Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Securización mediante Keyrock y Wilma de Aplicación y Servicios Web para Vehículo Inteligente.

Autor: Álvaro Carmona Palomares
Tutor: María Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Securización mediante Keyrock y Wilma de Aplicación y Servicios Web para Vehículo Inteligente.

Autor:
Álvaro Carmona Palomares

Tutor:
María Teresa Ariza Gómez
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Securización mediante Keyrock y Wilma de Aplicación y Servicios Web para Vehículo Inteligente.

Autor: Álvaro Carmona Palomares

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Me gustaría agradecer a toda mi familia por tanto apoyo brindado, en especial a mis padres por su apoyo para poder alcanzar mis metas y a mis tíos por su gran apoyo y facilidades durante toda mi etapa universitaria.

Gracias a mi tutora, Dra. doña María Teresa Ariza Gómez por un trato tan cercano, ayudándome en todo momento durante nuestras reuniones y por su colaboración.

Gracias a todos mis compañeros por todos estos años, los cuales me han formado profesionalmente e influido en la persona que soy en la actualidad.

Álvaro Carmona Palomares

Sevilla, 2020

Resumen

El desarrollo de nuevas tecnologías en el ámbito del *Internet de las Cosas* posibilita la creación de nuevas soluciones, ofreciendo determinadas opciones con un enfoque en facilitar la vida de las personas. Dentro de estos enfoques se puede destacar la idea de vehículo inteligente, permitiendo una gran variadad de funcionalidades, tales como poder detectar accidentes o medir las velocidades del vehículo, avisando al conductor en caso de no circular a una velocidad adecuada.

El Proyecto abordado que se expone en la presente memoria es una continuación del proyecto iniciado por Pablo Bermejo Pérez en 2019 para su Trabajo Fin de Grado, el cual permite la lectura de información de contexto mediante el uso del *Fiware Generic Enabler Orion Context Broker* y haciendo uso de una *API REST*[1] para consumir la información producida por sensores instalados en vehículos, dándose así la situación de vehículo inteligente.

El presente proyecto aborda la seguridad del proyecto anteriormente mencionado, haciendo uso de tres nuevos *Fiware Generic Enablers, Keyrock* (Identity Manager), *PEP-Proxy Wilma y IoT Agent Ultralight*, los cuales serán foco de explicación durante toda la memoria. No obstante, se han implementado otras mejoras de seguridad básicas como el uso del protocolo *Https*[2] en la comunicación del usuario con la aplicación web o el uso del protocolo Oauth 2.0 para aumentar la seguridad en el acceso a las APIs.

También ha sido desarrollada una aplicación web mediante el uso de diversas tecnologías. Para la parte de Front-End se ha optado por el uso de HTML, CSS y JavaScript, mientras que para el diseño de la parte Back-End se han usado JavaServer Pages (JSP) y dos Servlets.

La información de contexto será generada mediante sensores instalados en teléfonos móviles, los cuales simularán sensores reales en un vehículo, comunicándose con un Agente IoT Ultralight. El Agente IoT ejercerá de barrera entre el mundo del Internet de las Cosas y la parte de Context Broker del sistema desarrollado.

El último gran cambio abordado es la creación de un nuevo servicio web RESTful mediante el framework Spring para la comunicación con Keyrock idm. De esta forma se permite separar toda comunicación directa con el elemento *Gestor de Identidad* del sistema y las comunicaciones para el acceso a la información de los usuarios.

[1] Estilo de arquitectura software para sistemas distribuidos.
[2] El Protocolo de aplicación basado en HTTP, agregando transferencias seguras gracias al uso de SSL.

Abstract

The development of new technologies enables the creation of new solutions which offer certain options facilitating people's lives. Within these approaches, the idea of a smart vehicle can be highlighted, allowing a wide variety of functionalities such as being able to detect accidents or measure vehicle speed.

The Project that is exposed in this report is a continuation of the project started by Pablo Bermejo Pérez in 2019 for his Final Degree Project which allows the reading of context information by using the Fiware Generic Enabler Orion Context Broker. Pablo's project uses a REST API to consume the information produced by sensors installed in vehicles thus giving the situation of a smart vehicle.

This project addresses the security of the mentioned project, making use of three new Fiware Generic Enablers, Keyrock (Identity Manager), PEP-Proxy Wilma and IoT Agent Ultralight, which will be the focus of explanation throughout the memory.

However, other basic security improvements have been implemented, such as the use of the Https protocol in user communication with the web application or the use of the Oauth 2.0 protocol to increase security in accessing APIs. A web application has also been developed through the use of various technologies. For the Front-End part, the use of HTML, CSS and JavaScript has been chosen, while for the design of the Back-End part, JavaServer Pages (JSP) and two Servlets have been used.

This project addresses the security of the aforementioned project, making use of three new Fiware Generic Enablers, Keyrock (Identity Manager), PEP-Proxy Wilma and IoT Agent Ultralight, which will be the focus of explanation throughout the memory.

Índice

Índice de Tablas

ÍNDICE DE FIGURAS

GLOSARIO

PEP	Policy Enforcement Point (Punto de Aplicación de la política)
PAP	Policy Administration Point (Punto de Administración de políticas)
PDP	Policy Decision Point (Punto de Decisión de la política)
REST	Transferencia de Estado Representacional (Arquitectura Software)
HTTP	Protocolo de transferencia de hipertexto
IoT	Internet of Things
IDM	Identity Manager (Gestor de Identidad)
JSON	JavaScript Object Notation

1 INTRODUCCIÓN

Internet facilita la información adecuada, en el momento adecuado, para el propósito adecuado.

- Bill Gates -

La idea de un mundo conectado no es una novedad, pero está llegando a su máximo apogeo gracias al concepto de dispositivos conectados a través de Internet, los cuales presentan grandes ventajas debido al amplio marco de características ofrecidas al cliente final (televisores inteligentes, electrodomésticos inteligentes, etc.). El gran aumento de esta tendencia es irrefutable, teniendo un aumento de casi 6000 millones de dólares en el mercado americano y estimándose un volumen de 75 billones de dispositivos conectados para el año 2025, requiriéndose un gran ancho de banda y posibilitando el desarrollo de nuevas ideas como la de vehículo conectado, idea que sienta las bases para el proyecto a abordar.

1.1 Motivación

Este Proyecto profundizará en la idea de vehículo conectado debido a su gran repercusión en la sociedad actual. La implementación de vehículo conectado aportará mayor seguridad en la condución vial debido a la instalación de determinados sensores, los cuales permitirán la medición de parámetros como la velocidad y localización del vehículo. Mediante estas mediciones se pueden realizar diversas acciones:

- Notificación de exceso de velocidad mediante respuestas luminosas generadas por un LED.
- Estudios de comportamiento gracias a Big Data mediante los datos de una población determinada.
- Detección de accidentes de tráficos en la vía por la que circule el conductor.

Para poder implementar la idea de vehículo contectado es necesario el diseño de un sistema que permita el procesamiento de toda la información generada por cada vehículo, y poder ofrecer a cada usuario acceso a su propia información. Estos requisitos imponen el uso de determinados componentes especializados en desempeñar determinadas tareas de procesamiento y gestión de información.

El grueso de los componentes del sistema pertenecen a la plataforma Open Source *FIWARE* impulsada por la Unión Europea a principios de 2012 para el desarrollo y despliegue de soluciones IoT. La propia definición de vehículo conectado hace idóneo el uso de la plataforma *FIWARE* para su securización, debido a los mecanismos ofrecidos. Dentro del amplio catálogo de *FIWARE* se usará Keyrock idm como Gestor de Identidad, PEP-Proxy

Wilma, Orion Context Broker y IoT Agent.

El procesamiento de la información es la parte fundamental dentro de cualquier solución IoT, lo que hace indispensable el uso de un componente que permita procesar dicha información, así como de informar a ciertas aplicaciones en caso de que la información de contexto sea modificada. El componente Orion Context Broker lleva a cabo las funcionalidades descritas, siendo el componente principal de cualquier aplicación "Powered by Fiware", proporcionando una API NGSIv2 para la comunicación. Un aspecto a destacar es el mecanismo de subscripciones para poder almacenar los cambios de la información de contexto deseada. La información procesada será generada por sensores, los cuales enviarán toda la información a un IoT Agent Ultralight encargado de ejercer de barrera entre el exterior (mundo del Internet de las Cosas) y Orion Context Broker, permitiendo la securización adicional del sistema en los puertos norte y sur del IoT Agent seleccionado.

Se hace necesario definir dos servicios web RESTful. El primer servicio web define una API para la comunicación exclusiva con Keyrock idm. En cambio, el segundo es el encargado de acceder a la información de contexto generada por los sensores y de llevar a cabo las subscripciones pertinentes para poder almacenar la información ante cambio en los valores de las entidades del Context Broker. Ambos servicios han sido creados usando el framework Spring basado en Eclipse para el desarrollo de aplicaciones Java.

1.2 Antecedentes

El proyecto realizado por Pablo Bermejo Pérez en 2019 titulado *Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring* [1] para su Trabajo de Fin de Grado sentó las bases para el desarrollo del presente proyecto. En dicho proyecto se utilizo la plataforma FIWARE, concretamente su componente central Orion Context Broker para poder procesar y gestionar toda la información producida por sensores instalados en los vehículos.

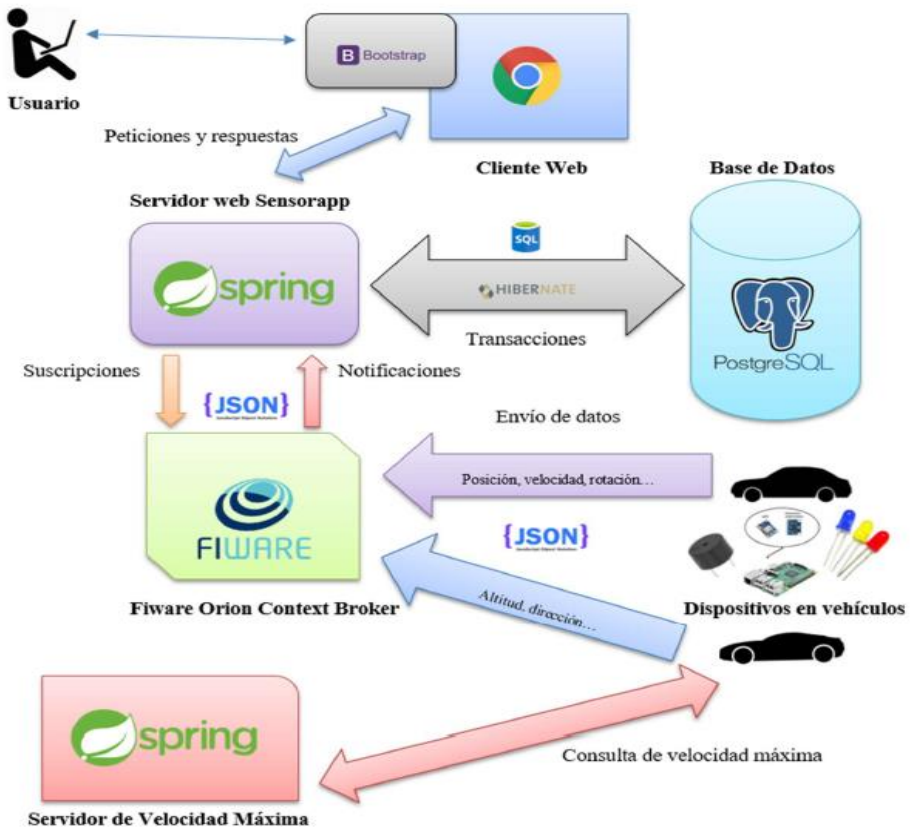


Ilustración 1 Arquitectura del Sistema de Pablo Bermejo

El sistema desarrollado por Pablo Bermejo Pérez, cuya arquitectura se muestra en la ilustración 1 hace uso de una base de datos externa a cualquier componente FIWARE para almacenar toda la información de contexto. Por lo tanto, mediante un mecanismo de subscripciones se puede notificar a una aplicación interesada cambios de la información y ésta puede hacer uso de un *Sistema Gestor de Base de Datos* para poder almacenar toda la información en una base de datos.

Toda la información almacena en el sistema es consultada por usuarios, permitiendo la lectura de toda la información generada para cada vehículo en función de su matrícula, tales como la velocidad, latitud, longitud, fecha, hora, etc.

1.3 Objetivos

El objetivo del Proyecto es la securización y avance del sistema de vehículo conectado que se empezó a desarrollar en el Trabajo de Fin de Grado *Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring*, generando información de contexto mediante sensores y procesando toda la información mediante componentes de la plataforma FIWARE.

La solución desarrollada pretende securizar todo el sistema, aportando la identificación de usuarios dentro del sistema mediante un Gestor de Identidad. Un usuario quedaría identificado dentro del sistema mediante un token y un "Bearer" token solicitados a Keyrock. El primer token autentifica al usuario dentro del sistema, mientras que el segundo autentica al usuario en la aplicación creada en Keyrock. Este segundo token será vital, siendo solicitado a Keyrock mediante un flujo definido por el protocolo Oauth 2.0, el cual aportará mayor seguridad a la hora de poder solicitar dicho token.

También se avanza respecto a la arquitectura del sistema que precede al presente proyecto con la inclusión de IoT Agent Ultralight, el cual facilitará la comunicación entre la interfaz del Internet de las Cosas (sensores) y Orion Context Broker. El mundo del Internet de las Cosas será representado mediante sensores móviles en movimiento, simulando así la instalación de sensores reales en un vehículo conectado. Estos sensores enviarán toda la información a un servicio web externo, el cual se encargará de gestionar la información y realizar las peticiones correspondientes al IoT Agent Ultralight para poder modificar los datos de las entidades correspondientes a cada sensor en el Orion Context Broker.

Se hace indispensable el uso de otros mecanismos de seguridad básicos como las sesiones en Java implementadas en la aplicación web desarrollada y el uso de protocolos de transferencia seguros como Https, permitiendo así la seguridad en las comunicaciones mediante el cifrazo de las peticiones que un usuario realice al sistema.

El proyecto abarcará todo el escenario de un sistema real, cubriendo tanto el área de procesamiento y visualización de la información de contexto, como el área de seguridad del sistema, ofeciendo así una visión fiel de un sistema funcional en el mundo real.

1.4 Descripción de la solución

Se presenta a continuación la solución desarrollada para poder alcanzar los objetivos establecidos en la presente memoria.

1.4.1 Funcionalidades

El sistema presenta las siguientes funcionalidades:

- ✓ Autenticación y verificación de usuario mediante Keyrock Identity Manager.
- ✓ Interfaz de usuario diferenciada en función de usuario administrador y usuario estándar.

- ✓ Visualización de manuales de usuario y administrador.
- ✓ Personalización de interfaz de usuario, mostrando los parámetros de autenticación en todo momento.
- ✓ Opciones de administrador para la alta y baja de usuarios.
- ✓ Consulta de la información propia de cada usuario, en función de su user_id y matrícula.
- ✓ Consulta de las entidades subscritas, actualizando la información en caso de cambio de la información de contexto.
- ✓ Cancelar una subscripción por baja de vehículo en el sistema.

1.4.2 Arquitectura del sistema

La arquitectura del sistema, la cual se muestra en la ilustración 2 detalla todo el sistema desarrollado, pudiendo identificarse los componentes de la plataforma FIWARE mediante figuras cuadradas. Los cuadrados de color rojo representan los componentes de FIWARE encargados de la seguridad del sistema, mientras que los cuadrados de color azul representan los componentes encargados de la gestión y procesamiento de la información de contexto.

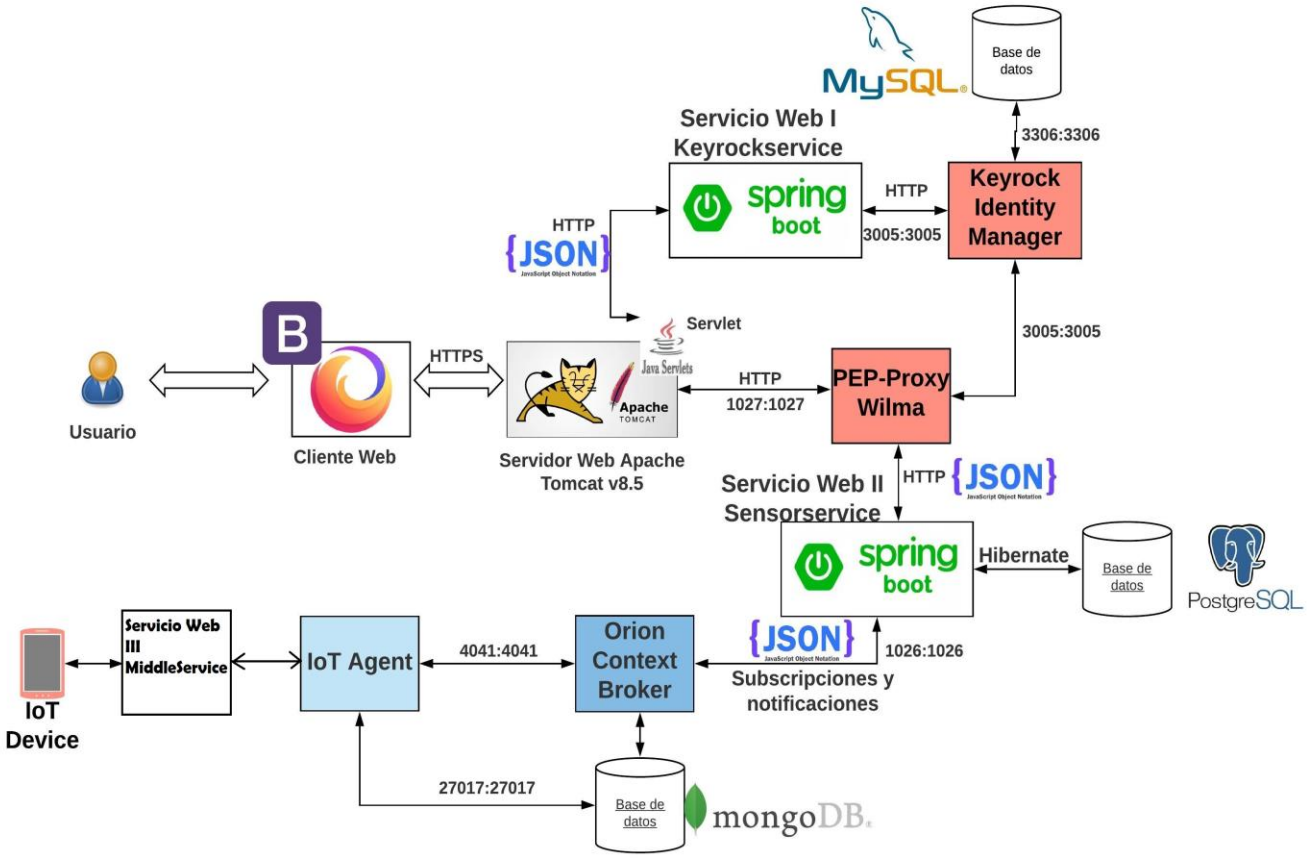


Ilustración 2: Arquitectura del Sistema

También han sido detallados los protocolos usados en las comunicaciones y los puertos estándar usados por los componentes de la plataforma FIWARE.

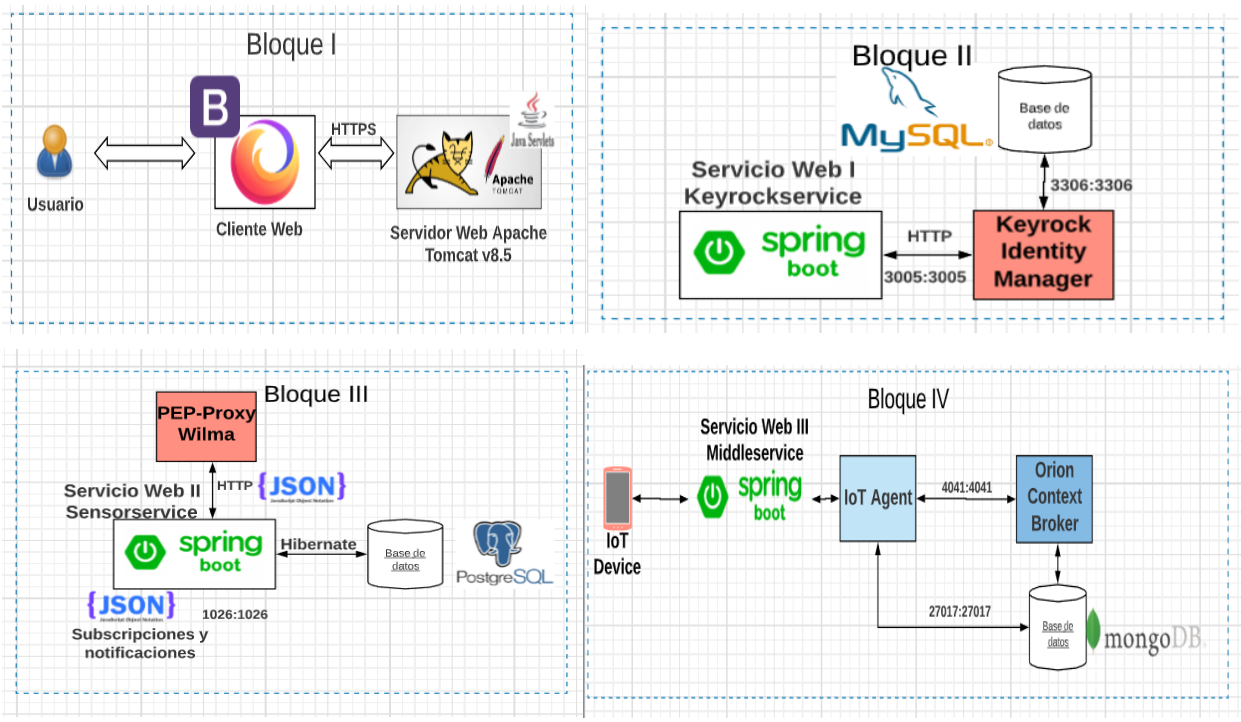


Ilustración 3 División Arquitectura del Sistema

La arquitectura es compleja, por lo que se procede a una división en bloques según lo mostrado en la ilustración 3 para poder simplificar el estudio del sistema:

- **Bloque I:** Formado por los elementos necesarios para la aplicación web. La aplicación web se alojará en un servidor web Apache Tomcat v8.5 y será necesario la presentación de una interfaz de usuario en el lado cliente para poder interactuar con el sistema.
- **Bloque II**: Formado por los elementos que actúan en las comunicaciones con el Gestor de Identidad Keyrock. Para poder interceptar las peticiones realizadas mediante la aplicación web se ha diseñado un servicio web Restful llamado Keyrockservice mediante Spring, cuyos Endpoint[3] serán los encargados de realizar la comunicación con Keyrock gracias a Keyrock API.
- **Bloque III**: Formado por los elementos necesarios para visualizar la información de contexto. En este bloque es necesario un elemento que actúe de Punto de Aplicación de la Política (la terminología XACML se detallará en siguientes capítulos), interceptando las peticiones a los recursos y verificando el "Bearer" token del usuario. Por ello se hace uso de PEP-Proxy Wilma, limitando el acceso a la información almacenada en la base de datos relacional, la cual es gestionada por el sistema gestor de base de datos PostgreSQL. El último elemento será el servicio web Sensorapp, encargado de crear las subscripciones y solicitar la información de contexto almacenada en la base de datos.
- **Bloque IV**: Formado por los elementos necesarios para procesar y generar la información de contexto. El eje de todo sistema "Powered by FIWARE" será Orion Context Broker, encargado de procesar y almacenar mediante la creación de entidades toda la información producida por los sensores IoT. El último elemento será IoT Agent, presentando una interfaz adicional entre los dispositivos del *Internet de las Cosas* y nuestro sistema. También se debe incluir el servicio web externo encargado de enviar la información de cada sensor al IoT Agent Ultralight.

[3] URL con la que se puede acceder a un recurso o método de un servicio web.

1.5 Estructura de la memoria

Se facilita a modo de guía un breve resumen de la estructura de la memoria, facilitando así su lectura y comprensión:

1. **Introducción**: Explicación de las motivaciones para acometer el proyecto, así como la solución desarrollada.
2. **Recursos utilizados**: Diferenciamos dos tipos de recursos, hardware, los cuales son los dispositivos físicos usados durante el desarrollo del proyecto y software, listando los recursos usados durante el desarrollo del mismo.
3. **Tecnologías utilizadas**: Resumen de las tecnologías empleadas durante el desarrollo del proyecto.
4. **Plataforma FIWARE**: Explicación detallada de la plataforma FIWARE, así como de los componentes FIWARE usados y las funciones que desempeñan.
5. **Solución desarrollada: Servicios Web y Base de Datos**: En primera instancia es necesario abordar el diseño completo del sistema, teniendo una visión global de los componentes explicados en el punto anterior. También se abordarán los dos sistemas Restful Keyrockservice y Sensorservice, así como la base de datos gestionada por PostgreSQL, en la cual se almacenará el grueso de la información del sistema.
6. **Solución desarrollada: Generación y consumo de información de contexto**: Tratamiento y gestión de la información de contexto generada mediante sensores instalados en dispositivos móviles en movimiento y explicación del servicio web Restful externo encargado del envío de esta información al IoT Agent Ultralight.
7. **Solución desarrollada: Aplicación Web e Interfaz de Usuario**: Se abarca detalladamente toda la aplicación web desarrollada, tanto la parte de Front-End con la interfaz de usuario, como la parte Back-End con el Servtlet desarrollado.
8. **Conclusiones y líneas futuras**: Se mencionan algunos puntos del proyecto a mejorar en una futura actualización del mismo, los cuales no han sido optimizados en el presente proyecto.

Además de los ocho capítulos mencionados, se detallan una serie de anexos, los cuales clarifican en gran medida la instalación y configuración de todos los componentes del sistema, así como los posibles errores a subsanar durante dicho proceso:

Anexo A: Instalación y configuración de los componentes FIWARE.

Anexo B: Instalación y configuración servidor Base de Datos PostgreSQL.

Anexo C: Instalación y configuración del entorno Spring Tool Suite 4.

Anexo D: Puesta en marcha del sistema.

2 RECURSOS UTILIZADOS

La innovación distingue a los líderes de los seguidores.

- Steve Jobs -

En este capítulo se detallan los recursos empleados durante las pruebas realizadas, enumerando recursos hardware y software, con la finalidad de poder reproducir el sistema en las mismas condiciones iniciales.

2.1 Recursos hardware

2.1.1 Ordenador Portátil Intel® Core™ i7-7700HQ CPU @ 2.80GHz

Se ha utilizado un ordenador portátil con la finalidad de almacenar la máquina virtual Ubuntu, la cual contendrá la gran mayoría de los componentes del sistema para las pruebas.

Ilustración 4 Intel Core i7 7th Gen

- Modelo: Acer Predator G9-793.
- Procesador: Intel® Core™ i7-7700HQ CPU @ 2.80GHz.
- Memoria RAM: 16 GB.

- Tarjeta gráfica: NVIDIA GeForce GTX 1060.
- Almacenamiento: 1TB HDD + 118GB SSD.
- Sistema Operativo: Windows 10 Pro.

2.2 Recursos software

2.2.1 Máquina virtual Ubuntu 18.04 LTS

Para el desarrollo del sistema se ha optado por un sistema operativo Ubuntu 18.04 LTS. La máquina virtual se ha creado con VMWARE Workstation 15 Player, software gratuito con limitaciones respecto a las prestaciones de máquinas virtuales.

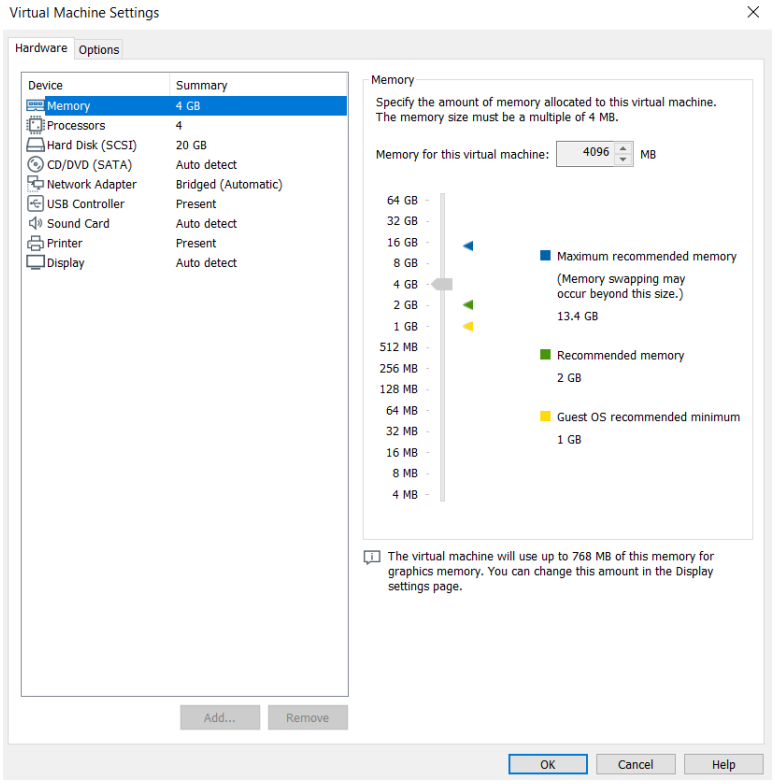


Ilustración 5 Configuración Máquina Virtual Ubuntu

Se debe puntualizar la configuración de red seleccionada, configurando el modo Bridged para asignar una dirección IPv4 de la red local a la máquina virtual, simulando un ordenador individual dentro de la red.

2.2.2 Spring Tool Suite 4

El entorno seleccionado para el desarrollo ha sido Spring Tool Suite 4, siendo ésta la versión más reciente.

Ilustración 6 Spring Tool Suite 4

Este Entorno de Desarrollo Integrado (IDE en su sigla del inglés) se basa en la versión Java EE de Eclipse y está customizado para el desarrollo fácil de aplicaciones usando el framework Spring y Spring Boot.

2.2.3 Android Studio

La aplicación móvil ha sido desarrollada mediante el entorno de desarrollo integrado Android Studio, siendo este entorno el oficial para la plataforma Android.

