UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESPLIEGUE DE PROYECTOS PARA LA MONITORIZACIÓN EN SMARTCITIES USANDO NODERED

Mario Berbel Bueno

Tutor: Juan Carlos Guerri Cebollada

Cotutor: Pau Arce Vila

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 3 de diciembre de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

Este proyecto tiene como objetivo integrar distintos servicios IoT a través de dispositivos de fácil acceso como RaspberryPi o SenseHat, utilizando la herramienta Node-RED y usando bases de datos como MongoDB o servicios cloud como FiWare para el almacenamiento de los datos obtenidos. Integra la utilización de una serie de sensores atmosféricos para la monitorización en dashboard de estos. También utiliza distintas APIs de diferentes webs para la integración de sus servicios en el dashboard. Otra parte importante es la integración del dispositivo móvil Android, que permite la comunicación de este con la Raspberry Pi y con el dashboard de Node-RED.

Resum

Este projecte té com a objectiu integrar diferents servicis IoT a través de dispositius de fàcil accés com RaspberryPi o SenseHat, utilitzant la ferramenta Node-RED i gastant bases de dades com MongoDB o servicis cloud com FiWare per a l'emmagatzemament de les dades obtingudes. Integra l'utilització d'una sèrie de sensors atmosfèrics per a la monitorització en dashboard d'estos. També utilitza distintes APIs de diferents webs per a l'integració dels seus servicis en el dashboard. Una altra part important és l'integració del dispositiu mòbil Android, que permet la comunicació d'aquest amb la Raspberry Pi i amb el dashboard de Node-RED.

Abstract

This project has the purpose of integrate different IoT services through easy access devices such as RaspberryPi or SenseHat, using programming tools like Node-RED and using databases like MongoDB or cloud services such as FiWare, with the objective of storing data. Integrates the use of different types of atmospheric sensor in order to show this data in a dashboard. Also, uses different APIs of different websites to integrate this services in the dashboard. Another important part is the integration of the Android mobilephone, which allows the communication with Raspberry Pi and the dashboard of Node-RED

Índice

Índice de Figuras

Capítulo 1. Introducción

Con el avance de la tecnología y el crecimiento de los núcleos urbanos, aparecen las "*Smart Cities".* La "*Smart Citiy"* o Ciudad Inteligente consiste en la capacidad de respuesta a necesidades básicas del núcleo urbano, ya sean habitantes, empresas o instituciones. Se basan en el desarrollo económico sostenible apoyándose en el uso y la modernización de las nuevas tecnologías y así, resultando en una gestión prudente de los recursos naturales, junto a la participación y el compromiso de los ciudadanos, la mejora de la calidad de vida de sus habitantes y la facilidad de comunicación de estos con su entorno. Para la creación de estas ciudades, es necesario la convivencia de distintos sistemas independientes, que se comunican y se conectan entre sí.

Valencia es un gran ejemplo de comienzo "*Smart City*" con el ejemplo del proyecto desarrollado por *Mobility Strategy,* basado en la implementación de un sensor que pretende analizar la movilidad en la zona urbana de la ciudad. Estos sensores están localizados en farolas o señales de tráfico en las zonas más transitadas por coches. Este dispositivo permite la identificación de un vehículo en distintas zonas, cuando este se acerca al sensor. El vehículo es localizado por los sensores por los que este pasa, colocados por distintos lugares de Valencia.[1]

Estos sistemas independientes pueden tener distintas características, pueden ser a grande escala, como todo el sistema que controla el tráfico de los radares en la ciudad de Valencia, o pueden ser sistemas más pequeños a modo individual, un dispositivo que envíe parámetros con características de la zona residencial donde se encuentra, a otro sistema más complejo donde se procesan.

Uno de esos dispositivos es la RaspBerry Pi, un sistema similar a un ordenador, barato y al alcance de todos que nos permite usarlo de manera de recolector de datos. También usaremos el SenseHat, otro dipositivo que se conecta a la RaspBerry Pi y captura los datos del ambiente de la zona donde está localizado y los manda a nuestra RaspBerry Pi.

Junto a estos dos dispositivos, necesitaremos un lugar de almacenamiento para poder utilizarlos, y si fuera necesario, comunicarlo a otra estación o estaciones de recolección de datos. Como lugar de almacenamiento local hemos elegido una base de datos llamada MongoDB, donde encontraremos los datos obtenidos por SenseHat y RaspBerry Pi. Para lugar de almacenamiento remoto, hemos escogido la plataforma Fiware, un servidor localizado en la UPV.

También utilizaremos nuestro dispositivo Android, con una aplicación que envía datos de geolocalización al servidor de Fiware y comandos a la RaspBerry, para después realizar un análisis.

Interconectamos APIs de diferentes servicios webs, como la obtención de los datos meteorológicos de la ciudad deseada, o la obtención de los tweets actuales, relacionados con un tema.

Para la comunicación entre SenseHat, RaspBerry Pi, MongoDB y Fiware utilizaremos un entorno de programación llamado Node-RED. Esto nos permitirá utilizar programas JavaScript que obtendrán los datos, los procesarán y los transmitirán a la base de datos deseada. Para ello utilizaremos objetos JSON como intercambio de datos entre las bases de datos y los distintos dispositivos.

Para mostrar los datos, utilizamos un dashboard que nos permite la visualización de todos los datos. En una primera pestaña, vemos tres gráficas que contienen, la temperatura, la presión y la humedad de la habitación donde se encuentra la RaspBerry. En esta pestaña también podemos enviar comandos a la RaspBerry para que reaccione de la manera que deseamos, como mostrar

un mensaje o mostrar alertas por los LEDs de SenseHat. En una segunda pestaña se nos muestra un mapa del mundo donde, introduciendo un tema, podemos localizar donde se está twiteando sobre ese tema. Gracias a la aplicación Android también podemos localizar nuestro dispositivo en el mapa. En esta pestaña también consultamos el tiempo atmosférico en la ciudad deseada y se nos mostrará en el mapa.

1.1 Raspberry Pi

1.1.1 Introducción a Raspberry Pi

Raspberry Pi es una única placa con función de ordenador, del tamaño de una tarjeta de crédito, desarrollada en Reino Unido por la fundación Raspberry. En este proyecto, utilizaremos la Raspberry Pi 3, modelo B. La Raspberry Pi 3, modelo B es la tercera generación de dispositivos Raspberry Pi lanzada al mercado en febrero de 2016. Este modelo tiene las siguientes características. [2]

- Quad Core 1,2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 Wireless LAN y Bluetooth Low Energy (BLE) en la placa
- 40-pin extended GPIO
- 4 puertos USB 2.0
- 4 Pole stereo de salida y puerto de video de composite
- HDMI
- Puerto de cámara CSI para conectar una cámara Raspberry Pi
- Puerto de monitor DSI para conectar una pantalla táctil Raspberry Pi
- Puerto para una Micro SD para cargar el sistema operativo y guardar datos

Puerto Micro USB para conectarlo a una corriente, hasta 2,5A. [2] (Figura 1-1)

Figura 1-1 Raspberry Pi 3 Modelo B

Para la instalación del sistema operativo, consulte el Anexo 1 (A-1)

1.1.2 Empezando con Raspberry Pi

Cuando ya tengamos instalado el sistema operativo en nuestra Raspberry Pi, tenemos que insertar la tarjeta micro SD en la parte inferior de nuestra Raspberry Pi. También será necesario un monitor, fuente de alimentación, teclado y ratón para poder trabajar con nuestro dispositivo. Debemos conectar la fuente de alimentación al puerto micro USB, después, conectar a través de un cable HDMI el monitor al puerto HDMI de nuestra Raspberry Pi y a continuación conectar nuestro teclado y ratón a través del puerto USB de la misma.

Cuando todo esté conectado correctamente, podemos iniciar la Raspberry Pi y comenzará el proceso de configuración. Debemos aplicar la configuración que nosotros deseemos. [3]

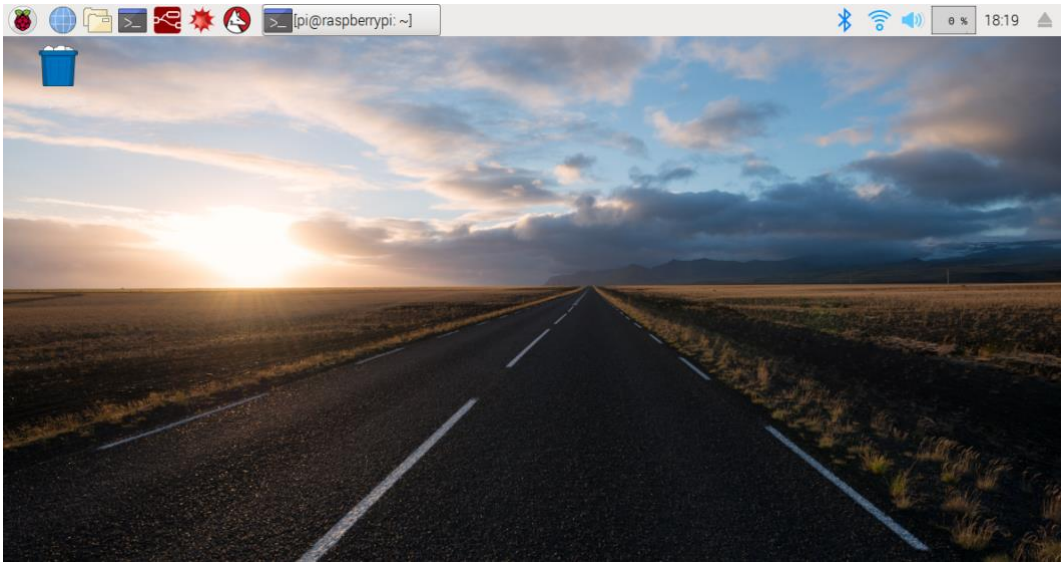

Figura 1-2 Escritorio Raspberry Pi

En la figura 1-2 vemos el escritorio al haber configurado nuestra Raspberry Pi. En la barra superior, parte derecha vemos un icono que nos representa los dispositivos extraíbles que tenemos conectados, a continuación encontramos la hora actual, el siguiente icono nos indica un monitor con el porcentaje de ocupación de CPU, después vemos el icono que representa el volumen (Si tuviéramos un dispositivo emisor de sonido conectado), el siguiente es el icono que representa la red wifi y finalmente el bluetooth. [3]

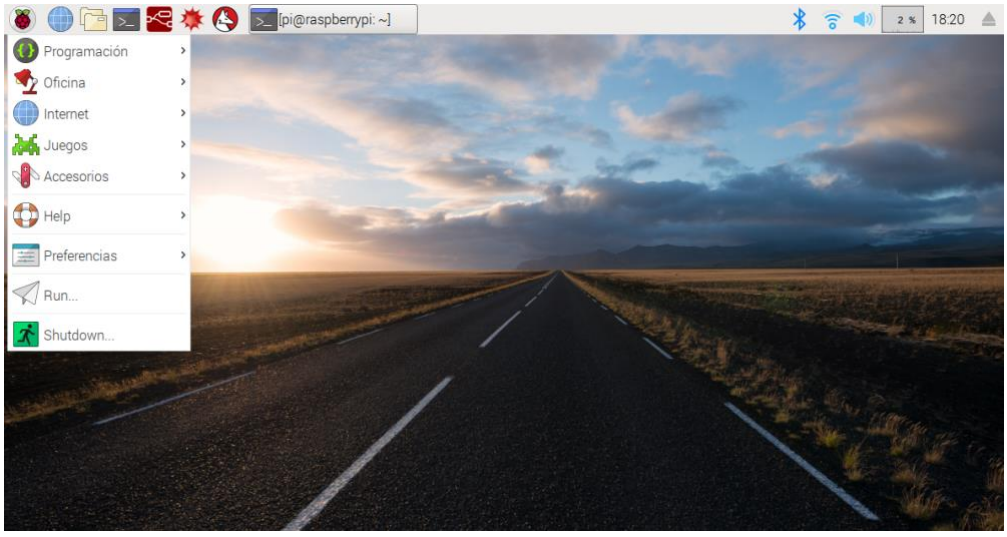

Figura 1-3 Menú Raspberry Pi

En la figura 1-3 vemos la parte izquierda de la barra superior. Primero nos encontramos un menú desplegable donde vemos varios apartados. Estos apartados se corresponden al tipo de programa que vemos dentro. También vemos un apartado de ayuda, donde tenemos varias guías que nos ayudarán a distintos propósitos como, comenzar a trabajar con nuestra Raspberry Pi. El apartado de preferencias nos permite configurar nuestra Raspberry Pi. Con el apartado de Run podemos ejecutar comandos, por ejemplo para iniciar programas. Y finalmente el apartado de Shutdown, sirve para apagar, reiniciar o cambiar de usuario de nuestra Raspberry Pi. A continuación del menú de despleglables, podemos encontrar varias aplicaciones, que, podremos colocar nosotros ahí para un acceso más rápido. [3]

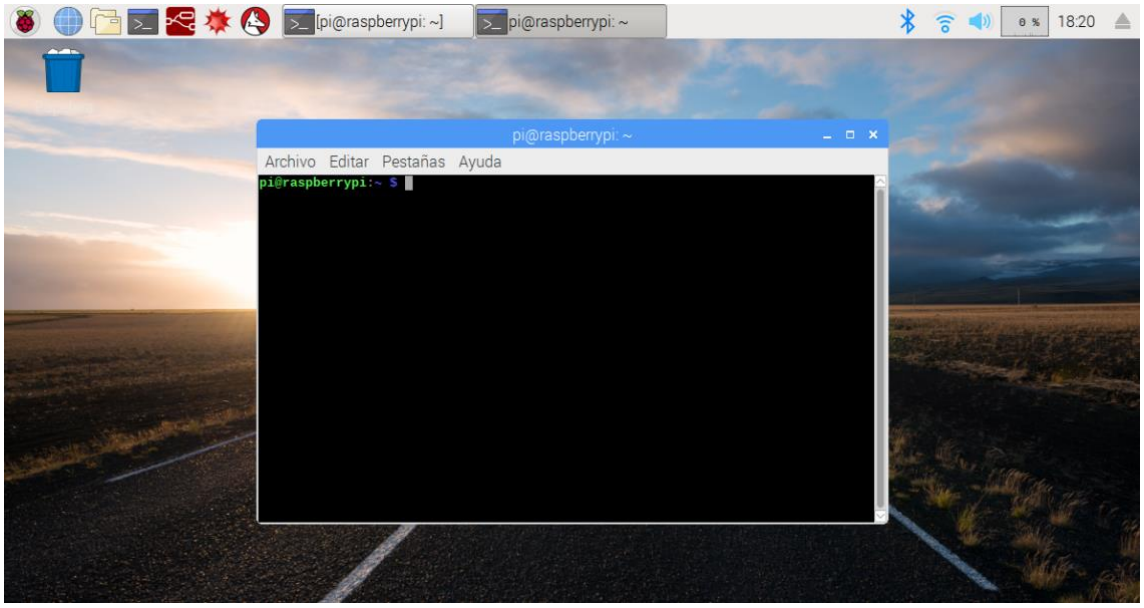

Figura 1-4 Terminal de Raspberry Pi.

En la Figura 1-4 vemos el terminal. El terminal nos sirve para, prácticamente, realizar todo en nuestra Raspberry Pi. Con el instalaremos las aplicaciones y herramientas necesarias, además de ejecutar comandos que nos ayudarán también con la configuración de la Raspberry Pi. Para abrirlo solo necesitamos hacer click en el icono de terminal que nos aparece en la barra superior. [3]

1.2 Node-RED

1.2.1 Introducción a Node-RED

Node-RED es una herramienta de programación para IoT, que junta en un único sitio, dispositivos, APIs y servicios online, utilizando un editor basado en un navegador, que facilita y hace más visual la conexión entre funciones, usando una amplia selección de nodos provenientes de una paleta introducida por Node-RED, que pueden ser introducidos en nuestro esquema con un simple click.

Al ser un editor basado en un navegador, hace la tarea de conectar distintas funciones mucho más sencilla, simplemente arrastras la función (o nodo) deseada al esquema principal y lo conectas con otros nodos.

También existe la posibilidad de crear tu propia función JavaScript e incluirla a la librería de funciones, plantillas o nodos para reutilizarlos.

El ligero tiempo de ejecución está basado en Node.js, aprovechando al máximo sus eventos sin bloqueo. Gracias a esto, podemos ejecutarlo en los bordes de la red con dispositivos de bajo coste, como Raspberry Pi o como la nube. Con más de 225.000 nodos en el repositorio, es sencillo añadir más nodos a la paleta para ampliar nuevas funcionalidades.

Los esquemas creados en Node-RED se almacenan usando el lenguaje JSON, un lenguaje muy sencillo que se puede importar y exportar fácilmente, para compartirlo con otros. Existe una biblioteca online que te permite compartir tus esquemas con el mundo.

La gran ventaja de que Node-RED esté implementado en Node.js, lo que equivale a que tiene sus propios modelos sin bloqueo, permite que se pueda ejecutar al borde de la red de manera local, en otro dispositivo (como Raspberry Pi, Arduino, Android…) o en una nube. [4]

1.2.2 Empezando en Node-RED

Para comenzar a usar Node-RED, debemos ejecutar en nuestro dispositivo el comando `node-red`. Con este comando, lo que hacemos es encender un servidor local en el puerto 1880, accesible desde http://localhost:1880.

