Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Smart ecoMobility: movilidad sostenible en ciudades inteligentes

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Héctor Porta Albentosa

Tutor: Joan Fons i Cors

2º Tutor: Vicente Pelechano Ferragud

2017-2018

Resumen

Cada día miles de dispositivos trabajan, comparten y se comunican a través de internet, esto permite tener en tiempo real más información del entorno. Esto es debido ya que al tener más dispositivos conectado a Internet, esto nos facilita la interacción con otras aplicaciones. Como resultado de esto obtenemos toda una fuente de información.

El problema al que nos enfrentamos es que todos los equipos que recogen ésta información no siempre siguen un modelo de **comunicaciones** único. Todos estos equipos transmiten sus **mensajes** de una forma diferente al resto con lo cual es más difícil integrar estos equipos con otros.

En este proyecto se expone un modelo de comunicación entre sensores y una plataforma de ***IoT(Internet de las cosas)*** llamada ***Fi-ware***, la cual utilizamos para obtener los recursos necesarios para la comunicación entre sensores y **plataforma**. Una vez conseguido esto creamos un proceso que se encargue de recoger toda la información relevante para la plataforma y estandarice las lecturas para la posterior recopilación de estas.

Palabras clave: mensajes, IoT, Internet de las cosas, Fi-Ware, plataforma, comunicaciones.

Abstract

Every day, thousands of devices works, shares and communicates through the internet, this allows obtaining in real time more information of the environment. This is due to the fact of having more and more devices connected to the Internet; this makes easier the interaction with other applications. Because of this, we obtain a whole source of information.

The problem that we face is that all of the devices that collect this information not always follow a unique communication model. All of these devices transmit these messages in many different ways, making the integration between devices more difficult.

This proyect exposes a model of communication between sensors and an IoT (Internet of Things) platform called Fi-ware, which we use to obtain the necessary resources for the communication and between sensors and the platform. Once is this achieved, we create a process which is responsible of collect the relevant information for the platform and standardizes the readings for the compilation of these.

Keywords : communication, messages, IoT, Fi-ware, platform.

Resum

Cada dia milers de dispositius treballen, compartixen y es comuniquen a través de la Internet, permitent obtenir en temps real més informació de l'entorn. Aquest és degut al fet de tindre més i més dispositius connectats a la internet, aquest fa facilitar la interacció entre aplicacions. Com a resultat d'açò, obtenim tota una font d'informació.

La problemàtica al que ens enfrontem és que tots els equips que arrepleguen aquesta informació no sempre segueixen un model únic de comunicació. Tots aquests equips transmeten els seus missatges d'una forma diferent del rest amb el qual és més difícil aquests equips amb altres.

En aquest projecte s'exposa un model de comunicació entre sensors i una plataforma de **IoT** (Internet de les Coses) anomenat **Fi-Ware**, la qual utilitzarem per a obtenir els recursos necessaris per a les **comunicacions** entre sensors i la **plataforma**. Una vegada aconseguit açò creem un procés que s'encarregue d'arreplegar tota la informació rellevant per a la plataforma i normalitze les lectures per a la posterior recopilació d'aquestes.

Paraules clau : comunicacions, missatges, IoT, Fi-ware, platforma.

Tabla de contenidos

Índice de imágenes

1. Introducción

Actualmente el término *Smart Cities* (Ciudades Inteligentes) es cada día más común. La mayoría de las ciudades importantes ya disponen de entornos inteligentes y comparten información pública, los cuales se integran con aplicaciones propias u otras con los cuales ofrecen información sobre el entorno.

Estas ciudades inteligentes están relacionadas muy de cerca con el tema del Internet de las Cosas (IoT) ya que gracias a estos podemos estar en comunicación constante con nuestro entorno proporcionando información o recibiendo.

1.1 Smart Cities

[1] La imperiosa necesidad del ser humano a crear un modelo de vida sostenible siempre ha sido uno de los objetivos primordiales de este, ya que a mayor sostenibilidad mayor será la calidad de vida. La idea consiste en poder crear una gestión eficiente de todas las infraestructuras de las ciudades, tales como electricidad, contaminación, servicios, sanidad, etc. para sacar el máximo provecho de estas.

Las Smart Cities o ciudades inteligentes es la utilización de la tecnología y la innovación en estas áreas para la promover un desarrollo más eficaz y sostenible en todas las áreas además de encontrar el equilibrio entre el bienestar de los ciudadanos y el entorno.

El término de Smart City se asocia a la evolución de un proyecto llamado Ciudades Digitales del Ministerio de Industria de España en el 2004. En este proyecto se planteaba la creación de una ciudad totalmente digitalizada, todas las infraestructuras estaban integradas entre ellas, este proyecto fue planteada por la empresa española ACCEDA.

En este proyecto reunió a más de 30 empresas de diferentes sectores para conseguir una unión productiva entre ellas, planteando los que sería llamado una Comunidad Digital. El proyecto finalmente no se desarrolló físicamente pero sí de manera cinematográfica. Más tarde IBM se encargaría de nombrar este planteamiento como Smart Cities.

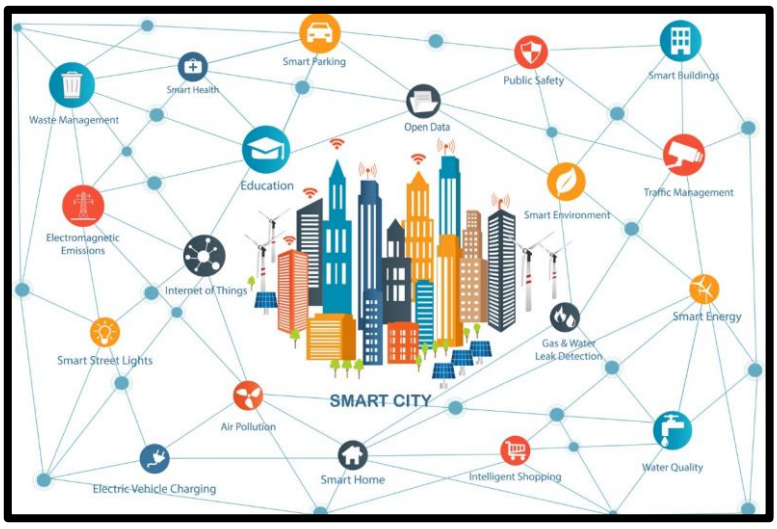

Figura 1. Smart City

1.2 Ventajas

La constante evolución de la tecnología hace que una Smart City siempre esté en desarrollo con lo cual hace que siempre esté mejorando y optimizando las tareas de la infraestructura. Algunas otras ventajas:

- Mejora del estado del medio ambiente
- Optimización de los servicios públicos
- Mejorar transparencia en la gestión de las administraciones
- Mejorar comunicación con la ciudadanía

1.3 Ámbitos de aplicación

- **Medio Ambiente**: el ámbito en el que más se ha notado el desarrollo de estas smart cities. Con este planteamiento, se gestiona, controla y crea mecanismos con los cuales podemos controlar el gasto de recursos como agua o gas, la polución creada en ciertas zonas de la ciudad. La idea es proporcionar los servicios básicos asegurando el bienestar de los ciudadanos, pero teniendo en cuenta el medio natural.
- **Sanidad**: telemedicina, gestión de datos, historiales de pacientes, son alguno de los ejemplos en los que podríamos aplicar soluciones como asistencia a personas discapacitadas, control de personas con Alzheimer, etc.
- **Transporte y urbanismo**: el transporte y la logística de los transportes públicos estarían integrados e interconectados. Se gestiona más eficientemente los sistemas facilitando el acceso a los ciudadanos. También podríamos incluir el transporte ecológico, ya que priorizamos el acceso a opciones de transporte accesible, para ahorrar tiempo, reducir costes y disminuir emisiones.
- **Gobierno**: administración electrónica e integración con otros ámbitos como medicina agiliza todo tipo de gestiones que requieran de información externa así reduciendo costes y tiempo.

- **Seguridad**: Coordinación entre los cuerpos de policía y de emergencias para mejorar la gestión de tiempo de reacción en caso de emergencia.

1.4 Requisitos

[2] Para plantear un modelo de ciudad inteligente, esta ciudad debe tener ciertos requisitos mínimos para que esta ciudad sea sostenible y pueda gestionarse correctamente:

- **Desarrollo económico, social y medioambiental sostenible**: la estabilidad de estos tres puntos es básico ya que sin estabilidad en algún punto no nos podría asegurar el mantenimiento de la smart city.

- **Gestión óptima de los recursos**: la participación de los ciudadanos a de ser activa tanto en el uso como en su fomento, ya que sin esta participación no se alcanzarian los objetivos planteados.

- **Infraestructuras comprometidas**: al igual que los ciudadanos: las infraestructuras también deben estar comprometidas con el proyecto.

- **Instituciones dotadas de soluciones tecnológicas** para hacer la vida de los ciudadanos más sencilla.

1.5 Objetivos

El proyecto que presentamos a continuación persigue esta idea, crear un entorno de simulación que emule un modelo de comunicación entre sensores y la plataforma IoT Fi-Ware y plantear una solución que utilice los datos de manera que puedan recoger y utilizar la información sobre estos sensores. Lo principales objetivos son:

- Investigación y evaluación sobre las diferentes plataformas IoT para desarrollar la solución.

- Investigación y análisis sobre Fi-ware y sus utilidades.

- Investigación sobre diferentes lenguajes de programación y seleccionar el más correcto para el caso.

- Diseño de los mensajes que envían los dispositivos y creación de entidades en una API REST

- Implementación de procesos de comunicación que envían y transforman mensajes a la API REST.

- Implementación de entidades a la API REST para la persistencia de los mensajes.

- Implementación de una solución que haga uso de estos datos.

1.6 Metodología

La metodología aplicada a este proyecto ha sido la siguiente:

- **Aprendizaje e instrucción** (28 semanas): fase en la que principalmente se dedicó el esfuerzo de entender y aprender a usar Fi-Ware, plataforma en la que integraremos el modelo de comunicación y cómo crear una instancia con la imagen necesaria para dicho propósito.

- **Caso de estudio** (7 semanas): planteamiento del caso de estudio que utilizaremos para prototipar e intentar representar el caso real de un modelo de comunicación entre sensores y la plataforma Fi-ware.

- **Diseño e implementación** (4 semanas y 8 semanas): planteamiento y desarrollo del código necesario para plantear el caso de estudio. En este caso hemos utilizado Fi-ware como plataforma de comunicaciones y el protocolo MQTT para el intercambio entre mensaje. Java para simular sensores y html para realizar demo de las pruebas.

- **Pruebas** (8 semanas): testeo de los programas y código desarrollado.

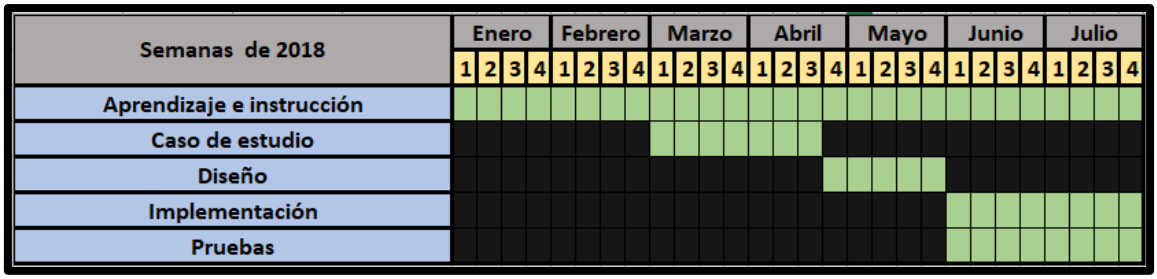

Figura 2. Diagrama de tiempo empleado por proceso

2. Contexto Tecnológico

Una vez introducido el trabajo, pasamos a describir el software utilizado para desarrollar el proyecto. Cabe destacar que, aunque hayamos usado este software existen diversas herramientas que puedan sustituir a las planteadas o que pueden trabajar complementando a otras.

2.1 Software de desarrollo: Eclipse Photon

Figura 3. Logo Eclipse Photon

[5] Eclipse es una plataforma software compuesta por un conjunto de herramientas de programación de código abierto. Internamente funciona a través de plug-ins, es decir, es una herramienta adaptable al usuario que lo está utilizando. Cabe destacar que es uno de los entornos de desarrollo más utilizado por los profesionales por su capacidad de adaptación, facilidad de uso y capacidad de crear entornos de trabajo.

Algunas de sus IDE's más usadas son Java, C++, PHP, Web (HTML, Css, JavaScript).

Cada una de las aplicaciones que se desarrolla en el entorno eclipse se agrupan en proyectos, estos proyectos son donde se agrupan las clases, librerías y estructura de carpetas en las cuales organizamos el trabajo.

En general podemos destacar:

- **Funcionalidades mediante plug-ins**: Al trabajar mediante plug-ins hace que la herramienta no pese demasiado y ralentiza demasiado el sistema con lo cual mejora el rendimiento del pc mientras se está trabajando.

- **Tiempo de carga**: eclipse trabaja en función de los módulos que se estén usando en ese momento con lo cual, si en un momento específico no estás usando alguno de los plug-ins instalados no los arranca, mejorando asi el tiempo de carga de la aplicación.

- **Soporte de la aplicación**: Al ser código abierto se desarrollan varias soluciones para un problema en concreto, estas soluciones pueden ser planteados por el propio repositorio de eclipse o mediante diferentes repositorios que se pueden ofrecer abiertamente.

