ÍNDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

Memoria

1. Introducción

1.1. Motivación

El interés por conocer la situación de procesos e instalaciones en tiempo real es algo actualmente en auge con el incremento de plataformas y servicios relacionados con el denominado IoT (Internet of Things).

El IoT provocará una transformación en las empresas y traerá aparejados grandes cambios en la forma en la que los individuos y las sociedades entienden cómo la tecnología y sus aplicaciones se relacionan con el mundo.

Diversos estudios han mostrado que el mercado del IoT crecerá exponencialmente en los próximos años (figura 1). En particular, dos compañías tecnológicas (Cisco y Ericsson) han publicado datos acerca del potencial asociado a los dispositivos IoT; ambas han pronosticado en torno a 50.000 millones de dispositivos conectados para el año 2020. Además, empresas de investigación como Gartner, IHS Global Insight, ABI Research o la empresa especializada en IoT Habor Research han elaborado sus propias estimaciones. Todas ellas coinciden en el elevado potencial de crecimiento de este mercado, en el que se perfilan las siguientes tendencias:

- Un crecimiento anual estimado en el número de dispositivos conectados comprendido entre el 14% y el 29%.
- Una media de hasta seis dispositivos (no sólo smartphones) por cada habitante del planeta en 2020.
- Un incremento destacado en los ingresos generados por IoT durante los próximos años.

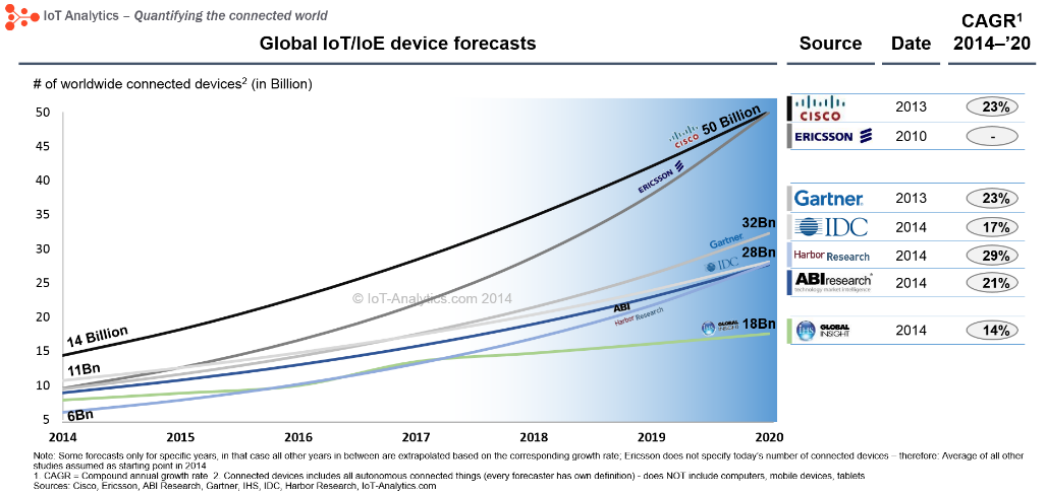

Figura 1 Previsiones de crecimiento en el número de dispositivos IoT.

Los expertos también destacan la gran variedad de campos en los cuales se va a generalizar el uso de la tecnología IoT. Como se muestra en la figura 2, pueden distinguirse dos grandes ramas:

- Los sistemas enfocados al consumidor, con un 45% de los dispositivos, dentro de los cuales los dedicados a la salud supondrán entre un 5% y un 15% del total.
- Los orientados al mundo empresarial, con el 55% de los dispositivos, entre los que destacan los relacionados con los procesos productivos.

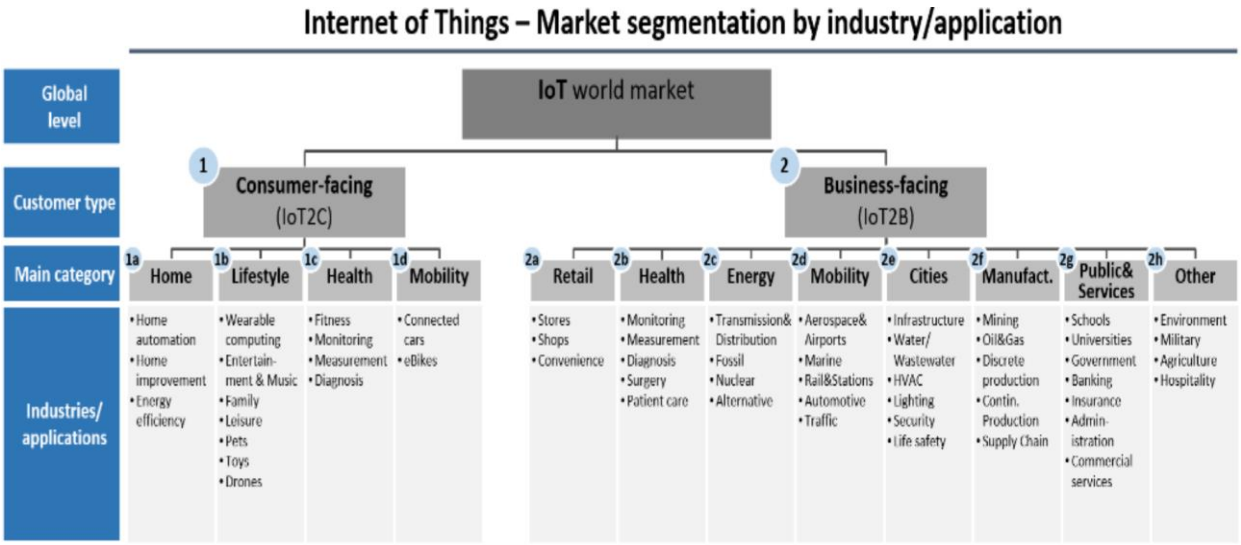

Figura 2 Clasificación por sectores de las áreas con mayor potencial IoT.

En el ámbito del IoT merece una mención especial, por su relevancia creciente, el concepto de "*Smartcity*". La aplicación de la tecnología IoT a todos los elementos que componen una ciudad, desde la iluminación de las calles hasta el riego de los jardines, pasando por el control de la recogida de residuos o la regulación del tráfico y los aparcamientos, va a suponer un cambio trascendental en la forma en la que los ciudadanos se relacionan con la ciudad y reciben sus servicios.

Esta gran proyección del IoT puede suponer una gran oportunidad laboral para los Ingenieros Electrónicos que se sepan adaptar a los nuevos sistemas y a la nueva forma de concebir el mundo.

1.2. Antecedentes

El objeto de este trabajo nace de la curiosidad que siempre he tenido por los dispositivos electrónicos microprogramables de pequeño tamaño, debido especialmente a la enorme versatilidad que ofrecen para la realización de tareas muy diversas. En este contexto, la irrupción de los nuevos dispositivos orientados al IoT contribuyó a acrecentar ese interés por su conocimiento y utilización, de modo que la necesidad de seleccionar un tema sobre el que desarrollar el TFG constituía una oportunidad idónea para ello.

De forma paralela, al iniciar mi actividad profesional en el mundo laboral tuve que acometer el desarrollo de un dispositivo destinado al control de la calidad de un proceso, lo cual contribuyó a consolidar mi interés por los dispositivos IoT. Durante la investigación previa que realicé para definir el modo de llevar a cabo ese proyecto, pude constatar que la gran mayoría de los sistemas comerciales presentaba un coste muy elevado en relación con la funcionalidad que ofrecía.

Dado que ello dejaba abierta la posibilidad de ofertar nuevos productos en condiciones más ventajosas que las de mercado, y podía constituir, de hecho, una oportunidad de negocio, tomé una doble decisión. Por una parte, para dar solución a los requerimientos del cliente, desarrollaría un equipo nuevo a medida, adaptado a las necesidades del proceso a controlar. Por otra parte, como objeto del TFG, me plantearía la creación de una solución más global, que pudiera ser fácilmente aplicable a proyectos diversos.

De este modo fue como surgió la idea de desarrollar una solución para la sensorización de un lugar o proceso, bien sea en el ámbito agrario, industrial o residencial, de una forma económica y sencilla de implementar.

1.3. Objeto del trabajo

El objeto general de este TFG es el diseño y configuración de una plataforma para el desarrollo de redes de sensores de bajo coste y fácil configuración, mediante la utilización de hardware libre, así como su control desde un dispositivo inteligente e inalámbrico, como por ejemplo un teléfono móvil o tablet.

Con el propósito de poder supervisar una gran cantidad de elementos, incluso aunque no se encuentren físicamente interconectados entre sí, pueden emplearse nodos situados en los diversos lugares de interés, dotados de comunicación inalámbrica entre sí. Los sistemas de este tipo pueden variar en cuanto al formato de la comunicación o la frecuencia en la que trabajan. El objeto de este TFG comprende la selección del dispositivo empleado para la transmisión y recepción de información, junto con la configuración de cada uno de los nodos, así como del protocolo de comunicación empleado.

Asimismo, para poder hacer uso de los datos generados por los sensores se pueden emplear diferentes sistemas: conexión directa a un ordenador, almacenamiento en la nube, etc. La solución apropiada dependerá tanto de la ubicación del sistema como de los elementos disponibles para el conexionado entre la red de sensores y el medio de visualización, como, por ejemplo, bluetooth, wifi, GSM, SIGFOX, o cualquier otro sistema de comunicación o de conexión a red. La correcta programación del sistema empleado para la visualización de los datos, junto con el formato de la comunicación, es también objeto de este TFG. Concretamente, se ha contemplado como objetivo la utilización de la reciente tecnología blockchain para el almacenamiento y recuperación de la información generada, abriendo nuevas posibilidades, como pueden ser la venta de los datos o la garantía de su inalterabilidad.

1.4. Objetivos

El objetivo general del TFG es el desarrollo de una plataforma para la creación de redes de sensores de bajo coste y de fácil instalación, de tal manera que cualquier persona sin un gran conocimiento sobre electrónica, y sin nociones sobre desarrollo de aplicaciones para dispositivos móviles, pueda crear la red y usar la aplicación para supervisarla (siempre y cuando se respete el protocolo de comunicación). Para ello, se establecen dos objetivos principales.

El primero de ellos consiste en la selección de los diferentes elementos que conforman cada uno de los nodos, junto con su configuración. Todo ello debe obedecer a las siguientes premisas:

- Debe de ser un sistema fácil de adaptar a las necesidades del usuario, es decir, añadir un nuevo nodo a la red o un nuevo sensor al nodo no debe suponer una gran inversión de tiempo.
- Debe de ser un sistema de bajo coste.

Durante el desarrollo de esta primera parte se creará el formato de la red y se diferenciarán tres tipos de nodos:

- Nodo principal: aquél al que se conectará el dispositivo inteligente, de forma inalámbrica, para visualizar los datos.
- Nodos intermedios: cada uno de ellos tendrá interconexión inalámbrica con otros dos nodos.
- Nodo final: último nodo de la red, conectado únicamente a otro nodo.

El segundo objetivo del trabajo consistirá en realizar una aplicación para el sistema operativo Android, desde la cual se visualizará la información generada por los sensores, que será recibida desde el nodo principal mediante bluetooth. El desarrollo de esta aplicación parte de una única premisa: debe ser válida para cualquier número de nodos y para cualquier número de sensores en cada nodo, de modo que una variación en la cantidad de cualquiera de ellos no debe requerir una modificación de la aplicación.

Como objetivo adicional, que fue contemplado durante la etapa final de desarrollo del proyecto, se decidió incorporar un nuevo elemento de carácter innovador, como es el almacenamiento de datos en blockchain, para aportar una nueva dimensión al almacenamiento de los datos.

Con el objeto de que el presente documento resulte más visual, y pueda ser entendido y seguido de una forma más clara, se ilustrará mediante el uso de una red de muestra, formada por tres nodos, controlada desde un móvil con sistema operativo Android. Se considera que dicha red resulta suficientemente representativa para permitir la comprensión del funcionamiento y del formato de la comunicación de la red, ya que consta de un nodo principal, uno intermedio y uno final. Sin embargo, ello no significa que el número máximo de nodos a emplear en el sistema sea de tres, ya que, como se ha mencionado con anterioridad,

no se contempla un límite teórico en el número de nodos del sistema. Desde el punto de vista práctico, el límite quedaría marcado por la capacidad para procesar toda la información intercambiada, tanto por parte de los microcontroladores de los distintos nodos como por el dispositivo móvil.

2. Referencias

Las siguientes fuentes han sido empleadas durante la fase de desarrollo del sistema electrónico:

- http://forum.arduino.cc/index.php?topic=421081
- https://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo
- https://en.wikipedia.org/wiki/Wireless_sensor_network
- ftp://imall.iteadstudio.com/Modules/IM120606002_nRF24L01_module/DS_nRF24L01.pdf

En la etapa de desarrollo de la aplicación, se han consultado los siguientes recursos:

- https://developer.android.com/index.html
- https://stackoverflow.com/questions/19961457/how-to-share-a-socket-class-instance-between-activities-in-android
- https://stackoverflow.com/questions/15025852/how-to-move-bluetooth-activity-into-a-service
- https://stackoverflow.com/questions/151777/saving-android-activity-state-using-save-instance-state?rq=1
- https://danielggarcia.wordpress.com/2013/10/23/bluetooth-iii-el-esquema-cliente-servidor/
- https://stackoverflow.com/questions/7066657/android-how-to-dynamically-change-menu-item-text-outside-of-onoptionsitemssele
- https://stackoverflow.com/questions/7133141/android-changing-option-menu-items-programmatically
- https://stackoverflow.com/questions/19451715/same-navigation-drawer-in-different-activities
- https://stackoverflow.com/questions/30918026/change-navigationview-items-when-user-is-logged
- http://gpmess.com/blog/2013/09/11/viewholders-en-android/
- https://stackoverflow.com/questions/7279501/programmatically-adding-tablerow-to-tablelayout-not-working
- https://developer.android.com/guide/topics/security/permissions.html?hl=es-419#permissions
- https://developer.android.com/training/permissions/requesting.html?hl=es-419

Para el desarrollo de la parte relacionada con blockchain:

- https://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica

3. Protocolo de comunicación

Con el objetivo de poder conectar los diferentes elementos que forman el sistema de un modo sencillo, se ha desarrollado un protocolo de comunicación propio, gracias al cual se pueden controlar todos los parámetros intercambiados entre los distintos elementos, así como el formato y el tiempo de actualización de la información en el muestreo de los datos.

3.1. Comunicación entre el nodo principal y un dispositivo móvil

En la comunicación con el dispositivo móvil, una vez ha sido establecida la conexión, se procede a realizar la solicitud de información a la red. Esta solicitud siempre tiene el mismo formato:

- Numero de nodo + carácter de finalización; por ejemplo, si se selecciona el nodo 1, la trama que se enviará será 1F.

De este modo, la red sabrá que debe enviar su respuesta, que será diferente según si se trata del primer envío, o si, por el contrario, ya se ha recibido algún valor leído de los sensores.

Si se trata del primer envío, es decir, cuando no se ha recibido nada del sistema electrónico con anterioridad, éste responderá con una cadena informativa acerca de la estructura de la red, que tendrá el siguiente formato:

- El carácter de inicio (“%%”), que sirve para que la aplicación pueda diferenciar si se trata de una cadena que contiene los valores leídos de los sensores, o si contiene otro tipo de información (en este caso, los parámetros relacionados con la denominación de la red).
- A continuación del carácter de inicio se especifica el número de nodos en la red, y seguidamente un elemento separador (“/”).
- Después de este carácter, se incluye el nombre asignado a la red, y seguido, nuevamente, otro elemento separador (“%”).
- Para finalizar la trama, se añaden los nombres de cada uno de los nodos que forman la red, separados entre sí mediante el mismo carácter empleado anteriormente (“%”).