Cuando entramos en esta dirección, con el servidor encendido, nos encontramos con el espacio de trabajo de Node-RED. [4]

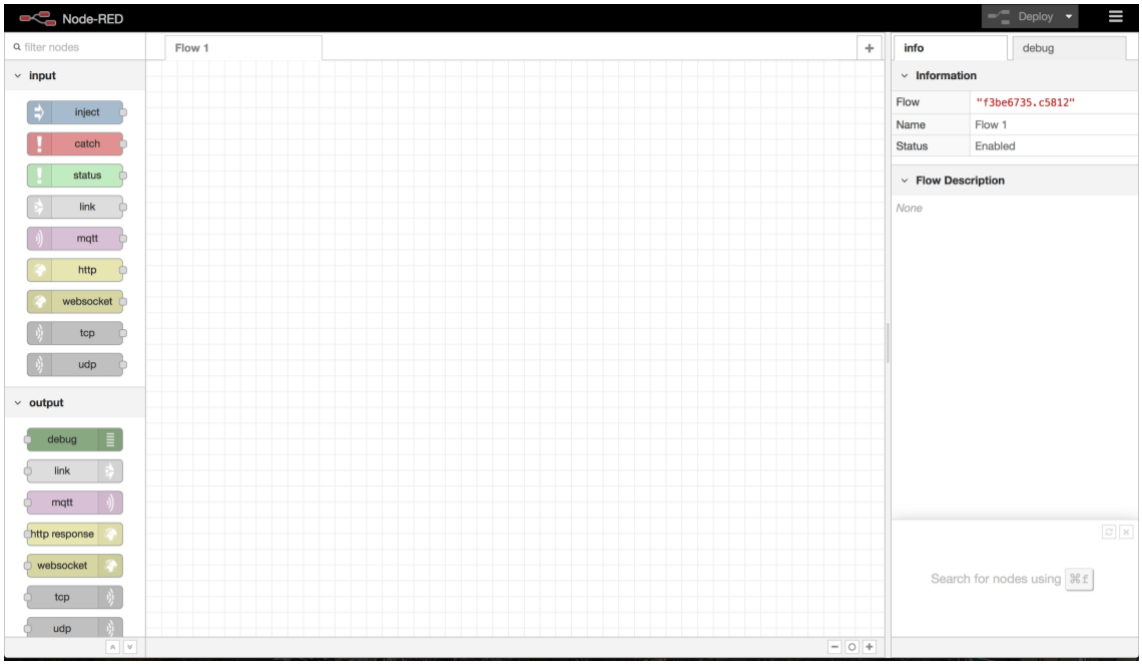

Figura 1-5 Espacio de trabajo Node-RED

Este es el espacio de trabajo de Node-RED, podemos ver que en el centro tenemos una cuadrícula donde podemos arrastrar y conectar nuestros nodos. A la izquierda nos encontramos con los nodos que Node-RED nos ofrece.

Para introducir un nodo al espacio de trabajo, simplemente lo seleccionamos y lo arrastramos al centro del espacio de trabajo. Para conectar dos nodos, simplemente unimos los puntos de conexión que estos tienen en sus extremos. La conexión entre nodos, funciona con el intercambio de mensajes entre ellos, estos mensajes, son objetos JavaScript. Muchos nodos, te permiten controlar o modificar el comportamiento de un nodo, añadiendo a los mensajes que se intercambian, distintas propiedades.

Puedes añadir un nodo entre dos nodos para hacer la función de interconexión entre ellos.

Existe la opción de juntar muchos nodos en uno solo para organizar el espacio de trabajo, esto se llama subflow, y podemos encontrar la opción en el menú > subflow > select to subflow, para juntar los nodos seleccionados en uno.

Al crear un subflow, en la paleta de nodos se crea un apartado donde aparece el subflow creado, y al hacer click sobre él, podemos ver las funciones que contiene, y las salidas y entradas. Podemos modificarlo como queramos y reutilizarlo para otros flows. También puedes añadir un nombre y una documentación a los subflows. [4]

1.2.3 Nodos

A continuación, veremos los nodos principales ya instalados en el entorno de trabajo de Node-RED. Posteriormente instalaremos otros tipos de nodos que nos harán de conexión entre dispositivos y base de datos. [4]

1.2.3.1 Nodos principales

- Nodo de depuración:

Los nodos de depuración pueden ser activados y desactivados, pulsando el botón que aparece junto a ellos. Con estos, al cambiar a la pestaña de "debug", nos permite ver los mensajes intercambiados en el flow (*Figura 1-6*).

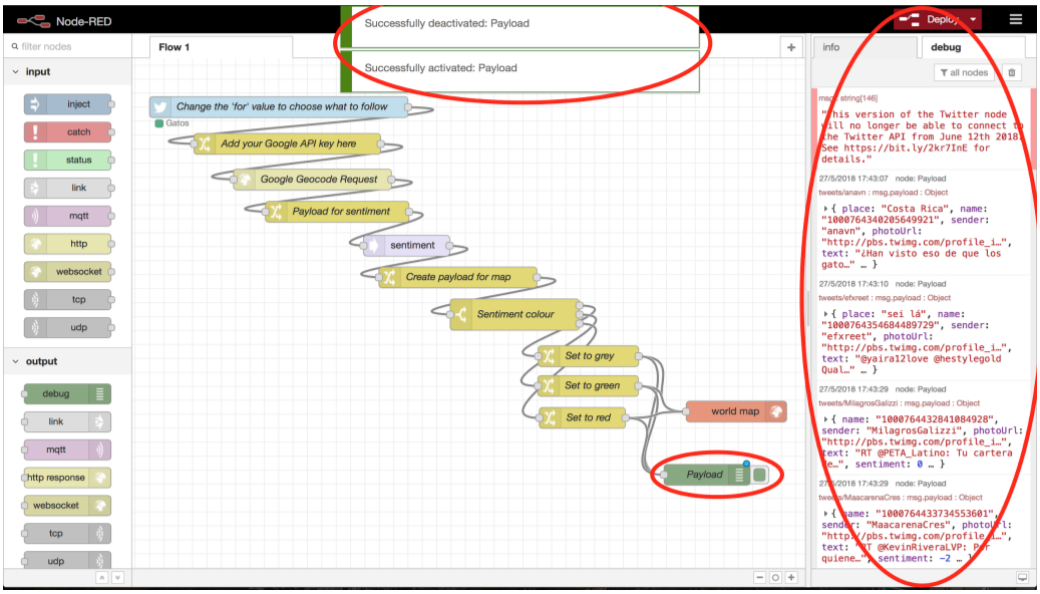

Figura 1-6 Nodo depuración

- Nodo Entrada:

Nos permite introducir manualmente mensajes al flow. Se puede configurar para mandar mensajes de manera regular. (*Figura 1-7)*

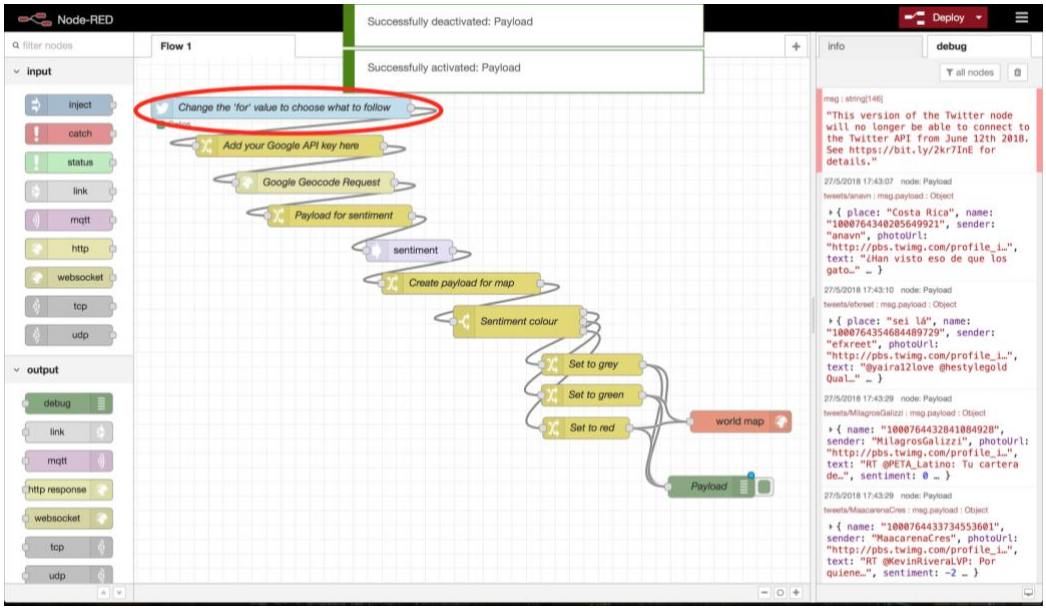

Figura 1-7 Nodo entrada

- Nodo IoT:

Manda o recibe información de los dispositivos IoT. Utiliza un ID de dispositivo.

- Nodo función:

Permite introducir una función de JavaScript para condicionar un comportamiento deseado.

- Nodo Interruptor:

Dirige los mensajes a una u otra rama, dependiendo de las condiciones introducidas en éste. (*Figura 1-8*)

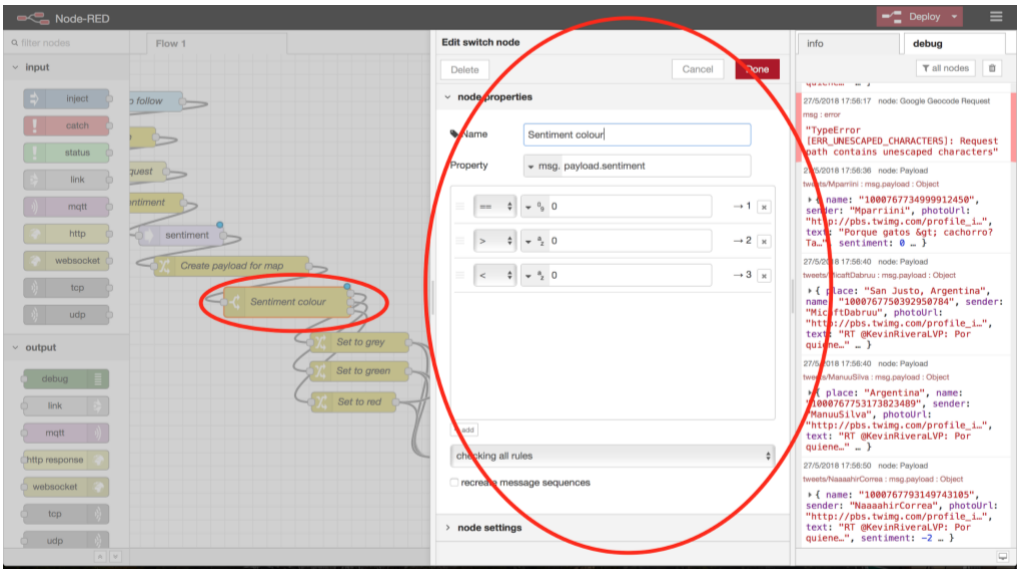

Figura 1-8 Nodo interruptor

- Nodo Plantilla:

Permite modificar el mensaje de un nodo usando una plantilla.

Para eliminar un nodo o un flow, simplemente se selecciona, haciendo click en un espacio en blanco y arrastrando para cubrir los nodos deseados, y se pulsa el botón suprimir.

Se necesita hacer click en Deploy para parar su funcionamiento. [4]

1.2.3.2 Nodos SenseHat

Para nuestro dispositivo SenseHat se precisa de la instalación de unos nodos específicos, que realicen la comunicación entre Raspberry Pi y los sensores de SenseHat. A continuación, mostramos los nodos utilizados. (Figura 1-9) [4]

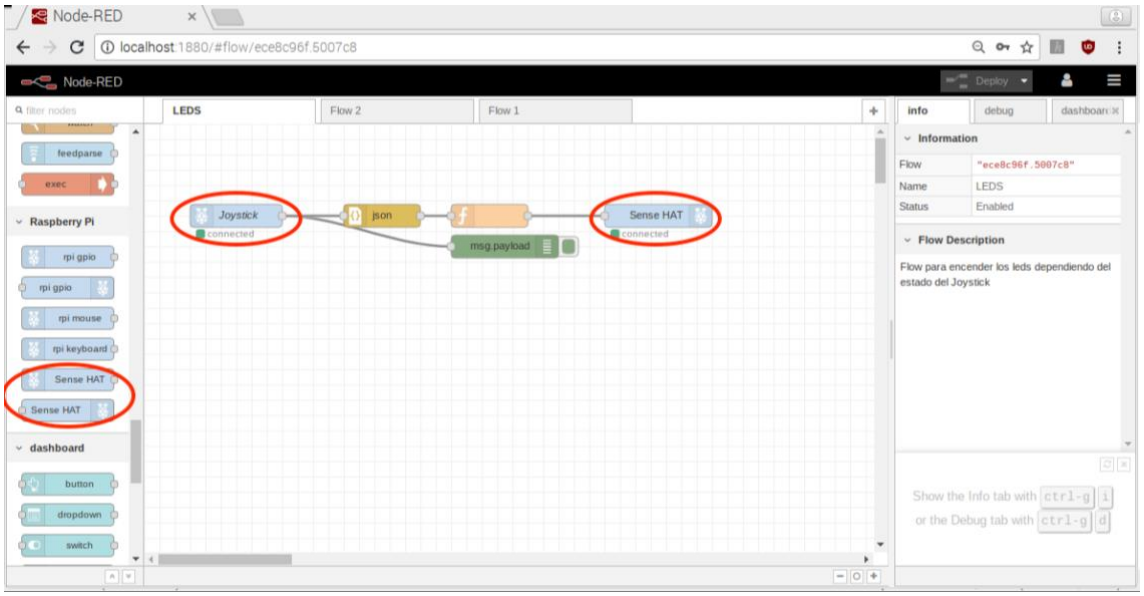

Figura 1-2 Nodos SenseHat

- Nodo entrada SenseHat:

Este nodo obtiene los parámetros del dispositivo SenseHat deseados, tales como el joystick, parámetros del ambiente y parámetros de movimiento. Es configurable para seleccionar solo los datos que necesitamos que se envíen a Node-RED. (Figura 1-10)

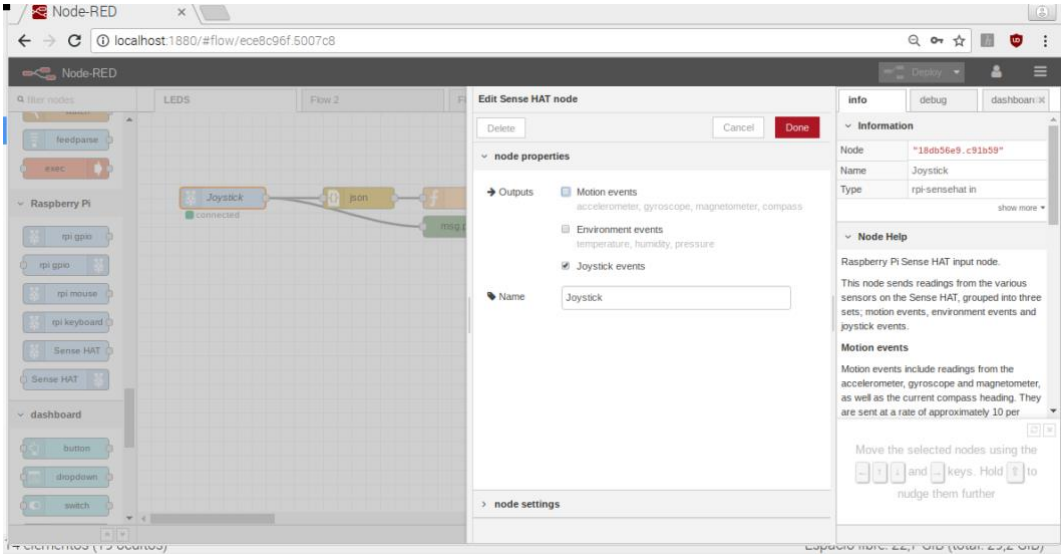

Figura 1-10 Configuración nodo entrada SenseHat

- Nodo salida SenseHat:

Este nodo envía al dispositivo SenseHat comandos para que este actúe de una manera u otra. Principalmente se utiliza para la conexión con el panel de 8x8 LEDs. [4]

1.2.3.3 Nodos MongoDB

Los datos deseados serán almacenados en la base de datos MongoDB alojada en la propia Raspberry Pi, por lo tanto necesitaremos instalar los nodos necesarios para la comunicación con la base de datos. (Figura 1-11) [4]

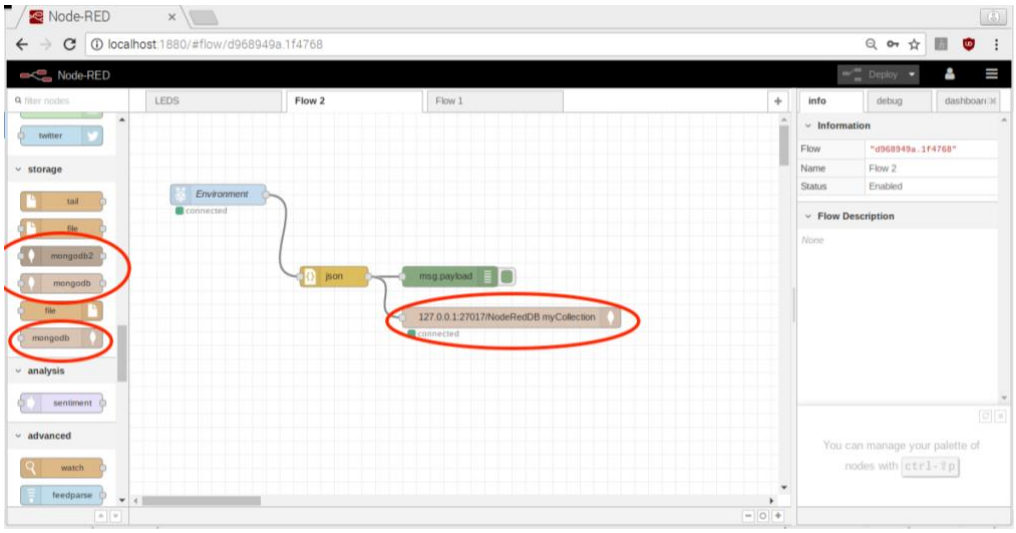

Figura 1-11 Nodos MongoDB

Ambos nodos necesitan ser configurados con los datos de identificación de la base de datos.