- **Ayuda**: El mecanismo de Ayuda de la plataforma Eclipse permite que las herramientas definan y aporten documentación a uno o más libros en línea. Por ejemplo, una herramienta generalmente contribuye con la documentación del estilo de ayuda a una guía del usuario, y la documentación de API (si tiene alguna) a una guía de programador separada. La información se expresa en html mientras que las estructuras de navegación se plantean en XML.

Otras herramientas parecidas:

- **NetBeans**: Es el entorno principal que hace competencia a eclipse, una de las principales diferencias es que trabaja mediante módulos, además que muchos de los plug-ins que necesita eclipse ya vienen predeterminados en la aplicación, en contra es que la memoria que consume ya que en esta aplicación sí que carga los módulos que no se están utilizando

- **VisualStudio**: Entorno de desarrollo desarrollado por Windows. A diferencia de los demás entornos, este entorno es de pago ya que requiere una licencia de windows pero también es uno de los entornos más potentes que podemos encontrar para el desarrollo de aplicaciones.

- **JetBrains**: En este caso JetBrains no es un entorno en particular, es un desarrollador de entornos concretos para diferentes herramientas.

2.2 Cliente TTY: Ubuntu for Windows

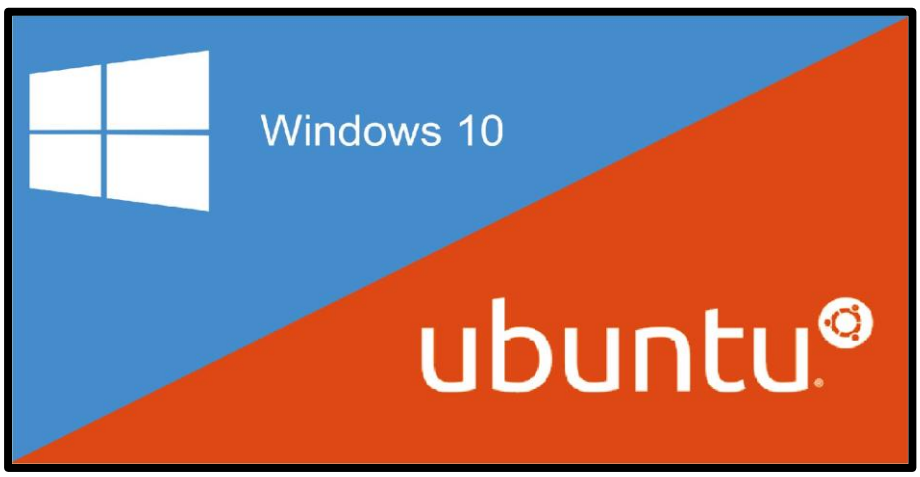

Figura 4. Logo Ubuntu for Windows

[6] Una de las nuevas características que ofrece Windows 10 es una aplicación sencilla que simula una terminal bash de ubuntu. Con ello tenemos una terminal perfectamente equipada para poder realizar nuestras conexiones con los servidores que luego montaremos y poder administrarlos, ya que en este caso nosotros no tenemos que instalar ningún paquete adicional proporcionándonos un entorno ligero para poder realizar estos comandos.

Otros terminales:

PuTTY: Otro cliente ssh para Windows con el cual podríamos hacer la gestión de estos servidores, nos proporciona varias herramientas como PuTTYGen para generar claves RSA y DSA, además de poder gestionar nuestras conexiones a diferentes IP's.

MobaXterm: Cliente similar al PuTTY, con entorno gráfico propio en el cual simula la conexión, gestión de conexiones y personalización.

2.3 Plataforma IoT: Fi-ware

Figura 5. Logo Fiware

[3] Fi-ware es una plataforma Open-Source creada para el desarrollo y despliegue de aplicaciones. Combina componentes que permiten la conexión a IoT con Context Information Management y servicios Big Data en la nube.

Algunas empresas como Telefónica y Orange han apostado fuerte por esta tecnología para desarrollar estándares para smart cities.

La comunidad FIWARE es una comunidad comprometidos con la plataforma intentado construir lo que ellos llaman un ecosistema sostenible abierto en torno a estándares de software públicas, libres de regalías y orientadas a la implementación que faciliten el desarrollo de nuevas aplicaciones inteligentes en sectores múltiples.

Los principales pilares en los que se sostiene Fiware son:

- **Fiware**: La plataforma FIWARE proporciona un conjunto simple, pero potente de API's que facilitan el desarrollo de aplicaciones inteligentes en múltiples sectores. Las especificaciones de las API's son públicas y libres de regalías.

Figura 6. Logo Fiware 2

- **Fiware Lab**: es un entorno no comercial donde se llevan a cabo innovaciones y experimentaciones basadas en tecnologías FIWARE. Aquí se pueden probar la tecnología y sus aplicaciones, aprovechando los datos abiertos publicados por las ciudades y otras organizaciones.

Figura 7. Logo Fiware Lab

- **Fiware Accelerate**: este programa tiene como objetivo promover la adopción de las tecnologías FIWARE entre los integradores de soluciones y los desarrolladores de aplicaciones, con especial énfasis en las pymes y las nuevas empresas.

Figura 8. Logo Fiware Accelerate

- **Fiware Mundus**: este programa está diseñado para brindar cobertura a este esfuerzo estableciendo contactos con los gobiernos locales en diferentes partes del mundo.

Figura 9. Logo Fiware Mundus

- **Fiware iHubs**: tiene como objetivo apoyar la creación y las operaciones de los nodos iHubs en todo el mundo, para que los desarrolladores aporten a Fiware de forma local.

Figura 10. Logo Fiware iHubs

Otras plataformas:

- **IBM Watson IoT**: IBM Watson es una tecnología pionera de computación cognitiva. IBM logra interaccionar de una manera similar a como lo hacen las personas así facilitando la comprensión de los datos a integrar.

- **Microsoft Azure IoT Suite**: Azure IoT Suite es una oferta integrada que aprovecha todas las capacidades relevantes de Azure para conectar dispositivos y otros activos (es decir, "cosas"), captura los diversos y

voluminosos datos que generan, integran y organizan el flujo de esos datos, y administran, analízalo y preséntalo como información utilizable a las personas que lo necesitan para tomar mejores decisiones y automatizar inteligentemente las operaciones.

- **Google Cloud IoT**: es un conjunto de servicios completamente administrado e integrado que te permite conectar, administrar e ingerir datos a gran escala, así como de forma fácil y segura, a partir de dispositivos repartidos por todo el mundo. También te ayuda a procesar, analizar y ver datos en tiempo real, implementar cambios operacionales y llevar a cabo las acciones que creas pertinentes.

- **GE Predix**: Predix es la plataforma de software de General Electric para la recopilación y el análisis de datos de máquinas industriales. General Electric planea apoyar el creciente Internet industrial de cosas con servidores en la nube y una tienda de aplicaciones. GE es miembro del Consorcio de Internet Industrial que trabaja para ayudar al desarrollo y uso de tecnologías industriales de Internet.

- **Kaa**: es una tecnología de habilitación de IoT aplicable para cualquier escala de desarrollo de IoT empresarial. Proporciona una gama de características que permiten a los desarrolladores crear aplicaciones avanzadas para productos inteligentes, administrar de forma flexible sus ecosistemas de dispositivos, organizar el procesamiento de datos de extremo a extremo, y muchos más. Con Kaa, puede crear sus aplicaciones IoT hasta 10 veces más rápido que antes.

- **Ubidots**: Nacido como una firma privada de servicios de ingeniería en 2012, Ubidots se especializó en hardware conectado y soluciones de software para monitorear, controlar y automatizar de manera remota los procesos para clientes de atención médica en startups bien financiadas y Fortune 1,000 en el sureste de América y en Latinoamérica.

2.4 Lenguajes de programación: Java, HTML5, JavaScript

En este proyecto hemos utilizado estos tres lenguajes de programación:

- **Java**: para desarrollar la parte de simulación de sensor y transformador de mensajes.
- **HTML y JavaScript**: para desarrollar la parte del mapa donde registramos la actividad del sensor.

A continuación, procederemos a describir más específicamente estos lenguajes.

- **Java**

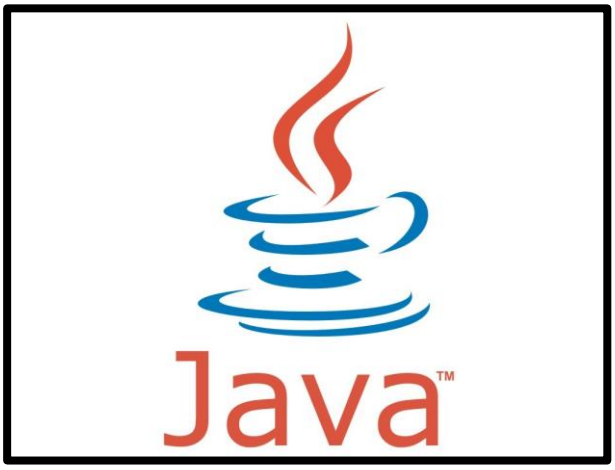

Figura 11. Logo Java

[4] Lenguaje de programación orientado a objetos, compilado e interpretado. Esto quiere decir que lo que genera la compilación del programa no lo ejecuta directamente el procesador del ordenador sino una máquina virtual. Esto permite que los Applets de una web pueda ejecutarlos cualquier máquina que se conecte a ella independientemente de qué sistema operativo emplea.

Algunas de sus características:

- Lenguaje simple
- Robusto
- Seguro
- Multi-hilo
- Totalmente orientado a objetos

Otros lenguajes orientados a objetos:

- **C++**: La intención de su creación fue el extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Fue diseñado con un sesgo hacia la programación del sistema y sistemas integrados, de recursos limitados y de gran tamaño, con rendimiento, eficiencia y flexibilidad de uso a medida que se destaca su diseño.

- **Ruby**: es un lenguaje de programación dinámico, interpretado, reflexivo, orientado a objetos y de propósito general. Fue diseñado y desarrollado a mediados de la década de 1990 por Yukihiro "Matz" Matsumoto en Japón.

- **ADA**: es un lenguaje de programación de computadora estructurado, estático, imperativo y orientado a objetos de alto nivel, que se extiende desde Pascal y otros idiomas. Tiene soporte de lenguaje incorporado para el diseño por contrato, tipeo extremadamente fuerte, concurrencia explícita, tareas, paso de mensajes sincrónicos, objetos protegidos y no determinismo. Ada mejora la seguridad del código y el mantenimiento mediante el uso del compilador para encontrar errores a favor de los errores de tiempo de ejecución.

- **Perl**: La estructura completa de Perl deriva ampliamente del lenguaje C. Perl es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas.

- **HTML**

Figura 12. Logo HTML5

Lenguaje estándar de programación web, es un lenguaje que se estructura a partir de etiquetas, el uso de este lenguaje se extiende desde 1993, desde entonces se han desarrollado varias versiones de este lenguaje siendo la 5 la última. Esta versión incluye diversas mejoras de tipo multimedia, como incluir etiquetas de video incluyendo sin necesidad de elementos externos para poder incluirlo en el html.

- **JavaScript**

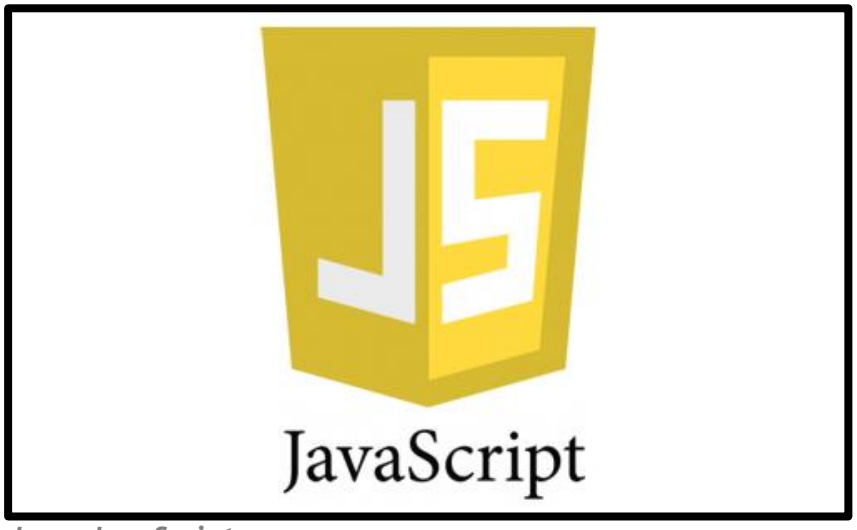

Figura 13. Logo JavaScript

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Básicamente se utiliza en HTML para crear páginas webs dinámicas, es decir páginas web interactivas que realizan efectos según acciones que se realicen en la página, se pueden programar acciones para que se realicen en algún momento determinado, etc.