Ilustración 7 Logo Android Studio

2.2.4 Apache Tomcat v8.5 Server

La aplicación web desarrollada será alojada en un servidor Apache Tomcat v8.5 Server, el cual puede ser descargado de su página web oficial.

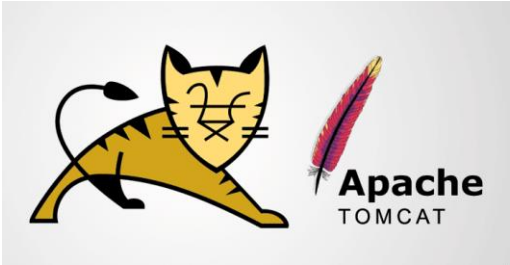


Ilustración 8 Apache Tomcat Server

En el momento de publicación de la memoria existe una versión más reciente de Apache, siendo la versión v9.0. Sin embargo, dicha versión no recibe soporte, obteniendo errores de tipo “reflection”, por lo que se ha optado por el uso de la versión v8.5.

2.2.5 Navegador Web Mozilla Firefox

Ilustración 9 Mozilla Firefox

El navegador usado durante las pruebas en el lado cliente ha sido Mozilla Firefox, pero la aplicación web ha sido diseñada para poder ser accesible desde cualquier navegador web actual.

2.2.6 Postman

Una herramienta muy útil que se usará frecuentemente es Postman, la cual nos permitirá realizar peticiones a Orion Context Broker directamente para solicitar información o realizar distintas tareas.

Ilustración 10 Logo Postman

No se ha desarrollado una API que permita a los usuarios administradores del sistema interactuar directamente con Orion Context Broker, a diferencia de la API Restful que se ha desarrollado para la comunicación con Keyrock idm. Al no existir esta posibilidad Postman permitirá de forma sencilla realizar las siguientes peticioes a Orion Context Broker:

- Solicitar las entidades creadas, las cuales identifican a un sensor.
- Solicitar las subscripciones creadas, las cuales generarán las notificaciones para comunicar cambios en la información de contexto.
- Creación de un servicio a través del puerto norte del IoT Agent Ultralight.
- Solicitar todos los sensores y servicios creados.

Todas las peticiones recurrentes que han sido usadas a lo largo del proyecto pueden ser descargadas en [2], pudiendo importarse directamente a Postman para realizarlas cumplimentando el entorno específico de cada usuario.

3 TECNOLOGÍAS UTILIZADAS

Tu margen es mi oportunidad.

- Jeff Bezos -

En este capítulo se introducen las principales tecnologías empleadas durante el desarrollo del presente proyecto, siendo indispensables a la hora de reproducir el comportamiento del mismo.

3.1 FIWARE Generic Enablers

Se procede a detallar los distintos componentes dentro del marco de FIWARE que han sido utilizados en nuestro sistema. En la presente sección se pretende introducir los componentes, los cuales serán detallados en profundidad en los Capítulos 4 y 5 respectivamente.

3.1.1 Orion Context Broker

FIWARE Orion Context Broker es el componente principal de cualquier solución IoT "Powered by Fiware", siendo el componente encargado de procesar toda la información producida por nuestros sensores.

Ilustración 11 Orion Context Broker

Orion Context Broker es una implementación en C++ de la API REST NGSIv2, permitiendo procesar y gestionar la información de contexto mediante entidades en formato JSON. La gran ventaja a destacar es un mecanismo de subscripciones, permitiendo a aplicaciones subscribirse a las entidades de interés, pudiendo ser avisadas por medio de notificaciones generadas por el Context Broker frente a cambios en la información de contexto.

3.1.2 Identity Manager Keyrock

El administrador de identidad Keyrock ejercerá de Punto de Decisión de la Política (Policy Decision Point, PDP) evaluando las distintas peticiones de acceso al sistema frente a unas políticas de acceso definidas y emitiendo las decisiones de acceso, autorizando o denengando el acceso de un usuario al sistema.

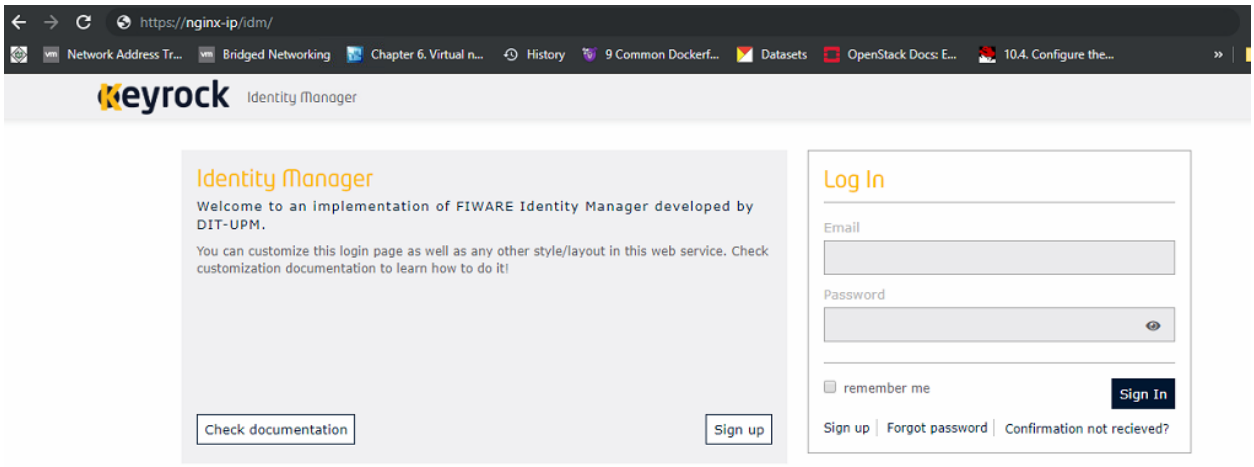


Ilustración 12 Interfaz de Usuario de Keyrock idm

Keyrock ofrece una interfaz de usuario fácil e intuitiva, la cual se muestra en la ilustración 12, permitiendo la creación de aplicaciones, permisos específicos para los usuarios dados de alta o la creación de roles y organizaciones. Todas estas opciones serán abordadas de forma detallada en el Capítulo 7.

Este FIWARE Generic Enabler es pues el principal componente encargado de la seguridad de nuestro sistema, el cual, combinado con PEP-Proxy Wilma y Authzforce otorgará un nivel de seguridad avanzado.

3.1.3 PEP-Proxy Wilma

PEP-Proxy Wilma ejercerá de Punto de Aplicación de la Política (Policy Enforcement Point, PEP), el cual escuchará las peticiones de los usuarios a un recurso y realizará una petición de decisión a nuestro PDP Keyrock, el cual emitirá la decisión a Wilma, permitiendo el acceso al recurso o denegándolo.

Wilma nos permitirá alcanzar un nivel intermedio de seguridad, siendo el nivel de seguridad alcanzado dentro del presente proyecto en el marco de FIWARE.

3.2 Docker y Docker-compose

Es importante automatizar el despliegue de los distintos componentes de nuestro sistema a la hora de ser usado en el mundo real. Debido a esta necesidad es necesario usar Docker, proyecto de código abierto para automatizar el despliegue de aplicaciones en unidades denominadas contenedores de software, utilizando el kernel de Linux.

Ilustración 13 Logo de Docker

La principal característica de Docker es el uso de las imágenes, permitiendo obtener una aplicación junto a todas sus dependencias, característica usada en la gran mayoría de los componentes del sistema. Se debe mencionar la diferencia entre los contenedores y las imágenes usados en Docker:

- **Imágenes**: Una imagen es simular a un snapshot de una máquina virtual, con la principal ventaja de ser mucho más ligero, conteniendo así cualquier aplicación con todas las dependencias necesarias para su ejecución. Las imágenes son utilizadas para crear contenedores, por lo tanto, cuando ejecutemos nuestro contenedor por primera vez hará un "pulling" de una imagen que contendrá la aplicación y sus dependencias deseadas.
- **Contenedores:** Un contenedor es la ejecución de una imagen, ejecutando nuestra aplicación.

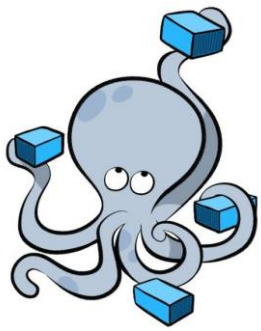


Ilustración 14 Logo de Docker-compose

Sin embargo, es necesario instalar la herramienta Docker-compose para poder hacer un pulling de aplicaciones compuestas por varios contenedores relacionados entre sí. En nuestro caso será necesario ya que los contenedores de Keyrock y MySQL están relacionados, así como los de Orion Context Broker y MongoDB.

Debido a la importancia de esta tecnología de cara al correcto despliegue del proyecto, se detallará la instalación en el Anexo A.

3.3 Sistemas de Gestión de Base de Datos

Todo Sistema "Powered by Fiware" será cursado por una elevada cantidad de información, la cual es modificada cada segundo y debe ser alamacenada. Por esa razón son empleadas hasta tres sistemas de gestión de bases de datos diferentes en el proyecto, los cuales se describen a continuación.

3.3.1 PostgreSQL

Uno de los aspectos más importantes de nuestro sistema es el almacenamiento de toda la información producida por cada vehículo, con la finalidad de poder ser consultada por su usuario una vez el PDP emita el acceso correspondiente.

Ilustración 15 Logo PostgreSQL

Para esta finalidad se ha optado por el uso del sistema de bases de datos relacional PostgreSQL, de código abierto, el cual hace uso del lenguaje de consulta estructurada (SQL de sus siglas en inglés), siendo un proyecto con más de 30 años de desarrollo activo.

No es estudio de la presente memoria profundizar en el funcionamiento de PostgreSQL o del funcionamiento de las bases de datos relacionales, pero a continuación se enumeran las principales características de PostgreSQL para su uso en el proyecto:

Tipos de datos:

- Primitivos: Integer, Numeric, String, Boolean.
- Estructurados: Date/Time, Array, Range, UUID.
- Documentos: JSON/JSONB, XML, Key-value (Hstore).
- Geométricos: Point, Line, Circle, Polygon.
- Personalizados: Composite, Custom Types.

Integridad de los datos:

- UNIQUE, NOT NULL
- Primary Keys
- Foreign Keys
- Explicit Locks, Advisory Locks
- Exclusion Constraints

Concurrencia

- Transacciones.

Seguridad

- Autenticación: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificado.
- Control de acceso robusto.
- Seguridad a nivel de filas y columnas.
- Autenticación: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificado.

Extensibilidad

- Procedimientos y funciones almacenadas.
- Posibilidad de conectar a otra base o flujo de datos con una interfaz SQL estándar.
- Lenguajes procedurales: PL/PGSQL, Python.

Todas estas características pueden ser consultadas y profundizadas mediante documentación exhaustiva en la página oficial de PostgreSQL.

3.3.2 MySQL

El sistema de bases de datos relacional MySQL es el encargado de almacenar toda la información referente al PDP Keyrock idm, por lo que es de gran importancia a la hora de poder autenticar al usuario en el sistema.

Ilustración 16 Logo MySQL

MySQL fue desarrollado en primera instancia por MySQL AB, empresa adquirida por Sun Microsystem en 2008. En la actualidad el soporte de MySQL depende de la empresa Oracle Coporation, la cual adquirió los derechos en el año 2010.

3.3.3 MongoDB

El último sistema de bases de datos usado es MongoDB, el cual presenta notables diferencias con los sistemas de base de datos detallados anteriormente.

Ilustración 17 Logo MongoDB

MonDB es un sistema de bases de datos NoSQL, de código abierto y orientado a documentos. La principal diferencia reside en cómo se almacena la información en la base de datos, siendo ésta almacenada como estructuras de datos BSON (formato similar a JSON), logrando una mayor facilidad y rapidez a la hora del acceso a la información, frente al almacenamiento en tablas relacionadas del modelo relacional.

Debido a la gran cantidad de información a procesar, Orion Context Broker almacena toda la información referente a las entidades en una base de datos MongoDB.

3.4 Spring Framework

Spring Framework proporciona un modelo integral de programación y configuración para aplicaciones basadas en Java. En el presente proyecto se ha optado por el uso del framework Spring para la creación de los dos servicios web Restful diseñados por la facilidad en la creación de los mismos.

Ilustración 18 Logo Spirng Framework

Se hace indispensable el uso de la herramienta Spring Boot, la cual facilita en gran medida la construcción y ejecución de nuestros servicios web.

El entorno Spring Tool Suite 4 integra el framework Spring y la herramienta Spring Boot para facilitar la creación de los servicios.

3.4.1 Hibernate

Hibernate permitirá la comunicación entre nuestro servicio web y la base de datos PostgreSQL, la cual debe ser modificada ante cambios en la información de contexto. Hibernate es una herramienta de mapeo objeto-relacional (ORM en su sigla del inglés), permitiendo abstraer la comunicación con una base de datos relacional.

Hibernste facilita el mapeo de atributos en una base de datos mediante clases en Java, permitiendo asignar atributos de una relación de nuestra base de datos con un atributo de la clase correspondiente, logrando así una traducción automática al lenguaje relacional que se emplee (SQL en el caso de PostgreSQL).

Ilustración 19 Logo Hibernate

Hibernate puede ser usado en nuestro entorno Spring Tool Suite 4 gracias a un set denominado Hibernate Tools, permitiendo la creación de ficheros de configuración para establecer la comunicación con la base de datos, así como las clases necesarias para mapear las relaciones de nuestra base de datos. La instalación y creación de estos elementos se explicará con más detalle en el Anexo C.

3.4.2 Glassfish Jersey

Glassfish Jersey, actualmente Eclipse Jersey es un framework de código abierto para el desarrollo de servicios web RESTful. Este framework será de gran importancia en el desarrollo de los tres servicios introducidos en el capítulo introductorio de la memoria, permitiendo realizar las distintas peticiones HTTP necesarias a Keyrock, Orion Context Broker o al IoT Agent Ultralight.

Los principales componentes del framework son:

- Core Server: Construir servicios RESTful basados en anotaciones (jersey-core, jersey-server, jsr311-api)
- Core Client: Ayuda en la comunicación con servicios REST (jersey-client)
- Soporte JAXB
- Soporte JSON
- Módulos de integración para Spring

En el proyecto han sido necesarios los componentes de soporte para JSON y el componente *Core Client*, el cual permitirá la comunicación con las API de los componentes FIWARE.

3.5 Desarrollo Web

Una parte importante del presente proyecto ha sido la creación de la aplicación web, la cual presenta la insterfaz de usuario para acceder al sistema.

3.5.1 Front-End

En el diseño del lado del cliente se ha optado por el uso de las siguientes tecnologías:

- HyperText Markup Language (HTML): Lenguaje de marcado para la elaboración de páginas web.
- Cascading Style Sheets (CSS): Lenguaje de diseño gráfico con el propósito de aportar una presentación customizada a nuestra página web.
- JavaScript: Lenguaje de programación interpretado, el cual aporta mejoras a la interfaz de usuario para una mayor comodidad del cliente.

3.5.2 Back-End

Las tecnologías usadas en la parte Back-End de la aplicación web se indican a continuación:

Servlet: Clase Java utilizada en el lado del servidor para extender las capacidades del mismo ante distintas peticiones de los clientes.

Ilustración 20 Java Servlet

JavaServer Pages (JSP): Tecnología para la creación de páginas web dinámicas basadas en HTML y XML, usando el lenguaje de programación Java.

4 PLATAFORMA DE CÓDIGO ABIERTO FIWARE

El mayor riesgo es no correr ningún riesgo.

- Mark Zuckerberg -

En este capítulo se detallarán todos los componentes FIWARE Generic Enablers, los cuales fueron introducidos en el capítulo primero representados en la arquitectura del sistema mediante figuras rectangulares. Estos componentes permiten llevar a cabo la tarea de securización del sistema y el procesamiento de la información de contexto producida por los sensores. La instalación y despliegue se aborda en el Anexo A.

La plataforma FIWARE es extensa y pone a disposición de los desarrolladores un catálogo extenso según las necesidades de la solución inteligente a desarrollar, por lo que el primer paso será profundizar en qué es la plataforma FIWARE y cuáles son sus ventajas.

4.1 Qué es la plataforma FIWARE

La plataforma FIWARE [3] es una iniciativa de código abierto impulsada por la Unión Europea, desarrollada en el año 2011 y contando con el respaldo de un gran número de empresas como Telefónica o Atos.

La finalidad es la creación de un marco común para el desarrollo de soluciones inteligentes, conformándose por una serie de componentes de plataforma, los cuales permiten el ensamblamiento con componentes de terceros para acelerar, facilitar y abaratar la creación de nuevas soluciones basadas principalmente en el Internet de las Cosas.

Los componentes de plataforma son detallados en el catálogo de FIWARE y se denominan FIWARE Generic Enablers. Todo componente de plataforma presenta una Interfaz de Programación de Aplicaciones (Application Program Interface, API) pública. Así mismo se pueden dividir los componentes en función de la tarea concreta que desempeñen según lo mostrado en la ilustración 21.

Ilustración 21 Bloques de la Plataforma FIWARE

El factor más importante de cualquier solución IoT es el procesamiento y adminitración de la gran cantidad de información de contexto generada, la cual es de gran interés para el correcto funcionamiento del sistema. Por ello, el único componente obligatorio de cualquier solución "Powered by FIWARE" es el Context Broker (su implementación dentro de la plataforma FIWARE es llamada Orion Context Broker).

Debido a la propia definición de una solución inteligente, se pueden abstraer una seríe de bloques comunes de la gran mayoría de soluciones que se enumeran a continuación:

- Procesamiento, análisis y visualización de contexto.
- Gestión central (Context bróker).
- Interfaces al Internet de las Cosas.
- Despliegue de herramientas.
- Control de acceso a datos.

Estos bloques son el conjunto de componentes de la plataforma FIWARE, los cuales se describirán en profundidad en la siguiente sección.

4.2 Gestión de contexto

La gestión de contexto es la parte fundamental de cualquier sistema inteligente, por lo que debemos realizar una gran variedad de tareas para el correcto funcionamiento del mismo, desde aspectos fundamentales en la gestión de la información de contexto, hasta el historial de contexto o los datos vinculados. Todas estas tareas pueden ser realizadas gracias a distintos FIWARE Generic Enablers, siendo el componente Orion Context Broker el núcleo de cualquier solución FIWARE y eje central del presente proyecto.

4.2.1 Orion Context Broker

Orion Context Broker [4] es el único componente obligatorio de cualquier solución "Powered by FIWARE", siendo el núcleo de cualquier solución inteligente debido a las funciones que desempeña: administrar, consultar y actualizar la información de contexto, siendo ésta el recurso fundamental de cualquier solución.

Para su funcionamiento se proporciona una API FIWARE NGSIv2, siendo ésta una API RESTful sencilla, la cual permite realizar un gran número de operaciones, donde se destaca:

- Creación de entidades.

- Actualizar la información de contexto.
- Consultar la información de contexto.

La API RESTful NGSIv2 se inspira y basa en la especificación OMA NGSI, quedando la información de contexto en FIWARE representada mediante estructuras de datos genéricos referidos como elementos de contexto según lo mostrado en la ilustración 22. Es importante puntualizar cómo es representada la información de contexto en OMA NGSI, representándose mediante estructuras de datos llamadas elementos de contextos, las cuales tienen asociado:

La información de contexto en OMA NGSI se representa a través de estructuras de datos llamadas elementos de contexto (ContextElements), los cuales tienen asociado:

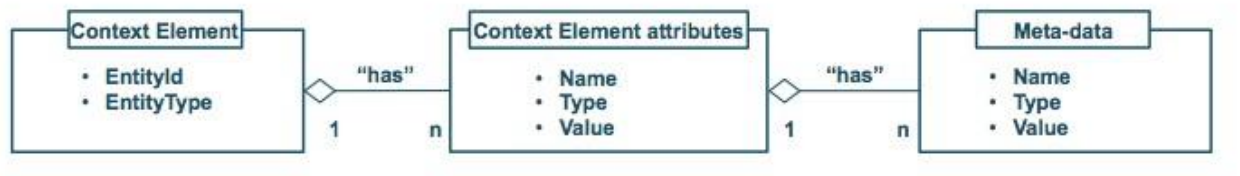


Ilustración 22 Elementos de Contexto Fuente

- Un EntityId y un EntityType único que identifica la entidad a la cual los datos de contexto hacen referencia.
- Una secuencia de uno o más atributos de elementos de datos.
- Metadatos opcionales vinculados a atributos.

Según la definición de la API FIWARE NGSI se define pues un modelo de datos para la información de contexto, usando entidades de contexto, las cuales contendrán toda la información generada y los atributos definidos según OMA NGSI.

Un vehículo inteligente quedará representado mediante un elemento de contexto o entidad, la cual es mostrada en la ilustración 23 y contendrá los siguientes atributos de contexto:

- Latitud: Indicará los grados Norte medidos por el sensor GPS.
- Longitud: Indicará los grados Este medidos por el sensor GPS.
- Velocidad: Indicará la velocidad actual del vehículo en kilómetros por hora.

```
{
    "id": "urn:ngsi-ld:Turismo:2233KAL",
    "type": "Turismo",
    "latitud": {
        "type": "Float",
        "value": 37.56975,
        "metadata": {
            "unidades": {
                "type": "Text",
                "value": "Grados Norte"
            }
        }
    },
    "longitud": {
        "type": "Float",
        "value": -6.100469,
        "metadata": {
            "unidades": {
                "type": "Text",
                "value": "Grados Este"
            }
        }
    },
    "velocidad": {
        "type": "Float",
        "value": 45,
        "metadata": {
            "unidades": {
                "type": "Text",
                "value": "Kilometros por hora"
            }
        }
    }
}
```


Ilustración 23 Entidad de Orion Context Broker

Es una buena práctica usar el formato de EntityId urn:ngsi-ld:Tipo:Valor, evitando así que dos entidades tengan el mismo identificador, siendo "Tipo" el tipo de vehículo que se esté considerando.

Es necesario un mecanismo de subscripciones [5] que permita la publicación de nueva información de contexto por los denominados productores de contexto, por ejemplo, los sensores y la posibilidad de consumir dicha información por los denominados consumidores de contexto, los cuales pueden ser aplicaciones externas interesadas en procesar dicha información. Este mecanismo aporta gran variedad de posibilidades, pudiendo ser productores y consumidores de contexto cualquier aplicación u otros componentes de la propia plataforma de FIWARE.

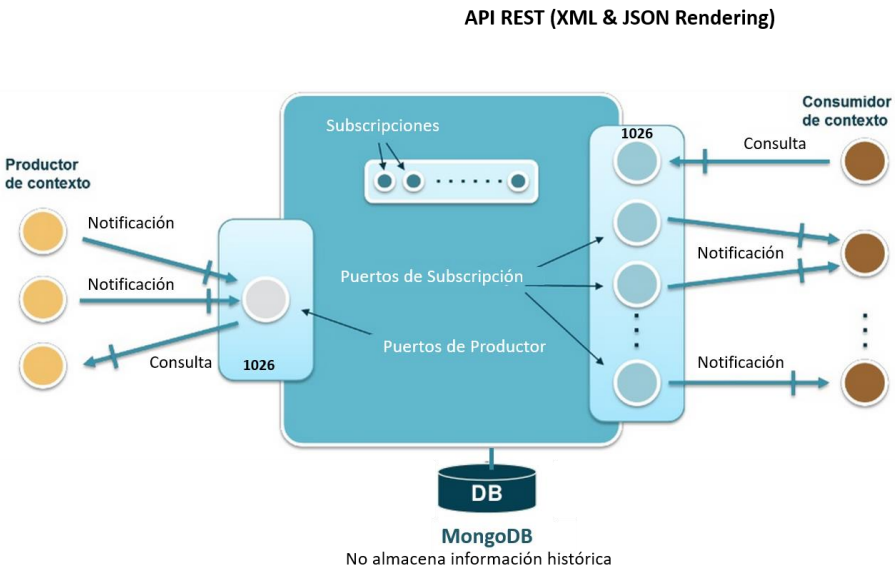


Ilustración 24 Mecanismo de Subscripciones de Orion Context Broker

El mecanismo de subscripciones, el cual se muestra en la ilustración 24, permitirá indicar un Localizador de Recursos Uniforme (Uniform Resource Locator, URL) a la hora de crear la subscripción para poder recibir la información de contexto en caso de cambio. Es por esta razón que se indicará la URL de un Endpoint de nuestro servicio web Sensorservice encargado de obtener un objeto Notification y poder obtener la entidad con los datos a almacenar en la base de datos.

4.3 IoT, Robots y sistemas de terceros

Una parte fundamental de cualquier solución inteligente es aquella encargada de producir la información de contexto, por lo tanto, se hace necesario una serie de componentes que faciliten la interfaz entre nuestro Context Broker Core y el Internet de las Cosas o sensores.

Esa función es lograda mediante los denominados agentes IOT. Dentro de la nomenclatura referente a agentes IOT encontramos el denominado Puerto Norte (North Port) y Puerto Sur (South Port). El puerto norte es usado para establecer la comunicación con el Context Broker, Orion Context Broker en el caso de una aplicación "Powered by FIWARE", además de ser usado para la configuración del propio agente IOT. En cambio, el puerto sur es usado para recibir los mensajes generados por los distintos sensores IOT generadores de la información de contexto.

En la actualidad se pueden encontrar una variedad de agentes IoT atendiendo a las necesidades de nuestra interfaz con el mundo del Internet de las Cosas, entre los que destacan los siguientes agentes IoT:

- **IoTAgent-JSON** – Puente de unión entre mensajes HTTP/MQTT (con cuerpo JSON) y NGSI.
- **IoTAgent-LWM2M** – Puente de unión entre el protocolo Lightweight M2M y NGSI.
- **IoTAgent-UL** – Puente de unión entre mensajes HTTP/MQTT (con cuerpo UltraLight2.0) y NGSI.
- **IoTagent-LoRaWAN** – Puente de unión entre el protocolo LoRaWAN y NGSI.

Se observa la necesidad de adaptar los protocolos usados por los elementos del mundo del Internet de las Cosas (sensores, generadores de información de contexto, etc.) a NGSI para la correcta comunicación con Orion Context Broker.

Las peticiones necesarias y la relación existente entre el componente Orion Context Broker y el Agente IoT Ultralight se abordarán en el capítulo 6.

4.4 Mecanismo de seguridad en FIWARE

El aspect principal del presente proyecto es la securización de todo el sistema gracias a determinados FIWARE Generic Enablers que facilitan la autenticación y verificación de usuarios dentro de un sistema.

Paso previo de la definición de los distintos niveles de seguridad mediante los componentes de la plataforma FIWARE, es esencial detallar los mismos, indicando sus funciones e importancia.

4.4.1 Keyrock Identity Manager

El primer aspecto a mencionar de Keyrock [6] es su función dentro del sistema, ejerciendo de Punto de Decisión de la Política (PDP) como se mencionó en capítulos anteriores, evaluando las peticiones de los usuarios frente a unas políticas de autorización establecidas, antes de emitir una decisión de acceso. Se debe puntualizar la función

de Keyrock según el nivel de seguridad de nuestro sistema, dejando de actuar como Punto de Decisión de la Política para un nivel avanzado de seguridad (se explicará con más detalle en la siguiente sección).

Un Gestor de Identidad es el eje central de cualquier arquitectura, debido a la existente necesidad de controlar el acceso de usuarios a redes, servicios o aplicaciones.

Keyrock ofrece una serie de herramientas para los administradores del sistema, reduciendo el esfuerzo en materia de creación y administración de usuarios. Todas estas opciones se presentan mediante una interfaz de usuario muy fácil de utilizar o mediante una API Restful [7], la cual es mostrada en la ilustración 25.

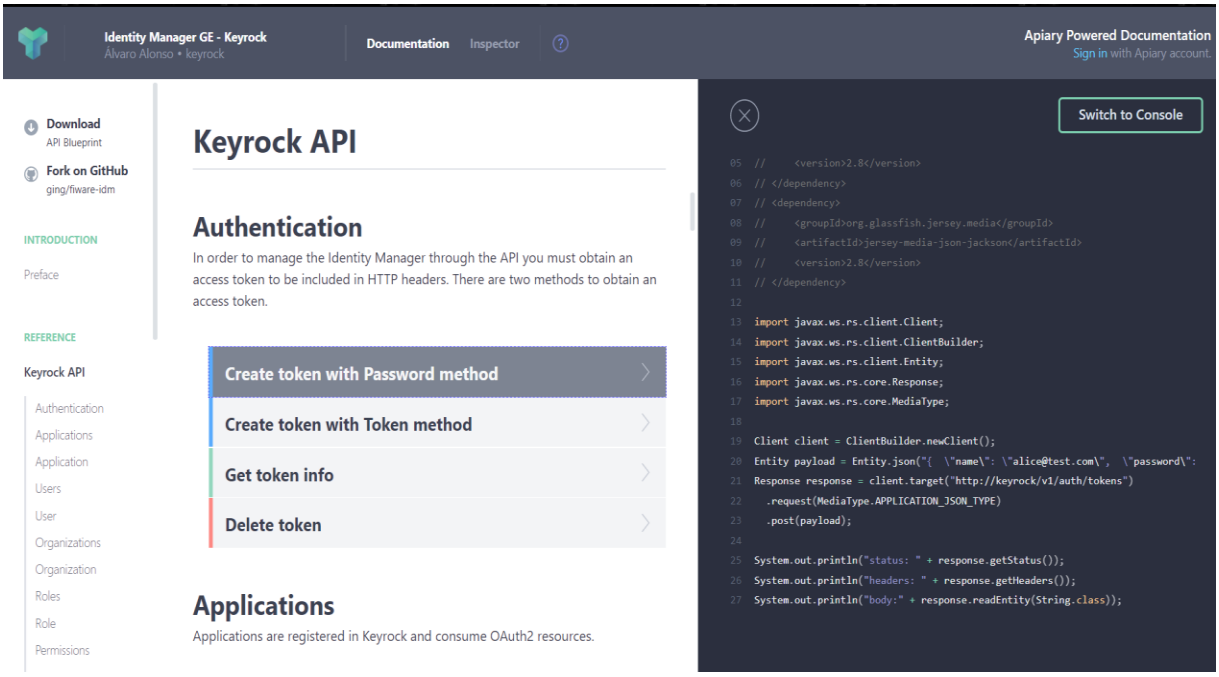


Ilustración 25 Keyrock API en Apiary

La API de usuario es similar a:

- Servicios web RESTful
- HTTP/1.1
- JSON y XML

Los usuarios pueden realizar las siguientes acciones mediante el uso de la API de Keyrock:

- Autenticación
- Gestionar aplicaciones
- Gestionar usuarios
- Gestionar organizaciones
- Gestionar roles
- Gestionar permisos
- Gestionar IoT Agents
- Gestionar PEP-Proxy

Keyrock cumple con el protocolo de autorización OAuth 2.0 (RFC 6749) para ofrecer mayor seguridad, soportando los cuatro tipos de flujos (grant types):

Código de Autorización: El código de autorización es obtenido mediante el uso de un servidor de autorización (el gestor de identidad) como intermediario entre el cliente (la aplicación registrada) y el propietario del recurso (usuario).

```
GET /oauth2/authorize?response_type=code&client_id=1&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcallback_url HTTP/1.1
Host: account.lab.fiware.org
```

Ilustración 26 Petición Http Código de Autorización

Implícita: Flujo implícito de autorización optimizado para los clientes implementandos en navegadores mediante el uso de un lenguaje como JavaScript.