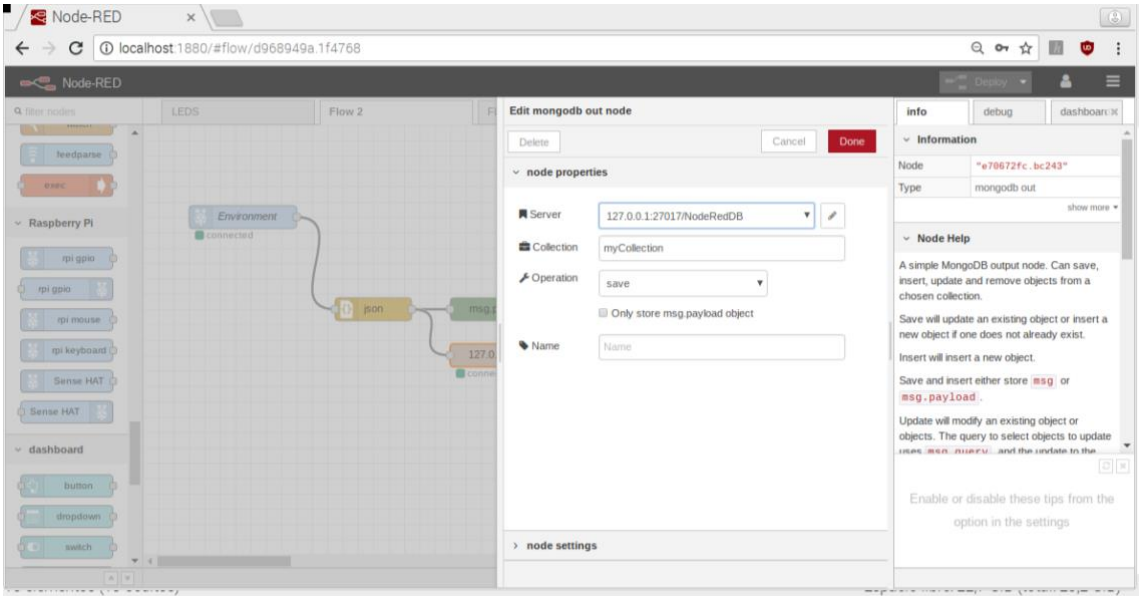

Figura 1-12 Configuración nodos MongoDB

- Nodo entrada MongoDB:

Se utiliza para obtener parámetros de la base de datos.

- Nodo salida MongoDB:

Sirve para enviar y guardar los parámetros a la base de datos. [4]

Para la instalación de Node-RED consulte el Anexo 2 (A-2)

1.3 SenseHat

1.3.1 Introducción a SenseHat

La placa SenseHat de Raspberry Pi es una placa sin cables de sensores avanzada, que integrada con Raspberry Pi y sus diferentes sensores como, Temperatura, humedad, sensor de presión, giroscopio, acelerómetro y magnetómetro. En los últimos años fue usada en Astro pi Space para monitorizar las condiciones meteorológicas. (Figura 1-13) [5]

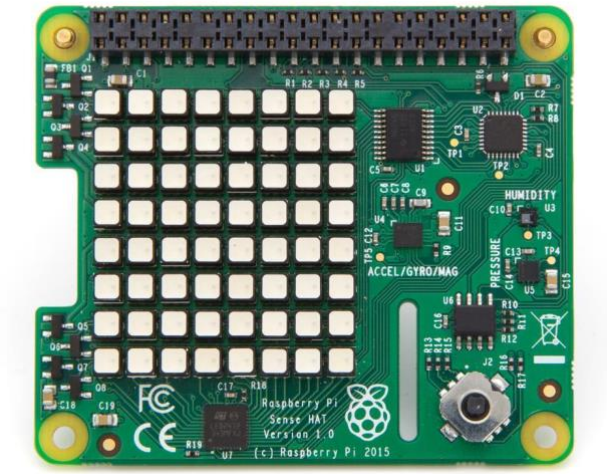

Figura 1-13 Placa SenseHat

Solo soporta las placas Raspberry Pi, sus 40 GPIO pins están confrontados con la parte superior de la Raspberry Pi. Tiene un coste bajo, un tamaño y un peso muy bajo. Sirve para realizar varios experimentos incluso videojuegos. También tiene características adicionales, como un Joystick de cinco posiciones y una pantalla de 8x8 RGB (Rojo, verde y azul) LEDs (Diodos emisores de luz) que emiten luz a la intensidad y el color que deseemos. La pantalla de LEDs tiene los componentes rojo, verde y azul, por lo tanto, somos capaces de crear cualquier color o cualquier tonalidad usando una matriz de valores entre 0 y 255. [5]

Las características específicas de SenseHat son las siguientes:

- Giroscopio – Sensor de velocidad angular: ±245/500/2000dps
- Acelerómetro – Sensor de aceleración lineal: ±2/4/8/16g
- Magnetómetro – Sensor magnético: ±4/8/12/16 gauss
- Garómetro: 260 – 1260 hPa rango absoluto (la precisión depende de la temperatura y la presión ±0,1 hPa bajo condiciones normales)
- Temperatura del sensor (La temperatura es precisa ±2ºC en el rango de 0-6ºC)
- Sensor de humedad relativa (Precision de un ±4,5% en el rango de 20,80%rH, y una precisión de ±0,5ºC en el rango de 15-40ºC) [5]

Para instalación y montaje de SenseHat consulte los Anexos 3 y 4 (A-3, A-4)

1.4 MongoDB

1.4.1 Introducción a MongoDB

MongoDB es una base de datos, que no usa SQL, gratuita y escalable. Usa el almacenamiento de documentos, al contrario que una estructura de tablas de dos dimensiones. Fue desarrollada por la empresa 10gen como un componente de una plataforma que utilizarían como producto. Fue adoptada como software de backend por la mayoría de servicios y compañías, incluyendo eBay, SourceForge, FourSquare y New York Times. [8]

Las diferencias entre MongoDB y RDBMS normales, se muestran en la figura a continuación (Figura 1-14). La figura muestra como las Tablas en las RDBMS son sustituidas por collecciones en MongoDB, las filas se sustituyen por documentos JSON y la unión a estas, son sustituidas por links y enlaces. [8]

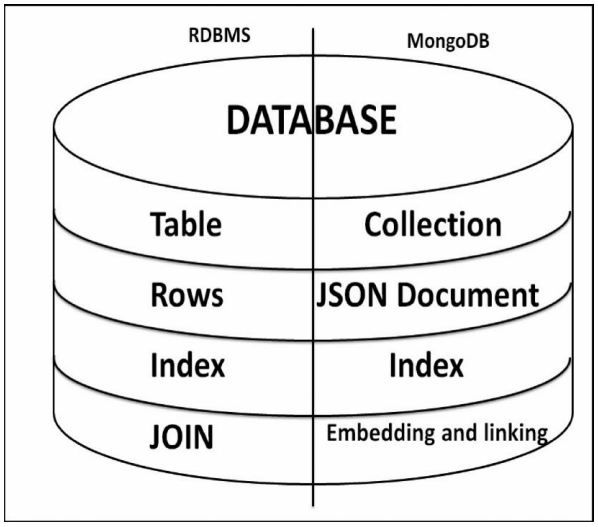

Figura 1-14 MongoDB vs RDBMS [8]

1.4.2 Ventajas del uso de MongoDB

MongoDB ha atraído la atención de muchos desarrolladores dadas sus ventajas sobre las bases de datos tradicionales. MongoDB ha evolucionado como un nuevo tipo de base de datos, que puede ser usada por desarrolladores. Las razones por las que se usa MongoDB son las siguientes.

Una de las ventajas más importantes sobre MongoDB, es el poco esfuerzo necesario para crear una. MongoDB utiliza comandos por terminal para interactuar con la base de datos, por lo que los desarrolladores no tienen que gastar mucho tiempo en aprender como escribir código para la base de datos. En lugar de escribir "SELECT * FROM…" se utilizan comandos como los mostrados a continuación:[8]

```
Mongo m = new Mongo("DBServer",27017); //Servidor y Puerto
DB db=m.getDB("nombredb"); //Apertura de la base de datos
```

El ejemplo a continuación muestra la conexión a la base de datos:

```
DBCollection coll = db.getCollection("NombreCollecion");
BasicDBObject doc = new BasicDBObject();
Doc.put("atributo", "valor");
Coll.insert(doc);
```

Otra ventaja de MongoDB es que tiene una gran escalabilidad. La escalabilidad se consigue con "Sharding". Sharding es un método que se utiliza para distribuir mucha información a través de distintos dispositivos. Se utiliza cuando grandes grupos de datos se requieren constantemente. Existen dos formas de utilización de este método.

Escalabilidad Vertical, su función es la de aumentar la capacidad de un servidor, aumentando su CPU o su memoria RAM.

Escalabilidad Horizontal, divide los datos del sistema entre distintos servidores para aumentar la capacidad requerida. [9]

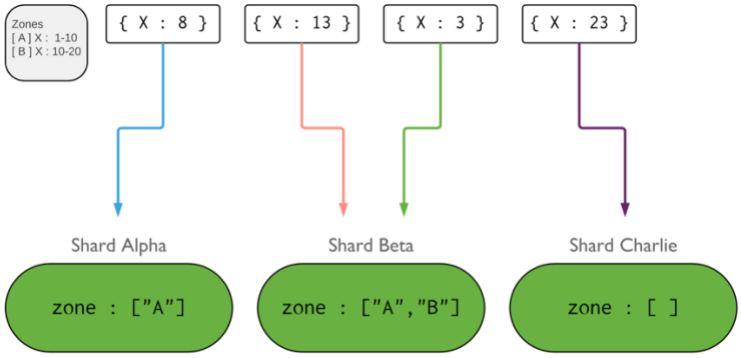

Figura 1-15 Sharding en MongoDB [9]

También, MongoDB es una base de datos orientada principalmente a documentos, en la que una colección, almacena distintos documentos con distintos campos. Los documentos también pueden tener distinta estructura. Además, el usuario puede añadir, eliminar o modificar los documentos cuando quiera. Esto hace que sea una base de datos que no tenga esquemas.

MongoDB también soporta el desarrollo para APIs, la estructura de almacenamiento por documentos, hace que los recursos obtenidos por cualquier API por un GET o un POST, sean tratados de manera más sencilla. [8]

1.5 FIWARE

1.5.1 Introducción a FIWARE

Cloud computing, Big Data y dispositivos IoT son la clave para el establecimiento del internet del futuro. En este contexto, la Comisión Europea (CE) visionó la posibilidad de aumentar la adopción de muchos sistemas, aumentando la simplicidad de nuevos sistemas. Además, la CE, vió la necesidad de establecer un compromiso entre los campos industriales y académicos. Por ello, la CE estableció el programa *Future Internet Private Public Partnership* (FI-PPP), el cual sirvió de desarrollo para una, nueva y compleja, plataforma en la nube Europea, llamada FIWARE. El objetivo de FIWARE es crear una plataforma estándar, abierta, sostenible y global. La Arquitectura de FIWARE pretende establecer unas funciones en su plataforma, que apoyen el desarrollo de APIs, llamadas *Generic Enablers* (GEs). Estas son interfaces, que interconectan dispositivos y redes, con ecosistemas de aplicaciones, servidores, servicios IoT, seguridad en la red… FIWARE aporta especificaciones de GE (públicas) y sus implementaciones (GEi). [10]

La ventaja del uso de Fiware es que la arquitectura del software depende de un código abierto de uso general, apoyado por una comunidad por todo el mundo. De hecho, existen GEs de necesidades muy específicas, y los desarrolladores pueden diseñar una gran variedad de sistemas específicos, simplemente, integrando y usando las APIs. Además, FIWARE cumple de manera muy correcta los requerimientos de un rápido desarrollo, porque incita al aumento de la mejora del futuro de los sistemas y aplicaciones de Internet. Para conocer los requerimientos de los clientes, proveedores de datos, emprendedores, desarrolladores y proveedores de tecnología FIWARE, el FI-PPP ha llevado al laboratorio de desarrollo de FIWARE (FI-LAB) los sistemas y aplicaciones en los que se puede desplegar el servicio de FIWARE. [10]

1.5.2 Comunicacióin con FIWARE

Orion es una implementación en C++ de la REST API NGSIv2 que forma parte del desarrollo de la plataforma FIWARE.

El contexto de Orion Broker te permite controlar el flujo de la información, incluyendo actualización, peticiones, registros y suscripciones. Es una implementación del servidor NGSIv2, para controlar la información y su disponibilidad. Usando el contexto Orion Broker, tienes la posibilidad de crear elementos de contexto y manejarlos a través de actualizaciones y peticiones. Además, puedes suscribirte a la información de contexto, para que, cuando se realice un cambio, recibas una notificación. Describiremos el tipo de escenario y características de Orion Broker que hemos usado para este proyecto. [11]

Primero vamos a describir la estructura de los objetos que se encuentran en el servidor de FIWARE. Los elementos principales con los que trabaja FIWARE se denominan entidades. Estas entidades están compuestas por distintos datos. Los datos de las entidades son los que iremos actualizando, creando, modificando o borrando.

La comunicación con el servidor FIWARE se realizará a través de peticiones HTTP hacia la url del servidor, añadiéndole las cabeceras necesarias. Se necesitará tener un servicio y un path para poder trabajar. El formato de las cabeceras es el siguiente:

```
msg.headers['Accept'] = 'application/json';
msg.headers['FIWARE-Service'] = 'service';
msg.headers['FIWARE-ServicePath'] = '/path';
```

Si hacemos una petición de actualización o de creación, se añadirá una cabecera más que será del formato `msg.headers['Content-Type']='application/json';` y dependiendo de los datos que mandemos, habrá que modificar el campo a otro valor.

Para empezar a guardar datos, tenemos que crear una entidad. Para crear la entidad es necesario realizar una petición HTTP del tipo POST. Esta petición, se debe realizar a la dirección del servidor FIWARE y añadiéndole al final `/v2/entities`. El formato de la petición, debe ser un texto en formato JSON, siendo el "id" y el "type" del primer objeto, el nombre de la entidad. Dentro de este objeto, se incluirán los datos con el formato que se muestra a continuación:

```
{
  "id": "Room1",
  "type": "Room",
    "temperature": {
      "value": 23,
      "type": "Float"
    },
    "pressure": {
      "value": 720,
      "type": "Integer"
    }
}
```

En este ejemplo, vemos que, la entidad es `Room1` y que tiene como datos, `temperature` con valor `23` y tipo `Float`, y la `pressure` que tiene valor `720` y tipo `Integer`.

De esta manera hemos creado la entidad. Para actualizar la entidad, tenemos que realizar una petición HTTP del tipo PUT, añadiéndole a la url del servidor al final `/v2/entities/{entidad}/attrs`, poniendo en `{entidad}` el nombre de la entidad. El formato de la petición será el siguiente [11]:

```
{
  "temperature": {
    "value": 26.5,
    "type": "Float"
  },
  "pressure": {
    "value": 763,
    "type": "Float"
  }
}
```

Como vemos, cambia con el anterior, ya que ahora no tenemos que añadir ni el "id" ni el "type" de la entidad.

Para leer los datos que tenemos guardados en el servidor, la petición a realizar es una petición HTTP de tipo GET, dependiendo de lo que queramos habrá que añadir a la url del servidor una parte u otra. Si lo que queremos es obtener todas las entidades con sus respectivos datos, se le añadirá `/v2/entities` al final. Si lo que queremos es obtener los datos de una entidad en concreto, se le añadirá `/v2/entities/{entidad}` al final, siendo `{entidad}` el nombre de la entidad. La respuesta de esta petición GET será un texto con formato JSON. [11]

1.6 Android Studio

1.6.1 Introducción a Android Studio

Android Studio es un IDE (Integrated Development Environment) para el desarrollo de aplicaciones Android, basado en IntelliJ IDEA. Es un grn editor de códigos. Gracias a sus herramientas de IntelliJ para desarrolladores, ofrece muchas funciones que aumentan la productividad entre compilación de aplicaciones Android. [12]

-Sistema basado en Gradle muy flexible

-Emulador rápido

-Entorno unificado

-Compliación instantánea

-Plantillas de código y GitHub que te facilitan el trabajo.