2.5 REST

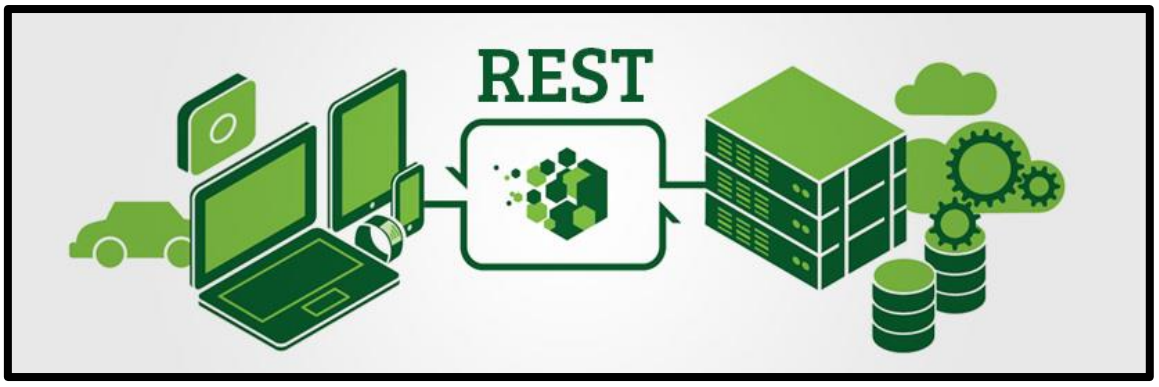

Figura 14. Logo REST

[7] REST (Representational State Transfer) es un estilo arquitectónico para proporcionar estándares entre sistemas informáticos en la web, lo que facilita la comunicación entre los sistemas. Un concepto importante en REST es la existencia de recursos (elementos de información), que pueden ser accedidos utilizando un identificador global. Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura, en la actualidad se usa en el sentido más amplio para

describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc.).

Las operaciones que se pueden realizar a menudos se equiparan a las operaciones CRUD (Crear, Leer, Actualizar y Borrar) ya que son pocas las operaciones que se pueden realizar en el protocolo y todas pertenecen al HTML, estas son **GET**, **PUT**, **POST** y **DELETE**.

Todas las entidades creadas en el protocolo REST están referenciadas mediante una URL y se pueden operar con ellas mediante peticiones creadas con las anteriores opciones.

2.6 JSON

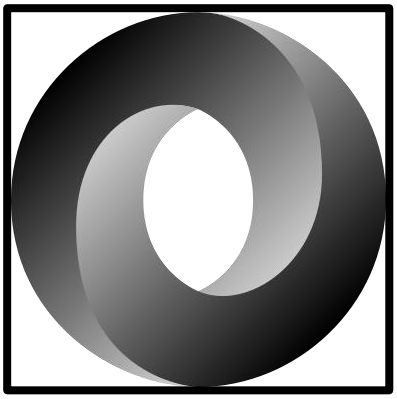

Figura 15. Logo JSON

JSON es un formato de texto ligero para el intercambio de datos. Nosotros lo utilizaremos para crear entidades y cambiar propiedades de estos. La ventaja de utilizar JSON es la simpleza de los datos, ya que en XML se necesita un formato específico, JSON es una cadena de texto que una vez obtenida se puede tratar como un objeto.

Como hemos explicado antes, lo utilizaremos para comunicarnos con REST ya que este servicio obtiene datos mediante JSON, luego el servicio REST se encarga de procesar estos datos. Esto nos permite una comunicación rápido ya que solo se envían textos en plano.

2.7 RabbitMQ

Figura 16. Logo RabbitMQ

[8] Es un software de negociación de mensajes de código abierto, con librería para java con el cual realizaremos el intercambio de mensajes entre sensor y Fi-ware.

Con este software crearemos colas de mensajerías en las cuales colgaremos los mensajes de los sensores y desde estas colas obtendremos y procesaremos para realizar el envío a Fi-ware para que en las entidades precisas se creen o modifiquen los valores nuevos obtenidos.

2.8 POSTMAN

Figura 17. Logo POSTMAN

[9] Es una herramienta gratuita que permiten realizar tareas en el dentro del mundo de la API REST. En nuestro caso nos servirá para realizar algunas tareas de testeo y de administración de nuestra plataforma, específicamente crear las entidades necesarias para la lectura de los sensores.

En principio se creó como extensión de Google Chrome, pero a día de hoy dispone de aplicaciones nativas para MAC y Windows. Tiene versión gratuita y otra de pago en el que se incluye un cloud colaborativo para que varios equipos de trabajo puedan trabajar en común.

La elección de estas tecnologías está ampliamente relacionada con lo aprendido en el grado ya que la mayoría de estas se estudia y se desarrollan habilidades con estas, facilitando el desarrollo del proyecto y evitar así problemas en el desarrollo.

3. Caso de estudio

3.1 Introducción

En este proyecto hemos expuesto una serie de tecnologías que están directamente relacionadas con el mundo del IoT y con las ciudades inteligentes. Lo que se ha creado en este proyecto es la creación de un prototipo de ciudad inteligente sobre Valencia, simulando una serie de sensores en varios puntos de la ciudad para simular medidas de estos como, por ejemplo: viento, temperatura, contaminación.

En este caso nos hemos centrado en el ámbito del medio ambiente, ya que por simpleza no permite hacer una simulación más sencilla, además de intentar crear un entorno escalable.

3.2 Descripción

Valencia, como la tercera ciudad más ocupada de España, es una de las ciudades con más contaminación por CO. Como de todos es conocido, la contaminación producida por los coches es uno de los factores ambientales que condicionan el nivel de gases emitidos al ambiente. Alguno de estos gases es: NO (Óxidos Nitrosos), CO (Monóxido de Carbono) y CO2 (Dióxido de Carbono).

El monóxido de carbono es un tipo de gas nocivo que en altas cantidades puede resultar mortal. Este gas es producido debido a la combustión de sustancias como gas, petróleo, carbón, gasolina, tabaco, etc. Al tener un nivel tan alto de contaminación, en este proyecto se ha propuesto una manera de representar esto datos registrados y en función de los datos obtenidos restringir el acceso a ciertos vehículos contaminantes.

Otro de los sensores propuestos en este proyecto, son sensores de temperatura. Valencia es una de las ciudades en la que la temperatura en verano suele ser elevada poniendo en riesgo la salud de sus habitantes, afectado de misma manera a la contaminación.

Por último, otro de los sensores propuestos es el de viento. Valencia es conocida por sus playas y su buena temperatura, por ello hemos propuesto sensores en la costa que midan la velocidad del viento y según su velocidad decidir qué bandera poner para los bañistas. Para la elección de la bandera nos hemos basado en la escala de Beaufort. La escala de Beaufort es una medida empírica para la intensidad del viento, basada principalmente en el estado del mar, de sus olas y la fuerza del viento. Su nombre completo es escala de Beaufort de la fuerza de los vientos. En la implementación explicaremos qué niveles tomaremos como normalizados para que una persona pueda entrar al mar o no.

Otro punto a favor de la implantación de la smart city en valencia es, a pesar de que aún queda mucho trabajo, el compromiso que muestran los ciudadanos y las

instituciones en mejorar el medio ambiente para reducir estas emisiones hacen que Valencia sea una ciudad perfecta para realizar el prototipo.

3.3 Propuesta de solución

En nuestro caso situaremos diversos sensores por la ciudad como implementación inicial:

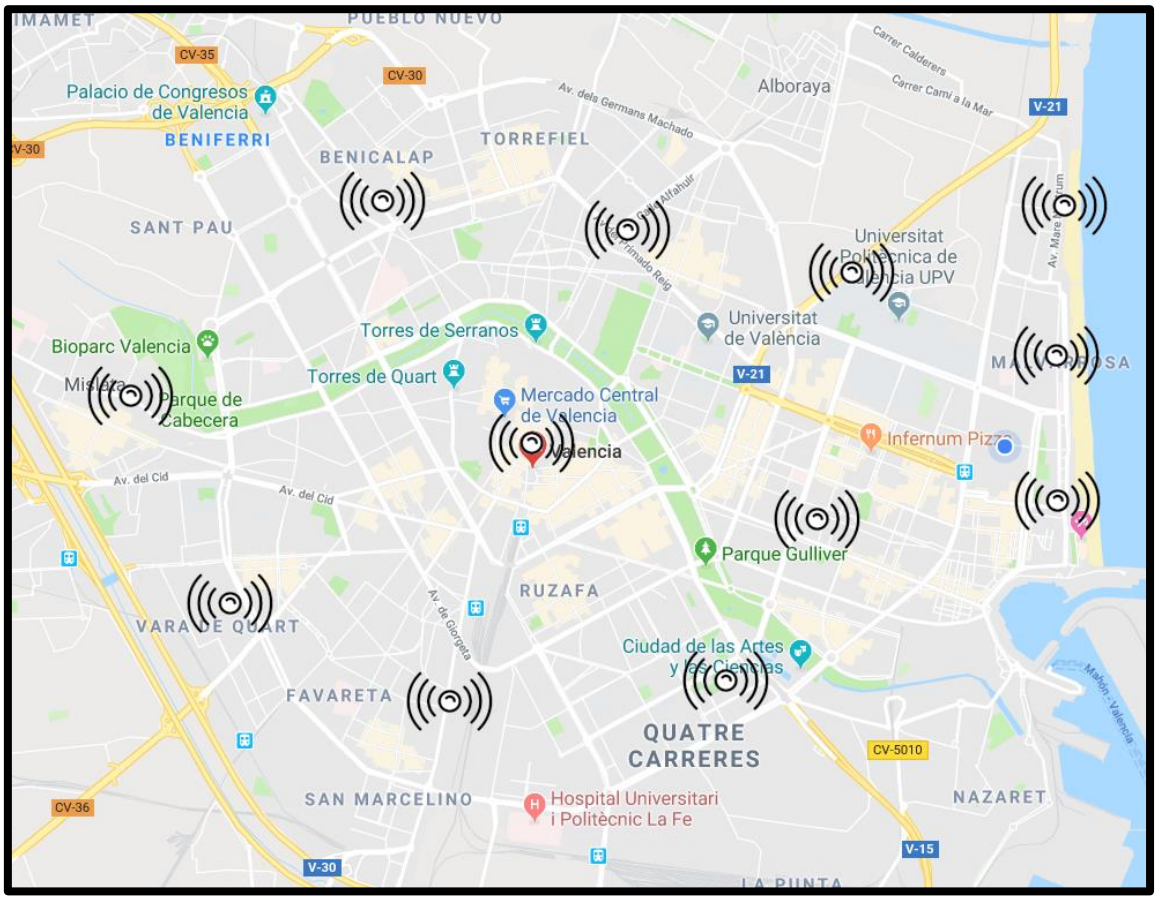

Figura 18. Mapa con lo sensores posicionados

Podemos destacar en el mapa las zonas que están cubiertas mediante los sensores que emiten valores de temperatura, velocidad de viento y nivel de CO. El proyecto funcionara de esta manera:

- Se desplegarán los sensores generando los mensajes antes mencionados.

- Estos mensajes se enviarán a una cola MQTT que se estará ejecutando en otro equipo remoto que estará en contacto con la plataforma Fi-ware.

- Este proceso se encargará de interpretar el mensaje y transformarlo en una sentencia REST para enviárselo a la plataforma Fi-ware.

- Por otra parte, tendremos una página web dinámica que estará obteniendo las lecturas cada X segundos, actualizando en el un mapa integrado en la página.

De esta manera se generan datos en público los cuales pueden ser utilizados por otros desarrolladores para crear aplicaciones de utilidad. Además, este diseño nos aporta un modelo escalable, ya que se separan las partes de canal de comunicación, persistencia y sensores, permitiendo añadir hardware diferente simplemente modificando el canal de comunicación y estandarizando estos mensajes.

4. Diseño de la solución

Una vez planteado el caso de uso, vamos a plantear el diseño que vamos a usar para poder realizar la solución, el primer paso será plantear la distribución del proyecto:

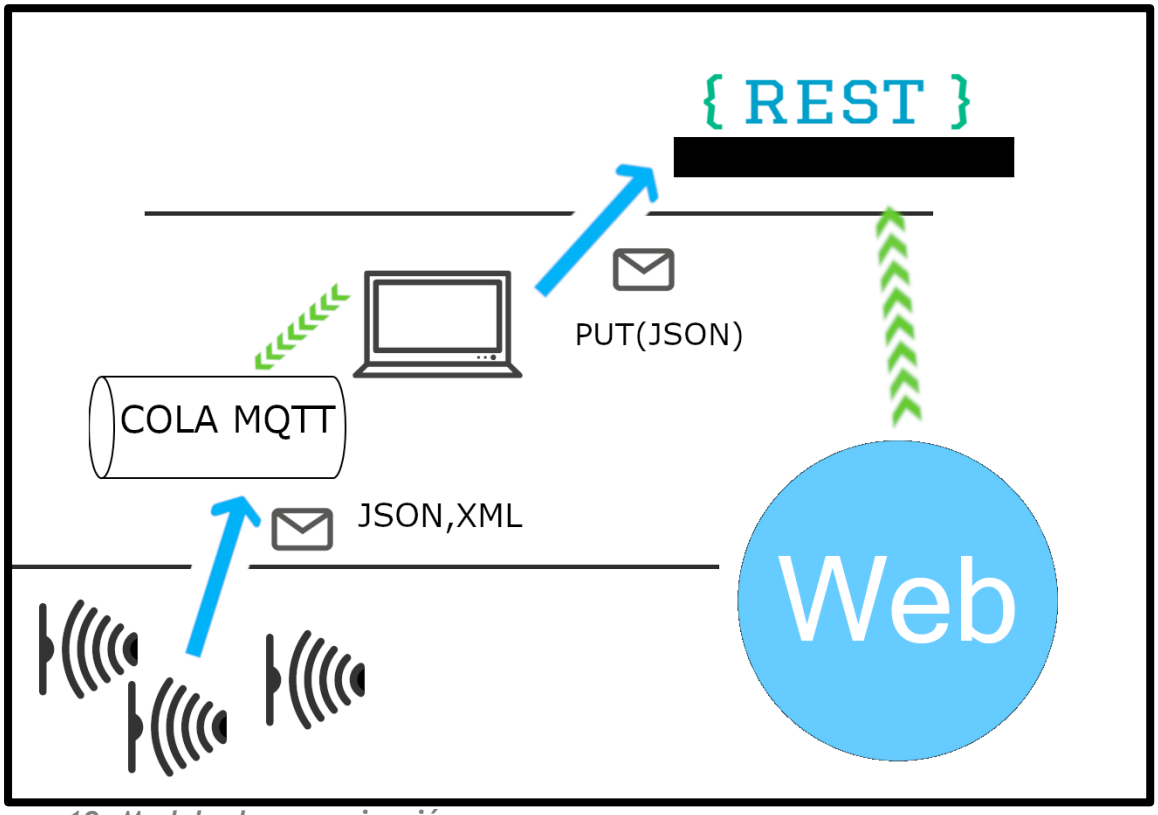

Figura 19. Modelo de comunicación

En la imagen podemos observar una representación de lo que hemos intentado realizar en el proyecto. Tenemos una serie de sensores que envían mensajes a una cola MQTT con unos formatos predeterminados por el fabricante, en este caso hemos simulado mensajes XML y JSON, que se cuelgan en un topic de rabbitMQ.

