industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Plataforma de telemetría basada en motes BLE y dispositivos móviles para la gestión de infraestructuras de distribución de agua.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA.

Autor: **José Benavente Pérez**
Director: Javier Garrigós

Cartagena, 21 de Junio de 2018.

Propuesta de Resolución y plan de Trabajo.

Estudiante:	José Benavente Pérez
DNI:	48454069-S
Correo electrónico:	josebenaventeperez@hotmail.es
Titulación:	Grado en electrónica industrial y automatización

Título del Trabajo:

Plataforma de telemetría basada en motes BLE y dispositivos móviles para la gestión de infraestructuras de distribución de agua.
Telemetry platform base on BLE motes and mobile devices for the management of water distribution infrastructures.

Director del TFE (y codirector en su caso):

Garrigós Guerrero, Francisco Javier.

Departamento responsable:

Electrón,Tecnol Computadoras y Proyectos.

Desarrollo de la propuesta de resolución y plan de trabajo

Para el desarrollo del proyecto podríamos separarlo por distintas fases:

Bloque 1. Asimilación de conocimientos previos.

Se comenzará con la asimilación del código básico del proyecto anterior, entendiendo los protocolos BLE, los servicios GAP y GATT, los diferentes frameworks y APIs para desarrollo de dispositivos y aplicaciones de BLE.

Bloque 2. Incorporación de nuevo código y depuración.

Se incorporará al código nuevos sensores como por ejemplo un caudalímetro por pulsos para mejorar y expandir dicho código además de conferir una mayor robustez y estabilidad al sistema.

Una vez hecho el código habría que depurarlo y probarlo, pero para ello se necesitaría crear la PCB.

Bloque 3. Placa PCB.

Para diseñar la PCB, primero hay que estudiar que componentes hay que incorporar. En concreto para esta PCB, se usará el microcontrolador Mbed BLE nano, el chip bluetooth (nordic nRF51822) además de una alimentación de la misma. En este caso la alimentación es una batería por lo que necesitaré hacer un acondicionamiento de la misma, incorporando un led (indicando la carga de la misma y una vez cargada se apagará), un interruptor para que no esté pasando corriente al nordic mientras se está cargando, condensadores para la eliminación de ruidos, las resistencias necesarias del acondicionamiento y nordic, el chip que utilizaría para el procesamiento del status y carga de batería sería en concreto el MCP73831, el conector necesario de la batería y por último un USB – B (como el de arduino) para la carga de la batería. Se ha elegido este tipo de USB porque da más juego a la hora de colocar en una carcasa además de la soldadura del mismo.

El diseño de la PCB se hará en Altium Designer, con un ruteo de pistas manual para poder conferir de una mayor facilidad al imprimir la placa, ya que sólo puedo imprimirla manualmente y soldar manualmente. Este trabajo es muy tedioso porque también hay que tener en cuenta el tamaño de la carcasa en la que se va a colocar y la posición de cada componente milimétricamente.

Bloque 4. Fabricación de la placa.

La fabricación de la placa se confeccionará en el laboratorio de PCB de la Universidad, por lo que una vez comprados los componentes con ayuda del técnico de laboratorio se fabricaría la placa diseñada previamente.

Bloque 5. Encapsulación.

Una vez diseñada y fabricada la PCB, se comprobará que todo está correctamente antes de encapsular (conectando la placa y probando el código previamente volcado), una vez probado encapsular en la carcasa conveniente y así se podrá implementar en campo para poder probarla. Se corregirían todos los fallos del código (si los hay) y se resolverían todos los problemas que puedan ir surgiendo, ya sea porque la intensidad de salida de los sensores no es la esperada, la necesidad de calibración de los sensores o la dificultad de sujeción de la carcasa en la arqueta, por ejemplo.

Bloque 6. Aplicación Android.

Por otra parte, comprobar la aplicación Android, ponerla en funcionamiento y depurar los errores que pueda tener o nos podamos encontrar; la batería aguantaría por meses por lo que podría hacer un estudio mediante dicha app por meses, viendo los fallos que puedan ir ocurriendo como por ejemplo las falsas medidas o si por el contrario funciona bien como se esperaba en un tiempo prolongado.

Para este apartado haré un curso en Android para ayudarme a manejar dicho código y depurar el programa lo máximo posible, además de poder añadir mejoras al programa.

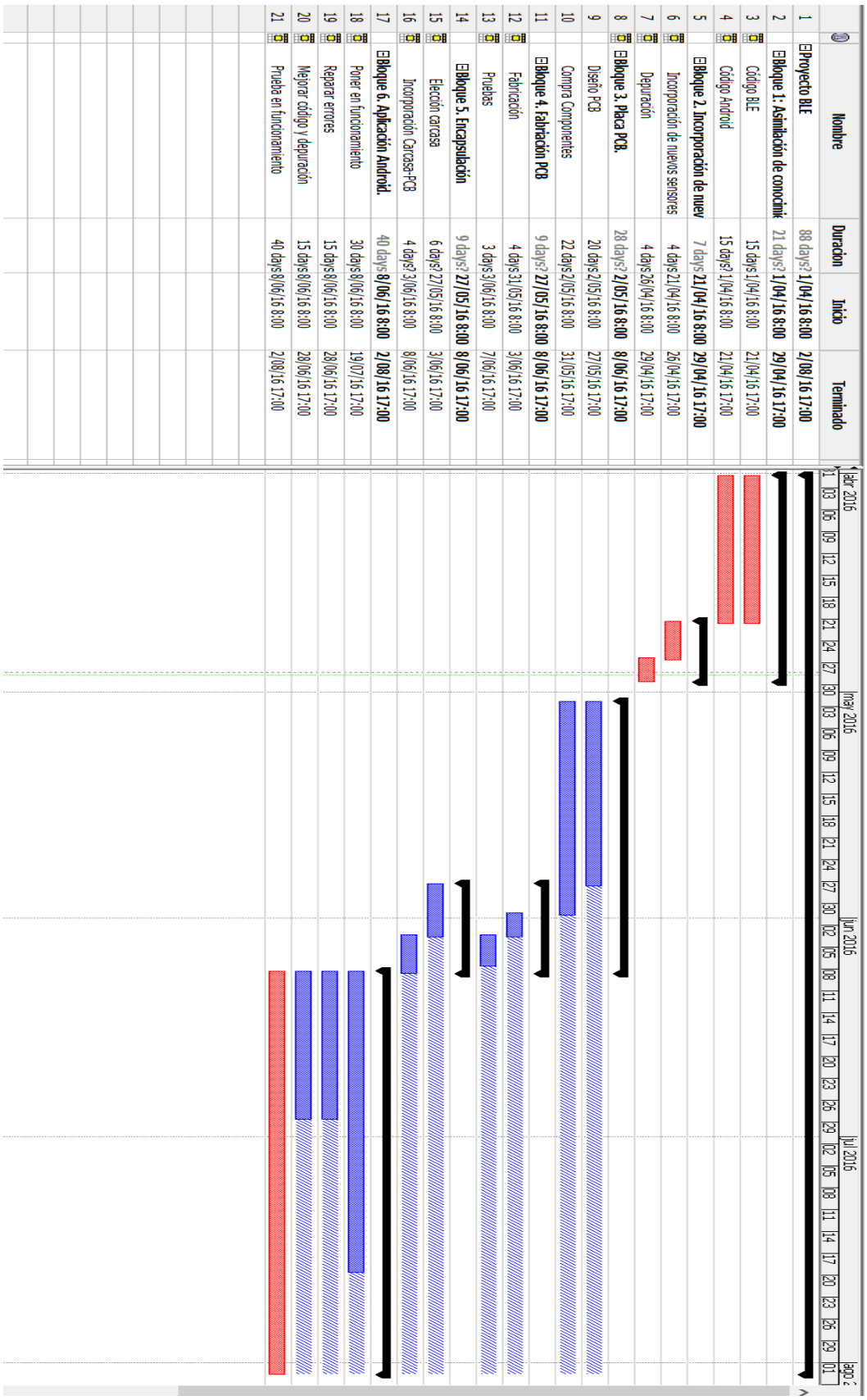
Bloque 7. Objetivo final.