-Muchas herramientas y frameworks.

-Herramientas para detectar problemas de usabilidad, rendimiento, compatibilidad…

-Es compatible con NDK y C++

-Tiene soporte para Google Cloud Platform. [12]

1.6.2 Interfaz de usuario

1. La barra de herramientas, puedes realizar una gran variedad de acciones, como la ejecución de tu aplicación.
2. La barra de navegación, te ayuda a explorar el proyecto y abrir ficheros para poder editarlos. Tiene una vista más simple de la estructura del proyecto.
3. Ventana de editor, area donde creas y modificas tu código. Dependiendo del archivo que estes modificando, el editor es distinto.
4. Barra de ventana de herramientas, se extiende por la parte exterior de la vetnana del IDE y tiene botones para modificar la disposición de las ventanas.
5. Ventanas de herramientas, con ellas puedes acceder a ciertas tareas, como administración de proyectos, controles de versiones… se pueden contraer y expandir.
6. Barra de estado, muestra el estado del proyecto y del IDE en sí, como mensajes y advertencias.

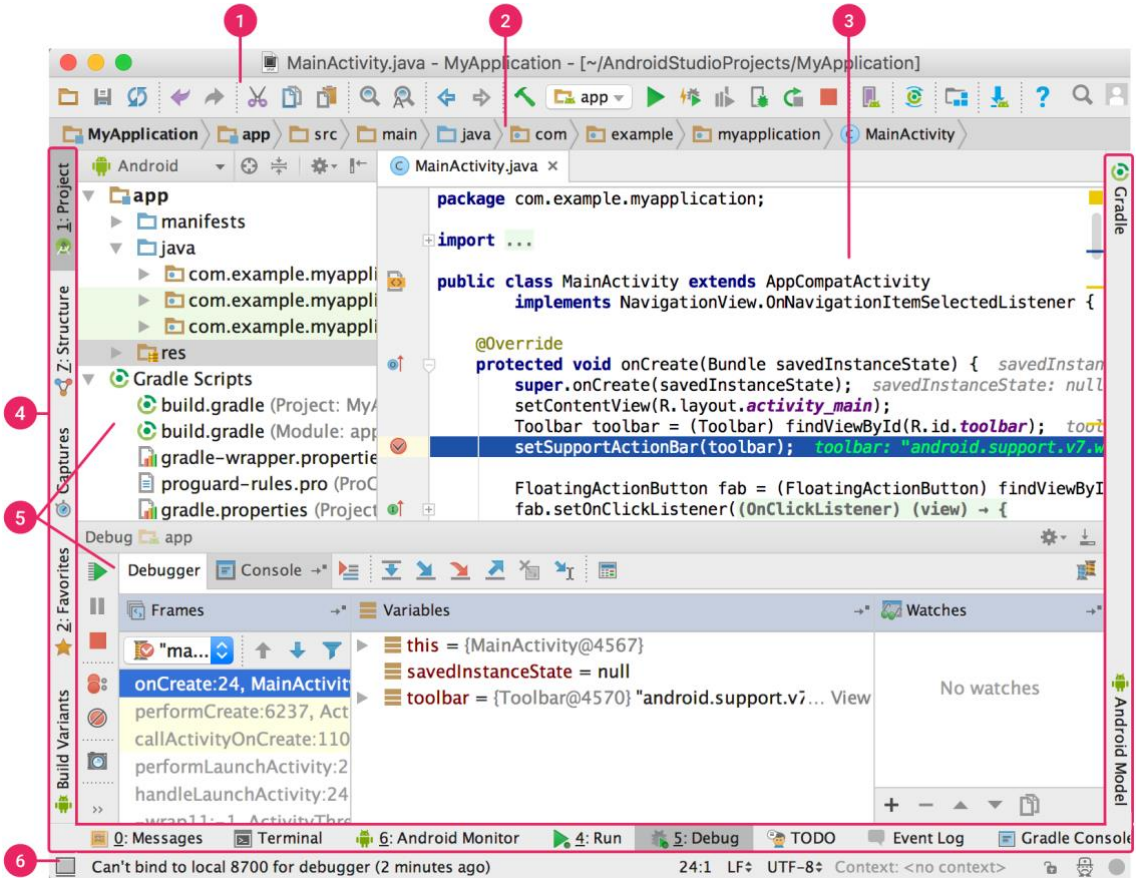

Figura 1-16 Ventana de Android Studio [12]

Se puede organizar la ventana principal para tener más espacio libre, cambiando la disposición de todas las barras o ventanas. Existen combinaciones de teclas para acceder a las funciones del IDE.

Puedes realizar búsquedas en el código, acciones, bases de datos… presionando la tecla SHIFT dos veces o haciendo click en la lupa que se encuentra en la esquina superior derecha. Te resultará muy útil a la hora de localizar una acción específica del IDE. [12]

Capítulo 2. Objetivos del TFG

En este apartado vamos a mencionar los objetivos que nos hemos marcado a la hora de realizar el proyecto.

- Investigar en el avance de las Smart Cities
- Integrar distintos servicios IoT en un mismo Dashboard
- Demostrar la facilidad de trabajar con dispositivos como Raspberry Pi o SenseHat
    - Costes baratos
    - Sencillez de uso
    - Manejables al tener un tamaño pequeño
- Aprender a utilizar Node-RED
- Obtener datos del ambiente
- Transferir esos datos ambientales a una base de datos para poder ser usados por otros.
- Obtención de datos de otras APIs e integrarlos con servicios IoT
- Desarrollo de una aplicación Android que interactue con servicios IoT
- Dependiendo de los datos obtenidos, tener la posibilidad de crear alertas

Capítulo 3. Metodología

3.1 Gestión del proyecto

Este proyecto comenzó el día 7 de mayo de 2018, al ponerme en contacto con mi tutor Juan Carlos Guerri Cebollada. Juntos decidimos que podríamos realizar un programa, que usando varios dispositivos como Raspberry Pi y SenseHat, y la herramienta de programación Node-RED, nos realizara la función de estación de recolección de datos en una Smart City. Para ello nos pusimos en contacto con el grupo de investigación iTeam y realizamos una reunión para hablar sobre este proyecto.

A continuación, Juan Carlos me presentó a Pau Arce, el cual es mi cotutor y los tres, hemos conseguido llegar al proyecto que presento. Para ello primero pensamos en realizar un dispositivo que funcionara en casa, y que recogiera datos sobre la habitación en la que se encontraba, usando los distintos sensores.

Como base de datos llegamos al acuerdo de tener una base de datos en la UPV, usando el servicio de FIWARE, y que fue proporcionada por el iTeam. También, como copia de base de datos añadimos de manera local la base de datos MongoDB.

Después, decidimos que debíamos interconectar más información sobre el entorno, por ello decidimos añadir APIs que nos ayudaran a obtener diferentes datos.

También vimos que sería una buena idea integrar una herramienta que localizara nuestro dispositivo Android y, además, este pudiera enviar comandos a la RaspBerry.

Queríamos integrar todo esto en un dashboard, para poder acceder a todos los datos desde una misma página.

3.2 Distribución de tareas

Durante los primeros 4 meses de proyecto se realizaron 2 o 3 reuniones para hablar sobre el proyecto, y durante los últimos 3 meses, se realizaban reuniones todos los miércoles (que fueran posibles) para comentar avances, errores y dudas.

Para comenzar con el proyecto, mientras esperábamos que nos llegaran los materiales necesarios, comenzamos a aprender a utilizar la herramienta de programación Node-RED. Para ello realizamos el curso "*A developer's guide to the Internet of Things (IoT)"* ofrecido por IBM [6]

Cuando teníamos los materiales, realizamos la instalación del sistema operativo para trabajar con la Raspberry Pi. Tras ello, instalamos en nuestra Raspberry Pi, la herramienta Node-RED y posteriormente instalamos la placa SenseHat.

Al tener todo instalado, procedimos a instalar los nodos necesarios para SenseHat y MongoDB y comenzamos a aprender cómo realizar la comunicación con los sensores de la placa SenseHat.

Cuando se consiguió obtener datos de SenseHat, se procedió a almacenarlos en las distintas bases de datos. Primero en la base de datos de MongoDB, y con peticiones HTTP a la base de datos situada en la UPV, Fiware.

Al tener todos los datos en las dos bases de datos, aprendimos a cómo obtenerlos y dejarlos en el mismo formato, por si una de las dos bases de datos fallaba, tener una copia funcional en la otra.

A continuación, aprendimos a realizar dashboards con Node-RED y a mostrar los datos obtenidos en gráficas de este dashboard.

Cuando los datos ya se mostraban como se deseaba, se procedió a buscar más información sobre distintas APIs que se podrían integrar en el dashboard de Node-RED.

Al encontrar estas APIs, se integraron en Node-RED y se interconectaron entre ellas para tener una mejor experiencia de usuario.

3.3 Diagrama temporal

A continuación se muestra el diagrama temporal que se ha seguido para la realización del proyecto:

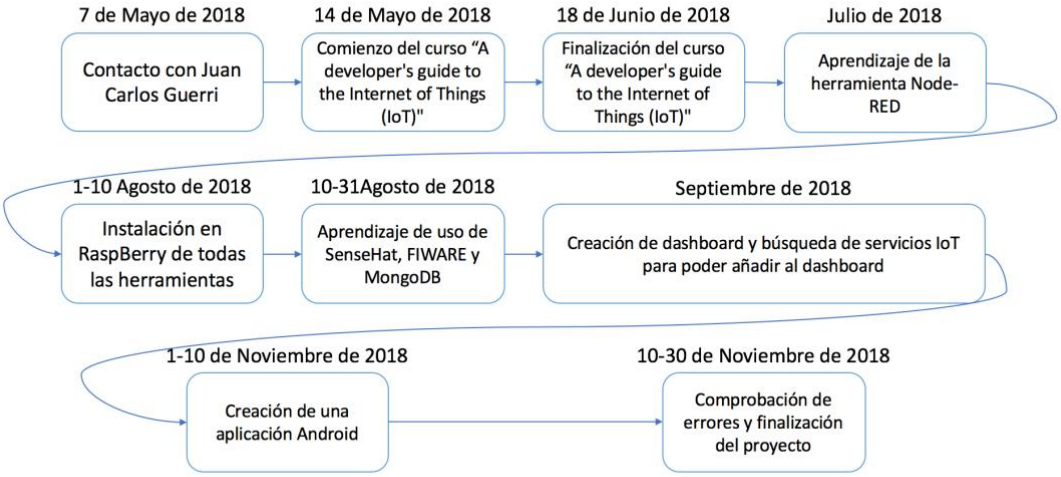

Figura 3-1. Diagrama temporal

Capítulo 4. Desarrollo y resultados.

4.1 Introducción del Proyecto

El objetivo de este proyecto es concentrar en un único lugar, muchos servicios IoT y poder interactuar con ellos de manera sencilla. Para que la experiencia de usuario sea cómoda, utilizaremos el dashboard proporcionado por Node-RED.

Para ello, utilizaremos una RaspberryPi que nos servirá de base principal donde alojaremos la herramienta de Node-RED. Esta herramienta servirá para obtener los datos, modificarlos y tratarlos, y transmitirlos donde deseemos.

Primero, debemos definir donde vamos a almacenar los datos. Utilizaremos dos bases de datos, una local y una remota. Como base de datos local utilizaremos MongoDB, debidamente instalada en la RaspberryPi. Como base de datos remota vamos a utilizar una base de datos localizada en la UPV, que utiliza el servicio de FiWare. Esta base de datos ha sido proporcionada por el tutor y cotutor del TFG.

Para poder realizar la comunicación de manera remota, se ha establecido en la RaspberryPi y en el router al que estaba conectado, una IP fija en la RaspberryPi. También se han abierto los puertos necesarios para poder realizar comunicaciones con esta, como por ejemplo, el puerto 1880 de Node-RED el puerto 23 para SSH (Se ha elegido ese puerto ya que había otro dispositivo RaspberryPi que ocupaba el puerto típico 22 de SSH) y el puerto 5900 para poder utilizar la aplicación VNC Viewer/Server y visualizar la pantalla de RaspberryPi en un ordenador remoto.

Adicionalmente, como el servidor FiWare se encontraba en la UPV, se ha necesitado realizar VPN hacia la UPV desde la RaspberryPi, utilizando el paquete vpnc de RaspberryPi.

A continuación, definimos como obtendremos los datos atmosféricos de la habitacíon donde se encuentre la RaspberryPi. Para ello, utilizaremos los sensores de nuestro dispositivo SenseHat, conectado a la Raspberry. Estos son sensores de temperatura, presión y humedad. Para ello utilizaremos los nodos apropiados de SenseHat que encontramos en Node-RED. Al obtener los datos, tendremos que tratarlos para tener un archivo JSON y poder mandarlos a nuestras bases de datos, con el formato que cada una nos pide.

Ahora, tras obtener datos de SenseHat, procedemos a enviar datos al mismo, para poder mostrar alertas y mensajes en su pantalla de LEDs. Para ello necesitaremos enviar comandos con el formato apropiado, a nuestro dispositivo SenseHat. Para enviar comandos, usaremos el nodo apropiado de SenseHat que se encuentra en Node-RED.

Al comprobar que funciona, creamos un dashboard con todos estos elementos, mostrando así, tres gráficas que almacenan los datos de temperatura, presión y humedad de nuestra habitación, y un menú en el que puedes mandar mensajes, alertas de colores y modificar el comportamiento de los leds de nuestro dispositivo SenseHat.

Vamos a proceder a integrar más servicios IoT a nuestro dashboard de Node-RED. Como primera instancia se ha elegido obtener datos atmosféricos de la ciudad que nosotros deseemos. Utilizaremos la API de la página web OpenWeather. Para mostrar esto en el dashboard, se han utilizado los nodos de Node-RED que te permiten añadir elementos a tu dashboard. Se ha añadido un campo en el que introduces el nombre de la ciudad, un indicador, que nos muestra la temperatura de la ciudad y dos campos de texto que nos confirman la ciudad y nos indican el estado del cielo. También existe un mapa donde se muestra la ubicación de esa ciudad.

Para complementar este servicio, también se ha utilizado la API de twitter. Gracias a esta API, podemos ver lo que la comunidad de twitter, twittea sobre un tema y donde lo hace. Para ello se ha introducido un campo donde se puede insertar texto, y en el introduces el tema deseado. Al hacer la búsqueda, automáticamente empezaran a salir puntos en el mapa del mundo, y si haces

click en uno de ellos, te mostrará en detalle, el twit, la persona que ha twitteado y muchos datos más.

Para finalizar el proyecto, se ha realizado una aplicación para dispositivos Android. La función de esta aplicación es, por un lado actualizar la localización de nuestro dispositivo, y por otro, poder mandar comandos al dispositivo SenseHat de nuestra Raspberry Pi. En esta aplicación tendremos dos actividades. La primera actividad muestra un mapa de GoogleMaps y dos botones. El primer botón actualizará la posición de nuestro dispositivo, y esta se mostrará en el mapa de Google Maps. En el momento que hemos pulsado el botón, se han enviado los datos a FiWare de la localización del dispositivo. El segundo botón de la primera actividad te lleva a la segunda actividad. Esta segunda actividad manda comandos a nuestro dispositivo SenseHat de la Raspberry. Para ello, existe un formulario con tres campos de texto y un botón, en el que el primero incluiría el mensaje a mandar, el segundo sería el color del texto y el tercero el color de fondo del texto. Al pulsar el botón del formulario, se enviarían los datos a FiWare, y de FiWare a través de Node-RED se mandarían a SenseHat.

En los apartados que vienen a continuación, explicaremos con más detalle todo el proceso del proyecto.

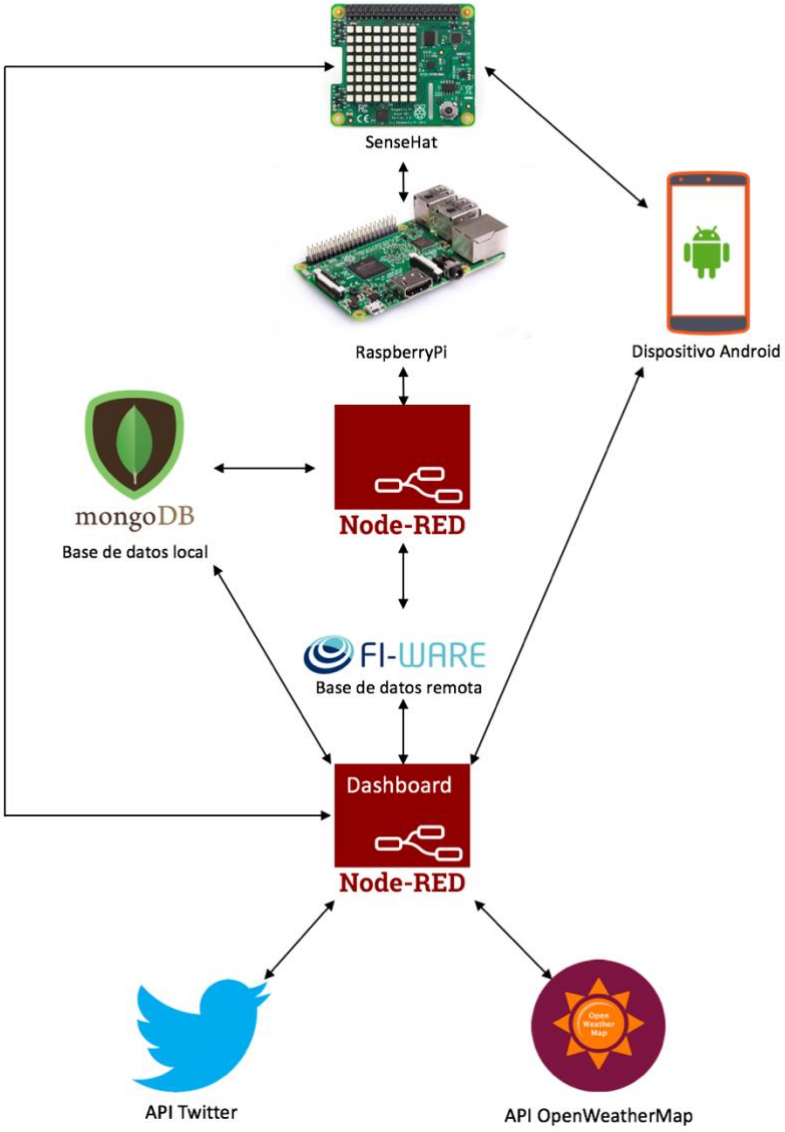

Figura 4-1 Diagrama del proyecto

4.2 Tab Home

4.2.1 Obtención de datos

SenseHat ofrece varios tipos de datos, como datos del ambiente y datos de movimiento. Nosotros nos centraremos en los datos de ambiente, ya que son los que nos interesan. Para obtener estos datos, previamente hay que haber instalado los nodos de SenseHat en Node-RED.

