuperación de cuentas, el inicio de sesión, etc., siguiendo procedimientos avanzados de validación de las credenciales de los usuarios a través de correo electrónico, OTP u otros.

Por otra parte, además de la autenticación y autorización en el acceso a los recursos, ésta debe ser posible de realizarse desde cualquier dispositivo que tenga conexión a Internet, independientemente de las características de estos (sin interfaz de usuario o navegador web, sin teclado, etc.).

3.2 Planteamiento de la solución

Para poder solucionar estas dos problemáticas, vamos a optar por la siguiente solución.

De entre todos los IAM disponibles, vamos a emplear Keycloak, puesto que como podemos observar en Tabla 2.1, el resto no son tan completos. Así entre las razones por las que se ha seleccionado este IAM frente a otros se considera que es un producto *open software*, gratuito y muy flexible, lo que nos permite a futuro ampliar las funcionalidades. Su despliegue es sencillo y existen constantes actualizaciones que lo mejoran y hacen más seguro. La documentación es muy completa y tiene una gran comunidad detrás que lo afianza. Además, se puede desplegar en contenedores y se adapta a multitud de servicios de distintas naturalezas. Integra los protocolos estándar de uso más común, lo cual facilita su utilización en todo tipo de situaciones y dispositivos.

Una vez seleccionado el IAM, debemos elegir el procedimiento empleado para la autenticación y autorización. Las alternativas ya descritas son SAML y OpenID Connect

(OIDC). Para el caso del presente trabajo, vamos a elegir OIDC. De esta forma tenemos asegurada tanto la autenticación como la autorización, ya que OIDC se sustenta sobre los intercambios y modelos de datos de OAuth2. Esta alternativa también presenta la ventaja que la aplicación en cuestión no recibe las credenciales del usuario, ya que estas se validan directamente en Keycloak, quien retorna un *token* que integra la identificación y las condiciones de acceso. Adicionalmente, el protocolo OAuth2 incorpora un modelo de intercambio, *Device flow*, que solventa de forma natural la accesibilidad de dispositivos controlados por usuarios. A pesar que SAML es más completo en cuanto a características, es más pesado en cuanto a realizar tareas básicas. OIDC y OAuth2 son seguros y están fuertemente testeados. En Tabla 2.3 observamos las distintas características descritas anteriormente de todos los protocolos.

De entre los flujos proporcionados por OAuth2, vamos a desplegar y evaluar dos de ellos, dependiendo del caso de uso. Si el usuario accede de manera tradicional desde ordenadores o móviles con interfaz de usuario, de entre los flujos disponibles, el utilizado será el *Authorization Code flow*. En este caso, no se confía en la aplicación y ésta en lugar de las credenciales del usuario recibe un *token* que habilita el acceso como el usuario en cuestión. Está orientado a aplicaciones web, entre otras. En Tabla 2.2 podemos ver este flujo con respecto a los otros.

En el caso que el usuario emplee un dispositivo que no disponga de navegador web o interfaz de interacción amigable y completo, vamos a optar por *Device Authorization flow*. Esta decisión es obvia, ya que no tiene alternativas como tal. Es simplemente el protocolo que soluciona todos los problemas de accesibilidad, otorgando los *tokens* a los dispositivos.

3.3 Arquitectura de alto nivel

En la Figura 3.1 podemos observar la arquitectura de la solución propuesta. Esta se compone de tres grandes bloques funcionales: a la izquierda se encuentran los clientes con diferentes dispositivos, a la derecha encontramos el servicio, Smart Santander, y, por último, en el centro, encontramos el *proxy* y Keycloak, como IdP, los cuales gestionan el acceso al servicio por parte de los clientes.

Como se comentó en la descripción del problema, Keycloak requiere interactuar con el servicio a proteger, lo que supone introducir modificaciones en el servidor a proteger. Para eludir dicho inconveniente, optamos por elaborar un *proxy*. Este *proxy* se introducirá entre el usuario y el conjunto de servicios ofrecidos por Smart Santander. Este desarrollo usará las librerías de Keycloak y ocultará todos los procesos de autenticación y autorización logrando así no solo proteger, sino también no tener que modificar los servicios actualmente en ejecución de Smart Santander. Cuando el usuario cumpla con todas las condiciones necesarias, el *proxy* accederá a la información requerida de los sensores. La conexión entre el *proxy* y la plataforma de servicios de Smart Santander se supone segura y con acceso únicamente habilitado para el *proxy*. En el *proxy* de la Figura 3.1, podemos observar sus tres componentes esenciales: servidor web, necesario para presentar los recursos a los usuarios; cliente HTTP (*Hypertext Transfer Protocol*), para pedir los recursos solicitados por los usuarios a Keycloak, y el último, el adaptador Keycloak, que proporciona el conjunto de funcionalidades que permiten realizar intercambios con Keycloak de forma transparente para facilitar los procesos de autenticación y autorización.

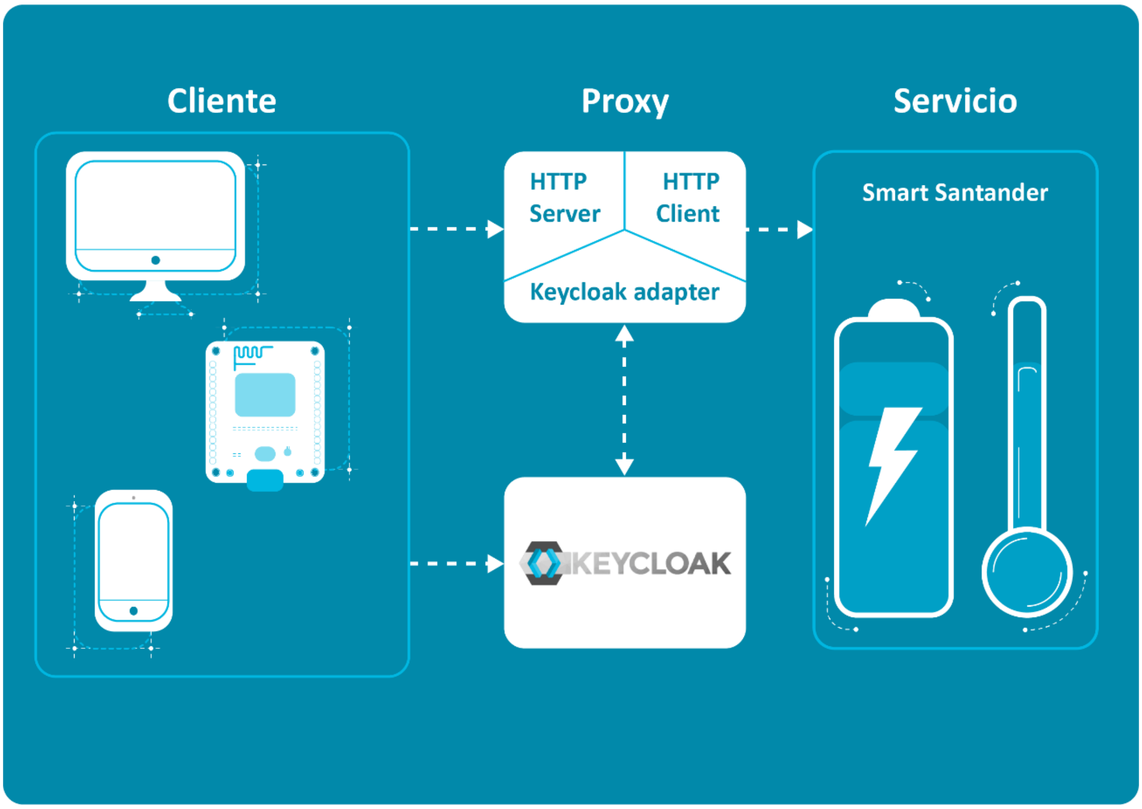


Figura 3.1 Arquitectura propuesta

Entre los diferentes adaptadores que existen de Keycloak, optamos por el basado en Node.js [28]. Node.js es un entorno de ejecución de código abierto, multiplataforma y *back-end* que se ejecuta en un motor de JavaScript. Esto deriva en programación asíncrona y arquitectura orientada a eventos. Podríamos haber elegido otro, pero se ha considerado esta elección pues, además de que es la tecnología usada en el API de Smart Santander, nos permite desarrollar el *proxy* y conectarnos a Keycloak de manera sencilla. Es cierto que a nivel de seguridad y otras características pueden existir mejores productos, pero no es objeto de este proyecto, cumpliendo Node.js con los requisitos necesarios.

Por último, en cuanto a la arquitectura interna del *proxy*, usaremos Axios [29] para elaborar el cliente HTTP. La interfaz de interacción con el cliente se elaborará mediante Express [30]. Existen otras opciones para realizar dichas funciones, pero de nuevo, estas son completas y sencillas de implementar con respecto a otras.

4 Evaluación del sistema

Antes de realizar el despliegue de la arquitectura propuesta en el capítulo 3, vamos a profundizar en la operativa de la herramienta de Keycloak, haciendo hincapié en su instalación y configuración para proteger un servicio y en la realización de las pruebas pertinentes para comprender y validar su funcionamiento.

4.1 Introducción a Keycloak

Como hemos visto en apartados anteriores, Keycloak es una herramienta SSO para aplicaciones y servicios web. Su principal objetivo es asegurar aplicaciones y servicios de una organización de manera fácil y sencilla. Se puede utilizar como plataforma de integración para conectarlo a servidores LDAP y *Active Directory*. También puede delegar la autenticación a terceros proveedores de identidad como Facebook y Google.

4.1.1 Terminología

Para entender correctamente las descripciones que a continuación se realizarán, es necesario entender la terminología básica de Keycloak, y de los actores que maneja.

Los usuarios son entidades que pueden acceder a un sistema. Pueden tener atributos asociados a sí mismos como correo electrónico, nombre de usuario, dirección, número de teléfono o día de nacimiento, conformando lo que se denomina perfil. Se les puede asignar roles específicos.

Conjuntos de usuarios conforman grupos, permitiendo simplificar la gestión de dichos usuarios. Se pueden definir atributos para un grupo o asignarle roles. Los usuarios que se convierten en miembros de un grupo heredan los atributos y las asignaciones de funciones y roles que el grupo define.

Los clientes son entidades que pueden solicitar a Keycloak la autenticación o autorización de un usuario. No debemos de confundirlos con usuarios. La mayoría de las veces, los clientes son aplicaciones y servicios que quieren utilizar Keycloak para asegurarse y proporcionar una solución de inicio de sesión único. Los clientes también pueden ser entidades que sólo quieren solicitar información de identidad o un *token* de acceso para que puedan invocar de forma segura otros servicios en la red que están asegurados por Keycloak. En este sentido, se puede habilitar que únicamente clientes autorizados (previamente configurados) sean reconocidos por Keycloak.

Las aplicaciones suelen asignar el acceso y los permisos a tipo o categoría de usuario específicos en lugar de a usuarios individuales, ya que tratar con los usuarios puede ser un grado de granularidad demasiado fino y difícil de gestionar. A dichas categorías se le conoce como rol.

Los elementos anteriores se agrupan en dominios o *realms*. Un dominio gestiona un conjunto de usuarios, credenciales, roles, grupos y clientes. Los usuarios pertenecen a estos, se registran en ellos y están aislados de otros usuarios de dominios distintos. Lo mismo ocurre con los roles, clientes, etc. definidos en el ámbito de un dominio.

Resta ahora, tras describir quienes acceden y cómo se agrupan, entender los términos y conceptos relativos a la autorización de Keycloak.

Un recurso es un objeto de una aplicación o servicio el cual hay que proteger. Puede ser un conjunto de uno o más *endpoints*. Un ejemplo de recurso en Smart Santander sería la temperatura de un sensor específico alojado en la ruta `/temperatura/num_sensor`. Otro recurso podría ser el alojado en `/batería/*`, el cual engloba a todos los sensores.

Para Keycloak, un servidor de recursos es el servidor que aloja los recursos protegidos y es capaz de aceptar y responder a las solicitudes de estos. Cualquier aplicación cliente confidencial puede actuar como servidor de recursos. Los recursos de este cliente están protegidos y gobernados por un conjunto de políticas de autorización.

Sin embargo, en muchas ocasiones se quiere limitar las acciones sobre un determinado recurso, por ejemplo, permitir solamente la lectura y restringir su edición. Es ahí cuando surgen los *scopes*, es decir, lo que se puede hacer con un determinado recurso. Ejemplos de *scopes* son ver, editar, eliminar, etc.

Por otra parte, una política define las condiciones que deben cumplirse para conceder el acceso a un objeto, sin especificar dicho objeto. Están fuertemente relacionadas con los diferentes mecanismos de control de acceso (ACM, *Access Control Mechanism*) que se utiliza para proteger recursos. Con las políticas, se puede implementar estrategias para el control de acceso basado en atributos (ABAC, *Attribute Based Access Control*), el control de acceso basado en roles (RBAC, *Role Based Access Control*), el control de acceso basado en el contexto, o cualquier combinación de estos.

Finalmente, señalar que en Keycloak la forma de relacionar los recursos o *scopes* con las políticas son los permisos. Estos deben de evaluarse para ver si se concede el acceso.

4.1.2 Arquitectura

En la Figura 4.1 podemos observar la composición lógica de Keycloak. Principalmente, como la mayoría de entornos de autenticación y autorización se conforma de 4 módulos, PAP (*Policy Administration Point*), PDP (*Policy Decision Point*), PEP (*Policy Enforcement Point*), PIP (*Policy Information Point*).

El PEP es el punto de aplicación de las políticas, es decir, controla el acceso a los servidores de recursos interceptando las peticiones. Dependiendo de la naturaleza del servidor de recurso, existen diferentes tipos soluciones. Interroga al PDP para solicitar peticiones de autorización. Es un mero actuador, que funciona como si de un portero se tratara.

El PDP recibe las solicitudes de autorización, y apoyándose en el PIP, es capaz de obtener todos los datos relativos a dicha solicitud, evaluarlos y emitir una respuesta. El PIP es el elemento que gestiona la información relativa a las autorizaciones, es decir, a recursos, *scopes*, políticas, atributos de identidades, datos del usuario, etc. Se puede considerar que actúa como una base de datos de propósito específico.

El último módulo es el PAP. Este básicamente está vinculado a las funcionalidades relacionadas con el entorno de administración. Proporciona un interfaz visual para gestionar servidores de recursos, recursos, ámbitos, permisos y políticas. Parte de esto también se logra de forma remota mediante el uso de *Protection* API, la cual ofrece un conjunto de *endpoints* que proporcionan gestión de recursos, gestión de permisos, gestión de políticas, etc.

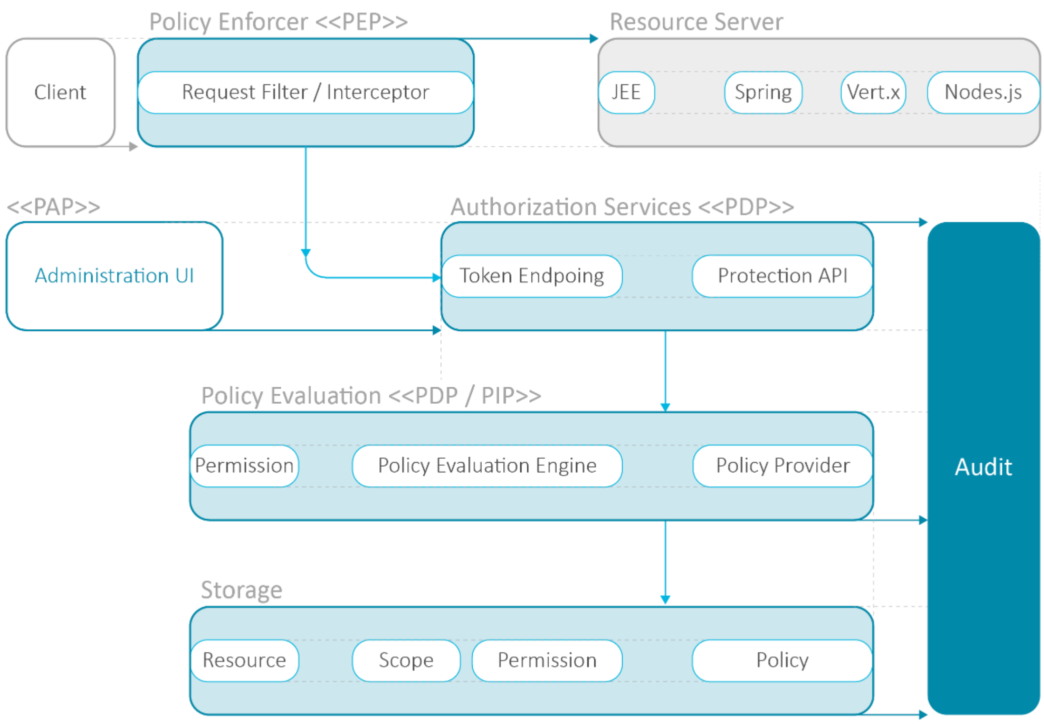


Figura 4.1 Arquitectura de Keycloak

Llegados a este punto, entendiendo los conceptos y la arquitectura, se procede a la instalación y configuración de Keycloak.

4.2 Instalación

Antes de comenzar con la instalación, vamos a asentar el entorno de trabajo. El sistema operativo en el que vamos a trabajar va a ser Ubuntu 20.04.4 LTS. Linux es un entorno ampliamente adoptado para desarrollo y Ubuntu es una de las distribuciones de Linux más comúnmente empleadas.

En cuanto a la instalación de Keycloak tenemos varias formas de realizarlo: tradicional, a través contenedores de imágenes para Docker, Podman, etc. e incluso específicas para operadores, como Kubernetes y OpenShift. Esto lo observamos en el apartado de descargas [31]. Para el proyecto se ha seleccionado el despliegue a través de contenedores Docker [32].