Credenciales de contraseña del propietario del recurso: Flujo usado directamente como un flujo de autorización para la obtención de un token de acceso.

```
POST /oauth2/token HTTP/1.1
Host: account.lab.fiware.org
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=demo&password=123
```

Ilustración 27 Petición Htpp Credenciales de contraseña

Credenciales de cliente: El cliente puede pedir un token de acceso usando únicamente sus credenciales de cliente.

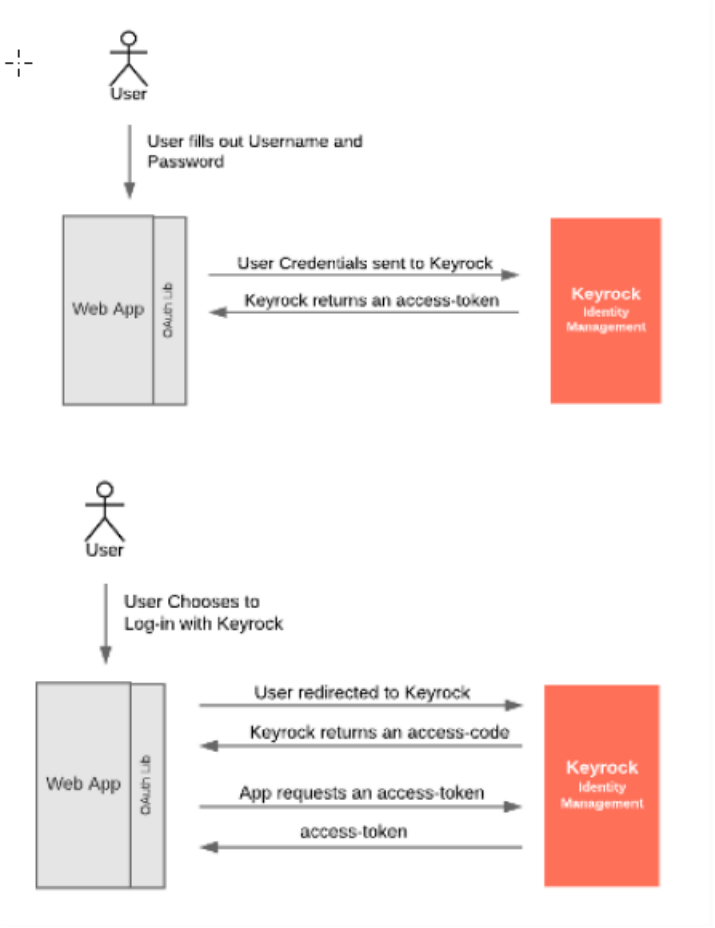


Ilustración 28 Flujos Oauth 2.0 Fuente: FIWARE

La autenticación de usuario es fundamental en el sistema, siendo la petición de token un proceso vital en la seguridad. La secuencia para petición de un token se detalla mediante un diagrama de secuencia en la ilustración 28.

La interfaz de gráfica de Keyrock permite gestionar qué flujos Oauth 2.0 serán aceptados en base a nuestra aplicación, los cuales se indican en la ilustración 29.

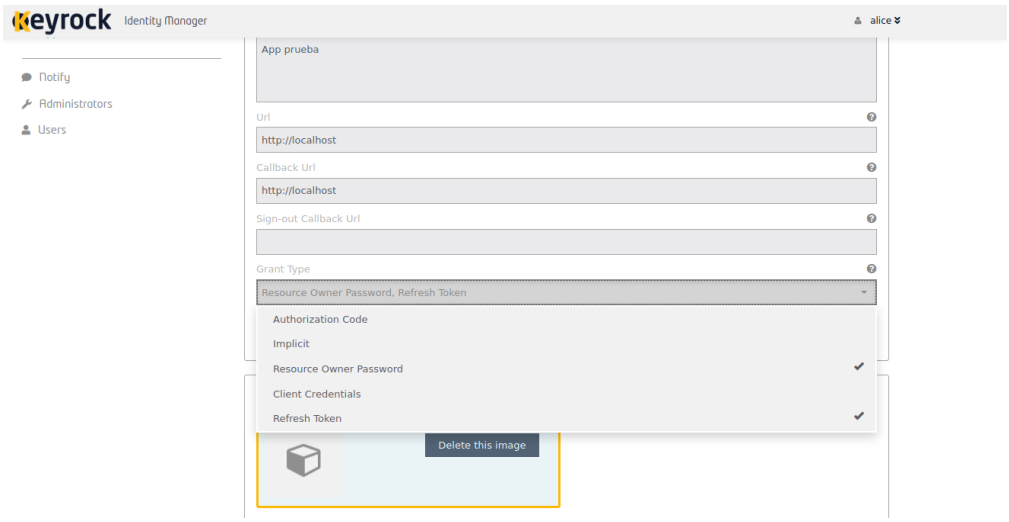


Ilustración 29 Gestión Flujos Admitidos Keyrock

Keyrock Identity Management GEI cumple con el estándar existente de autenticación de usuarios y provee información para funcionar como plataforma Single Sign-On. El uso de Keyrock idm tiene gran peso en la arquitectura, almacenando toda la información referente a los usuarios del sistema en una base de datos MySQL, por lo que todas sus posibilidades se detallarán en la siguiente sección, en la cual se hablarán de los distintos niveles de seguridad alcanzables y las características de la base de datos MySQL se detallarán en el Anexo A.

4.4.2 PEP-Proxy Wilma

El FIWARE Generic Enabler Wilma, en conjunto con nuestro Gesto de Identidad Keyrock fortalecerá el control de acceso a nuestras aplicaciones de Backend.

PEP-Proxy Wilma actuará como Punto de Aplicación de la Política (PEP en sus siglas del inglés), siguiendo el siguiente comportamiento:

1. Intercepta las peticiones de los usuarios a un recurso. Una buena medida es que nuestra aplicación escuche únicamente peticiones de Wilma.
2. Realiza una petición de decisión al Gestor de Identidad, el cual consultará los permisos de dicho usuario mediante su token y emitirá una decisión de acceso.
3. Wilma escuchará la decisión de acceso, permitiendo o denegando el acceso del usuario.

La gran importancia del uso del PEP-Proxy Wilma radica en la verificación, obteniendo así dos factores claves en un sistema seguro, la autenticación y la verificación.

Wilma carece de interfaz gráfica, por lo que sus notificaciones deben ser analizadas en una terminal o bien redirigir la salida a un fichero de logs.

4.5 Niveles de Seguridad

Dentro del marco de la seguridad en sistemas informáticos se definen tres niveles de control de acceso de Punto de Decisión de la Polítca (PDP):

Nivel 1: Acceso mediante autenticación - Se permiten todas las acciones a cualquier usuario autenticado y ninguna acción a los usuarios anónimos.

Nivel 2: Autorización básica - Se comprueban los permisos de un usuario autenticado para acceder a un determinado recurso.

Nivel 3: Autorización avanzada - Control especializado mediante XACML.

Debido a la necesidad de detallar todos los niveles de seguridad alcanzables se mostrarán ilustraciones referentes a tutoriales detallados de FIWARE [5] que cubren cada nivel de seguridad de forma individual. El proyecto desarrollado implementa un nivel 2 de seguridad dentro del marco de FIWARE.

4.5.1 Nivel 1: Acceso mediante autenticación

El primer nivel de seguridad es alcanzable mediante el uso de un componente que permita la autenticación de usuarios dentro de nuestro sistema, es decir mediante el uso de un Gestor de Identidad como Keyrock.

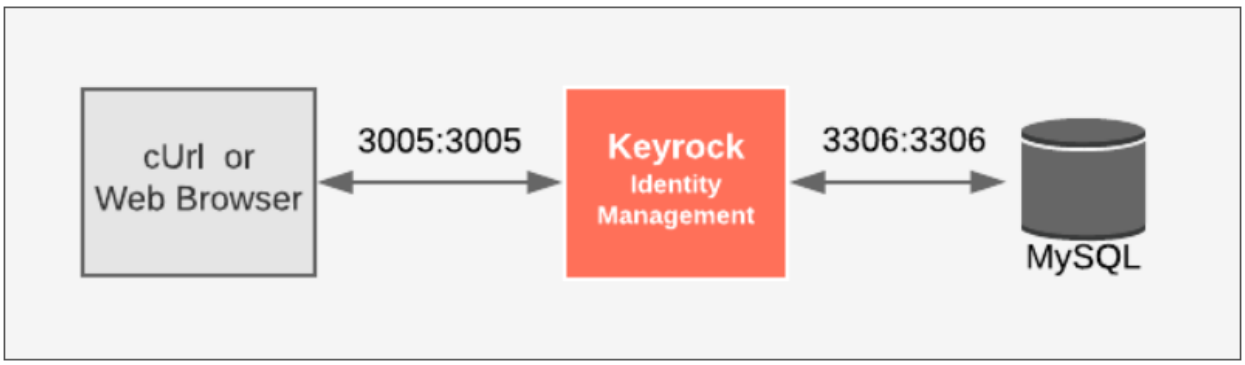


Ilustración 30 Nivel de Seguridad Básico

El uso del nivel de acceso mediante autenticación, el cual se muestra en la ilustración 30 implicaría la existencia de distintos usuarios en el Sistema, los cuales serán asociados a distintas aplicaciones para su gestion, permitiendo la obtención de tokens mediante los flujos Oauth 2.0 permitidos.

El requisito fundamental para establecer el nivel de acceso mediante autenticación es la creación de un mínimo de un usuario, el cual deberá asociarse a un mínimo de una aplicación. En este nivel de seguridad, Keyrock actuará como Punto de Decisión de la Política (PDP) sin necesidad de otros componentes de la plataforma FIWARE.

4.5.2 Nivel 2: Autorización básica

El nivel de autorización básica es alcanzable mediante mecanismos que permitan la verificación dentro de nuestro sistema, es decir, permitir discernir si un usuario que previamente ha sido autorizado tiene los privilegios requeridos para acceder a cierto recurso.

La verificación es alcanzable mediante la creación de permisos en Keyrock. El modo de procedimiento es el siguiente:

- Creación de organizaciones, permitiendo la agrupación de usuarios en distintas organizaciones lógicas.
- Creación de roles a asignar a las organizaciones previamente creadas.
- Creación de permisos a asignar a los distintos roles creados.

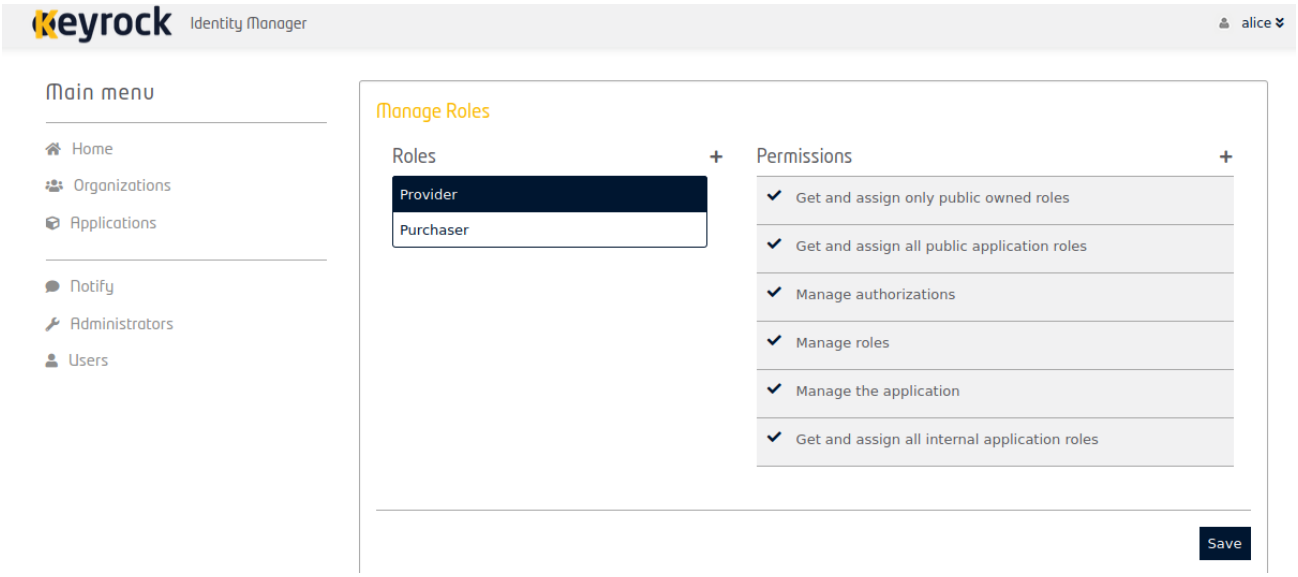


Ilustración 31 Administrando Roles y Permisos en Keyrock

Un permiso indica un recuro mediante una URL para permitir el acceso de un usuario a dicho recuros. En la ilustración 31 se pueden observar permisos creados por defecto dentro de Keyrock.

Para crear un permiso debemos especificar los siguientes aspectos:

- Nombre del permiso.
- Descripción.
- Método HTTP de la petición al recurso.
- URL del recurso.

El nivel 2 PDP puede ser solo usado con una instancia propia de Keyrock, haciendo uso de un flujo de Contraseña, práctica habitual que ha sido seguida en el presente proyecto.

El nivel básico de seguridad puede ser también implementado mediante el uso de un PEP-Proxy, el cual aportará la verificación requerida en nuestro sistema. Una vez obtengamos un token mediante los flujos Oauth 2.0 permitidos, se deberá realizar la petición correspondiente a PEP-Proxy Wilma para solicitar acceso al recurso.

```
X-Nick-Name: nickname of the user in IdM
X-Display-Name: display name of user in IdM
X-Roles: roles of the user in IdM
X-Organizations: organizations in IdM
```

Ilustración 32 Información Enviada por PEP-Wilma al Recurso Solicitado

Si el usuario tiene los permisos adecuados será permitida su petición, enviándose una petición HTTP al recurso solicitado con su user_id, rol y organización verificando así a dicho usuario, cuya petición queda representada mediante la ilustración 32.

4.5.2.1 Autorización Básica en el marco del Proyecto

El nivel de seguridad que se detallará en los capítulos 5 y 6 corresponden a un nivel de seguridad de Autorización Básica. Los requisitos que cumple el proyecto para poder ser considerado un sistema con un nivel de seguridad 2 son los siguientes:

- El Gestor de Identidad Keyrock ejerce de Punto de Decisión de la Política (PDP) al hacer uso del componente Authzforce que permite alcanzar un nivel de Autorización Avanzada.
- Se crean permisos por roles dentro de la organización administrativa del proyecto para poder limitar los recursos a los que se tienen acceso.
- Se alcanza la verificación de usuarios mediante PEP-Proxy Wilma, analizando el bearer token de un usuario y consultando sus credenciales con Keyrock, el cual ejercerá la decisión de acceso.

Un punto importante a mencionar es la limitación en el uso de permisos en el proyecto, ya que no es escalable crear permisos exclusivos por cliente, limitando el acceso al único recurso referente a su matrícula. Por ello, se realizará la gestión en el acceso a los recursos referentes a las matrículas en el Servicio Web Sensorservice, una vez el usuario ha sido verificado.

4.5.3 Nivel 3: Autorización Avanzada

La autorización avanzada da un paso firme mediante la introducción de un nuevo componente denominado Authzforce. La inclusión de Authzforce supone el cambio de funcionalidad de algunos componentes, ya que FIWARE Authzforce ejercerá como Punto de Decisíon de la Política (PDP) en lugar de Keyrock como se detalló en los niveles de seguridad anteriores.

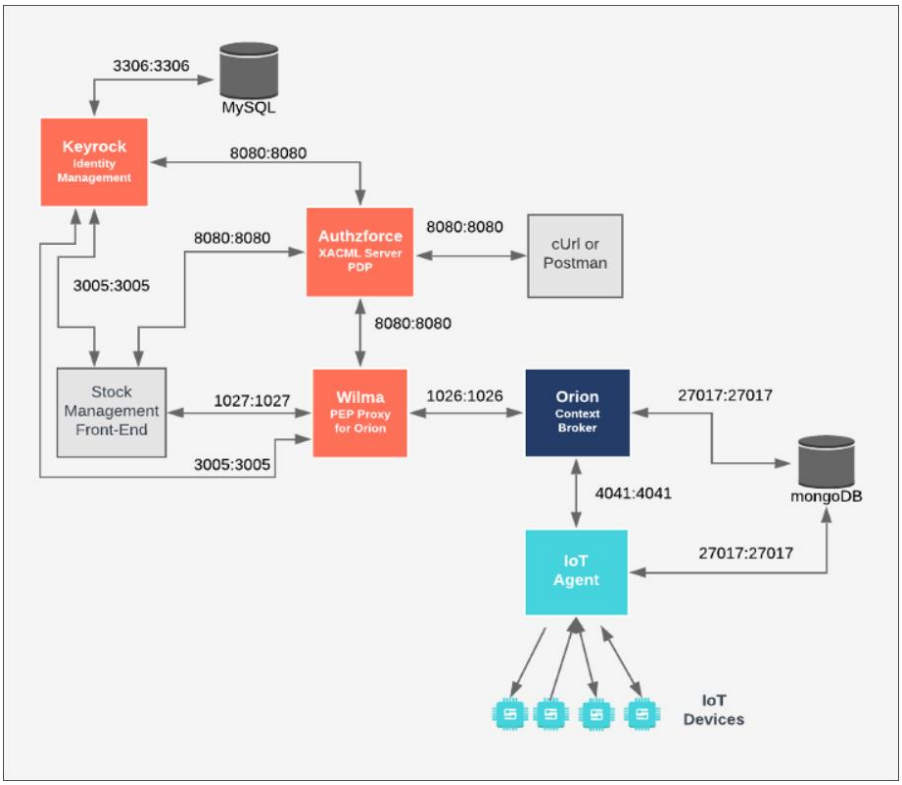


Ilustración 33 Autorización Avanzada

Un ejemplo de sistema con un nivel de seguridad avanzado sería el mostrado en la ilustración 33 procedente de un sistema de Stock de un tutorial oficial de FIWARE [8], observándose el uso de los componentes Keyrock, PEP-Proxy Wilma y Authzforce.

Este último Generic Enabler de control de acceso Avanzado es capaz de interpretar reglas mediante el uso del estándar XACML, el cual es un estándar OASIS que describe el formato de las políticas de autorización y lógica de evaluación. También se definen el formato de la solicitud o respuesta de la decisión de autorización, recogido en un archivo en formato XML.

Ilustración 34 Ejemplo XACML

En la ilustración 34 se observa la creación de distintas reglas en un fichero con formato XML.

Se llega a una nueva arquitectura, en la cual Keyrock deja su papel central de Punto de Administración de la Política, delegando su función en Authzforce, comunicándose éste con PEP-Proxy Wilma, cuyo componente realiza las mismas funciones descritas en la sección anterior. Keyrock podrá ser usado como Punto de Administración de la Política (PAP), diriengo así las políticas de autorizaciónde acceso.

5 SOLUCIÓN DESARROLLADA: SERVICIOS WEB Y BASE DE DATOS

Pregunta lo que no sepas, y podrás pasar por tonto unos minutos. No lo preguntes, y serás tonto la vida entera.

- Confucio -

En este capítulo se detalla los servicios web RESTful Keyrockservice encargado de la comunicación directa con el *Identity Manager Keyrock* y Sensorservice encargado de las subscripciones con Orion Context Broker y el almacenamiento de la información de contexto generada para cada vehículo en la base de datos mediante PostgreSQL, los cuales fueron introducidos en la arquitectura del sistema mostrada en el capítulo I.

Una vez se han introducido todos los componentes de la plataforma FIWARE en el capítulo anterior, es necesario detallar al lector el diseño del sistema completo, siendo éste el primer paso para entender la relación existente entre todos los componentes y la tarea que desempeñan. De esta forma, se conseguirá crear una visión completa del sistema, la cual facilitará en gran medida el entendimiento de todos los componentes que se detallará de forma individual en los capítulos de la *Solución Desarrollada*.

5.1 Diseño del Sistema

El aspecto más importante de cualquier sistema desarrollado es su diseño, evitando innumerables problemas en el futuro. Por ello, es de gran importancia mostrar de forma superficial al lector el diseño del sistema, permitiendo un mejor entendimiento de sus componentes y las relaciones existentes entre ellos.

La realización y estudio de un diseño completo consta de muchas partes (Análisis de Requisitos, Diseño de Pruebas, Diseño del Sistema, etc.) y queda fuera del marco del Proyecto, pero se intenta abordar de forma concreta la fase de Diseño del Sistema mediante un Lenguaje Unificado de Modelado (Unified Modeling Language, UML) [9].

Concretamente se hará uso de dos tipos de diagramas UML:

- **Diagramas de Casos de Uso**: Es una forma de diagrama de comportamiento UML mejorado, representando los distintos componentes del sistema (subsistemas) y la relación entre los distintos casos de uso.

- **Diagramas de Flujo**: Un diagrama de flujo representa distintas acciones o pasos en un proceso, con la finalidad de detallar un caso de uso.

5.1.1 Diagrama de Casos de Uso

El primer paso para un correcto entendimiento del sistema es entender los actores que toman parte en cualquier diagrama de casos de uso que conforme el diseño de un sistema. Para el sistema detallado se han creado dos actores diferenciados:

- **Usuario**: El usuario representa un usuario básico sin privilegios dentro del sistema, el cual solo podrá acceder a su información y modificar sus datos personales.
- **Administrador**. El administrador es el superusuario del sistema, el cual tiene privilegios para modificar los usuarios existentes, dar de alta o de baja un usuario y consultar la información de todos los usuarios de acuerdo a la *Ley de Protección de Datos*.

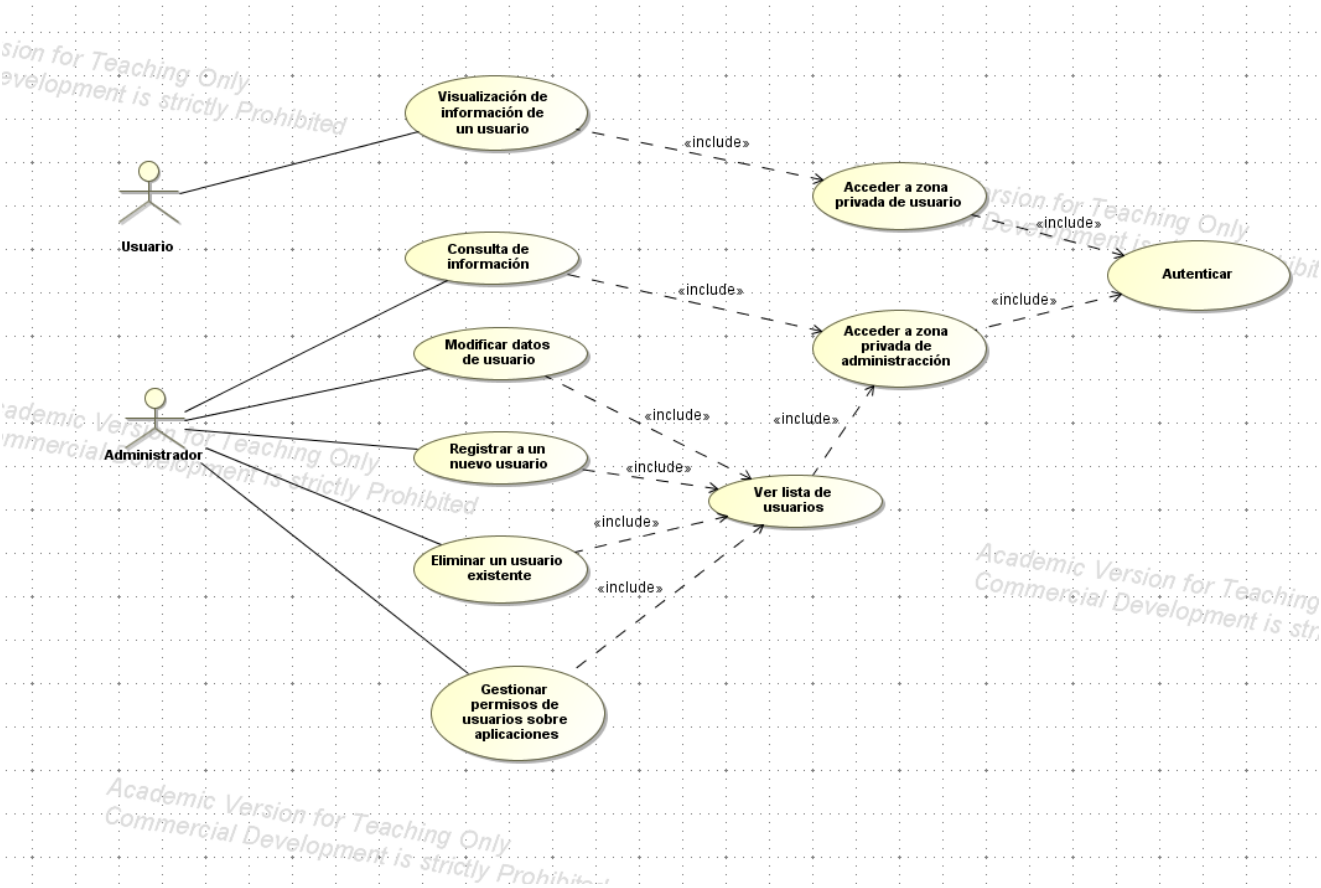


Ilustración 35 Diagrama de Casos de Uso

Los actores del sistema pueden realizar los distintos casos de uso mostrados en la ilustración 35, los cuales se detallan a continuación:

Usuario:

En la tabla 1 se detalla el caso de uso para la visualización de la información personal de un usuario.

ID:CU-001	*Visualización de información de un usuario*		
Versión	*1.0*		
Dependencias	• *Uso de API propietaria*		
Precondición	*Un usuario desea acceder a su información personal mediante sus credenciales de usuario.*		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso *abstracto* cuando *se desee hacer uso del sistema.*		
Secuencia Normal	**Paso**	**Acción**	
	1	*El usuario introduce el usuario y contraseña*	
	2	Se realiza una petición al servicio Keyrockservice para la comunicación con Keyrock idm.	
	3	Se envía a Keyrock las credenciales y se comprueba el usuario.	
	3	Si el usuario y contraseña son correctos	
		3.1	*Se permite el acceso y se pasa a la interfaz principal de usuario.*
		3.2	Si no lo son, se vuelve al paso 1.
	4	Se selecciona la opción de consultar información dentro del menú lateral de la interfaz de usuario.	
	5	Se realiza una petición a PEP-Proxy Wilma con la matrícula de usuario y su bearer token.	
	6	Si el token es correcto se redirije a Sensorservice y se comprueba la matrícula del usuario.	
	7	Si la matrícula es correcta se devuelve la información de usuario, la cual se muestra en la interfaz de usuario.	
Postcondición			
Excepciones	**Paso**	**Acción**	
	1	Si *el acceso es denegado*	
		1.1	*Se permite volver a introducir usuario y contraseña. Si se detecta un intento masivo de solicitudes de un usuario se le expulsará del sistema.*
Frecuencia	*Cada vez que se desee cambiar la configuración del sistema.*		
Importancia	*Media*		
Prioridad	*Media*		
Estado	*Analizado.*		
Comentarios	*La comprobación de la matrícula de usuario se realiza en el servicio web Sensorservice, aunque es posible la creación de roles personalizados con permisos por usuarios en Keyrock (poco escalable).*		

Tabla 1 CU-001 Solicitar Información de un Usuario

Administrador:

El usuario administrador concentra el grueso de las funcionalidades del sistema, siendo el superusuario que gestiona el correcto funcionamiento del mismo, así como la gestión de usuarios.

Sin embargo, todas las opciones del sistema tienen el mismo impacto y modo de procedimiento, por lo que solo se indicará en la presente subsección un único caso de uso, siendo el resto simulares, pero realizando una petición HTTP distinta a Keyrock, en función de las opciones que se quiera desempeñar.

Las opciones para un usuario administrador serán las siguientes:

- Modificar datos de un usuario: Permite modificar los datos personales de un usuario a partir de sus credenciales de usuario. Esta información es almacenada en la base de datos dependientes del Gestor de Identidad Keyrock.
- Eliminar un usuario existente: Permite eliminar un usuario del sistema y su vehículo asociado. Es necesario eliminar toda la información referente al usuario almacenada por Keyrock y toda la información del vehículo del usuario almacenada en la base de datos gestionada por PostgreSQL.
- Crear un nuevo usuario: Permite dar de alata un usuario en Keyrock y crear un nuevo vehículo asociado a la matrícula del usuario, el cual ha sido dado de alata previamente.
- Consultar información: Permite consultar la información de cualquier usuario del sistema, así como los datos generados por los vehículos de los usuarios. Esta funcionalidad es similar a la expuesta para el usuario básico.

En la tabla 2 se detalla el caso de uso de eliminar un usuario existente, siendo éste similares en procedimiento a los casos de uso de modificación de usuario y crear un nuevo usuario.