Cuando ya tenemos los nodos de SenseHat, hay que escoger el nodo de SenseHat de salida y hay que configurarlo de la manera correcta, para que nos envíe los datos que nosotros queremos. Para configurarlo, tenemos que marcar la casilla de *Environment events* para que nos mande los datos del ambiente. Para acceder a la configuración, solo hace falta hacer doble click en el nodo.

Con los sensores de SenseHat, el dato que recibimos de temperatura no es el real, ya que el sensor de temperatura se encuentra muy cerca de la cpu, que esta tiene una temperatura muy elevada, por lo tanto, hay que realizar una corrección, restándole a la temperatura obtenida, un 85% de la temperatura de la cpu. Este valor se ha obtenido, observando durante cierto tiempo la temperatura en la habitación y adaptándola para que la temperatura fuera la correcta.

El formato de los datos recibidos por SenseHat, es un objeto JSON, el cual tiene dentro los parámetros de temperatura:

```
object
temperature: 36.78
humidity: 34.63
pressure: 1023.62
```

Estos datos deberán ser transformados para tener un tipo de formato específico y poder almacenarlos.

4.2.2 Almacenamiento de datos

Cuando ya tenemos los datos deseados, deberemos almacenarlos en las dos bases de datos que tenemos.

4.2.2.1 Almacenamiento en MongoDB

Para almacenar los datos en MongoDB, primero debemos instalar la base de datos en nuestra RaspberryPi y los nodos necesarios en Node-RED.

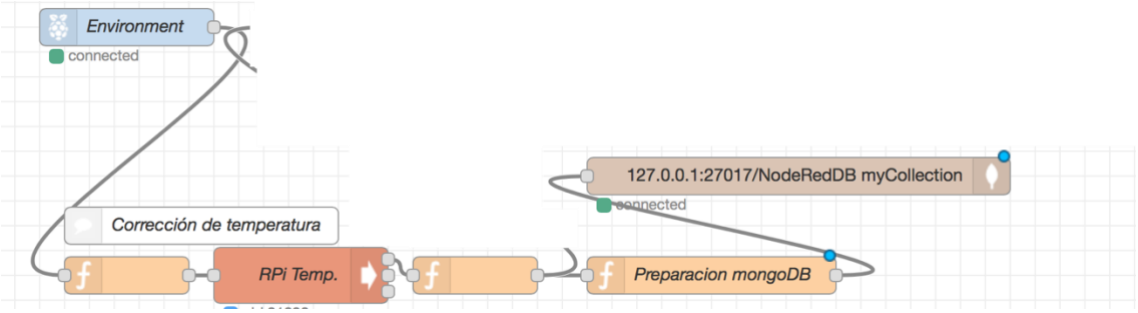

Figura 4-2 Diagrama de nodos de almacenamiento de datos en MongoDB

Antes de preparar los datos, tenemos que corregir el dato de Temperatura, ya que como hemos comentado anteriormente, este no es real debido a la cercanía del sensor de temperatura y la cpu. Se ha usado la siguiente ecuación para la corrección de la temperatura:

$$Temperatura = Temperatura_{sensor} - \frac{Temperatura_{cpu}}{1.15}$$

Ecuación 1 Corrección de temperatura

La Temperatura de la cpu se obtiene del nodo *RPi Temp* y en un nodo *function* a continuación se utiliza esta ecuación, ya teniendo el dato del sensor de temperatura.

Si ya tenemos todo como deseamos, debemos preparar los datos para que la base de datos sea capaz de comprenderlos, y los almacene de manera correcta. Para ello tenemos que utilizar un nodo *function*, proporcionado por Node-RED que te permite introducir tus propias funciones JavaScript. El formato a utilizar es el siguiente:

```
input=msg.payload;
msg.payload={
    "_id":"ObjectId('Environment')",
    "temperature":{
        "value":input.temperature,
        "type": "double"
    },
    "humidity":{
        "value":input.humidity,
        "type": "double"
    },
    "pressure":{
        "value":input.pressure,
        "type":"double"
    }
}
return msg;
```

Siendo, `msg.payload`, los datos recibidos de SenseHat. Este script lo que realiza es, coge los valores de los datos recibidos por SenseHat (`input.temperature`, `input.humidity` e `input.presure`) y los coloca correctamente en el archivo JSON que este genera. Al enviar (`return msg;`) el texto a la base de datos, este lo interpreta como un archivo JSON y almacena los datos en un Objeto con id Environment.

Para enviar los datos ya preparados a la base de datos, tenemos que configurar el nodo de recepción de datos de MongoDB, para que se conecte con ella. La configuración es la siguiente:

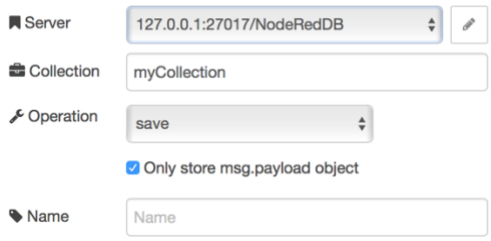

Figura 4-3 Configuración nodo MongoDB para recepción de datos

Ya tenemos todo configurado para almacenar datos en MongoDB

4.2.2.2 Almacenamiento en FiWare

El servidor de FiWare se encuentra dentro de la UPV, por lo tanto, para poder almacenar datos en este, tenemos que, o estar dentro de la UPV, o realizar una vpn a la UPV. Al tener nuestra Raspberry en una casa, lo que haremos es hacer vpn a la UPV.

Para almacenar datos en FiWare, no es necesario la instalación de ningún nodo adicional, ya que la comunicación con este servidor se realiza a través de peticiones HTTP. Usaremos los nodos ya instalados de HTTP.

Figura 4-4 Diagrama de nodos de creación de entidad en FiWare

Primero, tenemos que crear la entidad que vaya a almacenar los datos de nuestra habitación, por lo tanto, se hará una petición HTTP de tipo POST. Para ello debemos preparar el payload del mensaje con un nodo *function* (Preparación payload). La preparación del payload es la siguiente:

```
msg.payload={
    "id":"Environment",
    "type":"Environment",
    "temperature":{
        "value":0,
        "type": "double"
    },
     "humidity":{
        "value":0,
        "type": "double"
    },
    "pressure":{
        "value":0,
        "type":"double"
    }
}
msg.payload=JSON.stringify(msg.payload);
return msg;
```

A continuación, debemos poner las cabeceras necesarias para que la recepción de los datos sea correcta. Para ello también usaremos un nodo *function* (Cabeceras). Antes de poner las cabeceras nuevas, tenemos que eliminar las que ya había previamente, para que no existan errores. Las cabeceras son las siguientes:

```
msg.headers={};
msg.headers['Content-Type']='application/json';
msg.headers['Accept']='application/json';
msg.headers['FIWARE-Service']='aulaiot';
msg.headers['FIWARE-ServicePath']='/mario';
return msg;
```

Ya tenemos el mensaje formado como queremos, ahora utilizaremos el nodo *http request* para mandar una petición POST al servidor. La configuración del nodo *http request* es la siguiente:

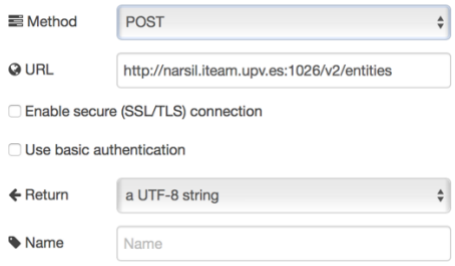

Figura 4-5 Configuracíon nodo HTTP request para creación de entidades

Ya está todo preparado para la creación de entidad, simplemente hacemos click en el botón que tiene el nodo *timestamp* para mandar una marca de tiempo, y esta enviará al servidor la petición HTTP POST. El nodo de *msg.payload* nos muestra por la consoloa de debug la respuesta del servidor. Si la respuesta es una string vacía, la entidad ha sido creada. Si ha habido algún error, este se mostrará en la consola de debug.

Ya tenemos la entidad creada, ahora lo que tenemos que hacer es actualizar los datos que recibimod y subirlos al servidor.

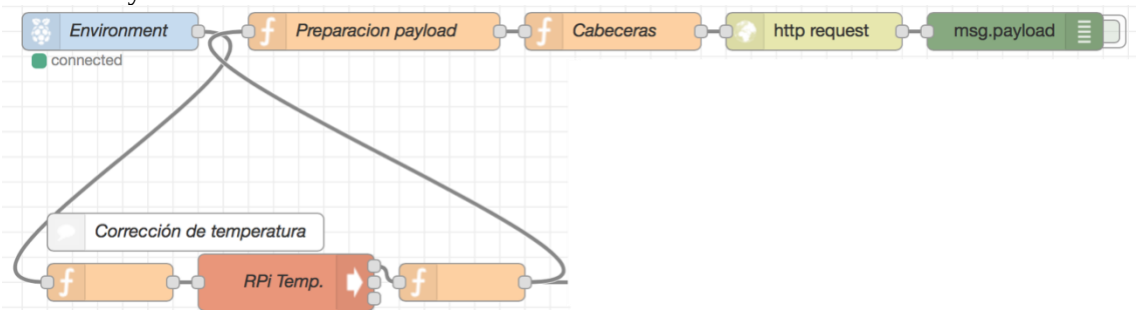

Figura 4-6 Diagrama de nodos para la actualización de los datos en FiWare

Tenemos que corregir el dato de la temperatura, siguiendo los mismos pasos que se han comentado en el apartado anterior.

Cuando tenemos todos los datos que queremos, tenemos que preparar el payload del mensaje. Este mensaje sigue una estructura similar que el que utilizábamos para el almacenamiento en MongoDB. La estructura del payload del mensaje se establece en el nodo *function* (Preparacion payload), y es la siguiente:

```
input=msg.payload;
msg.payload={
    "temperature":{
        "value":input.temperature,
        "type": "double"
    },
    "humidity":{
        "value":input.humidity,
        "type": "double"
    },
    "pressure":{
        "value":input.pressure,
        "type":"double"
    }
}
msg.payload=JSON.stringify(msg.payload);
return msg;
```

Como vemos, para actualizar el mensaje no necesitamos introducir el nombre de la entidad que vamos a actualiar, en la preparación de payload. Esto se reflejará en la URL que vamos a utilizar.

A continuación, tenemos que preparar las cabeceras para la actualización de los datos, esto se realiza en el nodo *function* (Cabeceras), siendo las mismas cabeceras que utilizamos para la creación de la entidad.

El nodo *http request* lo configuraremos para que realice peticiones PUT a una determinada URL. Esta URL contendrá la entidad a actualizar, en este caso, Environment. También será necesario añadir a la URL el path /attrs. La configuración es la siguiente:

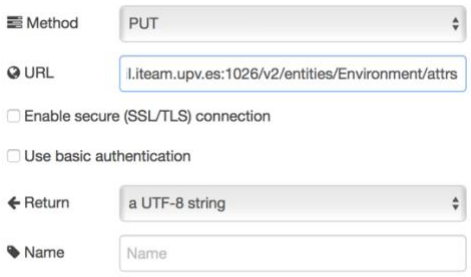

Figura 4-7 Configuración nodo HTTP request para la actualización de entidades

Si el nodo *msg.payload* nos devuelve una string vacía, significa que ha sido actualizada correctamente. Si nos da un error, este se reflejará en la consola de debug.

El nodo Environment automáticamente envía los datos cada segundo, por lo tanto la entidad será actualizada cada segundo.

4.2.3 Recepción de datos

Una vez ya tenemos los datos almacenados en nuestra base de datos, podemos proceder a descargarlos para visualizarlos en nuestro dashboard. Como tenemos dos bases de datos, tenemos que prepararlas, por si falla una, poder utilizar la otra.

4.2.3.1 Recepción desde MongoDB

Para MongoDB, necesitaremos usar los nodos instalados en Node-RED de MongoDB.

Figura 4-8 Diagrama de nodos de recepción de datos desde MongoDB

El primer nodo, *timestamp*, envía una marca de tiempo para que recibamos los datos en cada segundo.

Tendremos que configurar el nodo de MongoDB para que se conecte a la base de datos local y recibamos los datos. La configuración de este nodo es la siguiente:

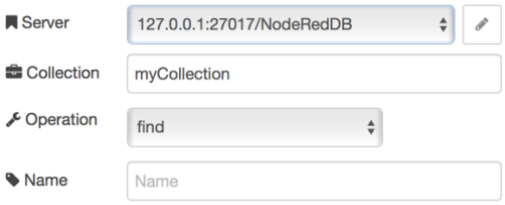

Figura 4-9 Configuración nodo MongoDB

Los siguientes nodos *function* preparan los datos para poder dejarlos en el formato que queremos leerlos.

Primer nodo *function*

```
return {"payload" : msg.payload.pop()};
```

Segundo nodo *function*

```
input=msg.payload;
pos1=input._id.indexOf("'");
pos2=input._id.lastIndexOf("'");
id=input._id.slice(pos1+1,pos2);
msg.payload={
    "id":id,
    "type":id,
    "temperature":{
        "value":input.temperature.value,
        "type": "double"
    },
```

```
    "humidity":{
        "value":input.humidity.value,
        "type": "double"
    },
    "pressure":{
        "value":input.pressure.value,
        "type":"double"
    }
}
return msg;
```

El nodo de *msg.payload* nos mostrará que los datos han sido recibidos correctamente

4.2.3.2 Rececpción desde FiWare

Como la comunicación con FiWare funciona a través de peticiones HTTP, en este caso deberemos utilizar una petición HTTP del tipo GET a la URL de la entidad deseada, para poder recibir los datos.

Figura 4-10 Diagrama de nodos de recepción de datos desde FiWare

Como antes, el primer nodo, *timestamp*, envía una marca de tiempo para que recibamos los datos en cada segundo.

El nodo *function* añade las cabeceras necesarias a la petición HTTP de tipo GET para recibir los datos en el formato que qureremos. Las cabeceras son las siguientes:

```
msg.headers={};
msg.headers['Accept']='application/json';
msg.headers['FIWARE-Service']='aulaiot';
msg.headers['FIWARE-ServicePath']='/mario';
return msg;
```

Gracias a estas cabeceras, recibiremos los datos en formato JSON. Cuando ya tenemos las cabeceras preparadas, podemos realizar la petición con el nodo *http request* configurándolo para que sea una petición de tipo GET e introduciendo la URL correcta.

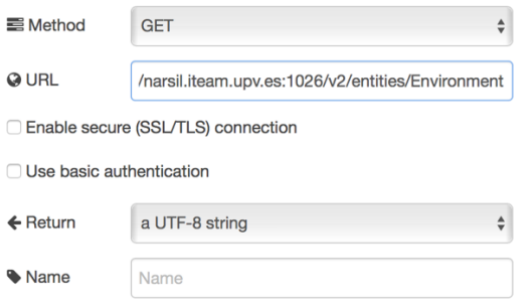

Figura 4-11 Configuración nodo HTTP request para recepción de datos desde FiWare

Como vemos, en la URL encontramos el path de la entidad sobre la que queremos hacer la reepción de datos.

Hemos recibido un texto que tiene formato JSON, pero Node-RED no podrá entenderlo hasta que no sea un objeto JSON, por lo tanto usaremos el nodo *JSON* que nos traducirá el texto a un objeto JSON.

Si todo ha salido bien, usando el nodo *msg.payload*, veremos e la consola de debug los datos recibidos.

4.2.4 Mostrar los datos

Para mostrar los datos usaremos el dashboard de Node-RED. Los nodos de dashboard y el dashboard requieren de instalación previa, por lo que se instalarán desde la librería de Node-RED. El dashboard está dividido en Tabs y Links. Los Tabs son pestañas del dashboard donde se pueden mostrar distintos elementos y los links son enlaces que te dirigen a otras webs. Para esta parte vamos a crear un Tab que denominaremos *"Home"*.

En este Tab añadiremos gráficas para mostrar los datos que hemos recibido a través de nuestra base de datos. Utilizaremos una gráfica para temperatura, otra gráfica para presión y otra gráfica para humedad.