Una vez se insertan los mensajes en la cola, un proceso ejecutado en una máquina remota se encarga de obtener estos mensajes y procesarlos para su posterior envío a la API REST del servidor de Fi-ware.

Fi-ware interpreta el comando enviado por el proceso de comunicación y almacena en sus entidades creadas, los datos proporcionados por los mensajes enviado.

A su vez, tenemos un servlet ejecutándose en otra máquina, que requiere de los datos publicados por la API Rest de Fi-ware, obtiene estos mensajes y los mostrará según la utilidad de esta web. La web de prueba consiste en 3 páginas

representando los datos, dependiendo si son datos de viento, temperatura o contaminación.

En esta solución implementaremos 2 procesos en java para la recuperación de datos y 3 servlets java ejecutado en un server apache con tomcat.

4.1 Proceso Sensor

La clase sensor es un ejecutable de java que se encarga de crear mensajes de tipo XML o JSON según un método de aleatoriedad. No contiene ningún atributo ya que en este caso no era conveniente ya que es un elemento que solo ha de producir mensajes pero que sí debe tener una serie de requisitos:

- Ha de enviar mensaje tanto de XML como JSON.

- El sensor solo ha de comunicarse con la cola MQTT y publicar las medidas producidas por el mismo.

- Debe publicar las medidas dependiendo de unas posiciones concretas ya fijadas.

En este proyecto hemos usado una estructura de mensaje específica para cada uno de los mensajes ya que, al tratarse de una simulación, enviar diferentes tipos de mensajes JSON o XML supondría un retraso en el proyecto.

De cualquier manera, en un entorno real, la adaptación del proceso Canal del que hablaremos más adelante lo haría mucho más sencillo ya que no tendríamos que modificar el mensaje que nos envía el sensor sino el proceso de tratamiento del mensaje, lo cual nos permite introducir sensores nuevos sin ningún tipo de problema.

La estructura de los mensajes son las siguientes:

- **JSON**

```
{
        “key”: String,
        “version”: String,
        “unit”: String,
        “message”: {
                “type”:String,
                “measure”:String,
                “unit”:String
```

```
        },
        “position”: {
                “latitude”:String,
                “length”:String
        },
        “link-date”: String,
        “battery”: String,
        “type”: String,
        “status”: String,
        “frequency”: String
}
```

Como podemos ver el mensaje enviar datos de la medida proporcionada en este momento del tiempo. Una de las curiosidades es que el mensaje que envían no es personalizado según si es viento, temperatura o contaminación.

Todos los mensajes JSON se envían con la misma estructura, pero el orden de los atributos puede variar. Utilice el lenguaje del inglés para estandarizar el idioma y así trabajar en un lenguaje específico.

- **XML**

```
<?xml version = “1.0”>
<sensor-info>
      <key></key>
      <version></version>
      <message>
            <type></type>
            <value></value>
            <unit></unit>
      </message>
      <position>
            <latitude></latitude>
            <length><length>
      </position>
      <link-date></link-date>
      <battery></battery>
      <status></status>
      <frequency></frequency>
</sensor-info>
```

La estructura del mensaje es muy parecida a JSON, mismos datos y misma disposición. En el caso contrario de JSON es que XML sí mantiene el formato introducido desde un principio.

El objetivo es el mismo, enviar datos a la cola MQTT sin procesar, para que luego el proceso canal se encargue de analizarlos y transformarlos.

En un principio, se planteó la posibilidad de controlar los mensajes enviado por los sensores, de esta manera se consiguió un control total sobre los mensajes. El problema era que el envío de mensajes no se realizaba en masa, sino uno por uno, con lo cual limitaba las pruebas y tiempos de reacción.

Al final se ha decidido por un modelo de sensor aleatorio, lo único que está controlado en los sensores son las posiciones y las claves de estos, el resto es totalmente aleatorio puedo realizar medidas de cualquier tipo, agilizando así las pruebas.

4.2 Proceso Canal

Al igual que el sensor, el proceso canal no crea ningún tipo de objeto para almacenar datos, sino que lo que se trata con este proceso es recoger en la cola MQTT creada los mensajes que van depositando los sensores. Este proceso tiene 3 objetivos:

- Analizar los mensajes que recogemos de la cola MQTT y comprobar que son correctos.

- Procesar y transformar estos datos de XML o JSON a una petición HTTP.

- Ejecutar la petición y actualizar las entidades creadas en Fi-ware

Este proceso se ejecuta en una máquina remota, la cual actúa como servidor MQTT. En proceso canal se encarga de asociarse con esta cola de mensajería que funciona mediante Pull In, Pull Out.

4.3 Servlet Web

El diseño de estas páginas se realiza en servlets ya que combinan las funcionalidades de Java junto con HTML.

Para simular un cliente que hace uso de los datos que hemos recuperado en nuestra aplicación hemos implementado tres servlets web los cuales representan 3 mapas en los cuales están realizan 3 funciones:

- **Mapa de Contaminación** representada mediante círculos en función de la posición de los sensores

En este mapa lo que representaremos en el serán círculos que representarán las medidas que abarcan los sensores. Tienen 2 estados:

  - **Alerta**: Momento en el que las medidas que está tomando el sensor son demasiado altas y suponen un riesgo para la salud.

- **Normal**: Momento en el que las medidas que esta toma el sensor son normales y no suponen un riesgo para la saludo.

En este caso, este mapa se basará en las medidas proporcionadas por los mensajes de CO2 que se colgaran en las entidades de Fi-ware

- **Mapa de Temperaturas** por zonas.

Al igual que el anterior, este mapa se encargará de representar círculos encima de las posiciones donde se encuentran los sensores que realizan la medida y el área que abarcan estos.

El mapa es simple ya que no trata de realizar ninguna funcionalidad más que medir la temperatura de esa zona.

- **Mapa de vientos**, según el viento que haga sustituye las banderas de los sensores de la playa para ver qué bandera se debe mostrar para los bañistas.

En este mapa, aparte de representar la medida de los vientos de la zona y también se encarga de la automatización del proceso de decisión de banderas para los bañistas en las playas cercanas a los sensores.

Estos sensores miden la velocidad del viento mediante los mensajes que van recibiendo de la aplicación Canal que cuelga en la API REST de Fi-ware, una vez los recoge los procesa y compara en la escala de Beaufort.

La escala de Beaufort es una medida empírica para la intensidad del viento, basado en el estado del mar, olas y fuerza del viento. La tabla es la siguiente:

Número escala	Fuerza del viento (km/h)	Denominación
0	0 a 1	Calma
1	2 a 5	Ventolina
2	6 a 11	Flojito
3	12 a 19	Flojo
4	20 a 28	Bonacible
5	29 a 38	Fresquito

6	39 a 49	Fresco
7	50 a 61	Frescachón
8	62 a 74	Temporal
9	75 a 88	Temporal Fuerte
10	89 a 102	Temporal Duro
11	103 a 117	Temporal Muy Duro
12	+118	Temporal Huracanado

Figura 20. Escala de Beaufort

Los colores representados en la tabla son los que tomaremos como referencia para representar en el mapa. El servlet comprará las medidas de los sensores próximos a la playa, comparará los valores obtenidos por el sensor con la tabla y mostrará una bandera en función de la comparación.

5. Implementación

Una vez presentado todas las utilidades y herramientas utilizados para realizar el proyecto, explicaremos los pasos para implementar la solución, configurar procesos y desplegarlos.

5.1 Creación de la instancia Fi-ware y entidades

En este proyecto comenzamos configurando una imagen orion-context-broker en la plataforma Fi-ware.

Esta imagen, nos permite realizar las operaciones REST que hemos explicado con anterioridad y que en este punto explicaremos como realizar estas operaciones.

Lo primero que realizamos para poder crear las entidades es crearnos una cuenta en la plataforma Fi-ware.

Para registramos entramos en https://account.lab.fiware.org/sign_up/. Una vez nos hemos registrado, nuestra pantalla principal es la siguiente:

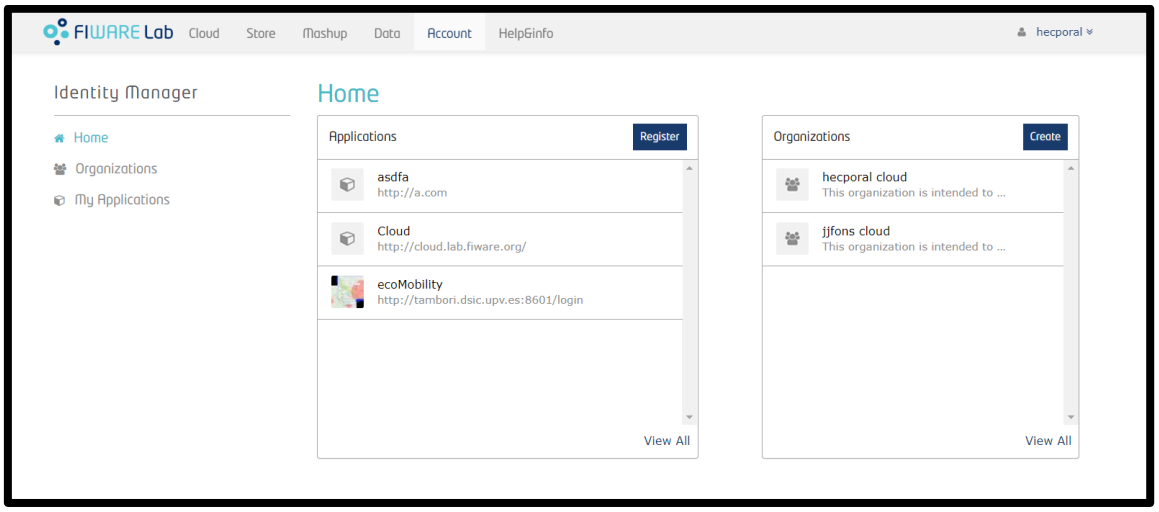

Figura 21. Pagina inicial FI-ware Lab

En esta pantalla podemos ver las organizaciones a las que pertenecemos y las aplicaciones que se nos ha asignado parte de alguna aplicación.

En esta pantalla lo que nos interesa es la pestaña de Cloud, una vez accedemos a ella nos encontraremos con esta pantalla:

Figura 22. Log In Fi-ware

Cabe destacar que no es la misma cuenta que acabamos de crear con lo cual pulsaremos a Create Account. El proceso tardó varios días en proveerme de un usuario, así que el registro no es instantáneo.

Una vez nos han provisto de una contraseña y usuario accedemos al Fi-ware Lab Cloud:

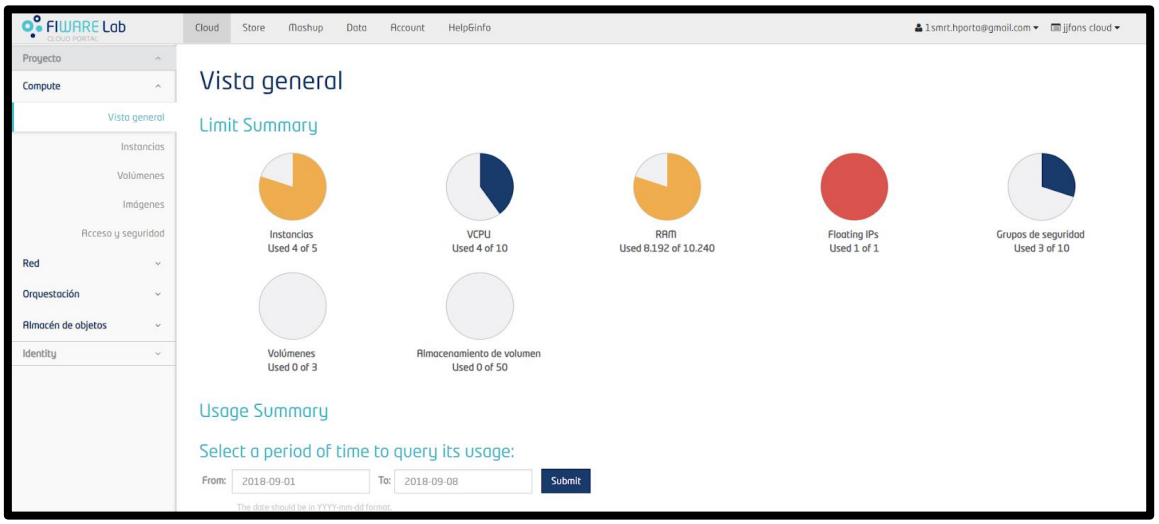

Figura 23. Vista general Fi-ware Cloud

De esta pantalla destacar varias opciones:

- **Vista General**: Nos muestra el estado actual de nuestro Cloud, podemos ver las características generales del servidor y la cantidad de recurso gastados. En este caso nos fijamos en la cantidad de Floating IP's e instancias. Son Ip's que nos provee Fi-ware para desplegar nuestra solución.