ID:CU-002	*Eliminar un usuario existente*		
Versión	*1.0*		
Dependencias	• *Uso de API propietaria*		
Precondición	*El administrador desea eliminar un usuario del sistema.*		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso *abstracto* cuando *se desee hacer uso del sistema.*		
Secuencia Normal	**Paso**	**Acción**	
	1	*El administrador introduce el usuario y contraseña*	
	2	Se realiza una petición al servicio Keyrockservice para la comunicación con Keyrock idm.	
	3	Se envía a Keyrock las credenciales y se comprueba si el usuario es administrador.	
	3	Si el usuario y contraseña son correctos, y el usuario es un usuario administrador.	
		3.1	*Se permite el acceso y se pasa a la interfaz principal de administrador.*
		3.2	Si no lo son, se vuelve al paso 1.
	4	Se selecciona la opción de eliminación de usuario.	
	5	Se rellena el formulario de baja de usuario con la matrícula a eliminar y el identificador de usuario.	
	6	Se elimina la información del usuario con identificador indicado de la base de datos MySQL dependiente de Keyrock.	
	7	Se realiza una petición a PEP-Proxy Wilma al endopoint correspondiente de eliminación de matrícula.	
	8	El servicio Sensorservice hará uso de Hibernate para poder eliminar la información referente a la matrícula de la base de datos gestionada por PostgreSQL.	
Postcondición			
Excepciones	**Paso**	**Acción**	
	1	Si *el acceso es denegado*	
		1.1	*Se permite volver a introducir usuario y contraseña. Si se detecta un intento masivo de solicitudes de un usuario se le expulsará del sistema.*
Frecuencia	*Cada vez que se desee cambiar la configuración del sistema.*		
Importancia	*Alta*		
Prioridad	*Alta*		
Estado	*Analizado.*		

Comentarios	*Es importante hacer comprobación previa de si el usuario existe en la base de datos del sistema, por si se da el supuesto de que el usuario existierá en Keyrock, pero no exista en la base de datos gestionada por PostgreSQL.*

Tabla 2 CU-002 Eliminar un usuario existente

5.1.2 Diagrama de Flujo

Por cada caso de uso se debe representar un diagrama de flujo que detalle los pasos a seguir para poder completar dicho caso de uso. En la subsección correspondiente se pretende mostrar algunos diagramas de flujo que clarifiquen el funcionamiento del sistema, pero no se mostrarán todos los casos de uso detallados en la subsección 5.1.1.

Concretamente se detallarán los casos de uso de "Consulta de información" y "Eliminar un usuario existente". Se debe mencionar la gran similitud existente en los casos de uso correspondientes a opciones de la interfaz desarrollada, teniendo elementos comunes como la autenticación y comunicación con los servicios web Keyrockservice y Sensorservice.

Diagrama de Flujo Consulta de Información

Los pasos necesarios para que el usuario pueda consultar información del sistema referente a los vehículos dados de alta son los siguientes:

- Introducir las credenciales de usuario en el formulario de inicio de sesión.
- Envio de credenciales de usuario al servicio Keyrockservice, el cual consultará mediante el intento de generación de token a través de contraseña si el usuario existe o no.
- Si el usuario existe se debe comprobar si es administrador o no mediante la información referente a su token y si no existe se deniega el acceso al sistema.
- Si el usuario es administrador se redirige a la página de la interfaz de usuario administrador y se debe de cumplimentar el formulario de información al pulsar sobre la opción correspondiente en el menú lateral.
- Se envía una petición Http a PEP-Proxy Wilma para verificar el bearer token de usuario, y en caso de estar autorizado se da acceso al servicio Sensorapp.
- El servicio Sensorapp consulta la información seleccionada en el formulario previo y se devuelve en la respuesta.

En la ilustración 36 se muestra el diagrama de flujo correspondiente al caso de uso "Consulta de Información".

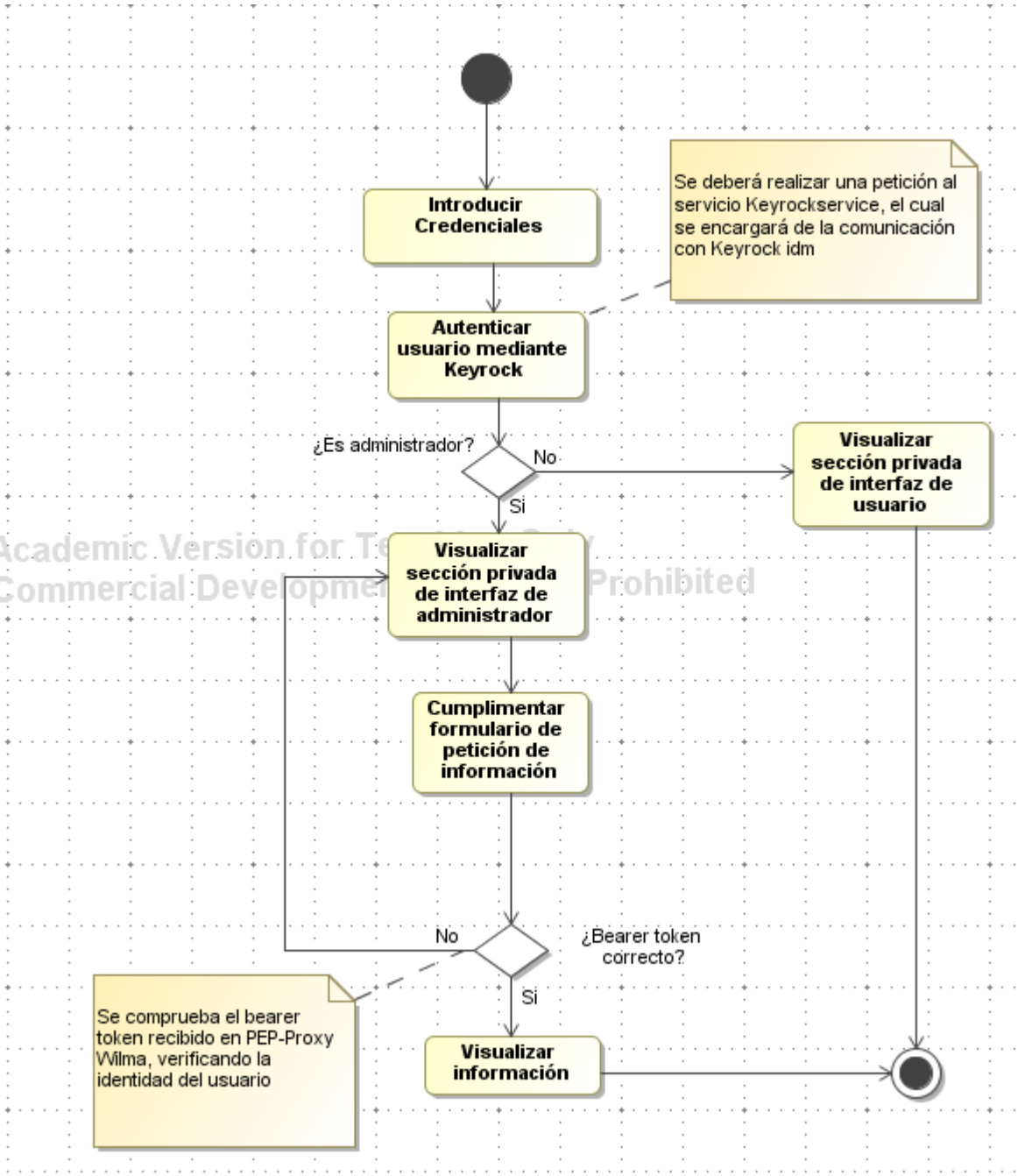


Ilustración 36 Diagrama de flujo Consultar Información

Diagrama de Flujo Eliminar un usuario existente

Los pasos necesarios para que el administrador pueda dar de baja un usuario existente serán los siguientes:

- Introducir las credenciales de usuario en el formulario de inicio de sesión.
- Envio de credenciales de usuario al servicio Keyrockservice, el cual consultará mediante el intento de generación de token a través de contraseña si el usuario existe o no.
- Si el usuario existe se debe comprobar si es administrador o no mediante la información referente a su token y si no existe se deniega el acceso al sistema.
- Si el usuario es administrador se redirige a la página de la interfaz de usuario administrador y se debe de cumplimentar el formulario debaja de usuario, el cual se muestra al pulsar la opción "Eliminar Usuario".
- Si el usuario existe en Keyrock se elimina de la base de datos MySQL y se realiza una petición a PEP-Proxy Wilma al endpoint correspondiente a la baja de usuarios.
- Si existe el usuario en la base de datos gestionada por PostgreSQL se da de baja al usuario.

El flujo a seguir para el caso de uso "Eliminar un usuario existente" se muestra en la ilustración 37.

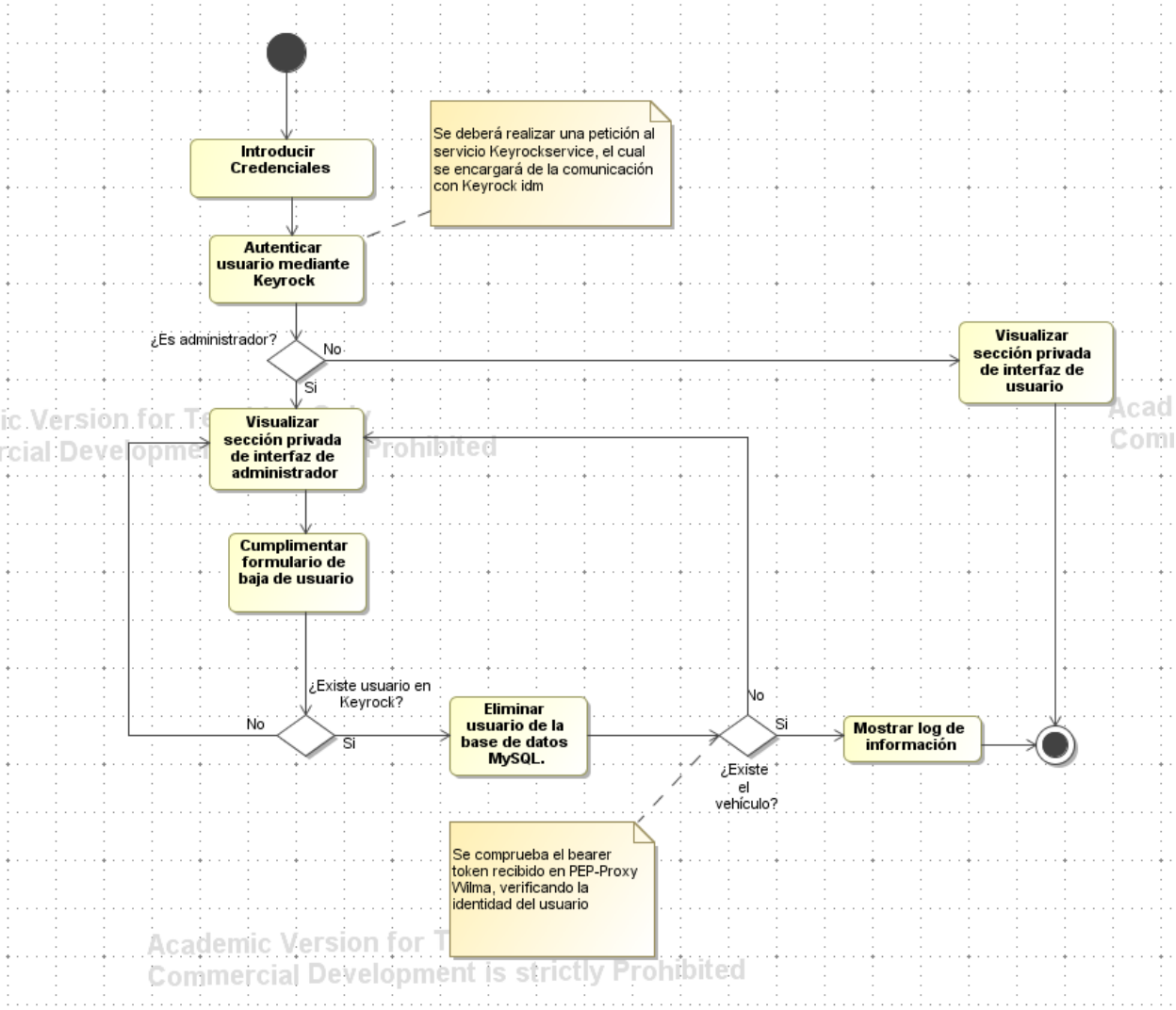


Ilustración 37 Eliminar un usuario existente

5.2 Servicio Web RESTful: Keyrockservice

Keyrockservice es un servicio web RESTful creado mediante el framework Spring en el entorno de desarrollo Spring Tool Suit 4, encargado de realizar todas las comunicaciones con Identity Manager Keyrock mediante la API de Keyrock.

En esta sección se abordarán las distintas clases que forman el servicio y la API RESTful que se facilita para la comunicación con el mismo.

5.2.1 Clases

Una tarea importante en un servicio web es la separación de responsabilidades, creando distintos niveles para realizar tareas concretas. Esta separación es necesaria y ha sido implementada tal y como se muestra en la ilustración 38 en los dos servicios web desarrollados.

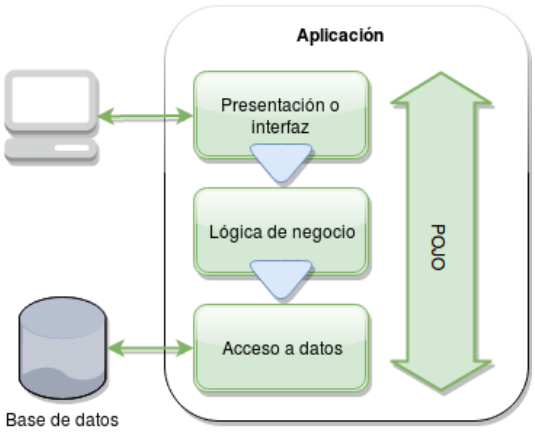


Ilustración 38 Separación de responsabilidades

En el servicio web Keyrockservice no se realizará la presentación final de la información, por lo que los niveles usados serán el de lógica de negocio, acceso a datos y POJO.

Una vez se accede a la información almacenada en la base de datos gestionada por PostgreSQL es necesario objetos que transporten esta información entre las distintas capas definidas. Estos objetos son los llamados Data Transport Object (DTO) o Value Object (VO) que se implementan como objetos planos de java (Plain Old Java Object o POJO).

Un punto a destacar de la división mostrada en la ilustración 38 es la función que toman los servicios web, reduciéndose únicamente a la capa de lógica de negocio y la creación de la clase POJO. En cualquier situación la capa de acceso a datos será efectuada por el sistema gestor de base de datos y la presentación será realizada por la aplicación web.

Clases que controlan el funcionamiento de la aplicación

- **KeyrockserviceApplication.java**

Clase @SpringBootApplication que contiene el método main del programa. El método main será el primero en llamarse una vez se ha ejecutado la aplicación y mediante la etiqueta @SpringBootApplication (scanBasePackages = {"com.example"},exclude = {SecurityAutoConfiguration.class }) queda la aplicación inicializada.

- **TokenController.java**

Clase que ejerce de Control en la aplicación Spring desarrollada, encargada del nivel de lógica de negocio, ya que debe de hacer las peticiones a Keyrock (aplicación encargada del acceso a datos) y enviar la información a la capa de presentación.
La clase hará uso de la anotación Spring @Controller especificando que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. También es esencial el uso de las anotaciones @RequestMapping y @PostMapping para cualquier método, indicando qué petición Http (GET, POST, DELETE, etc.) recibirá el método en un URI determinado.

```java
/* Método para pedir token para un usuario
 *
 * @param: Usuario user: Objeto POJO serializable. Se recibe una cadena en formato JSON y se traduce directamente
 *         en un objeto POJO Usuario.
 *
 * @return: Objeto Usuario con el token o mensaje de error en el campo auth.
 *
 */
@RequestMapping(method = RequestMethod.POST, value="/keyrock/token")
@ResponseBody
public ResponseEntity<Usuario> createTokenPassword(@RequestBody Usuario user) throws IOException {
```

Ilustración 39 Anotaciones Spring @RequesMapping y @ResponseBody

También será necesario el uso de anotaciones @RequestBody y @RequestParam para poder obtener el body de la petición Http en formato JSON.

Un factor importante es la modificación de atributos que variarán entre distintos despliegues del sistema. Para solucionar este problema se indican los parámetros en el fichero application.properties y son obtenidos en la clase mediante la etiqueta @Value según lo indicado en la ilustración 40.

```java
// Obtenemos las variables de interés
@Value("${ip_port_keyrock}")
private String ip_port_keyrock;
@Value("${client_id}")
private String client_id;
@Value("${client_secret}")
private String client_secret;
@Value("${application_id}")
private String application_id;
@Value("${organization_id}")
private String organization_id;
```

Ilustración 40 Obtención de los parámetros de application.properties

Un punto importante a destacar es el uso del framework Jersey enfocado en el desarrollo de servicios web RESTful para realizar todas las peticiones al Identity Manager Keyrock.

Los métodos de la clase @Controller son los que implementan todas las funcionalidades de la comunicación con el Identity Manager Keyrock. Los métodos necesarios para poder realizar todos los requisitos del sistema son los siguientes:

- **public ResponseEntity<Usuario> createTokenPassword**: Método para crear un token con un método de contraseña. Recibe las credenciales de usuario y devuelve un token obtenido de Keyrock si la operación ha tenido éxito o no autorizado en caso contrario.

```java
*/
protected Usuario getKeyrockToken(String mail, String password) throws IOException  {

        String token=null;

        // Hacemos la petición a la URL correspondiente de Keyrock, pasando el user y password pasados mediante el formulario
        Client client = ClientBuilder.newClient();
        Entity payload = Entity.json("{  \"name\": \""+mail+"\",  \"password\": \""+password+"\"}");
        // Realizamos la petición POST
        Response response = client.target("http://"+ip_port_keyrock+"/v1/auth/tokens")
          .request(MediaType.APPLICATION_JSON_TYPE)
          .post(payload);

        // Usuario estandar para poder devolverlo vacio en caso de error
        Usuario usu =new Usuario();

        // Una vez tenemos respuesta a la petición, comprobamos si tenemos acceso a Keyrock (201) o no tenemos acceso (401)
        if (response.getStatus() == CREADO) {
```

Ilustración 41 Método getKeyrockToken

Se debe puntualizar que el método createTokenPassword no realiza la acción de creación de token

directamente, sino que delega esta acción en otro método llamado getKeyrockToken, el cual se muestra en la ilustración 41. Esto es debido a la demanda de diferentes métodos para obtener nuevos tokens, lo que obliga a la creación de un método genérico que cree un token a partir de credenciales de un usuario.

- **public ResponseEntity<String> userInfo**: Método para obtener información de un usuario a partir de su token. Este método permite diferenciar usuarios administradores de usuarios estándar.
- **public ResponseEntity<String> createUser:** Método para la creación de un nuevo usuario en el sistema. Este método permite dar de alta un nuevo usuario en Mysql-db a partir de los datos de usuario que el administrador ha introducido previamente en el formulario de alta de usuario, los cuales son enviados por el controlador.
- **public ResponseEntity<String> modificarUsu**: Método para la modificación de la información de un usuario almacenada en Mysql-db. Para que un usuario pueda modificar sus credenciales debe cumplimentar previamente el formulario de modificación de usuario.
- **public ResponseEntity<String> eliminarUsu**: Método para la eliminación de un usuario de la base de datos gestionada por MySQL. Para poder eliminar un usuario el administrador debe cumplimentar previamente el formulario de baja de usuario.

```
@RequestMapping(method = RequestMethod.POST, value="/keyrock/baja")
@ResponseBody
public ResponseEntity<String> eliminarUsu(@RequestBody Usuario user) throws IOException {

    // Hacemos la petición a Keyrock para eliminar al usuario
    Client client = ClientBuilder.newClient();
    Response response = client.target("http://"+ip_port_keyrock+"/v1/users/"+user.getMail())
      .request(MediaType.TEXT_PLAIN_TYPE)
      .header("X-Auth-token", user.getAuth())
      .delete();


    if (response.getStatus()==NO_CONTENT) {

        // Exito al eliminar el usuario
        return  new ResponseEntity<String>("OK", HttpStatus.OK);
    } else {

        // No se pudo eliminar el usuario
        return  new ResponseEntity<String>("ERROR", HttpStatus.OK);
    }

}
```

Ilustración 42 Método eliminarUsu

En la ilustración 42 se muestra el método eliminarUsu, el cual hace uso del framework Jersey, concretamente de las clases Client y Response para poder hacer la petición a Keyrock.

- **public ResponseEntity<String> readPermission**: Método para leer los permisos de un usuario, devolviendo error en caso de no existir o el rol del usuario en caso de que esté dado de alta en la aplicación.
- **public ResponseEntity<String> getOauth2token**: Método similar a createTokenPassword, el cual solicita un "bearer" token a Keyrock mediante un flujo de contraseña defino mediante Oauth 2.0. Debido a la misma razón explicada en el primer método detallado, no se pide el token directamente, sino que se delega la acción en un método protected del @Controller, el cual se muestra en la ilustración 43.

```
protected String getToken(String username, String password)  {

    // Variables a enviar en la petición para obtener un oauth2 token con flujo de password
    // La url donde escucha el idm Keyrock
    String urlString = "http://"+ip_port_keyrock+"/oauth2/token";
    // Codificamos client_id y client_secret en base64
    String idColonSecret = client_id+":"+client_secret;
    String basicAuthPayload = "Basic " + Base64.encode(idColonSecret);

    // Creamos los elementos necesarios para hacer la peitición http
    URL url = null;
    InputStream stream = null;
    HttpURLConnection urlConnection = null;


     try {

         // Creamos el urlConnection, al cual debemos añadir todos los parámetros correspondientes de la petición
         url = new URL(urlString);
         urlConnection = (HttpURLConnection) url.openConnection();
         urlConnection.setRequestMethod("POST");
         urlConnection.setRequestProperty("Authorization", basicAuthPayload);
         urlConnection.setDoOutput(true);

         // Creamos la cadena data con el formato a seguir en un body urlencoded
         String data = URLEncoder.encode("password", "UTF-8")
                 + "=" + URLEncoder.encode(password, "UTF-8");

         data += "&" + URLEncoder.encode("grant_type", "UTF-8") + "="
                 + URLEncoder.encode("password", "UTF-8");

         data += "&" + URLEncoder.encode("client_id", "UTF-8") + "="
                 + URLEncoder.encode(client_id, "UTF-8");

         data += "&" + URLEncoder.encode("client_secret", "UTF-8") + "="
                 + URLEncoder.encode(client_secret, "UTF-8");

         data += "&" + URLEncoder.encode("username", "UTF-8") + "="
```

Ilustración 43 Método getToken

Todos los métodos detallados se mapearán con direcciones URI mediante @RequestMapping. La API RESTful de Keyrockservice se detallará en la siguiente subsección.

Clase POJO para la gestión de usuarios

La clase Usuario.java ejercerá de clase POJO transportando la información obtenida mediante la capa de acceso de datos y transportándola a la capa superior de representación. Esta clase se utiliza para poder guardar la información de usuarios recibidas en objetos directamente manipulables y serializables.

Se representa así un usuario mediante un objeto, el cual almacenará el correo, contraseña, nombre, token e identificador de usuario, lo cual se muestra en la ilustración 44.

```java
package com.example.dto;

public class Usuario {

    // Correo para la consulta a Keyrock
    String mail;
    // Contraseña para la consulta a Keyrcok
    String password;
    // Variable para indicar si un usuario es autorizado o no
    String auth;
    // Nombre del usuario en el gestor de identidad
    String nombre;
    // Identificador de usuario
    String user_id;

    public Usuario() {

    }

    public Usuario(String mail, String password, String auth, String nombre, String token, String bearer) {
        super();
        this.mail = mail;
        this.password = password;
        this.auth = auth;
        this.nombre = nombre;

    }


    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getMail() {
```

Ilustración 44 Clase POJO Usuario.java

Estos atributos pueden ser modificados o solicitados mediante los métodos getters y setters correspondientes a cada atributo.

5.2.2 Keyrockservice REST API

En esta subsección se documenta la API REST del servicio web Keyrockservice, detallando los distintos URI y métodos mapeados según la etiqueta @RequestMapping, quedando así definida la API que las aplicaciones externas deberán de usar para comunicarse con nuestro servicio web.

En el marco del proyecto todas las peticiones realizadas al servicio web Keyrockservice mediante su API son realizadas mediante peticiones HTTP con cuerpo con formato JSON, el cual contendrá los parámetros de usuarios. Este hecho será representado en la tabla 3 mediante la columna “Parámetros” y no mediante la notación datos?(&{parámetro}={valor}…) características de peticiones con cuerpo url-encoded.

Método	URI	Parámetros	Descripción
POST	/keyrock/token	{“mail”:”ejemplo@test.com”, “password”:”test”} Mail: Correo del usuario. Password: Contraseña del usuario.	Petición a Keyrock para crear un token a partir de las credenciales de usuario pasadas como parámetro.
POST	/keyrock/info	String token Token del usuario del cual se quiere obtener información.	Petición a Keyrock para obtener información de un usuario a partir de su token.
POST	/keyrock/alta	{“mail”:”ejemplo@test.com”, “password”:”test”, “token”:”834udfkfus”, “nombre”:” alvaro”} Token: Token del usuario obtenido de Keyrock. Nombre: Nombre del usuario a dar de alta en Mysql-db.	Petición a Keyrock para crear un usuario en el sistema mediante el gestor de base de datos Mysql.
POST	/keyrock/dato_modificado	{“mail”:”ejemplo@test.com”, “token”:”834udfkfus”, “nombre”:” alvaro”} Mail: Nuevo correo del usuario. Nombre: Nuevo nombre del usuario.	Petición a Keyrock para modificar los datos de un usuario dado de alta en el sistema.
POST	/keyrock/baja	{“user_id”:”3849502094”, “token”:”834udfkfus”} User_id: Identificador de usuario de Keyrock que se desea eliminar. Token: Token del administrador con permisos para eliminar usuarios.	Petición a Keyrock para eliminar a un usuario de la base de datos a partir de su identificador de usuario.
POST	/keyrock/permisos	String token	Petición a Keyrock para consultar los

		Token: Token del usuario del cual se quieren consultar sus permisos dentro de la organización de la aplicación.	permisos referentes a un usuario.
POST	/keyrock/bearertoken	{“mail”:”ejemplo@test.com”, “password”:”test”} Mail: Correo del usuario. Password: Contraseña del usuario.	Petición a Keyrock para obtener un bearer token gracias a un flujo de contraseña establecido por el protocolo de seguridad Oauth 2.0.

Tabla 3 Keyrockservice API

Una característica fundamental del servicio web es la actuación indirecta sobre el recurso, es decir es Keyrock quien actúa directamente sobre el recurso deseado, mientras que Keyrockservice se encarga de realizar las peticiones necesarias a Keyrock mediante su API. Debido a esta razón solo se ha empleado el método POST para el mapeo de los distintos métodos del servicio web, ya que el servicio web Keyrockservice debe recibir en todo momento toda la información referente a la acción CRUD a realizar.

5.3 Servicio Web RESTful: Sensorservice

El servicio web Sensorservice presenta una división en capas idéntica a la detallada en la sección 5.2 del presente capítulo. Concretamente, Sensorservice se encargará de la lógica de negocio y acceso de datos, apoyándose en distintas clases POJO para poder transportar la información entre las distintas capas.

Sensorservice juega un papel fundamental en el proyecto debido a dos razones:

- Encargado de las subscripciones y desuscripciones realizadas con Orion Context Broker para poder almacenar la información de contexto frente a cambios.
- Solicitudes a la base de datos mediante Hibernate.

Sensorservice estará protegido por PEP-Proxy Wilma, siendo Wilma el encargado de interceptar las peticiones realizadas al servicio y verificando la identidad del usuario, paso previo para poder dar acceso al servicio.