El objetivo final es realizar un dispositivo funcional en campo, haciendo pruebas durante los meses anteriormente citados.

Bloque 8. Tiempo necesario estimado.

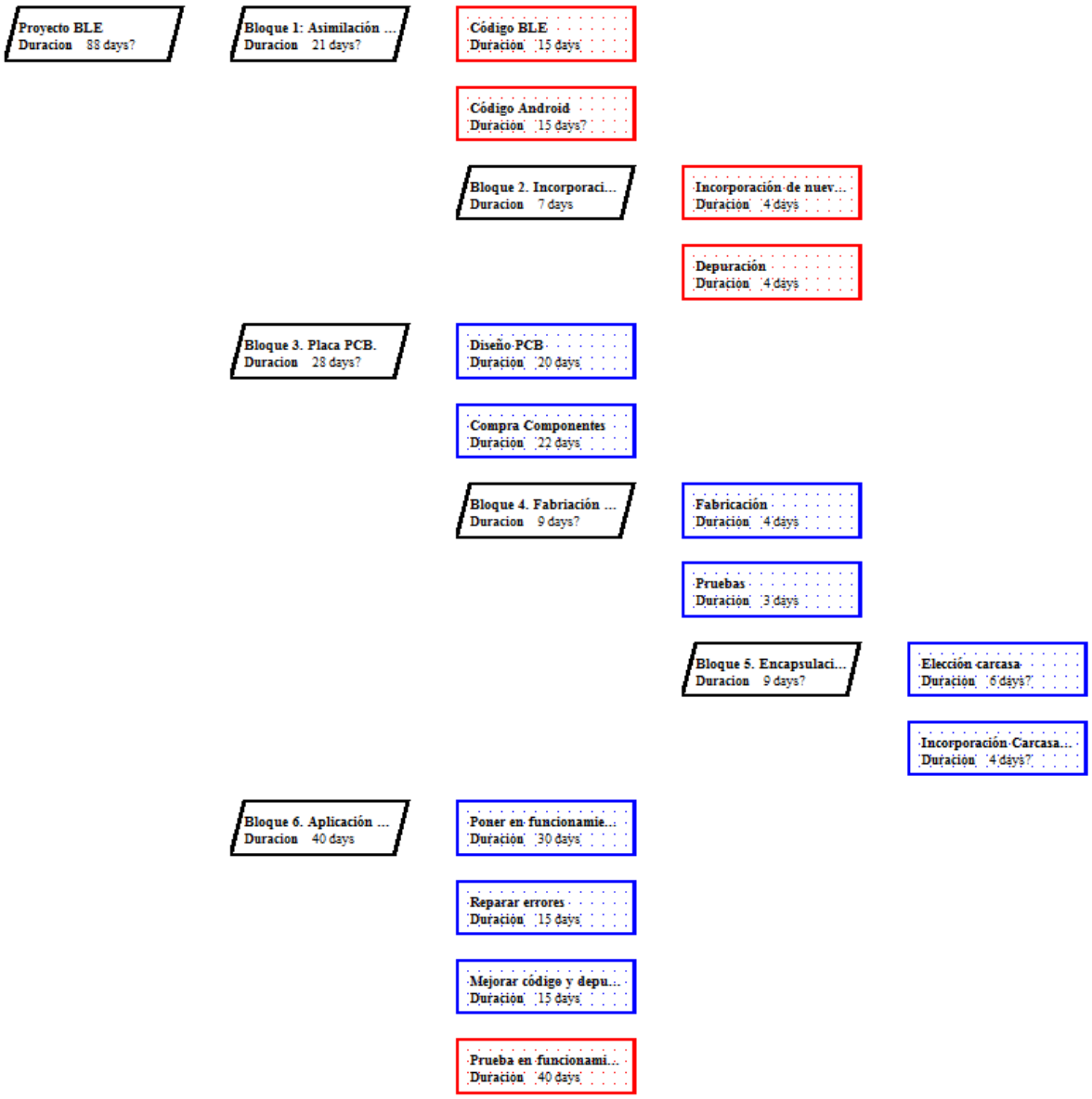
El tiempo invertido para cada fase es relativo, ya que pueden ir surgiendo problemas en cualquiera de ellas, pero como estimación en las primeras fases de adaptación más o menos serían unos 15 días, la PCB alrededor de un mes y las pruebas pertinentes unos 2 meses (ya que es donde más problemas me podría encontrar). Para este apartado incorporo el siguiente diagrama de Gantt que me ayudará en la organización del proyecto y sea más gráfico además de un desglose de tareas:

Diagrama de Gantt:



	Nombre	Duración	Inicio	Terminado
1	Proyecto BLE	88 days?	1/04/16 8:00	2/08/16 17:00
2	Bloque 1: Asimilación de conocimie	21 days?	1/04/16 8:00	29/04/16 17:00
3	Código BLE	15 days	1/04/16 8:00	21/04/16 17:00
4	Código Android	15 days?	1/04/16 8:00	21/04/16 17:00
5	Bloque 2. Incorporación de nuev	7 days	21/04/16 8:00	29/04/16 17:00
6	Incorporación de nuevos sensores	4 days	21/04/16 8:00	26/04/16 17:00
7	Depuración	4 days	26/04/16 8:00	29/04/16 17:00
8	Bloque 3. Placa PCB.	28 days?	2/05/16 8:00	8/06/16 17:00
9	Diseño PCB	20 days	2/05/16 8:00	27/05/16 17:00
10	Compra Componentes	22 days	2/05/16 8:00	31/05/16 17:00
11	Bloque 4. Fabricación PCB	9 days?	27/05/16 8:00	8/06/16 17:00
12	Fabricación	4 days	31/05/16 8:00	3/06/16 17:00
13	Pruebas	3 days	3/06/16 8:00	7/06/16 17:00
14	Bloque 5. Encapsulación	9 days?	27/05/16 8:00	8/06/16 17:00
15	Elección carcasa	6 days?	27/05/16 8:00	3/06/16 17:00
16	Incorporación Carcasa-PCB	4 days?	3/06/16 8:00	8/06/16 17:00
17	Bloque 6. Aplicación Android.	40 days	8/06/16 8:00	2/08/16 17:00
18	Poner en funcionamiento	30 days	8/06/16 8:00	19/07/16 17:00
19	Reparar errores	15 days	8/06/16 8:00	28/06/16 17:00
20	Mejorar código y depuración	15 days	8/06/16 8:00	28/06/16 17:00
21	Prueba en funcionamiento	40 days	8/06/16 8:00	2/08/16 17:00

Desglose de tareas:



Índice

Índice de ilustraciones, tablas y programas.

Capítulo 1

1. Introducción y objetivos del proyecto.

En este primer capítulo se hace la introducción del proyecto, así como los objetivos que se han abordado a lo largo de toda la investigación y elaboración de dicho proyecto.

1.1 Introducción.

Se ha llevado a cabo a partir de una previa investigación sobre un sistema distribuido que se encargará de la motorización de redes de distribución de agua y desarrollo de una aplicación Android, además de un desarrollo de un software desarrollado en la plataforma Mbed para dar funcionalidades a un dispositivo bluetooth.

Las funcionalidades del proyecto son diversas, como pueden ser:

- La recogida de datos de una red de distribución de aguas, que tendrá una serie de sensores los cuales enviarán sus señales de envío de datos pertinentes.

- Crear una comunicación estable y a la vez de bajo consumo de energía, ya que en este caso se utiliza una batería como método de aporte de energía, por lo que se han tenido en cuenta los estados de "sleep" o bajo consumo de una manera determinante y aprovechando el bajo consumo de la tecnología de Bluetooth Low Energy. En concreto, se utilizará una forma de transmisión de datos llamada baliza.

- El dispositivo será el encargado de la recogida de datos, procesarlos y enviarlos a los servidores online o nube, para después poder tener una mayor flexibilidad de trabajo, tanto de modo directo en el campo como de modo indirecto mediante el acceso a la nube desde cualquier dispositivo que disponga de WiFi.

- Se utilizarán unos servidores para la captación de los datos enviados a la nube, los cuales tendrán la función de almacenar y crear una capa de servicios que podrán ser utilizados posteriormente.

- Dotar de una base "física" al proyecto, una PCB que compone un hardware encargado de recibir la energía de la batería escogida,

de la carga de la misma mediante un USB tipo B (como el de Arduino que es muy común) y para poder dotar de energía al dispositivo utilizado (Ble Nano) para que pueda establecer así, las comunicaciones con los sensores anteriormente comentados.

1.2 Objetivos.

En cuanto a los objetivos del proyecto podemos concretar los siguientes:

- Completar el desarrollo de la aplicación Android a dispositivos con Android más actual y la adaptación a las pulgadas de la mayoría de dispositivos que se pueden encontrar en el mercado.

- Estudio de las problemáticas y evaluar las diferentes opciones que pueden ser encontradas en el transcurso de la evolución del proyecto, pudiendo así mejorar el proyecto que se había considerado en primera instancia.

- Desarrollo del equipo hardware, que se encargará de recibir los datos de los sensores y de dar soporte físico al dispositivo Ble Nano así como a la batería y a los conectores que irán conectados a los sensores correspondientes.

- Emular con un Arduino las señales de los sensores que en principio nos vamos a encontrar en la red de distribución de aguas, para poder saber si está bien adaptado y validar la funcionalidad de dicho dispositivo, antes de ser probado en campo abierto.

Capítulo 2

2. Estado del arte

2.1 Internet de las cosas (IoT).

Internet de las cosas abre una puerta a una transformación no sólo de los productos, que pueden personalizarse en tiempo real, sino también de los modelos de negocio con la creación de servicios y de pago por uso donde nunca se había monetizado anteriormente.

Es una corriente que tiene como aspiración ser el futuro de la red, conectando todos los dispositivos que anteriormente no se había planteado, como pueden ser un frigorífico o la tostadora.

Este año 2017, es el año del internet de las cosas, en concreto se prevé que para finales de año existirán un total de 8.400 millones de dispositivos conectados, es decir, un 31% más que el año anterior.

La falta de estándares dominantes es una barrera a la hora de adaptar e implantar esta tecnología y lograr que se extienda en el mercado. Uno de los puntos más negativos es que no todos los dispositivos se desarrollan teniendo en cuenta los más altos estándares de seguridad, dando lugar a los ciberataques tan famosos actualmente (Wannacry por ejemplo).

En este proyecto, por tanto, se tratará de explorar todas las capacidades de IoT para poder hacer más robusto y potente el dispositivo que se va a desarrollar.

2.2 Tecnologías inalámbricas.

En este proyecto se ha decidido utilizar como método de transporte de datos inalámbrico la tecnología bluetooth, pero anteriormente se han estudiado las demás tecnologías para elegir la opción más conveniente u óptima. A continuación, se van a comentar las principales tecnologías actuales que se podrían haber implementado:

2.2.1 WiMAX.

WiMAX es la abreviatura de Worldwide Interoperability for Microwave Access, nombre con el que se conoce al grupo de estándares IEEE 802.16, que es un estándar inalámbrico aprobado por el WiMAX fórum, al que pertenecen fabricantes de una gran diversidad de productos de telecomunicaciones. WiMAX es una tecnología inalámbrica diseñada para una red de área metropolitana con cobertura de 50 km por celda y tasas de transmisión de hasta 70 Mbps, utilizando la tecnología portátil LMDS (Local Multipoint Distribution Service). Sin embargo, el hecho de transmitir en una banda licenciada condiciona su uso a un Proveedor de Servicios de Internet (ISP por sus siglas en inglés); así mismo, como el tráfico de información médica viaja por Internet, esto la hace más insegura y menos confiable.