A continuación se muestra el formato de la trama para la red ejemplo, donde “NRed” es la variable en la que se almacena la trama a enviar:

```
NRed="%%" + Numero_Nodos + "/" + Nombre_Red + "%" +
Nombre_Nodo_0 + "%" + Nombre_Nodo_1 + "%" + Nombre_Nodo_2 +"%"
```

Si, por el contrario, se trata de la cadena que contiene los valores leídos de los sensores, su formato será el siguiente:

- En primer lugar, se sitúa el nombre identificativo del sensor. Es el valor que queremos que se muestre en el dispositivo móvil, para poder distinguir de qué sensor se trata. Tras este valor se coloca un carácter separador ("|").
- A continuación de este carácter se añade la lectura del valor medido por el sensor. Seguidamente se coloca otro carácter separador (en este caso, "/").
- Por último, se añade el número identificativo de las unidades de la magnitud medida. Para que la aplicación muestre las unidades de la medida, se le debe enviar su código identificador, según se muestra en la tabla 1. Para terminar se añade otro carácter final ("!").

Tabla 1 Código Identificador/Unidades

Valor recibido	Unidades
0	ºC
1	m
2	cm
3	bar
4	mL
5	Lm (lumen)
6	RH%
7	V
8	mV
9	A
10	mA
11	True (1)
12	False (0)
13	m/s
14	g
15	Kg

El formato explicado corresponde a una trama para un único sensor; si se envía la información de varios sensores, la trama estaría compuesta por ese mismo esquema concatenado repetidamente:

```
ID_Sensor1 + "|" + Valor_Sensor1 + "/" + Unidades_Sensor1 + "!"+
ID_Sensor2 + "|" + Valor_Sensor2 + "/" + Unidades_Sensor2 + "!"
```

3.2. Comunicación entre nodos

Las tramas enviadas por los nodos difieren para cada tipo de nodo, que puede ser el nodo principal, uno de los nodos intermedios, o el nodo final.

- En el caso del nodo principal, como ya se ha explicado, primero se recibe la trama que indica el nodo seleccionado. A continuación, dependiendo del valor recibido, o bien se responde con la trama de la medida, si el nodo seleccionado

era el propio nodo principal, o bien se envía un nuevo valor de activación al nodo siguiente. Este valor responde al siguiente patrón:

Tabla 2 Valor de activación correspondiente a cada nodo

Nodo Seleccionado	Valor activación enviado
Nodo 1	**10**
Nodo 2	**20**
…	**…**
Nodo X	**X*10**

Una vez enviado este valor, se pausa el sistema durante 100 ms para evitar posibles interferencias, y pasado este tiempo el sistema se pone en modo escucha, esperando la respuesta del nodo siguiente. Dicha respuesta, que estará contenida dentro de un array de cadenas de caracteres en el nodo emisor, se almacenará en otro array del mismo tipo en el nodo receptor, array que se irá sobrescribiendo con cada nueva recepción de datos, siempre respetando el siguiente esquema:

```
//Sensor 1                          //Sensor 1
EnviarNl[0]=Numero_Sensorl          EnviarNl[0]=l
EnviarNl[l]=Valor_Sensorl           EnviarNl[1]=10.1
EnviarN1[2]=ID_Unidadesl            EnviarN1[2]=2
//Sensor 2                          //Sensor 2
EnviarN1[3]=Numero_Sensor2          EnviarN1[3]=2
EnviarN1[4]=Valor_Sensor2           EnviarN1[4]=23
EnviarNl[5]=ID_Unidades2            EnviarNl[5]=l
//Sensor 3                          //Sensor 3
EnviarN1[6]=Numero_Sensor3          EnviarN1[6]=3
EnviarN1[7]=Valor_Sensor3           EnviarN1[7]=50.6
EnviarN1[8]=ID_Unidades3            EnviarN1[8]=8
```

- En los nodos intermedios, primeramente se recibe el valor de activación procedente del nodo anterior; cuando dicho valor corresponde al nodo receptor, éste procederá al envío de los diferentes parámetros de los sensores, que, como se ha mencionado anteriormente, estarán contenidos en un “array” de cadenas de caracteres conforme al esquema anterior.
  Si, por el contrario, el valor de activación no corresponde con el del nodo receptor, se redirigirá hacia el nodo siguiente, y después de un pequeño tiempo de espera (50 ms) se pondrá en modo escucha, esperando la respuesta.
- Si se trata del último nodo de la red, responderá automáticamente enviando el conjunto de datos leídos de los sensores.

4. Diseño de la red

El primer condicionante que se tuvo en cuenta a la hora de diseñar la red fue la selección del dispositivo empleado para la emisión y recepción de la información. Después de haber considerado diversas opciones que implicaban el uso de sistemas comerciales como XBee, Sigfox o LoRa, su elevado coste me hizo decantarme por crear mi propio protocolo mediante el empleo de un módulo de bajo coste, como es el nrf24l01.

El principal motivo para la selección de este módulo fue precisamente su coste, que, tal como se muestra en la tabla de precios recogida en el Presupuesto, es muy inferior al de los sistemas comerciales. La segunda razón para su elección fue la disponibilidad del módulo en dos versiones de diferente potencia de transmisión. Existe una versión de corto alcance, unos 20 metros, y otra de largo alcance, con una distancia efectiva teórica de 1100 metros (en condiciones reales, y con una tasa de error baja, unos 250 metros). La existencia de esta segunda versión fue uno de los factores que determinaron la topología de red seleccionada, como se explica a continuación.

Como topología de red, en un primer momento se consideró la posibilidad de creación de una red centralizada, en la que todos los nodos secundarios estuviesen conectados a un único nodo central. El problema que presentaba esta topología, por el cual fue descartada, es que la necesidad de tener que conectar todos los nodos con el central limita la distancia máxima a la que se pueden situar los nodos distribuidos, impidiendo de este modo el despliegue de una red amplia capaz de cubrir grandes distancias.

Teniendo en cuenta este último requisito, se decidió diseñar una red multi-salto (multi-hop), en la cual cada nodo está conectado con el nodo anterior y con el siguiente (en caso de existir). La principal desventaja que presenta esta configuración es que obliga a viajar a la información a través de diversos nodos intermedios hasta alcanzar el nodo principal. Por el contrario, cuenta con la ventaja de permitir una mayor distancia entre el nodo principal y otros nodos de la red, pudiéndose desplegar redes de mayor extensión.

4.1. Nodo principal

En el nodo principal, cuya programación puede consultarse en el Anexo 1, se ubican el módulo bluetooth, destinado a la comunicación con el dispositivo móvil, y el emisor-receptor, para la comunicación con otros nodos, ambos conectados a la placa principal Arduino, según se muestra en la figura 3.

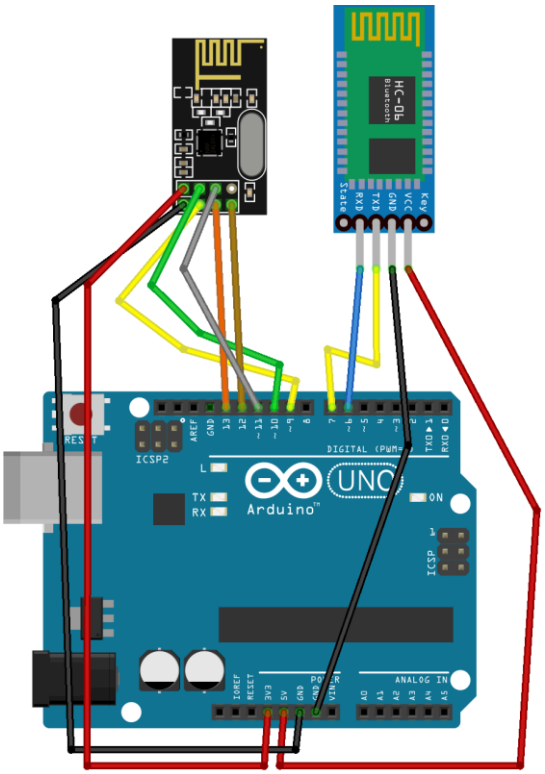

Figura 3 Esquema de interconexión de módulos en el nodo principal.

Para la comunicación bluetooth se seleccionó un HC-06, módulo ampliamente empleado y de coste reducido. Cuenta con 4 patillas: VCC, GND, RX, TX. Las patillas de alimentación, VCC y GND, se conectan a los pines de 5V y tierra, respectivamente, de la placa Arduino, mientras que las otras dos se conectan a los pines 6 y 7, que han sido configurados mediante programación para este fin.

Figura 4 Módulo HC-06 para la comunicación bluetooth.

En el pin 7 de la placa Arduino se conecta directamente la patilla TX del módulo bluetooth, mientras que en la conexión de RX se emplea un divisor de tensión para reducir la

tensión de entrada desde los 5V proporcionados por Arduino hasta los 3.3V de trabajo del módulo. El esquema de conexionado se muestra en la figura 3.

Para la programación de la comunicación bluetooth se emplea la librería "SoftwareSerial", con la que se puede crear un puerto serie simulado asignando los pines de TX y RX que se desee. En este caso se han empleado los pines 6 y 7, como se indicado anteriormente. El funcionamiento es bastante simple. Una vez definido el sistema, mediante la indicación de un nombre y los pines deseados, se inicializa la velocidad de transmisión (baudrate), en este caso 9600, en la función *setup()*. Una vez hecho esto, ya se pueden emplear dos funciones: *read()* y *write()*. La primera lee la información recibida a través de bluetooth, que llega en formato ASCII, mientras que la segunda permite enviar la información deseada, también vía bluetooth.

En lo referente al módulo para la comunicación con otros nodos, nrf24l01, la conexión es algo más compleja. Como se muestra en la figura 5, cuenta con 8 pines.

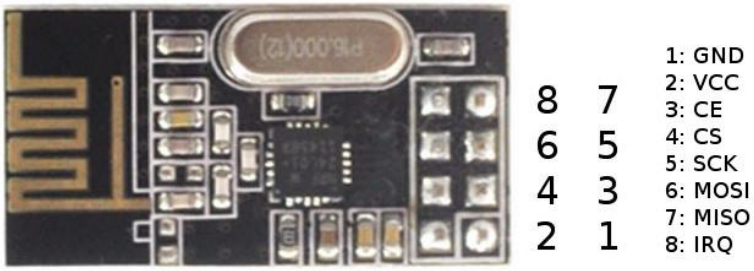

Figura 5 Módulo nrf24l01 para la comunicación con otros nodos.

Los terminales de alimentación, VCC y GND, se conectan a las salidas correspondientes de la placa Arduino, teniendo en cuenta que, puesto que la tensión de trabajo del módulo es de 3.3V, el pin VCC va conectado con la salida de 3.3V de Arduino.

De los otros 6 pines se emplean 5, quedando sin uso el número 8, que corresponde a IRQ (Interrupt Request). Los pines SCK (Serial Clock), MOSI (Master Out Slave In) y MISO (Master In Slave Out) necesariamente deben estar conectados a las patillas 13, 11 y 12, respectivamente, de la placa Arduino, dado que corresponden a los pines de los mismos nombres y funciones que posee el microprocesador, como se puede ver en la figura 6. Sin embargo, los 2 pines restantes, CE (Chip Enable) y CS (Chip Select), pueden conectarse a distintas patillas de la placa Arduino; en este caso el sistema ha sido configurado para su conexión a los terminales 9 y 10, respectivamente.

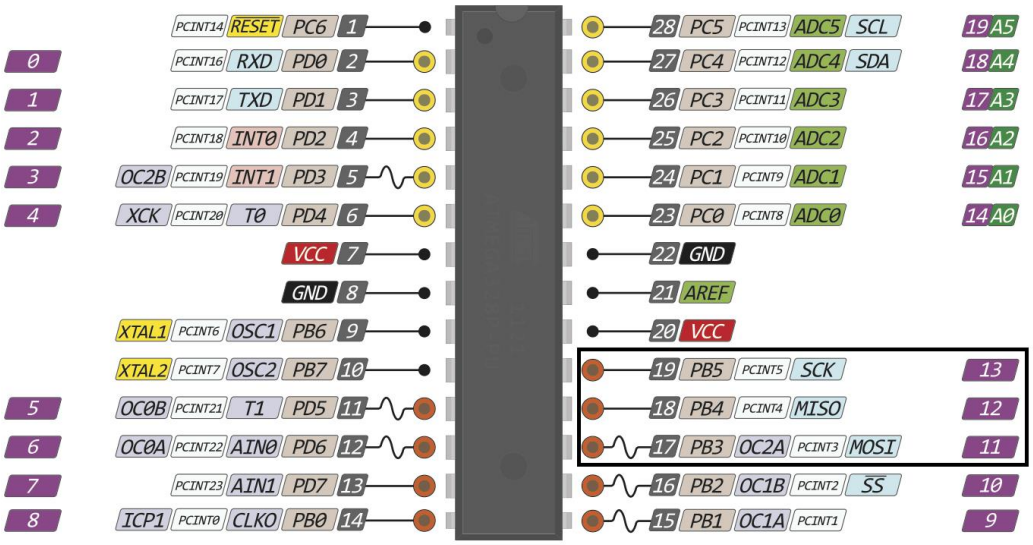

Figura 6 Patillaje del microprocesador de la placa Arduino.

Para el correcto funcionamiento del módulo se emplean las librerías "nRF24L01" y "RF24". Lo primero es asignar los pines a los cuales se conectarán CE y CS, como ya se ha comentado en este caso se han usado el 9 y el 10, junto con un nombre identificativo, para el que se ha empleado "radio". Después se crea una variable para contener la dirección de conexión con el nodo siguiente, que será posteriormente empleada para establecer la conexión para el envío y recepción de datos. A continuación, ya en la función *setup()* de Arduino, se realizan las siguientes operaciones:

- Se ejecuta la función *begin()*, que hace al módulo comenzar a funcionar.
- Se ajusta la potencia de la trasmisión; en este caso, debido a la topología de la red, se configura a máxima potencia, ya que interesa poder situar el siguiente nodo a la mayor distancia posible.
- Se selecciona el canal de transmisión, se ha seleccionado el 100.
- Se abre la conexión con el nodo siguiente a través de la dirección contenida en la variable anteriormente comentada.

Después de la configuración, cuando el sistema ya está funcionando, en la función *loop()* del programa de Arduino se emplean las siguientes funciones contenidas en las librerías del módulo:

- *openWritingPipe(dirección)*: abre una conexión de escritura con la dirección deseada.
- *startListening()*: activa el modo escucha del módulo para recibir los datos del nodo siguiente.
- *available()*: comprueba que existe conexión con el nodo siguiente.
- *read(Values,size)*: lee la información enviada desde el nodo siguiente y la almacena en la variable *Values,* mientras el tamaño del mensaje no supere el

valor de la variable *size*, la cual representa el tamaño máximo permitido para el mensaje.