5.3.1 Servidor de Base de Datos

La base de datos Matriculas se ha diseñado con la finalidad de almacenar toda la información referente a los vehículos y sus dueños. Para ello se ha diseñado una base de datos relacional gestionada por el sistema gestor PostgreSQL [10] con las siguientes relaciones, las cuales son mostradas en la ilustración 45:

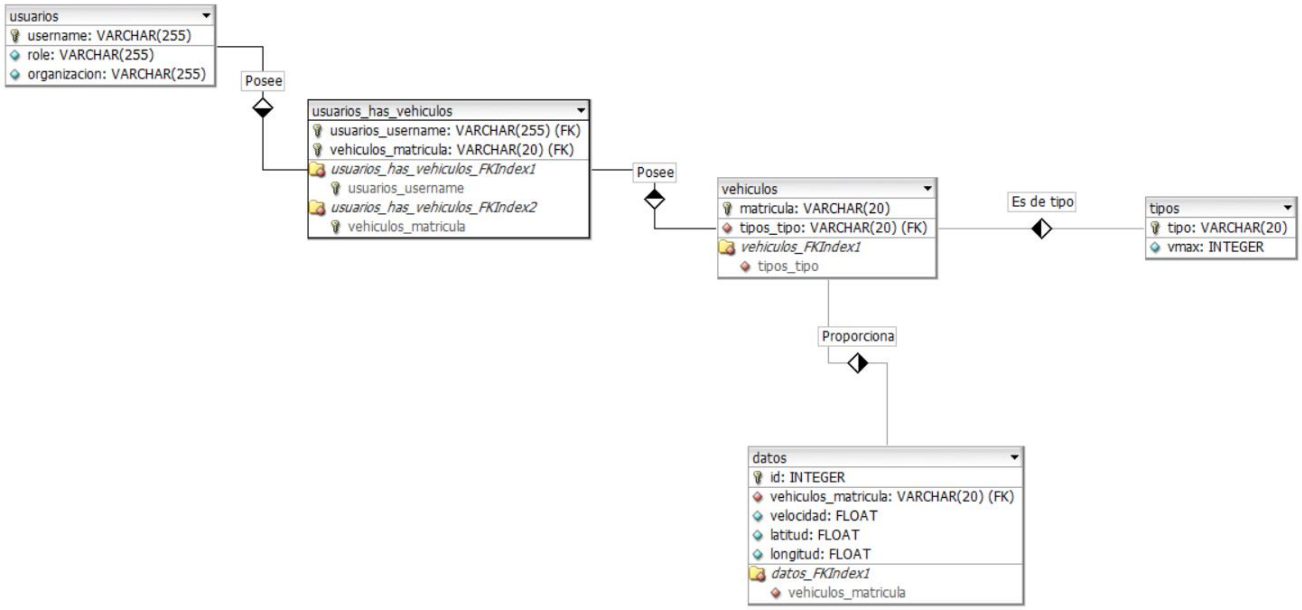


Ilustración 45 Relaciones Base de Datos

Las relaciones existentes son las siguientes:

- Usuarios: Relación para almacenar un usuario dado de alta en el sistema.
- Vehiculos: Relación para almacenar un vehículo dado de alta en el sistema.
- Tipos: Tipos existente para cada vehículo y tu velocidad máxima permitida.
- Datos: Datos almacenados por cada vehículo en función de su matrícula.
- Usuarios_has_vehiculos: Relación que almacena el usuario y su vehículo. Es la relación más "delicada" ya que no puede ser modificada hasta que se modifiquen las relaciones Usuarios y Vehiculos, pudiendo "violarse" las claves externas en caso contrario.

```
matriculas=> select * from usuarios_has_vehiculos;
          usuarios_username           |    vehiculos_matricula
--------------------------------------+---------------------------
 admin                                | 3456ABC
 203528e9-1697-45e2-9877-f0d53ce62e47 | 2233KAL
 13340651-197d-4b06-91c4-9d7d0db19b8a | 8976FDS
 f8c95255-e7d3-4c4c-8c77-80ce302db397 | 4545AFG
 4d45e36a-46fb-4212-8379-356977de1468 | 1234JDK
 8428b8ab-298e-4ee5-95fc-e10678e3ed3a | 1212KEY
 9e544127-1e29-42bd-8df2-2c7d61b430d2 | 1314POG
 d2e6d3eb-f750-4153-8311-cd54379d39cd | 3489DFG
 6c1cef77-9028-455d-82de-bc8f63366fd7 | 4567ASD
 122458ae-46c7-4377-be61-42b6aa52710f | 3532DAD
 664e22cd-a32a-4517-8d54-66357e6796d6 | 1315SOS
 31dca248-9240-43bc-973d-3350688dca53 | 3423DEF
 86d53c28-46f7-461c-872b-fd2332ced8f5 | 2345SAS
 3315b8e0-ced6-47d1-9661-a52c8d8af79a | 2324SAS
 795c1574-ee22-4d98-99a4-5621cda2d6c4 | 2929LOL
 a008633e-5197-4e4e-8981-b2de650bbfca | urn:ngsi-ld:Sensor:0101ALV
 536b8654-dbca-4d91-8f3b-a7314d439776 | urn:ngsi-ld:Sensor:2929LOL
 8d0d8b57-6e9e-427f-b7e6-d972431bb007 | urn:ngsi-ld:Sensor:1236KYL
 e03f561e-b3ba-4633-80f0-85815ae77623 | urn:ngsi-ld:Sensor:4444TNT
(19 rows)

matriculas=>
```

Ilustración 46 Contenido relación Usuarios_has_vehiculos

En la ilustración 46 se muestra el contenido de la relación Usuarios_has-vehiculos, perteneciendo las primeras líneas a pruebas del sistema con matrículas simples.

5.3.2 Clases

En esta subsección se detalla de forma simular a la sección anterior las clases necesarias para el correcto funcionamiento del servicio web.

Clases POJO para la comunicación con el context broker

El funcionamiento de una clase POJO fue indicado en la sección anterior, siendo necesario nuevas clases POJO para poder recibir las notificaciones de Orion Context Broker, de manera que la información pueda ser guardada en objetos Java y convirtiendo las entidades que recibe el servicio web en formato JSON, en objetos de los cuales podemos obtener todos sus parámetros.


```
{
    "id": "5ea608d13da3d4356ea4a3da",
    "description": "A subscription to get info about GPS",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": "urn:ngsi-ld:turismo:2929LOL",
                "typePattern": "Sensor"
            }
        ],
        "condition": {
            "attrs": [
                "latitud",
                "longitud",
                "altitud"
            ]
        }
    },
    "notification": {
        "attrs": [],
        "onlyChangedAttrs": false,
        "attrsFormat": "normalized",
        "http": {
            "url": "http://192.168.0.33:9000/sensor/notificaciones"
        }
    }
```


Ilustración 47 Ejemplo Subscripción

En la ilustración 47 se muestra un ejemplo de subscripción, la cual debemos de descomponer en una clase POJO general Notification.java que representará la subscripción.

- **Notification.java**

Clase para poder convertir una notificación que se recibirá en formato JSON cuando se actualice la información de context, en un objeto JAVA del cual se podrá obtener directamente mediante Getters valores como identificador de subscriipción. Una notificación es algo complejo, por lo que necesitaremos varias clases POJO para poder convertir cada aspecto de la notificación en una clase JAVA. La clase Notification.java contendrá una lista con todas las entidades que han sido notificadas.

- **Entity**

La clase Entity.java define los objetos que representan la información (velocidad, longitud y latitud) de las entidades del OCB, conservando la estructura que, en formato JSON, tenían estas. Tiene, por tanto, un atributo por cada uno de los atributos de dichas entidades. Los atributos “id” y “type”, al igual que en el context broker, son simplemente cadenas de caracteres que contienen la matrícula y el tipo de

vehículo, respectivamente. Cada parámetro de la clase Entity.java representa un atributo que se recibe de la notificación con el siguiente formato mostrado en la ilustración 48:

Ilustración 48 Formato Atributo de una Entidad del OCB

Debido a esto un parámetro de Entity.java será de tipo Attribute que representará el formato indicado en la ilustración 48, y un attribute contendrá otro objeto POJO de tipo "metadata".

- **Attribute.java**

La clase Attribute.java representa un atributo del cual se envían mediciones del OCB, el cual tendrá un tipo, valor y un objeto POJO de tipo Metadata.java.

- **Metadata.java**

Especificará qué es el parámetro que se ha medido y es representado mediante una instancia de Attribute.java, por ejemplo, un objeto Attribute de tipo longitud.

Clases POJO para la comunicación con la base de datos

Estas clases son utilizadas para abstraer la comunicación con la base de datos relacional mediante Hibernate, creando una clase por relación existente en la base de datos, las cuales serán mapeadas mediante ficheros de configuración Hibernate con las relaciones de la base de datos. De esta forma tendríamos las siguientes clases:

- Dato.java: Clase para representar una medición de velocidad, longitud y latitud de la base de datos relacional Datos.
- Tipo.java: Clase para representar un tipo de vehículo (turimos, camión, etc.) en la base de datos relacional.
- User.java: Clase para representar un usuario de la relación Usuarios, el cual queda representado por su identificado de usuario de Keyrock, role y organización.
- Vehiculos: Clase para representar un vehículo de la relación Vehiculos mediante su matrícula y tipo de vehículo.
- Usuarios_has_vehiculos: Clase para representar un vehículo y su usuario asociado.

Estás clases deben ser mapeadas mediante ficher Hibernate, para que una clase JAVA pueda representar una relación de la base de datos. Esto se consigue creando un fichero por cada clase POJO, tal y como se muestra en la ilustración 49 para la clase Dato.java:

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 03-feb-2019 13:01:42 by Hibernate Tools 3.5.0.Final -->
<hibernate-mapping>
    <class name="com.sensor.dto.Dato" table="datos">
        <id name="id" type="long">
            <column name="id"/>
            <generator class="sequence-identity">
                <param name="sequence">datos_id_seq</param>
            </generator>
        </id>

        <property name="vehiculos_matricula" column="vehiculos_matricula" type="string"/>
        <property name="velocidad" column="velocidad" type="double"/>
        <property name="latitud" column="latitud" type="double"/>
        <property name="longitud" column="longitud" type="double"/>

    </class>
</hibernate-mapping>

```

Ilustración 49 Fichero Dato.hbm.xml

El proceso será el mismo para el resto de las clases POJO, siguiendo el mismo proceso y cambiando la relación a la cual el fichero hace referencia. Una vez creados los ficheros de configuración, debemos crear el fichero de configuración general de Hibernate para la conexión con la base de datos, tal y como se muestra en la ilustración 50:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
        <property name="hibernate.connection.password">641997Al</property>
        <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/matriculas</property>
        <property name="hibernate.connection.username">admin</property>
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>

        <mapping resource="Dato.hbm.xml"></mapping>
        <mapping resource="Vehiculo.hbm.xml"></mapping>
        <mapping resource="Tipo.hbm.xml"></mapping>
        <mapping resource="User.hbm.xml"></mapping>
        <mapping resource="Usuarios_has_vehiculos.hbm.xml"></mapping>

    </session-factory>
</hibernate-configuration>
```

Ilustración 50 Fichero hibernate.cfg.xml

En el fichero “hibernate.cfg.xml” [11] se debe especificar el drive, url y las credenciales para poder realizar la conexión con la base de datos.

Clases que controlan el funcionamiento de la aplicación

Las clases encargadas del funcionamiento de la aplicación son las siguientes:

- **SensorserviceApplication.java**

Clase @SpringBootApplication que contiene el método main del programa. El método main será el primero en llamarse una vez se ha ejecutado la aplicación y mediante la etiqueta @SpringBootApplication (scanBasePackages = {"com.sensor"},exclude = {SecurityAutoConfiguration.class }) queda la aplicación inicializada.

- **AppController.java**

Esta clase ejercerá de controlador de la aplicación Spring desarrollada, presentando una API RESTful para poder ser usada en las comunicaciones con nuestro servicio. La clase AppController ejercerá de capa lógicad de negocio, definiendo una serie de métodos mapeados mediante etiquetas Spring en direcciones URIs, las cuales recibirán las peticiones procedentes de PEP-Proxy Wilma.

Un punto importante que se debe destacar es que el servicio web Sensorservice no es accedido directamente por los usuarios, sino que los usuarios enviarán sus peticiones a PEP-Proxy Wilma y éstas serán redirigisas a la URI correspondiente. Esta traducción es realizada por Wilma del siguiente modo:

1. El controlador envía una petición realizada por el usuario desde la aplicación web a la dirección http://wilmahost:1027/baja.
2. PEP-Proxy Wilma intercepta la petición y comprueba el bearer token del usuario. En caso de que el usuario tenga los privilegios y permisos requeridos, PEP-Proxy Wilma redirigirá la petición.
3. Wilma reenvía los parámetros de la petición al endpoint /baja , el cual se habrá mapeado en la clase controlador a un método concreto, en este caso la de baja de un usuario.

Es esencial detallar una serie de parámetros en el fichero application.properties, tal y como se explicón en el servicio Keyrockservice. Estos parámetros serán:

- server.port=9000
- admin_id=adin
- ip_port_contextbroker=localhost:1026
- ip_port_sensorapp=localhost:9000

Se detallan a continuación los métodos mapeados mediante @RequestMapping, los cuales implementan todas las funcionalidades de consulta de datos y subscripciones con Orion Context Broker:

- **public ResponseEntity<String> crearusu**: Método para dar de alta un usuario en la base de datos gestionada por PostgreSQL, en la cual se almacenan los vehículos y sus usuarios propietarios correspondientes.
- **public ResponseEntity<String> pedirinfo:** Método para consultar una matrícula dada, la cual se debe comprar que pertenezca al usuario que realiza la petición en caso de no ser administrador.
- **public ResponseEntity<String> eliminarusu**: Método para dar de baja a un usuario, es decir eliminar los vehículos y el usuario de la base de datos.
- **public void notificaciones**: Método que recibirá un objeto de tipo Notification, el cual contendrá la información actualizada de un sensor.

El método notificaciones debe ser mapeado a una URI conocida, la cual será indicada a la hora de realizar una subscripción con la finalidad de que sea a esta URI a la que se avise cuando el mecanismo de subscripciones detecte un cambio en la información de contexto.

```
],
{
    "id": "5ea98f4bbfee5fa5981cf757",
    "description": "Notify me of GPS changes",
    "status": "active",
    "subject": {
        "entities": [
            {
                "idPattern": " urn:ngsi-ld:Sensor:0101ALV ",
                "type": "Sensor"
            }
        ],
        "condition": {
            "attrs": [
                "latitud",
                "velocidad",
                "longitud"
            ]
        }
    },
    "notification": {
        "attrs": [],
        "onlyChangedAttrs": false,
        "attrsFormat": "normalized",
        "http": {
            "url": "http://192.168.0.34:9000/sensor/notificaciones"
        }
    }
}
```


Ilustración 51 Ejemplo Subscripción OCB

En la ilustración 51 se observa un ejemplo de subscripción a la entidad "urn:ngsi-ld:Sensor:0101ALV", la cual se indica en el campo "notification" la URL del método indicado mediante la URL "http://192.168.0.34:9000/sensor/notificaciones".

También se han creado métodos no mapeados a ninguna URI concreta, siendo éstos necesarios como un método que permita crear una subscripción para una matrícula dada, un método para crear un nuevo sensor o un método para desuscribirse de una matrícula que ha sido dada de baja.

```
public boolean suscribir(String mat, String tipo) throws IOException {

    // Id para realizar la subscripción
    String matricula=null;
    matricula="urn:ngsi-ld:Sensor:"+mat;

    // Mensaje en caso de error en la subscripción
    boolean mensaje = false;


    Client client = ClientBuilder.newClient();
    Entity payload = Entity.json("{\"description\": \"A subscription to get info about GPS\", "
            + "\"subject\": {\"entities\": [{\"idPattern\": \""+matricula+"\", \"typePattern\": \"Sensor\"}], "
            + "\"condition\": {\"attrs\": [\"latitud\", \"longitud\", \"altitud\"]}}, " //Si cambia alguno se envia notif
            + "\"notification\": {\"http\": {\"url\": \"http://"+ip_port_sensorapp+"/sensor/notificaciones\"}, "
            + "\"attrs\": [], " //Si no se especifican atributos se notifican todos
            + "\"attrsFormat\": \"normalized\"}}");

    Response response = client.target("http://"+ip_port_contextbroker+"/v2/subscriptions")
            .request(MediaType.APPLICATION_JSON_TYPE)
            .header("fiware-service","smartcar")
            .header("fiware-servicepath","/")
            .post(payload);
```

Ilustración 52 Método para realizar subscripción

En la ilustración 52 se muestra el método encargado de realizar una subscripción a una matrícula dada, el cua será llamado cuando un nuevo vehículo sea dado de alta.

- **DataBase**

La clase DataBase contiene los métodos que realizan las transacciones con la base de dastos mediante el uso de Hibernate. Cada método que necesite realizar una acción concreta sobre la base de datos de la clase

AppController.java invocará un método de la clase DataBase.java para poder realizar dicha tarea. Los métodos de esta clase son:

- **public boolean crearUsuario:** Método para dar de alta un nuevo usuario, creando el usuario en la relación Usuarios y el vehículo en la relación Vehiculos. No se pueden "violar" las claves externas de la base de datos, por lo que antes de modificar la relación Usuarios_has_vehiculos se deben modificar las relaciones Usuarios y Vehiculos.
- **public List<Dato> consultarDatos:** Método para consultar los datos de una matrícula dada, la cual debe haber sido creada previamente.

```
public List<Dato> consultarDatos(String mat) {

        //Comenzamos una transacción con la base de datos
        SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
        Session session = sessionFactory.openSession();
        session.beginTransaction();
        //Configuramos los parámetros necesarios para las consultas dinámicas
        Criteria cr = session.createCriteria(Dato.class);

        //Añadimos las condiciones dinámicas según los parámetros introducidos
        if(mat!=null) {
            cr.add(Restrictions.eq("vehiculos_matricula", mat));
        }

        List<Dato> datos = cr.list();

        session.getTransaction().commit(); //Terminamos transacción
        session.close(); //Cerramos sesión
        sessionFactory.close(); //Cerramos conexiones con BD

        return datos;
}
```

Ilustración 53 Método consultarDatos DataBase.java

En la ilustración 53 se muestra el método consultarDatos, el cual devolverá una lista con objetos Dato con las mediciones para ese vehículo.

- **public void insertarDatos:** Método para actualizar los datos de un vehículo cuando el mecanismo de subscripciones envie las nuevas medidas al método notificaciones de AppController.java.
- **public boolean eliminarDatos:** Método para dar de baja un usuario. Se debe seguir el mismo procedimiento que en el método crearUsuario, pero realizando la acción opuesta.

5.3.3 Sensorservice REST API

En esta subsección se documenta la API REST del servicio web Sensorservice, detallando los distintos URI y métodos mapeados según la etiqueta @RequestMapping, quedando así definida la API que las aplicaciones externas deberán de usar para comunicarse con nuestro servicio web.

La API RESTful ofrecida para la comunicación con el servicio web es idéntica a la detallada para el servicio Keyrockservice, ya que las acciones a realizar son similares (alta de usuario, baja de usuario, etc.). La API RESTful presentada se detalla en la tabla 4.

Método	URI	Parámetros	Descripción
POST	/sensor/alta	-@RequestHeader("X-NICK-NAME") String username -@RequestBody Usuario user String username: Identificador enviado por Wilma. Usuario user: Identificador y matrícula enviada por el usuario.	Método para crear un usuario nueva en la base de datos a partir de su matrícula y su identificador.
POST	/sensor/info	-@RequestHeader("X-NICK-NAME") String username -@RequestBody Usuario user String username: Identificador enviado por Wilma. Usuario user: Identificador y matrícula enviada por el usuario.	Método para solicitar la información de un usuarios a partir de su matrícula e identificador.
POST	/sensor/baja	-@RequestHeader("X-NICK-NAME") String username -@RequestBody Usuario user String username: Identificador enviado por Wilma. Usuario user: Identificador y matrícula enviada por el usuario.	Método para eliminar un usuario a partir de su identificador y la matrícula de su vehículo.
POST	/sensor/notificaciones	@RequestBody Notification notificacion -Notification notificación: Objeto POJO para convertir la notificación generada por el mecanismo de subscripciones en un objeto JAVA sobre el cual se podrá trabajar.	Método donde se recibirán las notificaciones por parte del OCB cuando cambie la información de contexto.

Tabla 4 Sensorservice API

Al igual que en el caso anterior los parámetros son enviados en el cuerpo de la petición en formato JSON y son convertidos a objetos JAVA mediante la clase POJO Usuarios. Todas las peticiones a la API se realizan mediante peticiones POST, ya que las peticiones deben ser realizadas a PEP-Proxy Wilam en primera instancia enviando el bearer token de usuario.

6 SOLUCIÓN DESARROLLADA: GENERACIÓN Y CONSUMO DE INFORMACIÓN DE CONTEXTO

Vivimos en una sociedad profundamente dependiente de la ciencia y la tecnología y en la que nadie sabe nada de estos temas. Ello constituye una fórmula segura para el desastre.

- Carl Sagan -

En este capítulo se detalla la aplicación móvil desarrollada, la cual leerá el sensor GPS del teléfono móvil y generará valores aleatorios dentro de valores establecidos para simular las mediciones reales de sensores instalados previamente en el vehículo inteligente.

También es necesario explicar la relación existente entre los dos componentes que se encargan del procesamiento y gestión de la información de contexto, Orion Context Broker y IoT Agent Ultralight. Todos estos componentes pertenecen al bloque IV detallado en la arquitectura del sistema mostrada en el capítulo I.

El último punto a abordar será el último de los servicios web desarrollados denominado Middleservice, encargado de ser una barrera entre el mundo del Internet de las Cosas y IoT Agent Ultralight, pudiendo usar así las distintas ventajas ofrecidas por el framework Eclipse Jersey.

6.1 IoT Agent Ultralight

El primer punto que se debe mencionar es la configuración de IoT Agent Ultralight [10] para el correcto funcionamiento del procesamiento de la información de contexto. Para poder realizar la configuración se trabajará directamente con Postman sobre el puerto norte, realizando las siguientes peticiones:

Crear un servicio: El primer paso será crear un servicio, paso previo a la creación de cualquier vehículo en el sistema como ya se detalló en el capítulo 5 de la memoria. En la ilustración 54 se muestra la petición que se tendría que crear en Postman para automatizar la creación de un servicio.

```
-----------------------------
CREAR SERVICIO
-----------------------------
curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: smartcar' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "entity_type": "Sensor",
     "resource":    "/iot/d"
   }
 ]
}'
```

Ilustración 54 Crear servicio

Los parámetros más importantes son:

- Apikey: Clave que tendrá que enviar un sensor para poder autenticar y enviar las nuevas mediciones.
- Cbroker: Dirección de Orion Context Broker
- Fiware-service: Nombre del servicio
- Entity_type: Tipo de las entidades que se crearán en OCB.

Una vez se ha creado el servicio se podrá consultar los servicios creados mediante la petición mostrada en la ilustración 55.

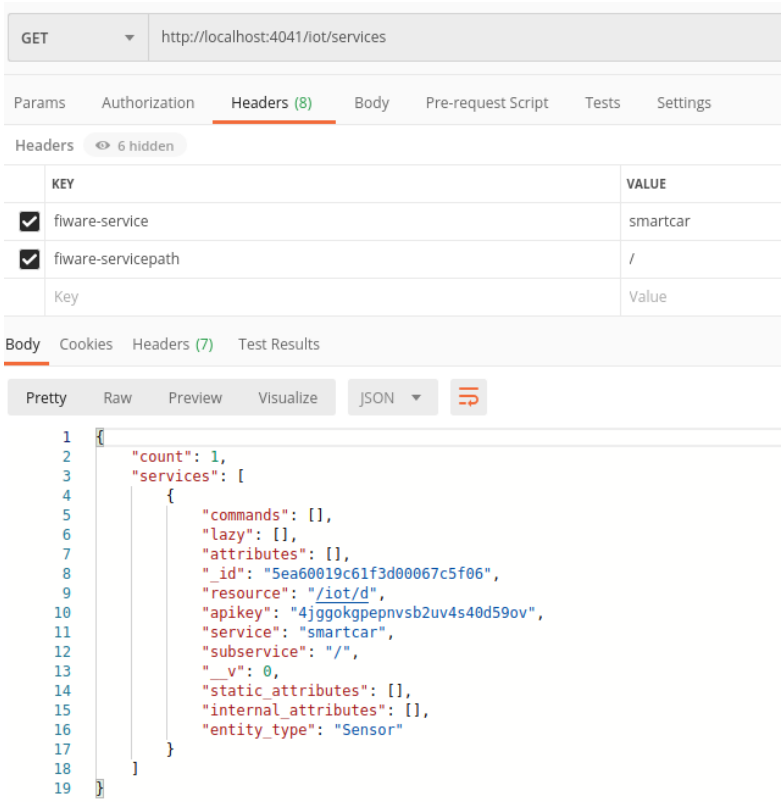


Ilustración 55 Solicitar servicios existentes

Una vez el servicio ha sido creado, el resto de la configuración será realizada por el servicio Sensorservice tal y como se detalló en el capítulo 5, creando los distintos sensores para cada vehículo y dándolos de alta en el servicio previamente en el Agente IoT creado mediante Postman.

También se podrá consultar directamente en Orion Context Broker las entidades creadas mediante una petición que debe especificar el servicio cuyas entidades se quieren obtener.

Toda la configuración se realiza directamente sobre el puerto norte de IoT Agent Ultralight, siendo éste quien se encargue automáticamente de la comunicación con Keyrock.

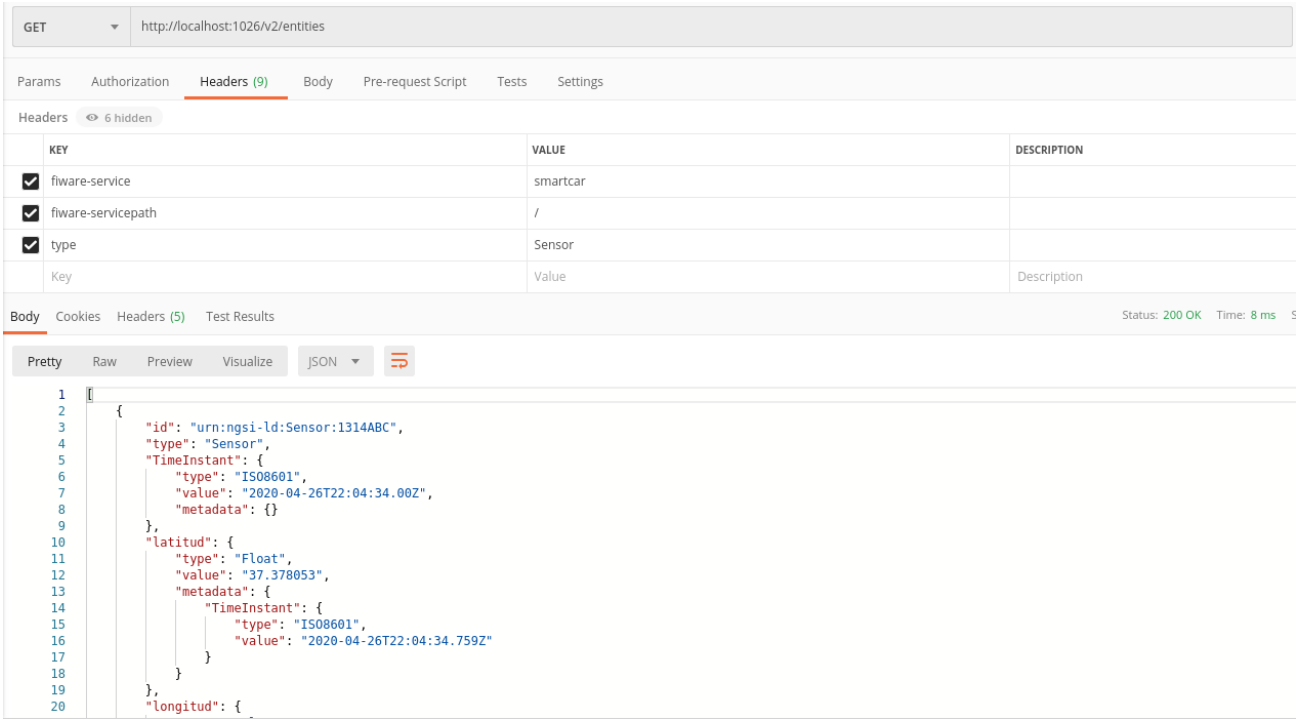


Ilustración 56 Solicitar entidades existentes

En la ilustración 56 se muestra la petición a crear en Postman y las cabeceras para obtener todas las entidades del servicio smartcar previamente creado en el Agente IoT.

6.2 Aplicación Móvil

Para la medición de los sensores se ha realizado una aplicación móvil muy sencilla, la cual se encarga de la lectura del sensor GPS del teléfono móvil y la generación aleatoria de un valor válido de velocidad cada vez que se detecta un cambio en la localización.

La aplicación móvil se ha desarrollado con la idea de ser una aplicación con poco impacto en necesidad de recursos del teléfono y con un layout muy sencillo el cual se muestra en la ilustración 57.

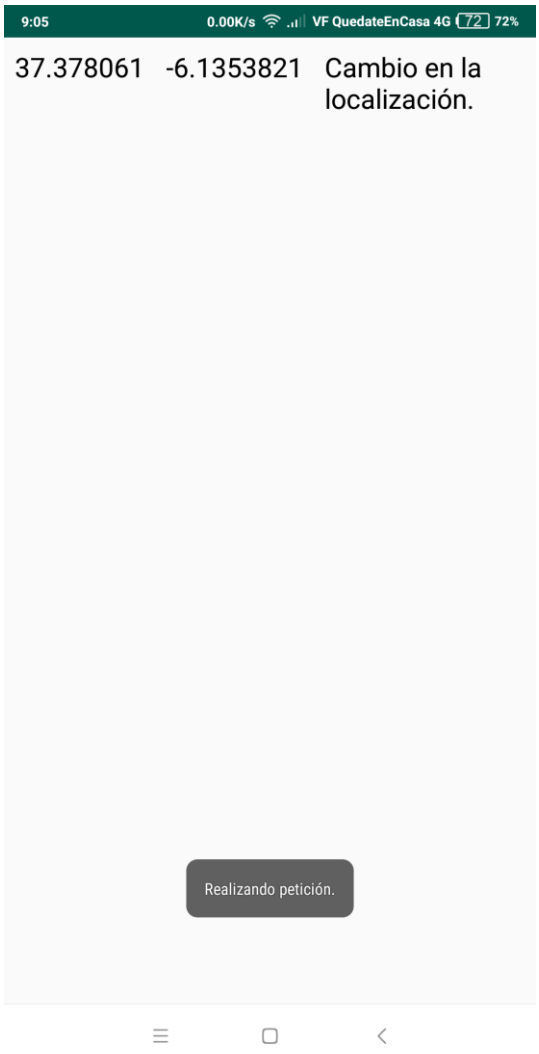


Ilustración 57 Layout Aplicación Móvil

El primer paso para el correcto funcionamiento de la aplicación será definir los permisos necesarios en el fichero AndroidManifest.xml, los cuales se muestran en la ilustración 58.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="com.google.android.things.permission.MANAGE_GNSS_DRIVERS" />

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Ilustración 58 Android Permisos

Serán necesarios los siguientes permisos:

- Localización: Permisos para poder acceder a la medición del sensor GPS.
- Internet: Permisos para poder realizar peticiones HTTP mediante Volley al servicio web Middleservice.

La medición del sensor GPS se realiza de forma sencilla con LocationManager, siendo el primer paso verificar si existen los permisos necesarios. En caso de existir los permisos se obtiene la última localización conocida y

si esta ha cambiado se llama al método onLocationChanged, tal y como se muestra en la ilustración 59.

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
Location location = locationManager.getLastKnownLocation(locationManager.NETWORK_PROVIDER);

onLocationChanged(location);
```

Ilustración 59 LocationManager

El paso final será realizar la petición HTTP correspondiente con las mediciones obtenidas, para lo que se usará la librería de HTTP Volley, tal y como se muestra en la ilustración 60.