Para ver sus características más intuitivamente, podemos ver la siguiente tabla:

	802.16	***802.16a***	***802.16e***
Espectro	10 – 66 GHz	< 11 GHz	< 6 GHz
Funcionamiento	Sólo visión directa	Sin visión directa (NLOS)	Sin visión directa (NLOS)
Ancho de Banda	32 – 134 Mbps con canales de 28 MHz	Hasta 75 MHz con canales de 20 MHz	Hasta 15 Mbps con canales de 5 MHz
Modulación	QPSK, 16QAM y 64 QAM	QFDM con 256 subportadoras QPSK, 16QAM, 64QAM.	QFDM con 256 subportadoras QPSK, 16QAM, 64QAM.
Movilidad	Sistema Fijo	Sistema Fijo	Movilidad pedestre
Anchos del espectro	20, 25 y 28 MHz	Seleccionables entre 1.25 y 20 MHz	Seleccionables entre 1.25 y 20 MHz
Distancia	2 – 5 Km Aprox.	5 – 50 Km Aprox.	2 – 5 Km Aprox.

Tabla 1. WIMAX.

2.2.2 WiFi.

Se trata de un estándar internacional que implementa los niveles inferiores del modelo OSI, en concreto, el nivel físico y el de enlace, sobre un canal inalámbrico.

Existen varios tipos de wifi, basados cada uno de ellos en un estándar IEEE 802.11 aprobado. Son los siguientes:

- Los estándares IEEE 802.11b, IEEE 802.11g e IEEE 802.11n que tienen una aceptación internacional debido a que la banda de 2.4 GHz está disponible prácticamente universalmente, con una velocidad de hasta 11 Mbps, 54 Mbps y 300 Mbps, respectivamente.

- En la actualidad ya se maneja el estándar IEEE 802.11ac, conocido por WIFI 5, que trabaja en la banda de 5 GHz y que disfruta de una operatividad con canales casi relativamente limpios. Es muy reciente esta banda, por lo que otras tecnologías como Bluetooth, ZigBee, RFID… todavía no la están utilizando, por lo tanto, casi no se tienen interferencias. Un contra es que su alcance es un poco más bajo, aproximadamente en un 10%.

A continuación, se hará una breve descripción sobre los estándares actualmente más utilizados:

- **802.11a.**

La primera revisión del estándar 802.11 nació allá por el año 1999 y opera sobre la banda de frecuencias de 5 GHz, con una velocidad máxima de 54 Mbps, seguía ofreciendo no obstante el problema de una excesiva atenuación en el aire debido a la banda en la que operaba, por lo que era necesario estudiar la expansión a nuevas bandas de frecuencias.

- **802.11b.**

La revisión 802.11b comenzó a gozar pronto de una gran aceptación en general debido a que al operar en la banda de 2,4 GHz se reducía la atenuación eliminando muchas interferencias mejorando la calidad de la señal Wi-Fi. La velocidad de transmisión que ofrecía quedó establecida en unos teóricos 11 Mbit/segundo, pero su principal lastre

fue que la cobertura en interiores quedaba limitada a un radio de 50 metros.

- **802.11g.**

Sin salir del ancho de banda de 2,4 GHz, el Wi-Fi g aprobado en el año 2003 igualaba en lo que respecta a la velocidad de transmisión máxima teórica de 54 Mbit/seg, al estándar *a* pero **mejoraba a su vez también la cobertura en interiores y exteriores** que ofrecía el estándar *b,* lo que provocó la popularización de equipos que la implantaron en todo el mundo.

- **802.11n**

Sin duda uno de los grandes puntos de inflexión en las conexiones inalámbricas, gracias a la implementación de las redes MIMO en el estándar Wi-Fi, ya que, aunque dichas antenas estaban ya presentes en equipos 802.11g, aquí comenzaron a normalizarse gracias a las ventajas de esta tecnología. Además de ser compatible con los estándares anteriores, con el Wi-Fi 802.11n se cubren velocidades de transferencia de entre 150 y 600 Mbps, garantizando velocidades de conexión de 300 Mbps estables en este último caso.
Por otra parte. la tecnología MIMO hace uso de varias antenas instaladas en el router para el envío y recepción de datos de manera simultánea. Aplicada a este estándar se ayuda a lograr coberturas de hasta 120 metros en interiores y 300 metros en exteriores.

- **802.11ac.**

El nuevo estándar WiGig trajo consigo las grandes velocidades a las conexiones inalámbricas y prueba de ello es el avance conseguido con el Wi-Fi 802.11ac. Gracias a la tecnología beamforming para focalizar las señales de radio, el alcance de estas redes inalámbricas es superior incluso a pesar de operar en la banda de 5 GHz y a velocidades mucho mayores gracias a las antenas múltiples –hasta un máximo de 4-. En este caso, la velocidad teórica queda fijada hasta en 1.300 Mbps.

- **802.11ad.**

La tecnología Wi-Fi bautizada como WiGig también de las más recientes y aunque todavía no es muy conocida para el gran público, pronto comenzará a serlo dado su increíble potencial. No solo porque ya se han anunciado equipos compatibles con la misma, como el Talon Ad7200 de TP-Link, sino sobre todo por las velocidades de hasta 4,6 Gbps que puede ofrecer en la banda de frecuencias de 60 GHz.

- **802.11ah.**

Esta revisión conocida por "HaLow" ha sido la última en llegar y lo hace dispuesta a plantear una seria alternativa al Bluetooth de cara a explotar el sector del Internet de las Cosas y encauzar las conexiones de los dispositivos conectados en el hogar el día de mañana. En este caso hablamos de un ancho de banda de 900 MHz por lo que ofrece un alcance mayor que las redes que operan sobre 2,4 GHz además de ayudar a aligerar dicha banda dirimiendo el tráfico de conexiones de los dispositivos conectados del hogar.