- **Instancias**: imágenes que hemos desplegado y montado en nuestra nube. Todas pueden estar en ejecución. Puede haber varias con Ip pública. En nuestro caso solo tenemos una Ip pública.

- **Imágenes**: repositorio con las imágenes que oferta Fi-ware para montar tu nube.

- **Acceso y Seguridad**: en esta zona encontraremos todo lo referente a la configuración de red de la instancia.

Para empezar a crear nuestra instancia, lo que tuvimos que hacer primero es generar nuestra IP' flotante con lo cual primero iremos a la sección Acceso y Seguridad -> Ip's flotantes. Una vez dentro clicamos en allocate Ip to Project. Una vez hecho esto ya dispusimos de una Ip para nuestro proyecto. Nos quedó algo así:

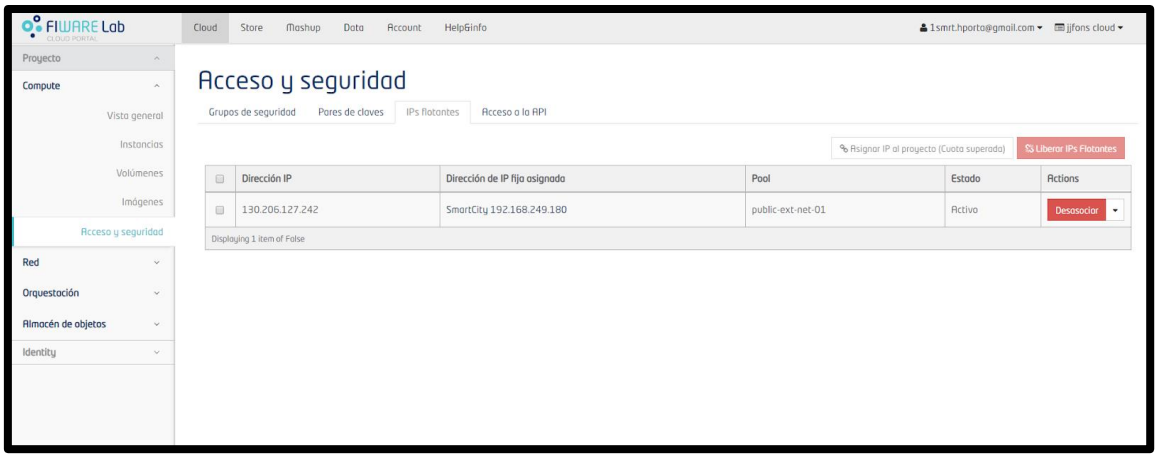

Figura 24. Acceso y Seguridad Fi-ware Cloud

El siguiente paso que realizamos fue crear los grupos de seguridad, consiste en un conjunto de regla que prohíben o permiten el acceso a diferentes puertos de la instancia. Los que nos interesan son:

- **Puerto 22**: puerto ssh para poder conectarnos a la máquina virtual y poder configurarla.

- **Puerto 1026**: puerto de comunicación en el que escucha el Context Broker

Para configurar estos grupos, accedimos a la sección de Acceso y seguridad -> Security Groups. Una vez accedemos, nos permite ver los grupos que hay creados. Para crear un grupo de seguridad clicamos en crear grupo de seguridad y le ponemos nombre. Una vez hecho esto clicamos en administrar reglas y nos aparece esta ventana:

Figura 25. Grupos de seguridad Fi-ware Cloud

Necesitamos acceder a los puertos anteriormente mencionados con lo cual pulsaremos en administrar reglas y rellenamos el formulario con el tipo de regla, puerto, tipo, protocolo. Una vez terminado nos queda así:

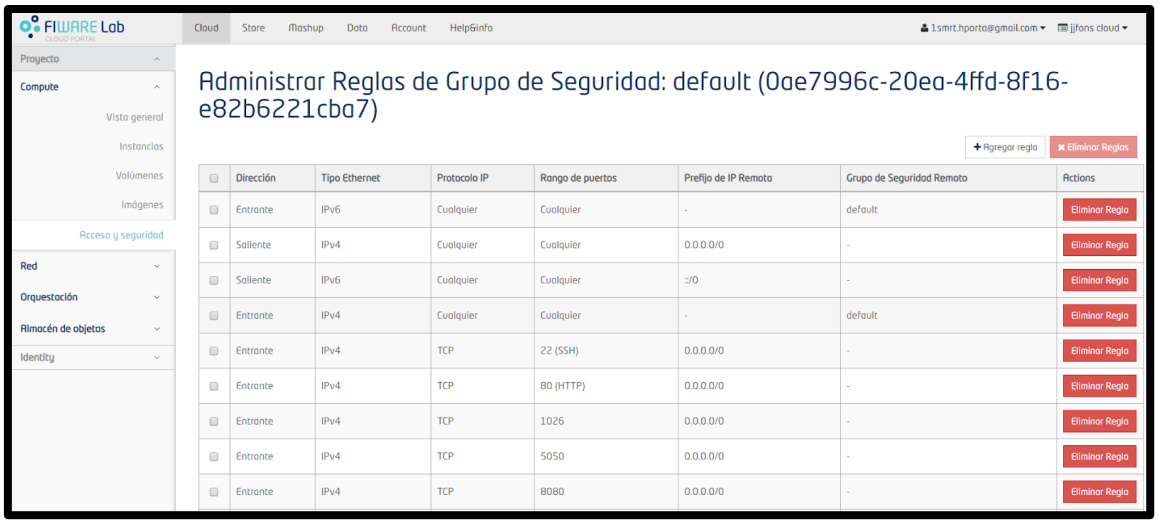

Figura 26. Reglas del grupo Fi-ware Cloud

El siguiente paso fue crear las claves para la autenticación en la instancia. Para ello accedimos a Acceso y seguridad -> Par de claves. Clicamos en crear par de claves y nos genera un archivo ".pem" que usamos más tarde para poder acceder a la imagen mediante ssh. Una vez creado, se añade a la lista de las claves quedando de esta manera:

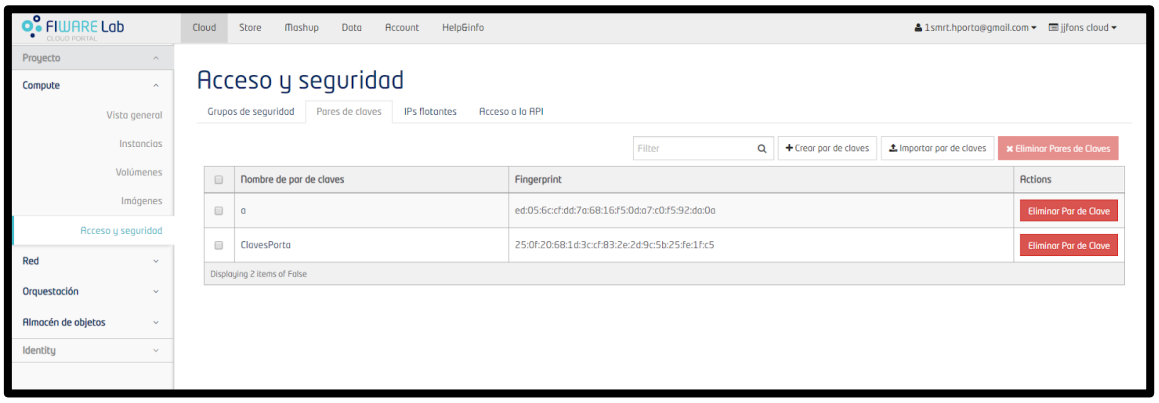

Figura 27. Par de Claves Fi-ware Cloud

Una vez tenemos esto ya tenemos todo lo necesario para crear nuestra imagen y poder empezar a trabajar con nuestro context broker.

Lo primero que tenemos que hacer es seleccionar en el apartado de imágenes y seleccionamos el orion-psb-image-R5.4. y clicamos en lanzar instancia.

Una vez hecho esto nos saltará un menú con varias pestañas, destacaremos las que más nos interesan:

- **Detalles**: información general de la instancia, aquí podemos configurar el nombre de nuestra instancia, características generales como tamaño de disco y de memoria RAM. Es necesario tener en cuenta que nuestras cuotas de disco y RAM son limitadas con lo cual tendremos que ajustar al máximo todo lo

que se pueda de espacio.

- **Acceso y seguridad**: aquí especificaremos que grupos de seguridad activaremos en la instancia, es decir a qué puertos tendremos acceso en la instancia. También configuraremos que par de claves asignaremos a la imagen y podremos acceder a ella.

- **Redes**: en este apartado seleccionamos las redes a las que queramos que pertenezca nuestra imagen. En este caso seleccionaremos solo la red 1.

Una vez finalicemos estos pasos nos devolverá la página de instancias con la instancia ya creada.

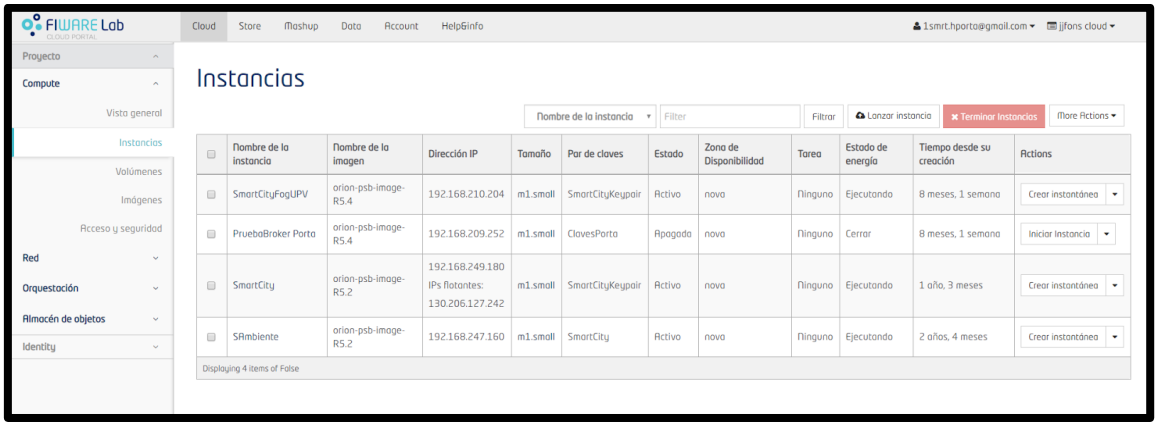

Figura 28. Instancias Fi-ware Cloud

[10] Una vez finalizado podremos acceder a la instancia mediante un comando ssh, para ello necesitaremos un cliente tty para acceder al sitio, en nuestro caso hemos optado por la terminal que proporciona Windows ya que emula las características de una terminal.

Antes de conectarnos, necesitamos cambiar los permisos de la key ya que por defecto viene con permisos 777 y la imagen lo permite el acceso de una key con permisos completos con lo cual se asigna a esa key permisos 400. Para conseguir esto debemos ejecutar:

chmod -R 400 <ruta_de la key>

Para acceder a la terminal necesitaremos la <key>.pem que se genera cuando creamos el par de claves. Una vez lo tenemos ejecutamos:

ssh -i <ruta_del_archivo_pem> -d <usuario@ip-imagen>

La principal operación que debemos realizar es el cambio de contraseña para poder acceder a root en la imagen, para ello ejecutamos el comando *passwd.*

Otro de lo procedimiento que debemos ejecutar es la activación del módulo de comunicaciones para ello ejecutamos el comando:

/etc/init.d/contextBroker start

Una vez hecho esto ya tendremos todo configurado la máquina virtual para atender peticiones REST.

Una vez activo el Context Broker ya podemos utilizar la API REST para realizar peticiones y crear las entidades necesarias para realizar el caso de estudio.

El siguiente paso fue diseñar y crear las entidades que íbamos a utilizar para realizar el caso de estudio. En este caso solo hemos necesitado una entidad para realizar el intercambio de datos ya que lo único que necesitamos transmitir son los mensajes.

El diseño de esta entidad Sensor es la siguiente:

```
    {
"id": "SensorP1",
"type": "String",
"metadata": {
    "key": "String",
    "name": "String",
    "trademark": "String"
},
"ns": "String",
"sensor_data": {
    "position": [
        "float",
        "float"
    ],
    "version": "String",
    "link_date": "String",
    "battety": int,
    "status": "String",
```

```
        "frequency": "float"
    },
    "message_data":{
        "msgtype": "String",
        "unit": "String",
        "measure": "int"
    }
        }
```

La entidad sensor tiene incorpora estos atributos:

- **id**: identificador de la entidad, también nos servirá para recuperar datos de una entidad en concreto. Sirve como nombre a la hora de recuperar los datos.

- **type**: tipo de sensor, en este campo identificamos los sensores según el tipo de medida que envían. Este dato lo envían tanto los JSON como los XML. No servirá para filtrar los mensajes a la hora de tratarlos en las páginas web.

- **metadata**: datos generales del sensor, es un objeto JSON para mejorar la organización de estos datos y no tenerlos sueltos por el mensaje. Nos permite descartar los metadatos sin necesidad de leerlos.