Además de emplear las funciones propias de cada módulo, en este nodo principal se realizan varias acciones para implementar el protocolo de comunicación propio que ha sido diseñado, y para adecuar los datos, tanto para su envío como para su recepción. Las funciones nuevas que han sido creadas, y que son empleadas en este nodo, todas ellas en la función *loop()*, son:

- ***ComienzoBluetooth()***: esta función se ejecuta la primera vez que se conecta el dispositivo móvil con el nodo principal. En primer lugar se envía la información relativa a la red y posteriormente se reciben los datos enviados por el dispositivo móvil hasta que se detecta el carácter de finalización.

- ***Enviar()***: en esta función se procesa el valor recibido desde el dispositivo móvil a través del módulo bluetooth. Cuando coincide con el identificador del propio nodo principal, realiza una llamada a la función *Convertir()*. Si, por el contrario, el valor recibido corresponde a uno de los nodos siguientes, entonces envía el valor de activación correspondiente, según muestra la tabla 3, al siguiente nodo, usando para ello la función de escritura del módulo nrf24l01. Después de enviar este dato, ejecutará la función *Recibir()*.

Tabla 3 Identificadores de los nodos.

Número de nodo	Valor transmitido vía bluetooth	Valor de activación enviado al nodo
0 (Principal)	--	--
1	49	10
2	50	20
…	…	…
X	ASCII(X)	X*10

- ***Recibir()***: dentro de esta función primero se comprueba si existe un transmisor enviando información y, en caso afirmativo, se usa la función de lectura del nrf24l01 para guardar la información recibida en una variable. Después se inicia la función *Convertir()*, con la cual se adapta la información recibida al formato de envío al dispositivo móvil. Cuando este proceso se completa se realiza el envío por bluetooth usando la función *EnviarBluetooth(String cad)*, siendo *cad* la cadena creada en *Convertir()*.

- ***Convertir()***: En esta función se adapta la información recibida desde el nodo anterior, o la generada por los sensores del propio nodo, al formato del protocolo de comunicación presentado en apartados anteriores. Para ello se hace uso de diferentes bucles *for*, que adaptan la información contenida en la

variable *Recibido[]*, la cual almacena los valores relacionados con las lecturas de los sensores:

- En primer lugar, se sustituye el numero identificativo del sensor por su nombre correspondiente. Estos nombres están almacenados en unas variables declaradas al comienzo del código.
- El siguiente paso es transformar los números en cadenas de caracteres, para poder concatenar todos los valores en una única cadena.
- Por último, se recorre toda la variable concatenando los valores para formar una cadena para cada uno de los sensores (nombre sensor, valor sensor y unidades), de acuerdo con el protocolo de comunicación diseñado, y se añaden los caracteres de separación necesarios entre cada uno de los valores leídos de la variable. Una vez formada esta cadena, se envía empleando la función "EnviarBluetoothFila" y se resetea la cadena para prepararla para el siguiente sensor. Una vez que se ha recorrido todo el array, se envía el carácter de finalización ("#").

- ***EnviarBluetooth(String cad)***: La última función en ejecutarse es la que realiza el envío de la información al dispositivo móvil. En primer lugar convierte la cadena de entrada a un array de caracteres, transformación que resulta necesaria porque la función de escritura propia del módulo bluetooth sólo permite enviar caracteres individuales. Después, y dado que la aplicación del dispositivo móvil nunca llega a recibir el primer carácter enviado, se hace necesario enviar un carácter cualquiera de relleno, por ejemplo "0", seguido de los caracteres contenidos en el array recién creado, y, en último lugar, el carácter de finalización ("#").

- ***EnviarBluetoothFila(String Cad, int contenvio):*** similar a la anterior, compara si se trata del primer valor de la cadena (para ello comprueba el valor de "contenvio"); si es así, envía el primer carácter ("0"), y a continuación la cadena "Cad".

4.2. Nodos intermedios

En los nodos intermedios, cuya programación puede consultarse en el Anexo 2, se ubica únicamente el emisor-receptor para la comunicación con otros nodos, conectado a la placa principal Arduino según se muestra en la figura 7.

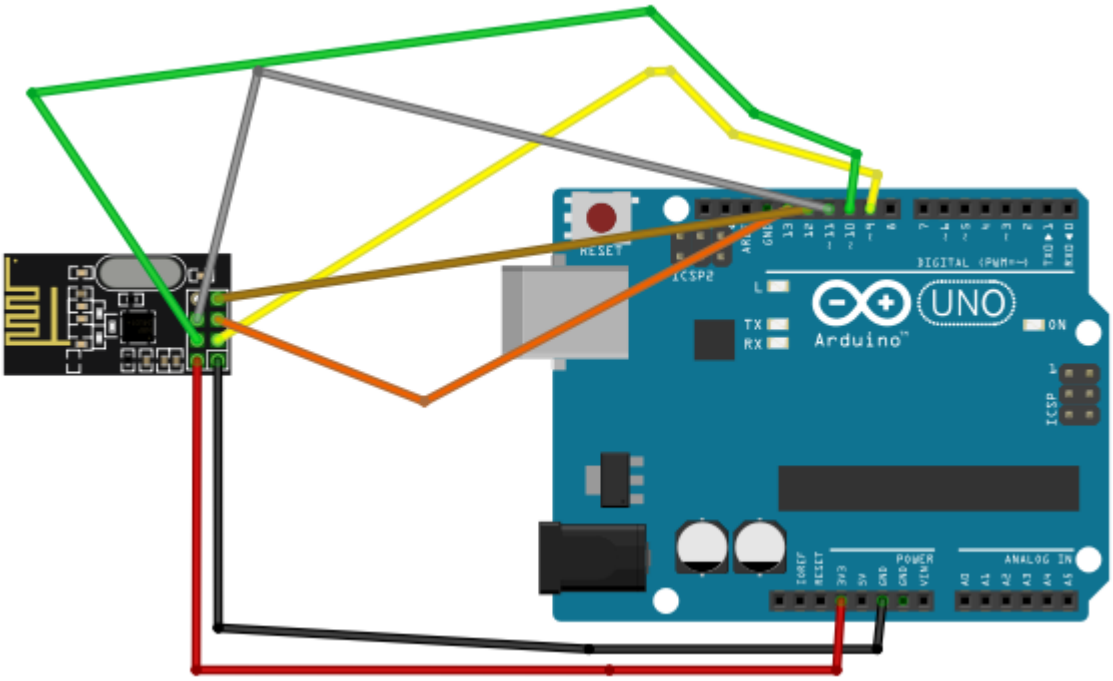

Figura 7 Esquema de interconexión en un nodo intermedio.

Dado que en el caso de los nodos intermedios el sistema empleado es el mismo que en el nodo principal, Arduino UNO, el conexionado del módulo es igual. Si se hubiesen empleado otros sistemas, como Arduino Mega (el cual será utilizado con fines ilustrativos en el nodo final), Arduino Nano, etc., tanto el conexionado del módulo como su programación hubieran sido diferentes.

Al igual que en el nodo principal, para el correcto funcionamiento del módulo se emplean las librerías "nRF24L01" y "RF24". Como los nodos intermedios se conectan con otros dos (el anterior y el siguiente en la red), se han de emplear dos variables distintas para almacenar la dirección de conexión con cada nodo. Después de la declaración de variables, se configura el nodo en la función *setup()* de Arduino, en la que se realizan las siguientes operaciones:

- Primero se ejecuta la función que inicia el módulo emisor-receptor, *begin()*.
- A continuación se configura la potencia de la red y el canal de emisión del mismo modo que en el nodo principal.
- Por último, se abre la comunicación con los nodos anterior y posterior, usando para ello las direcciones previamente declaradas en las variables comentadas con anterioridad, y se pone el sistema en modo escucha.

Cuando el sistema ya se encuentra en régimen de funcionamiento continuo, en la función *loop()* del programa de Arduino se emplean las funciones contenidas en las librerías del módulo ya explicadas en el apartado 5.1.

Los nodos intermedios funcionan del modo que se describe a continuación. Con el sistema en modo escucha, cuando se recibe un valor de activación desde el nodo anterior se almacena en una variable y se comprueba si corresponde al propio nodo intermedio que lo ha recibido, o si, por el contrario, hace referencia a uno de los nodos posteriores, en base a la tabla 2 (tabla del 5.1).

Si el valor de activación corresponde al propio nodo, primero se detiene el modo escucha y se inicia el modo escritura hacia el nodo anterior. Seguidamente se leen los valores de los sensores y se almacenan en una matriz, junto con los parámetros identificadores de cada uno de ellos (el nombre asignado al sensor y las unidades de la medida). Por último, se envían los valores contenidos en la matriz al nodo anterior.

Si, por el contrario, el valor de activación corresponde a un nodo posterior en la red, se detiene el modo escucha y se abre la escritura hacia el nodo siguiente, al que se envía dicho valor de activación recibido del nodo anterior. Si el envío es correcto, se inicia el modo escucha y se queda a la espera de recibir información desde el nodo siguiente. Cuando esto ocurre, se detiene nuevamente el modo escucha y se abre el envío hacia el nodo anterior. Se guardan los valores recibidos en la matriz a enviar mediante un bucle *for*, y a continuación se envían. Finalmente, vuelve a comenzar la función *loop()*.

4.3. Nodo final

Para este nodo se ha empleado, con fines ilustrativos, un modelo de Arduino distinto, Arduino Mega, mostrando de este modo la polivalencia del sistema en cuanto a sus posibilidades de configuración, que permiten adaptarlo a las necesidades de cada aplicación. Este modelo de Arduino, gracias a sus 54 entradas digitales y 16 analógicas, es el más apropiado para nodos que requirieran la monitorización de una gran cantidad de parámetros. Por el contrario, cuando se desea que el nodo sea del menor tamaño posible, es más conveniente emplear un Arduino Nano o Mini.

En el nodo final, cuya programación puede consultarse en el Anexo 3, se ubica únicamente el emisor-receptor para la comunicación con otros nodos, conectado a la placa principal Arduino según se muestra en la figura 8.

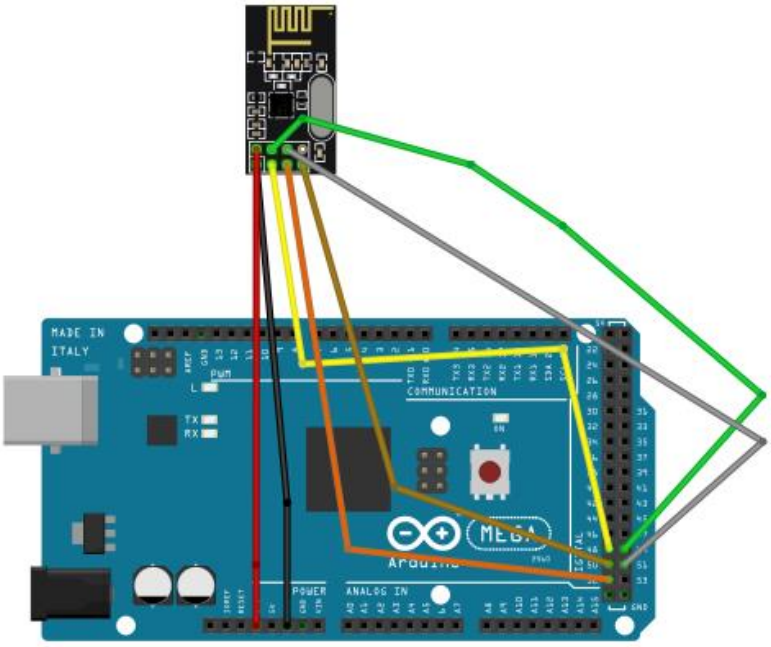

Figura 8 Esquema de interconexión en el nodo final.

Al tratarse de un modelo distinto de Arduino, el conexionado es distinto del correspondiente a los nodos principal e intermedios. Mientras que los pines de alimentación se mantienen, es decir, VCC se conecta a la salida de 3.3 V y GND se conecta al terminal GND de la placa de Arduino, la conexión de los otros 5 pines utilizados en la placa del emisor-receptor cambia (al igual que en los otros nodos, no es necesario utilizar el pin IRQ). Los pines SCK (Serial Clock), MOSI (Master Out Slave In) y MISO (Master In Slave Out) necesariamente deben estar conectados a las patillas 52, 51 y 50, respectivamente, de la placa Arduino, dado que corresponden a los pines de los mismos nombres y funciones que posee el microprocesador, como se puede ver en la figura 9. Los 2 pines restantes, CE (Chip Enable) y CS (Chip Select), pueden conectarse a distintas patillas de la placa Arduino; en este caso el sistema ha sido configurado para su conexión a los terminales 48 y 49, respectivamente.

Figura 9 Patillaje del microprocesador de la placa Arduino.

Al igual que en los otros nodos, para el correcto funcionamiento del módulo se emplean las librerías “nRF24L01” y “RF24”. Como este nodo se conectará únicamente con el anterior, su configuración, así como su funcionamiento, es bastante simple.

En el *setup()* se realiza la misma configuración que en los nodos anteriores. Primero se ejecuta la función que inicia el módulo emisor-receptor, *begin()*, después se configura la potencia de la red y el canal de emisión, y, por último, se abre la comunicación con el nodo anterior y se pone el sistema en modo escucha.

En el *loop()* se espera hasta que se recibe un valor de activación a través del emisor-receptor. Cuando esto ocurre, se comprueba si coincide con el valor de activación del nodo, como es de esperar, en cuyo caso se deja de escuchar y se abre una conexión de escritura con el nodo anterior. Seguidamente se leen los sensores y se añaden los valores leídos a la matriz, junto con los parámetros identificativos (nombre del sensor y unidades de la medida). Hecho esto, se envían los valores al nodo anterior.

5. Diseño de la aplicación para el dispositivo móvil

La aplicación ha sido desarrollada mediante el programa Android Studio, herramienta gratuita desarrollada por Google que cuenta con una extensa documentación. Además, se ha empleado la librería "Butterknife", junto con el plugin "Zelency", que facilita la gestión de los diferentes elementos que componen la interfaz, permitiendo de este modo una mayor velocidad de desarrollo.

El planteamiento inicial consistía en haber implementado una aplicación específica para el ejemplo desarrollado, la cual tuviese en su menú de selección los tres nodos previamente descritos, de modo que se abriese una actividad distinta al seleccionar cada uno de ellos. Dentro de esta actividad se encontrarían los sensores concretos del nodo, tal como hubiesen sido preconfigurados en la aplicación. Sin embargo, ello provocaría tener que actualizar la aplicación cada vez que se añadiese un sensor nuevo en alguno de los nodos.

A medida que fue avanzando el desarrollo del proyecto, se modificó el planteamiento inicial en favor de una aplicación dinámica más versátil, que no tuviese predefinidos ni el número de nodos ni el número de sensores por nodo, de tal manera que se emplease una única actividad para todos los nodos. De este modo, tanto el contenido de la actividad como el número de nodos a mostrar estarían únicamente condicionados por la información recibida desde el nodo principal.

Para mayor claridad a la hora de exponer la aplicación, se desarrolla primero su funcionamiento general, detallando posteriormente cada una de sus actividades, clases, hojas de interfaz, etc. en los siguientes subapartados.

5.1. Funcionamiento general

Al iniciar la aplicación, lo primero que aparece es la actividad principal. En su parte superior se muestra una lista con los dispositivos bluetooth conocidos, y en la parte inferior un botón que permite buscar nuevos dispositivos (figura 10). Cuando se pulsa, comienza la búsqueda y los dispositivos encontrados se van añadiendo a un nuevo campo que aparece en la zona intermedia de la pantalla (figura 11). Una vez seleccionado el dispositivo deseado, se establece la conexión.