2.2.3 RFID.

La tecnología de Identificación por Radiofrecuencia RFID (RadioFrequency Identification) es, sin duda, una de las tecnologías de comunicación que ha experimentado un crecimiento más acelerado y sostenido en los últimos tiempos. Las posibilidades que ofrece la lectura a distancia de la información contenida en una etiqueta, sin necesidad de contacto físico, junto con la capacidad para realizar múltiples lecturas (y en su caso, escrituras) simultáneamente, abre la puerta a un conjunto muy extenso de aplicaciones en una gran variedad de ámbitos, desde la trazabilidad y control de inventario, hasta la localización y seguimiento de personas y bienes, o la seguridad en el control de accesos.

Son muchas las grandes compañías que apoyan la implantación y el uso sensato de la RFID, por lo que se puede esperar que su futuro sea muy prometedor. No hay duda de que se trata de una tecnología que puede aportar sustanciales ventajas en muchos ámbitos de aplicación. Sin embargo, el éxito final en la implantación de esta tecnología está sujeto a la superación de una serie de obstáculos, entre los que es necesario destacar los aspectos de seguridad y privacidad.

El funcionamiento y las características de sus principales elementos son:

- Etiquetas (tags o transpondedores): almacenan la información.

- Lectores (interrogadores): leen la información de las etiquetas, proporcionándoles energía para que éstas transmitan.

- Programadores: escriben información en las etiquetas.

- Middleware: gestiona el intercambio de información.

- Sistema de información: procesa la información del middleware y toma las decisiones oportunas.

Se consideran también los diferentes tipos de sistemas atendiendo principalmente a la frecuencia de trabajo, distinguiéndose:

- Los sistemas de baja frecuencia.

- Los sistemas de alta frecuencia.

- Los sistemas de ultra alta frecuencia.

- Los sistemas en frecuencia de microondas.

2.2.4 ZigBee.

ZigBee es un estándar que define un conjunto de protocolos para el armado de redes inalámbricas de corta distancia y baja velocidad de datos.

Opera en las bandas de 868 MHz, 915 MHz y 2.4 GHz y puede transferir datos hasta 250Kbps. Este estándar fue desarrollado por la Alianza ZigBee, que tiene a cientos de compañías desde fabricantes de semiconductores y desarrolladores de software a constructores de equipos OEMs e instaladores. Desarrolla un protocolo que adopta al estándar IEEE 802.15.4 para sus 2 primeras capas, es decir, la capa física (PHY) y la subcapa de acceso al medio (MAC) y agrega la capa de red y de aplicación.

El estándar ZigBee fue diseñado con las siguientes especificaciones:

- Ultra bajo consumo que permita usar equipos a batería.

- Bajo costo de dispositivos y de instalación y mantenimiento de ellos.

- Alcance corto (típico menor a 50 metros).

- Optimizado para ciclo efectivo de transmisión menor a 0.1%.

- Velocidad de transmisión menor que 250 kbps. Típica: menor que 20 kbps.

2.2.5 Bluetooth.

Bluetooth es un estándar global de comunicación inalámbrica establecido por la IEEE 802.15.1, donde se pueden realizar conexiones de Red Inalámbricas teniendo la posibilidad de transmitir voz, datos, imagen, multimedia entre diferentes dispositivos utilizando la tecnología de radio frecuencia de corto alcance.

Con esta nueva Tecnología nos podemos olvidar de los todos los tipos de cables (par trenzado, coaxial, fibra óptica, telefónicos, etc.) utilizados como medios físicos dependientes estando así propensos a posibles fallos, interferencia y gastos no necesarios. Otro de los aspectos relevantes es la configuración de los equipos, anteriormente se necesitaba que pudieran ser compatibles tanto hardware como software, este ya no sería un inconveniente, ya que se pueden conectar diferentes tecnologías y dispositivos.

En cuanto a la forma de administración de esta tecnología se puede mencionar que es sencilla, ya que solo se necesita hacer un enlace entre los dispositivos que se quieren conectar entre sí y sin necesidad de configurar los equipos solos se conectan, únicamente es cuestión de implementar la búsqueda de los dispositivos que se encuentran dentro de la zona de comunicación para que se pueda establecer la conexión.

La mayoría de las redes LAN que se implementan por lo regular en los hogares, pequeñas y medianas empresas tienen la necesidad de intercambiar información de la forma más sencilla que existe invirtiendo el mínimo de recursos y obteniendo el máximo de los resultados, en estas circunstancias la Tecnología Bluetooth ofrece una respuesta favorable a estas necesidades, ya que cumple con las expectativas de este tipo de redes LAN, las cuales si se implementaran se obtendrían mayores beneficios, menores costos, mayor seguridad, facilidad de comunicación, se eliminarían cables (mejor estética, sin cableado) y facilidad de sincronización de datos entre otros.

Bluetooth implementa Ondas de Radiofrecuencia de corto alcance, ya que es uno de los medios de transmisión sencillos, económicos, fáciles de configurar y administrar.

Los enlaces inalámbricos de más de dos dispositivos implementando la tecnología Bluetooth ocupando el mismo medio físico se llaman Piconet, cada dispositivo tiene una dirección única de 48 Bits, cuando un dispositivo busca o detecta a otro se le llama "Inquiry", para regular él trafica en el canal, uno de los dispositivos se le llama "Maestro" quien es el que establece la conexión y todos los demás dispositivos son llamados "Esclavos". Este tipo de red opera en un ambiente Multiusuario y esto es transparente para ellos, como todo enlace es codificado y protegido contra interferencia y perdida de enlace, esta tecnología se puede considerar como una Red Inalámbrica de corto alcance y muy segura.

El protocolo que se implementa es el de Bandabase el cual provee canales de transmisión para voz y datos, siendo capaz de soportar un enlace asíncrono de datos y hasta tres enlaces de voz asíncronos, esto se combina con las técnicas de circuitos y paquetes para asegurar que lleguen en orden.

Algunas de las características de la tecnología Bluetooth son las siguientes:

- Frecuencia: 2.4 GHz.

- Tecnología: Spread Spectrum.

- Potencia de transmisión: 1mW para 10 metros, 100mW para 100 metros.

- Canales máximos de voz: 3 por piconet.

- Canales máximos de datos: 7 por piconet.

- Velocidad de datos: 721 Kbps por piconet.

- Cobertura: 10 Metros.
- Número de dispositivos: 8 por piconet y hasta 10 piconet en 10 metros.

- Alimentación: 2.7 Voltios.

- Consumo de potencia: Desde 30µA a 30µA transmitiendo.

- Interferencia: Es mínima, se implementan saltos rápidos en frecuencia de 1600 veces / segundo.