    - **key**: clave generada por el sensor para identificarse y tener una manera de poder asegurar que el emisor del mensaje es el propio sensor.

    - **name**: nombre de la máquina, identificador no único del equipo. Este valor puede ser obtenido mediante el login del sensor con el API. No se representa en este ejemplo.

    - **trademark**: nombre del fabricante del sensor, datos automáticos generados por el fabricante.

- **ns**: número de serie. Identificador de fábrica del dispositivo.

- **sensor_data**: datos sobre el sensor. En este objeto no se incluye el mensaje, pero si se incluye su posición y estado. Agrupándolos de esta manera lo que conseguimos que todos los datos pertenecientes al sensor.

    - **position**: posición física en el mapa real. Detectada mediante GPS. En este caso nosotros decidimos en qué posición quedan. Representado mediante latitud y longitud.

    - **version**: versión del software que montan los sensores.

    - **link_date**: fecha de unión del sensor con la entidad Fi-ware

    - **battery**: batería del sensor. Si el sensor está conectado mediante corriente

    - **status**: estado del sensor.

    - **frequency**: frecuencia de actualización de los mensajes, puede variar si el sensor adopta varios tipos de frecuencias.

- **message-data**: datos pertenecientes a las medidas que se toman mediante el sensor, el tipo es estándar para que se pueda utilizar independientemente del tipo de mensaje que se mida.

    - **msgtype**: tipo de mensaje que enviaremos, en este caso serie Viento, Temperatura o CO.

    - **unit**: unidades de medida del sensor que utilizaremos, en este caso serían km/h, ºC o ppp.

    - **measure**: medida tomada por el sensor, se representa en float.

Una vez hemos presentado la entidad que utilizaremos crearemos 20 entidades representados por id SensorXX. Para crear y manipular estas entidades utilizaremos varias peticiones HTTP las cuales serán: GET, POST y PUT cuyas equivalencias serian recuperar, crear y actualizar.

POST (Creación de entidades)

La primera operatoria que con la que pude familiarizarme fue la petición de creación POST. Requerimos primero crear la entidad ya que sin entidades no se puede hacer consultas a nada con lo cual sigue un orden lógico.

Las peticiones POST las podemos realizar de dos maneras. Uno es especificando completamente la entidad o podemos crear la entidad solamente con los atributos y valores. Ahora explicaremos como mediante la aplicación de escritorio POSTMAN:

- POST con entidades completas

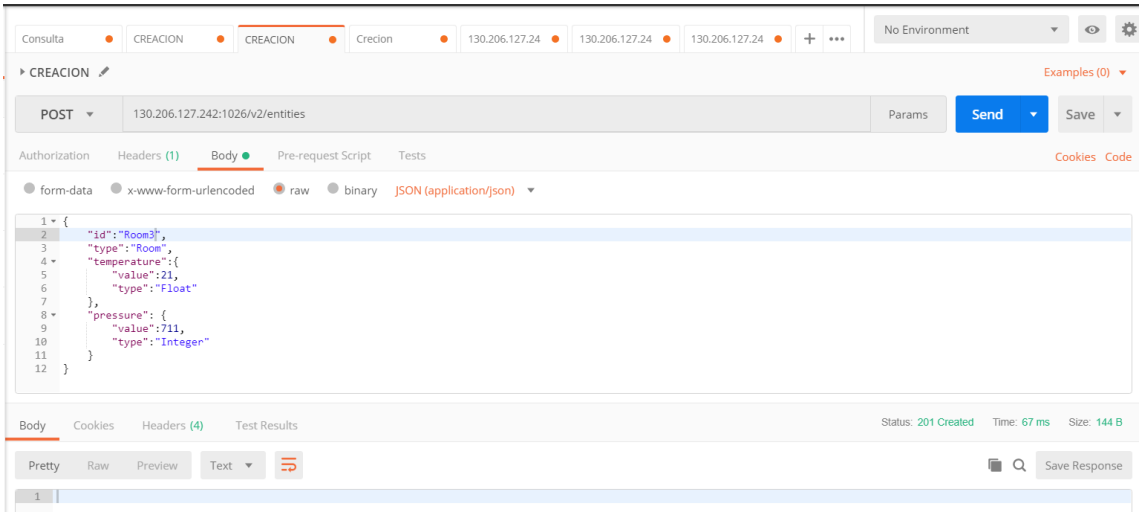

Figura 29. POSTMAN: Ejemplo de POST con entidades completas

De esta manera necesitamos crear la entidad con sus atributos con dos valores: el valor y el tipo. Si no pusiéramos los valores como en la imagen, nos daria un error de petición y nos rechazaría el cambio.

- POST sin entidades completas

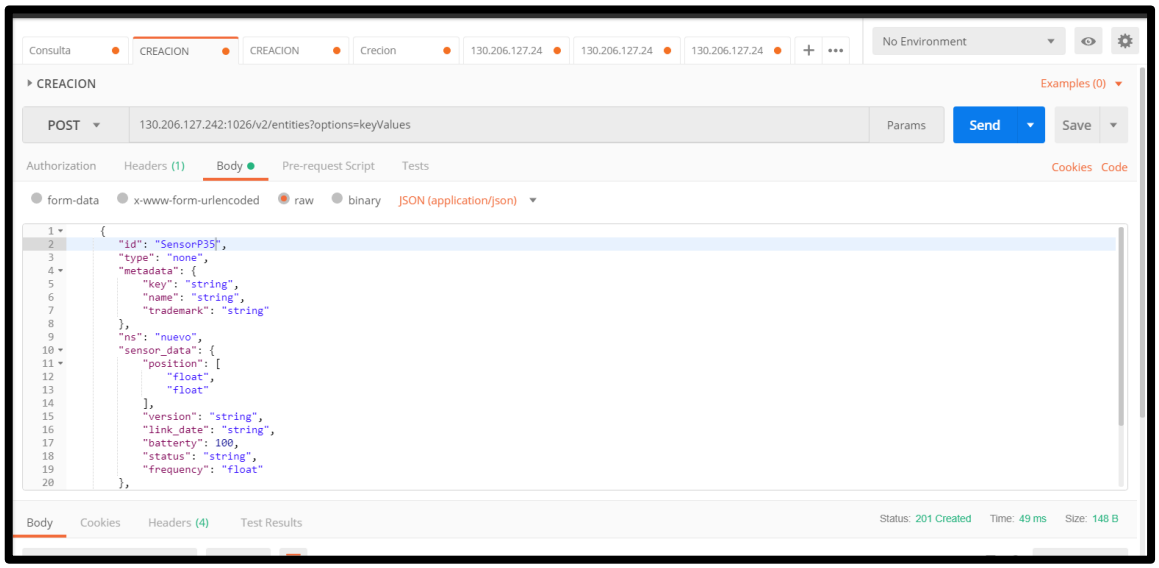

Figura 30. POSTMAN: Ejemplo de POST con la opción de keyValues

Como podemos comprobar en el enlace vemos que hemos añadido una opción: keyValues. Esta opción nos permite crear valores sin necesidad de establecer el tipo, simplemente poniendo el atributo y su valor. La desventaja es que te crea campos extravacíos como ahora comprobaremos en el siguiente punto.

GET (Consulta de entidades)

La siguiente en orden natural sería la consulta de entidades, lo curioso de la consulta es que se puede plantear de diferentes maneras, ya sea sobre una entidad completa, sus key values o un atributo en concreto. En este caso explicaremos las dos maneras que se utiliza en el proyecto.

- GET completo a entidad solo a sus keyValues

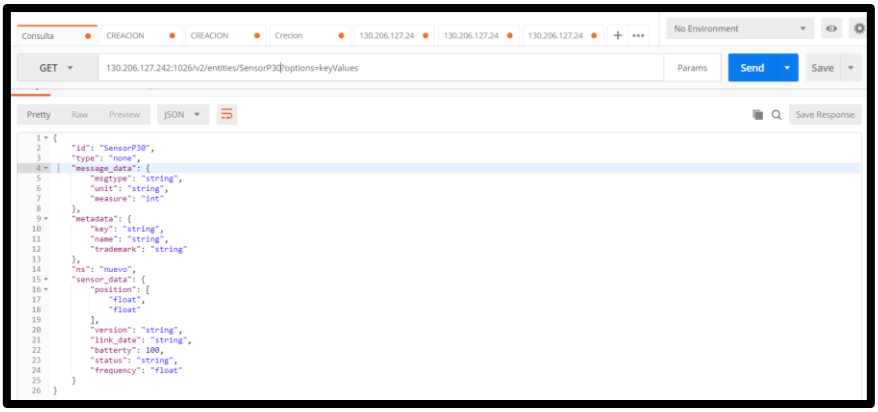

Figura 31. POSTMAN: Ejemplo de GET con keyValues

Como podemos observar hemos hecho el GET solo sobre los valores que nos interesan. Para el proyecto es más útil solo obtener los valores directamente ya que un exceso de atributos puede dar problemas. Ahora veamos la entidad sin los keyValues:

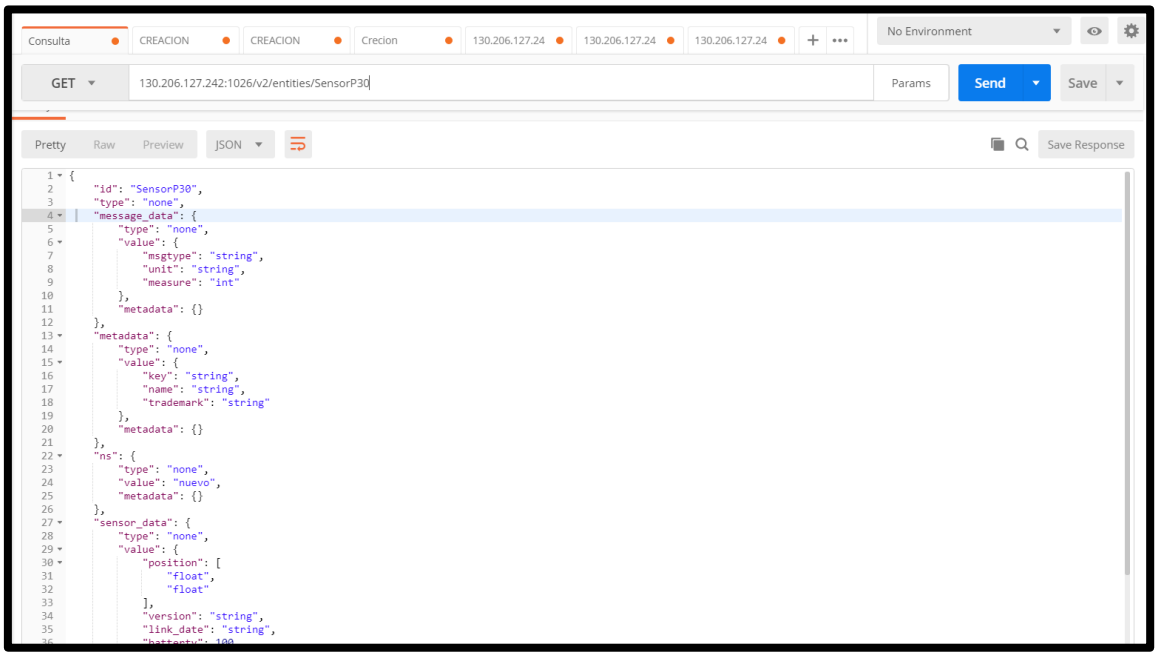

Figura 32. POSTMAN: Ejemplo de GET entidad completa

Vemos que la cantidad de atributos es mayor es más difícil de interpretar.

- GET sobre un atributo en concreto con keyValues

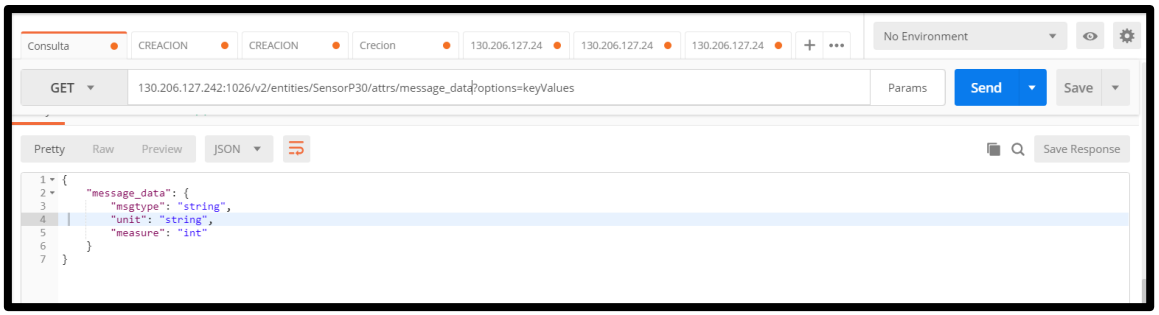

Figura 33. POSTMAN: Ejemplo de GET de atributos con keyValues

Comprobamos que obtenemos los datos sin los campos extra que nos añade Fi-Ware.

PUT y DELETE (Operación de actualización y Borrado)

Por último, nos centramos en las operaciones de manejo de datos, ya que estas operaciones son más críticas. PUT permite la actualización de los valores de forma general, es decir, a todos los atributos o solo a algunos en concreto.

Esta función se puede combinar con la opción de keyValues y así facilitando la actualización de estas entidades.