4.2.1 Docker

Docker es un programa *open software* que permite virtualizar aplicaciones, junto con las dependencias y librerías necesarias en lo que se denomina un contenedor. Estos contenedores son 100% portables a otros entornos, indistintamente de su naturaleza, gracias a la virtualización a nivel de sistema operativo.

Cuando se ejecuta un contenedor, este se encuentra aislado del resto del sistema u otros contenedores, de forma que nada va a interferir con este. Los contenedores se caracterizan por ser ligeros y rápidos de iniciar en comparación con otras alternativas de virtualización. Esto convierte a Docker es un excelente entorno para el despliegue y distribución de soluciones, ya que garantiza la misma operativa independientemente del sistema operativo base sobre el que se ejecutan los contenedores. Su estanqueidad permite evaluar de forma limpia diferentes aplicaciones, servicios o herramientas. Si algo no funciona correctamente o la aplicación se ha corrompido de repente, se puede restaurar fácilmente el estado o volver a lanzar un nuevo contenedor, el cual operará como una nueva instalación limpia. Si esto mismo lo debemos de realizar en un entorno clásico, nos llevaría mucho más tiempo de desinstalación y limpieza, o bien, en casos extremos, deberíamos de formatear.

Existen diversos repositorios donde la comunidad pone a disposición contenedores Docker para su utilización de forma libre y abierta [33].

Por todas estas ventajas, se considera que el uso de Docker es una excelente opción para desplegar Keycloak. Si no se encuentra instalado, basta con ejecutar el siguiente comando:

```
$ sudo apt install docker-ce
```

4.2.2 Instalación de Keycloak

Para descargar y lanzar el contenedor con Keycloak, podemos ejecutar el siguiente comando:

```
$ docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e
  KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:18.0.2 start-dev
```

Básicamente este comando descarga el contenedor (*pull*) del repositorio, si es la primera vez que se instala, y se ejecutar la lógica interna (*run*). El parámetro `p` configura como se realiza la correspondencia entre los puertos del contenedor y la máquina que lo aloja. Así, para este caso, el puerto 8080 interno del contenedor se corresponde o redirige al 8080 del OS (*Operating System*). Las variables de entorno se emplean para configurar parámetros internos del contenedor de forma sencilla. Para el contenedor específico de Keycloak, las variables incluidas permiten crear el usuario administrador “`admin`” con contraseña “`admin`”.

A modo de aclaración este contenedor emplea la versión 18.0.2 de Keycloak, que es la más reciente en el momento de la realización del proyecto, pero se espera una nueva versión en un futuro no muy lejano.

La ejecución exitosa del comando expone nuestro Keycloak en la URL `http://localhost:8080`, donde tras ingresar con las credenciales de administrador, entramos a la consola de gestión. En la Figura 4.2 podemos observar dicho panel.

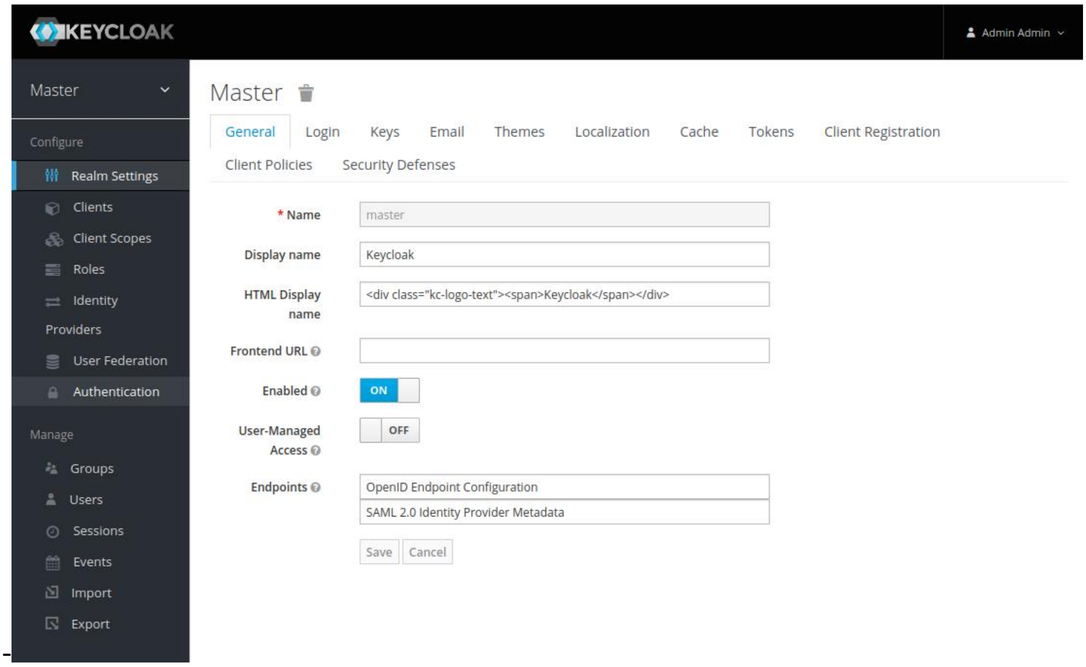


Figura 4.2 Consola de administrador

4.3 Configuración

Una vez lo tenemos instalado iremos configurando todo lo necesario para llevar a cabo la solución propuesta de manera local.

Lo primero que vamos a realizar es a crear un *realm*. De manera predeterminada, Keycloak trae el *realm* `Master`. Sin embargo, vamos a crear uno para configurar nuestro proyecto. A modo de ejemplo, lo llamaremos “`smartsantander`”.

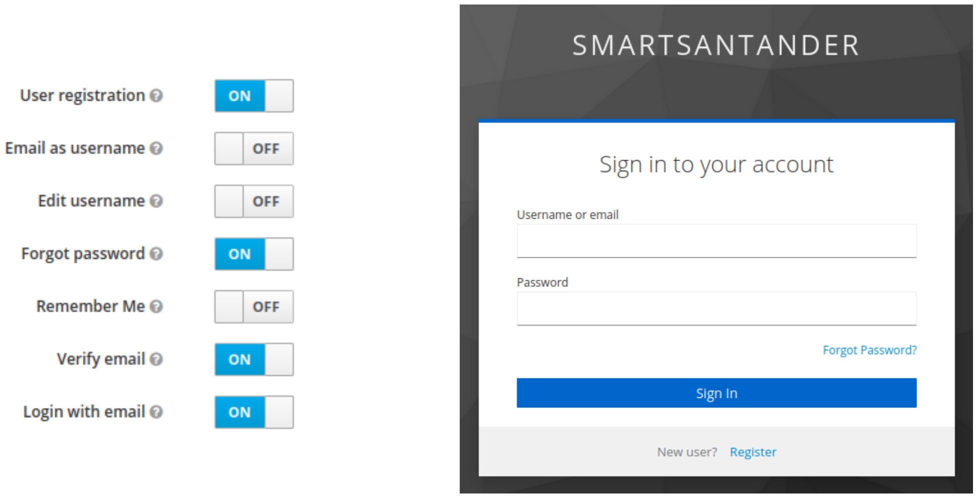
4.3.1 Configuración de formularios en login

Una vez tenemos nuestro *realm*, vamos a habilitar el registro de nuevos usuarios, recuperar contraseña y verificación de correos de los usuarios. Estas son funcionalidades indispensables y básicas en un login corporativo, entidad gubernamental, etc.

Para ello, dentro del menú `Realm Settings > Login`, seleccionamos las mencionadas anteriormente.

También vamos a permitir que las conexiones sean en HTTP (*HyperText Transfer Protocol*), y no en HTTPS (*HyperText Transfer Protocol Secure*), ya que queremos observar y capturar el tráfico. Para ello seleccionamos *none* en Require SSL (*Secure Sockets Layer*). En la documentación [34] podemos encontrar cómo se ha de configurar Keycloak para habilitar HTTPS. Esta medida se realiza ya que estamos en un entorno de desarrollo. En fase de despliegue de ninguna manera debemos de permitir la operativa en HTTP sin seguridad.

Por lo tanto, la configuración sería la mostrada en la Figura 4.3a. Una vez guardado, los formularios de nuestros futuros servicios tendrán las características descritas anteriormente. En la Figura 4.3b observamos cómo quedaría el inicio de sesión, donde apreciamos "*Forgot Password?*" y "*Register*".



a) Configuración **b) Formulario**

Figura 4.3 Inicio de sesión

4.3.2 Configuración del correo electrónico

Para poder verificar cuentas de usuarios, modificar o recordar contraseñas, o enviar acciones a usuarios desde gestión, como es el caso de configuración OTP o actualización del perfil del usuario, es necesario que Keycloak disponga de un servicio de STMP (*Simple Mail Transfer Protocol*) para su ejecución.

Para configurarlo, debemos de acceder a `Realm Settings > Email`. Dependiendo del servidor y su configuración, rellenaremos unos campos u otros. Sin embargo, se recomienda que se acceda con TLS (*Transport Layer Security*) y con autenticación.

Para probar la funcionalidad de STMP de Keycloak, hemos usado una cuenta de Google. A modo de ejemplo, en la Figura 4.4 observamos cómo quedarían los campos de `Email`. El acceso a la cuenta, en este caso de Google, se puede confrontar con el botón de `Test connection`.

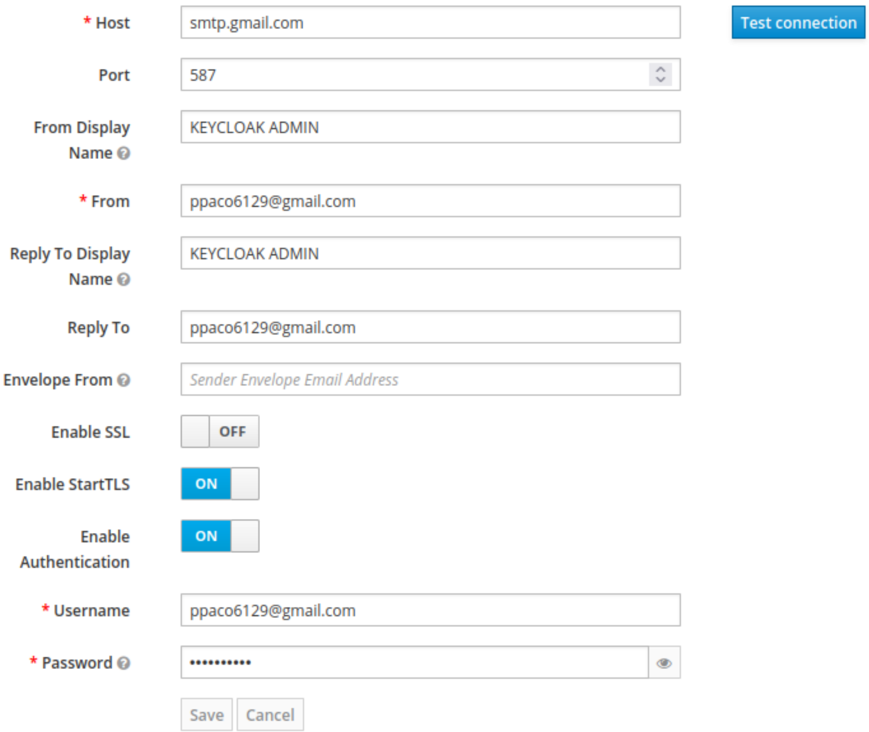


Figura 4.4 Configuración de Email

4.3.3 Configuración OTP

Para incrementar la seguridad de los usuarios, vamos a requerir el uso de un segundo factor de autenticación mediante OTP. Así los usuarios no solo van a tener que introducir usuario y contraseña, sino que también tendrán que validarse también con contraseñas de un solo uso.

Por defecto, Keycloak proporciona los formularios necesarios para desplegar la autenticación basada en OTP. A través del menú `Authentication > Required Actions`, para obligar a los usuarios a que empleen OTP cuando creen la cuenta debemos de seleccionar `Default Action Configure OTP`. En la Figura 4.5 podemos observar cómo quedarían las `Require Actions`.

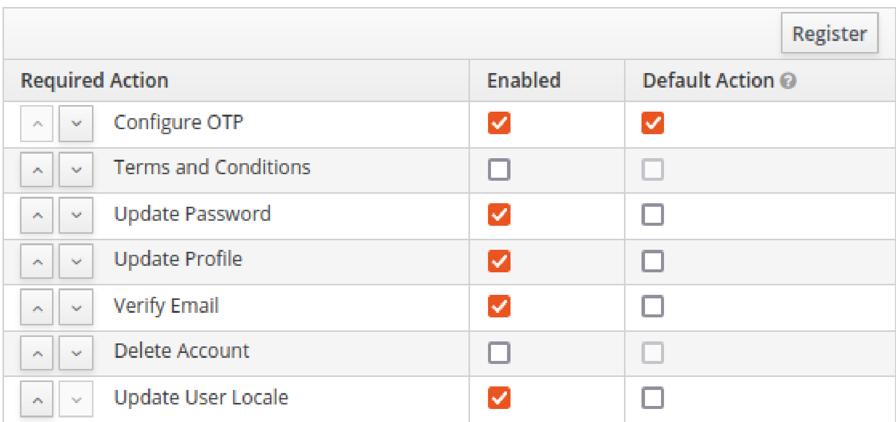


Figura 4.5 Configuración por defecto de OTP

Adicionalmente, hay que definir los parámetros y características del sistema OTP que se emplee. La configuración de los mismos se encuentra en `Authentication > OTP Policy`.

Así podemos seleccionar el tipo de OTP que queremos. El TOTP (*Time-based One-Time Password*) [35] se considera un poco más seguro porque el OTP coincidente sólo es válido durante un corto periodo de tiempo, mientras que el OTP para el HOTP (*HMAC (Hash-Based Message Authentication Code)-based One-Time Password*) [36] puede ser válido durante un tiempo indeterminado. Por ello seleccionamos la versión temporal.

Los algoritmos *hash* disponibles son SHA1, SHA256 o SHA 512 (*Secure Hash Algorithm*). La versión más segura sería la de 512, seguida de 256. El número de dígitos indica el número de caracteres necesarios para validar el OTP. Aunque 8 dota de mayor robustez, se opta por emplear 6 puesto que es más cómodo y estamos en un entorno de evaluación. Por su parte, el periodo es el tiempo que el OTP es válido, es decir, que, tras alcanzar ese tiempo, se genera uno nuevo. Este parámetro es exclusivo de TOTP. También existe un parámetro exclusivo para HOTP, el contador inicial. Se trata del valor que toma como referencia para el inicio el algoritmo.

Por último, *Look Around Window* es un valor para extender la validez del OTP, lo que permite abrir el abanico de posibles OTP válidos, suprimiendo potenciales errores vinculados a fallos de sincronismo entre el reloj de usuarios y servidor. Así, fijarlo a valor 1 significa que solo se acepta un único código OTP, el actual; a valor 2, significa que se acepta el OTP actual y el del siguiente intervalo; y así sucesivamente con el resto de valores e intervalos.

En la Figura 4.6 podemos observar la configuración seleccionada para nuestro *realm*.

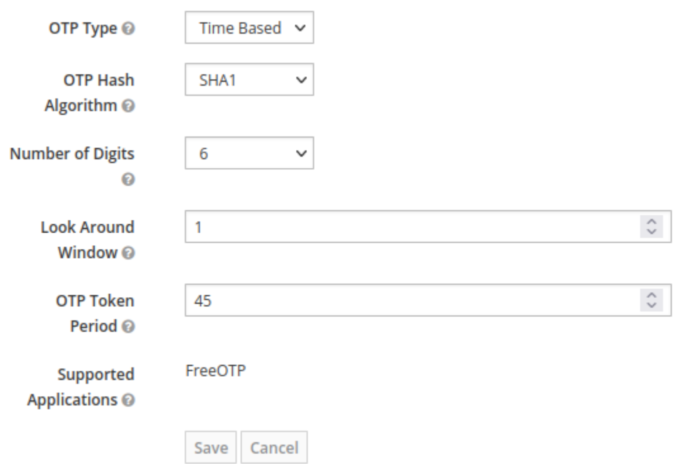


Figura 4.6 OTP Policy

Una vez tenemos habilitado y configurado OTP, cuando un usuario cree una cuenta, o recupere la contraseña, se le pedirá configurar OTP, como se muestra en la Figura 4.7.