Por otro lado, como se puede observar las características que componen esta nueva tecnología son muy pocas en comparación a las necesidades que actualmente hay en la implementación de Redes Inalámbricas, ya que se requiere mayor velocidad de transmisión, un número mayor de computadoras conectadas, así como, una cobertura más amplia para empresas grandes, una de las implementaciones para esta tecnología sería establecer redes LAN, conexiones pequeñas, funcionales y sencillas donde no sea tan importante el ancho de banda.

Más, sin embargo, bluetooth ha estado cambiando de versión ampliando sus características para mejorar su desempeño en cuanto a velocidad de transmisión de datos, cobertura, numero de dispositivos, entre otros.

En cuanto al funcionamiento, la tecnología Bluetooth funciona de la siguiente manera:

Cada dispositivo debe de contar con un microchip "CMOS" (tranceiver) que transmite y recibe la señal estimada en frecuencia de 2.4 GHz, cada dispositivo como se menciona anteriormente contiene 48 Bits, trabaja dentro de un rango de 10 metros teniendo una velocidad de transmisión de datos de hasta 3Mbps en la versión 2 de Bluetooth con una frecuencia de saltos que permite comunicarse en áreas donde existe mucha interferencia electromagnética utilizando esquemas de encriptación y verificación.

Cuando se lleva a cabo la transmisión de datos se hace mediante la siguiente forma: un paquete puede ser intercambiado en cada slot entre la unidad maestro y uno de los esclavos, cada paquete contiene 72 Bits de código de acceso que corresponde a la identidad del maestro, la cabecera del paquete contiene 54 Bits contiene información de control como la dirección de control de acceso al medio (MAC), tipo de paquete, bits de control de flujo, el esquema ARQ (Repetición de Transmisión Automática) y un error de cabecera, posteriormente de 0 a 2745 Bits se encuentran los datos los cuales pueden cubrir uno, tres o cinco slots.

Dentro de la importancia que tiene actualmente la tecnología bluetooth es gracias a su seguridad, existen diferentes formas en las que se puede romper la integridad de la seguridad de transferencia de datos dentro de la telefonía celular, una de ellas es el Bluejacking, el cual consiste en el envío anónimo de tarjetas de visita tecnológica inalámbrica, por lo general contienen un mensaje de tono ligero o coqueto, en lugar del nombre y el número de teléfono del remitente.

Otra de las formas es llamada Bluebugging, el cual consiste en que personas con los conocimientos necesarios obtienen el acceso a las funciones del teléfono móvil a través de la tecnología inalámbrica Bluetooth (esto sin que el usuario sea notificado o alertado de ello).

El Bluesnarfing, es el procedimiento mediante el cual los piratas informáticos acceden a los datos almacenados en un teléfono con tecnología bluetooth sin alertar al usuario que se ha establecido una conexión con su dispositivo.

El Gusano Cabir, es un software conocido también como "malware", este se instala en el teléfono y sé auto envía a otros dispositivos mediante Bluetooth, para que el usuario se vea afectado debe de aceptar el gusano e instalarlo de forma manual dentro del dispositivo.

Sin embargo, estas invulnerabilidades se reparan con las actualizaciones de esta tecnología.

Por último, se pueden concretar las siguientes ventajas de esta tecnología:

- Comunicación sencilla y cómoda, sin la necesidad de estar lidiando con los cables.

- Se puede implementar en todo el mundo gracias al estándar al que pertenece, IEEE 802.15.1 sin la necesidad de usar controladores para los diferentes dispositivos.

- Bajo consumo de energía, por el bajo consumo de frencuencia..

- Mínimo costo (imprescindible en este proyecto).

- Fiabilidad y facilidad de uso.

- Se ha implementado en todo el mundo la frecuencia que utiliza, la 2.4GHz, no requiere licencia y puede ser utilizada con facilidad.

- Una de las ventajas más significativa es que está disponible en casi todos los entornos de la actualidad, teléfonos móviles, automóvil, instrumentos médicos, redes inalámbricas…

2.3 Tecnología inalámbrica óptima.

La tecnología que se va a adoptar en este proyecto será seleccionada con respecto a las ventajas y desventajas que se pueden encontrar en un entorno real, dichas tecnologías han sido mencionadas en el apartado anterior 2.2Tecnologías inalámbricas.

Encapsulamos las diferencias en una tabla, cuyas diferencias son mucho más visibles e intuitivas:

Tecnología	WIFI	WIMAX	ZigBee	Bluetooth	RFID
Alcance m.	50	50 Km	10 – 75	50	0.01 - 10
Banda de transmisión	2,4 -2,4835 Ghz	5 x Ghz	868-870 Mhz. 2,4 – 2,4835 Ghz.	2.4 GHz.	125 Khz – 5.4 Ghz
Velocidad de transmisión	300 Mbps	124 Mbps	250 Kbit/s	1 Mbps	200 Kbps
Licenciada	No	Si	Si/No	No	
Implementación	Alta	Medio – Bajo	Bajo	Alto	Medio
Costo	Bajo	Medio	Bajo	Bajo	Medio

Tabla 2 Tecnologías Inalámbricas.

La tecnología RFID ha sido descartada porque no se estaría resolviendo uno de los problemas que se pretende solventar, que es el acercamiento a las arquetas (por ejemplo, las soterradas). Tanto el WIFI como el WIMAX son descartadas por el alto consumo energético

y la cobertura que pudieran tener a campo abierto para cada una de las arquetas en las que se implantaría el dispositivo.

Por tanto, según las comparaciones anteriores y la información que plantea la tabla, se elige como tecnología más óptima, eficiente y barata el Bluetooth. Una de las ventajas más significativas es que está mucho más implementada en cualquier dispositivo que en la tecnología ZigBee, que pierde en este sentido. También es muy barata, una de las razones por las que está muy extendida además de un soporte mucho mayor a otras de las opciones planteadas.

2.4 Dispositivos móviles.

En este apartado se detalla cada una de las herramientas que han sido utilizadas en este proyecto para la aplicación Android (más adelante se explica la razón de la utilización de este sistema operativo).

Tanto los dispositivos móviles como tablets o teléfonos móviles son la tecnología indiscutiblemente más extendida y utilizada en el mundo entero. A pesar del pequeño tamaño que pueden tener actualmente pueden tener una capacidad de procesamiento casi igual a la de un ordenador, lógicamente con algunas limitaciones inevitables.

Tanto en un teléfono móvil como en una Tablet tienen una serie de competencias básicas que se detallan a continuación.