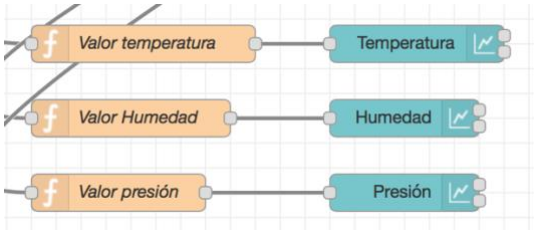

Figura 4-12 Diagrama de nodos para las gráficas

Antes de mandar los datos a las gráficas, primero tenemos que prepararlos para dibujar lo que nosotros deseemos, por lo tanto, necesitaremos un nodo *function* para obtener los valores de los distintos datos, y los enviaremos a los nodos de las gráficas. Para obtener estos valores, usaremos la siguiente línea de JavaScript.

```
return {topic:(dato),"payload":msg.payload.(dato).value};
```

Siendo, *(dato)* el nombre del dato a enviar a la gráfica. Este nodo *function* lo conectamos al nodo gráfica de dashboard. Como tendremos 3 gráficas, para cada una elegiremos un color. Se ha configurado el eje X para tener como máximo 1000 puntos.

Estas gráficas se deben añadir a un grupo, perteneciente a un Tab. Para cada gráfica se creará un Grupo, y ese grupo pertenecerá al Tab Home.

Ya tenemos todo preparado para visualizar. Ahora accederemos al dashboard para observar los resultados. Para acceder al dashboard, podemos añadir /ui/ al path con el que configuramos Node-RED, o haciendo al icono que se encuentra debajo de la pestaña de dashboard en la parte derecha de la pantalla de Node-RED.

Al entrar tenemos que ver que estamos en el Tab del dashboard que queremos. Eso se comprueba haciendo click en las tres líneas que se encuentran arriba a la izquierda, y hacemos click en el que nos interesa. En este caso, en Home. Estas son las gráficas que hemos configurado anteriormente:

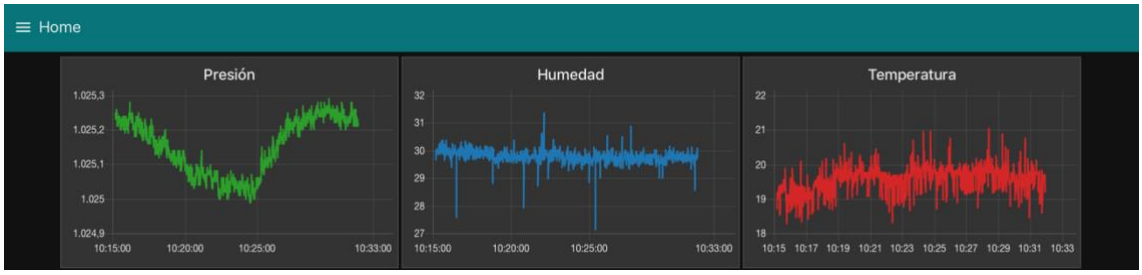

Figura 4-13 Gráficas de Presión, Humedad y Temperatura

4.2.5 Envío de instrucciones a SenseHat

Como se ha comentado antes, vamos a enviar comandos a RaspberryPi, para enviar instrucciones a SenseHat. Estas instrucciones lo que van a hacer es activar la pantalla de LEDs que SenseHat tiene colocada en la parte superior. Vamos a distinguir dos tipos de instrucciones, una primera que encenderá toda la pantalla del color deseado, puede ser utilizada para mandar alertas. La segunda enviará un mensaje, configurable así el color del texto y el de fondo. También podemos cambiar el brillo de la pantalla de LEDs. Todo esto será configurable desde el Tab Home del dashboard. A continuación hablaremos sobre el primer tipo de instrucción.

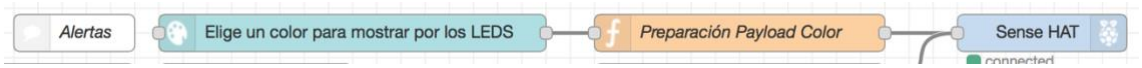

Figura 4-14 Diagrama de nodos instrucción alertas

Para el primer tipo de instrucción, utilizaremos un nodo *colour picker* que nos permitirá mostrar en el dashboard un elector de colores y al elegir un color, este nodo mandará el código de ese color al siguiente nodo. Al recibir el color, tenemos que prepararlo para que, al mandarlo a SenseHat lo entienda, y consiga encender toda la pantalla. La preparación del mensaje que se enviará es la siguiente:

```
color=msg.payload;
if(color!==""){
msg.payload="";
msg.payload="*,*,#"+color;
return msg;
}
```

De esta manera, a SenseHat le llegará un mensaje `"*,*,#FF0000"` (color Rojo como ejemplo) y toda la pantalla se encenderá de color Rojo. Los asteriscos indican las posiciones vertical y horizontal del cuadro de leds que deben encenderse, al utilizar los `*` le indicamos que son todos los LEDs. Finalmente conectamos este nodo *function* al nodo de entrada de SenseHat y cada vez que elijamos un color, la pantalla se encenderá.

Para la segunda parte, necesitábamos tres tipos de instrucciones que enviar a SenseHat, y necesitábamos que se mandaran a la vez, por lo tanto necesitamos el uso de FiWare para poder obtener el resultado deseado. Los tres tipos de datos son, el mensaje que queremos enviar, el color del mensaje y el color de fondo del mensaje. Para que los tres se mandaran a la vez, la solución a la que llegamos fue, crear una entidad en Node-RED denominada *LEDSmens*. Los atributos que tenía esta entidad son, mensaje de los leds, color del mensaje, color de fondo y un atributo *update* que permitiría saber cuando se debe mostrar el mensaje en los LEDs.

Figura 4-15 Diagrama de nodos instrucción de mensaje a SenseHat

En la primera fila de nodos vemos como tenemos un elector de colores, este es el que nos elegirá el color del mensaje a mostrar. Tiene similar funcionamiento al elector de colores de las alertas, al escoger un color, este lo manda al siguiente nodo. El nodo de preparación de Payload, introduce el valor del color, en formato JSON para poder mandarlo mediante una petición HTTP de tipo PUT a la entidad creada en FiWare previamente. También hay que añadirle al color una almoahdilla para que SenseHat lo entienda. Este es el script para la preparación del color del mensaje:

```
color="#"+msg.payload;
msg.payload="";
msg.payload={
    "value":color,
    "type":"string"
};
return msg;
```

Cuando el mensaje ya esta preparado para que FiWare y SenseHat lo entiendan, tenemos que añadirle las cabeceras apropiadas para realizar la petición PUT. Las cabeceras son las mismas que hemos utilizado para otras actualizaciones de otras entidades. Al llegar al nodo de *http request* la url cambiaría ya que ahora solo vamos a actualizar un dato, por lo que tenemos que añadir a la url el path del atributo. En nuestro caso, este atributo sería el atributo *color*, y la URL sería http://narsil.iteam.upv.es:1026/v2/entities/LEDSmens/attrs/color. En este momento, la petición ha sido enviada y el dato almacenado.

Seguimos el mismo proceso para la elección del color de fondo, utilizando otro nodo elector de color y otra fila de nodos para que no sustituya el valor de color de mensaje previamente elegido. Aquí también cambiaría la URL ya que ahora estamos modificando otro atributo, para este caso el atributo es llamado *background*. Siguiendo todos los pasos anteriores la petición será enviada y el dato almacenado.

Para el mensaje, la diferencia respecto a los anteriores es que ahora tendremos un campo de texto a rellenar, y se deberá pulsar intro para mandar el mensaje escrito. La preparación del payload es igual, y el atributo se llama *mens*. Al enviar el mensaje, también enviaremos el atributo *update* y lo almacenaremos con valor 1 cada vez que introduzcamos un mensaje nuevo. A continuación se explicará la función de este atributo.

Cuando ya tenemos todos los datos almacenados y hemos enviado el mensaje, el atributo *update* tendrá valor 1. Para mostrar los mensajes por los LEDs, estaremos constantemente leyendo este atributo. En el momento que tenga valor 1, comenzará el proceso de envío de la instrucción a SenseHat.

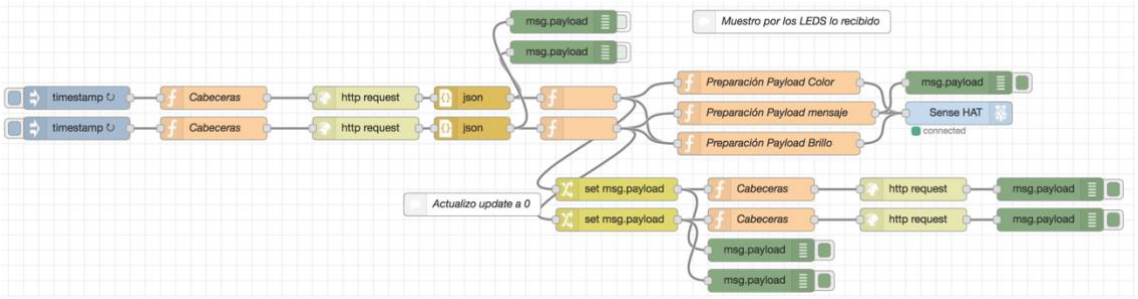

Figura 4-16 Diagrama de nodos mostrar mensaje por LEDs

Como vemos, en el diagrama, al principio tenemos dos caminos principales, uno para mostrar el mensaje y otro para mostrar alertas. El camino de mostrar alertas se utilizará posteriormente y tiene un proceso similar al de mostrar el mensaje. Ahora vamos a la explicación del camino para mostrar el mensaje.

El nodo de *timestamp* manda una marca de tiempo cada segundo para realizar una petición del tipo GET al servidor de FiWare. Las cabeceras son las mismas que hemos utilizado en otras peticiones de tipo GET. En este caso la URL a la que se realizará la petición es `http://narsil.iteam.upv.es:1026/v2/entities/LEDSmens` para obtener los atributos que queremos. Como previamete se ha hecho, necesitamos transformar el texto recibido a un objeto JSON con el nodo de JSON. El siguiente nodo, es un nodo *function* que nos permitirá ver en que valor está el atributo *update*, si es 1 o si es 0. Si el valor es 0, la comunicación acaba ahí puesto que eso significa que no hay que enviar nada a SenseHat. Si el valor es 1, el camino se divide en 2 para realizar dos acciones simultaneas. El camino superior preparará los atributos recibidos para poder enviarlos a SenseHat. Para que SenseHat entienda lo recibido, es necesario atribuirle un valor y nombre concreto a los datos que se han recibido. Esto se realizará en el nodo *function* Preparación payload mensaje.

```
mens=msg.payload.mens.value;
color=msg.payload.color.value;
background=msg.payload.background.value;
speed=msg.payload.speed.value;
msg.payload="";
msg.payload=mens;
msg.color=color;
msg.speed=speed;
msg.background=background;
return msg;
```

De esta manera, SenseHat entenderá el mensaje y actuará de la manera que queremos.

La función del segundo camino es la de actualizar el atributo *update* para que vuelva a tener valor 0. Esto se realiza asignando el valor 0 al *msg.payload* y enviándolo por una petición PUT. En este proceso utilizaremos una cabecera distinta, ya que lo que vamos a actualizar es solo el valor del atributo, por lo tanto la cabecera de `msg.headers['Content-Type']` tendrá el valor `'text/plain'`. La URL de la petición PUT también cambiará y será

http://narsil.iteam.upv.es:1026/v2/entities/LEDSmens/attrs/update/value. De esta manera, para enviar otro mensaje tendremos que cambiar los parámetros y actualizar el atributo *update* de la manera descrita anteriormente.

Como se ha mencionado al principio de este punto, también tenemos la opción de cambiar el brillo de la pantalla. Esto se reflejará cada vez que queramos mandar un mensaje o una alerta. Es decir, será un atributo añadido de las instrucciones que mandamos. A continuación vemos como se visualizaría esta parte en el dashboard.

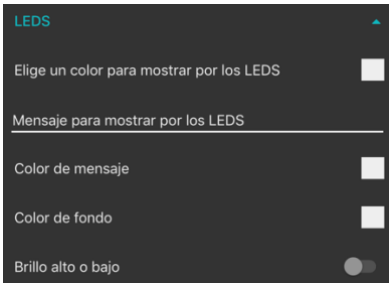

Figura 4-17 Campos para mandar instrucciones a SenseHat

Se han añadido dos elementos más al Tab Home, con la simple función de relleno y que los elementos queden bien colocados.

El resultado final del Tab Home es el siguiente, donde podemos ver todos los elementos colocados correctamente.

Figura 4-18 Tab Home

4.3 Tab El mundo

4.3.1 Función

Ya hemos creado una pestaña para la visualización de datos relacionados con nuestro hogar, donde visualizamos datos y podemos interactuar con él. El propósito actual es crear otra pestaña o Tab, en la que podamos interactuar con distintos servicios IoT relacionados con el mundo, ya sea visualizar el tiempo atmosférico de una ciudad, ver lo que twittea la gente sobre un tema en concreto y localizar donde lo han twitteado, buscar una ciudad y localizar nuestro dispositivo. Para ello, en la pestaña *El mundo* del dashboard necesitaremos un mapa del mundo. Instalaremos *worldmap* desde la biblioteca de Node-RED. Este mapa nos servirá para mostrar los datos que queremos.

4.3.2 OpenWeatherMap y worldmap

OpenWeatherMap nos ofrece una API, con la cual realizando una petición HTTP de tipo GET a una URL determinada, podremos obtener los datos de la ciudad que queramos. Para ello necesitamos completar en la URL los campos necesarios para obtener los datos deseados.

Previamente, tenemos que obtener una API key, necesaria para que la API nos conceda los permisos para realizar búsquedas. Para obtener la API key tenemos que registrarnos en la página web de OpenWeatherMap. Una vez nos hayamos registrado, nos llegará al correo un mensaje con la API lista para ser usada.

La URL a la que tenemos que hacer la petición sigue el siguiente formato:

```
http://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={APIKEY}
```

Siendo `{CITY}` la ciudad sobre la que vamos a realizar la consulta y `{APIKEY}` la clave API que nos ha proporcionado la página web al registrarnos.

Esta petición nos devolverá un objeto JSON con muchos datos de la ciudad. Nosotros solo nos centraremos en los datos del día en el que estamos.

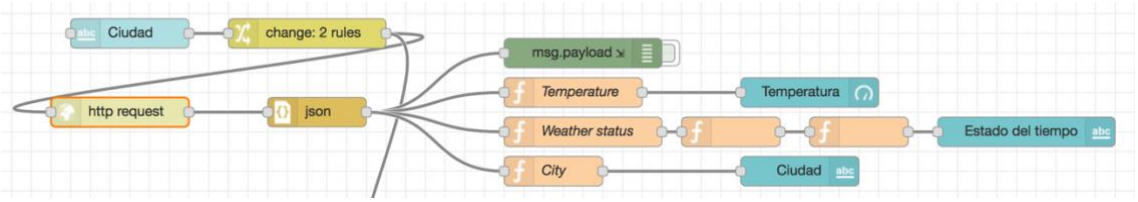

Figura 4-19 Diagrama de nodos OpenWeatherMap

Este es el diagrama de nodos que seguimos para obtener los datos del tiempo atmosférico. El primer nodo que vemos es un campo para introducir texto. Este campo se mostrará en la pestaña, en el cual introduciremos el nombre de la ciudad. El siguiente nodo renombra el valor de la ciudad para introducirlo en la URL del nodo *http request*. Cuando ya tenemos el nombre de la ciudad, lo introducimos en la URL y junto a la API key realizamos una petición HTTP de tipo GET. La respuesta de esta conexión es un texto con formato JSON que gracias al nodo JSON podremos transformar en un objeto JSON.

Ya tenemos los valores como objeto JSON, ahora tendremos que prepararlos para mostrarlos en distintos elementos de la pestaña. Primero, obtendremos el valor de la temperatura, el cual la API nos lo envía en grados kelvin, por lo que tendremos que pasarlo a grados Celsius y redondearlo, para que al mostrar la temperatura, los decimales no nos molesten. Este es el contenido del nodo *function* (Temperature)

```
msg.payload = Math.round((msg.payload.main.temp-273)*100)/100;
return msg;
```

Este valor es enviado al nodo indicador de dashboard y este mostrará la temperatura.

La siguiente rama obtiene el valor del estado del ambiente, este es una cadena de texto que se mostrará explicando la situación actual. Por ello, simplemente con obtener el objeto *msg.payload.description* tendremos acceso al estado del ambiente. Este será enviado a un nodo mostrador de texto y será visualizado en la pestaña.

La siguiente rama simplemente coge el nombre de la ciudad, del objeto JSON y lo envía a un nodo mostrador de texto.

Junto a esta funcionalidad, se ha añadido otra adicional que complementa a esta, y lo que hace es buscar la ciudad en el mapa mundial que se encuentra al lado de estos.