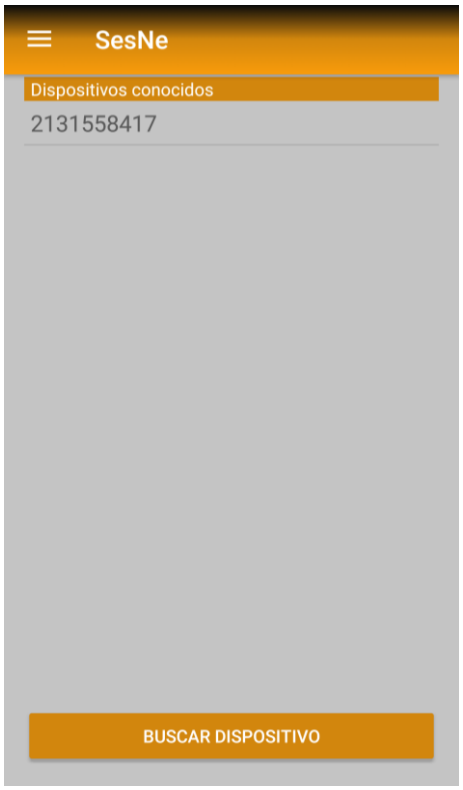

Figura 10 Pantalla inicial.

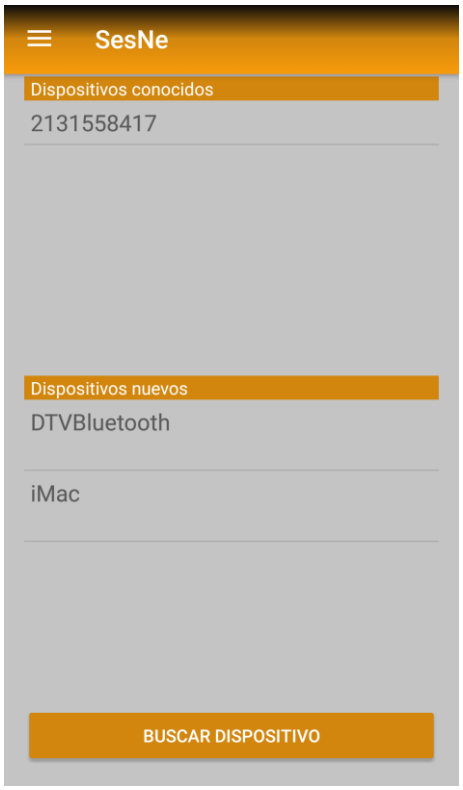

Figura 11 Pantalla inicial con los dispositivos encontrados.

Seguidamente debe abrirse el menú lateral (figura 12), destinado a centralizar el acceso a los distintos nodos de la red, para pulsar el nodo principal, que inicialmente es el único disponible. De este modo, se accede a una actividad vacía con un único botón para recibir datos (figura 13). Cuando se pulsa, se muestra en la parte superior una tabla con la información recibida de los sensores del nodo principal. Además, a partir de los datos relativos a la estructura de la red recibidos del nodo principal, se procede a actualizar su nombre en el menú lateral, así como a añadir el resto de los nodos. A partir de ese momento, ya se puede seleccionar el nodo que se desee.

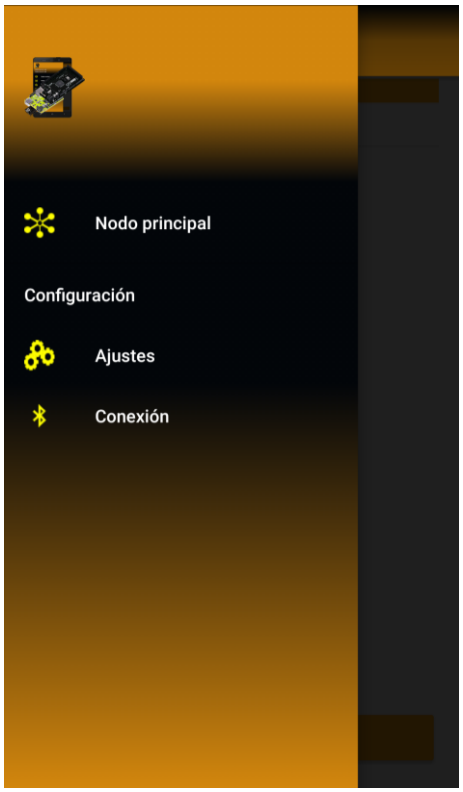

Figura 14 Menú lateral, estado inicial.

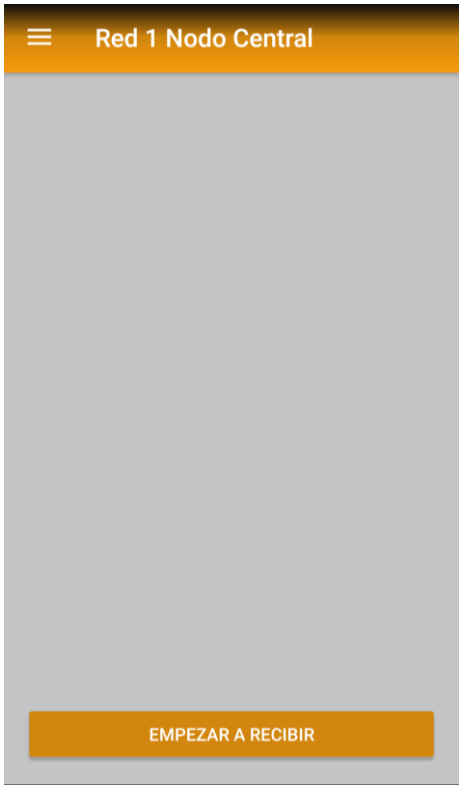

Figura 15 Pantalla nodos sin datos.

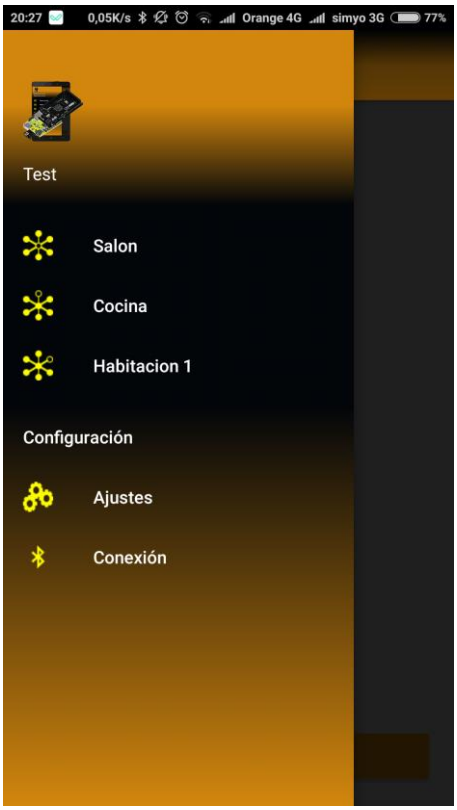

Figura 13 Menu lateral completo

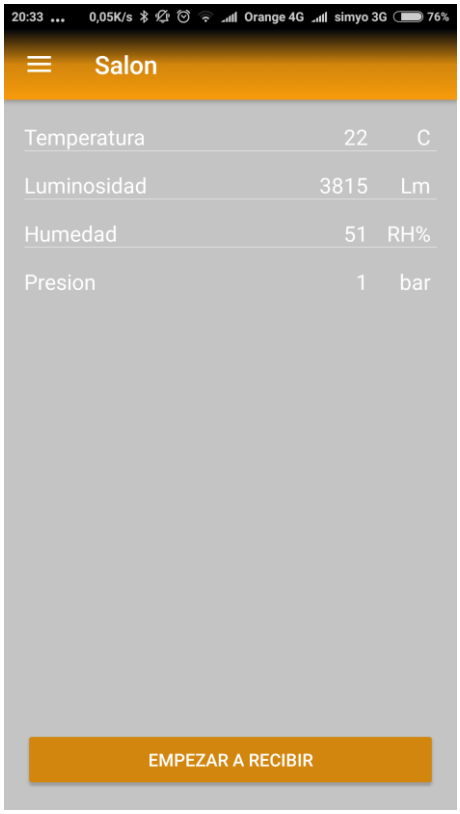

Figura 12 Actividad nodo con datos

5.2. Conexión Bluetooth

La figura 16 muestra un diagrama de flujo que representa el proceso de conexión bluetooth, incluyendo las funciones que en él intervienen, que serán detalladas a lo largo del presente apartado.

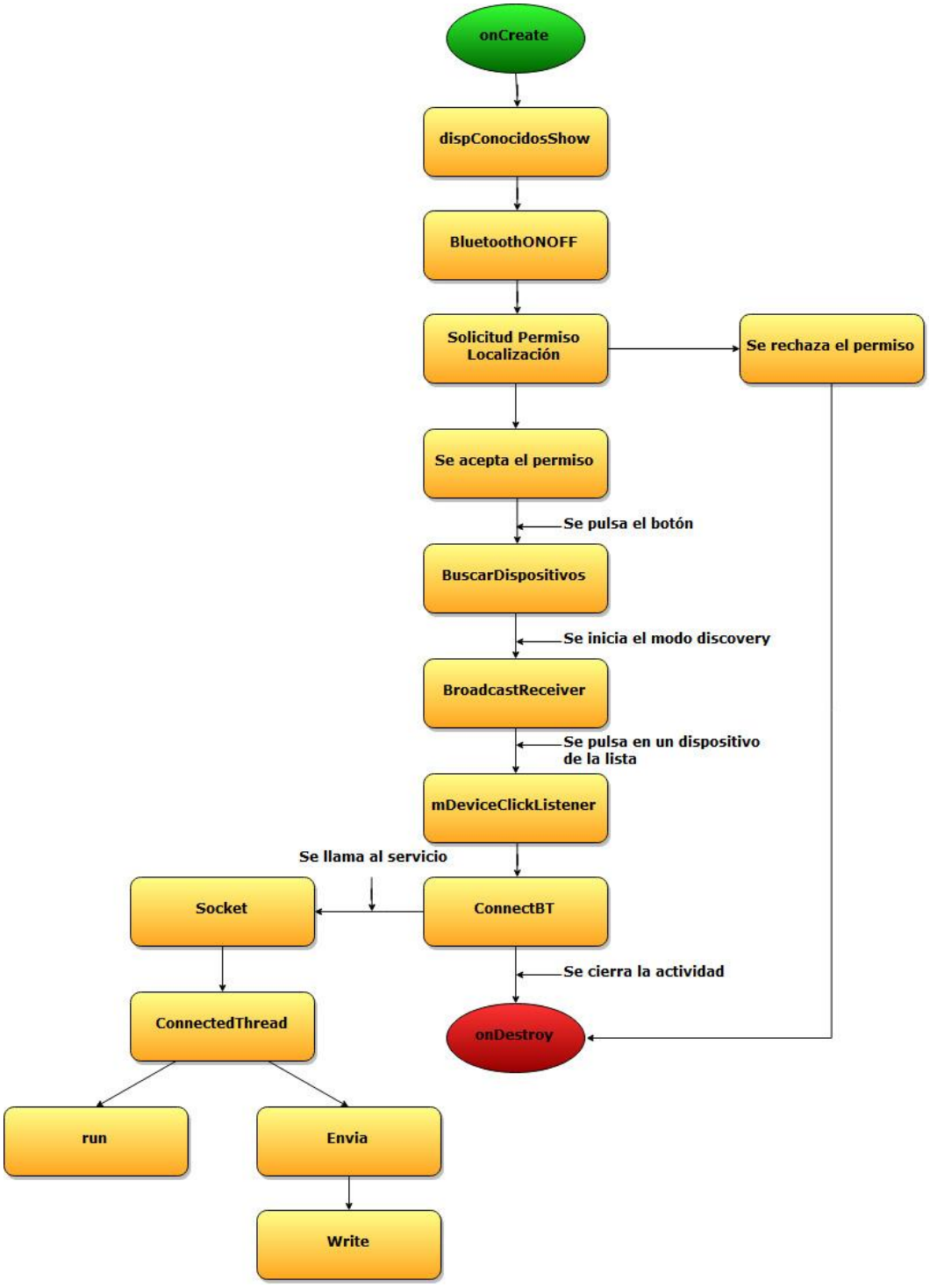

Figura 16 Diagrama de flujo conexión.

La conexión bluetooth se compone de dos partes diferenciadas:

- ***MainActivity* (Anexo 4.1):** muestra los dispositivos disponibles para conectarse.
- ***LocalService* (Anexo 4.2):** establece la conexión.

En primer lugar se describe *MainActivity* que contiene las funciones necesarias para detectar y mostrar los dispositivos bluetooth a los que poder conectarse, las cuales son descritas a continuación.

***OnCreate()*:** engloba las acciones que se llevan a cabo cuando se inicia la actividad. En primer lugar, se llama al archivo de diseño correspondiente ("activity_main"), seguidamente se asigna el adaptador bluetooth a una variable, y, finalmente, se inician los arrays que contendrán los nombres y direcciones MAC de los dispositivos, junto con las clases vista, propias de Java, correspondientes a los diferentes elementos que conforman el interfaz.

A continuación se ejecutan las funciones *dispConocidosShow()* y *BluetoothONOFF()*, las cuales se exponen seguidamente. Además, se registra el transmisor-receptor "mReceiver", que es un elemento propio de Android que se encarga de la recepción y gestión de los intentos de transmisión.

Como medida adicional, debido a una modificación en el sistema de permisos de Android para las versiones posteriores a la 6.0, es necesario contar con el permiso de localización para poder buscar dispositivos bluetooth. Por esta razón, lo último que se realiza en esta función es la comprobación de la versión de Android y, si es pertinente, la solicitud del permiso.

- ***BucarDispositivos()*:** función que se ejecuta cuando se pulsa el botón que se encuentra en la parte inferior de la pantalla. Al pulsar una vez, se inicia la búsqueda de dispositivos (lo cual implica que se ejecuta la función que inicia el modo Discovery) y se cambia el texto del botón. Cuando se pulsa por segunda vez, se detiene la búsqueda y se recupera el texto original en el botón.
- ***dispConocidosShow():*** muestra el array correspondiente a los dispositivos con los cuales ya se ha establecido conexión anteriormente y lo rellena.
- ***BluetoothONOFF():*** comprueba si la funcionalidad bluetooth está activada, y en caso negativo solicita su activación.
- ***onDestroy():*** función propia del ciclo de actividad de Android que se ejecuta cuando se cierra la actividad, provocando la cancelación del modo Discovery y la eliminación del registro del transmisor-receptor "mReceiver", el cual permite la comunicación bluetooth en esta actividad.
- ***BroadcastReceiver():*** función que se registra con el transmisor-receptor "mReceiver". Cuando se ejecuta, hace visible el título "dispNuevosTitulo" y añade al array "dispNuevosArray" el nombre y la dirección MAC de los

dispositivos encontrados. Si no encuentra ningún dispositivo nuevo, añade un mensaje indicativo.

- ***mDeviceClickListener():*** función que se ejecuta al pulsar alguno de los elementos que forman las tablas. Cuando esto ocurre se inicia la función ConnectBT().
- ***ConnectBT():*** Primero ejecuta en segundo plano la llamada a la función Socket() contenida en el servicio (explicado más adelante). Cuando termina de ejecutarse esa función, muestra un mensaje en pantalla indicando el éxito o el fallo del intento de conexión.
- ***Msg(String s):*** muestra el mensaje indicado en "s" como un texto en pantalla.
- ***onStart():*** función propia de Android que se ejecuta cuando se inicia la actividad. En este caso se utiliza para realizar una llamada al servicio y establecer una conexión con él.
- ***ServiceConnection():*** establece la conexión con el servicio.