```
JSONObject jsonBodyObj = new JSONObject();
String url = "http://192.168.0.34:9001/sensor/info";
try{
    jsonBodyObj.put( name: "velocidad", velocidad);
    jsonBodyObj.put( name: "longitud",String.valueOf(location.getLongitude()));
    jsonBodyObj.put( name: "latitud", String.valueOf(location.getLatitude()));
    jsonBodyObj.put( name: "sensor",  value: "Sensor001");

}catch (JSONException e){
    e.printStackTrace();
}
final String requestBody = jsonBodyObj.toString();

JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST,
        url,  jsonRequest: null, new Response.Listener<JSONObject>(){
    @Override    public void onResponse(JSONObject response) {
        Log.i( tag: "Response",String.valueOf(response));
    }
}, new Response.ErrorListener() {
    @Override    public void onErrorResponse(VolleyError error) {
        VolleyLog.e("Error: ", error.getMessage());
    }
}){
    @Override    public Map<String, String> getHeaders() throws AuthFailureError {
        HashMap<String, String> headers = new HashMap<~>();
        headers.put("Content-Type", "application/json");
        return headers;
    }
```

Ilustración 60 Petición POST mediante Volley Android

Volley nos permitirá realizar de forma muy sencilla peticiones HTTP, indicando los valores de velocidad, longitud, latitud y el identificador de sensor.

Se debe realizar unas observaciones previas que condicionan en gran medida el funcionamiento de la aplicación:

- El identificador del sensor (“Sensor001”) debe ser previamente conocido por la aplicación y ha debido ser creado y dado de alta en el servicio antes de que se empiecen a tomar las medidas.
- Las peticiones HTTP mediante Volley se podrían realizar directamente al puerto Sur del IoT Agent mediante el formato Text_Plain. Sin embargo, Volley no soporta dicho formato, por lo que los datos obtenidos serán enviados mediante un formato JSON al servicio web Middleservice, el cual se

encargará mediante Eclipse Jersey de realizar la petición al puerto sur.

- El valor de velocidad no es obtenido de un sensor, sino que es generado aleatoriamente a partir de unas restricciones previas para poder obtener siempre un valor de velocidad lógico y válido.

6.3 Servicio Web RESTful Middleservice

El servicio web Middleservice tiene una única función, recibir los parámetros obtenidos por cada sensor y enviarlos por el puerto sur del IoT Agent Ultralight. Para esta finalidad se utilizan las etiquetas @Spring mencionadas en el capítulo 5, tal y como se muestra en la ilustración 61:

```
@Controller
public class MiddleController {

    //Método de testing para generar informaciçon de contexto
    @RequestMapping(method = RequestMethod.POST, value="/sensor/info")
    @ResponseBody
    public void test(@RequestBody Sensor sensor){

        // Creamos la entidad correspondiente, en caso de existir se nos indicará que ha habido un conflicto
        Client client = ClientBuilder.newClient();
        Entity payload = Entity.text("lat|"+sensor.getLatitud()+"|lon|"+sensor.getLongitud()+"|vel|"+sensor.getVelocidad());
        Response response = client.target("http://192.168.0.34:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=Sensor008")
                .request(MediaType.TEXT_PLAIN_TYPE)
                .post(payload);
```

Ilustración 61 Petición puerto sur IoT Agent

Para realizar la petición se utiliza el framework Eclipse Jersey y se realiza al puerto 7896, coincidiendo éste con el puerto sur configurado previamente en el IoT Agent. Se deben realizar las siguientes aclaraciones:

- La petición al puerto sur debe incluir el apikey configurado en el servicio.
- La petición al puerto sur debe ser realizad a la dirección /iot/d configurada previamente en el servicio.
- La petición debe incluir el identificador del sensor previamente creado cuyos datos quieren ser actualizados.

Una vez configurado el servicio se puedo comprobar el correcto funcionamiento y envío de los datos mediante un programa que permita analizar los paquetes enviados como Wireshark, tal y como se muestra en la ilustración 62.

```
  9 0.3913… 192.168.0.17    192.168.0.34    HTTP 4… POST /sensor/info HTTP/1.1  (application/json)
 10 0.3913… 192.168.0.34    192.168.0.17    TCP   66 9001 → 51390 [ACK] Seq=1 Ack=386 Win=64896 Len=0 TSval=4174441345 TSecr=122
 11 0.9410… Vmware_6e:2d:af Broadcast       ARP   42 Who has 192.168.0.33? Tell 192.168.0.34
 12 0.9581… 192.168.0.34    192.168.0.17    HTTP 1… HTTP/1.1 200
```

```
▸ Frame 9: 451 bytes on wire (3608 bits), 451 bytes captured (3608 bits) on interface 0
▸ Ethernet II, Src: XiaomiCo_ce:a4:fd (9c:2e:a1:ce:a4:fd), Dst: Vmware_6e:2d:af (00:0c:29:6e:2d:af)
▸ Internet Protocol Version 4, Src: 192.168.0.17, Dst: 192.168.0.34
▸ Transmission Control Protocol, Src Port: 51390, Dst Port: 9001, Seq: 1, Ack: 1, Len: 385
▸ Hypertext Transfer Protocol
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member Key: velocidad
        Number value: 82.57686670613714
        Key: velocidad
    ▾ Member Key: longitud
        String value: -6.1354963
        Key: longitud
    ▾ Member Key: latitud
        String value: 37.3781266
```

Ilustración 62 Petición Volley Wir

7 SOLUCIÓN DESARROLLADA: APLICACIÓN WEB

No te puedo dar la fórmula para el éxito, pero puedo darte la fórmula para el fracaso: trata de complacer a todo el mundo.

- Hebert B. Swope -

En este capítulo se profundiza en los elementos del bloque I, de acuerdo a lo detallado en la subsección 1.4.2: Arquitectura del Sistema. Se hace indispensable la creación de una aplicación web capaz de ofrecer una interfaz de usuario para acceder al servicio, la cual contribuya a una mayor satisfacción del usuario.

El capítulo se dividirá en dos secciones. La primera detallará la parte Backend de la aplicación web, la cual junto a los servicios web RESTful expuestos en el capítulo 5 componen el núcleo del sistema. La segunda parte detallará de forma visual la interfaz que los usuarios tendrán disponibles para poder realizar las distintas funcionalidades del sistema.

7.1 Aplicación web: Backend

Parte del Backend de nuestro sistema estará formado por la lógica del servidor web que posibilita el funcionamiento de la aplicación web desarrollada, haciendo uso de un estilo de arquitectura software Modelo Vista Controlador (MVC), el cual se muestra en la ilustración 63:

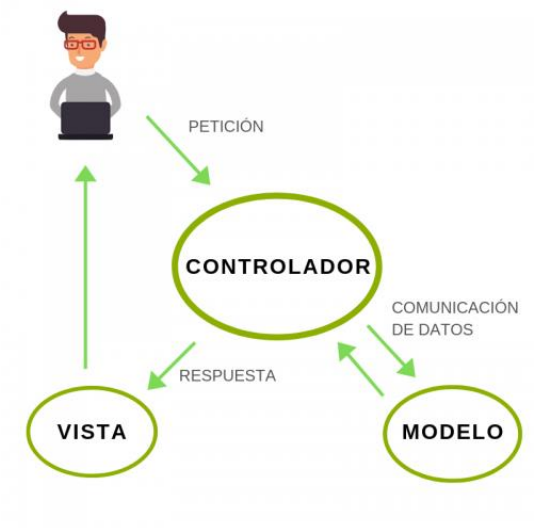


Ilustración 63 Modelo Vista Controlador

- **Modelo**: los datos y lógica del negocio. En nuestra aplicación el modelo será toda la información almacenada en el sistema, la cual es almacenada en tres bases de datos diferenciadas:
    1. **Base de datos de Keyrock**: Base de datos gestionada mediante MySQL, la cual almacena toda la información referente al Identity Manager Keyrock referente a la información de los usuarios (identificadores, corresos y contraseñas) dados de alta en el sistema.
    2. **Base de datos de Vehículos**: Base de datos gestionada mediante PostgreSQL, la cual almacena toda la información referente a los usuarios y sus vehículos, así como de todas las mediciones realizadas por los sensores.
    3. **Base de datos Context Broker**: Base de datos MongoDB, la cual almacenará toda la información de contexto enviada al OCB y las subscripciones realizadas.
- **Controlador**: encargado de la gestión de eventos y comunicaciones entre la vista y el modelo. Se ha optado por el uso de un único Servlet que ejerce de controlador, el cual escucha las peticiones que realizamos al introducir los datos en los formularios.
- **Vista**: interfaz de usuario que presenta la información procedente del modelo. Presentará distintas opciones, según la información del modelo que se desee obtener.

La parte Backend de la aplicación se encarga de ejercer de controlador para poder determinar qué información del modelo es requerida por el usuario. Esto es posible gracias a la tecnología Servlet y al uso de sesiones Java, lo cual detallaremos en profundidad en las siguientes subsecciones.

7.1.1 Servlet: Controlador.java

La clase Controlador.java será la encargada de recibir las peticiones realizadas por los formularios, bien sea el formulario de acceso o los formularios de alta, baja y modificación de usuarios. Estas peticiones serán tratadas en los métodos doPost y doGet en donde se obtendrá el parámetro "accion" de la petición y su valor, el cual permitirá identificar que acción se debe realizar en función de la petición solicitada.

En la ilustración 64 se puede observar como se obtiene el parámetro "accion" de la petición "request" y se comprueba si su valor es "Ingresar", lo cual indicaría que un usuario ha introducido sus credenciales de usuario y desea autenticarse en el sistema.

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    doPost(request, response);

}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

                // Una vez pulsemos input de index.jsp con nombre accion comprobamos si su valor es Ingresar
                response.setContentType("text/html;charset=UTF-8");
                String accion = request.getParameter("accion");

                // En caso de ser Ingresar significará que se ha pulsado el botón, por lo que hacemos la petición http al endpoint del SW Restful para valida
                if (accion.equals("Ingresar")) {
```

Ilustración 64 Método doPost Controlador.java

De esta forma el controlador puede identificar que acción se debe realizar en cada consulta y la información del modelo que se solicita, siendo el servlet Controlador.java mapeado en @WebServlet("/Controlador").

A continuación, se detallan los posibles valores que el controlador aceptará como válidos para proceder a realizar determinadas acciones:

- **Ingresar**: Un usuario ha rellenado el formulario de acceso y pretende ingresar en el sistema, por lo tanto, se debe confirmar las credenciales introducidas por el usuario para poder emitir la decisión de acceso.
- **Crear**: El administrador ha rellenado el formulario de alta de usuario.
- **Eliminar**: El administrador ha rellenado el formulario de baja de usuario.
- **Modificar**: El usuario o administrador ha rellenado el formulario de información de su información personal.
- **Solicitar:** El usuario o administrador ha rellenado el formulario para solicitar información acerca de una matrícula determinada.

Una vez la clase Controlador.java es capaz de discernir la acción requerida, el proceso a realizar conllevará el envío de una petición Http al servicio web Keyrockservice en caso de existir una comunicación con Keyrock o a PEP-Proxy Wilma en caso de que se solicite información de contexto referente a los vehículos de un usuario.

```
// Creamos el httpClient a usar
CloseableHttpClient httpClient = HttpClientBuilder.create().build();

// Hacemos la petición POST al endpoint del SW Rest encargado de pedir el token de acceso
HttpPost peticion = new HttpPost("http://localhost:8080/keyrock/getToken");
StringEntity params = new StringEntity(myString);
// Añadimos todos los campos necesarios a la peticion POST y la ejecutamos finalmente
peticion.addHeader("content-type", "application/json");
peticion.setEntity(params);

// Obtenemos la respuesta del servicio web para ver si el usuario es autorizado o no
HttpResponse respuesta=httpClient.execute(peticion);

// Creamos un httpentity para poder obtener el body de la respuesta
HttpEntity httpEntity = respuesta.getEntity();
    // Comprobamos si hay respuesta
    if (httpEntity != null) {
```

Ilustración 65 Librería Apache HTTP

Para realizar las peticiones Http necesarias al servicio web Keyrockservice se ha optado por el uso de la librería Apache HTTP Client [12], tal y como se muestra en la ilustración 65. Concretamente se hace uso de las clases HttPost para poder crear la petición POST, la clase HttpResponse para poder obtener la respuesta y HttpEntity para poder obtener los parámetros de interés de la respuesta.

7.1.2 Descriptor de implementación: web.xml

Un aspecto muy importante en una aplicación web es especificar cómo se asignan las URL en los Servlets y qué URL requieren determinada autorización. Estos objetivos pueden ser logrados mediante el uso de un descriptor de implementación de una aplicación web llamado web.xml.

La importación de un descriptor de implementación de una aplicación web radica en la descripción de clases, recursos y configuración de la aplicación, indicando la forma en que el servidor web los usará para entregar las correspondientes solicitudes.

Un aspecto importante de un descriptor de implementación es el mapeo de Servlets, asignando una ruta URL con un Servlet, el cual manejará las solicitudes realizadas a dicho endpoint. Esta asignación se realiza mediante las etiquetas <servlet> y <servlet-mapping>.

```
<servlet>
    <servlet-name>redteam</servlet-name>
    <servlet-class>mysite.server.TeamServlet</servlet-class>
    <init-param>
        <param-name>teamColor</param-name>
        <param-value>red</param-value>
    </init-param>
    <init-param>
        <param-name>bgColor</param-name>
        <param-value>#CC0000</param-value>
    </init-param>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>redteam</servlet-name>
    <url-pattern>/red/*</url-pattern>
</servlet-mapping>
```

Ilustración 66 Uso de Etiquetas <servlet> y <servlet-mapping>

En la ilustración 66 se observa un ejemplo de mapeo del Servlet llamado redteam en la url "/red/*" gracias al uso de la etiqueta <servlet-mapping>.

El siguiente punto a destacar de un descriptor de implementación es la de implementar ciertas medidas de seguridad y autenticación en Java, adicionales a las medidas de seguridad que puede ofrecer un Gestor de Identidad.

Mediante el descriptor de fichero se crearán distintas restricciones de seguridad para poder hacer uso de roles en Java, los cuales serán identificados mediante sesiones por cada usuario. Una restricción de seguridad se define mediante un elemento <security-constraint>, definiendo una URL a proteger y los roles permitidos para dicha URL.

```
<security-constraint>
  <display-name>Contenido Seguro de Usuario</display-name>

  <web-resource-collection>
    <web-resource-name>Usuario estandar</web-resource-name>
    <url-pattern>/user/usuario/*</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <role-name>usuario</role-name><!--Roles permitidos a esta rutaAproteger-->
  </auth-constraint>
</security-constraint>
```

```
<security-role>
  <role-name>usuario</role-name>
</security-role>
```

Ilustración 67 Restricciones de Seguridad en web.xml

En la Ilutración 67 se puede observar la creación de un <security-role> llamado usuario y la creación de una restricción de seguridad <security-constraint>, la cual especifica la ruta /user/usuario/* indicando que para poder visualizar la información de esa dirección se requiere un rol usuario.

Quedaría por resolver cómo podemos asignar un rol a un usuario dentro de la aplicación. La asignación de roles se realiza en el Servlet controlador mediante sesiones Java, almacenando el rol de dicho usuario hasta que la sesión correspondiente sea invalidada. Para la asignación de la sesión se utilizan los métodos getSession() y SetAttribute(). El procedimiento a seguir será el siguiente:

Se comprueba el tipo de usuario en la base de datos de Keyrock.

- Una vez tenemos la información procedente de Keyrock se asigna la sesión Java correspondiente al usuario en el Servlet controlador.
- Se redirige a la interfaz correspondiente del tipo de usuario, el cual tendrá acceso a la información de una ruta protegida en función de su usuario.
- Para cerrar la sesión se debe pulsar el botón salir sesión del margen inferior izquierdo.

```
// También debemos de configurar la sesión correspondiente al usuario administrador
HttpSession sesion=request.getSession(true);

sesion.setAttribute("admin",mail);
```

Ilustración 68 Asignación de rol a una sesión Java

En la ilustración 68 se observa la creación de una nueva sesión para un usuario autenticado exitosamente, asignando un rol u otro dependiendo de sus credenciales.

Mediante el uso de las sesiones se logra denegar el acceso de un usuario estándar a la información y opciones de la interfaz de administrador y viceversa, eliminándose la sesión en caso de pulsar el botón de cierre de sesión.

La entrada a nuestra aplicación también debe ser especificada en el descriptor de fichero, así como la vista que se mostrará en caso de un error en la autenticación del usuario, tal y como se muestra en la ilustración 69.

```
<login-config>
   <auth-method>FORM</auth-method> <!--Metodo de autenticacion por formulario-->

   <form-login-config>
     <form-login-page>/index.jsp</form-login-page><!--Estos archivos estan fuera de la ruta a proteger-->
     <form-error-page>/error.jsp</form-error-page>
   </form-login-config>
</login-config>
```

Ilustración 69 Uso de etiqueta <login-config> en web.xml

El último punto a detallar será la invalidación de sesiones de los usuarios. Para esta tarea se ha creado un segundo Servlet, al cual se hará una petición al pulsar el botón de cierre de sesión por parte de un usuario.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    HttpSession sesion=request.getSession();
    sesion.invalidate();

    request.getRequestDispatcher("index.jsp").forward(request, response);
}
```

Ilustración 70 Servlet CerrarSesion.java

El servlet CerrarSesion.java, el cual es mostrado en la ilustración 70 invalidará la sesión con el método invalidate() y redirigirá al formulario de acceso index.jsp, imposibilitando el acceso del usuario al sistema sin introducir sus credenciales de nuevo.

7.2 Aplicación web: Frontend

Se denomina Frontend a la parte de una aplicación o programa a la que los usuarios tienen acceso de forma directa, abarcando todas las tecnologías de diseño y desarrollo web ejecutadas en los navegadores web y que se encargan de interactuar con el usuario.

Una vez se ha detallado la secuencia seguida por los componentes para la autenticación de un usuario, se debe detallar la interfaz de usuario y tecnologías que posibilitan el correcto funcionamiento del control de acceso y selección de opciones por parte de un usuario.

7.2.1 Interfaz de Usuario

Se procede a detallar en la siguiente subsección la interfaz de usuario, la cual ha sido diseñada mediante los lenguajes Cascading Style Sheets (CSS), JavaScript y HTTP. También se ha hecho uso de la biblioteca multiplataforma Bootstrap para el diseño de ciertos aspectos gráficos.

A continuación, se enumera el orden lógico de procedimiento para acceder al sistema:

1. Introducción de credenciales de usuario (correo y contraseña) en el formulario de acceso para poder acceder al sistema en caso de ser autorizado, el cual se muestra en la ilustración 71.

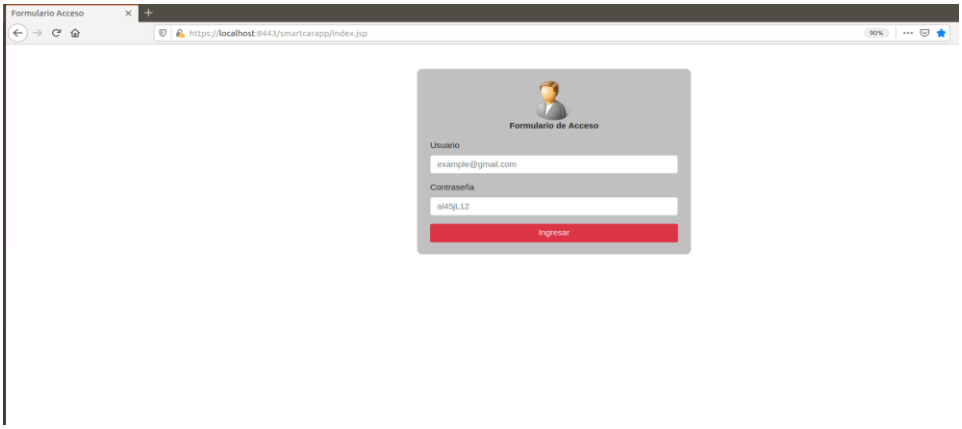


Ilustración 71 Formulario de Acceso

2. Comprobación de la información de usuario mediante nuestro Backend, distinguiendo entre usuario estándar y administrador (todo el proceso referente se detalla mediante un diagrama de secuencia en la sección 5.1).
3. Selección de la opción deseada mediante el menú de opciones interactivo.

Una medida de seguridad básica es el uso de protocolos de transferencia segura, siendo necesario el uso del protocolo HTTPS para las comunicaciones entre los clientes y el servidor web Apache Tomcat v8.50. El procedimiento para la habilitación de HTTPS en nuestro servidor será abordado en el Anexo C de la memoria

En el punto 5 se detalló los distintos actores del sistema, por lo que a continuación se presentarán las distintas funcionalidades de el área de un usuario básico y el de un usuario administrador.

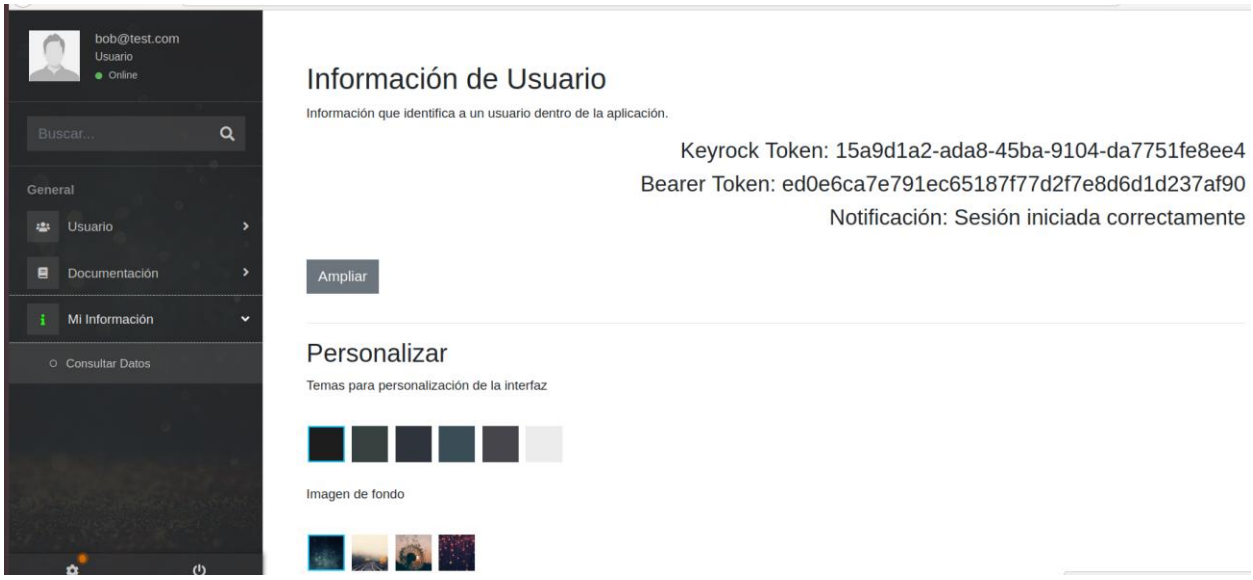


Ilustración 72 Interfaz de Usuario Básico

La interfaz para un usuario básico, la cual se muestra en la ilustración 72 presenta las siguientes funcionalidades:

- Visualización de la información básica de usuario, Keyrock Token y Bearer Token obtenida del componente Gestor de Identidad Keyrock.
- Manual de usuario.
- Personalización de interfaz de usuario.
- Acceso a la información referente a los vehículos dados de alta para un usuario, los cuales son identificados mediante sus matrículas.
- Modificación de datos de usuario.

Sin embargo, la interfaz de administrador, la cual se muestra en la ilustración 73 añade funcionalidades adicionales para permitir la creación de nuevos usuarios y la visualización de toda la información:

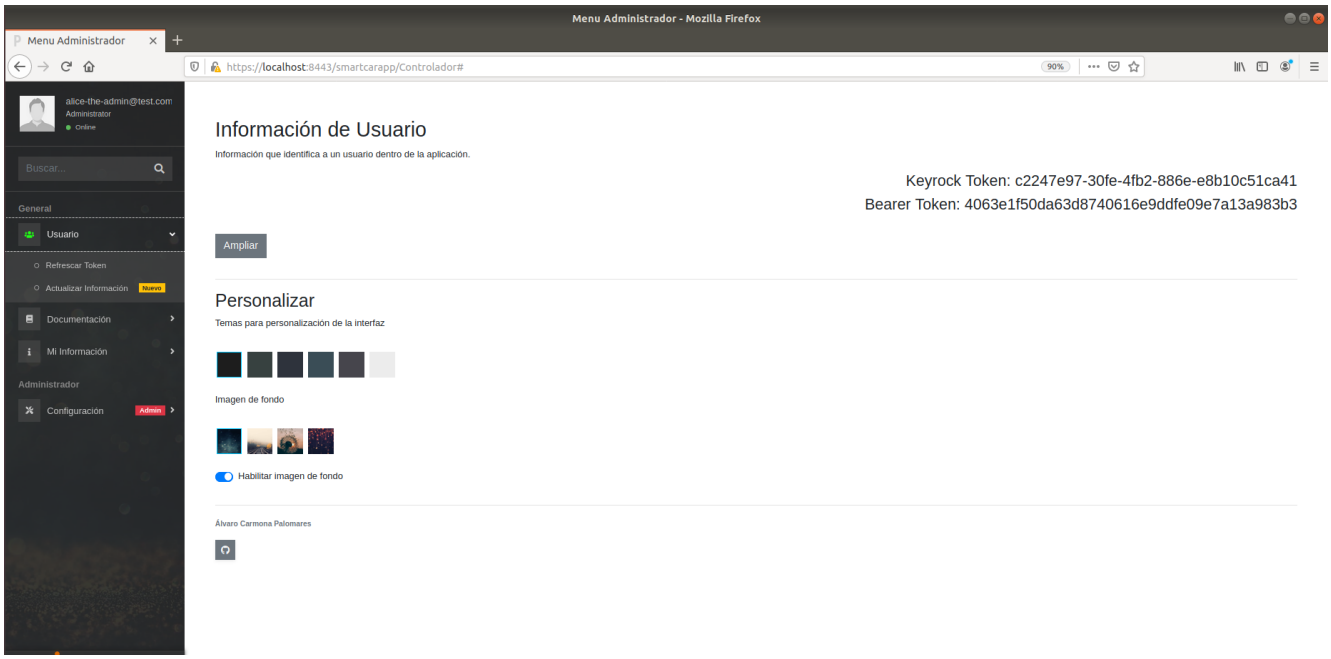


Ilustración 73 Interfaz de Usuario Administrador

- Visualización de la información básica de usuario, Keyrock Token y Bearer Token obtenida del componente Gestor de Identidad Keyrock.
- Manual de usuario y administrador.
- Personalización de interfaz de administrador.
- Acceso a toda la información de contexto almacenada en la base de datos PostgreSQL.
- Opciones de alta de usuario en el sistema y modificación de usuario.
- Modificación de datos de administrador.

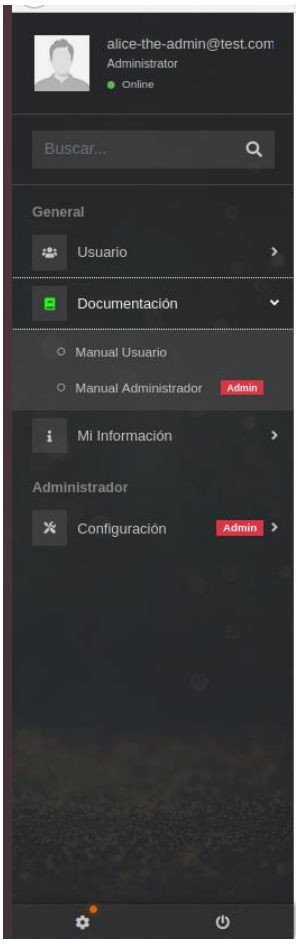


Ilustración 74 Menu lateral desplegable de lainterfaz

Para la selección de las opciones detalladas se dispone de un menú lateral, el cual hará uso de métodos JavaScript para poder mostrar las distintas opciones disponibles en función de la opción del menú lateral seleccionada, tal y como se muestra en la ilustración 74.

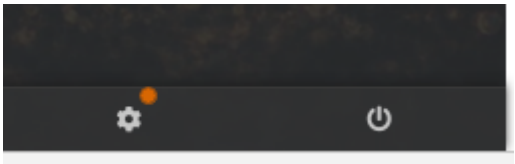


Ilustración 75 Botón Configuración de Usuario y Cerrar Sesión

Finalmente, se dispondrá de dos botones en el margen inferior izquierdo, los cuales se muestran en la ilustración 75, siendo el primero para acceder a la configuración de usuario y el segundo para cerrar sesión. En caso de cerrar sesión se redirigirá al usuario al formulario de acceso y se eliminará su sesión de Java, impidiendo que el usuario pueda volver a cargar la interfaz de usuario sin pasar previamente por el formulario de acceso.

7.2.2 Interfaz de Administrador: Opciones

Una vez se ha comentado las opciones en la subsección anterior las opciones habilitadas en función del tipo de usuario, se debe profundizar en las opciones permitidas para el usuario administrador de gestión de usuarios, las cuales tienen gran repercusión en el sistema.

Para dar de alta un nuevo usuario, el administrador debe cumplimentar un formulario con los datos del usuario a crear, indicando los siguientes valores:

- Nombre: Nombre del usuario o username que se dará de alta en Keyrock al crear el usuario.
- Contraseña: Contraseña del usuario que se creará en Keyrock.
- Email: Correo con el que se iniciará sesión en Keyrock.
- Tipo de vehículo: Tipo que se añadirá a la base de datos PostgreSQL.
- Matrícula: Matrícula que se añadirá a la base de datos PostgreSQL.

Ilustración 76 Formulario para la Creación de Usuario

El formulario a cumplimentar para el alta de un nuevo usuario se indica en la ilustración 76. La opción de eliminar un usuario tendrá el efecto opuesto a la opción de alta de usuario, siendo el formulario a cumplimentar idéntico al indicado en la ilustración 76.

Un punto a destacar en el alta y baja de un usuario serán los efectos que tendrán estas acciones respecto de la persistencia de la información de nuestro sistema. La primerá acción será la creación o eliminación del usuario correspondiente en la base de datos gestionada por MySQL, la cual contiene toda la información correspondiente a Keyrock. La segunda acción será la eliminación de toda la información asociada al usuario y sus vehículos de la base de datos gestionada por PostgreSQL.

8 CONCLUSIONES Y LÍNEAS FUTURAS

La mente es como un paracaídas… Solo funciona si la tenemos abierta.

- Albert Einstein -

El sistema detallado durante toda la memoria ha intentado recrear al lector un sistema real con una aplicación en el mundo del Internet de las Cosas. No obstante, un sistema complejo abarca muchas características que ofrecen un margen de mejora de cara a futuras implementaciones o comercializaciones del sistema.

Uno de los aspectos más importantes en un sistema formado por un gran número de componentes son los fallos o bugs que pueden ocurrir por causas ajenas al propio sistema o por la interacción inadecuada de algunos componentes. Dentro de este marco de errores que no han podido ser subsanados debido a la extensa duración del proyecto para un Trabajo Fin de Grado se deben mencionar:

Errores en las subscripciones: El componente FIWARE Orion Context Broker manejará grandes cantidades de información, por lo que no está exento de errores en el almacenamiento de ésta. Debido a esta razón podrían darse errores en las subscripciones por algún error interno del componente Orion Context Broker, lo cual hace necesario un mecanismo de confirmación periódica de subscripciones, pudiendo detectar si determinadas subscripciones a entidades han tenido algún error y logrando subscribirse nuevamente a dichas entidades.

Confirmación periódica de tiempo de vida de un token: Otro aspecto importante es el tiempo de vida de un token solicitado a Keyrock. El tiempo de vida de un token es de 1h, lo cual implica que es necesario determinar si un token es invalido o válido cada vez que se solicite realizar una operación.

Limitar el acceso a los servicios web a los puertos estrictamente necesarios: Otra medida necesaria es limitar el acceso de los servicios web del exterior o de otros componentes cuya funcionalidad no implique la comunicación con los mismos.

Otro aspecto a destacar es la información manejada, siendo esta inferior respecto a la que se manejaba en el proyecto de Pablo Bermejo. Por lo tanto, en el futuro se deberían de incluir las mediciones directamente de los sensores instalados en los vehículos, así como aumentar la cantidad de información que un vehículo produce. Ejemplos de parámetros a introducir serían marcas de tiempo en las mediciones.

El último punto que permite un gran margen de mejora debido al amplio factor de diseño inherente en su creación será la aplicación web. La aplicación web actual presenta una interfaz de usuario completa, la cual permite la selección de múltiples opciones y customización. Sin embargo, esta interfaz puede ser mejorada en gran medida, añadiendo mapas interactivos que indiquen todas las mediciones tomadas para el vehículo de un usuario o mostrar el estado de los sensores del vehículo de un usuario, entre otras mejoras.

Por último, será interesante hacer un planteamiento a futuro del sistema, estudiando qué tecnologías permiten una mejor escalabilidad del sistema a medida que se requieran más recursos y se den de alta a más usuarios.

ANEXO A: INSTALACIÓN DE LOS COMPONENTES FIWARE

En este anexo se detallará paso a paso la instalación de los Fiware Generic Enablers implementados en el presenten proyecto, los cuales han sido detallados extensamente en la memoria.

Los requisitos para la instalación pueden ser encontrados en la página oficial de Fiware. En el marco de la memoria la instalación se ha llevado a cabo en una máquina virtual con sistema operativo Ubuntu 18.04 LTS, tal y como se muestra en la ilustración 77.

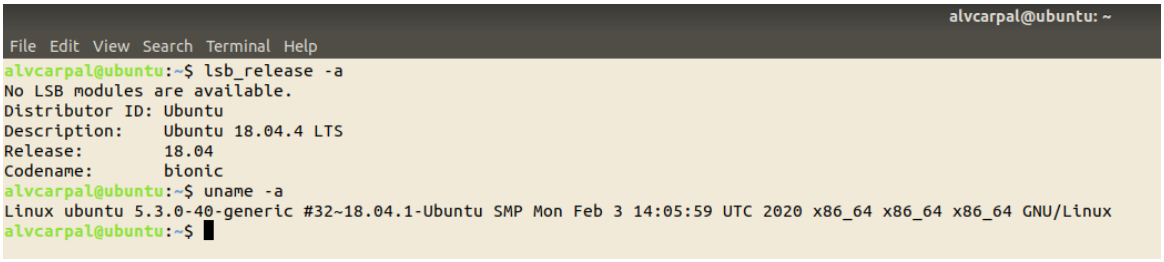


Ilustración 77 Sistema Operativo

También será necesario la instalación de Docker y Docker-compose, para los cuales serán necesarios unos recursos mínimos de 2 GB de RAM, 3 GB de espacio libre en disco y Kernel Linux 3.10 o superior.

A.1 Instalación de Docker y Docker-compose

La instalación de Docker y Docker-compose es obligatoria para poder desplegar el sistema, lo cual se detallo en el capítulo 3 de tecnologías usadas.

Para la instalación de la versión más reciente de Docker ejecutar los siguientes comandos:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"

$ sudo apt-get update


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

Para la instalación de docker-compose debemos ejecutar los siguientes comandos:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
$ docker-compose --version
```