La función DELETE nos permite borrar entidades completas que no estemos utilizando o que hayamos creado por error. Vemos cómo se usan:

- Estado inicial

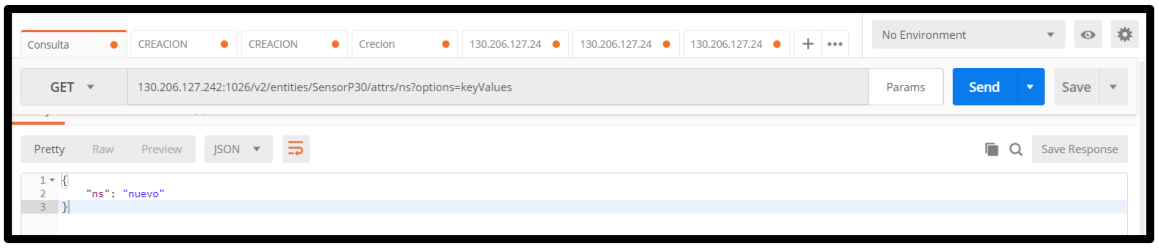

Figura 34. POSTMAN: Ejemplo de GET de atributos con keyValues antes de actualizar

- Función PUT para cambio de atributo

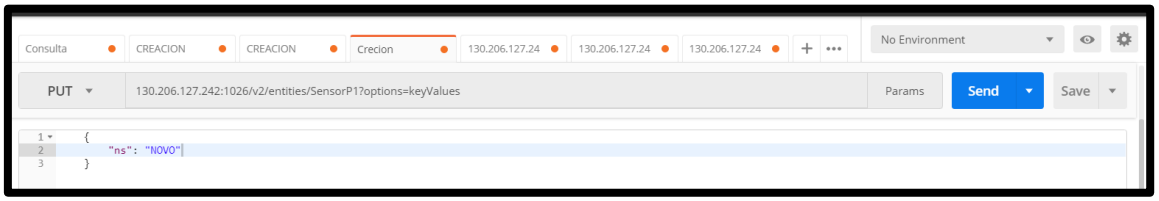

Figura 35. POSTMAN: Ejemplo de PUT de atributos con keyValues

- Estado final

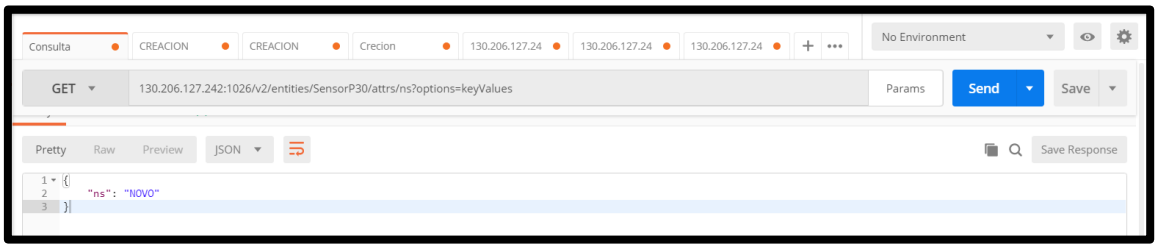

Figura 36. POSTMAN: Ejemplo de GET de atributos con keyValues ya actualizados

Para realizar un DELETE tendríamos que ejecutar una petición y poner en la ruta la entidad a borrar:

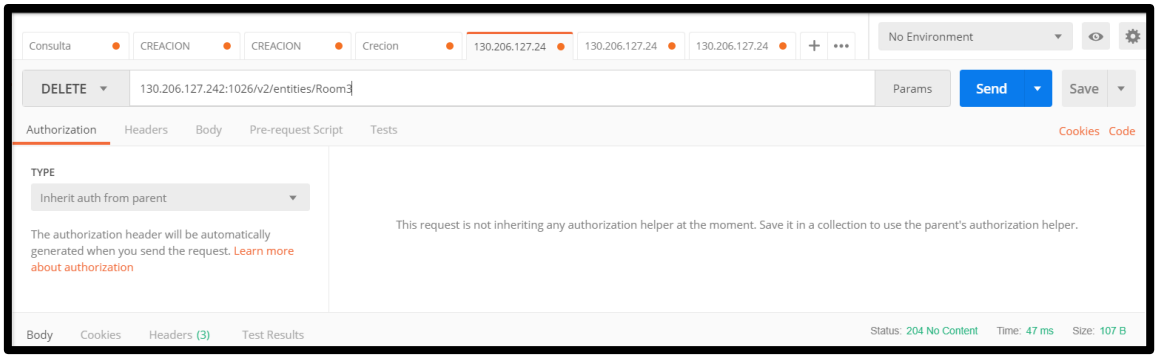

Figura 37. POSTMAN: Ejemplo de DELETE de una entidad

Todas las pruebas realizadas para testear y implementar en el Fi-ware se han realizado mediante la aplicación de escritorio POSTMAN, pero no es la única manera de crear entidades. Mediante cUrl podemos realizar todas estas peticiones mediante la terminal de comandos de Linux. Se eligió POSTMAN por su entorno amigable y las herramientas extras que incluye.

5.2 Implementación Java de Canal y Sensor

Para realizar la simulación de un canal transformador de mensajes y unos sensores se han desarrollado 2 aplicaciones ya antes descritas. Su funcionalidad consiste en emular estos componentes y realizar sus funciones.

Sensor

El sensor se ha desarrollado para que envíe 200 mensajes, 1 cada 10 segundos. Se programó de esta manera para poder revisar en el momento de la ejecución que podría estar fallando y comprobar que funciones correctamente.

El código ejemplo desarrollado en este proyecto lo podéis encontrar en el Anexo A.

Las librerías más importantes usadas en esta aplicación son:

- Java.io: Libreria java estandar que se encarga del proceso de input y output
- Java.io.file: Librería java para realizar acciones con ficheros almacenados en el sistema. Sobre todo, es importante en la parte de tratamiento con XML.
- Rabbit.mqtt: Librería que se encarga de la comunicación entre procesos.

Canal

El canal es el proceso que se encarga de recoger los mensajes que cuelgan los sensores en la cola rabbitMQ, una vez los recoge, los procesa según sea JSON o XML y una vez los transforma se encarga de enviarlos a la entidad Fi-ware correspondiente.

El código ejemplo desarrollado en este proyecto lo podéis encontrar en el Anexo B.

Las librerías más importantes en esta aplicación son:

- okhttp3: librería que se utiliza para realizar las peticiones http que están basadas en el código que genera POSTMAN automáticamente para ejecutar en ciertos entornos. En este caso java.
- java.io.File: librería que se utiliza para descargar el texto xml del mensaje recibido. La única solución que se encontró contra el problema de no poder parsear un string como XML. Se decidió descargar el texto en un fichero y una vez creado el fichero, se analiza y procesa.
- RabbitMQTT: librería que se utiliza para la comunicación entre proceso y cola. consiste en emular estos componentes y realizar sus funciones.

5.3 Implementación de los servlet

Para demostrar la funcionalidad se ha propuesto crear un servlet con 3 mapas que permiten conocer la cantidad de CO2 en una zona en concreto, ver si hay bandera roja en las playas y medir su velocidad y conocer la temperatura por zona.

Mapa de contaminación de CO2

Figura 38. POSTMAN: Ejemplo de DELETE de una entidad

Aquí podemos ver un ejemplo de cómo plasmar la información que hemos recogido mediante los sensores y poder comprobar en este preciso momento que zonas son las más contaminadas de la ciudad.

Mapa de temperaturas

Figura 39. POSTMAN: Ejemplo de DELETE de una entidad

Podemos ver que según las medidas que marcan los sensores, las áreas que cubren estos cambian de color en función de la temperatura que haga en la zona. A parte cada uno de los puntos proporciona información sobre la temperatura referente a la zona.

Mapa de viento

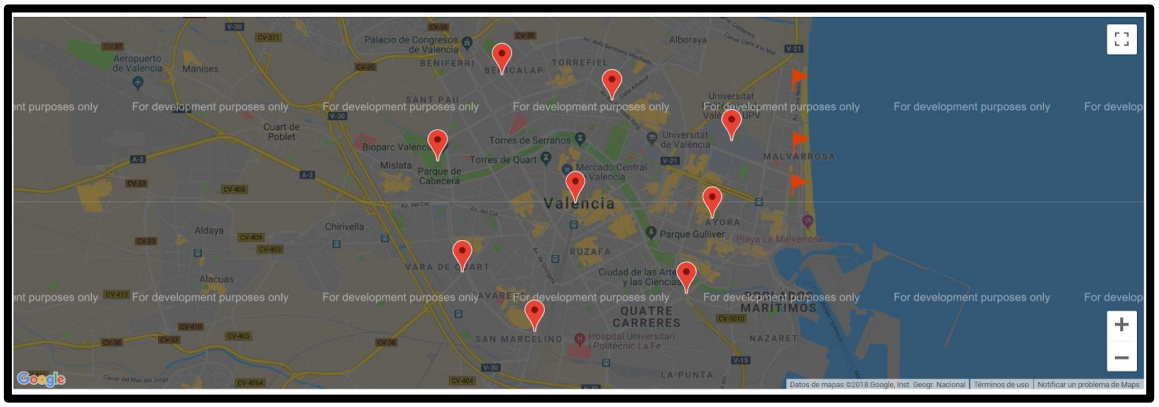

Figura 40. POSTMAN: Ejemplo de DELETE de una entidad

Como podemos comprobar, por último, en el mapa de vientos, vemos los sensores que están marcando la zona en la que están y en la parte derecha donde está la cosa, los iconos que se representan en el mapa son banderas, que según el estado del viento, cambian de color si es seguro o no para el bañista, verde es seguro, amarillo precaución y rojo prohibido.

6. Conclusión

Cuando elegí el proyecto me sorprendió la idea de Smart Cities y de ecoMovilidad o conceptos como internet de las cosas. Una vez hecho el proyecto no me cabe la menor duda de que esta parte de la tecnología será una de las más importantes del mundo, no solo por sus posibles futuras aplicaciones. Sino por la enormidad de sistemas que puede conectar y crear comunicación entre ellas.

A lo largo del proyecto me he ido dando cuenta de que la sociedad va evolucionando hacia un sistema organizado general. Ya tenemos bicicletas que calculan trayectos más cortos, coches que informa sobre un accidente a miles de usuario en plena marcha del coche, los primeros coches auto pilotados. Es una realidad que nos tendremos que amoldar al hecho de que absolutamente todos los datos que se cuelgan en internet, que tienes en las administraciones públicas o twits que cuelgas en tu página, se comunicará y se relacionarán creando un mundo de información delante tuya.

He dedicado todo mi esfuerzo en este trabajo y he aquí el resultado. Una de las partes que más costó fue el tema de redacción, cuando te plantean un tema general con tanta libertad puedes plantearte miles de formas de desarrollar el proyecto, pero hay que dar con la tecla correcta para acertar con el tema del proyecto correcto. Otro de los problemas más graves fue la plataforma Fi-ware. Se entiende que Fi-ware es una plataforma en constante evolución y que suelen cambiar las aplicaciones más de vez en cuando y los métodos de ejecución de instrucciones los cambian completamente. Pero una vez adaptado de nuevo.

En este proyecto he podido relacionar las asignaturas de integración de aplicaciones del cual el responsable de la asignatura es mi profesor-tutor, no llegue a imaginar la envergadura de la empresa que es una smart city y todo ello se ha plasmado en intentar conseguir el mejor resultado posible. Además de IAP, he podido relacionarlo con desarrollo y diseño web, planteando un front-end enfocado al usuario y ofreciendo servicios que pueden ser de utilidad para el usuario.

A pesar de que solo se haya representado la medida de los sensores que hemos desplegado por la ciudad, este sistema podría proporcionar muchísimas más soluciones como: desvío de transporte público en función de contaminación, localización de transporte público en directo, etc…

Con este trabajo he intentado plasmar todo lo aprendido en la carrera que es empeño, constancia y trabajo, mucho trabajo.

La valoración general del proyecto considero positivamente la resolución del mismo, creo que se han alcanzado los objetivos que no hemos intentado proponer y se podrían plantear mejoras a este trabajo pero para lo que nos habíamos propuesto es más que suficiente.

Por último, me gustaría agradecer a mi tutor el apoyo y la ilusión que tenía en el proyecto que el propuesto y me aceptó para realizarlo. La paciencia que ha tenido en los momentos en los que tenía los nervios a cien y agradecer todo el esfuerzo que ha dedicado a ayudarme.

Bibliografía

[1] Articulo sobre las Smart Cities, publicado en panelesach el 9 de enero de 2018. Consultado en https://bit.ly/2meGl1q

[2] Articulo sobre ventajas y requisitos de las Smart Cities, publicado en economipedia por Janire Carazo.

Consultado en https://bit.ly/2MjyDNK

[3] Documentación sobre Fi-ware.

Consultado en https://bit.ly/1WZdHfc

[4] Principales características de java, publicado en UPV.es.

Consultado en https://bit.ly/2CbVdYH

[5] Documentacion sobre eclipse,

Consultado en https://bit.ly/2O32qM8

[6] Documentacións sobre Ubuntu for windows, publicado en microsoft.

Consultado en https://bit.ly/2oRXI8T

[7]Documentación API REST, publicado en restapitutorial.

Consultado en https://bit.ly/2wXccJf

[8]Documentación RabbitMQ, publicado en rabbitmq.

Consultado en https://bit.ly/1GLTgJ3

[9]Documentación POSTMAN, getpostman.

Consultado en https://bit.ly/19MnN02

[10] API sobre orion context bróker, publicado en fiware-orion.