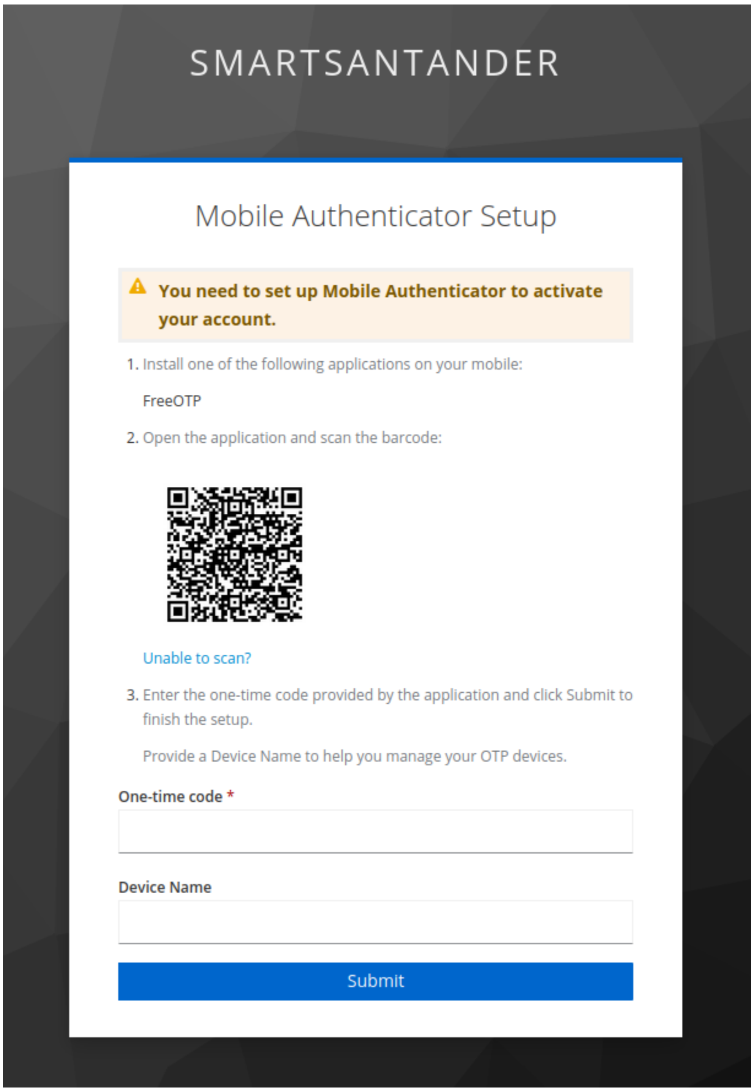


Figura 4.7 OTP Setup

4.3.4 Configuración de clientes

Para gestionar la autenticación y autorización tenemos 2 alternativas: la primera sería hacer un único cliente que englobe toda la configuración necesaria para el proxy y la segunda sería crear 2 clientes diferentes. Uno de ellos se encargaría de conectarse a la aplicación *proxy*, emitir *tokens* y demás, mientras que el otro se encargaría únicamente de gestionar la autorización. Esta segunda opción ha sido la elegida, ya que ofrece más ventajas tanto organizativas como de seguridad. Así lo afianza la documentación [37] .

En los siguientes apartados, vamos a crear y configurar 2 clientes, uno para conexión con el *proxy* y otro para control de acceso.

4.3.4.1 Cliente Proxy

Para agregar el servicio, nos tenemos que ir a `Clientes` y crear uno nuevo. Como ID, para el caso que nos conlleva, le asignamos “`proxy`” y se fija el protocolo cliente a OIDC, como se había propuesto. Este cliente va a ser el que se conecte con el *proxy*. En la Figura 4.8 observamos cómo se crea.

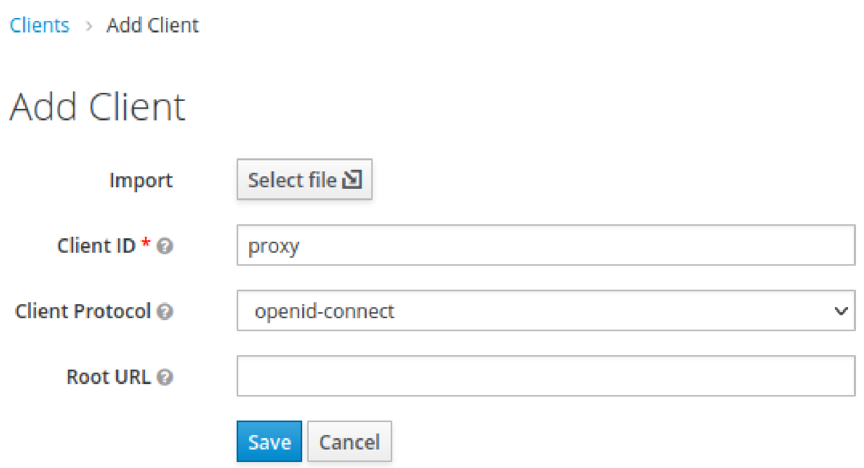


Figura 4.8 Creación de cliente proxy

Tras esto, observamos un gran número de opciones ya solo en la pestaña de `Settings`. Solo vamos a explicar algunas pestañas y los elementos más relevantes de estas.

Empezamos por la de *Settings*. Aquí encontramos el ID del cliente, el cual era obligatorio en la pestaña de creación de cliente. Este es el que identifica al cliente dentro del *realm*.

Nombre y descripción son campos no obligatorios con la finalidad de describir y nombrar al cliente. El botón de `Enabled` permite activar o desactivar al cliente. Esto es útil por si queremos deshabilitar un cierto servicio o aplicación sin afectar al resto.

El `Access Type` puede ser de 3 tipos, `confidential`, `public` o `bearer-only`. `Confidencial` indica que para interactuar el cliente con Keycloak es necesario un *Client Secret*, es decir, un conjunto de caracteres que solo el cliente sabe. Para obtenerlo o regenerarlo, nos debemos de ir a la pestaña de `Credentials`. En cambio, si seleccionamos `public`, este *Secret* no es necesario. La última opción sería `bearer-only`. Esto indica que el cliente no puede participar en los inicios de sesión, es decir, no permite la emisión de *tokens* a los usuarios. Se utiliza de manera exclusiva para asegurar el *back-end*, donde este solo podría ser llamado por otro microservicio. Se emplea en casos muy particulares. Para nuestro *proxy* vamos a usar `confidential`.

Los siguientes botones nos indican los tipos de flujo que queremos. En la Tabla 4.1 podemos ver la correspondencia entre la terminología empleada en Keycloak y la incluida en el estándar.

Tabla 4.1 Flujos en Keycloak

Keycloak	*Flujo OIDC*
Standard flow	Authorization Code
Implicit flow	Implicit
Direct Access Grants	Resource Owner Password
Service Accounts	Client Credentials
OAuth 2.0 Device Authorization Grant	Device Authorization

Como habíamos decidido, vamos a habilitar *Standard flow* (*Authorization Code flow*) y *OAuth 2.0 Device Authorization Grant* (*Device Authorization flow*).

El último parámetro que veremos es `Valid Redirect URIs`. Se trata de un elemento indispensable al iniciar sesión, ya que una vez la aplicación redirige al usuario al entorno para iniciar sesión y este se valida correctamente, posteriormente debe ser posible continuar con el flujo de autorización. Así pues, Keycloak emplea este parámetro para redirigir al usuario tras *login* o *logout*. El valor debe de coincidir con la URL proporcionada durante el proceso, pues en caso opuesto no podremos autenticarnos. En nuestro caso, vamos a poner donde se encuentra el servidor *proxy*.

La configuración descrita es la más básica y sencilla para identificar y autorizar a los usuarios y proporcionar la credencial de acceso. Podría ampliarse el conjunto de parámetros de configuración para definir el tiempo de vida de los *tokens* de acceso, el uso de *Refresh Token*, los algoritmos de firma de los *tokens*, etc.

4.3.4.2 Cliente de autorización

Ahora nos vamos a centrar en cuanto al control de acceso. En el apartado anterior, hemos configurado el servicio y únicamente se ha implementado la identificación. Ya vimos que la documentación nos invitaba a separar el *front end* del *back end* [37].

Por lo tanto, lo que vamos a hacer es añadir un nuevo cliente, al cual identificaremos como "*auth_client*". El protocolo empleado será una vez más OIDC.

Tras esto, se procede a realizar la configuración de forma similar al caso anterior haciendo uso del menú `Settings`.

En este caso el `Access Type` se fijará nuevamente a `confidential`, ya que `public` no tiene ese extra de seguridad que da el *secret* y `bearer-only` no es indicado para gestión de acceso, sino para servicios llamados por otros servicios. Podríamos pensar en ponerlo en modo `bearer-only`, ya que es destinado al *back end*. Sin embargo, si queremos gestionar recursos, *scopes*, políticas y permisos no es el indicado.

Para habilitar la pestaña de autorización, debemos de activar `Authorization Enabled`. A modo de nota, esto activa `Service Accounts Enabled` automáticamente, ya que se requiere para su funcionamiento.

Antes de centrarnos en la pestaña de autorización, vamos a crear los roles del cliente. Una vez en el cliente, seleccionamos la pestaña de `Roles` y añadimos los roles de `admin` y `user`. También debemos de añadir el rol `user` a los roles por defecto de los usuarios. Así cuando se creen nuevas cuentas, estos los tendrán añadidos. Así lo observamos en la Figura 4.9.

Ya en la pestaña de autorización, se va a manejar de manera fina los parámetros y políticas de la autenticación. Las pestañas más relevantes son las de `Resources`, `Authorization Scopes`, `Policies` y `Permissions`. En `Scopes` se configurará una única opción, *read*, ya que el acceso a los recursos solo va a ser de tipo lectura. Ese *scope* después se ligará exactamente al recurso en cuestión a la hora de su creación.

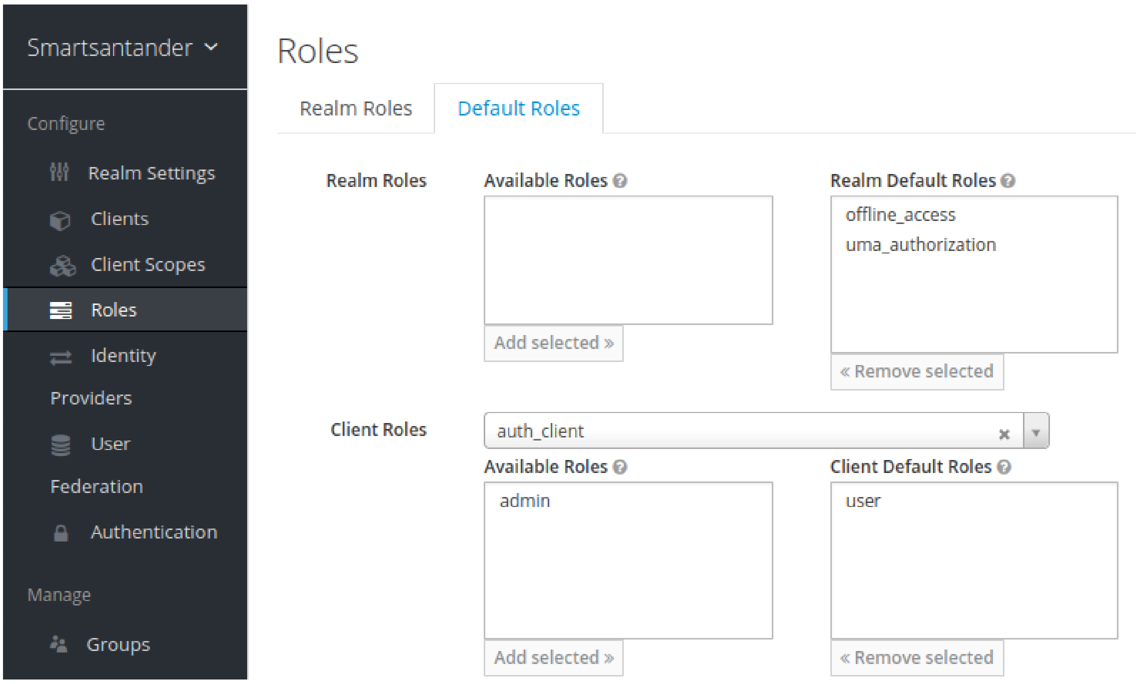


Figura 4.9 Client Default Roles

Las URIs a proteger se configuran en `Resources`. Es ahí donde ligamos los recursos y acciones que se pueden realizar sobre el mismo (*scopes*). Para este proyecto, vamos a añadir temperatura y batería, ya que son los recursos que debemos de proteger en Smart Santander. También les agregamos el *scope* de *read*. En la Figura 4.10 vemos como queda el relativo a la batería.

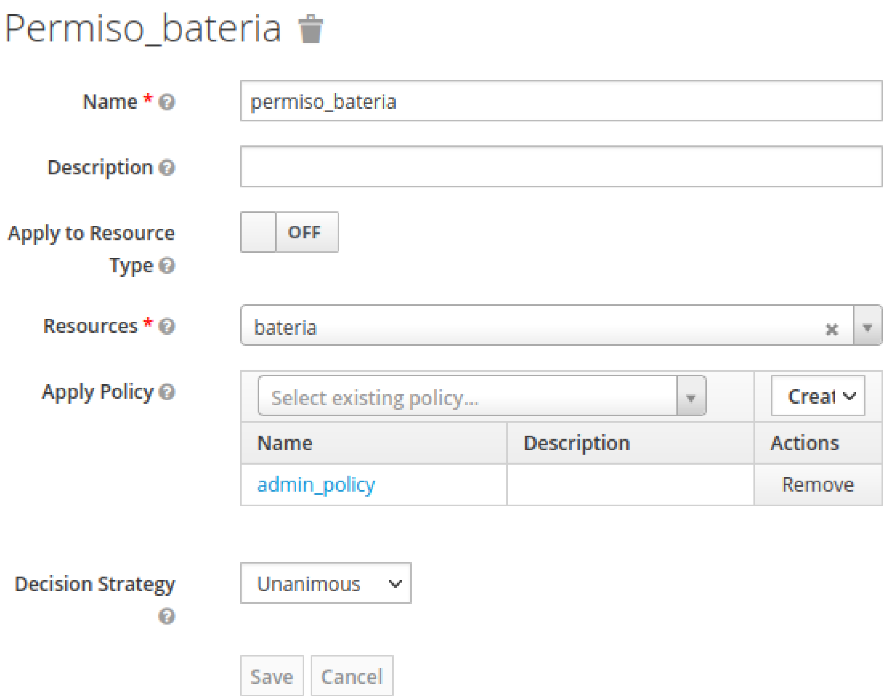


Figura 4.10 Permiso batería

En el apartado de `Policies` es donde añadimos las políticas, es decir, la relación de reglas que establece si permitimos acceder a grupos de usuarios, a un usuario en concreto, a usuarios con un determinado rol, a cualquiera en un determinado tiempo, etc. a unos determinados recursos. Existen diferentes tipos de políticas e incluso políticas de políticas. Estas están explicadas en profundidad en la documentación [38]. Las políticas nos brindan un sinfín de posibilidades.

En el ámbito del proyecto, se empleará una sencilla configuración de políticas cuyas restricciones se harán en función del rol del usuario. Primero, configuraremos la relacionada con el usuario. La llamamos `user_policy` y agregamos el rol de cliente de `user`. La lógica la dejamos en positivo, para que se evalúe únicamente cuando el rol sea de usuario. En caso contrario, se habilitaría para todo usuario sin el rol `user`. Repetimos una configuración similar para el caso del rol de `admin`.

En este punto tenemos definidos los recursos con los *scopes* correspondientes y las políticas de roles de forma independiente. Para vincularlos, debemos de establecer un permiso de recurso. Estos se establecen en la pestaña `Permissions`. Para facilitar la evaluación crearemos una para cada recurso definido anteriormente, “`permiso batería`” y “`permiso temperatura`”, los cuales gestionarán los recursos de batería y temperatura respectivamente. En cuanto a las políticas, el permiso de temperatura tendrá tanto la relativa al `user` como la de `admin`, ya que ambos roles deben de ser capaces de acceder a la temperatura de los sensores. Sin embargo, en el permiso de la batería, puesto que solo los administradores pueden acceder, solo agregamos la política de `admin`.

En la Tabla 4.2 recogemos un resumen de toda configuración de autorización de nuestro servicio. La Tabla 4.3 muestra observamos la evaluación de la autorización en función del rol y recurso al que se accede.

Tabla 4.2 Resumen de control de acceso

Recurso	*Scope*	*Política*	*Permiso*
Temperatura	Read	admin_policy	permiso temperatura
Batería		user_policy	permiso batería

Tabla 4.3 Casuístico de acceso

Usuario	*Recurso*	*Respuesta*
Usuario	Temperatura	Accede
Usuario	Batería	No accede
Administrador	Temperatura	Accede
Administrador	Batería	Accede

4.3.5 Configuración del Proxy

Una vez vista la configuración de Keycloak, vamos a detenernos brevemente en los 3 componentes que conforman el *proxy*, tal como vimos en su arquitectura mostrada en la Figura 3.1. Primero expondremos toda la configuración necesaria del adaptador, después la elaboración del servidor y por último la relativa al cliente HTTP.

4.3.5.1 Adaptador Node.js

Las librerías para emplear el adaptador de Keycloak para Node.js están disponibles en GitHub [39]. Se compone de scripts que automatizan y facilitan el acceso a numerosas funcionalidades como la gestión de los flujos de autenticación y autorización, sesiones de usuarios, funciones de protección, *middlewares*, etc. Todo esto no viene integrado en la librería de Keycloak y es la razón por la que lo usaremos de base.

En cuanto a la conexión con Keycloak, el adaptador requiere de una serie de parámetros, los cuales o los añadimos manualmente al código o utilizamos un archivo de configuración. Dicho archivo lo autogenera Keycloak. Para ello, debemos de acceder a `Clientes > proxy` y seleccionar la pestaña de `Installation`, obteniéndolo del desplegable *Keycloak OIDC JSON*.

Gracias a esta herramienta de Keycloak, la configuración del adaptador del *proxy* es trivial. Este archivo debe de estar en la raíz del proyecto de Node.js para que el adaptador lo lea correctamente y bajo ningún concepto debe de ser expuesto al exterior, ya que aparece información como el *realm* al que se une, la dirección del server, el *secret*, etc. En la Figura 4.11 observamos todos de campos y variables.