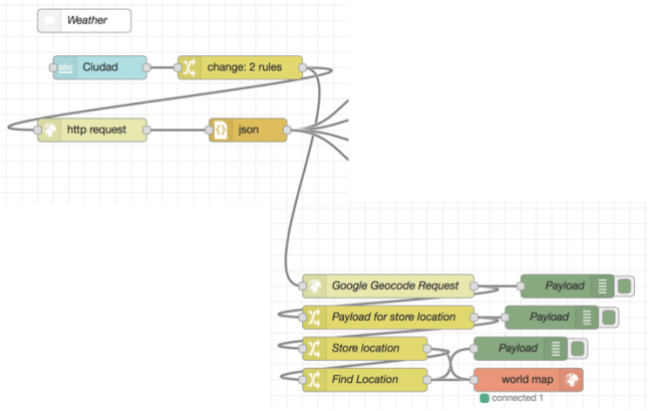

Figura 4-20 Diagrama de nodos buscador de ciudad

Este diagrama, utiliza la API de Geocoding de Google Maps. Para esta API también es necesario una API key, que se obtiene al registrarse en la página web de desarrolladores de Google, y al seguir ciertos pasos que te indican. Cuando ya has obtenido la API key de google, se realiza una petición HTTP del tipo GET a la siguiente URL.

```
https://maps.googleapis.com/maps/api/geocode/json?address{CITY}&key={APIKEY}&format=json
```

Siendo {CITY} la ciudad a buscar y {APIKEY} la API key obtenida anteriormente. Esta petición nos devuelve un objeto JSON del que obtendremos los valores de latitud y longitud de la ciudad consultada.

Para buscar la ciudad en el mapa de Node-RED, primero hay que almacenarla, ya que este no tiene los valores de las ciudades guardados, por lo tanto, utilizaremos la propiedad *store* de *msg* para guardarlo en el mapa. La longitu será almacenada en *msg.store.lon*, la latitud en *msg.store.lat* y será necesario almacenarla con un nombre para poder buscarla posteriormente, por lo tanto usaremos *msg.store.name*.

Cuando ya se ha almacenado la ciudad, procederemos a buscarla, utilizando la propiedad *msg.payload.command.search* teniendo almacenado ahí el nombre de la ciudad. Además también podemos ajustar el zoom del mapa usando *msg.payload.command.zoom*. Nosotros hemos utilizado el valor 13.

Cuando estos valores llegan al mapa, previamente habiendo almacenado los valores de longitud y latitud, este nos marcará el mapa la ciudad buscada.

Con todo esto, tenemos la parte de buscar la ciudad en el mapa y consultar su tiempo atmosférico. Se mostraría de la siguiente manera:

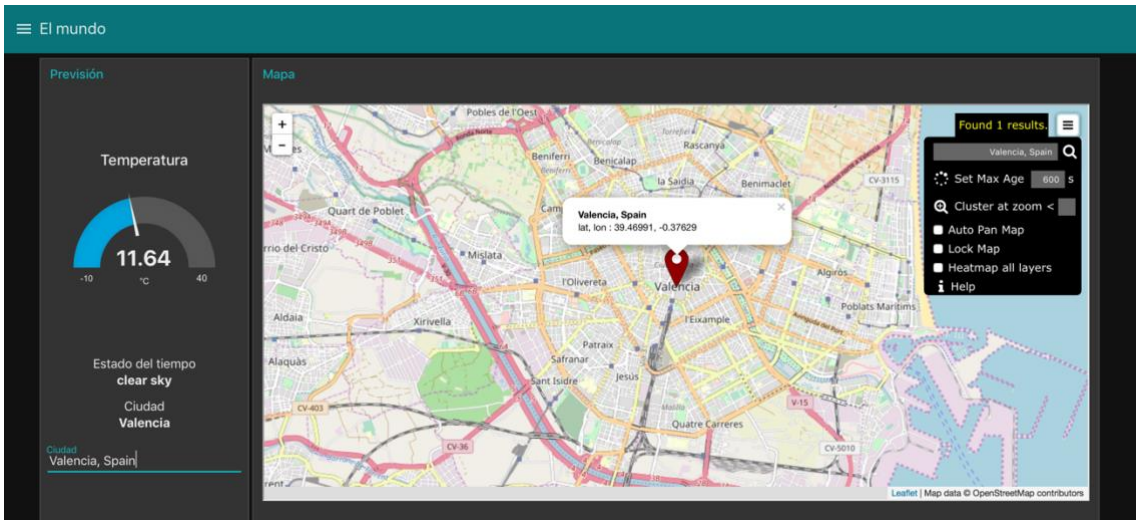

Figura 4-21 Tiempo atmosférico y mapa del mundo

4.3.3 Twitter y worldmap

La función de la API de twitter es obtener twits de cuentas públicas, introduciendo una palabra o frase, y este nos devolverá quien, el que y donde han twitteado. Volviendo a utilizar la API de Google Maps, podremos colocar en el mapa del mundo los twits. A continuación vamos a mostrar el diagrama de nodos que hemos utilizado.

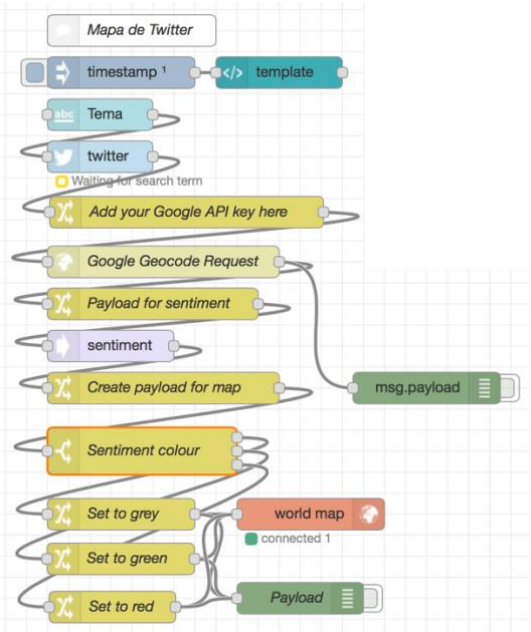

Figura 4-22 Diagrama de nodos de obtención de twits

Como vemos, primero tenemos que introducir un nodo para introducir las palabras sobre las cuales vamos a realizar la consulta. Para ello utilizaremos un nodo de introducción de texto que se mostrará en el dashboard, debajo del mapa del mundo. Este texto será enviado al nodo de twitter, y este nos devolverá todos los twits públicos que contengan esa palabra o palabras. Previamente debemos habernos logueado con nuestra cuenta de twitter, siguiendo los pasos que te indica el nodo de twitter al hacer doble click sobre el.

Cuando ya tenemos los twits a mostrar, tenemos que añadir al siguiente nodo nuestra API key, y en el siguiente nodo es en el que utilizaremos la API de Google Maps para obtener la latitud y longitud del lugar. Para ello se hace lo mismo que hemos comentado antes, una petición HTTP de tipo GET que nos devuelve un objeto JSON. A continuación vamos a crear unos sentiment,

que simplemente darán color al icono de twitter que será mostrado en el mapa. Esto depende de la interacción que haya tenido la gente con el twit. Esto sirve simplemente para asignarle un color al icono. Cuando ya tenemos todo preparado, mandamos todos los datos al wordmap, y este se encarga de colocarlos donde debe.

A continuación vemos una demostración de cómo se mostraría.

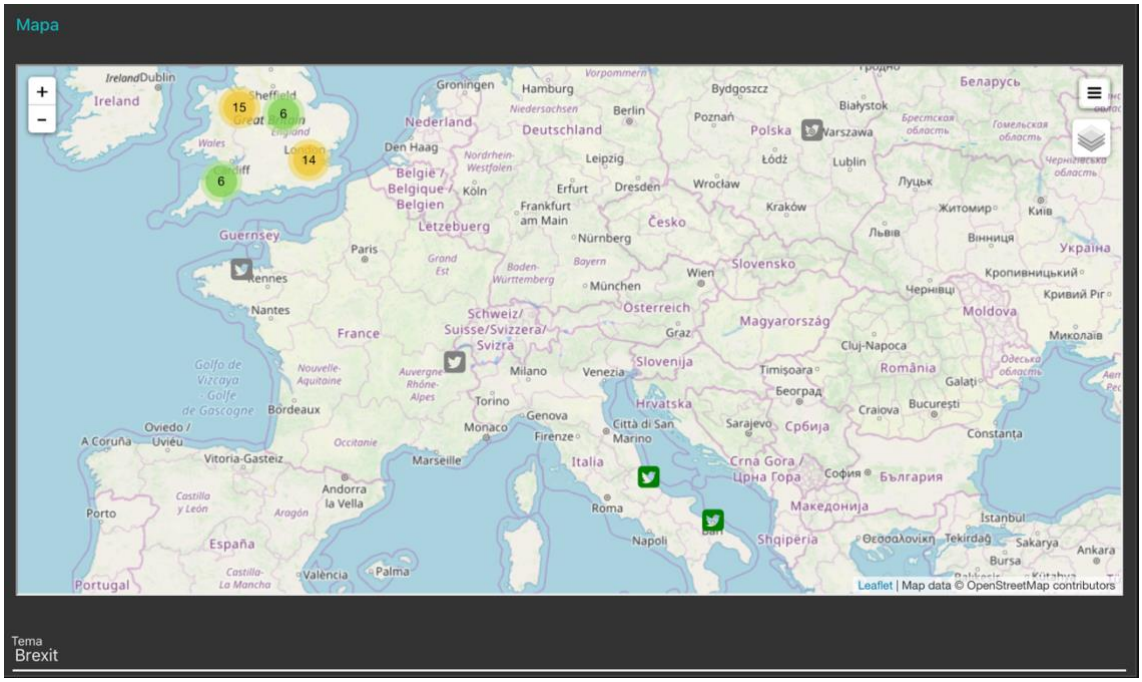

Figura 4-23 Ejemplo uso de la API de twitter y Google Maps

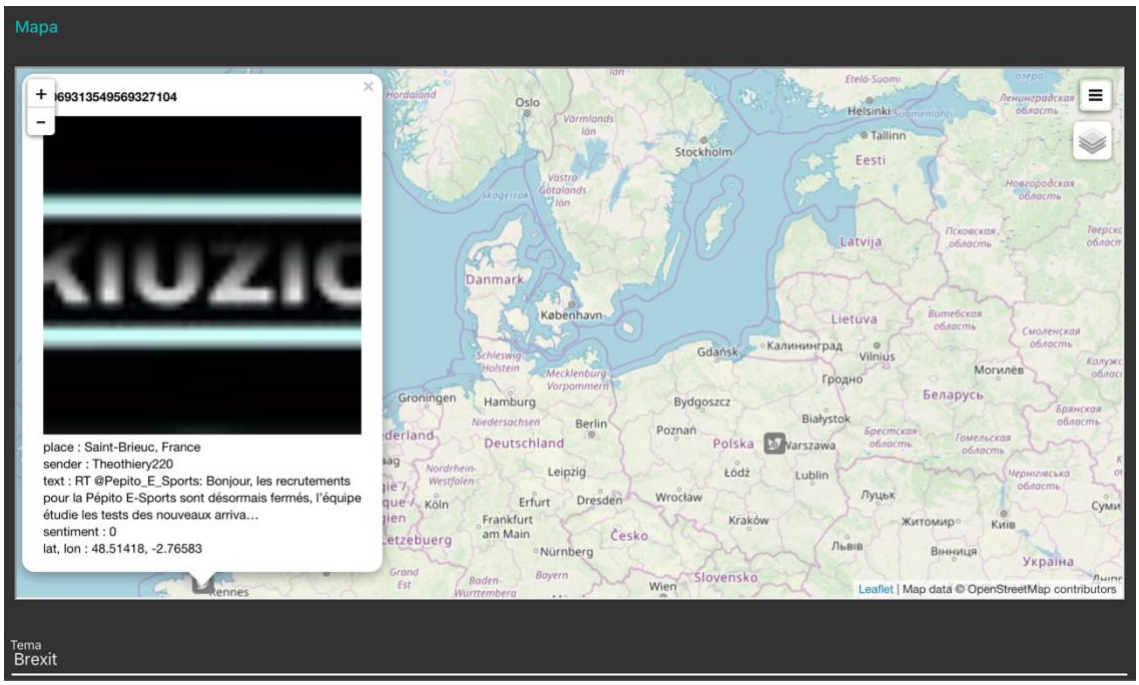

Figura 4-24 Visualizando los detalles de un twit

Como vemos, se muestran distintos iconos por el mapa del mundo y al pulsar en uno de ellos, se nos muestran los detalles.

4.3.4 Aplicación Android

La aplicación Android tiene como objetivo principal integrar el dispositivo móvil en los servicios IoT mencionados en los apartados anteriores. Nuestra aplicación Android permitirá conocer la última actualizacón de nuestro dispositivo Android y además permitirá enviar instrucciones a nuestra RaspberryPi como hacíamos desde el dashboard.

Para realizar nuestra aplicación, primero tendremos que crear un proyecto nuevo en Android Studio. Lo llamaremos GPStracking. Necesitamos que la aplicación tenga permisos para obtener la geolocalización del dispositivo asique habrá que darle los permisos necesarios en el archivo *manifest* de nuestro proyecto. El archivo manifest se encuentra en la siguiente ruta:

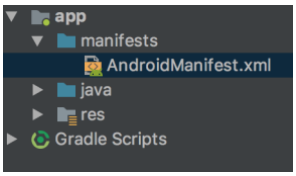

Figura 4-25 Ruta AndroidManifest

Este archivo contiene todos los permisos por los que la aplicación va a pedir acceso al iniciarse. Vamos a añadirle una línea que permita el acceso a la geolocalización.

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

Ahora al iniciar la aplicacíon, esta nos pedirá aceptar los permisos de geolocalización.

La aplicación consistirá en dos actividades, una para enviar la geolocalización y otra para enviar instrucciones a la RaspberryPi. Esto también será necesario reflejarlo en el archivo manifest, añadiendo la siguiente línea:

```
<activity android:name=".RaspBActivity"></activity>
```

De esta manera, la aplicación ya sabe que tiene otra actividad más.

La actividad principal (*ActivityMain*) el mapa de Google Maps y dos botones, uno para actualizar la geolocalización y otro para acceder a la actividad de RaspberryPi (*RaspBActivity*). La actividad principal tiene el código con el que se obtiene la geolocalización, que hace uso de objetos de la clase *LocationManager*.

```
locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
loc = locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

Con estos dos objetos, obtenemos la localización. Solo queremos que la localización se actualice cuando nosotros pulsemos el botón por lo tanto, tendremos que meter la obtención de la localización y el envío de la misma dentro de un *button.setOnClickListener(v)*, lo que hace esta función es, que no se ejecuta todo lo que hay dentro hasta que el botón no sea pulsado.

Los datos de localización serán enviados al servidor FiWare para que estos puedan ser descargados en el dashboard. Por lo tanto, antes de enviar los datos, será necesario crear una entidad nueva. Crearemos una entidad nueva llamada "Coords" de la misma manera que hemos creado todas las entidades. Esta tendrá el siguiente formato:

```
msg.payload={
    "id":"Coords",
    "type":"Coords",
    "lng":{
```

```
        "value":0,
        "type": "double"
    },
    "lat":{
        "value":0,
        "type": "double"
    },
}
msg.payload=JSON.stringify(msg.payload);
return msg;
```

Una vez la entidad está creada, podemos proceder a mandar los datos al servidor FiWare.

Para obtener los datos de localización, utilizaremos el objeto *loc* mencionado anteriormente y sus funciones *getLatitude()* y *getLongitude()*. Al tener los datos, procederemos a realizar la petición HTTP de tipo PUT a través de nuestra aplicación Android. Para ello tenemos primero que establecer una nueva conexión a la URL de nuestra entidad y establecer las cabeceras oportunas. Cuando ya hemos establecido la conexión, crearemos nuestro objeto JSON con el formato correcto, el que utilizamos para crear la entidad. En el código JavaScript, será necesario crear un objeto *OutputStreamWriter* para realizar la petición. Para mandar los datos de la longitud y latitud, utilizaremos la función *write(datos)* y estos se enviarán a la URL mencionada al principio. También moveremos el marcador del mapa que se encuentra en la aplicación, para que marque nuestra localización correcta. Al finalizar el envío de datos, tenemos que cerrar todas las conexiones abiertas anteriormente. No podemos olvidarnos, que al ser una petición HTTP, será obligatorio el control de excepciones, mediante las rutinas *try*-catch. El código que realiza todo lo descrito anteriormente es el siguiente:

```
Button boton = (Button) findViewById(R.id.button);
boton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      try {
         lat = loc.getLatitude();
         lng = loc.getLongitude();

         URL url = new URL("http://narsil.iteam.upv.es:1026/v2/entities/Coords/attrs");
         HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
         httpURLConnection.setDoOutput(true);
         httpURLConnection.setRequestMethod("PUT");
         httpURLConnection.setRequestProperty("Content-Type", "application/json");
         httpURLConnection.setRequestProperty("Accept", "application/json");
```

```
            httpURLConnection.setRequestProperty("FIWARE-Service", "aulaiot");
            httpURLConnection.setRequestProperty("FIWARE-ServicePath", "/mario");
            httpURLConnection.connect();

            JSONObject lng2 = new JSONObject();
            lng2.put("value", lng);
            lng2.put("type", "double");

            JSONObject lat2 = new JSONObject();
            lat2.put("value", lat);
            lat2.put("type", "double");

            JSONObject jsonObject = new JSONObject();
            jsonObject.put("lat", lat2);
            jsonObject.put("lng", lng2);