Conviene destacar que cuando se estable una conexión bluetooth con otro dispositivo solamente se mantiene mientras la actividad permanece activa, tal como refleja la figura 17. Ello supone un problema cuando se desea, como en el caso presente, poder representar conjuntamente la información recibida en distintas actividades. Esta fue una de las dificultades encontradas durante el desarrollo del trabajo que más tiempo costó resolver, dado que todas las alternativas encontradas o bien no eran aplicables a las necesidades del proyecto, o bien no funcionaban como era deseado una vez implementadas.

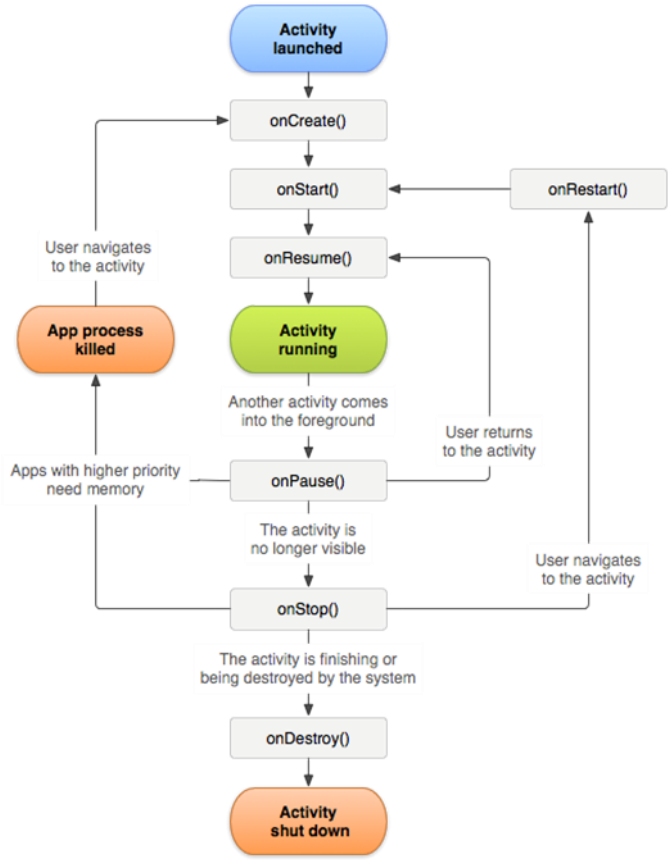

Figura 17 Ciclo de vida actividad Android.

Después de mucho investigar, se encontró la solución al problema mediante la utilización de un *servicio*. Como se describe en la guía de Android, un servicio es un componente de una aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario. Con esta idea se desarrolló el servicio denominado "*LocalService*", que lleva a cabo las funciones que se enumeran en los siguientes párrafos.

Primero se declaran una serie de variables que serán empleadas en las diferentes funciones, entre las que se encuentran:

- Un adaptador bluetooth: *mBtAdapter*.
- Un socket con el cual se establecerá la conexión: *btSocket*.
- Una función de conexión: *mConnectedThread*.
- Un UUID o "Identificador único universal", que sirve para identificar y validar las conexiones entre dispositivos bluetooth. Como en este caso se va a conectar con un dispositivo serie, hay que emplear uno concreto, el 00001101-0000-1000-8000-00805F9B34FB.
- Un string en el cual se almacenarán los valores recibidos por bluetooth: *readMessage*.

Después figuran las funciones empleadas en el servicio.

- ***LocalBinder()*:** crea la función que permite a las actividades acceder a las funciones contenidas en el servicio.
- ***BluetoothSocket Socket (String addrress)*:** en esta función se usa la cadena de entrada que contiene la dirección MAC del dispositivo al que se quiere conectar. Se crea el "socket" con que se realizará la conexión y seguidamente se intenta conectar; si la conexión es correcta se ejecuta la función "*ConnectedThread (btSocket)*", si no lo es se captura la excepción para que no haya problemas de funcionamiento.
- ***Enviar (String string)*:** ejecuta la función "write" con la cadena "string".
- ***ConnectedThread*:** se crean los dos flujos de datos, tanto de entrada como de salida.
- ***Run()*:** recibe los caracteres que llegan vía bluetooth y los almacena en una variable hasta que recibe el carácter final '#', momento en el que guarda la cadena en "readMessage".
- ***write (String input)*:** envía la cadena de entrada a un buffer, desde el que es transmitida vía bluetooth.
- ***getReadMessage()*:** envía el valor contenido dentro de la variable "readMessage" a la actividad que esté siendo ejecutada en el momento.

Cada actividad, además del archivo .java que contiene las funciones expuestas, también cuenta con otro archivo de interfaz en el cual se diseña la forma en la que se representa la información para el usuario.

5.3. Representación de datos

La representación de la información recibida se realiza en la actividad denominada "Nodo". Esta actividad implementa el layout "activity_nodo", el cual incluye a su vez, como se explica más adelante, otro layout denominado "content_nodo", en el que únicamente hay dos elementos, una tabla y un botón. En la tabla se representarán los datos recibidos, mostrando en cada fila los parámetros propios de cada sensor: su nombre, el valor medido y las unidades de la medición. El botón se emplea para iniciar la recepción de datos en la actividad; cuando es pulsado se ejecuta la función "RecibirDatos()", la cual se explica más adelante.

Para explicar el funcionamiento de esta actividad se van a detallar por separado las funciones propias de Android, comunes para todas las actividades, y las creadas específicamente para esta actividad.

Entre las propias de Android se encuentran:

- ***OnCreate()*:** al igual que en la actividad principal, esta función integra las acciones que se llevan a cabo cuando se inicia la actividad. En primer lugar, se

llama al archivo de diseño correspondiente ("activity_nodo"). Seguidamente se almacena (en la variable nodoSel) el valor enviado desde la actividad anterior, el cual indica cuál de los nodos ha sido el seleccionado. A continuación, se genera la cadena a enviar al nodo principal (cadInicial), la cual le indicará qué nodo ha sido el seleccionado. El siguiente paso consiste en generar el menú lateral de selección de nodos, para lo cual es necesario primeramente leer de la memoria del dispositivo los valores almacenados que contienen tanto el nombre de la red como los de los nodos que la forman. Entonces se llama a las funciones "SetNombreRed" y "SetNombreOpcionMenu", contenidas en la actividad base, para asignar los valores leídos de la memoria a los elementos correspondientes del menú. Finalmente se ejecuta un bucle *for*, desde el que se llama nuevamente a la función "SetNombreOpcionMenu", junto con la función "showOpcionMenu", para mostrar el resto de los nodos de la red con sus respectivos nombres.

- ***onStart():*** función propia de Android que se ejecuta cuando se inicia la actividad. En este caso, al igual que en la actividad principal, se utiliza para realizar una llamada al servicio y establecer una conexión con él.
- ***ServiceConnection():*** establece la conexión con el servicio.

Por su parte, las funciones desarrolladas para esta actividad son:

- ***RecibirDatos():*** en esta función se realiza el envío de los datos al nodo principal, empleando para ello las funciones contenidas en el servicio "LocalService". Una vez se ha enviado la cadena de inicio (contenida en "cadInicial") al nodo, se emplea una clase de Java denominada "Handler" para enviar y procesar mensajes. Junto a ella también se utiliza la función "postDelayed" para poder ejecutar una acción transcurrido un tiempo. En este caso se ejecuta un interface denominado mAction de tipo "Runnable", esto es, una función que se ejecuta en un hilo de ejecución.
- ***mAction:*** contiene a la función "run()", en la que se reciben los datos desde el nodo principal (en la variable "recib"), y se adaptan eliminando caracteres innecesarios. Una vez hecho esto, se llama a la función "separar()", la cual se explica a continuación. Por último, se vuelve a llamar al handler y se ejecuta nuevamente el runnable mAction, con lo que se consigue que la recepción de los datos se produzca de forma cíclica y continuada durante toda la ejecución de la actividad.
- ***separar():*** en esta función se adapta la cadena de datos recibidos al formato deseado; para ello se emplea un bucle *for* que recorre la cadena concatenando los diferentes caracteres hasta que encuentra los caracteres limitadores. Se distinguen dos situaciones:
    - La primera tiene lugar cuando se recibe un "%" como carácter inicial de la cadena, lo cual indica la recepción del número de nodos que forman

la red junto con sus nombres. Esta cadena se divide en tres elementos: el número de nodos que forman la red, el nombre de la red y el nombre de los nodos. En primer lugar, estos valores se almacenan en el dispositivo, empleando para ello "SharedPreferences", esto es, un sistema clave-valor para el almacenamiento de datos. Posteriormente se representan en los elementos correspondientes del menú, haciendo uso de las funciones contenidas en la actividad base.

- La segunda posibilidad que se contempla se produce cuando se reciben datos de los sensores, en cuyo caso se separa la cadena en tres elementos: nombre del sensor, valor medido y valor indicador de las unidades. Una vez separados los elementos, se llama a la función "AddTabla", especificando como parámetros de entrada el nombre del sensor y el valor medido, tal y como han sido recibidos, junto con las unidades apropiadas, determinadas por medio de su valor identificativo conforme a la correspondencia recogida en la tabla 1. El reemplazo de las unidades se realiza mediante una llamada a la función Unidades, en la que, dependiendo del valor de la entrada, se devuelven las unidades correspondientes

  Una vez ha finalizado la ejecución de la función "AddTabla", se comprueba si está habilitado el almacenamiento de datos en blockchain; si es así, se hace una llamada a la función "Api", detallada en el apartado 7.3 (Blockchain, Aplicación Android).

Además, cada vez que se recibe una nueva cadena se borra el contenido de la tabla que muestra la información. Dado que esta tabla se define en la aplicación, si se quisiese añadir más unidades bastaría con publicar una actualización de la aplicación en la que se incluyesen.

- ***AddTabla(String Unidades, String Nombre, String Valor):*** en esta función se generan las filas con los parámetros de entrada y se añaden a la tabla.

5.4. Actividad base

Cada una de las actividades anteriores (el servicio no) supone una extensión de una actividad jerárquicamente superior denominada *actividad base*, nombrada como "BaseActivity" en este caso. De este modo, es posible utilizar todas las funciones que contiene, lo cual permite reducir la extensión del código en cada una de las actividades que dependen de ella.

Dentro de esta actividad base, se ubican los elementos que se desea que estén disponibles para todas las actividades, que son todos los relacionados con el menú lateral (figura 18) y la barra superior (figura 19), así como la función para el botón con forma de flecha (figura 20) disponible en el sistema operativo Android.

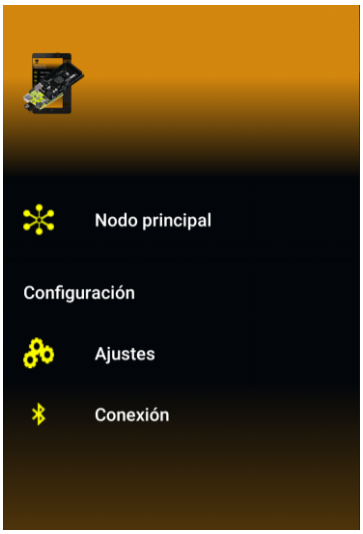

Figura 18 Menú lateral.

Figura 19 Barra superior.

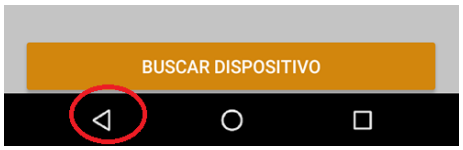

Figura 20 Botón flecha.

Más detalladamente se encuentran los siguientes elementos:

Una actividad:

- **AppCombarActivity**: actividad propia de AndroidStudio que permite añadir la barra superior a la aplicación. Además, se implementa otro elemento denominado "NavigationVie…" el cual nos permite añadir el menú lateral.

Se declaran una serie de variables:

- **menú:** variable de tipo Menu que se emplea cuando se realizan modificaciones sobre el menú lateral.
- **NodoSel:** entero que recoge el número del nodo seleccionado cuando se pulsa en un elemento del menú lateral.
- **EXTRA_MESSAGE:** cadena empleada cuando se pulsa un elemento del menú y se inicia una nueva actividad. Contiene un identificador que relaciona el valor enviado desde la actividad origen (en este caso el valor de NodoSel) con el valor que se recibirá en la nueva actividad.

Se emplean las siguientes funciones:

- ***onStart()*:** crea los elementos que constituyen tanto el menú como la barra superior: una variable "toolbar", que corresponde a la barra superior, y es inicializada mediante la función "setSupportActionBar", un "drawerLayout" que permite "localizar" los diferentes elementos que se mostrarán en la

pantalla, un "ActionBarDrawerToggle" que posibilita una interacción fácil y cómoda con el menú lateral, y un "navigationView" en el cual se encuentra el menú.

- ***onBackPressed()*:** atribuye acciones a la flecha del sistema operativo. Tiene dos funcionalidades, dependiendo de la situación: si el menú lateral está abierto, lo cierra, y si está cerrado, abre la actividad principal.
- ***onNavigationItemSelected(MenuItem ítem)*:** detecta qué elemento del menú se ha seleccionado y lo almacena en la variable "id", después lo compara con los nombres de los elementos del menú, y si alguno coincide se abre una nueva actividad. Si el elemento seleccionado es un nodo, se abre la actividad "nodo" y se envía el valor de "NodoSel" para que la nueva actividad detecte cuál es el nodo pulsado.
  Cabe señalar que, dado que este documento presenta una aplicación ejemplo, en esta función se muestra únicamente un máximo de hasta seis nodos posibles. Si se quisiese aumentar el número de nodos, simplemente habría que copiar y pegar las funciones "if", así como añadir los elementos id correspondientes en el archivo de tipo menú (explicado en la sección "otros archivos"), pudiéndose añadir de este modo en el menú todos los elementos que se desee, sin un límite teórico.
  ***SetNombreRed (String NombreRed)*:** cambia el valor del elemento del menú "Nombre_Red" por el valor de la cadena "NombreRed".
- ***SetNombreOpcionMenu(int id, String string)*:** indica en la variable "id" el valor del elemento del menú que se desea cambiar y lo sustituye por el valor de la variable "string".
- ***hideOpcionMenu(int id)*:** oculta el elemento del menú indicado en la variable.
- ***showOpcionMenu(int id)*:** muestra, si está oculto, el elemento del menú indicado en el "id".
- ***deleteOpcionMenu(int id)*:** elimina el elemento indicado del menú.

5.5. Otros archivos

Al crear la aplicación se genera un elevado número de archivos, aunque muchos de ellos son simplemente declaraciones de ciertos elementos, o no son trascendentes para su funcionamiento. En esta sección se describen algunos de ellos.

- **Archivos de interfaz (layout):** definen los diferentes elementos que se muestran en la pantalla. Para facilitar el diseño se dividen en dos tipos:
  - Activity: en este archivo se declaran los elementos que forman parte del interfaz pero que no son exclusivos de la actividad, como son la barra superior y el menú lateral, los cuales se muestran en todas las actividades. Además, se incluye el layout siguiente: Content.