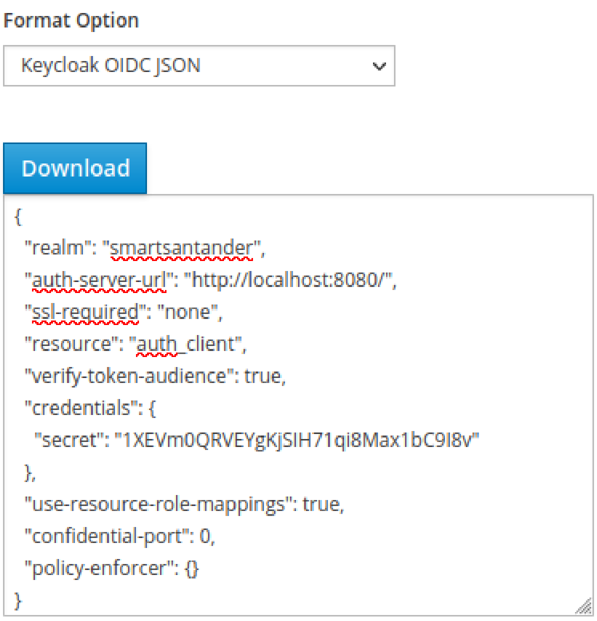


Figura 4.11 Archivo de configuración JSON

Dos funciones a resaltar del adaptador Node.js van a ser `protect()` y `enforcer()`. Estas se usan para proteger recursos. La primera de ellas básicamente es utilizada en ámbito de la validación de permisos en función roles (tanto de clientes como de *realm*). Si se usa sin parámetros, solo realiza autenticación, no autorización. Dicha validación se realiza en el adaptador del *proxy* y no en Keycloak. En la Tabla 4.4 observamos las diferentes opciones de `protect()`.

Otra característica es el redireccionamiento. En ocasiones, el usuario accede a un recurso sin aportar *token* o no tiene sesión previa, y `protect()` es capaz de redireccionarlo al login de la aplicación en Keycloak. De esta manera, se inicia el flujo de Authorization Code y mejora la experiencia de usuario.

Tabla 4.4 Definiciones de protect()

Definición función	*Utilidad*
Solo identificación	keycloak.protect()
Autorización vía rol del propio cliente	keycloak.protect('**rol**')
Autorización vía rol de otro cliente	keycloak.protect('**id_cliente**:**rol**')
Autorización vía rol de realm	keycloak.protect('**realm**:**rol** ')

La segunda función está destinada a control de acceso. No incorpora el reenvío al *login* de Keycloak como `protect()`, pero permite un mayor grado de control de acceso, ya que es la que utiliza los recursos, *scopes*, políticas y permisos de Keycloak. Para ello, realiza peticiones de permisos a Keycloak, y es este el que evalúa si concede o no el permiso de acceso al recurso en cuestión. En la Tabla 4.5 observamos las diferentes opciones de `enforcer()`.

Tabla 4.5 Definiciones de enforcer()

Definición función	*Utilidad*
Autorización avanzada con permisos	keycloak.enforcer('**recurso**:**scope**', {resource_server_id: '**id_cliente**'})

Más información relativa a dichas funciones se puede encontrar en la documentación de Keycloak [40].

4.3.5.2 HTTP Server

Para implementar la parte de servidor del *proxy*, usaremos Express.js. Con este *framework*, vamos a configurar tanto el puerto del *proxy* como las rutas que habilitarán el acceso a los diferentes recursos que se exporten. A modo de ejemplo, en la Figura 4.12 observamos un direccionamiento de la ruta `/admin`.

```
app.use('/admin',keycloak.protect('auth_client:admin'), (req, res) => {
        makeRequest(res,"get",ip_server,"4000",req.baseUrl)
});
```

Figura 4.12 Ejemplo de direccionamiento

En concreto, para realizar la prueba de laboratorio y ver que todo funciona perfectamente, vamos a crear las rutas que se recogen en la Tabla 4.6 y sobre las que aplicaremos diferentes métodos de protección.

Para los cuatro primeros recursos usaremos `protect()`, evaluando únicamente la identificación y comprobando los roles de `user` y `admin` en el propio adaptador. Por otra parte, para temperatura y batería usaremos `enforcer()` y `protect()`, donde además de la identificación también se evaluarán los permisos y políticas de acceso a los mismos.

Tabla 4.6 Rutas y mecanismo de protección

URI	*Protección utilizada*
/anonymous	keycloak.protect()
/user	keycloak.protect('auth_client:user')
/admin	keycloak.protect('auth_client:admin')
/all-user	keycloak.protect(['auth_client:admin', 'auth_client:user'])
/temperatura	keycloak.protect() keycloak.enforcer('temperatura:read', {resource_server_id: 'auth_client'})
/bateria	keycloak.protect() keycloak.enforcer('temperatura:read', {resource_server_id: 'auth_client'})

4.3.5.3 Cliente HTTPs

La última parte del *proxy* está compuesta por el cliente HTTP hacia el servicio a proteger. Se implementa mediante Axios, un librería HTTP basado en *promise* para Node.js y el navegador, que se usará para realizar las peticiones al servicio. En el banco de pruebas no nos conectaremos a Smart Santander, sino que lo simularemos con un servidor controlado que replique los recursos necesarios anteriormente reseñados. Para realizar las peticiones y mostrar el contenido al usuario, vamos a utilizar la función de la Figura 4.13.

```
async function makeRequest(res,method,host,port,path) {
    try {
        const config = {
                method: `${method}`,
                url: `http://${host}:${port}${path}`,
        }
    let response = await axios(config)
    res.send(response.data)

    } catch (error) {
        console.log(error);
        res.send(error)
    }
}
```

Figura 4.13 Función cliente HTTP

4.4 Comprobación de funcionamiento.

A continuación, se procede a evaluar el comportamiento del entorno configurado, que ayudará a comprender en un entorno real los flujos OAuth2 empleados, así como el conjunto de datos intercambiados para gestionar el control de acceso. Para ello, además de un conjunto de pruebas funcionales, vamos a analizar el tráfico intercambiado mediante la herramienta Wireshark.

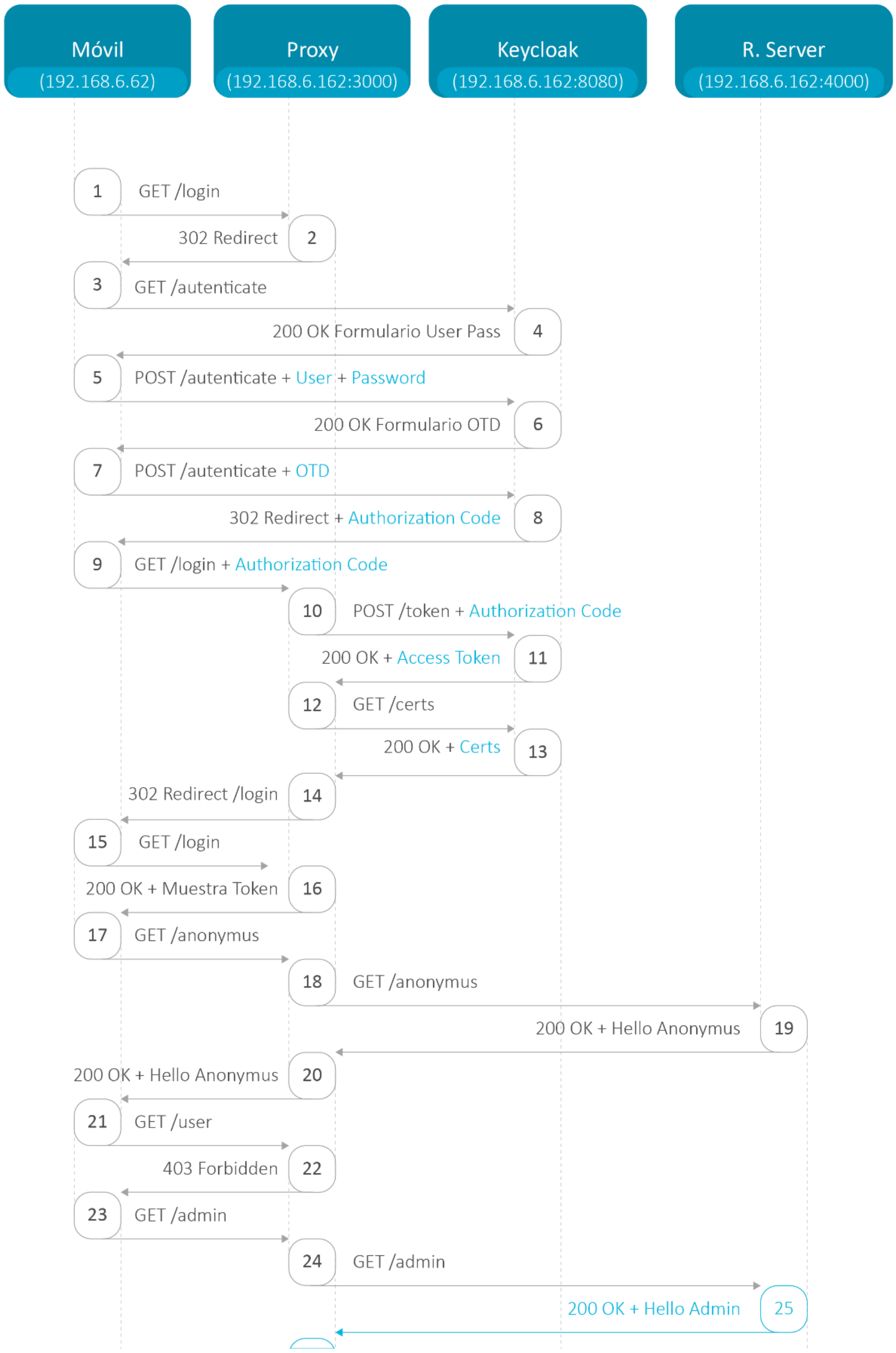


Figura 4.14 Parte 1 de captura realizada para Authorization Grant

Más concretamente, vamos a analizar 2 casos de uso. En el primero se empleará un navegador web como cliente de acceso al sistema. El segundo buscará emular el comportamiento de un dispositivo embebido sin interfaz de usuario, por lo que se hará uso de la herramienta Postman para acceder directamente a las API proporcionadas por el *proxy* y Keycloak.

4.4.1 Comprobación Authorization Code Grant

En el primer caso se emplea el flujo *Authorization Code Grant Type*, detallado en la sección 2.3.4.3. Este será comparado con el intercambio capturado, el cual se muestra en la Figura 4.14.

El análisis del flujo de datos intercambiado en un entorno real gestionado por Keycloak se muestra a continuación.

Tras acceder al recurso *login* de la aplicación, al no estar el usuario autenticado, esta redirige al usuario al entorno de confianza de Keycloak para que introduzca y valide sus credenciales. Esta redirección se remite a la URL de Keycloak (puerto 8080) con los parámetros de cliente *proxy* e incluye la URI de redirección que se usará posteriormente para retornar el *token* de sesión a la aplicación. Esto se corresponde con el intercambio de tramas 1, 2, 3 y 4 de la captura y coincide con las 1, 2, 3 y 4 de la Figura 2.7 *Authorization Code flow*.

Como ya se ha insistido anteriormente, esta debe de coincidir con la añadida en configuración. Se ha probado a cambiar la URL de la configuración y como habíamos dicho anteriormente, da error en *redirect_uri* (Figura 4.15), ya que esta se comprueba.

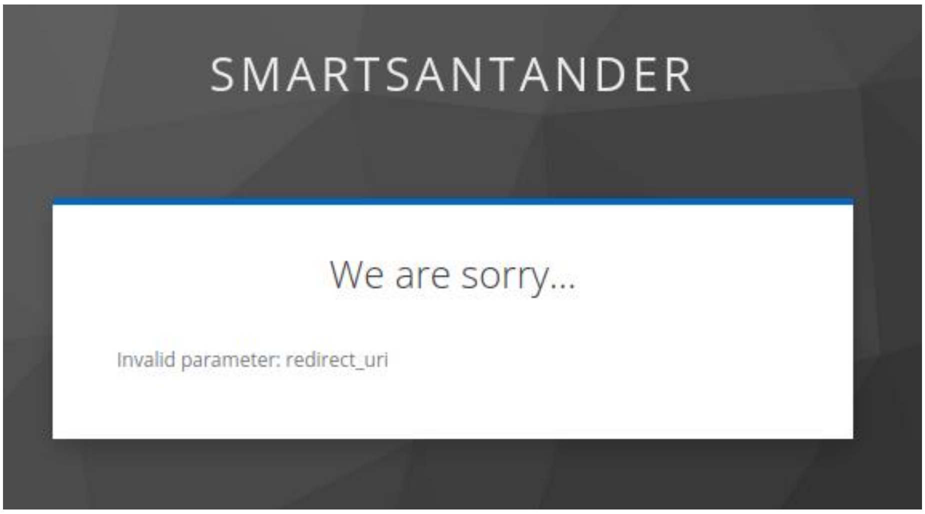


Figura 4.15 Error de redirect_uri

Puesto que Keycloak se ha configurado para operar sin intercambios seguros (HTTP), el análisis permite observar en la trama 5, donde el usuario envía su nombre y contraseña de forma directa y sin pasar por el *proxy*, como se muestra en la Figura 4.16.

```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "username" = "user_admin"
  Form item: "password" = "user_admin"
  Form item: "credentialId" = ""
```

Figura 4.16 Credenciales del usuario

El envío de OTP también lo encontramos (trama 7), como se muestra en la Figura 4.17.

```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "otp" = "667885"
  Form item: "login" = "Sign In"
```

Figura 4.17 OTP del usuario

Tras la identificación correcta, se crean *cookies* de sesión y se redirecciona de nuevo al proxy (puerto 3000) con el código (*Authorization Code*) en la URL como parámetro. En la Figura 4.18 encontramos las diferentes *cookies* que se envían en el paquete 8 de la captura. Nuevamente, es coherente con el flujo OAuth 2.0 (paquete 5).

```
Set-Cookie: KEYCLOAK_LOCALE=; Version=1; Comment
Set-Cookie: KC_RESTART=; Version=1; Expires=Thu,
 [truncated]Set-Cookie: KEYCLOAK_IDENTITY=eyJhbG
 [truncated]Set-Cookie: KEYCLOAK_IDENTITY_LEGACY
Set-Cookie: KEYCLOAK_SESSION=nodejs-example/b722
Set-Cookie: KEYCLOAK_SESSION_LEGACY=nodejs-examp
Set-Cookie: KEYCLOAK_REMEMBER_ME=; Version=1; Co
```

Figura 4.18 Creación de cookies para SSO

El *proxy* envía a Keycloak el *Authorization Code* proporcionado por el usuario con más parámetros necesarios. En la Figura 4.19 se recogen dichos parámetros. Y ante ello, Keycloak responde con los diferentes *tokens*, entre ellos el *Access Token*, mostrado en la Figura 4.20. Como no puede ser de otra forma, coincide la captura (paquetes 10 y 11) con el flujo teórico (6 y 8).

```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "client_session_state" = "QKb08aCU78ABgvOr5xG7Nug2U-XC14q_"
  Form item: "client_session_host" = ""
  Form item: "code" = "4d5d4b4d-a8f0-4cd6-82e7-3c7abe6fdd30.985979c9-2c5c-4f5f-b7b9
  Form item: "grant_type" = "authorization_code"
  Form item: "client_id" = "proxy"
  Form item: "redirect_uri" = "http://192.168.106.162:3000/login?auth_callback=1"
```

Figura 4.19 POST del proxy a Keycloak

El *token* de la captura, sería el de la Figura 4.21. Solo nos vamos a fijar en el *payload*, ya que este es la parte más relevante.

```
v JavaScript Object Notation: application/json
  v Object
    > Member Key: access_token
    > Member Key: expires_in
    > Member Key: refresh_expires_in
    > Member Key: refresh_token
    > Member Key: token_type
    > Member Key: id_token
    > Member Key: not-before-policy
    > Member Key: session_state
    > Member Key: scope
```

Figura 4.20 Envió de tokens

Las primeras claves `exp` e `iat` son enteros que indican los tiempos de expiración y de emisión del *token*, en *Unix time*. Si nos fijamos, existe una diferencia de 300 segundos, que coincide con un tiempo de vida del *token* de 5 minutos. También encontramos un identificador único del *token* además del emisor como tal "smartsantander". Más adelante aparece un campo que es de gran importancia, las audiencias [41]. Este indica que servicios pueden recoger el *token*. *Proxy* es uno de ellos, ya que es el servicio que recibe el *token* (*Authorized Party*) y aparece también *auth_client*, debido a que el usuario contiene un rol de este cliente.


```
{
  "exp": 1663511863,
  "iat": 1663511563,
  "auth_time": 1663511563,
  "jti": "5ed96e5e-df5a-47f0-84e2-81e03896530e",
  "iss": "http://192.168.106.162:8080/realms
/smartsantander",
  "aud": [
    "proxy",
    "auth_client"
  ],
  "sub": "b722ca08-6529-487d-a0c5-0bbdc08680d9",
  "typ": "Bearer",
  "azp": "proxy",
  "session_state": "985979c9-2c5c-4f5f-b7b9-
a2514098008d",
  "acr": "1",
  "allowed-origins": [
    ""
  ],
  "resource_access": {
    "auth_client": {
      "roles": [
        "admin"
      ]
    }
  },
  "scope": "openid profile email",
  "sid": "985979c9-2c5c-4f5f-b7b9-a2514098008d",
  "email_verified": true,
  "name": "a a",
  "preferred_username": "user_admin",
  "given_name": "a",
  "family_name": "a",
  "email": "test@gmail.com"
}
```


Figura 4.21 Payload del token de la captura

Luego aparecen más datos relativos a pertenencia del *token*, el estado de la sesión, la parte a la que se emite el *token* (*Authorized* Party), etc.

Un campo importante es el de "`resource_access`". Ahí podemos encontrar los roles del usuario relativos a cada cliente. En este caso, el usuario tiene el rol de `admin` en `auth_client`. Esto concuerda con el acceso a recursos protegidos únicamente con el rol de `admin`.