            OutputStreamWriter wr = new
OutputStreamWriter(httpURLConnection.getOutputStream());
            wr.write(jsonObject.toString());
            wr.flush();
            Log.d("response", httpURLConnection.getResponseMessage());
            Log.d("mensaje", jsonObject.toString());
            wr.close();
            httpURLConnection.disconnect();

            // Add a marker in Sydney and move the camera
            LatLng position = new LatLng(lat, lng);
            mMap.addMarker(new MarkerOptions().position(position).title("I'm here"));
            mMap.moveCamera(CameraUpdateFactory.newLatLng(position));
        } catch (MalformedURLException e) {
            e.printStackTrace();
            Log.d("error1", e.toString());
        } catch (IOException e) {
            e.printStackTrace();
            Log.d("error2", e.toString());
```

```
        } catch (JSONException e) {
            e.printStackTrace();
            Log.d("error3", e.toString());
        }
    }
});
```

Esta actividad también tiene un botón que nos redirige a otra actividad con otra función distinta, por lo tanto tendremos que establecer un segundo *button.setOnClickListener(v)* para que nos redirija a la siguiente actividad. Para ello crearemos un *Intent* de la nueva actividad y usaremos la función *startActivity(Activity)*. También tenemos que controlar las excepciones con la rutina *try-catch*. El código de esta funcionalidad es el siguiente:

```
Button boton2 = (Button) findViewById(R.id.button2);
boton2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            Intent Volver = new Intent(getApplicationContext(), RaspBActivity.class);
            startActivity(Volver);
        }catch(Exception e){
            e.printStackTrace();
            Log.d("error", e.toString());
        }
    }
});
```

Como el servidor FiWare se encuentra en la UPV, para realizar las pruebas de esta aplicación he requerido de la utilización de una VPN a la UPV desde mi dispositivo Android. A continuación vemos la aplicación en el dispositivo Android:

Figura 4-26 Actividad principal aplicación Android

Ahora vamos a hablar de la interconexión del dashboard con la aplicación, ya que en la pestaña *El mundo* encontramos un botón con el texto "Where's my device" que al pulsarlo, nos muestra la última actualización que hemos realizado desde nuestro dispositivo Android.

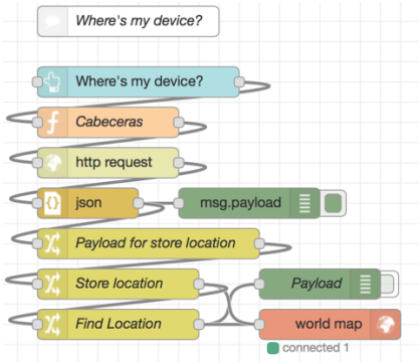

Figura 4-27 Diagrama de nodos para la obtención de localización en dashboard

El primer nodo es un botón que manda una marca de tiempo para obtener los datos. Las cabeceras son las mismas que hemos utilizado para otras peticiones HTTP de tipo GET. La URL que utilizaremos será la que tiene el path */Coords* y como en otras peticiones, también tendremos que transformar el texto a un objeto JSON, para poder trabajar con los datos. Primero, tendremos que guardar los datos en la propiedad *msg.store* para poder almacenarlos en el nodo *worldmap* de la misma manera que almacenábamos la ciudad consultada. Una vez almacenados, realizaremos la búsqueda con el nombre de "My position", la cual nos llevará al punto donde el dispositivo fue actualizado por última vez. A continuación se muestra como se visualizaría la localización al pulsar el botón "Where's my device":

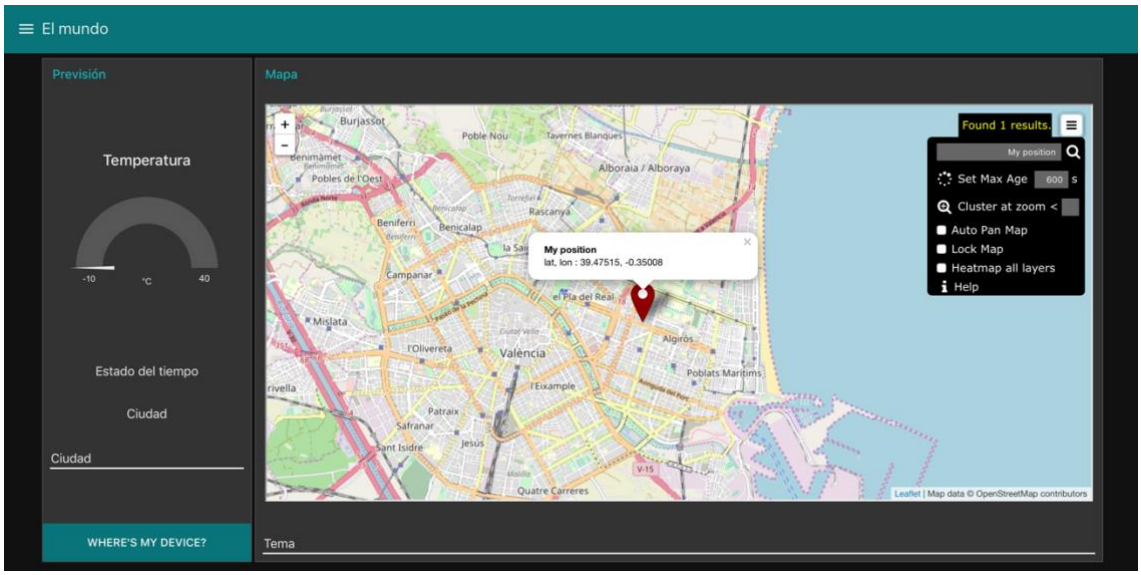

Figura 4-28 Visualización de la localización del dispositivo Android

Ahora vamos a hablar de la segunda actividad de nuestra aplicación Android. Esta actividad permite mandar instrucciones a la pantalla de LEDs de SenseHat, como hacíamos en la pestaña de Home. Para ello, hemos creado una entidad nueva en FiWare denominada "LEDS" que permite mandar alertas para que se encienda la pantalla de LEDs de un color único. Para el envío de mensajes a la pantalla se ha reutilizado la entidad creada anteriormente para la pestaña Home.

En nuestra aplicación, la actividad *RaspberryPActivity* contiene tres campos de texto a completar seguidos de un botón, y a continuación otro campo de texto a rellenar y otro botón. Los tres primeros campos de texto se utilizan para mandar un mensaje, con cierto color y otro color de fondo al pulsar el botón. El último campo de texto sirve para mandar un color y que encienda toda la pantalla de LEDs de un color al pulsar el último botón. Para ello utilizamos peticiones HTTP de tipo PUT como hemos hecho en el apartado anterior, introduciendo en el primer campo el mensaje deseado, el segundo el color en el que vaya el mensaje y el tercero, el color de fondo del mensaje. Tenemos que controlar que los campos no queden vacíos por lo que introducimos un *if-else* que lo controla. Las URL en este caso cambiaría respecto al anterior. El código que realiza toda la función es el siguiente:

```
Button boton = (Button) findViewById(R.id.button);
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            TextView mensaje = (TextView) findViewById(R.id.editText);
            TextView color = (TextView) findViewById(R.id.editText2);
            TextView fondo = (TextView) findViewById(R.id.editText3);
            if (mensaje.getText().toString().equals("") || color.getText().toString().equals("") ||
fondo.getText().toString().equals("")) {
                Context context = getApplicationContext();
                int duration = Toast.LENGTH_SHORT;
```

```
            CharSequence text = "No dejes ningún campo vacío";
            Toast toast = Toast.makeText(context, text, duration);
            toast.show();
        } else {
            URL url = new URL("http://narsil.iteam.upv.es:1026/v2/entities/LEDSmens/attrs");
            HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
            httpURLConnection.setDoOutput(true);
            httpURLConnection.setRequestMethod("PUT");
            httpURLConnection.setRequestProperty("Content-Type", "application/json");
            httpURLConnection.setRequestProperty("Accept", "application/json");
            httpURLConnection.setRequestProperty("FIWARE-Service", "aulaiot");
            httpURLConnection.setRequestProperty("FIWARE-ServicePath", "/mario");
            httpURLConnection.connect();

            JSONObject mensajeJ = new JSONObject();
            mensajeJ.put("value", mensaje.getText().toString());
            mensajeJ.put("type", "string");
            JSONObject colorJ = new JSONObject();
            colorJ.put("value", color.getText().toString());
            colorJ.put("type", "string");
            JSONObject fondoJ = new JSONObject();
            fondoJ.put("value", fondo.getText().toString());
            fondoJ.put("type", "string");
            JSONObject BrilloJ = new JSONObject();
            BrilloJ.put("value", "D1");
            BrilloJ.put("type", "string");
            JSONObject SpeedJ = new JSONObject();
            SpeedJ.put("value", "3");
            SpeedJ.put("type", "string");
            JSONObject UpdateJ = new JSONObject();
            UpdateJ.put("value", 1);
            UpdateJ.put("type", "int");
```

```
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("mens", mensajeJ);
            jsonObject.put("color", colorJ);
            jsonObject.put("background", fondoJ);
            jsonObject.put("speed", SpeedJ);
            jsonObject.put("bright", BrilloJ);
            jsonObject.put("update", UpdateJ);

            OutputStreamWriter wr = new
OutputStreamWriter(httpURLConnection.getOutputStream());
            wr.write(jsonObject.toString());
            wr.flush();
            Log.d("response", httpURLConnection.getResponseMessage());
            Log.d("mensaje", jsonObject.toString());
            wr.close();
            httpURLConnection.disconnect();

        }
      } catch (MalformedURLException e) {
        e.printStackTrace();
        Log.d("error1", e.toString());
      } catch (IOException e) {
        e.printStackTrace();
        Log.d("error2", e.toString());
      } catch (JSONException e) {
        e.printStackTrace();
        Log.d("error3", e.toString());
      }
    }
});
```

Ahora, para el envío de colores que nos servirán como alertas, utilizaremos el mismo método, cambiando los campos de texto que leeremos de la aplicación Android y cambiando la URL a la que va dirigida. El código en este caso es el siguiente:

```
Button boton2 = (Button) findViewById(R.id.button2);
boton2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            TextView alerta = (TextView) findViewById(R.id.editText4);
            if (alerta.getText().toString().equals("")) {
                Context context = getApplicationContext();
                int duration = Toast.LENGTH_SHORT;
                CharSequence text = "No dejes ningún campo vacío";
                Toast toast = Toast.makeText(context, text, duration);
                toast.show();
            } else {
                URL url = new URL("http://narsil.iteam.upv.es:1026/v2/entities/LEDS/attrs");
                HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
                httpURLConnection.setDoOutput(true);
                httpURLConnection.setRequestMethod("PUT");
                httpURLConnection.setRequestProperty("Content-Type", "application/json");
                httpURLConnection.setRequestProperty("Accept", "application/json");
                httpURLConnection.setRequestProperty("FIWARE-Service", "aulaiot");
                httpURLConnection.setRequestProperty("FIWARE-ServicePath", "/mario");
                httpURLConnection.connect();

                JSONObject alertaJ = new JSONObject();
                alertaJ.put("value", alerta.getText().toString());
                alertaJ.put("type", "string");
                JSONObject BrilloJ = new JSONObject();
                BrilloJ.put("value", "D1");
                BrilloJ.put("type", "string");
                JSONObject SpeedJ = new JSONObject();
                SpeedJ.put("value", "3");
                SpeedJ.put("type", "string");
                JSONObject UpdateJ = new JSONObject();
                UpdateJ.put("value", 1);
```

```
            UpdateJ.put("type", "int");



            JSONObject jsonObject = new JSONObject();
            jsonObject.put("color", alertaJ);
            jsonObject.put("speed", SpeedJ);
            jsonObject.put("bright", BrilloJ);
            jsonObject.put("update", UpdateJ);

            OutputStreamWriter wr = new
OutputStreamWriter(httpURLConnection.getOutputStream());
            wr.write(jsonObject.toString());
            wr.flush();
            Log.d("response", httpURLConnection.getResponseMessage());
            Log.d("mensaje", jsonObject.toString());
            wr.close();
            httpURLConnection.disconnect();

        }
      } catch (MalformedURLException e) {
        e.printStackTrace();
        Log.d("error1", e.toString());
      } catch (IOException e) {
        e.printStackTrace();
        Log.d("error2", e.toString());
      } catch (JSONException e) {
        e.printStackTrace();
        Log.d("error3", e.toString());
      }
    }
});
```

A continuación mostraremos como se vería la actividad de la aplicación y la pantalla de LEDs

Figura 4-29 Actividad RaspberryP con los campos vacíos

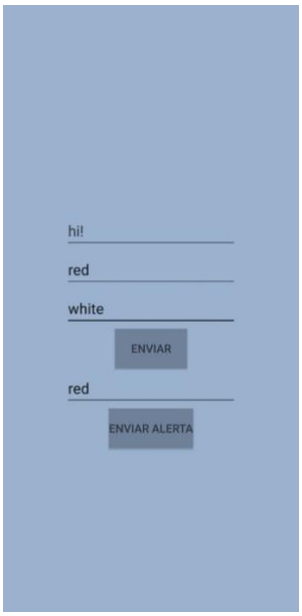

Figura 4-30 Actividad RaspberryP con los campos rellenos

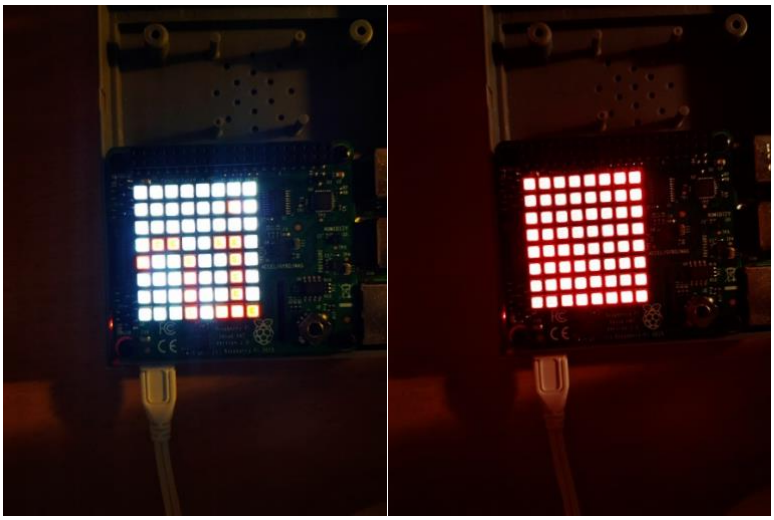

Figura 4-31 Reacción de Raspberry a las instrucciones

Capítulo 5. Pliego de condiciones

Dispositivo	**Precio**	**Lugar de compra**
Raspberry Pi Model 3B+	36,55€	Amazon
SenseHat	40,51€	Amazon
API key Geocoding Google	8,28€/mes	Google

Capítulo 6. Conclusiones y propuesta para trabajo futuro

Este proyecto me ha servido para la visualización de todos los servicios IoT que tenemos a nuestro alcance y lo importante que es en nuestro día a día. Los datos que son monitorizados por cualquier dispositivo, son muy valiosos, ya que el conjunto de todos ellos, establece una red de interconexión de datos que son muy útiles a la hora de la creación de nuevos productos. Con un dispositivo tan barato como es Raspberry Pi se puede conseguir muchos datos de una manera sencilla, por lo que servicios IoT como estos están al alcance de cualquier persona.

Dispositivos como estos no solo sirven para la monitorización de datos dentro de una casa, también sirven para la monitorización de datos en el exterior y establecer un mapa de datos muy importante.

Estos dispositivos se pueden colocar en distintas localizaciones, como por ejemplo en farolas, arboles, parkings e incluso en autobuses.

Esta interconexión de datos es lo que hace que una ciudad se convierta en SmartCity y pueda ofrecer servicios a las personas, adaptándose a ellas.

Capítulo 7. Bibliografía

[1] M. Pla-Castells, J. J. Martinez-Durá, J. J. Samper-Zapater and R. V. Cirilo-Gimeno, "Use of ICT in Smart Cities. A practical case applied to traffic management in the city of Valencia," *2015 Smart Cities Symposium Prague (SCSP)*, Prague, 2015, pp. 1-4.
[2] M. Lekić and G. Gardašević, "IoT sensor integration to Node-RED platform," 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, 2018, pp. 1-5.
[3] Raspberry. http://www.raspberrypi.org/ [Online].
[4] O'Leary, N. and Conway-Jones, D. (2013). Node-RED. https://nodered.org/ [Online]
[5] Manjula, T & Sreenivasulu, U & Hussain, Dr. S.. (2016). "A dynamic raspberry Pi sense HAT multimodality alerting system by using AWS IoT". Indian Journal of Science and Technology. 9.
[6] Coursera. https://www.coursera.org [Online]
[7]Blog de Sofia2 IoT Platform. https://about.sofia2.com [Online].
[8] S. Kanoje, V. Powar and D. Mukhopadhyay, "Using MongoDB for social networking website deciphering the pros and cons," *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, 2015, pp. 1-3.
[9] MongoDB Documentation. https://docs.mongodb.com/manual/ [Online]
[10] L. Barreto, A. Celesti, M. Villari, M. Fazio and A. Puliafito, "Identity management in IoT Clouds: A FIWARE case of study," *2015 IEEE Conference on Communications and Network Security (CNS)*, Florence, 2015, pp. 680-684.
[11] Fiware-Orion, https://fiware-orion.readthedocs.io/en/master/ [Online]
[12] Android Developers, https://developer.android.com/studio/intro/ [Online]