  - Content: dentro de él se crean los elementos que serán parte principal de la actividad. Por ejemplo, "content_main" responde al tipo "LinearLayout", que muestra todos los elementos verticalmente uno a continuación de otro. En cambio, "content_nodo" está configurado como "RelativeLayout", que hace que la posición de cada elemento quede condicionada por la de otro.

- **Archivos de menú:** se indican los diferentes elementos que formaran el menú. En este caso se dividen en dos grupos:
  - Los nodos disponibles: en el ejemplo presentado únicamente se han incluido seis nodos en el menú, pero este número se podría aumentar sin un límite teórico sin más que copiar y pegar un "ítem" de los anteriores y sustituir el id y el título.
    Además, todos los nodos a excepción del central están configurados para arrancar en modo oculto, lo cual significa que cuando se inicie la aplicación únicamente se mostrará el nodo principal en el menú, de modo que los demás no se harán visibles hasta que no se hayan recibido los datos vía bluetooth.
  - Los elementos de configuración: aquí se encuentran dos elementos, "conexión", que permite conectar con la actividad principal cuando es pulsado, y la parte de "Ajustes" donde se pueden configurar diferentes elementos relacionados con la parte de blockchain, explicada mas adelante.
- **Archivos de cadenas:** con el fin de fomentar buenas prácticas de programación en el desarrollo de la aplicación, cuando se crea el proyecto se genera un archivo denominado "strings", destinado a contener todas las cadenas empleadas en la aplicación. De este modo se facilita la adaptación de la aplicación a otros idiomas.
  Cuando se quiere adaptar la aplicación para su empleo en varios idiomas, se crea un archivo strings para cada uno de los idiomas deseados; en este proyecto se ha creado uno base en español y otro en inglés. Gracias a esta diferenciación, el idioma que mostrará la aplicación será el que tenga configurado el dispositivo en el que se esté ejecutando.
- **Manifest:** cuando se crea el proyecto de la aplicación automáticamente se genera un archivo denominado "AndroidManifest", que debe contener ciertos elementos que forman parte de la aplicación, junto con algunos elementos de configuración:
  - Los permisos que requiera la aplicación, en este caso el acceso a las funcionalidades bluetooth y localización del dispositivo.
  - El título de la aplicación, SesNe (llamado desde el archivo de cadenas) en este caso, así como su icono.
  - El tema de color a emplear.

- Las actividades que conforman el sistema, junto con su configuración: MainActivity, que no puede faltar, y Nodo. Como parte de la configuración se especifica la orientación con la que se muestra la información. MainActivity admite tanto orientación horizontal como vertical, mientras que Nodo solamente permite orientación vertical (android:screenOrientation="portrait").
- El servicio LocalService, explicado en el punto 6.2.

- **Gradle:** permite definir algunos de los parámetros base de la aplicación, como son la versión de Android para la cual se está diseñando ("compileSdkVersion"), así como la versión más antigua que aceptará ("minSdkVersion"). Además, en la parte de "dependencies" se definen las librerías empleadas en la aplicación.

6. Blockchain

Como elemento innovador, y debido a que sus posibles aplicaciones se ajustan al funcionamiento del sistema desarrollado, se han añadido una serie de funcionalidades basadas en blockchain. Dado que se trata de una tecnología nueva y relativamente desconocida, se ha optado por dedicarle un apartado íntegro de la memoria, con el fin de poder ofrecer una visión de conjunto del trabajo realizado.

Blockchain es una tecnología de la cual se habla mucho últimamente, aunque la mayoría de las veces únicamente mediante ideas, posibles funcionalidades, o sobre su aplicación en criptomonedas; en raras ocasiones se muestran hechos y aplicaciones reales y viables de la tecnología. Lo que se pretende mostrar con este proyecto es una de las posibles aplicaciones que esta tecnología tiene en el mundo de la electrónica. No se aportará una explicación al detalle de su funcionamiento, dado que el volumen de la información excedería el alcance de este Trabajo Fin de Grado.

La presentación se dividirá en diferentes subapartados para una mayor claridad. En primer lugar, se expondrá una introducción sobre qué es esta tecnología y por qué se ha usado en este proyecto, así como las herramientas, lenguajes de programación y otros elementos involucrados en el desarrollo. A continuación, se abordará la parte relacionada con el "Smart contract", concepto que se explica en el siguiente apartado, necesario para poder almacenar y extraer los datos. Seguidamente se presentará la parte de la aplicación de Android relacionada con blockchain. Por último, se expondrán los diferentes scripts que han sido desarrollados en Python para interactuar con la red.

6.1. Presentación de la tecnología y su funcionalidad

Con el empleo de esta tecnología se ha pretendido mostrar dos aplicaciones posibles que pueden resultar de utilidad. Por una parte se ha creado un almacenamiento permanente e inmutable de la información, de tal manera que se consiga un registro histórico de los valores generados por los sensores que pueda servir, si fuese necesario, como certificado de calidad. Por otra parte se ha tratado de dar un valor añadido a estos datos almacenados; en lugar de ser simplemente información disponible para el dueño del sistema, se ha creado un mecanismo por el cual cualquier persona interesada en los datos va a poder adquirirlos mediante el pago de una pequeña cantidad de un token ya existente en la red. De esta manera, cualquier persona que quiera realizar algún estudio, desarrollar un algoritmo de optimización, u otras aplicaciones para las cuales se necesitan datos, tendrá un modo de conseguir la información que necesita, a la vez que el dueño de los datos podrá obtener un rendimiento económico a partir de ellos.

Como red de blockchain en este proyecto se ha empleado Quorum, red desarrollada por el banco JP Morgan, basada en Ethereum pero con un protocolo de consenso distintito al de Ethereum estándar y sin la existencia de una criptomoneda, con lo cual las transacciones

son de coste cero. Este último factor es determinante para este proyecto, al ser un sistema que almacena mucha información pero en transacciones muy pequeñas. En una red como Ethereum estándar esto tendría un coste en ether (criptomoneda propia de la red Ethereum) muy elevado a la larga, de modo que no compensaría su utilización. Ello implicaría tener que buscar otro tipo de soluciones, como el empleo de redes paralelas como µRaiden, las cuales todavía se encuentran en un estado muy temprano de desarrollo y no aportan la fiabilidad necesaria al sistema.

Conviene aclarar algunos términos que aparecerán a lo largo del documento:

- **Dirección:** es un conjunto de caracteres alfanuméricos únicos que representan un par clave publica/privada.
- **Clave publica/privada:** concepto perteneciente al mundo de la criptografía asimétrica que se refiere a un método criptográfico que emplea un par de claves para el envío de mensajes cifrados. Ambas claves pertenecen a la misma persona. La clave pública, que es generada a partir de la privada puede ser conocida sin riesgo por todo el mundo, mientras que la clave privada únicamente debe ser conocida por su dueño.
  La red de blockchain tiene constancia del número de ether que le pertenecen a la clave pública. Cuando su dueño quiera realizar una transacción la firmará con su clave privada, de tal manera que la red sabrá que ha sido él quien la ha enviado. De este modo, únicamente el dueño de la clave privada puede interactuar con todo lo relacionado con su clave pública.
- **Smart contract:** es un programa que "vive" y se ejecuta en la red de blockchain. Una vez ha sido desplegado dentro de ella es imposible modificarlo. Los datos almacenados por el mismo nunca se podrán borrar y los pagos realizados serán automáticos y no se podrán evitar. Se ha empleado en este proyecto por su inmutabilidad.
- **Gas:** es un parámetro propio de Ethereum. Dado que se trata de una red descentralizada, en la que todos los nodos deben ejecutar todas las operaciones, es necesario limitar el periodo de ejecución máximo de cada transacción para evitar que se puedan producir bucles infinitos que bloqueen el sistema. Este límite, basado en la potencia computacional necesaria para cada transacción, es ~~el~~ lo que se denomina gas.
- **Token:** concepto relacionado con blockchain que admite explicaciones variadas en contextos diversos; en el caso de este proyecto puede interpretarse como una moneda con la que poder pagar, de igual modo que con una criptomoneda.

6.2. Smart contract

Como se acaba de mencionar, un Smart contract es un conjunto de líneas de código que se ejecutan dentro de la red. Al emplear una red basada en Ethereum es posible

desarrollar estos contratos en cinco lenguajes de programación distintos: Solidity, LLL, Serpent, Mutan y Viper.

Para este proyecto el contrato ha sido desarrollado empleando Solidity, por ser el más extendido en su uso. Para poner en contexto el desarrollo es interesante destacar que la versión 0.1.0 de Solidity, la primera en el changelog de Github, se publicó el 10 de julio de 2015, poniendo de manifiesto que se trata de un lenguaje de programación muy joven que está en constante desarrollo (prácticamente cada mes sale una nueva versión). Todavía cuenta con una comunidad bastante reducida, lo cual hace que la resolución de algunos problemas sea muy costosa.

El código del contrato desarrollado se encuentra en el anexo 5.1. Para poder hacer uso de él, debe existir previamente en la red; es decir, el contrato ha tenido que ser ejecutado en la red y, de este modo, podrá ser accesible desde otros contratos. La primera línea del código contiene el indicador de la versión de Solidity empleada en su desarrollo, en este caso la 0.4.21. A continuación se muestra un primer contrato, denominado "TokenInterface", que contiene únicamente las funciones del contrato del token ya desplegado en la red que se van a usar, en este caso únicamente "transferFrom". Esto permitirá realizar el envío de los tokens cuando se compren los datos.

El siguiente es un contrato empleado para "asignar propiedad", de modo que se designa como dueño a quien lo despliega. Contiene un modificador que, al ser aplicado a una función permite restringir su ejecución a su dueño.

El último contrato es el que se desplegará en la red. Dentro de él se encuentran las declaraciones de los eventos y las variables en la parte inicial, y de las funciones en la final. En primer lugar, se declara el evento que se activa cuando se compran los datos. Un evento es un modo de comunicarse con el exterior de la red, indicando que ha habido un cambio o algo relevante dentro de ella. En este caso se emplea para poder recibir los datos en el exterior de la red cuando se realiza el pago. El motivo para hacerlo mediante un evento se debe a que, en el momento de la redacción de este proyecto, las transacciones en Solidity no devuelven información fuera de la red. Para que una función devuelva información fuera de la red debe de ser del tipo "view", pero este tipo de funciones no pueden ejecutar transacciones, sino únicamente ejecutar el "return".

A continuación se declaran las variables:

- **Datos:** estructura que se empleará para almacenar los datos de los sensores. Contiene tres arrays de cadenas: una para las unidades, otra para el nombre del sensor, y se emplea una cadena también para el almacenamiento de las lecturas de los sensores porque Solidity no permite trabajar con números decimales. Asimismo contiene un entero sin signo de 256 bits para almacenar el "timestamp" de cuándo se publican los datos.

- **Nodo:** otra estructura que almacena dos contenidos: un "mapping" (esto es, un sistema de clave/valor) para poder almacenar los valores de una forma organizada, en el que se emplea la estructura "Datos" como valor y el nombre del nodo de la red como clave, y un array de cadenas que contiene la lista de nodos en la red.
- **Red:** mapping que emplea el nombre de la red como clave y la estructura Nodo como valor.
- **networks:** array de cadenas para almacenar todas las redes que hay.
- **owner:** mapping para relacionar las redes con sus dueños.

Lo siguiente son las funciones:

- **setTokenContractAddress:** establece la dirección del contrato del token y crea el interfaz de conexión con el contrato. Tiene como parámetro de entrada una dirección, y cuenta con el modificador "onlyOwner", que solo permite la ejecución por parte del dueño del contrato. Esto se hizo esto para evitar que cualquier persona pudiera cambiar la dirección por otra distinta de la del token, lo cual provocaría un mal funcionamiento del sistema.
- **SaveData:** almacena los valores recibidos dentro de la red. Para ello tiene como parámetros de entrada: el nombre de la red, el nombre del nodo, las unidades, el nombre del sensor y la medida del sensor. Con todo esto, en primer lugar se comprueba si la red especificada ya existe; en caso contrario se almacena dentro del array "networks". A continuación, se comprueba si existe el nodo y, al igual que con la red, si no existe se almacena en un array, en este caso "nodelist". Por ultimo se almacenan los valores leídos; para ello se hace un "push" sobre el array correspondiente para cada una de las medidas, empleando el mapeo en base a la red y al nodo.
- **BuyData:** esta función es la empleada para la adquisición de los datos por parte de un comprador. Entre los parámetros de entrada se especifican los nombres de la red seleccionada y de su nodo, además de indicarse la posición del array de valores que se desea extraer. Esto se hace de este modo debido a una limitación de Solidity, que como ya se ha mencionado es un lenguaje muy joven y aún en desarrollo. En el momento de elaborar este proyecto, Solidity no permite devolver arrays de cadenas, por lo que, para devolver un array completo, es necesario recorrerlo entero posición a posición e ir concatenando los valores (esto se detalla en el punto 7.4, Scripts Python). Una vez ejecutada la función, en primer lugar se comprueba si quien la ha ejecutado es el dueño de la red seleccionada; en caso negativo se hace una llamada a la función "transferFrom" del contrato del token con el fin de hacer un envío de 10 tokens desde la dirección de quien ejecuta la función (msg.sender) hacia la dirección del dueño de la red. Para que este envío se pueda llevar a cabo, el ejecutor de la función previamente ha tenido que autorizar al contrato (el de

almacenamiento de los datos) a gastar en su nombre una cierta cantidad de tokens.

Una vez hecho el envío de los tokens, se genera un evento que indica el nombre de la red y el nodo, así como las medidas leídas de los diferentes arrays.

- **OwnerReturn:** Indica quién es el dueño de la red.
- **tokenA:** Devuelve la dirección del contrato del token.

Debido a la limitación expuesta en la función BuyData, se desarrollaron una serie de funciones para determinar diferentes parámetros necesarios para poder obtener todos los datos pertenecientes a una red (por ejemplo, el número total de datos por nodo, el número total de nodos por red, etc.):

- **networksNumber:** Devuelve el número total de redes.
- **networkName:** Devuelve el nombre de la red en la posición _n.
- **nodesNumber:** Devuelve el número de nodos en la red.
- **nodeName:** Devuelve el nombre del nodo.
- **datosNumber:** Devuelve la cantidad de datos almacenados para ese nodo.

Para disponer de una visión de conjunto, lo que se permite con este contrato es, por un lado, almacenar los datos de los sensores dentro de la red empleando la función SaveData, y, por otro, permitir recuperar esos datos tanto por parte del dueño, sin coste, o de cualquier otra persona, pagando por la información. Para poder seleccionar los datos deseados, las redes se clasifican en base a su nombre y a los nodos que las componen. El procedimiento concreto de extracción de la información se detalla más adelante, en el apartado 7.4 (Scripts Python).

6.3. Aplicación Android

En lo concerniente a la aplicación, se han incluido dos elementos relacionados con la parte de blockchain: una nueva actividad, denominada “settings” (Anexo 5.2.1), a la cual se accede pulsando en el elemento del menú lateral “Ajustes”, y una nueva clase de Java (Anexo 5.2.2), denominada “apicall”, dentro de la cual almacenar la función o funciones necesarias para interaccionar con la red.

El funcionamiento de la actividad “settings” es bastante simple; como se muestra en la figura 21, contiene dos campos de texto, un “checkbox” y un botón para guardar. En los campos de texto se debe escribir la siguiente información: en el superior la dirección que se empleará para publicar los datos recibidos de los sensores, y en el inferior la clave privada correspondiente a esa dirección. El “checkbox” se emplea para activar o desactivar el almacenamiento de datos en blockchain, y el botón para guardar los ajustes.