Este *token* acaba con los *scopes* prefijados por OIDC e información relativa al usuario, correo electrónico, si el correo ha sido verificado, nombre y apellidos, nombre de usuario…

Volviendo al intercambio de la captura, en la Figura 4.22 se observa la petición 12 relativa a certificados. Esto nos indica que el *proxy* pide los certificados a Keycloak y comprueba la firma del *token*, mecanismo indispensable ya que se asegura que dicho *token* ha sido emitido por Keycloak y no ha sido falsificado por un tercero.

```
v Hypertext Transfer Protocol
  > GET /realms/nodejs-example/protocol/openid-connect/certs HTTP/1.1\r\n
    Host: 192.168.106.162:8080\r\n
```

Figura 4.22 GET de los certificados

Luego se observa las peticiones a los recursos desde la 17 hasta la 42. La Figura 4.23 contiene desde la 25 en adelante. Como vimos anteriormente, iniciamos sesión con el usuario de administración, por lo que tiene el rol de `admin` y los recursos estás protegidos de acuerdo a la Tabla 4.6. Al final, los administradores pueden acceder a todo, a excepción de `/user`, ya que no tiene el rol de usuario. También se observa que, para los 4 primeros recursos, no se realiza peticiones extra a Keycloak. Es decir, efectivamente la función `protect()` evalúa los *tokens* en el *proxy*, y no Keycloak. Sin embargo, para temperatura y batería, si se realizan, luego `enforcer()` pide la evaluación a Keycloak.

El último intercambio que se observa es el cierre de sesión. Este comprende el resto de paquetes (43 en adelante). Básicamente lo que se realiza es un redireccionamiento de `/logout` del *proxy* a `/logout` de Keycloak, ya que este y solo este debe de cerrar la sesión al tratarse de un sistema SSO. La sesión se cierra borrando las *cookies* de sesión y reenviando al usuario de vuelta a la aplicación (paquete 47).

4.4.2 Comprobación Device Authorization Grant

Para la segunda prueba, vamos a testear los *endpoints* a través de Postman. La prueba que vamos a realizar es obtener un *token* a través de Postman e iniciar sesión con la cuenta de `user`. Accederemos a `/temperatura` y `/batería`, donde el segundo debe de devolvernos "*acceso denegado*". Es básicamente seguir *OAuth 2.0 Device Authorization Grant*.

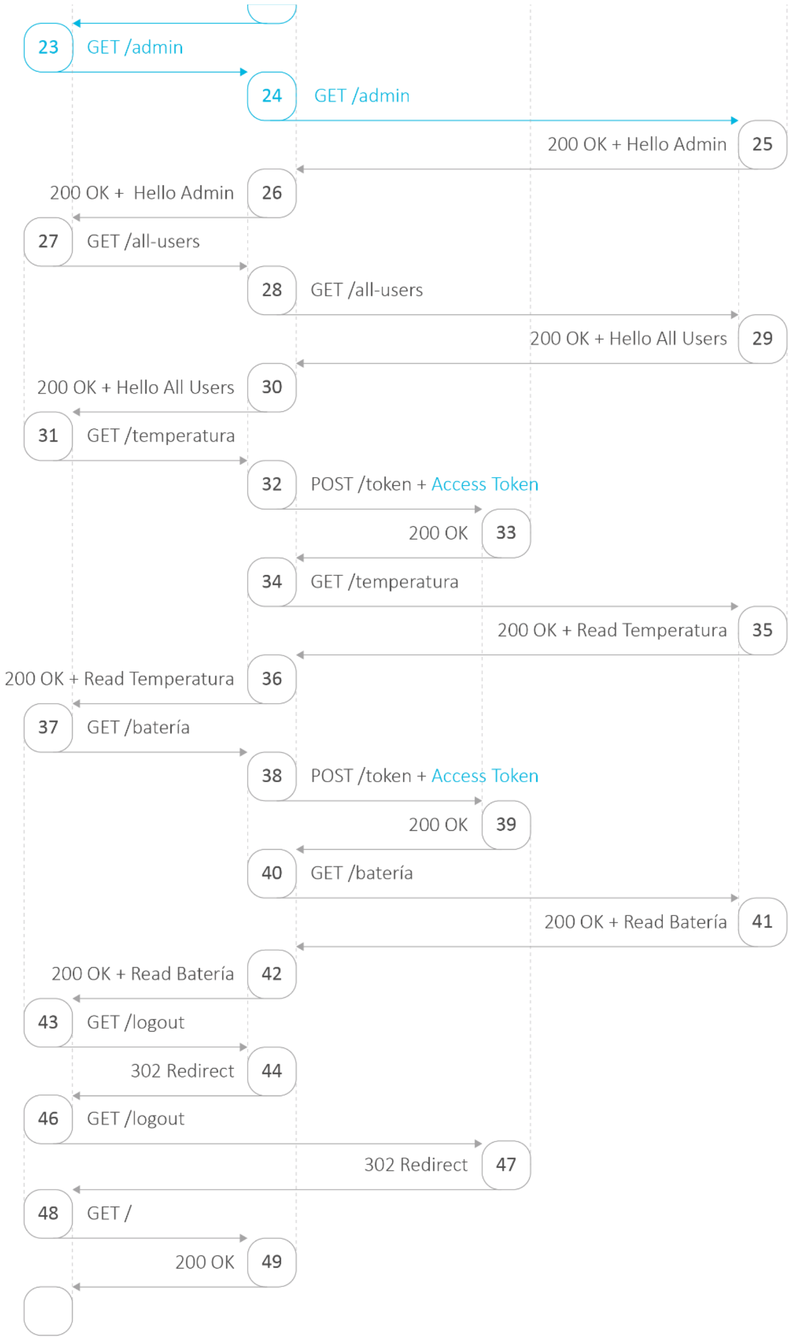


Figura 4.23 Parte 2 de captura realizada para Authorization Grant

Lo primero que realizamos es la obtención del *device_code*, *user_code* y *la verification_uri*. Para ello debemos de realizar un POST al *endpoint* del dispositivo con el *Client ID* y *Secret* en el *body*, como se muestra en la Figura 4.24.

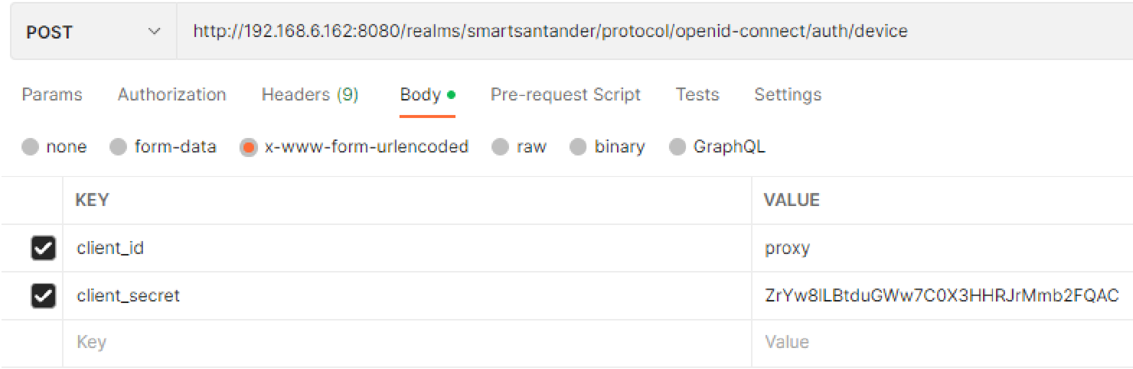


Figura 4.24 Petición al Device Authorization Endpoint

La respuesta que obtenemos en la Figura 4.25 contiene *device_code*, *user_code*, *verification_uri*, *verification_uri_complete*, *expires_in* e *interval*.

```
{
    "device_code": "ivj9KJIdz5jpSx32BBOH5by4v3MyndVoPQ8_raJux5U",
    "user_code": "YUVY-MGBO",
    "verification_uri": "http://192.168.6.162:8080/realms/smartsantander/device",
    "verification_uri_complete": "http://192.168.6.162:8080/realms/smartsantander/device?user_code=YUVY-MGBO",
    "expires_in": 600,
    "interval": 5
}
```

Figura 4.25 Respuesta a Device Authorization Endpoint

Estos dos pasos, se corresponden con los 2 y 3 respectivamente del *Device flow*.

Ahora el usuario debe de acceder a *verification_uri_complete* (menos pasos que con *verification_uri* y *user_code*), donde iniciamos sesión con un usuario no administrador.

Una vez que el usuario concede los permisos al dispositivo, este ya puede obtener el *token* en el *Token Endpoint*. Para ello, debemos de realizar un POST a dicha dirección y conteniendo en el *body* el *Client ID* y *Secret*, igual que anteriormente; el *device_code* del *device authorization endpoint* anterior y el *grant_type* a "*urn:ietf:params:oauth:grant-type:device_code*"; como nos indica el estándar [42]. La petición realizada en el laboratorio es la de la Figura 4.26.

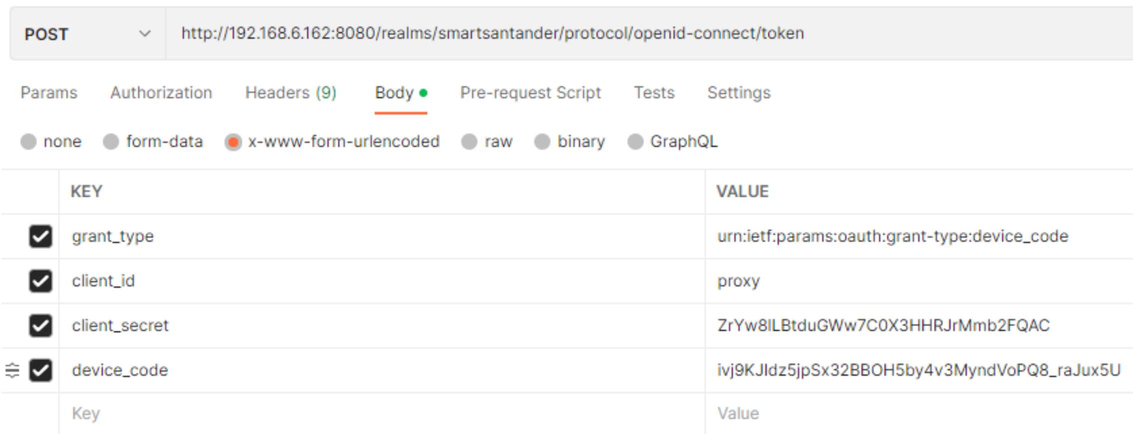


Figura 4.26 Petición al Token Endpoint

Si realizamos esta petición antes que el usuario autorice, responderá con el error "*The authorization request is still pending*". Sin embargo, si el usuario completa el *Browser flow*, se obtendrán los tokens de manera idéntica al *Authorization Code flow*, como se muestra en la Figura 4.27.

Body   Cookies (3)   Headers (9)   Test Results

Pretty   Raw   Preview   Visualize   JSON


```json
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI1c0w4
        eyJleHAiOjE2NjQyMTY1MzIsImlhdCI6MTY2NDIxNjIzMiwiYXV0aF90aW1lIjoxNjY0M
        xtcy9zbWFydHNhbnRhbmRlciIsImF1ZCI6WyJwcm94eSIsImF1dGhfY2xpZW50IiwiYWN
        b25fc3RhdGUiOiI0MWNmN2ZiZS02NDVmLTRkYjAtOGZmNy04ZDE3NTg0ZTZhMTciLCJhY
        lvbiIsImRlZmF1bHQtcm9sZXMtbm9kZWpzLWV4YW1wbGUiXX0sInJlc291cmNlX2FjY2V
        bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGUiXX19LCJzY29wZSI6InByb2ZpbGUgZW1haWwiL
        VmZXJyZWRfdXNlcm5hbWUiOiJ1c2VyX3VzZXIiLCJnaXZlbl9uYW1lIjoiZmRnIiwiZmF
        NhUXbjovN5TrBRIhrVQvmhfgerfKu4yMyPkGWI_iVGwrLL_Aij04ZEzly8kIJN3mHySsA
        3wHRKg9OWyY2KNIv6uLnkLBssnyrs1GDNFKUyIPktL3IFBzUSV0WuIvzHQ2Dt1mswazXy
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJkMTU
        eyJleHAiOjE2NjQyMTgwMzIsImlhdCI6MTY2NDIxNjIzMiwianRpIjoiYzFhMzg4MzktY
        QiOiJodHRwOi8vMTkyLjE2OC42LjE2Mjo4MDgwL3JlYWxtcy9zbWFydHNhbnRhbmRlciI
        dGUiOiI0MWNmN2ZiZS02NDVmLTRkYjAtOGZmNy04ZDE3NTg0ZTZhMTciLCJzY29wZSI6I
        tMMtxuS1b-JRSm3nR-1HMUNkUVWVyUU5s5smFHG8YqA",
    "token_type": "Bearer",
    "not-before-policy": 1663511779,
    "session_state": "41cf7fbe-645f-4db0-8ff7-8d17584e6a17",
    "scope": "profile email"
}
```


Figura 4.27 Respuesta con el token de acceso

Con respecto al *Device Authorization flow*, estos pasos han involucrado que el usuario realice el *Browser flow* y que se sondee y obtenga el *token*, lo cual corresponde con las peticiones 5 y 7 del flujo.

Antes de realizar las peticiones a los recursos, vamos a comprobar el contenido del *token*. En la Figura 4.28 observamos el cuerpo de este, al igual que hicimos en el primer caso.

Nuevamente se repite la estructura. `iat` nos indica cuando se emitió el *token* (21/7/2022 a las 20:07:52). El tiempo de expiración en este caso no aparece, pero sabemos por el parámetro `expires_in` presente en la entrega del *token* que es de 300 segundos. También encontramos el identificador único del *token* además del emisor como tal "smartsantander". Más adelante aparecen las audiencias, donde se encuentra *proxy*, ya que es el servicio que recibe el *token* (*Authorized Party*) y aparece también `auth_client`, debido a que el usuario contiene un rol de este cliente (`user`). Dicho rol lo podemos observar en "`resource_access`" ligado a `auth_client`.

```
{
  "exp": 1664216532,
  "iat": 1664216232,
  "auth_time": 1664216210,
  "jti": "9ef777a6-d6e9-409f-98b4-5f719cf4dc33",
  "iss": "http://192.168.6.162:8080/realms
/smartsantander",
  "aud": [
    "proxy",
    "auth_client",
    "account"
  ],
  "sub": "b99a9c70-66e2-4677-98d8-713f10beb21e",
  "typ": "Bearer",
  "azp": "proxy",
  "session_state": "41cf7fbe-
645f-4db0-8ff7-8d17584e6a17",
  "acr": "1",
  "allowed-origins": [
    ""
  ],
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization",
      "default-roles-nodejs-example"
    ]
  },
  "resource_access": {
    "auth_client": {
      "roles": [
        "user"
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "profile email",
  "sid": "41cf7fbe-645f-4db0-8ff7-8d17584e6a17",
  "email_verified": true,
  "name": "fdg dfg",
  "preferred_username": "user_user",
  "given_name": "fdg",
  "family_name": "dfg",
  "email": "rasdasdfsda@gamil.com"
}
```


Figura 4.28 Payload del token de la captura

Luego aparecen más datos relativos a pertenencia del *token*, el estado de la sesión, la parte a la que se emite el *token* (*Authorized Party*), etc.

Finalmente, observamos información relativa al usuario, correo electrónico (rasdasdfsda@gamil.com), correo verificado, nombre y apellidos, nombre de usuario (user_user) …

Ya para acabar, vamos a solicitar los recursos de temperatura y batería. Para ello lanzamos una petición a `/temperatura` y `/bateria` respectivamente con el *token* en la cabecera *Authorization* como se muestra en la Figura 4.29.

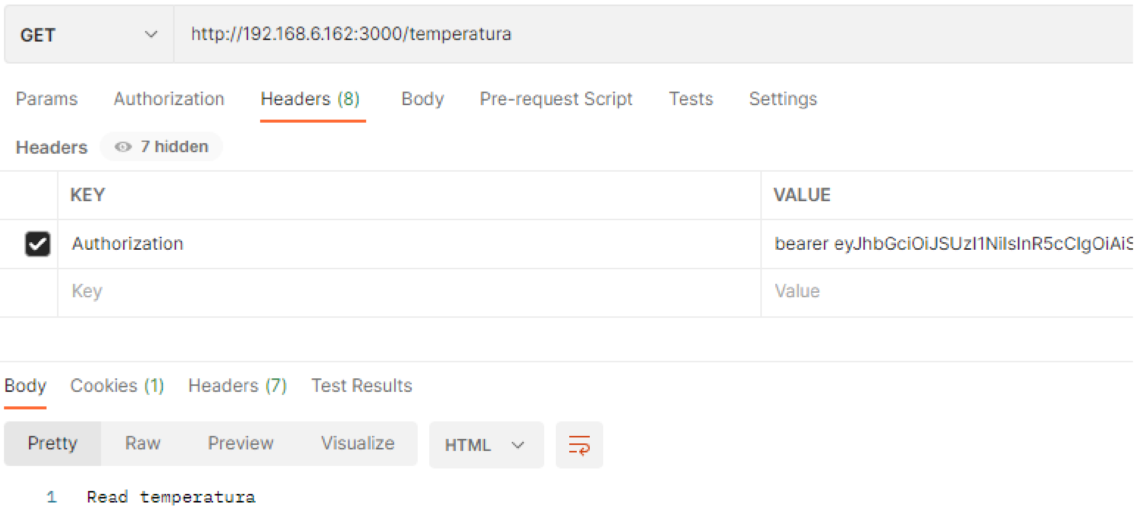


Figura 4.29 Petición y respuesta exitosa de temperatura

El resultado es exitoso para el caso de la temperatura, como era de esperar. Sin embargo, para batería no ocurre lo mismo. En la Figura 4.30 observamos la respuesta. Dichos intercambios corresponderían a las acciones 8 y 9 del *Device flow*.