Los teléfonos móviles en la actualidad no sólo sirven para enviar SMS o hacer llamadas telefónicas, si no que los llamados smartphones son capaces de interactuar mediante la tecnología bluetooth con otros dispositivos, llevar una agenda virtual e incluso mensajería instantánea con cualquier persona del mundo que disponga de otro dispositivo receptor wifi.

En el caso de las tablets tiene las mismas competencias exceptuando en su mayoría las llamadas telefónicas, pero en este caso serán más útiles gracias a su tamaño para distintos trabajos como hacer una agenda, o redactar un trabajo final de carrera con un teclado externo.

Para hacer posible todas estas funcionalidades, están conformados con:

- **Sistema operativo:** Es el encargado de ejecutar todas las aplicaciones y redireccionar todos los procesos internos.

- **Aplicaciones:** Son las encargadas de gestionar las tareas que le han sido acometidas, como por ejemplo encender la luz led para ver en ambientes poco luminosos o para hacer una foto. En este mismo proyecto se lleva a cabo una aplicación móvil.

- **Acceso a internet:** Se pueden conectar a internet gracias a las antenas WIFI que llevan incorporadas y al software que procesa sus peticiones. Además de WIFI también se puede tener acceso a internet mediante 3G o 4G, en casi cualquier parte del mundo.

- **Localización:** Gracias a los avances en las tecnologías de geolocalización se pueden geolocalizar los móviles en tiempo real, aprovechando esto para por ejemplo una aplicación que te guía a cualquier punto del mundo desde tu posición actualizada en tiempo real.

- **Interconectividad:** En todos los smartphones actuales, se cuenta con la conectividad Bluetooth, por ello es una de las razones por las que se escoge dicha tecnología para este proyecto. También se suele contar con un sensor de infrarrojos, actualmente usado sobre todo para la manipulación de televisiones (como un mando a distancia), aires acondicionados y otros aparatos electrónicos.

2.5 Sistemas operativos móviles.

En este apartado se van a detallar los tres mayores sistemas operativos actualmente más utilizados en la industria del teléfono móvil o smartphones y Tablets.

Un sistema operativo móvil es un conjunto de programas de bajo nivel que permite la abstracción de las propiedades del hardware específico del teléfono móvil y provee servicios a las aplicaciones móviles, que se ejecutan sobre él.
Al igual que los PCs que utilizan Windows o Linux, los dispositivos moviles tienen sus sistemas operativos como Android, iOS, Windows Phone, etc.

Los sistemas operativos móviles son mucho más simples y están más orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos.

Según el servicio de estadísticas NetMarketShare, la cuota de mercado

de sistemas operativos móviles a principios de 2017 es el siguiente:

- Android 66,71 % (en países como España las diferencias son más significativas, donde Android tiene más del 90% de la cuota de mercado).
- iOS 29,55 %.
- Windows Phone 1,41 %.
- BlackBerry OS 0,37 %.

Se van a comentar las características principales de los tres sistemas operativos más utilizados:

2.5.1 Android

Android Inc. es la empresa que creó el sistema operativo móvil. Se fundó en 2003 y fue adquirida por Google Inc. en el año 2005 y en 2007 fue lanzado al mercado.

Originalmente era un sistema pensado para las cámaras digitales profesionales, pero fue modificado por Google para ser utilizado en dispositivos móviles como los teléfonos inteligentes y tablets.

Cuenta con el mayor número de instalaciones de smartphones en todo el mundo y está basado en el núcleo Linux. Las aplicaciones para Android se escriben y desarrollan en Java aunque con unas APIs propias.

En 2007 Google fundó la Open Handset Alliance formada por un grupo de 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles. Juntos desarrollaron Android, la primera plataforma móvil completa, abierta y libre.
Algunos de sus miembros son Google, HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia y Wind River Systems.

Aunque el sistema operativo Android es software libre y de código abierto, en los dispositivos vendidos, gran parte del software incluido es software propietario y de código cerrado.

2.5.2 IOS

iOS (anteriormente denominado iPhone OS) es propiedad de Apple Inc. Tiene la segunda mayor base de smartphones instalada en todo el mundo después de Android.

Es de código cerrado y propietario y construido a partir de Darwin, o lo que es lo mismo, el kernel del sistema operativo de Apple, Mac OS X.

iOS es el sistema operativo que da vida a dispositivos como el iPhone, el iPad, el iPod Touch o el Apple TV.

2.5.3 Windows Phone

Windows 10 Mobile (anteriormente llamado Windows Phone) es de Microsoft, diseñado para teléfonos inteligentes y tabletas.

Es de código cerrado y propietario y utiliza como núcleo Windows NT.

En febrero de 2010 se dio a conocer Windows Phone que integra servicios de Microsoft como OneDrive y Office, Xbox Music, Xbox Vídeo, juegos Xbox Live y Bing, pero también se integra con otros servicios que no son de su propiedad, como Facebook y cuentas de Google.

A principios de 2015, Microsoft anunció que la marca Windows Phone sería reemplazada por Windows 10 Mobile con el objetivo de lograr una mayor integración y unificación con su homólogo para PCs Windows 10, y proporcionar una plataforma para smartphones y tablets con tamaños de pantalla de 8 pulgadas.

2.5.4 Selección del sistema operativo.

Como anteriormente, una tabla para ver las diferencias entre los tres sistemas operativos:

Sistema Operativo	Android	Ios	Windows Phone
Licencia	Libre y de código abierto, pero por lo general se incluye con aplicaciones y drivers propietarios.	Propietaria excepto para componentes de código abierto.	Propietaria
Lenguajes de programación	C, C ++, Java.	C , C ++ , Objective-C , Swift.	. NET C#, VB.NET, Silverlight, native C/C++, WinRTP (XMLA), DirectX
Coste del desarrollo para el OS móvil	Gratis	Gratis con xcode 7	Gratis
Coste para publicar en su tienda oficial	US$25 una vez por individuo	US$99 al año	US$19, una vez por un individuo; y $99 para una cuenta de la compañía
Navegadores Web disponibles	Muchos (entre ellos Google)	Bing, Google, Yahoo! Search, DuckDuckGo	Muchos (entre ellos Bing)

Como se puede observar, cualquiera de las tres opciones es aceptable para este proyecto, sin embargo, cabe mencionar que android es el más utilizado con bastante diferencia en los dispositivos móviles, además de que IOS no es código totalmente abierto y se necesitan una serie de requisitos para poder programar correctamente una aplicación en IOS.

Según los datos de principios de 2017, estos son los porcentajes de la cuota de mercado actual de cada uno de los sistemas operativos.