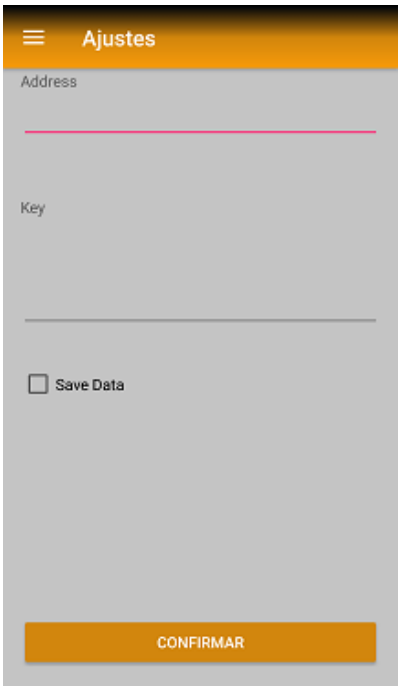

Figura 21 Settings.

Una vez pulsado el botón de guardar, estos datos se almacenan empleando "SharedPreferences", al igual que se hace para almacenar los nombres de la red y los nodos en la función "separar()".

La clase de Java, por su parte, contiene una función denominada "Api", la cual recibe como parámetros de entrada los siguientes valores:

- Contexto que llama a la función.
- Nombre de la red.
- Nombre del nodo.
- Medida del sensor.
- Unidades.
- Nombre del sensor.

Dentro de esta función se forma la URL con la cual se realizará una petición GET para enviar los datos a las ApiRest que los publicarán dentro de la red de Quorum. El motivo para el empleo de esta ApiRest se expone en el apartado siguiente, donde se detallan los scripts de Python desarrollados.

6.4. Scripts Python

Para poder establecer la conexión con la red, tanto para la publicación de los datos como para su compra, se desarrollaron una serie de scripts en Python que ofrecen toda la funcionalidad que necesita el sistema. Debido a que son varios, con mayor o menor

complejidad, se desarrollará cada uno en un apartado independiente, y se añadirá también un apartado en el que se explique el flujo de entrada y salida de los datos.

6.4.1. SensoresEvento

(Anexo 5.3.1) En este script se almacenan las funciones que serán empleadas en la comunicación con el contrato, las cuales serán llamadas desde los otros scripts. Dentro de él, en primer lugar se establece la conexión con la red; para ello se crea un conector http empleando la librería "web3", que nos devuelve una especie de socket que nos permitirá comunicarnos con la red.

Una vez tenemos el socket con la red, se establece la conexión con el contrato. Para ello se necesitan dos elementos, su dirección y el "abi" correspondiente al código. La dirección indica la "localización" del contrato dentro de la red, mientras que el abi es una variable con formato "json" que se obtiene al compilar el código y que representa el conjunto de funciones y variables con las que cuenta el contrato. Para dar una mayor claridad al código, el abi se ha almacenado en un script diferente denominado "Abisensores", el cual se importa al comienzo del script. Con estos dos valores se establece la conexión con el contrato, y se crea una variable llamada "contrato", la cual se empleará cada vez que se quiera interactuar con una función del contrato desplegado en la red.

Lo siguiente que se muestra en el script son las funciones necesarias para la interacción con el contrato:

- **GuardarDato (From, Clave, Red, nodo, sensor, unidades, medida):** esta función es una versión simplificada de la empleada para publicar los datos en la red que existe en el script "GuardarDato"; el motivo para mantenerla en este script se debe a que es una función que permite explicar una amplia variedad de elementos relacionados con las llamadas a las funciones de los contratos. Recibe siete parámetros de entrada:
  - **From:** indica la dirección desde la que se ejecutará la transacción; también será la dirección que se convertirá en la dueña de los datos enviados.
  - **Clave:** clave privada de la dirección anterior. Para poder ejecutar la transacción es necesario autorizarla mediante el empleo de esta clave.
  - **Red:** Nombre de la red de sensores a la que pertenecen los datos.
  - **Nodo:** Nombre del nodo al que pertenecen los datos.
  - **Sensor:** Nombre del sensor.
  - **Unidades:** Unidades de la medición.
  - **Medida:** Lectura del sensor.

  Dentro de esta función, en primer lugar se desbloquea la dirección "From", empleando la clave privada durante un periodo de tiempo para poder ejecutar la transacción (publicar los datos). Una vez desbloqueada la dirección se ejecuta la transacción, en la cual se indica quién la realiza, el gas que se va a emplear, y la función que se quiere ejecutar, SaveData, junto con los parámetros que necesita esta función para ser ejecutada.

- **Redes():** genera una lista de las redes que hay almacenadas en el contrato. Para ello, en primer lugar se hace una llamada a la función "networksNumber" del contrato, que indica la cantidad total de redes almacenadas. Posteriormente, en un bucle for, se ejecuta la función "networkName" desde cero hasta el valor de networksNumber, con el fin de ir añadiendo a la lista los nombres de todas las redes que hay en el sistema.
  La razón por la que esta función se ejecuta mediante ~~el~~ un bucle for radica en la limitación de Solidity comentada anteriormente de no permitir devolver arrays de cadenas.
- **Nodos (red):** funciona de una forma similar a la anterior. En este caso se usa como parámetro de entrada la red elegida y, a continuación, se realiza un proceso similar: primero se obtiene el numero de nodos en la red (función "nodesNumber") y después, en un bucle for, se va llamando a la función "nodeName" y añadiendo el resultado a una lista.
- **SacarDato (Red, From, Key):** se llama cuando se sacan los datos. Recibe como parámetros de entrada la red seleccionada, junto con la dirección que ejecuta la función y su clave privada. Se contemplan dos posibilidades: si se selecciona una red se devolverán únicamente los datos de esa red, mientras que si no se selecciona ninguna y se manda un "0" en su lugar, se devolverán los datos de todas las redes que hay almacenadas.
  En el primer caso, cuando se selecciona una red, en primer lugar se obtiene la lista de los nodos que forman parte de la red mediante una llamada a la función "Nodos", para ejecutar a continuación un bucle for que va recorriendo la lista de los nodos y ejecutando la función "MedidasHistorico", explicada más adelante.
  En el segundo caso, cuando no se indica la red, primero se obtiene la lista de redes y después se ejecuta un bucle for que recorre esta lista, dentro del cual se ejecuta la misma acción que en el caso anterior: se obtienen los nodos de la red y se va llamando a "MedidasHistorico".
- **NumDato (red, nodo):** devuelve el número de datos almacenados en el nodo de la red especificado como argumento de entrada.
- **MedidaN (red, nodo, n, From, Key):** se emplean los parámetros de entrada que indican la red seleccionada, el nodo y la posición en el array del valor que se desea extraer para ejecutar una transacción a la función del contrato "BuyData". Para ello se especifican también la dirección desde la que se ejecuta la transacción y la clave privada.
- **MedidasHistorico (red, nodo, From, Key):** en primer lugar obtiene el número de elementos que hay en el nodo; para ello emplea los parámetros de entrada del nombre de la red y el nodo elegido, y, mediante un bucle for, recorre desde cero hasta el numero total de elementos. En el interior de este bucle ejecuta la función "MedidaN" para ir ejecutando las transacciones correspondientes a cada elemento.
- **ContDatos (Red):** funciona de una forma similar a "SacarDato", con la diferencia de que dentro del bucle for, en lugar de ejecutarse la función "MedidasHistorico", se

ejecuta la función “NumDatos”, con lo que se obtiene el numero de datos que hay en la red indicada.

Este script está diseñado para simplificar el uso de los demás; por ello contiene todas las funciones necesarias para poder comunicarse con el contrato.

6.4.2. Abisensores

(Anexo 5.3.2) Dentro de este script se encuentra el abi correspondiente al contrato. Como se ha comentado, se trata de un fichero en formato “json” que contiene las funciones y las variables, así como una función que, al ser llamada, devuelve el propio abi.

6.4.3. APIsensores

(Anexo 5.3.3) Para poder publicar los datos dentro de la red desde la aplicación del móvil, se deben enviar a una APIREST que los reciba y los publique. Este script contiene una Api, creada con la librería flask, que se mantiene a la escucha en el puerto 15000.

Contiene un único método GET, el cual realiza una llamada a la función “GuardarDato”, contenida en el script del mismo nombre. Para realizar esta llamada se toman los argumentos que componen la llamada, y se asignan a las diferentes variables que son necesarias para ejecutar la función GuardarDato, tal como se muestra en el siguiente ejemplo:

```
"http://ip:15000/sensores/guardar?address=" + Address + "&clave=" + Key +
"&red=" + Nom_Red + "&nodo=" + Nom_Nodo + "&sensornombre=" + Nom_sensor +
"&unidades=" + Unidades + "&medida=" + Lectura;
```

6.4.4. GuardarDatos

(Anexo 5.3.4) En este script se realiza el envío de los datos al contrato. Al igual que en el caso de “sensoresEvento”, lo primero que se hace es establecer la conexión con la red y con el contrato. A continuación, se declara una clave privada de tipo Fernet, para lo cual se importa la librería “cryptography”.

La razón para emplear una clave para encriptar los valores recibidos nace de una de las características fundamentales de blockchain: la necesidad de que las transacciones sean públicas, para que toda la información sea visible y pueda conocerse en todo momento lo que sucede en la red. Sin embargo, cuando se trata de enviar los datos, no pueden ser publicados directamente sobre la red porque cualquiera que consultase el historial de transacciones del bloque podría conocer la información enviada y, de este modo, obtener los datos sin necesidad de pagar. Para evitarlo, una vez recibidos los datos en la ApiRest, y antes de ser publicados en Quorum, se encriptan empleando el sistema Fernet. Se trata de un sistema que emplea una clave privada para encriptar los datos, de tal manera que, si no se dispone de ella, no es posible descifrarlos y solo se visualiza un conjunto de caracteres sin sentido. De este modo se consigue compatibilizar la necesaria privacidad de la información con el carácter público y auditable inherente a blockchain.

Por ejemplo, si la API recibe un valor de 20.5 (supongamos que se trata de la medida de un sensor de temperatura), el resultado después de encriptarlo podría ser algo como lo siguiente:

```
gAAAAABbMVdQ0KD3k9bYL7OWulJOY5emoodtjAUmgIcXWjUyY6Gsq6v52OYsyRSF5d17kWcsj
gSxXMhaARgwyqK9sQHxFQMu7g==
```

Como se puede ver, no es posible conocer la información enviada, una vez encriptada, si no se dispone de la clave privada.

La función "GuardarDato lleva a cabo el siguiente proceso: en primer lugar desbloquea la cuenta, para lo que emplea los parámetros From y Key, y seguidamente los valores de entrada de los sensores.

Los valores de entrada son cadenas, pero para poder encriptar con Fernet se necesita que sean bytes, razón por la que, en primer lugar, se convierten las cadenas a bytes empleando la función "encode". A continuación se encriptan los bytes y se convierten en cadena. Por último, se eliminan los dos caracteres iniciales y el último, dado que son símbolos que se añaden al convertir en cadena y que generan problemas al desencriptar. Una vez los datos están listos, se hace una transacción a SaveData con los valores del nombre de la red y del nodo, junto con el nombre del sensor, las unidades y la medida del sensor encriptados.

Con esto los datos ya estarán publicados en el contrato.

6.4.5. Decodificar

(Anexo 5.3.5) Este script, al igual que APIsensores, contiene una APIrest desarrollada empleando flask que implementa un único método GET. Esta API se encuentra en el puerto 20500 y cumple con la función de desencriptar los datos leídos de la red cuando se extraen. Para ello, en la llamada a la API se suministran los datos encriptados como argumento, se desencriptan en la función "dec", y se devuelven en la respuesta a la petición.

De este modo se dispone de un sistema que permite desencriptar los datos, pero sin necesidad de facilitar la clave privada a quien los extrae de la red.

6.4.6. Frontend

(Anexo 5.3.6) Para poder extraer de una forma cómoda y sencilla los datos de la red, se ha desarrollado el front mostrado en la figura 29, empleando para ello la librería "tkinter" de Python.

En este script, al igual que en los anteriores, lo primero que se hace es establecer la conexión con la red y con el contrato; una vez ha sido realizada, se inicia la variable "cont", que se empleará para contar los valores que se extraigan. Para que la descripción del sistema sea más fácil de entender, en lugar de explicar las funciones según el orden de aparición en el script, se expondrán siguiendo el orden lógico de ejecución.

Se crea una clase llamada "Aplicación", dentro de la cual se agrupan las funciones directamente relacionadas con el front, y no con el tratamiento de los datos. Estas funciones son:

- **__init__(self):** en ella se definen las características de la interfaz: tamaño, si se pueden modificar las dimensiones por parte del usuario, icono y título. Junto a éstos, se definen los elementos que forman parte de la interfaz, en orden de superior a inferior a la hora de mostrarse:
  - **LAddress:** label que muestra el texto "Dirección".
  - **TextAddress:** campo de texto en el que se escribe la dirección que se empleará para comprar los datos.
  - **LClave:** label que muestra el texto "Clave".
  - **TextKey:** campo de texto en el que se escribe la clave privada de la dirección que se empleará para comprar los datos.
  - **BotonBalance:** cuando se pulsa, se ejecuta la función "Balance".
  - **BotonBuy:** ejecuta la función "buyData".
  - **combobox:** muestra la lista de redes almacenadas en el contrato.
  - **LMessage:** label empleado para mostrar diferentes mensajes al usuario durante el uso de la aplicación.

  Además de los elementos comentados arriba, en la ejecución de esta función se hace una llamada a la función "Redes", contenida en el script "sensoresEvento", y se añade el resultado a la lista de elementos del combobox.

- **buyData(self):** en primer lugar, se obtiene la información especificada a través de los diferentes campos de la aplicación: el nombre de la red seleccionada en el combobox, la dirección introducida en el primer campo de texto, y la clave privada facilitada en el segundo campo de texto. En el caso de estos dos últimos se debe eliminar el carácter final, dado que al ejecutar el "get" se añade "/n" y no es necesario.
  A partir del nombre de la red seleccionada se obtiene el número de datos que se van a extraer mediante una llamada a la función "ContaDatos" del script "sensoresEvento".
  A continuación, se crea el filtro de eventos que permitirá capturar los eventos generados por la función "BuyData" y se inicia la captura de los eventos mediante la función "log_loop". Durante el desarrollo de este sistema surgió un problema: era necesario poder capturar los eventos, pero al mismo tiempo ejecutar la función que los genera. Para solucionarlo se recurrió a la librería "threading", que permite ejecutar un proceso en un hilo en segundo plano y mantener la ejecución del programa en el principal. De este modo, se consigue que la función "log_loop" se ejecute en segundo plano mientras el programa continúa en el hilo principal. Una vez iniciada la captura se llama a la función "SacarDato" (contenida, al igual que las anteriores, en el script "sensoresEvento"), y cuando el proceso de extracción de los datos ha acabado se

modifica el texto del "label" "LMessage" para pasar a mostrar el texto "Process Complete", tal como se muestra en la figura 22.