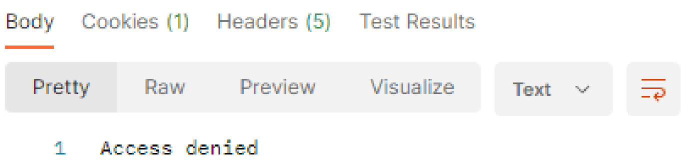


Figura 4.30 Respuesta denegada de batería

Con estas dos pruebas, hemos comprobado que el sistema opera correctamente (roles, políticas, evaluación de permisos tanto en Keycloak como en el conector del *proxy*, funciones de protección, OTP, etc.), además de testear los *endpoints* para permitir acceder a dispositivos a través del *Device Authorization Grant*. Por lo tanto, ya está listo para desplegarlo y proteger el acceso a la información de los sensores de Smart Santander.

5 Implementación

Sabiendo que el sistema se ha desarrollado con normalidad en los test de laboratorio, vamos a llevarlo a un entorno real. Para ello, vamos a realizar dos implementaciones, donde ambas se centrarán en el acceso seguro a los recursos exportados por Smart Santander. Primeramente, se describirá cómo se han protegido dichos recursos y realizas el acceso seguro a los mismos mediante OAuth2 a través de dispositivos tradicionales de usuario (ordenador, teléfono móvil, etc.). Posteriormente, se presenta la incorporación de dispositivos reales sin teclado o interfaz de interacción (p.ej. un reloj inteligente, estación meteorológica, etc.) que cuya operativa emplea *Device flow*.

5.1 Smart Santander

Partiendo de la arquitectura del capítulo anterior, se simplifica el trabajo. Realmente esta soporta proteger cualquier servicio sin necesidad de modificar su código. En este sentido, los trabajos a acometer se centran principalmente en adaptar las peticiones realizadas por la parte cliente HTTP a las de la API de Smart Santander. Por otra parte, hay que implementar en el servidor las interfaces necesarias para la recepción de los identificadores de los recursos de Smart Santander. Para el caso que nos conlleva nos centraremos en los termómetros.

Cabe destacar, que en el entorno de evaluación ya hemos preparado los permisos, recursos, cliente, etc. en la configuración de Keycloak con vista a dar soporte a Smart Santander, así ya no debemos de tocarlo. A modo de recordatorio, tenemos dos permisos. El primero está ligado al recurso `/temperatura` en modo lectura y con la política de rol `user`. En cambio, el segundo, está ligado a proteger `/bateria` en modo lectura y con la política de rol de `admin`.

En cuanto al *proxy*, se ha modificado para poder ser capaces de obtener el id del sensor en cuestión. En la Figura 5.1 podemos observarlo, y si nos fijamos bien, observamos que existen dos funciones dedicadas exclusivamente a realizar las peticiones a Smart Santander, `makeRequest_smartsantander_temperatura()` y `makeRequest_smartsantander_bateria()`. Podemos observar cómo son estas en detalle en la Figura 5.2.

```
app.use('/temperatura/:id',keycloak.protect(),
keycloak.enforcer('temperatura:read', {resource_server_id: 'auth_client'}),
(req, res) => {
        makeRequest_smartsantander_temperatura(res,"get",req.params.id)
});

app.use('/bateria/:id',keycloak.protect(), keycloak.enforcer('bateria:read',
{resource_server_id: 'auth_client'}), (req, res) => {
        makeRequest_smartsantander_bateria(res,"get",req.params.id)
});
```

Figura 5.1 Recepción de identificadores

```
async function makeRequest_smartsantander_temperatura(res,method,id) {
    try {
        const config = {
                method: `${method}`,
                url: `https://api.smartsantander.eu/v2/measurements/temperature:ambient/urn/urn:x-iot:smartsantander:u7jcfa:t${id}/last`,
        }
    let response = await axios(config)
    res.send(response.data)

    } catch (error) {
        console.log(error);
        res.send("PÁGINA NO EXISTENTE EN SMART SANTANDER")
    }
}

async function makeRequest_smartsantander_bateria(res,method,id) {
    try {
        const config = {
                method: `${method}`,
                url: `https://api.smartsantander.eu/v2/measurements/batteryLevel/urn/urn:x-iot:smartsantander:u7jcfa:t${id}/last`,
        }
    let response = await axios(config)
    res.send(response.data)

    } catch (error) {
        console.log(error);
        res.send("PÁGINA NO EXISTENTE EN SMART SANTANDER")
    }
}
```

Figura 5.2 Funciones relativas a peticiones a Smart Santander

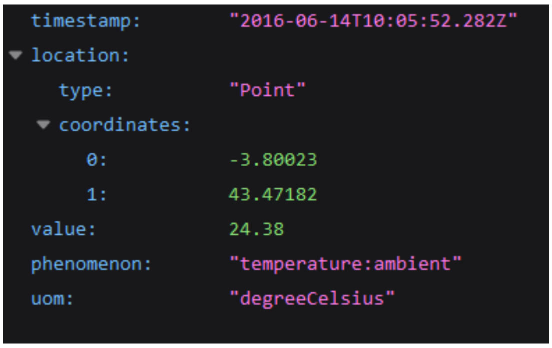
De cara a proteger otra aplicación o aplicaciones empresariales, vemos que solamente hay que adaptar la arquitectura a los requisitos de esta, es decir, modificar el *proxy* tanto en solicitud como en servicio al usuario y modificar Keycloak a nivel de clientes, roles, temas, etc.

Tras aplicar dichas transformaciones al *proxy*, se habilita el acceso seguro al API de Smart Santander. A modo de comprobación, vamos a realizar peticiones con diferentes roles a diferentes sensores. Por ejemplo, al acceder a `/temperatura/10000` nos devuelve el nivel de batería del sensor 10000. En la Figura 5.3 se puede observar el valor en concreto que se nos devuelve con el rol de `user`.

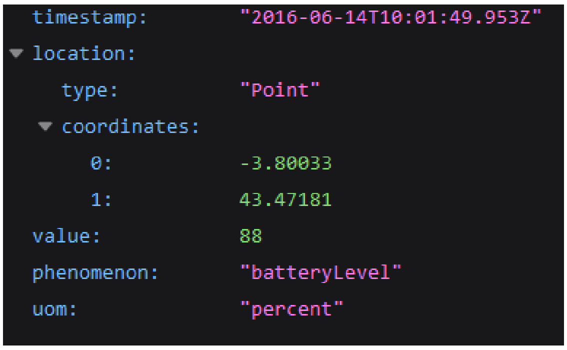

```
timestamp:        "2016-06-14T10:02:02.874Z"
▼ location:
    type:         "Point"
  ▼ coordinates:
      0:          -3.80014
      1:          43.47171
value:            23.22
phenomenon:       "temperature:ambient"
uom:              "degreeCelsius"
```


Figura 5.3 Temperatura del sensor 10000

Si accedemos con el rol de `admin`, pero esta vez al sensor 10001, debería de permitirnos acceder sin problema. La Figura 5.4.a así nos lo confirma. Solo nos queda comprobar con el nivel de batería. Los administradores, acceden sin problema. La respuesta a la solicitud de la batería del sensor 10002 es la de la Figura 5.4.b. Sin embargo, a los usuarios no se les permite acceder, ya que por diseño lo tienen prohibido. Se obtiene `Access denied`.



a) Temperatura del sensor 10001



b) Batería del sensor 10002

Figura 5.4 Información relativa al sensor 10001

5.2 Dispositivo embebido NodeMCU

Con anterioridad, hemos testeado que la solución permitiría emplear toda clase de dispositivos para el acceso autorizado a la información. Sin embargo, en la implementación se ha decidido ir un paso más allá, elaborando un dispositivo de usuario que emule un reloj inteligente, estación meteorológica o televisión que emita y reciba peticiones autorizadas para obtener información de los sensores.

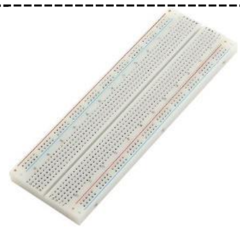
Independientemente de otros requerimientos *hardware*, se establece que el dispositivo deberá disponer de un interfaz de comunicaciones y un interfaz de visualización que permita mostrar el `user_code` requerido para autorizar al usuario y dispositivo.

Para este caso particular, se considera que el dispositivo esté alimentado externamente y su conexión a Internet sea mediante WiFi (*Wireless Fidelity*), para que disponga de plena movilidad. Se ha optado por incorporar una pantalla que muestre en un QR la `verification_uri_complete`, y así facilitar la interacción con el usuario durante el intercambio OAuth, pues tras escanear dicho código QR mediante un móvil, se le redirige automáticamente a la página de inicio de sesión. De esta forma, la experiencia de usuario se mejora con respecto a introducir manualmente códigos y URL. La pantalla también se usa para mostrar al usuario la información de los recursos, ya sea batería o temperatura.

5.2.1 Componentes

Para la implementación del dispositivo, se han empleado los elementos representados en la Tabla 5.1, en la que se incluye también su costo aproximado.

Tabla 5.1 Listado de materiales

Nombre	Imagen	Descripción	Precio	Referencia
NodeMCU V3 [43]		Plataforma IoT de código abierto de bajo coste.	9,99€	[44]
Protoboard		Dispositivo diseñado para permitirle crear circuitos sin necesidad de soldar.	4,79€	[45]
OLED Display I2C SSD1306		Pantalla monocromática OLED (*Organic Light-Emitting Diode*) de 128×64 puntos.	8,99€	[46]
Power Bank 2200 mAh		Batería recargable diseñada para suministrar energía o recargar dispositivos electrónicos.	13,33€	[47]
TOTAL			37,1€	

5.2.2 Montaje

Puesto que la interfaz de comunicaciones es a través de WiFi, las únicas conexiones requeridas son de la pantalla con la placa controladora. Al tratarse de un prototipo, se han optado por emplear un *protoboard*. En la Figura 5.5 y la Figura 5.6 observamos los pines de la placa y de la pantalla.

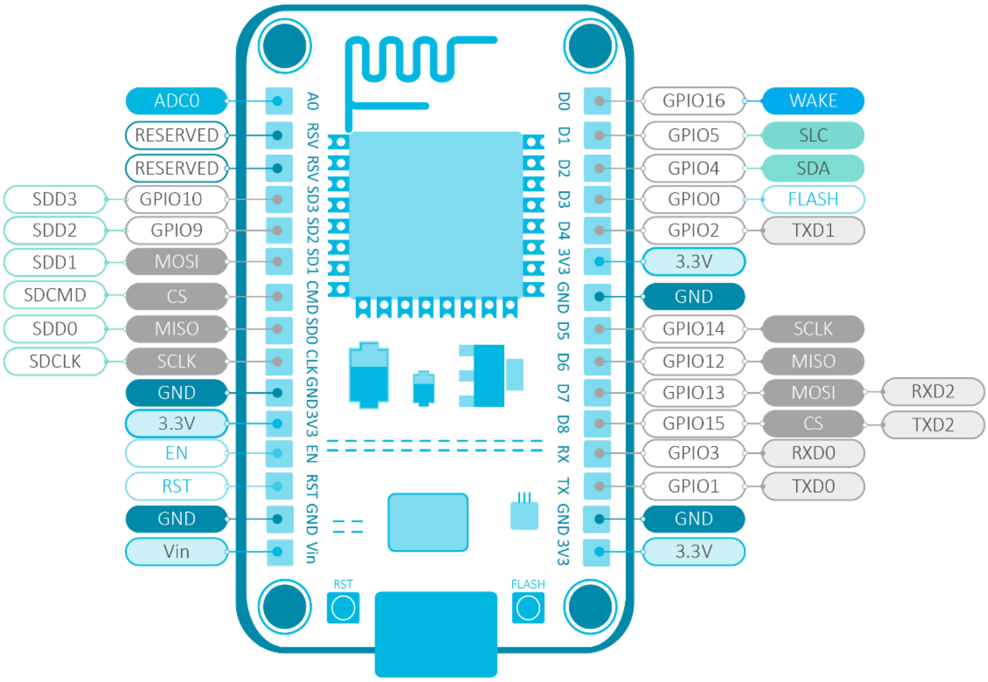


Figura 5.5 Pines de NodeMCU

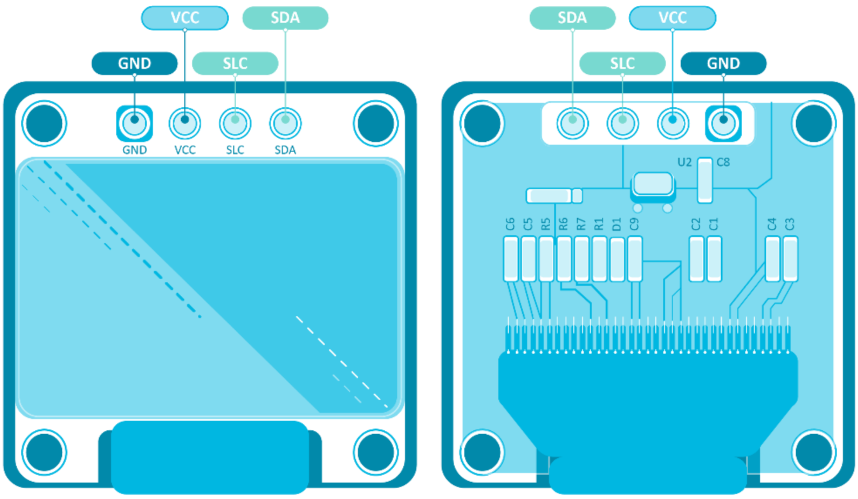


Figura 5.6 Pines de pantalla OLED

El intercambio de datos entre ambos se realiza mediante comunicación serie usando el protocolo I2C (*Inter-Integrated Circuit*). Debemos conectar el bus de datos (SDA, *System Data*) y el de reloj (SCL, *System Clock*) de ambos elementos. Los pines de VCC (*Voltage Common Collector*) y GND (ground) se vinculan a cualquiera de pines de 3.3V y GND de NodeMCU, ya que la pantalla trabaja entre 1.8V y 6V y la potencia requerida no compromete al NodeMCU. En la Figura 5.7 observamos estas conexiones.

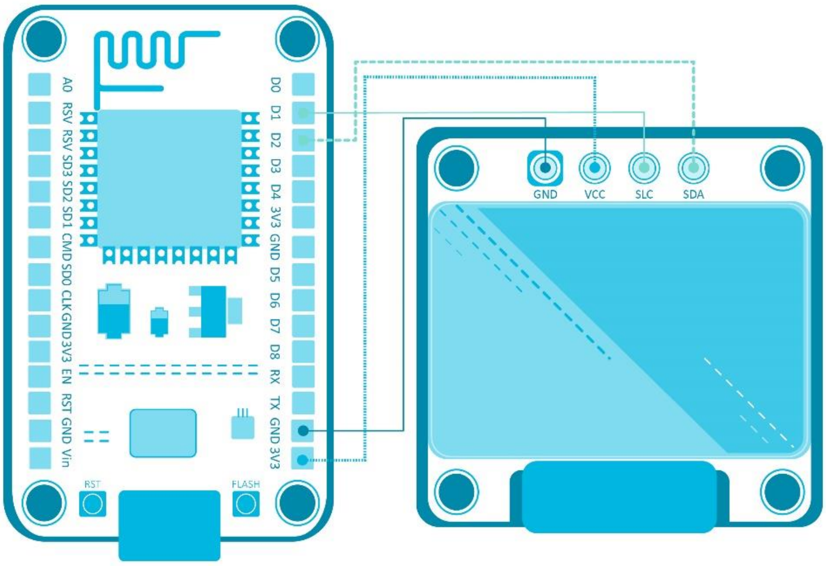


Figura 5.7 Esquema del circuito

Finalmente, el dispositivo elaborado tiene la apariencia real que se observa en la Figura 5.8.

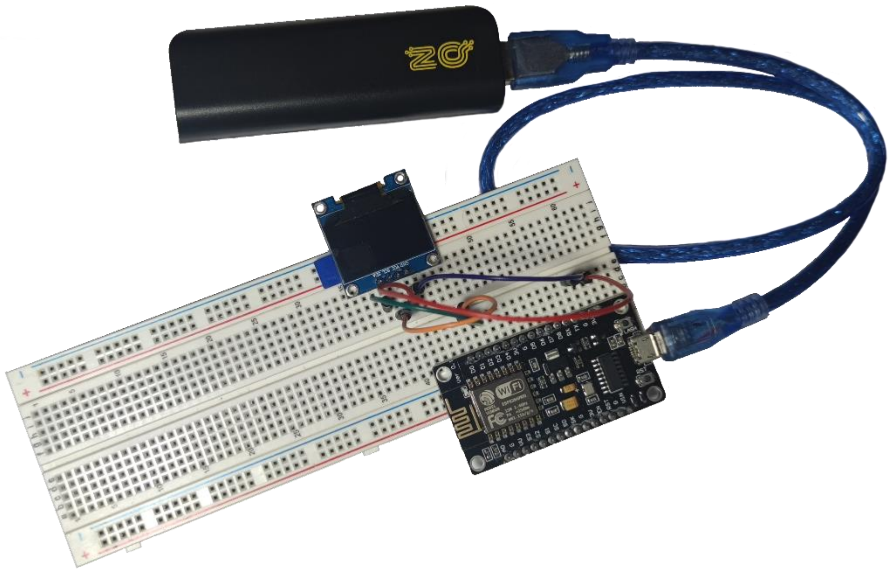


Figura 5.8 Dispositivo real

5.2.3 Programación

El programa cargado en dispositivo embebido NodeMCU, basado en ESP8266 [48], usa las correspondientes librerías para el acceso Wifi al punto de acceso y de soporte de cliente HTTP, para realizar las peticiones al *proxy*. Adicionalmente, se incluyen las relativas al control de la pantalla: una que controla la escritura de datos en esta y otra relativa a la transformación de texto a QR, necesaria para el proceso de obtención de *tokens*. En conjunto, en el desarrollo realizado se pueden considerar 2 grandes funcionalidades, una primera de comunicaciones para el acceso a Keycloak y al *proxy*, y otra de presentación al usuario, para mostrar el código QR y la información de los recursos.