Para comprobar la correcta instalación se comprueba la versión instalada ejecutando el comando indicado en la ilustración 78:

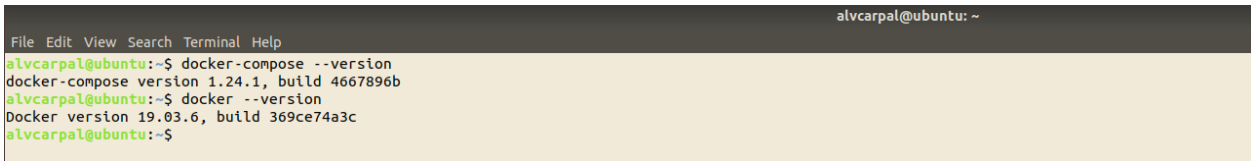


Ilustración 78 Versión de Docker y Docker-compose

A.1.1 Error con el demonio Docker

Un error común que podrá ocurrir durante el uso de Docker es el error "Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?", lo cual indica que el Docker Daemon no se ejecuta correctamente.

Para solucionar este problema debemos reiniciar el Docker Daemon con el siguiente comando:

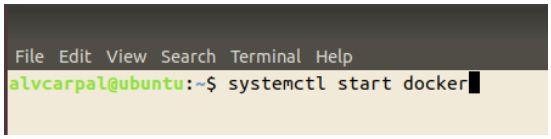


Ilustración 79 Reiniciar Docker Daemon

A.2 Instalación y despliegue Fiware Enabler idm Keyrock

Para la correcta instalación de Keyrock se deberá instalar Docker-compose y Docker. Todo el software necesario será desplegado mediante Docker, por lo que se debe de crear un directorio y descargar el fichero Docker-compose.yml de la información oficial de Docker [13].

Es importante que el archivo. yml descargado tenga el nombre 'docker-compose.yml', ya que éste es el archivo base usado por docker-compose para poder ejecutar los servicios que conforman la aplicación.

A.2.1 Configuración Docker-compose.yml

Se deberá modificar el contenido de docker-compose.yml para adecuarlo a las necesidades y características de nuestra aplicación:

```yaml
version: "3.5"
services:

    # Keyrock is an Identity Management Front-End
    keyrock:
        image: fiware/idm:7.6.0
        container_name: fiware-keyrock
        hostname: keyrock
        networks:
            default:
                ipv4_address: 172.18.1.5
        depends_on:
            - mysql-db
        ports:
            - "3005:3005"
        environment:
            - DEBUG=idm:*
            - IDM_DB_HOST=mysql-db
            - IDM_HOST=http://localhost:3005
            - IDM_PORT=3005
            - IDM_HTTPS_PORT=3443
            # Development use only
            # Use Docker Secrets for Sensitive Data
            - IDM_DB_PASS=secret
            - IDM_DB_USER=root
            - IDM_ADMIN_USER=alice
            - IDM_ADMIN_EMAIL=alice-the-admin@test.com
            - IDM_ADMIN_PASS=test

    # Databases
    mysql-db:
        restart: always
        image: mysql:5.7
        hostname: mysql-db
        container_name: db-mysql
        expose:
            - "3306"
        ports:
            - "3306:3306"
        networks:
            default:
                ipv4_address: 172.18.1.6
        environment:
            # Development use only
            # Use Docker Secrets for Sensitive Data
            - "MYSQL_ROOT_PASSWORD=secret"
            - "MYSQL_ROOT_HOST=172.18.1.5"
        volumes:
            - mysql-db:/var/lib/mysql

networks:
    default:
        ipam:
            config:
                - subnet: 172.18.1.0/24
volumes:
    mysql-db: ~
```

Ilustración 80 Archivo docker-compose.yml para Keyrock

En la ilustración 80 se muestra el archivo ‘docker-compose.yml’, el cual contiene toda la configuración de Keyrock detallada en la siguiente tabla:

Key	Valor	Descripción
IDM_HOST	http://localhost:3005	URL en la cual escucha peticiones el IdM Keyrock (se debe especificar el puerto Http).
IDM_PORT	3005	Puerto en el que Keyrock escucha peticiones Http.
IDM_HTTPS_PORT	3443	Puerto en el que Keyrock escucha peticiones Https.

IDM_ADMIN_USER	alice	Nombre del usuario root creado en la base de datos Mysql (primer usuario a crear en la base de datos).
IDM_ADMIN_EMAIL	alice-the-admin@test.com	Correo del usuario root creado.
IDM_ADMIN_PASS	test	Contraseña del usuario root creado,

Tabla 5 Configuración Keyrock

Durante la configuración de los parámetros correspondientes es importante confirmar si los puertos seleccionados no están siendo usados por otros procesos antes de ser asignados a Keyrock o a Mysql. También se debe indicar en el fichero docker-compose.yml el puerto, host y contraseña de mysql-db.

Para poder iniciar ambos componentes se ejecuta el comando “sudo docker-compose up” tal y como se muestra en la ilustración 81.

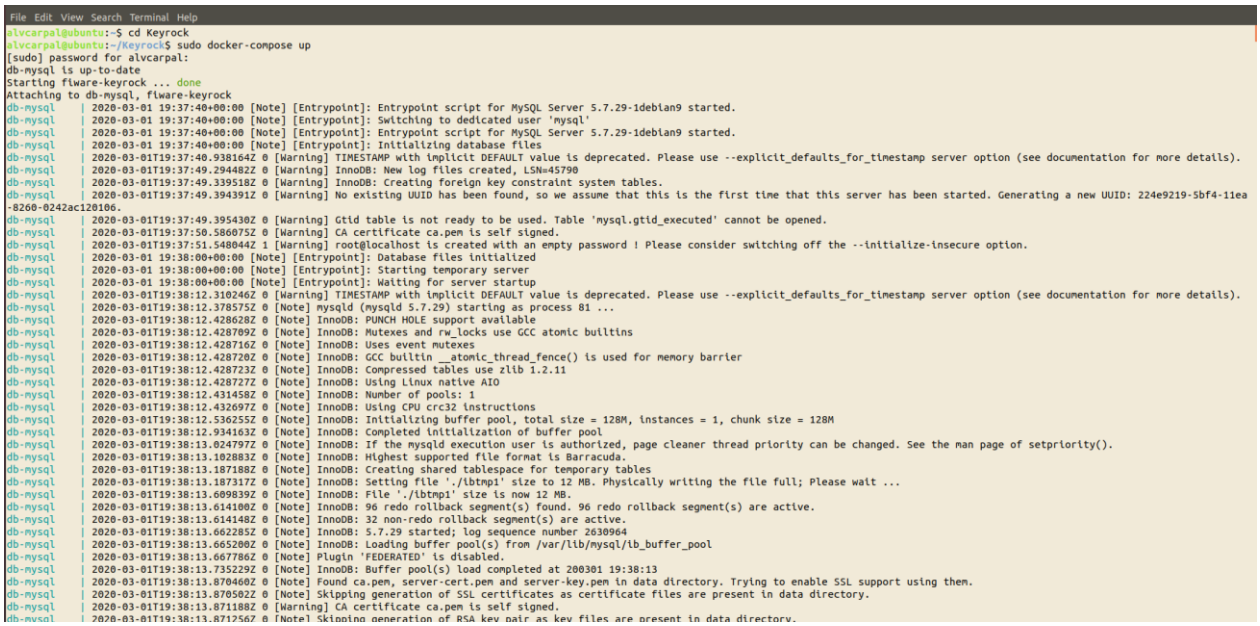


Ilustración 81 Inicialización Keyrock y Mysql-db

El tiempo aproximado para que se establezca una conexión correctamente es de 60 segundos.

A.2.2 Conexión con la Base de Datos MySQL

Todas las peticiones que se realicen a Keyrock se traducirán en consultas SQL a la base de datos MySQL relacionada con nuestro IdM según los parámetros que hemos configurado. Las consultas a la base de datos pueden ser consultas indirectas, es decir hacer peticiones a la interfaz de usuario de Keyrock y éste hará las peticiones correspondientes a la base de datos.

Para ejemplificar esta relación, crearemos un usuario directamente a través de la interfaz de usuario que Keyrock nos ofrece, lo cual se muestra en la ilustración 82.

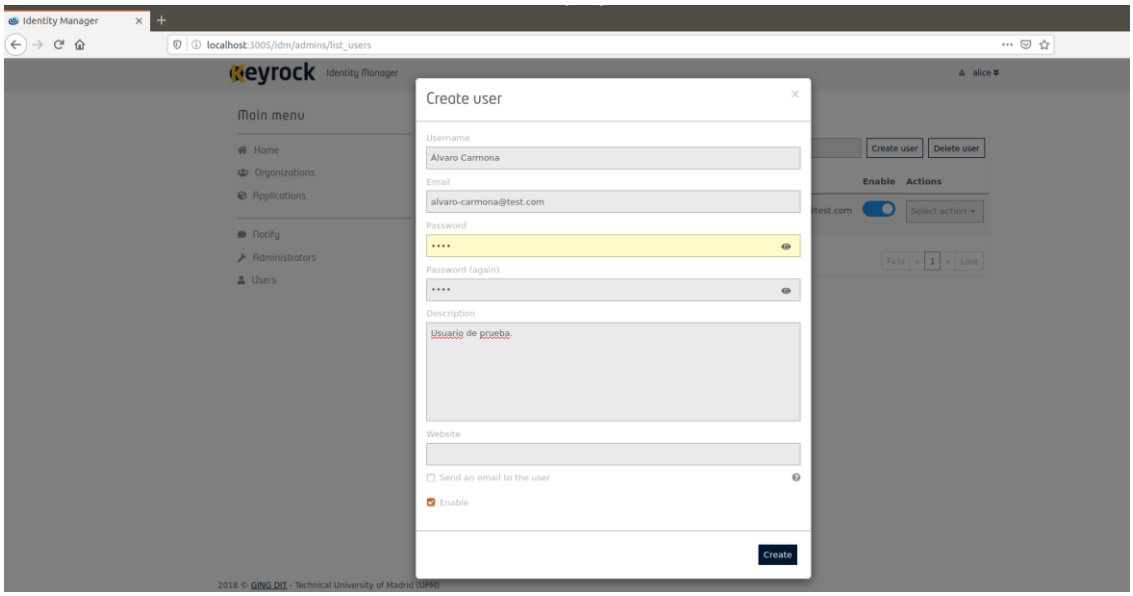


Ilustración 82 Crear usuario mediante interfaz de Keyrock

Una vez creado el usuario se comprueba la terminal, observando la petición realizada por Keyrock a mysql-db tal y como se muestra en la ilustración 83.

```
fiware-keyrock | [sass]  skip: /img/logos/medium/user.png nothing to do
fiware-keyrock | GET /img/logos/medium/user.png 304 0.636 ms - -
fiware-keyrock | Tue, 10 Mar 2020 09:28:48 GMT idm:web-session_controller --> login_required
fiware-keyrock | Tue, 10 Mar 2020 09:28:48 GMT idm:web-session_controller --> password_check_date
fiware-keyrock | Tue, 10 Mar 2020 09:28:48 GMT idm:web-admin_controller --> is_admin
fiware-keyrock | Tue, 10 Mar 2020 09:28:48 GMT idm:web-list_users_controller --> create
fiware-keyrock | Executing (default): SELECT `id`, `username`, `description`, `website`, `image`, `gravatar`, `email`, `salt`, `password`, `date_password`, `enabled`, `admin`, `starters_tour_ended`, `e
s_id`, `extra`, `scope` FROM `user` AS `User` WHERE `User`.`email` = 'alvaro-carmona@test.com';
fiware-keyrock | Executing (default): SELECT `id`, `username`, `description`, `website`, `image`, `gravatar`, `email`, `salt`, `password`, `date_password`, `enabled`, `admin`, `starters_tour_ended`, `e
s_id`, `extra`, `scope` FROM `user` AS `User` WHERE `User`.`email` = 'alvaro-carmona@test.com';
fiware-keyrock | Executing (default): INSERT INTO `user` (`id`,`username`,`description`,`website`,`image`,`gravatar`,`email`,`salt`,`password`,`date_password`,`enabled`,`admin`,`starters_tour_ended`,`e
s_id`) VALUES ('2b3e9018-de18-46ac-87d5-364a134c6b34','Álvaro Carmona ','Usuario de prueba.','','default',false,'alvaro-carmona@test.com','d7d657b09ceb1bd5','ddcba44dcf37231169d5723b7792119a05660fc7','
0-03-10 09:28:48',true,false,false,NULL);
fiware-keyrock | POST /idm/admins/list_users/users 200 37.265 ms - 226
```

Ilustración 83 Consulta a mysql-db

La otra posibilidad es hacer peticiones directamente a la base de datos MySQL. Para ello debemos conectarnos usando una terminal e introduciendo nuestras credenciales configuradas en docker-compose.yml, lo cual se muestra en la ilustración 84.

```
File  Edit  View  Search  Terminal  Help
alvcarpal@ubuntu:~$ docker exec -it db-mysql bash
root@mysql-db:/# mysql -u root -psecret idm
mysql: [Warning] Using a password on the command line interface can be insecure.
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 5.7.29 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Ilustración 84 Acceso a mysql-db

A.3 Instalación y despliegue PEP-Proxy Wilma

A pesar de las ventajas que nos ofrece Docker de cara al escalado de nuestro software, se ha decidido optar por el segundo método de instalación, para el que será necesario los siguientes requisitos:

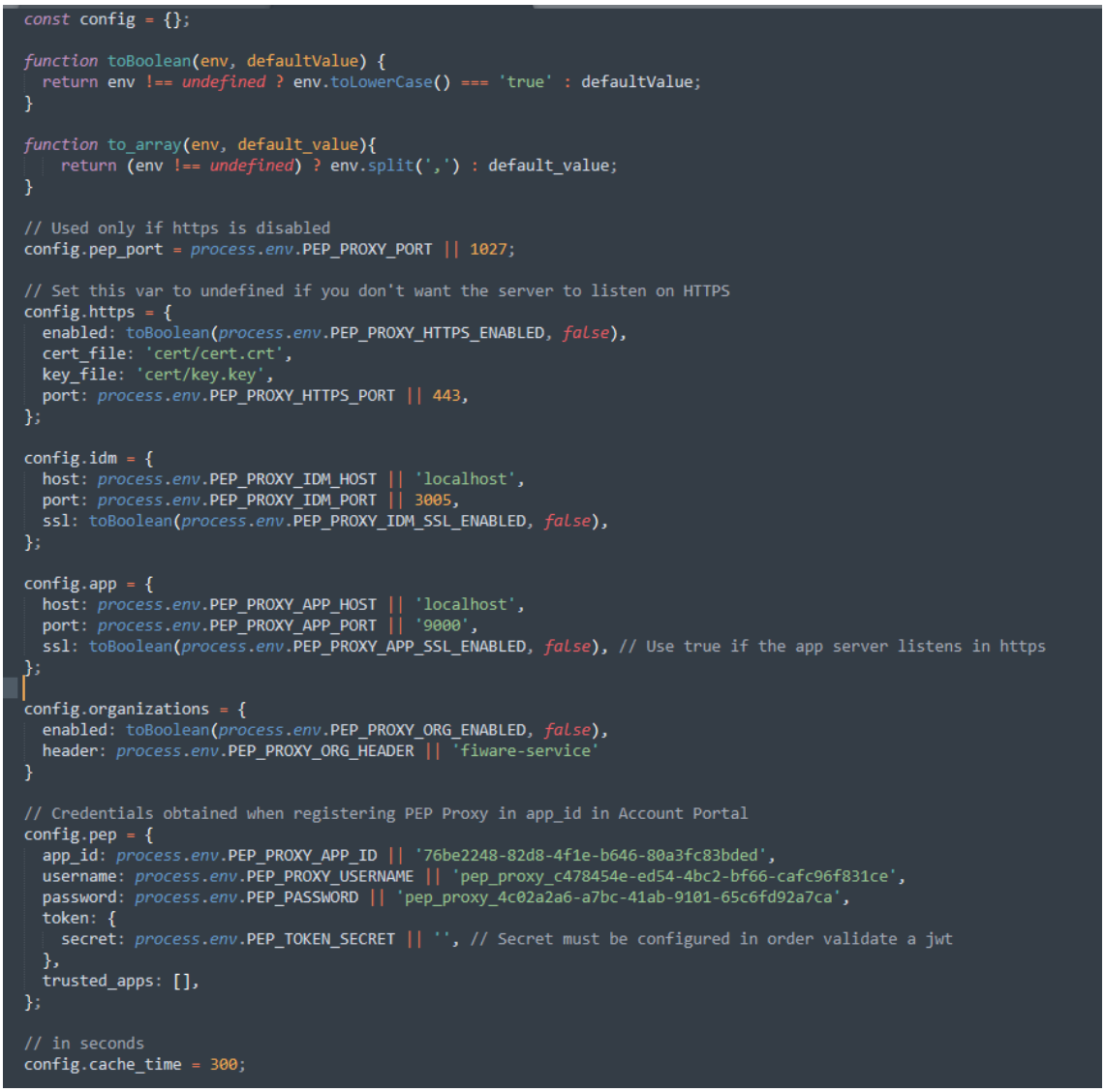
- nodejs >= 8.x.x
- npm >= 5.x.x

Para la instalación de nodejs y npm ejecutamos los siguientes comandos (en caso de no tenerlos instalados previamente o tener versiones inferiores a las requeridas):

```
En el momento del tutorial nodejs 12 es la versión para LTS:

$ sudo apt-get install curl

$ curl -sL https://deb.nodesource.com/setup_13.x | sudo -E bash -

$ sudo apt-get install nodejs

$ sudo apt install npm



Comprobamos las versions instaladas:

$ node -v

$ npm -v
```

Una vez instalados node.js y npm procedemos a clonar un repositorio de Github [14] con los archivos necesarios para el despliegue de Wilma. Todo el paso para poder clonar y configurar el directorio de forma adecuada se detallan en el propio repositorio.

La configuración de PEP-Proxy Wilma será la siguiente, la cual se muestra en la ilustración 85:



```
const config = {};

function toBoolean(env, defaultValue) {
  return env !== undefined ? env.toLowerCase() === 'true' : defaultValue;
}

function to_array(env, default_value){
    return (env !== undefined) ? env.split(',') : default_value;
}

// Used only if https is disabled
config.pep_port = process.env.PEP_PROXY_PORT || 1027;

// Set this var to undefined if you don't want the server to listen on HTTPS
config.https = {
  enabled: toBoolean(process.env.PEP_PROXY_HTTPS_ENABLED, false),
  cert_file: 'cert/cert.crt',
  key_file: 'cert/key.key',
  port: process.env.PEP_PROXY_HTTPS_PORT || 443,
};

config.idm = {
  host: process.env.PEP_PROXY_IDM_HOST || 'localhost',
  port: process.env.PEP_PROXY_IDM_PORT || 3005,
  ssl: toBoolean(process.env.PEP_PROXY_IDM_SSL_ENABLED, false),
};

config.app = {
  host: process.env.PEP_PROXY_APP_HOST || 'localhost',
  port: process.env.PEP_PROXY_APP_PORT || '9000',
  ssl: toBoolean(process.env.PEP_PROXY_APP_SSL_ENABLED, false), // Use true if the app server listens in https
};

config.organizations = {
  enabled: toBoolean(process.env.PEP_PROXY_ORG_ENABLED, false),
  header: process.env.PEP_PROXY_ORG_HEADER || 'fiware-service'
}

// Credentials obtained when registering PEP Proxy in app_id in Account Portal
config.pep = {
  app_id: process.env.PEP_PROXY_APP_ID || '76be2248-82d8-4f1e-b646-80a3fc83bded',
  username: process.env.PEP_PROXY_USERNAME || 'pep_proxy_c478454e-ed54-4bc2-bf66-cafc96f831ce',
  password: process.env.PEP_PASSWORD || 'pep_proxy_4c02a2a6-a7bc-41ab-9101-65c6fd92a7ca',
  token: {
    secret: process.env.PEP_TOKEN_SECRET || '', // Secret must be configured in order validate a jwt
  },
  trusted_apps: [],
};

// in seconds
config.cache_time = 300;
```

Ilustración 85 Configuración PEP-Proxy Wilma

Key	Valor	Descripción
PEP_PROXY_PORT	1027	Puerto en el que escucha peticiones Http el PEP Proxy.
PEP_PROXY_HTTPS_PORT PEP_PROXY_IDM_SSL_ENABLED	443 false	Puerto en el que escucha peticiones Https el PEP Proxy. Se indica si Https está activo o no.
PEP_PROXY_IDM_HOST PEP_PROXY_IDM_PORT	localhost 3005	Nombre de host del IdM (Keyrock en nuestro caso) y puerto en el que escucha peticiones.

PEP_PROXY_APP_HOST PEP_PROXY_APP_PORT	localhost 9000	Nombre de host del servicio detrás del PEP Proxy y el puerto en el que escucha las peticiones que realice el PEP Proxy.
PEP_PROXY_APP_ID	76be2248-82d8-4f1e-b646-80a3fc83bded	ID de la aplicación que realiza las peticiones al PEP Proxy.
PEP_PROXY_USERNAME	pep_proxy_c478454e-ed54-4bc2-bf66-cafc96f831ce	Nombre de usuario para la instancia del creada PEP-Proxy.
PEP_PASSWORD	pep_proxy_4c02a2a6-a7bc-41ab-9101-65c6fd92a7ca	Contraseña para la instancia del PEP-Proxy creada.
PEP_TOKEN_SECRET		Token secret especificado en la interfaz de Keyrock.

Tabla 6 Configuración Wilma

Se debe prestar especial atención en los últimos cuatro parámetros a configurar, ya que son parámetros que debemos obtener de la interfaz de usuario de Keyrock y dependerán de la aplicación, variando entre creaciones de distintas aplicaciones.

Para poder saber el valor de estos parámetros debemos entrar con nuestro usuario root en Keyrock y crear una aplicación:

Se accede a la aplicación, en la cual obtendremos la siguiente información, la cual una vez creada aportará toda la información necesaria dentro del apartado PEP-Proxy, una vez se ha pulsado el botón de crear una nueva instancia de PEP-Proxy Wilma en el mismo apartado, tal y como se indica en la ilustración 86.

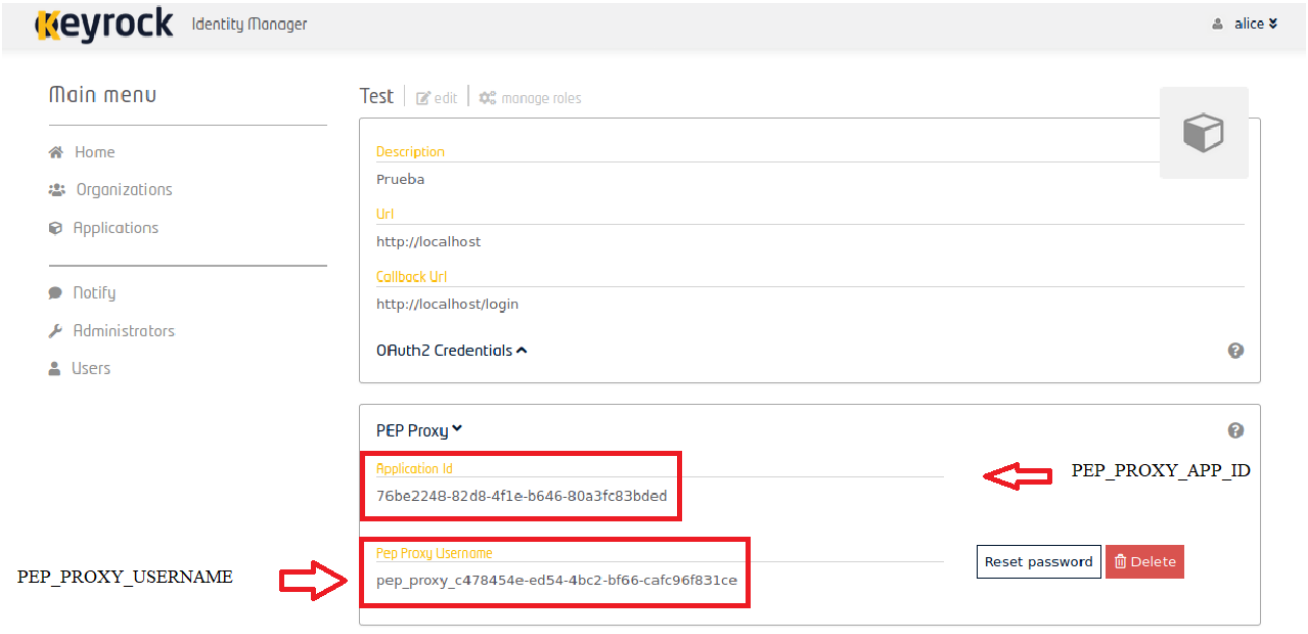


Ilustración 86 Parámetros de configuración PEP-Proxy Wilma

El parámetro PEP_PASSWORD se muestra únicamente cuando una instancia de PEP Proxy es creada para nuestra aplicación (si se accede por primera vez a la aplicación, el campo PEP Proxy aparece en blanco con la opción de crear una instancia mediante un botón). En caso de olvido de la contraseña, se puede volver a solicitar una nueva pulsando el botón 'Reset password'.

El valor del último parámetro, PEP_TOKEN_SECRET, dependerá de la opción seleccionada en Token Types dentro de la configuración de la aplicación creada.

En caso de seleccionar la opción 'Permanent', se dejará el valor en blanco, mientras que si selecciona la opción 'JSON Web Token' se debe aportar el valor del JWT secret generado, tal y como se muestra en la ilustración 87.

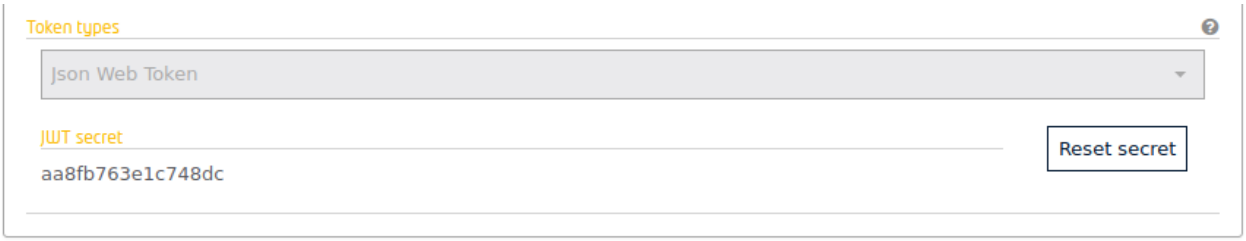


Ilustración 87 Token Types Keyrock

El último paso será iniciar PEP-Proxy Wilma mediante el comando "sudo node server" una vez se esté situado en el directorio creado con los ficheros de configuración previamente descargados. Se debe puntualizar que la instalación mediante Docker difiere de la instalación mostrada, debido a que la instalación mediante Docker es automática y no se debe de configurar ninguna instancia en la propia interfaz de Keyrock, tal y como se verá en la siguiente sección de instalación del Agente IoT Agent-Ul.

A.4 Instalación y despliegue Orion Context Broker y IoT Agent Ultralight

Para la instalación de Orion Context Broker se ha optado por una instalación mediante Docker-compose idéntica a la acometida en la instalación de Keyrock, lo cual facilitará en gran medida la instalación al evitar la configuración individualizada de todos los componentes que se relacionan con Orion Context Broker.

El procedimiento a seguir es similar, debiendo de descargarse el ficher docker-compose.yml mostrado en la ilustración 88 en un directorio previamente creado e inicializando los componentes una vez se han modificado los parámetros de configuración de interés.