Consultado en https://bit.ly/2wUhWnc

Anexo A.- Código Sensor.java

```
import java.io.File;
import java.io.IOException;
import java.io.StringWriter;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeoutException;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import java.math.*;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.io.SAXContentHandler;
import org.dom4j.io.SAXReader;
import org.json.JSONException;
import org.json.JSONObject;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class Main {

	private final static String QUEUE_NAME = "publicaciones";
	private final static String tiposMensaje[] = {"JSON","XML"};
	public static void main(String[] args) throws InterruptedException,
DocumentException, XMLStreamException, IOException, JSONException {
		// TODO Auto-generated method stub
		ConnectionFactory factory = new ConnectionFactory();
		factory.setUsername("mqadmin");
		factory.setPassword("mqadminpassword");
		factory.setHost("192.168.0.222");
		Connection connection;
		Scanner s = new Scanner(System.in);
		Channel channel = null;
		try {
			connection = factory.newConnection();
			channel = connection.createChannel();
		} catch (IOException | TimeoutException e) {
```

```
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                int n = 0;

                while(n<200){
                        double indice = Math.floor(Math.random() * 2);
                        String tipox = tiposMensaje[(int)indice];
                        String msg = generarMensaje(tipox);
                        n++;
                        try {

                                channel.basicPublish("", QUEUE_NAME, null,
msg.getBytes());
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        Thread.sleep(10000);
                }


        }


        public static String generarMensaje(String tipox) throws DocumentException,
XMLStreamException, IOException, JSONException{
                String a[] = {
                                "{latitud: 39.46981, longitud: -0.377011, clave:
jb9N8j2jsM3BZBJd6GzVfVDn8wylx4nSEW3lfZLKsmKAgiJpUZPbJAPiSEJ5ODo}",
                                "{latitud: 39.480858,longitud: -0.341005, clave:
Vh3qJ7yyz3cjDITr7LVXoK7cRL9L0nli7lUT65PUU7bs7zTlXs3Fs53Q63W594vaqQ}",
                                "{latitud: 39.487978, longitud: -0.368523, clave:
6U3ItV7Ssz6MBswDRluRfi9aYzyHF65uplEamAsMvWD67f36wdL8v9h5hvpVCm}",
                                "{latitud: 39.492701, longitud: -0.393937, clave:
wGrlhvPWC2OAmW4qBDu5aUkzClPCJ9jbR91e0TqoMLEVNJb5LnijT74w01JCg8}",
                                "{latitud: 39.477288, longitud: -0.408786, clave:
XETAapfoebPXV979CcVYlgLOYbRS2LkYjaQbq78AQCmN2Ev0vRDdB7N6GxX3}",
                                "{latitud: 39.457595, longitud: -0.403052, clave:
RUP1enpllx3Exy7QXl6Hf5cnuubSuTuZpHz1fFdXnJDAO6JW9cTHBz2V0cqhVc74}",
                                "{latitud: 39.446899, longitud: -0.386349, clave:
8keI8AjsP1YNap0qLQitIowCO0jDo9saIjAXocsOwVxkXShIvupzqloGY9Q81FCB4C}",
                                "{latitud: 39.453712, longitud: -0.351433, clave:
GHlC84Ny97pxQ2TFLBAeNzVoXkOWtve6HL7O6NlnJJCtwYBfPWAtHBWkYKlL}",
                                "{latitud: 39.467049, longitud: -0.345356, clave:
aas1dan0qhcrFYQbgS8WqZZe9OcI2Kw8fGwN9vV1QE4C5I4QERI6aOVPd6uuvy}",
```

```
                                    "{latitud: 39.470924, longitud: -0.325316, clave:
cPTVmO9ixqMpNm4dFAbqksvv7UJs5XFHSHQQWhyUVG0PxvkPlj7as3tqGCYy}",
                                    "{latitud: 39.47853 , longitud: -0.324904, clave:
6eyUcZiaiOYl8pBEBJa1RIAfh0wxPePLO5UrJb4wfkdooZNSc1EqYc3rjodUg7NIv}",
                                    "{latitud: 39.48958, longitud: -0.325178, clave:
mp62aPX9zkiOF7ClNcxgfWmoC0jmsdEjjyngdebwFDdQVJmHGrGuBglFmqvEI}"
                                    };
                        String b[] = {
                                    "{tipo: CO, unidad: ppm}",
                                    "{tipo: Temperatura, unidad: C}",
                                    "{tipo: Viento, unidad: kmh}"
                        };

                        double valor;
            double valorc = Math.floor(Math.random() * 30) + 1;
            double valorppm = Math.floor(Math.random() * 400) + 1;
            double valorkm = Math.floor(Math.random() * 100) + 1;
            JSONObject tipo = new JSONObject(b[(int) Math.floor(Math.random() * 3)]);
            if(tipo.getString("tipo").equals("Temperatura")) {
               valor = valorc;
            }
            else if(tipo.getString("tipo").equals("Viento")) {
               valor = valorkm;
            }
            else {
               valor = valorppm;
            }
            JSONObject coordenadas = new JSONObject(a[(int) Math.floor(Math.random()
* 12)]);
            java.util.Date fecha = new Date();
            Calendar c1 = Calendar.getInstance();

            String dia = Integer.toString(c1.get(Calendar.DATE));
            String mes = Integer.toString(c1.get(Calendar.MONTH));
            String annio = Integer.toString(c1.get(Calendar.YEAR));
            String factual = dia+"-"+mes+"-"+annio;
            String res = null;
            if(tipox.equals("XML")) {
                              StringWriter stringWriter = new StringWriter();
                        XMLOutputFactory xMLOutputFactory =
XMLOutputFactory.newInstance();
                        XMLStreamWriter xMLStreamWriter =
xMLOutputFactory.createXMLStreamWriter(stringWriter);
                        xMLStreamWriter.writeStartDocument();
                        xMLStreamWriter.writeStartElement("sensor-info");

                        xMLStreamWriter.writeStartElement("key");
```

```
                xMLStreamWriter.writeCharacters(coordenadas.getString("clave"));
                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeStartElement("version");
                xMLStreamWriter.writeCharacters("12.03");
                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeStartElement("message");

                xMLStreamWriter.writeStartElement("type");

                xMLStreamWriter.writeCharacters(tipo.getString("tipo"));
                xMLStreamWriter.writeEndElement();
                xMLStreamWriter.writeStartElement("value");

                xMLStreamWriter.writeCharacters(String.valueOf(valor));
                xMLStreamWriter.writeEndElement();
                xMLStreamWriter.writeStartElement("unit");
                xMLStreamWriter.writeCharacters(tipo.getString("unidad"));
                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeStartElement("position");

                   xMLStreamWriter.writeStartElement("latitude");

          xMLStreamWriter.writeCharacters(String.valueOf(coordenadas.getDouble("la
titud")));
                   xMLStreamWriter.writeEndElement();
                   xMLStreamWriter.writeStartElement("length");

          xMLStreamWriter.writeCharacters(String.valueOf(coordenadas.getDouble("lo
ngitud")));
                   xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeStartElement("link-date");
                xMLStreamWriter.writeCharacters(factual);
                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeStartElement("battery");
                xMLStreamWriter.writeCharacters("75");
                xMLStreamWriter.writeEndElement();

                xMLStreamWriter.writeStartElement("status");
```

```
            xMLStreamWriter.writeCharacters("online");
            xMLStreamWriter.writeEndElement();

            xMLStreamWriter.writeStartElement("frequency");
            xMLStreamWriter.writeCharacters("10000");
            xMLStreamWriter.writeEndElement();


            xMLStreamWriter.writeEndElement();
            xMLStreamWriter.writeEndDocument();

            xMLStreamWriter.flush();
            xMLStreamWriter.close();

            res = stringWriter.getBuffer().toString();

            stringWriter.close();
            return res;
    }
    else {
       JSONObject mensaje = new JSONObject();
       mensaje.put("type", tipo.getString("tipo"));
       mensaje.put("measure", valor);
       mensaje.put("unit", tipo.getString("unidad"));

       JSONObject posicion = new JSONObject();
       posicion.put("latitude", String.valueOf(coordenadas.getDouble("latitud")));
       posicion.put("length", String.valueOf(coordenadas.getDouble("longitud")));

       JSONObject sensorinfo = new JSONObject();
       sensorinfo.put("key", coordenadas.getString("clave"));
       sensorinfo.put("version", "12.03");
       sensorinfo.put("message", mensaje);
       sensorinfo.put("position", posicion);
       sensorinfo.put("link-date", factual);
       sensorinfo.put("battery", 75);
       sensorinfo.put("status", "online");
       sensorinfo.put("frequecy", 10000);



       return sensorinfo.toString();


    }
            //System.out.println(xmlString);
```


```
        }
}
```

Anexo B.- Código Canal.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.io.Writer;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Iterator;
import java.util.concurrent.TimeoutException;

import javax.swing.text.Document;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

import org.dom4j.io.SAXReader;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.XML;

import com.rabbitmq.client.AMQP;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Consumer;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;
```

```
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;

public class Main {
        private final static String QUEUE_NAME = "publicaciones";

        public static void main(String[] args) throws IOException, TimeoutException {
                ConnectionFactory factory = new ConnectionFactory();
           factory.setHost("localhost");
           Connection connection = factory.newConnection();
           Channel channel = connection.createChannel();

           channel.queueDeclare(QUEUE_NAME, false, false, false, null);
           System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

           Consumer consumer = new DefaultConsumer(channel) {
                   @Override
                   public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                                   String message = new String(body, "UTF-8");
                                   JSONObject J = new JSONObject();
                    try {
                          JSONObject nuevo = new JSONObject(message);
                          System.out.println("Mensaje recibido de tipo JSON");
                                           System.out.println(message);
                                           JSONObject b = new
JSONObject(nuevo.get("message").toString());
                                           J.put("msgtype", b.get("type"));
                                           J.put("unit", b.get("unit"));
                                           J.put("measure", b.get("measure"));

                                     } catch (JSONException e) {
                                           System.out.println("Mensaje recibido de tipo
XML");
                                           String ruta = "./a.xml";
                                   File archivo = new File(ruta);
                                   BufferedWriter bw;
                                   if(archivo.exists()) {
                                      bw = new BufferedWriter(new FileWriter(archivo));
                                      bw.write(message);
                                   } else {
                                      bw = new BufferedWriter(new FileWriter(archivo));
                                      bw.write(message);
```

```
                                }
                                bw.close();
                                XMLInputFactory factory =
XMLInputFactory.newInstance();
                                try {
XMLEventReader eventReader = factory.createXMLEventReader(new
FileReader("a.xml"));

                                                        String anterior2 = "fallo";
                                                        String anterior = null;
                                                        while(eventReader.hasNext()) {
                                                         XMLEvent event =
eventReader.nextEvent();

                                                        switch(event.getEventType()) {

                                                        case
XMLStreamConstants.START_ELEMENT:
                                                                StartElement startElement =
event.asStartElement();
                                                          String qName =
startElement.getName().getLocalPart();
                                                          if(!qName.equals("sensor-info") &&
!qName.equals("message") && !qName.equals("position")) {
                                                                // System.out.println(qName);
                                                           anterior2 = qName;
                                                          }
                                                        break;

                                                        case
XMLStreamConstants.CHARACTERS:
                        Characters characters = event.asCharacters();
                                                                //System.out.println(anterior2);

                    //System.out.println(characters.getData());
                                                                if(anterior2.equals("type")) {
                                                                        J.put("msgtype",
characters.getData());
                                                                }
                                                                if(anterior2.equals("unit")) {
                                                                        J.put("unit",
characters.getData());
                                                                }
                                                                if(anterior2.equals("measure")) {
                                                                        J.put("measure",
characters.getData());
                                                                }
```

```
                                        break;
                                        default:
                                                break;
                                }
                        }
                                File fichero = new File("a.xml");
                                fichero.delete();
                        } catch (XMLStreamException | JSONException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                URL url = new
URL("130.206.127.242:1026/v2/entities?idPattern=^SensorP[1-15]");
                HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
                conn.setRequestMethod("GET");
                BufferedReader rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
                String line;
                Writer result = null;
                while ((line = rd.readLine()) != null) {
                                result.append(line);
                }
                rd.close();
                        JSONArray g = null;
                                try {
                                        g = new JSONArray(result.toString());
                                } catch (JSONException e2) {
                                        // TODO Auto-generated catch block
                                        e2.printStackTrace();
                                }
                  for(int i = 0; i<g.length();i++) {
                     try {
                                                JSONObject k =
g.getJSONObject(i);
                                                String a = k.get("key").toString();
                                                if(a.equals(J.get("key"))){
                                                        OkHttpClient client = new
OkHttpClient();
                                                        String fin = J.toString();
                                                        MediaType mediaType =
MediaType.parse("application/json");
                                                        RequestBody body1 =
RequestBody.create(mediaType, fin);
                                                        Request request = new
Request.Builder()
```

```
            .url("http://130.206.127.242:1026/v2/entities/"+k.get("id").toString()+"/attrs/message_data/value?options=keyValues")
                                                                .put(body1)
                                                                .addHeader("Content-Type", "application/json")
                                                                .addHeader("Cache-Control", "no-cache")
                                                                .addHeader("Postman-Token", "c19be3a6-a3b6-43f6-9aef-ebca78c62973")
                                                                .build();

                                                            Response response = client.newCall(request).execute();

                                                                    }
                                                            } catch (JSONException e1) {
                                                                    // TODO Auto-generated catch block
                                                                    e1.printStackTrace();
                                                            }

                                        }

                                          }
                                  }
                                };
                                channel.basicConsume(QUEUE_NAME, true, consumer);
                        }
}
```