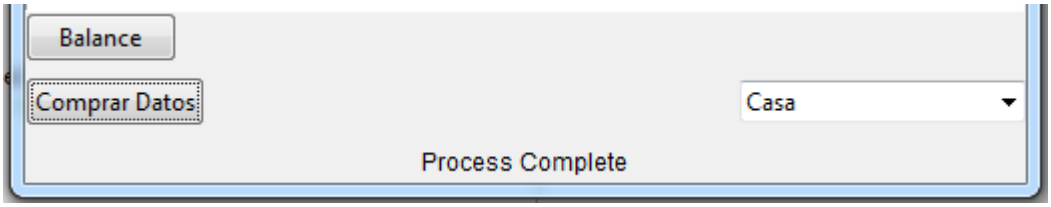

Figura 22 Proceso completado

- **Balance(self):** hace una llamada a la api que comunica con el contrato del token (esta api debe estar ejecutándose), y le envía la dirección introducida en el campo de texto "TextAddress". La api responde con la cantidad de tokens disponibles en la dirección, cantidad que se muestra en el label "LMessage", como puede verse en la figura 23.

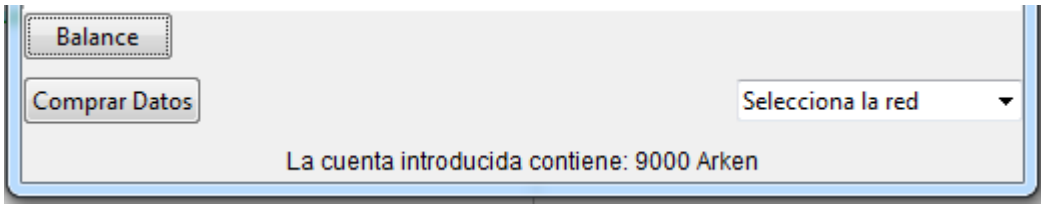

Figura 23 Balance

- **selection_changed (self, event):** detecta los cambios en la selección de los elementos del "combobox", pasando a mostrar en "LMessage" el coste que tendrá comprar los datos de la red seleccionada (figura 24).

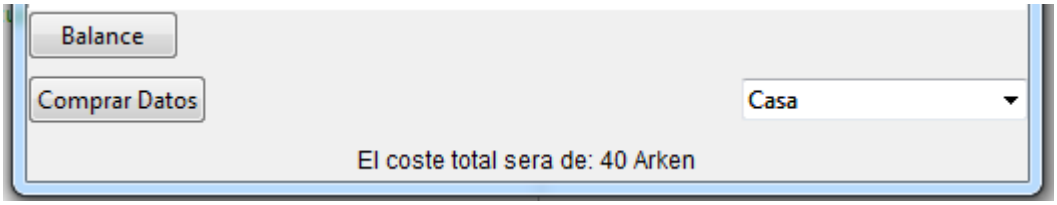

Figura 24 Coste

Junto a éstas se encuentran, fuera de la clase, las funciones relacionadas con el tratamiento de los datos extraídos de la red:

- **log_loop(event_filter, poll_interval, cont, last, red):** en la ejecución de esta función se emplean como parámetros de entrada: el filtro que detecta el evento, la frecuencia de repetición de la función, el valor del contador "cont", el número de datos que se van a extraer ("last") y el nombre de la red.
  Con estos valores se ejecuta un bucle while entre el valor de "cont" y el valor de "last", y en su interior un for que detecta cuándo se producen nuevos eventos en la red de Quorum. Cuando esto ocurre, se llama a la función "GetData" indicando el evento (una

variable de tipo diccionario) y el número del dato que se ha leído, lo cual devuelve los datos extraídos de la red y preparados para ser guardados. Se incrementa el valor de "cont" y se añaden los datos a la lista "valores". Si los valores de "cont" y "last" son iguales, significa que no hay más datos a extraer, y se llama a la función "SavaDataCsv" para guardar los datos.

- **GetData(event, cont):** recibe el diccionario "event" como parámetro de entrada, y selecciona el elemento "data" del mismo. En él están los datos que se han extraído de la red, pero sin haber sido procesados, por lo que están cifrados y además incluyen datos que no interesan. Para poder adaptarlos se recurre a la función "stripData".

- **stripData(data, cont):** la variable data contiene una gran cantidad de información en formato hexadecimal (figura 25) que debe ser procesada para poder acceder a los datos.


```
0x0000000000000000000000000000000000000000000000000000000000000c0000000000000000000000000000000000000000
0000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000140000000000000000
0000000000000000000000000000000000000001e0000000000000000000000000000000000000000000000000000000000000000000
0280000000000000000000000000000000000000000000000153695d7a77ddf490000000000000000000000000000000000000000
0000000000000000000000004436173610000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000c48616269746163696f6e5f3000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000646741414141414426248432d755474716d4a70
68515f664d416e7554774c542d777a6e6147314831776f4e346b6a76424c4f7a685f35346a6e356468797961336a4a4c4d4b33634a
6769335931536b3678552d43563679413172472d4e45354f6276413d3d0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000646741414141414426248432d754958314449
6e586659714b6866584476397444516333317a496e61664c5458754e77437653474950657a65486f665948513079734773726d4e35
4f6f495f3035566e484c34313950703479483561 6d686c6650584a413d3d0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000646741414141414426248432d75542d5657
65454467735 96a333441763142584362434652393364327 24b54334f5755624e675451626e5a647347797a39564d63766b3936774c
6a4d7744564d585052326e4a47646a3431357438464f79626e69614d513d3d0000000000000000000000000000000000000000000000
000000000000
```


Figura 25 Data original

Parte de esta información es añadida por la red con fines que no son de interés en esta aplicación. Por ello, el primer paso consiste en eliminar los 335 dígitos iniciales, y a continuación se divide la cadena eliminando la mayoría de los ceros que conectan cada una de las series de caracteres (Figura 26).


```
['00000153695d7a77ddf49', '000443617361', '000000000c48616269746163696f6e5f3',
'000646741414141414426248432d755474716d4a7068515f664d416e7554774c542d777a6e6147314831776f4e346b6a76424c4f7a685f35346a6e3564687979613
36a4a4c4d4b33634a6769335931536b3678552d43563679413172472d4e45354f6276413d3d',
'0000000064674141414141426248432d7549583144496e586659714b68665844763974445163333317a496e61664c5458754e77437653474950657a65486f665948
513079734773726d4e354f6f495f3035566e484c34313950703479483561 6d686c6650584a413d3d',
'0000000064674141414141426248432d75542d56576545446773596a333441763142584362434652393364327 24b54334f5755624e675451626e5a647347797a39
564d63766b3936774c6a4d7744564d585052326e4a47646a3431357438464f79626e69614d513d3d', '000000']
```


Figura 26 Data Paso 1: cadena dividida

Ejecutando un bucle for se recorren los seis elementos que ahora forman parte de data (pueden ser siete si hay un conjunto de ceros al final, pero se trata de un valor que no interesa) y se adaptan para mostrar la información. En todos los casos se eliminan los ceros contenidos al principio de la cadena (figura 27), dado que se sabe que ninguna

de ellas puede comenzar por cero, ya que los primeros dígitos de cada una de las seis cadenas indican el número de dígitos total, que nunca es cero.

```
153695d7a77ddf49
443617361
c486162697461636 96f6e5f3
6467414141414142624843 2d755474716d4a7068515f664d416e7554774c542d777a6e6147314831776f4e346b6a76424c4f7a685f35346a6e356
468797961336a4a4c4d4b33634a6769335931536b3678552d43563679413172472d4e45354f6276413d3d
6467414141414142624843 2d7549583144496e586659714b6866584476397444516333317a496e61664c5458754e7743765347495065 7a65486f6
659485130797347737 26d4e354f6f495f3035566e484c34313950703479483561 6d686c6650584a413d3d
6467414141414142624843 2d75542d5657654544677359 6a33344176314258436243465239336432724b54334f5755624e675451626e5a6473477
97a39564d63766b3936774c6a4d7744564d585052326e4a47646a3431357438464f79626e69614d513d3d
```

Figura 27 Paso 2: ceros eliminados

Después se hace una diferenciación dependiendo del valor que se vaya a leer:

- **Timestamp:** es siempre el primer valor de los seis que se reciben; debe ser convertido de hexadecimal a decimal, y posteriormente de formato "epochtime" a año, mes, día, hora, minuto, y segundo. Para ello es necesario eliminar las cifras fraccionarias que no son necesarias.
- **Otros datos:** el resto de datos, al tratarse de cadenas, deben ser procesados de un modo distinto. En primer lugar se eliminan los dígitos que indican la longitud. Después se comprueba si el número de dígitos es par o impar. Al tratarse de parejas de dígitos hexadecimales, de 4 bits cada uno, no puede haber un número impar de dígitos; si es impar significa que el último dígito era un cero que ha sido eliminado en uno de los pasos anteriores, por lo que se añade un cero al final. Con los datos ya ajustados en longitud se comprueba si se trata de valores encriptados o no. Si no lo son (nombre de la red y del nodo), se convierten de hexadecimal a ASCII y se añade el resultado a la lista "resultado". Si, por el contrario, se trata de datos encriptados, lo cual se sabe porque la longitud es notablemente mayor, se realiza una petición GET sobre la APIRest en la que se esta ejecutando el script "Decodificar". La respuesta de esta petición contiene los datos ya desencriptados que se añaden a la lista "resultado".

Acabado este proceso, se obtiene una lista que contiene los datos extraídos en forma legible (figura 28).

```
['2018-06-09 19:51:16', 'Casa', 'Habitacion_0', '0', 'SensorPresion_1', 'Bar']
```

Figura 28 Resultado

El último paso restante consiste en repetir este proceso tantas veces como se indique en la función "log_loop" y finalmente, como se ha comentado anteriormente, guardar los datos en el archivo csv.

- **SavaDataCsv(name, datos):** cuando se acaban de añadir los datos extraídos a la lista en la función "log_loop" se llama a esta función, que, empleando un "with", genera un

archivo CSV, y, mediante un bucle for, recorre la lista de entrada "datos" almacenando los valores dentro del archivo "name". De este modo se genera un archivo en un formato estándar para facilitar el uso de los datos extraídos.

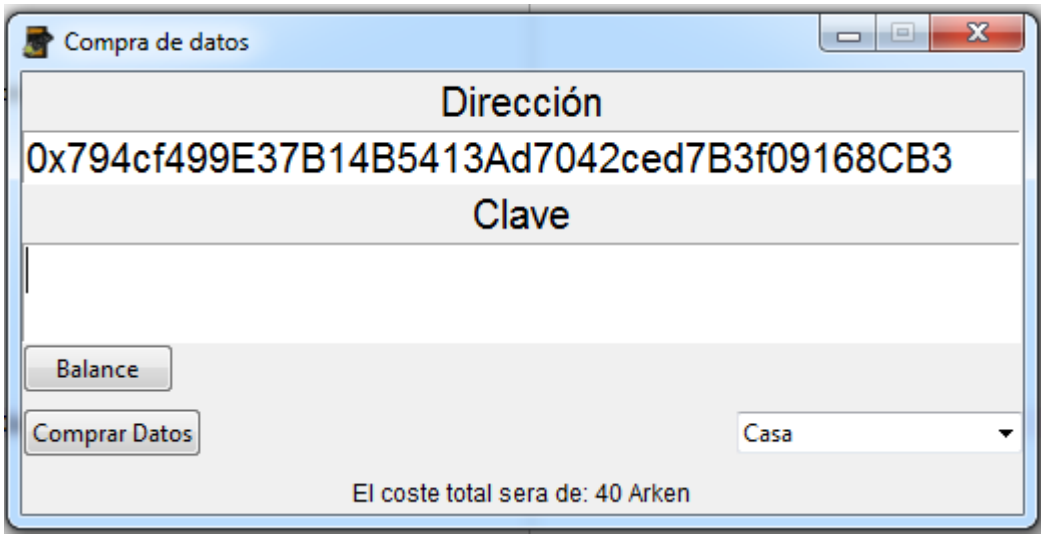

Figura 29 Front

6.4.7. BotTelegram

(Anexo 5.3.7) Para poder enviar los datos a la red es necesario poseer una dirección dentro de ella. Para generarla se consideraron diferentes alternativas, como, por ejemplo, hacerlo desde la propia aplicación del móvil, mediante un script en Python. Al final se optó por una solución más universal, que puede ser accesible para cualquiera de una forma fácil, y que, además, añade otro elemento novedoso al proyecto.

Se optó por desarrollar un bot para la aplicación de mensajería Telegram, que es una aplicación ampliamente usada que permite crear bots en Python de una forma muy cómoda. El primer paso para hacerlo consiste en obtener un token para el bot; para ello hay que hablar con el bot de la propia red, llamado "BotFather", al que se le indica la voluntad de crear un bot, junto con su nombre, de modo que devuelve el token correspondiente al bot.

Una vez se dispone del token, se emplea la librería "telegram" de Python para establecer, en primer lugar, la conexión con la red de Quorum, y, a continuación, la conexión con el bot que se ha creado, para lo que se emplea el token. Se crea la conexión denominada "dispatcher", que es una clase propia de la librería empleada que permite gestionar la conexión con los bot. A continuación, dentro de la función main se definen los diferentes comandos a los que responderá el bot, en este caso únicamente responde ante el comando "Direccion". Estos comandos se definen empleando el dispatcher y usando las funciones add_handler (crea un nuevo gestor para comandos) y CommandHandler (declara el comando). CommandHandler define el comando, y especifica si tiene argumentos de entrada, así como la función que se ejecutará cuando se envíe el comando.

En este caso, cuando se recibe “Direccion” llega como argumento la clave privada y se ejecuta la función “newAddress”; con esto se llama a la red de Quorum y se ejecuta la función “personal.newAccount(key)”, siendo key el valor de la clave privada. Esta función devolverá la dirección creada, que será la respuesta del bot de telegram ante el comando (figura 30).

Figura 30 Bot Telegram

Finalmente, solo habrá que copiar y pegar la respuesta del bot en el campo correspondiente de la aplicación de Android.

7. Conclusiones

Haciendo un esfuerzo por condensar todo el proceso de desarrollo seguido durante la realización del presente TFG, sus hitos más destacables serían:

- Selección de un sistema de comunicación entre nodos capaz de cubrir largas distancias con un coste reducido. Con tal fin se ha optado por una topología de red que, además, permite reducir las interferencias producidas entre nodos.
- Implementación de un prototipo que incluye los tres tipos posibles de nodos en la red: principal, intermedios y final.
- Selección del método de comunicación entre la red y el dispositivo inteligente.
- Desarrollo de una aplicación versátil para dispositivos móviles, válida para cualquier número de nodos y de sensores por nodo, que permite recibir datos desde el nodo principal empleando bluetooth.
- Habilitación del almacenamiento y recuperación de la información mediante blockchain, permitiendo la venta de los datos generados por los sensores.

El conjunto desarrollado podría ser considerado la primera versión de un sistema con un gran potencial. Entre las posibles mejoras que podrían tenerse en cuenta de cara a versiones posteriores, cabría plantearse la sustitución del Arduino Uno empleado en el nodo principal por un ESP32. Las principales razones para ello residen en su conectividad, pues incorpora bluetooth de baja energía y wifi, lo cual permitiría ofrecer al usuario también este último tipo de conexión. El empleo de este dispositivo no supondría un gran incremento de coste, dado que su precio es inferior a diez euros.

Una vez implementada esta modificación, la siguiente mejora posible sería la incorporación de un menú desplegable en la aplicación del dispositivo móvil que permitiese seleccionar el modo de conexión deseado: wifi o bluetooth.