```
version: "3.1"

volumes:
    mongodb: ~

services:
    iot-agent:
        image: fiware/iotagent-ul
        hostname: iot-agent
        container_name: fiware-iot-agent
        depends_on:
            - mongodb
        expose:
            - "4041"
            - "7896"
        ports:
            - "4041:4041"
            - "7896:7896"
        environment:
            - "IOTA_CB_HOST=orion"
            - "IOTA_CB_PORT=1026"
            - "IOTA_NORTH_PORT=4041"
            - "IOTA_REGISTRY_TYPE=mongodb"
            - "IOTA_MONGO_HOST=mongodb"
            - "IOTA_MONGO_PORT=27017"
            - "IOTA_MONGO_DB=iotagent-ul"
            - "IOTA_HTTP_PORT=7896"
            - "IOTA_PROVIDER_URL=http://iot-agent:4041"

    mongodb:
        image: mongo:3.6
        hostname: mongodb
        container_name: db-mongo
        ports:
            - "27017:27017"
        command: --bind_ip_all --smallfiles
        volumes:
            - mongodb:/data

    orion:
        image: fiware/orion
        hostname: orion
        container_name: fiware-orion
        depends_on:
            - mongodb
        expose:
            - "1026"
        ports:
            - "1026:1026"
        command: -dbhost mongodb
```

Ilustración 88 Fichero docker-compose.yml para la instalación de Orion

La estructura del fichero es similar a la ya vista para la instalación de Keyrock y musql-db, realizando una instalación estándar para los componentes Orion Context Broker y mongodb, por lo que solo se deberá de modificar los parámetros de configuración referentes a IoT Agent Ultralight de modo que pueda comunicarse correctamente con Orion Context Broker y Mongodb.

En la tabla 7 se indican los parámetros a modificar y sus valores estándar para el presente proyecto:

Key	Valor	Descripción
IOTA_CB_HOST	orion	Indica el nombre de la instancia de Orion Context Broker en el sistema, debiendo coincidir con el nombre de OCB en el ficher docker-compose.yml.
IOTA_CB_PORT	1026	Indica el puerto en el que escucha peticiones OCB.
IOTA_NORTH_PORT	4041	Indica el puerto norte del Agente IoT Agent-Ul que se comunicará con OCB.
IOTA_REGISTRY_TYPE	mongodb	Indica el tipo de registro donde se almacenará la información de los sensores.
IOTA_MONGO_HOST	mongodb	Indica el nombre de la base de datos MongoDB.
IOTA_MONGO_PORT	27017	Indica el puerto mediante el cual se comunicarán IoT Agent y MongoDB.
IOTA_HTTP_PORT	7896	Indica el puerto sur mediante el cual se comunicarán los sensores con el agent IoT.

Tabla 7 Configuración IoT Agent

El resto de los parámetros no mencionados pueden ser dejados con sus valores por defecto en el fichero docker-compose.yml descargado. El comando para iniciar los componentes es el visto en la sección A.2 “sudo docker-compose up”.

ANEXO B: INSTALACIÓN Y CONFIGURACIÓN SERVIDOR BASE DE DATOS POSTGRESQL

En este anexo se detalla la configuración del servidor base de datos PostgreSQL en el sistema operativo Ubuntu 18.04 LTS instalado para las pruebas del sistema.

```
$ sudo apt-get update
$ sudo apt-get -y install postgresql
```

Una vez se ha instalado postgresql en el sistema los ficheros de configuración del servidor base de datos podrán ser accedidos en la dirección /etc /postgresql/10/main, pudiendo variar la versión en función de cúando se realice la instalación.

El primer paso será acceder con el usuario por defecto creado postgres, al cual se podrá cambiar su contraseña mediante \password y crear un nuevo usuario propietario de la nueva base de datos a crear.

```
alvcarpal@ubuntu:~$ su postgres

postgres@ubuntu:/home/alvcarpal$ psql

postgres=# \password postgres

postgres=# CREATE USER admin WITH ENCRYPTED PASSWORD 'test'
CREATEDB;

postgres=# \q

postgres@ubuntu:/home/alvcarpal$ exit
```

Se crearía de este modo un nuevo usuario admin con la contraseña test.

Paso previo a la creación de la base de datos propietaria del usuario admin y al llenado mediante ficheros .sql, con la finalidad de recrear la base de datos detallada en el capítulo 6, será la configuración de determinados ficheros del servidor para permitir distintas acciones.

Los pasos a seguir serán los siguientes, ambos modificando el fichero pg_hba.conf:

1. Incluir la línea "local all all md5" en el fichero pg_hba.conf, permitiendo la autenticación por contraseña de los usuarios del sistema gestor de base de datos.
2. Incluir la línea "host all all 192.168.0.0/24 md5" permitiendo las peticiones de cualquier dispositivo dentro de la red 192.168.0.0/24.

```
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     md5
local   all             all                                     peer
# IPv4 local connections:
host    all             all             192.168.0.0/24          md5
host    all             all             0.0.0.0/0               md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5
```

Ilustración 89 Configuración del servidor base de datos

Ambas líneas se muestran en la ilustración 89, así como la posición exacta dentro del fichero, la cual será relevante. Tras realizar estos cambios, es necesario reiniciar el servicio mediante la ejecución como superusuario de la siguiente orden:

alvcarpal@ubuntu:~$ /etc/init.d/postgresql restart

3. Abrir el fichero postgresql, acción que requiere privilegios de superusuario y editar el parámetro listen_addresses cambiando su valor de “’localhost’” a “’*’” para permitir conexiones, tal y como se muestra en la ilustración 90. Se ha optado por el uso del puerto estándar 5432, pero en caso de necesitar ser cambiado puede ser modificado en el propio fichero postgresql.conf.

```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'                  # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
port = 5432                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                        # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                        # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
                                        # (change requires restart)
#bonjour_name = ''                      # defaults to the computer name
                                        # (change requires restart)

# - Security and Authentication -
```

Ilustración 90 Postgresql.conf

El siguiente paso será la creación de una base de datos, la cual será gestionada por el nuevo usuario admin creado previamente mediante la ejecución de los siguientes comandos mostrados en la ilustración 91.

```
alvcarpal@ubuntu:~$ psql -U admin -d postgres
Password for user admin:
psql (10.12 (Ubuntu 10.12-0ubuntu0.18.04.1))
Type "help" for help.

postgres=> CREATE DATABASE matriculas;
```

Ilustración 91 Creación Base de Datos Matrículas

Una vez se ha creado la base de datos, se podrá comprobar el éxito de la operación realizada mediante el comando \l, listando todas las bases de datos en el servidor y sus propietarios, tal y como se muestra en la ilustración 92.

```
matriculas=> \l
                                  List of databases
    Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
------------+----------+----------+-------------+-------------+-----------------------
 alvcarpal  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 matriculas | admin    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
            |          |          |             |             | postgres=CTc/postgres
 template1  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
            |          |          |             |             | postgres=CTc/postgres
(5 rows)
```

Ilustración 92 Bases de Datos en el servidor Postgres

Un último paso será la creación de las relaciones de nuestra base de datos. La creación de las distintas relaciones se podrá realizar de forma manual, según lo ilustrado en el capítulo 7 o de forma automática con el fichero Crea.sql, el cual se podrá encontrar en los recursos del proyecto [15].

Ahora el servidor ya es accesible desde cualquier dispositivo dentro de la red indicada y se podrá acceder a la base de datos matrículas mediante el usuario admin, la cual contendrá toda la información de contexto almacenada de los usuarios del sistema y los vehículos de cada usuario.

ANEXO C: INSTALACIÓN Y CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

El entorno de desarrollo elegido para la realización del proyecto ha sido Spring Tool Suite versión 3.9 (STS), el cual es un entorno de desarrollo basado en el IDE Eclipse, diseñado específicamente para el desarrollo de aplicaciones Springs. Estas características nos permitirán un desarrollo fácil y eficaz de nuestros servicios distribuidos REST haciendo uso del framework Sping.

Se debe mencionar que en el momento de realización de la presente memoria se ha lanzado la versión Spring Tool Suite 4, la cual ha sido probada en el marco del proyecto que nos compete. La versión mencionada es accesible a través de la dirección:

https://spring.io/tools3/sts/all

C.1 INSTALACIÓN STS 3.9

Los pasos a seguir serán los siguientes:

1. Acceder a la siguiente dirección y descargar STS correspondiente al sistema operativo:

   https://github.com/spring-projects/toolsuite-distribution/wiki/Spring-Tool-Suite-3

2. Descomprimir el archivo descargado y acceder a la carpeta sts-3.9.11 RELEASE:

3. Configurar nuestro directorio de trabajo, el cual nos será requerido la primera vez que iniciemos STS.

C.2 INSTALACIÓN LOS COMPONENTES NECEDARIOS PARA EL PROYECTO

El correcto funcionamiento del proyecto depende de una serie de componentes previos, los cuales se detallan a continuación.

C.2.1 JAVA EE

Para poder crear nuestra aplicación web es necesario la instalación previa de Java EE para poder crear proyectos web dinámicos.

Se debe acceder a la sección Eclipse Marketplace y descargar la extensión correspondiente. La cual se muestra en la ilustración 93.

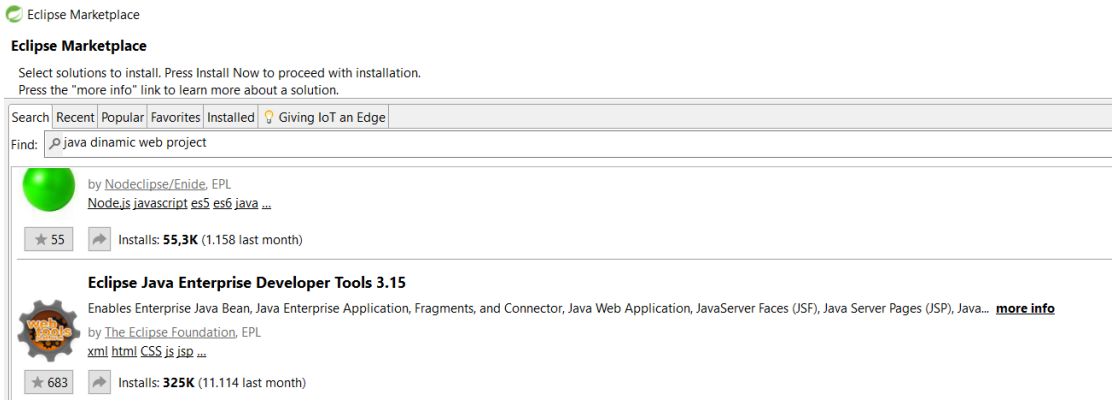


Ilustración 93 Extensión Java EE STS

C.2.2 INSTALACIÓN SERVIDOR APACHE TOMCAT V8.5

Un servidor web es necesario para poder ejecutar la aplicación web diseñada para el proyecto. Los pasos a seguir para la correcta instalación serán los siguientes:

1. Acceder a la siguiente dirección y descargar el archivo correspondiente de acuerdo a nuestro sistema operativo:

   https://tomcat.apache.org/download-90.cgi

2. Una vez completada la descarga, debemos crear un servidor en el entorno STS.

   Se deben seguir las instrucciones de instalación y especificar el directorio que hemos descargado previamente.

   Una vez especificado el directorio, se creará nuestro servidor, cuya configuración se muestra en la ilustración 94.

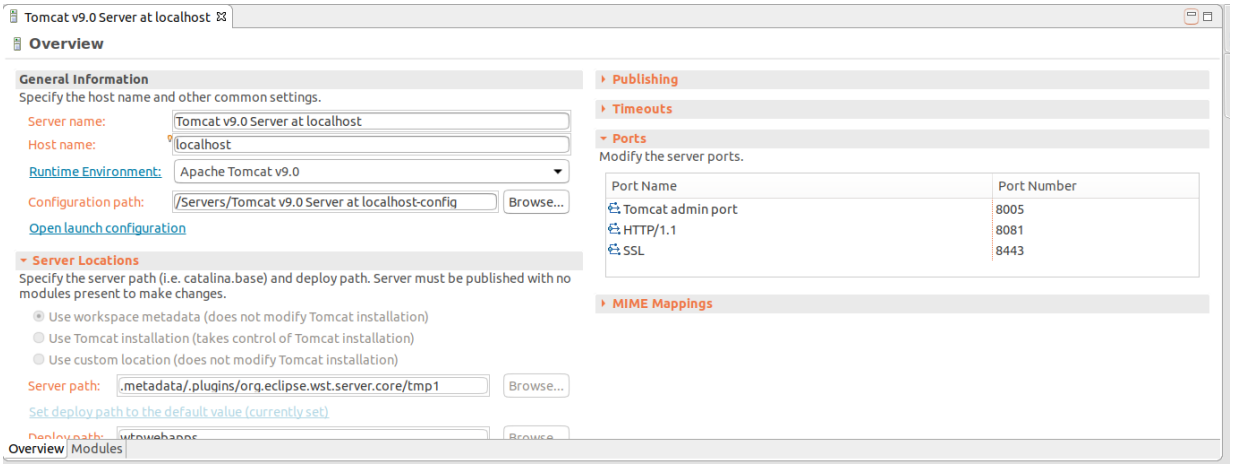


Ilustración 94 Configuración Servidor Apache TomcaT v8.5

C.2.2.1 HABILITAR EL PROTOCOLO HTTPS EN SERVIDOR TOMCAT V8.5

Visto la importancia del protocolo Https en nuestro proyecto, se procede a detallar cómo activar las peticiones Https en Tomcat v8.5. Para ello debemos seguir los siguientes pasos:

1. Acceder a la siguiente dirección, donde se detalla el procedimiento:

   https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html

   El proceso de habilitación de Https será el mismo para un servidor Apache Tomcat v9 y v8.5.

2. Se debe definir la variable de entorno $JAVA_HOME para poder crear una keystore, la cual almacenará las claves privadas del servidor y los certificados, haciendo uso de la herramienta keytool:

```
$ JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
```

3. El siguiente paso será generar una clave:

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
  -keystore /path/to/my/keystore
```

La información detallada se puede encontrar en la dirección indicada en el primer paso. Para finalizar la configuración debemos modificar el fichero de configuración de nuestro servidor, el cual se muestra en la ilustración 95.

```
<!-- Define an SSL/TLS HTTP/1.1 Connector on port 8443
     This connector uses the NIO implementation. The default
     SSLImplementation will depend on the presence of the APR/native
     library and the useOpenSSL attribute of the
     AprLifecycleListener.
     Either JSSE or OpenSSL style configuration may be used regardless of
     the SSLImplementation selected. JSSE style configuration is used below.
-->

<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true"
           scheme="https" secure="true"
           keystoreFile="${user.home}/.keystore" keystorePass="641997Al"
           clientAuth="false" sslProtocol="TLS"/>

<!-- Define an SSL/TLS HTTP/1.1 Connector on port 8443 with HTTP/2
```

Ilustración 95 Fichero de configuración Apache Tomcat

Se debe puntualizar que los pasos especificados nos permitirán firmar nuestro propio certificado. En un entorno de producción, será necesario un certificado emitido por una CA (entidad certificadora) verificada, el cual se debe especificar en la sección del fichero de configuración mostrada en la ilustración 96.

```
<!--
<Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
           maxThreads="150" SSLEnabled="true" >
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
    <SSLHostConfig>
        <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
                     certificateFile="conf/localhost-rsa-cert.pem"
                     certificateChainFile="conf/localhost-rsa-chain.pem"
                     type="RSA" />
    </SSLHostConfig>
</Connector>
```

Ilustración 96 Indicar un certificado externo

Por lo tanto, no es necesario llevar a cabo el paso anterior, el cual pretende ilustrar el uso de un certificado externo de la propia organización.

C.2.3 SPRING BOOT DASHBOARD

Una de las ventajas del uso de Spring Tool Suite es poder hacer uso de Spring Boot Dashboard. Se podrá ejecutar de forma rápida distintas aplicaciones boot seleccionando el botón ‘re(start)’.

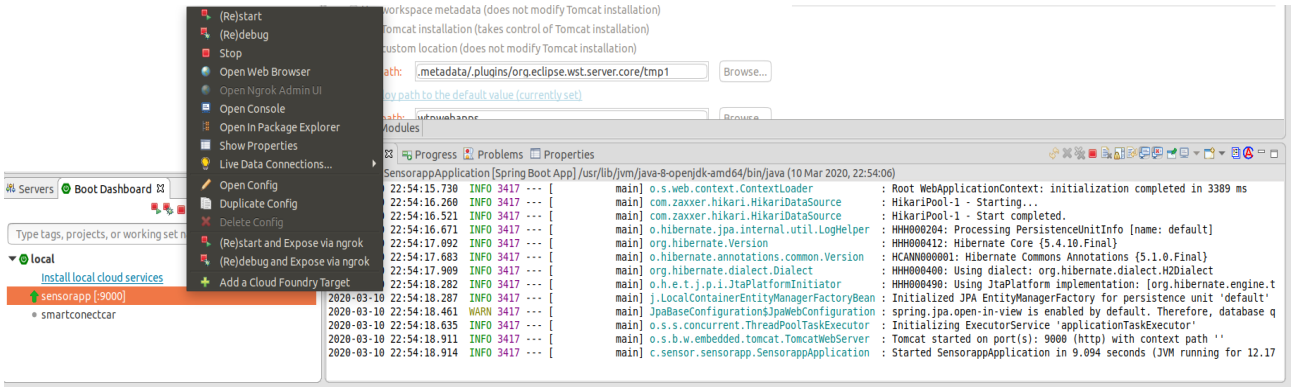


Ilustración 97 Spring Boot Dashboard

Una de las principales ventajas será la facilidad para poder ejecutar varias aplicaciones boot al mismo tiempo, de la cual se hará uso para poder ejecutar ambos servicios al mismo tiempo.

Una vez creado un Spring Starter Project, se deberá de ejecutar como aplicación Spring para que se cree la sección correspondiente a dicha aplicación en el dashboard. Se deberá de prestar atención a las configuraciones de cada aplicación, ya que no pueden tener el mismo puerto si se ejecutan simultáneamente.

Para cambiar las opciones de ejecución se deberá de modificar el archivo application.properties, concretamente nos interesará cambiar el puerto para que no usen dos aplicaciones el mismo puerto tal y como se muestra en la ilustración 98.

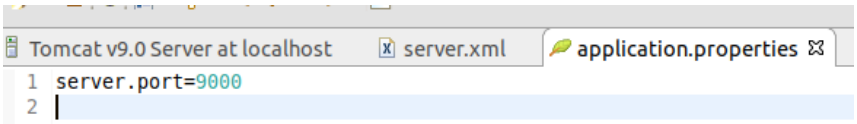


Ilustración 98 Application.properties

ANEXO D: PUESTA EN MARCHA DEL SISTEMA

En este anexo se pretende detallar la puesta en marcha del sistema, así como replicar su funcionamiento en un entorno diferente, permitiendo su correcto funcionamiento con propósitos variados como la realización de pruebas.

Todos los recursos del proyecto pueden ser accedidos mediante Github, almacenados en un repositorio público en el enlace https://github.com/alvcarpal/smartcarTFG.

D.1 Servicios y Aplicación Web

El primer paso será configurar adecuadamente todos los recursos necesarios a usar en Spring Tool Suite, siendo éste el entorno seleccionado para el desarrollo del proyecto. Los pasos a seguir son:

1. Instalación y configuración de Spring Tool Suite según lo expuesto en el Anexo C.

2. Instalación y configuración del servidor web Apache Tomcat v8.5 según lo expuesto en el Anexo C.

3. Clonar el repositorio de Github en la máquina a usar mediante Eclipse accediendo a la opción Window->Show View->Other, y seleccionando Git Repositories dentro de las opciones de Git según lo mostrado en la ilustración 99.

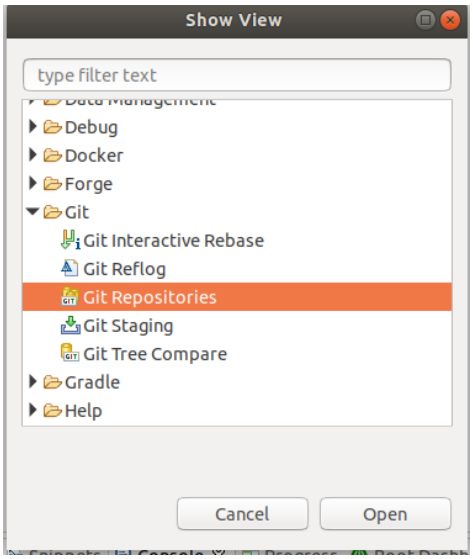


Ilustración 99 Eclipse Git Repositories

4. Seleccionar la opción "Clone a Git Repository" dentro de la pestaña "Git Repositories", cumplimentando correctamente la pestaña que se abrirá con la dirección del repositorio y el directorio local donde se clonará el repositorio del proyecto, tal y como se muestra en la ilustración 100.

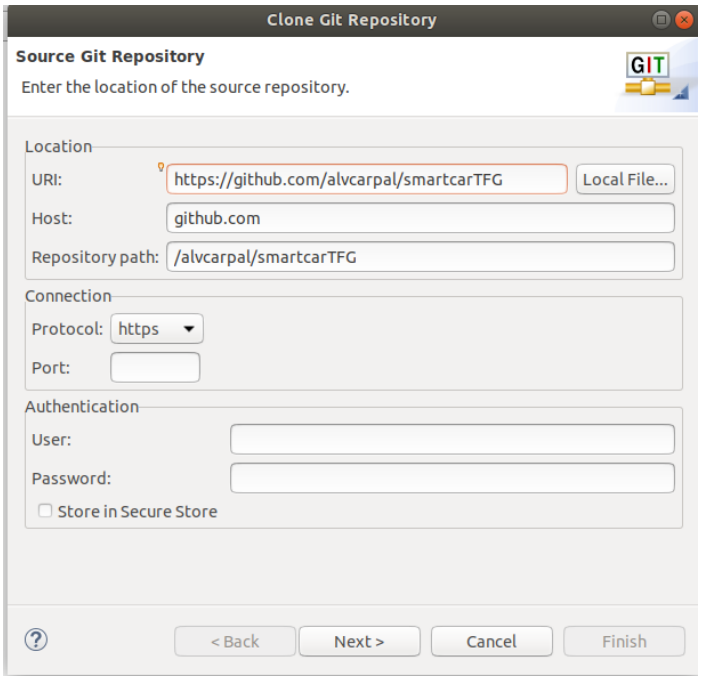


Ilustración 100 Clonar Repositorio del Proyecto

5. El último paso será seleccionar la rama principal del repositorio clonado e importar a Spring Tool Suite todos los proyecto usados:

   Smartcarapp: Aplicación web.

   Sensorservice: Servicio web encargado de almacenar la información de los sensores.

   Keyrockservice: Servicio web encargado de la comunicación con Keyrock.

   Middleservice: Servicio web encargado de procesar la información de los sensores.

Una vez realizado los pasos expuestos se tendrán todos los recursos del entorno configurados, necesitando inicializar el servidor Apache Tomcat en la pestaña servers y los servicios web mediante Spring Boot.

En caso de error con alguna de las librerías se debe de importar la librería específica de nuevo, encontrándose todas las librerías con sus formatos .jar en los recursos del repositorio. Para más detalle se pueden consultar los tutoriales desarrollados para el proyecto, los cuales se encuentran dentro de la carpeta tutoriales en los recursos del proyecto.

D.2 Componentes FIWARE

La instalación de los componentes FIWARE se detalló en el Anexo A, por lo que los pasos a seguir serán los siguientes:

1. Descarga de los archivos docker-compose.yml de los recuros del sistema o de DockerHub correspondiente al componente.

2. Configurar los ficheros según lo expuesto en el Anexo A en función de cada usuario.

3. Ejecutar primero el archivo dentro de la carpeta creada para Keyrock con sudo docker-compose up. Esto iniciará Keyrock y la base de datos gestionada por MySQL.

4. Realizar el paso anterior, pero en la carpeta correspondiente a Orion Context Broker. Ejecutando el mismo comando inicializaremos Orion Context Broker, la base de datos MongoDB y IoT Agent Ultralight.

5. Inicializar PEP-Proxy Wilma mediante el comando "sudo node server" dentro del directorio clonado para dicho componente.

Con estos pasos ya se han inicializado todos los componentes necesarios para el correcto funcionamiento del proyecto.

D.3 Base de Datos

La base de datos gestionada por PostgreSQL será el último elemento a configurar para el despliegue del sistema. Los pasos a seguir son:

1. Instalación del servidor PostgreSQL.

2. Creación de una nueva base de datos mediante psql.

3. Descargar el fichero Crea.sql de los recursos del proyecto, el cual contiene todas las ordenes necesarias para la creación de relaciones.

4. Situar el fichero en el directorio home y usar la orden \i para poder ejecutar todas las ordenes que crearán las relaciones usadas.

REFERENCIAS

[1] Pablo Bermejo Pérez, *Aplicación y Servicio web para la gestión de información de sensores usando Fiware y Spring*, Sevilla: Universidad de Sevilla, 2019

[2] Métodos Postman en recursos del Proyecto. Available: https://github.com/alvcarpal/smartcarTFG

[3] Plataforma FIWARE [En línea]. Available: https://www.fiware.org/

[4] FIWARE Step-by-Step Orion Context Broker [En línea]. Available: https://fiware-tutorials.readthedocs.io/en/latest/getting-started/index.html

[5] FIWARE Orion Context Broker Subscribing to changes of state [En línea]. Available: https://fiware-tutorials.readthedocs.io/en/latest/subscriptions/index.html

[6] FIWARE-idm Home Identity Manager-Keyrock [En línea]. Available: https://fiware-idm.readthedocs.io/en/latest/

[7] Álvaro Alonso, Identity Manager-GE Keyrock API – Apiary [En línea]. Available: https://keyrock.docs.apiary.io/#

[8] FIWARE XACML Rules-based Permissions [En línea]. Available: https://fiware-tutorials.readthedocs.io/en/latest/xacml-access-rules/index.html

[9] Unified Modeling Language (UML) [En línea]. Available: https://www.uml.org/

[10] FIWARE Read the Docs IoT Agent Ultralight [En línea]. Available: https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual/index.html

[11] Hibernate [En línea]. Available: https://hibernate.org/

[12] Apache HTTP Client [En línea]. Available: https://hc.apache.org/

[13] Fiware, fiware/idm Dockerhub [En línea]. Available: https://hub.docker.com/r/fiware/idm/

[14] Fiware, PEP-Proxy Wilma [En línea]. Available: https://fiware-pep-proxy.readthedocs.io/en/latest/

[15] Álvaro Carmona Palomares, Recursos Proyecto [En línea]. Available:

https://github.com/alvcarpal/smartcarTFG

[16] HTML Resources W3schools [En línea]. Available: https://www.w3schools.com/

[17] Docker Documentation [En línea]. Available: https://www.docker.com/

[18] Fiware Documentation Step-by-Step [En línea]. Available: https://fiware-

tutorials.readthedocs.io/en/latest/

[19] PostgreSQL, «PostgreSQL: About,» [En línea]. Available: https://www.postgresql.org/about/

[20] JavaScript Object Notation Documentation [En línea]. Available: https://www.json.org/json-es.html

[21] Oauth Flows [En línea]. Available: https://fiware-idm.readthedocs.io/en/latest/oauth/oauth_documentation/index.html

[22] Madeja, Guía para la redacción de casos de uso [En línea]. Available: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416

[23] FIWARE CATALOGUE [En línea]. Available: https://www.fiware.org/developers/catalogue/

[24] Oracle Documentation [En línea]. Available: https://docs.oracle.com/javaee/6/api/javax/servlet/Servlet.html

[25] Spring Boot Maven Plugin [En línea]. Available: https://docs.spring.io/spring-boot/docs/current/maven-plugin/examples/run-system-properties.html

[26] Spring Tool Suite 4 [En línea]. Available: https://spring.io/tools

[27] Android LocationManager [En línea]. Available: https://developer.android.com/reference/kotlin/android/location/LocationManager

[28] Android Studio Building [En línea]. Available: https://developer.android.com/studio/run

[29] MySQL Documentation [En línea]. Available: https://dev.mysql.com/doc/refman/8.0/en/mysql.html

[30] MongoDB Documentation [En línea]. Available: https://www.mongodb.com/es