Se han elaborado 3 funciones relativas a peticiones que están relacionadas con el *Device flow*. En la Figura 5.9 observamos como son llamadas en el bucle principal. La primera, `obtain_code()`, se encarga de obtener los códigos de dispositivo, usuario y URL necesario (*Device Endpoint*). También muestra al usuario la `verification_uri_complete` en la pantalla.

```
void loop() {

  // wait for WiFi connection

  String DEV_CODE;
  String TOKEN;
  int time_display=2000;


  if ((WiFi.status() == WL_CONNECTED)) {
    delay(1000);
    DEV_CODE=obtain_code();
    if(DEV_CODE!="CONNECTION ERROR"){
      do {
        TOKEN=check_code(DEV_CODE);
        delay(5000);
      } while (TOKEN=="CONNECTION ERROR"||TOKEN=="{\"error\":\"authorization_pendi
      while(1){
        display_url("/anonymous");
        display_request(request_resource(TOKEN,"/anonymous",0));
        delay(time_display);
        display_url("/user");
        display_request(request_resource(TOKEN,"/user",0));
        delay(time_display);
        display_url("/admin");
        display_request(request_resource(TOKEN,"/admin",0));
        delay(time_display);
        display_url("/all-user");
        display_request(request_resource(TOKEN,"/all-user",0));
        delay(time_display);
        display_url("/temperatura/10000");
        display_request(request_resource(TOKEN,"/temperatura/10000",1)+" C");
        delay(time_display);
        display_url("/bateria/10000");
        display_request(request_resource(TOKEN,"/bateria/10000",1)+" %");
        delay(time_display);
      }
    }
  }
}
```

Figura 5.9 Función principal del NodeMCU

La segunda, `check_code()`, comprueba si se ha otorgado acceso al *proxy*. (*Token Endpoint*). Junto con un bucle se encarga de realizar el sondeo hasta se recibe el *token* y la tercera, `request_resource()`, se encarga de solicitar recursos protegidos enviando el *token* en la petición, comunicándose con el *proxy*. También imprime por pantalla al usuario los recursos en cuestión.

5.2.4 Comprobación de funcionamiento

A continuación, se realiza la descripción del proceso de validación de la operativa real implementada, capturando los flujos intercambiados mediante Wireshark. Esta captura se comparará con el flujo teórico de *OAuth 2.0 Device Authorization Grant*. Emplearemos el rol de `user`, lo que permitirá además comprobar también la casuística de acceso denegado en cuanto a validación de permisos, puesto que la información de batería está restringida para el rol de `admin`.

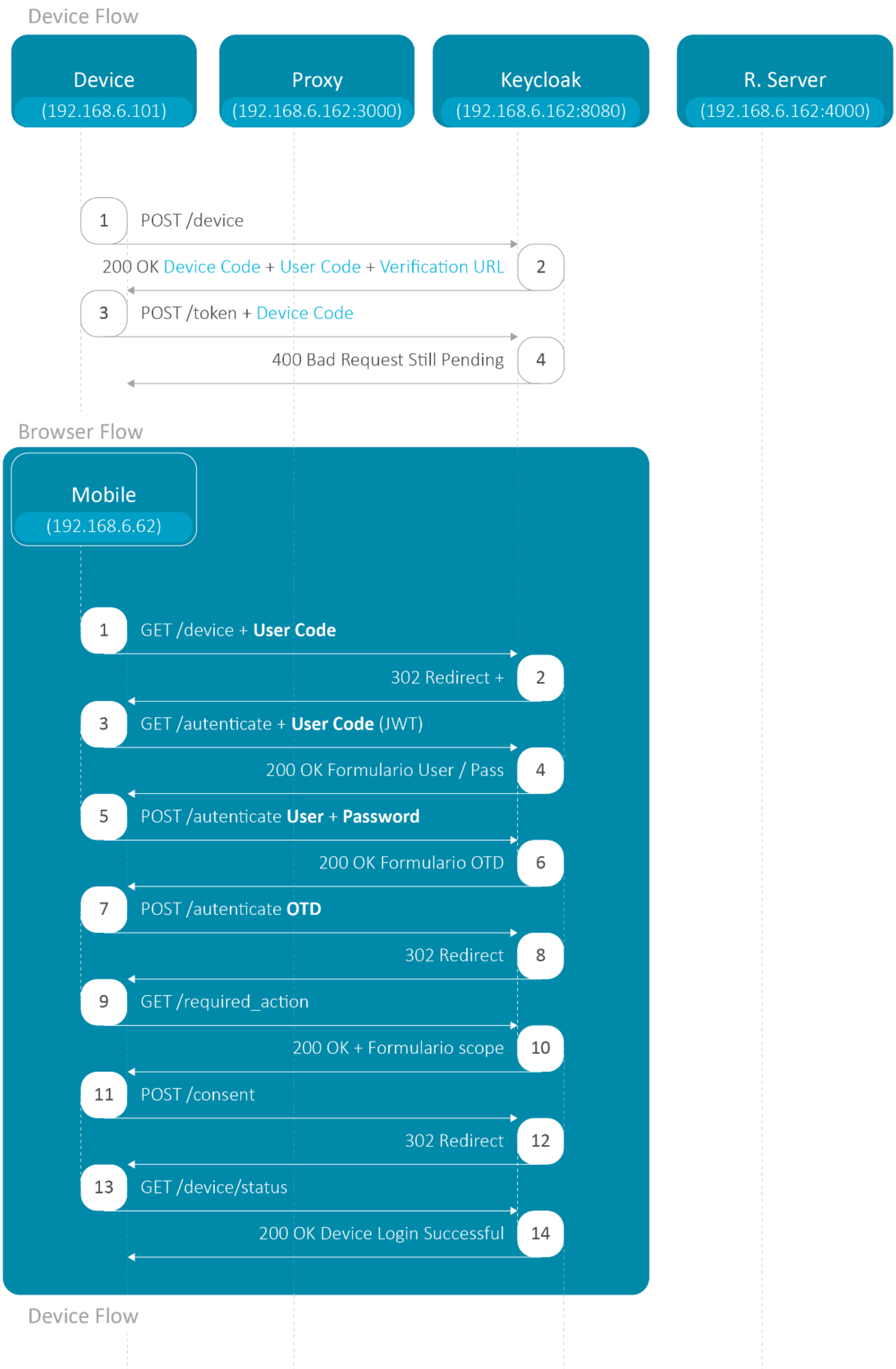


Figura 5.10 Parte 1 de la captura realizada para Device Authorization Grant

De cara a analizar la captura, vamos a dividirla en dos partes. En la Figura 5.10 encontramos la primera, la cual se centra en la obtención de los diversos códigos, parte del *Device flow*, y el *Browser flow*.

Comenzamos con el *Device flow*. Como se puede observar en el flujo teórico, lo primero que se ha de realizar es obtener del Device E*ndpoint*, presente en Keycloak, los códigos de usuario, dispositivo y la URL de verificación. La petición 1 es la que se encarga de ello y su respuesta, petición 2, es la de la Figura 5.11. Estas se corresponden con las 2 y 3 teóricas.

```
JavaScript Object Notation: application/json
  Object
    Member Key: device_code
        String value: xdilkSD7cV1LUnD3pUBNBLHUi5F45KAmNh48BTTml08
        Key: device_code
    Member Key: user_code
        String value: FXAF-QANF
        Key: user_code
    Member Key: verification_uri
        String value: http://192.168.6.162:8080/realms/smartsantander/device
        Key: verification_uri
    Member Key: verification_uri_complete
        String value: http://192.168.6.162:8080/realms/smartsantander/device?user_code=FXAF-QANF
        Key: verification_uri_complete
    Member Key: expires_in
        Number value: 600
        Key: expires_in
    Member Key: interval
        Number value: 5
        Key: interval
```

Figura 5.11 Device code, user code y verification URI

A su recepción, se extrae del documento la URL de la `verification_uri` y se genera el correspondiente código QR que se imprime por pantalla. Esto se corresponde con la petición teórica 4. En la Figura 5.12 observamos dicho QR.

Figura 5.12 QR con verification_uri_complete

Antes de analizar el *Browser flow*, debemos de centrarnos en el sondeo. El dispositivo comienza a solicitar periódicamente al `/token`, presente en Keycloak, que le devuelva el *token* correspondiente a su *device code*. Este se lo devuelve una vez finaliza con éxito el *Browser flow*. Los mensajes que se reciben mientras no se le conceden los permisos son los de la Figura 5.13, correspondientes con 3 y 4 de la captura. En el *Device Authorization flow*, se corresponde con la 5 y 6.

```
JavaScript Object Notation: application/json
  Object
    Member Key: error
        String value: authorization_pending
        Key: error
    Member Key: error_description
        String value: The authorization request is still pending
        Key: error_description
```

Figura 5.13 Autorización pendiente

Ahora sí que nos vamos a centrar en el *Browser flow*. Este se ha realizado desde un móvil, empleando una aplicación de lectura QR con la que podemos leer el QR y realizar la petición 1 fácilmente. La respuesta de esta es peculiar, ya que nos redirecciona, como es de esperar, pero nos introduce un *token* en una *cookie* de sesión. De esta forma, Keycloak es capaz de reconocer, tras el proceso de identificación, la finalidad del usuario de conceder permiso al dispositivo. En la Figura 5.14, observamos el cuerpo de este, el cual contiene parámetros necesarios como el *user code*, el *realm* y cliente asociados entre otros.


```
{
  "cid": "proxy",
  "pty": "openid-connect",
  "act": "AUTHENTICATE",
  "notes": {
    "iss": "http://192.168.6.162:8080/realms
/smartsantander",
    "OAUTH2_DEVICE_VERIFIED_USER_CODE": "FXAFQANF"
  }
}
```


Figura 5.14 Token de sesión

Como el móvil no tiene una sesión de Keycloak abierta, el usuario debe de identificarse a través de este. Nuevamente, Keycloak proporciona los formularios de usuario/contraseña y OTP (tramas 4 y 6) y el usuario responde a estos (tramas 5 y 7 respectivamente). Se observa que la operativa es idéntica a lo que ocurría en la primera prueba del laboratorio. En la Figura 5.15 podemos ver exactamente quien se ha identificado.

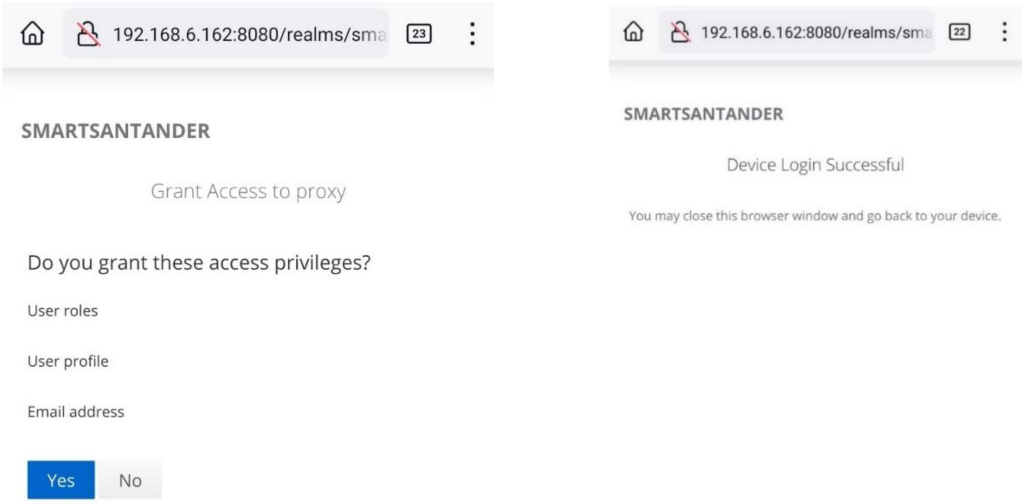
```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "username" = "user_user"
  Form item: "password" = "user_user"
  Form item: "credentialId" = ""
```

Figura 5.15 Usuario y contraseña enviados

Tras esto, Keycloak devuelve y solicita autorización de los permisos que se le van a conceder al dispositivo, como se observa en la Figura 5.16

Todas estas tramas, desde la 2 hasta la 12, están agrupadas en las 2 y 3 de la descripción teórica del *Browser Authorization flow* (*Browser flow*).

Para acabar el *Browser flow*, queda autorizar el dispositivo. En el marco teórico está representado en la trama 4, aunque en la captura, observamos que el móvil, tras aceptar el consentimiento, es redirigido al estatus del dispositivo, tramas 12 y 13. Keycloak responde al móvil con el veredicto de dicha elección, presente en la Figura 5.16. En este caso, el usuario respondió afirmativamente.



a) Concesión de privilegios de acceso **b) Dispositivo autorizado**

Figura 5.16 Concesión de permisos desde móvil

Finalizado el *Browser flow*, se retoma el *Device flow*, como se observa en la Figura 5.17. El dispositivo ha estado continuamente sondeando a Keycloak por su *token* (trama 5). Como el *Browser flow* ha acabado de manera exitosa, Keycloak envía los *tokens* (trama 6), entre ellos el *Access Token*, como se observa en la Figura 5.18.

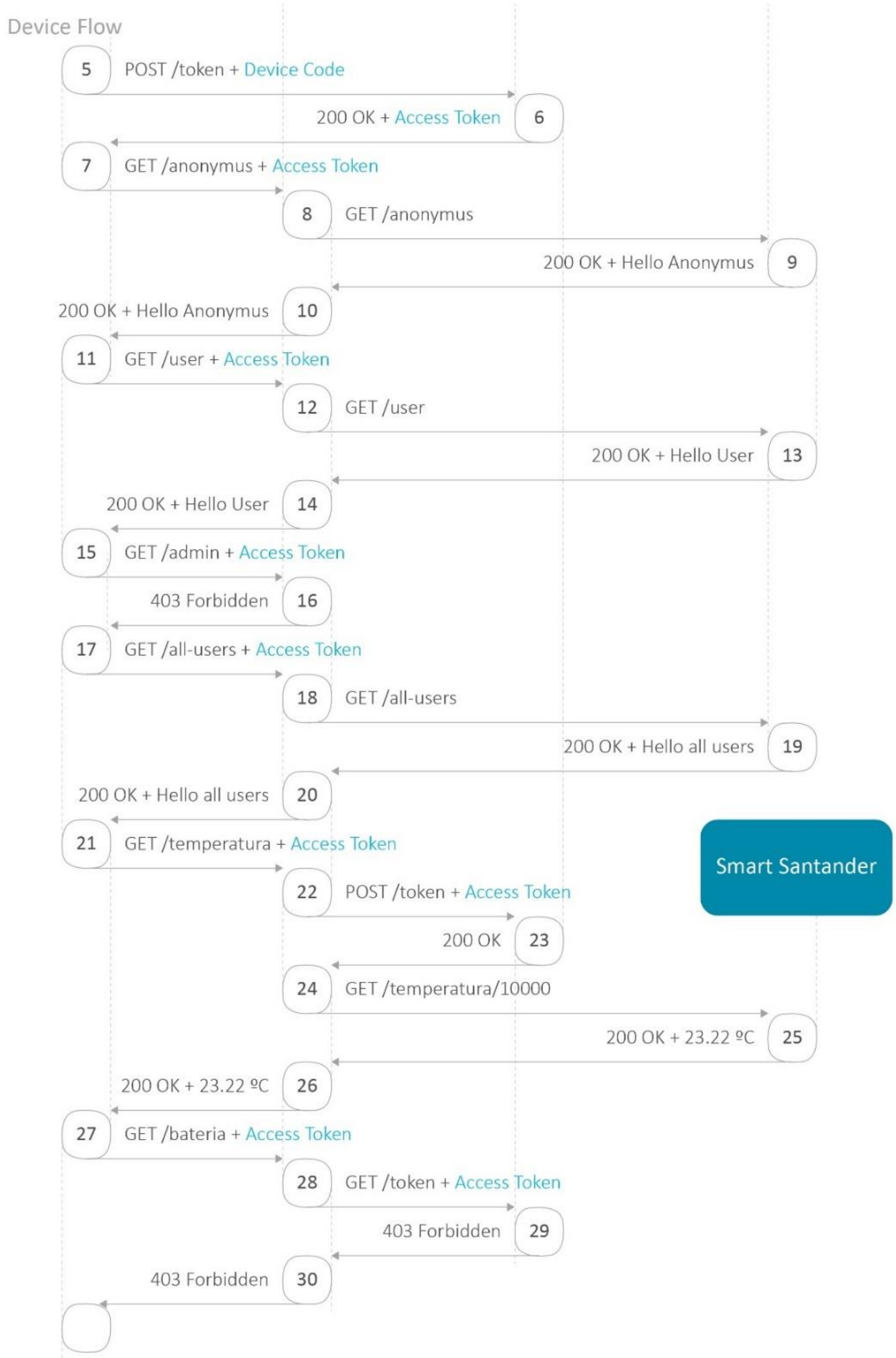


Figura 5.17 Parte 2 de la captura realizada para Device Authorization Grant

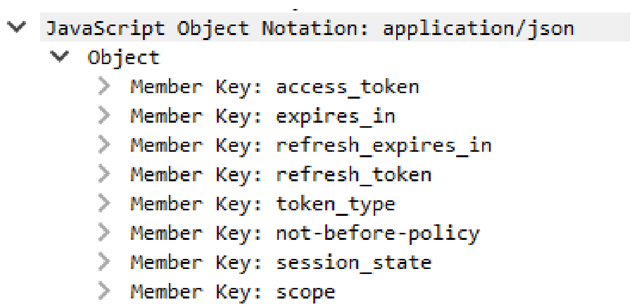


Figura 5.18 Tokens emitidos

Una vez más, vamos a comprobar el contenido del *Access Token* (Figura 5.19). Se observa que tanto su estructura como contenido es similar al realizado en la evaluación de pruebas realizada con anterioridad.


```
{
  "exp": 1664189916,
  "iat": 1664189616,
  "auth_time": 1664189613,
  "jti": "bb834e2e-46c8-43b6-846b-49267061f2e9",
  "iss": "http://192.168.6.162:8080/realms
/smartsantander",
  "aud": [
    "proxy",
    "auth_client",
    "account"
  ],
  "sub": "b99a9c70-66e2-4677-98d8-713f10beb21e",
  "typ": "Bearer",
  "azp": "proxy",
  "session_state": "5ab3f289-fa9d-404e-
82e1-58d7492f2718",
  "acr": "1",
  "allowed-origins": [
    ""
  ],
  "realm_access": {
    "roles": [
      "offline_access",
      "uma_authorization",
      "default-roles-nodejs-example"
    ]
  },
  "resource_access": {
    "auth_client": {
      "roles": [
        "user"
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "profile email",
  "sid": "5ab3f289-fa9d-404e-82e1-58d7492f2718",
  "email_verified": true,
  "name": "fdg dfg",
  "preferred_username": "user_user",
  "given_name": "fdg",
  "family_name": "dfg",
  "email": "rasdasdfsda@gamil.com"
}
```


Figura 5.19 Payload del token de la captura

Para acabar, queda analizar las tramas de solicitud de recursos. Estas ya se escapan del protocolo teórico. Las peticiones a los recursos abarcan desde la 7 hasta la 30. Como vimos anteriormente, iniciamos sesión con el usuario básico, por lo que tiene el rol de `user` y los recursos estás protegidos de acuerdo a la Tabla 4.6. Este rol permite acceder a todos los recursos excepto a dos, el recurso de administradores (`/admin`) y al nivel de batería (`/batería`). Se observa que las cuatro primeras peticiones de recursos son diferentes a las 2 siguientes. La primera diferencia es, como ocurría anteriormente, no se realiza peticiones extra a Keycloak. Es decir, efectivamente la función `protect()` evalúa los *tokens* en el *proxy*, y no Keycloak. Sin embargo, para temperatura y batería, sí se realizan, luego `enforcer()` pide la evaluación a Keycloak. La segunda diferencia es que el servidor de recursos, en las 4 primeras, sigue siendo en local, mientras que, los recursos restantes, son solicitados a Smart Santander. Y la última diferencia, es que en las relativas a temperatura y batería se solicita el recurso en concreto, es decir datos del sensor de temperatura 10000. En la Figura 5.20.a se observa la pantalla con el valor de temperatura, mientras que en la solicitud de batería (Figura 5.20.b), se deniega el acceso.

a) Temperatura del sensor **b) Acceso denegado a la batería**

Figura 5.20 Muestra por pantalla de recursos de Smart Santander

A modo de resumen de esta prueba, es que se ha conseguido realizar un dispositivo que comprobara exactamente el funcionamiento de la arquitectura propuesta, permitiendo una accesibilidad a toda clase de aparatos electrónicos. Además, se ha comprobado que el flujo teórico y la realidad van unidas, además del correcto funcionamiento de la solución propuesta, tanto el control de identidad como el de acceso.

6 Conclusiones y líneas futuras

Una vez concluida la implementación de la arquitectura propuesta y realizadas las comprobaciones pertinentes, plantearemos las conclusiones. En ellas, valoraremos si se respeta la propuesta de objetivos presentada en la fase inicial. Asimismo, comentaremos las posibles mejoras que se podrían realizar y que, por falta de tiempo, no se han podido llevar a cabo. Por último, expondremos las tendencias actuales en el ámbito de IAM.

6.1 Conclusiones

Al inicio del proyecto, se fijaron una serie de hitos a cumplir. Vamos a revisar todos ellos para evaluar su resultado.

El primero de ellos se centraba en realizar un estudio de soluciones IAM y protocolos relacionados con estos. Hemos expuesto a lo largo del capítulo 2 los protocolos y metodologías más comúnmente empleadas, implicando tanto la lectura exhaustiva de los RFC (*Request for Comments*) como la realización de pruebas y capturas previas de *proxies* de autenticación y autorización (*OAuth proxy*). También se ha presentado las soluciones mayoritariamente aceptadas como Keycloak, Gluu Server y Aerobase IAM. La comparativa realizada sobre todas ellas permitió posteriormente tomar una decisión justificada de las tecnologías a emplear en el desarrollo del proyecto.

Siguiendo las mismas, se ha seleccionado el protocolo OIDC/OAuth2 y la plataforma Keycloak como solución que implemente y soporte los mismos y las funcionalidades adicionales de gestión. Todo ello se ha formalizado en un entorno de laboratorio para poder analizar y comprender su funcionamiento. En concreto, se han cotejado las diferentes funciones de protección de recursos existentes con casuísticas tanto favorables como desfavorables. También se ha comprobado los flujos de OAuth2 tanto desde navegadores web como directamente desde clientes haciendo uso del API. Esto ha permitido testear la viabilidad de emplear cualquier tipo de dispositivo. Todo esto, sin olvidarnos de la seguridad, con doble factor de autenticación, validación de correos electrónicos, recuperación segura de contraseñas, etc.

Posteriormente se ha llevado a cabo, con éxito, el despliegue en entorno real, permitiendo securizar el acceso a valores de determinados sensores de una ciudad inteligente, particularizando para el caso de Smart Santander. Nuevamente se han realizado pruebas comprobando todas las casuísticas posibles, haciendo especial mención a la relativa del prototipo de dispositivo, la cual, ha involucrado a todo el proyecto, desde la identificación y gestión de acceso de Smart Santander, hasta la adaptabilidad de dispositivos de cualquier naturaleza. De esta forma, se ha conseguido integrar OAuth en toda clase de dispositivos, realizando un trabajo sobre un entorno real.

Por lo tanto, podemos asegurarnos de que los objetivos iniciales han sido plenamente satisfechos, e incluso, se han superado gracias al prototipo elaborado, ya que este no era objetivo inicial. A medida que el proyecto se desarrolló, surgió la idea de probarlo usando un dispositivo embebido, lo cual ha aumentado su valor para la implementación. Para la implementación del mismo, además de las pruebas de evaluación llevadas a cabo durante la fase de desarrollo en línea con el planteamiento inicial del proyecto, se han realizado actividades adicionales como la monitorización por protocolo serie del NodeMCU y los intercambios de paquetes o la valoración de diferentes alternativas de presentación e interfaz de usuario, implicando la programación de software que integre una pantalla, la comprensión de códigos QR y su generación, etc.

6.2 Líneas futuras

A continuación, el mejor conocimiento del ecosistema de autenticación y autorización que se tiene actualmente a diferencia del inicio del proyecto, nos permite definir mejoras o nuevas opciones que incrementen las funcionalidades del proyecto o incluso abran otros nuevos.

Uno de los motivos de la elección de Keycloak fue su flexibilidad, es decir, su capacidad para continuar mejorando y añadiendo funcionalidades al proyecto, pudiendo ser prolongado en el futuro. Por esta razón, este puede continuar creciendo de cara al futuro. La solución IAM puede encontrarse en continua exploración y mejora, añadiendo bases de datos, captcha en los registros para evitar creaciones masivas de cuentas, eventos para gestionar acciones programadas sobre grupos de usuarios, etc.

En este proyecto se ha introducido metodologías de control de acceso, pero se puede profundizar aún más, sobre todo en conceptos como políticas, permisos, *garants* o *claims*. Habría que estudiar su relación con aplicaciones de manera paralela, ya que hay una tendencia a que unas tengan ventajas frente a otras. También es interesante el concepto de *middlewares* en Node.js, los cuales se pueden personalizar y elaborar desde 0, aprovechando una implementación más ajustada a nuestros requisitos entre otras ventajas.

Un concepto interesante a explorar sería la configuración vía Admin REST (*Representational State Transfer*) API, la cual supondría una optimización sustancial en el ámbito de la configuración. Se facilitaría el despliegue de soluciones seguras interactuando directamente con Keycloak, evitando la forma manual.

Por otra parte, los protocolos pueden ser revisados. A medida que se trabajaba con el *Device flow*, surgió la posibilidad de utilizarse sobre ordenadores que no sean seguros para que estos obtengan un *token* de acceso. A modo de ejemplo, si nos encontramos en un ordenador, el cual está fuera de nuestro control, podría tener *malware* que extraiga las credenciales de acceso del usuario. A pesar de usar protocolos OAuth, las claves siguen estando en el ordenador infectado y se comprometería la seguridad. Para ello, podríamos usar el *Device flow*, siendo como *device*, el ordenador. De esta forma, el usuario concedería un *token* de acceso efímero al ordenador usando su móvil personal (entorno seguro), evitando que las credenciales del usuario no alcancen al PC (*Personal Computer*) y este obtendría acceso al servicio.

Otros protocolos, cuyo estudio se considera relevante y que no se han presentado en este trabajo, serían *OpenID Connect Client-Initiated Backchannel Authentication flow* (OIDC CIBA *Grant*) y *User-Managed Access* (UMA) *2.0 Grant*. El primero, introduce un dispositivo ("dispositivo de consumo") que aloja una aplicación cliente que llama a las API expuestas por los servidores de recursos y un dispositivo ("dispositivo de autenticación") en el que se realiza la autenticación del usuario final y la confirmación del consentimiento están desacoplados. Esto permite cubrir nuevos casos de uso. La cuestión es que la aplicación cliente no está bajo el control del usuario final y los dos dispositivos pueden estar físicamente separados.

El segundo se centra en permitir que el propietario de un recurso controle la autorización de la compartición de datos y otros accesos a recursos protegidos realizados entre servicios en línea en nombre del propietario o con la autorización de éste por parte de un solicitante autónomo.

Para finalizar las líneas futuras relativas a protocolos nos queda explorar las diferentes alternativas con respecto a SAML. Una posible opción sería la de añadir una extensión a Keycloak relativa a eIDAS (*electronic IDentification, Authentication and trust Services*). De esta manera, se proporcionar una forma segura a los usuarios de realizar negocios en línea como la transferencia electrónica de fondos o las transacciones con los servicios públicos entre otros, estableciendo un entorno confiable con las máximas garantías y seguridad en el marco europeo.

De igual modo, se puede continuar mejorando el desarrollo realizado sobre NodeMCU. Para hacer el prototipo más útil de cara al usuario, este se podría convertir en un asistente virtual. Se requeriría un módulo de reconocimiento de voz, ya sea elaborado con una Raspberry Pi y un micrófono o con un módulo prediseñado. También se requiere un altavoz para comunicar los datos de temperatura y batería. Otros ajustes que se podrían realizar, sería la adaptación dinámica del intervalo de sondeo, dotando de inteligencia a este para ser capaz de aumentar o disminuir la frecuencia en función de los paquetes “*Slow down*”.

Acrónimos

2FA	Two Factor Authentication
ABAC	Attribute Based Access Control
ACM	Access Control Mechanism
API	Application Programming Interface
BioID	Biometric IDentification
CIAM	Customer Identity Access Management
CIBA	Client-Initiated Backchannel Authentication
eIDAS	electronic IDentification, Authentication and trust Services
GND	Ground
HMAC	Hash-Based Message Authentication Code
HOTP	HMAC-based One-Time Password
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
I2C	Inter-Integrated Circuit
IAM	Identity and Access Management
IdP	Identity Provider
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
LDAP	Lightweight Directory Access Protocol
MFA	Multiple Factor Authentication
OAuth2	Open Authorization
OIDC	OpenID Connect
OLED	Organic Light-Emitting Diode
OS	Operating System
OTP	One Time Password
PAP	Policy Administration Point

PC	Personal Computer
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIN	Personal Identification Number
PIP	Policy Information Point
QR	Quick Response
RBAC	Role Based Access Control
REST	Representational State Transfer
RFC	Request for Comments
RP	Relying Party
SAML	Security Assertion Markup Language
SCL	System Clock
SDA	System Data
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SMS	Short Message Service
SP	Service Provider
SSL	Secure Sockets Layer
SSO	Single Sign On
STMP	Simple Mail Transfer Protocol
TLS	Transport Layer Security
TOTP	Time-based One-Time Password
UMA	User-Managed Access
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VCC	Voltage Common Collector
WiFi	Wireless Fidelity
XML	Extensible Markup Language

Bibliografía

[1] «Zero-day flaws in IoT baby monitors,» [En línea]. Available: https://portswigger.net/daily-swig/zero-day-flaws-in-iot-baby-monitors-could-give-attackers-access-to-camera-feeds.

[2] «AirTag clone,» [En línea]. Available: https://portswigger.net/daily-swig/airtag-clone-bypassed-apples-tracking-protection-features-claims-researcher.

[3] «Ley de cookies de la UE y aplicación en España,» [En línea]. Available: https://protecciondatos-lopd.com/empresas/ley-cookies/.

[4] «Autenticación,» [En línea]. Available: https://auth0.com/intro-to-iam/authentication-vs-authorization/.

[5] «MFA,» [En línea]. Available: https://auth0.com/intro-to-iam/what-is-multifactor-authentication-mfa/.

[6] «Autorizatión,» [En línea]. Available: https://auth0.com/intro-to-iam/authentication-vs-authorization/.

[7] «IAM,» [En línea]. Available: https://en.wikipedia.org/wiki/Identity_management.

[8] «SSO,» [En línea]. Available: https://auth0.com/intro-to-iam/what-is-single-sign-on-sso/.

[9] «Service provider,» [En línea]. Available: https://en.wikipedia.org/wiki/Service_provider.

[10] «Identity provider,» [En línea]. Available: https://en.wikipedia.org/wiki/Identity_provider.

[11] «Keycloak,» [En línea]. Available: https://www.keycloak.org/.

[12] «Keycloak Downloads,» [En línea]. Available: https://www.keycloak.org/downloads.

[13] «Keycloak Client dapters,» [En línea]. Available: https://www.keycloak.org/docs/latest/securing_apps/#client-adapters.

[14] «Aerobase,» [En línea]. Available: https://www.aerobase.io/.

[15] «Aerobase Documentation,» [En línea]. Available: https://aerobase.io/docs/gsg/index.html.

[16] «Gluu,» [En línea]. Available: https://gluu.org/.

[17] «Gluu Documentation,» [En línea]. Available: https://gluu.org/docs/gluu-server/4.4/.

[18] «The OAuth 2.0 Authorization Framework,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6749.

[19] «Client Credentials Grant Type,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6749#section-4.4.

[20] «Resource Owner Password Credentials Grant,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6749#section-4.3.

[21] «Authorization Code Grant,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6749#section-4.1.

[22] «Implicit Grant,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6749#section-4.2.

[23] «OAuth 2.0 Device Authorization Grant,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc8628.

[24] «OpenID Connect,» [En línea]. Available: https://openid.net/connect/.

[25] «RFC JSON Web Key (JWK),» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc7517.

[26] «Hybrid Flow,» [En línea]. Available: https://openid.net/specs/openid-connect-core-1_0.html#HybridFlowAuth.

[27] «SAML,» [En línea]. Available: https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language.

[28] «Node.js,» [En línea]. Available: https://es.wikipedia.org/wiki/Node.js.

[29] «Axios,» [En línea]. Available: https://www.npmjs.com/package/axios.

[30] «Express,» [En línea]. Available: http://expressjs.com/es/.

[31] «Descargar Keycloak,» [En línea]. Available: https://www.keycloak.org/downloads.

[32] «Web oficial de Docker,» [En línea]. Available: https://www.docker.com/.

[33] «Keycloak Docker image,» [En línea]. Available: https://hub.docker.com/r/jboss/keycloak/.

[34] «Setting up HTTPS/SSL,» [En línea]. Available: https://www.keycloak.org/docs/latest/server_installation/#_setting_up_ssl.

[35] «TOTP: Time-Based One-Time Password Algorithm,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6238.

[36] «HOTP: An HMAC-Based One-Time Password Algorithm,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc4226.

[37] «Node.js adapter, Protecting resources,» [En línea]. Available: https://www.keycloak.org/docs/latest/securing_apps/#protecting-resources.

[38] «Managing policies,» [En línea]. Available: https://www.keycloak.org/docs/latest/authorization_services/index.html#_policy_overview.

[39] « Keycloak nodejs connect,» [En línea]. Available: https://github.com/keycloak/keycloak-nodejs-connect.

[40] «Protecting resources and functions,» [En línea]. Available: https://www.keycloak.org/docs/18.0/securing_apps/#protecting-resources.

[41] «Audience support,» [En línea]. Available: https://github.com/keycloak/keycloak-documentation/blob/main/server_admin/topics/clients/oidc/con-audience.adoc.

[42] «Device Access Token Request,» [En línea]. Available: https://www.rfc-editor.org/rfc/rfc8628#section-3.4.

[43] «NodeMCU Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/NodeMCU.

[44] «Amazon NodeMCU V3,» [En línea]. Available: https://amzn.eu/d/ibyd6UA.

[45] «Amazon Protoboard,» [En línea]. Available: https://amzn.eu/d/9KMdcg9.

[46] «Amazon OLED Display I2C,» [En línea]. Available: https://amzn.eu/d/aAimOSt.

[47] «Amazon Power Bank 2200 mAh,» [En línea]. Available: https://amzn.eu/d/fDM7eNP.

[48] «Wikipedia ESP8266,» [En línea]. Available: https://es.wikipedia.org/wiki/ESP8266.