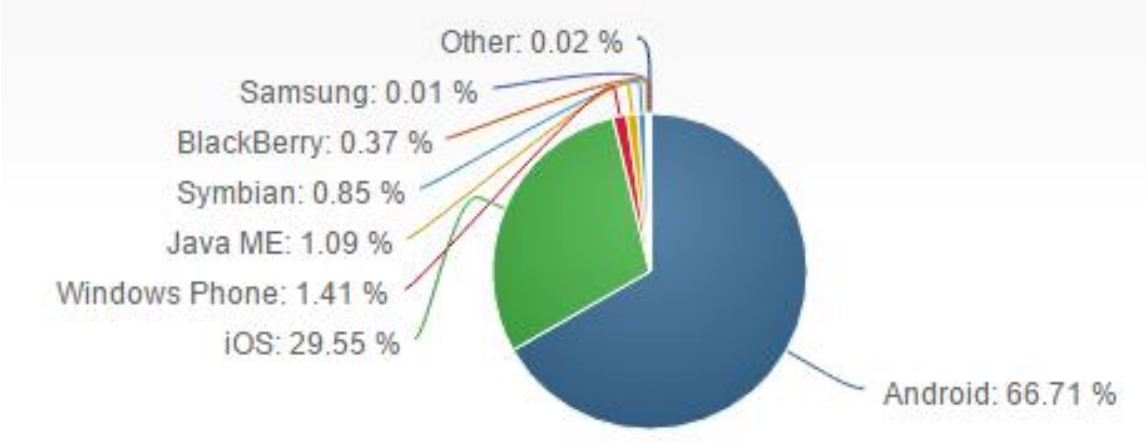


Ilustración 1 Gráfico Cuota Sistemas Operativos.

Corroboramos, por tanto, que Android será la mejor opción para desarrollar la aplicación correspondiente para este proyecto.

2.6 Plataforma hardware.

En esta sección se explicará cual ha sido el desarrollo hardware pertinente para el desarrollo del dispositivo físico que será parte de la solución que se ha llevado a cabo en el proyecto.
En concreto se ha utilizado la plataforma mbed con su plataforma BLE nano (similar a las plataformas que contiene el famoso entorno de Arduino IDE) ya que la plataforma ofrece un framework de trabajo facilitando así el desarrollo software permitiendo centrarse en la funcionalidad que se desea implementar.

Arduino también se podría utilizar para la programación de este dispositivo BLE nano, pero se ha escogido la plataforma Mbed por el soporte que tiene para dicho dispositivo.

Para la elección de la plataforma, para este proyecto, vamos a tener en cuenta 2 requisitos principalmente: tamaño del hardware a desarrollar y el precio del SoC.

Por todo ello, se ha escogido la placa de desarrollo llamada BLE nano de ReadBearLab.

Para profundizar un poco más sobre la plataforma que se va desarrollar, se pueden comentar las siguientes características:

Es ideal para proyectos de desarrollo o de prototipado especialmente en el campo del Internet de las cosas.
Su puerto USB MK20 USB permite instalar su firmware muy fácilmente.
El elemento central del BLE nano es el Nordic nRF51822, un Cortex-M0 ARM SoC con BLE capaz de operar a 16MHz con un consumo muy bajo.
El RedBearLab BLE Nano soporta también diferentes aparatos wireless con sistemas operativos iOS 7/8, Android 4.3 o superior, y Windows Phone 8.1.

El RedBearLab BLE Nano puede operar a partir de 1.8 V hasta 3.3V lo que le permite trabajar con una amplia gama de componentes electrónicos. Además, a través de su pin VIN, acepta entradas de 3.3V hasta 13V, aunque la placa regula la potencia a 3.3V.

En cuanto a las características técnicas tenemos:

- Tamaño de 18.5mm x 21.0mm.
- Nordic nRF51822 ARM Cortex-M0 SoC soporta a la vez papeles BLE centrales y periféricos.
- 2.4 GHz transceiver.
- Consumo ultra bajo.
- Soporte de 1.8V a 3.3V.
- Desarrollo en Mbed.org, GCC, Keil o Arduino.
- Firmware fácilmente instalable con la placa MK20 USB.

Ilustración 2 BLE Nano.

2.6.1 Mbed

Mbed es una plataforma de prototipado rápido y experimentación con microcontroladores ARM Cortex-M3 y ARM Cortex-M0 de 32 bits. Provee a los desarrolladores experimentados una plataforma productiva para realizar pruebas conceptuales y prototipos, mientras que a los principiantes sin experiencia previa les provee una forma accesible de realizar proyectos con microcontroladores de 32 bits mediante el acceso a las librerías, tutoriales y ejemplos, además de la comunidad online de mbed.

- **Hardware del Mbed.**
    - Microcontrolador ARM Cortex-M3.
    - Microcontrolador ARM Cortex-M0.

- **Sofware del Mbed.**
    - Entorno de Desarrollo Integrado (EDI) "en la nube".
    - Compilador, librerías y almacenamiento de proyecto "online".

- **Características Mbed ARM Cortex-M3.**
    - Microcontrolador NXP LPC1768.
    - 96MHz, 32 KB RAM, 512KB FLASH.

- Ethernet, USB Host/Device, 2xSPI, 2xI2c, 3xUART, CAN, 6xPWM, 6xADC, E/S de propósito general (GPIO).
- Empaque DIP 40-pin 0.1” (breadboard), 54x26mm.
- Alimentación de voltaje de 5V USB o 4.5 – 9V.
- Programador flash incorporado tipo “arrastra y suelta” USB (como un pen drive USB).
- Kit de desarrollo de software (SDK) de alto nivel en C/C++.
- Compilador Oline “liviano”
- Librerías y proyectos publicados como referencia inicial.

- **Características Mbed ARM Cortex-M0.**
  - Microcontrolador NXP LPC11U24.
  - 48MHz, 8 KB RAM, 32KB FLASH.
  - USB Host/Device, 2xSPI, I2c, UART, 6xADC, E/S de propósito general (GPIO).
  - Empaque DIP 40-pin 0.1” (breadboard), 54x26mm.
  - Alimentación de voltaje de 5V USB o 4.5 – 9V o baterías de 2.4 - 3.3V.
  - Programador flash incorporado tipo “arrastra y suelta” USB (como un pen drive USB).
  - Kit de desarrollo de software (SDK) de alto nivel en C/C++.
  - Compilador Oline “liviano”
  - Librerías y proyectos publicados como referencia inicial.

- **¿Por qué Mbed?**
    - Es ideal para iniciarse con microcontroladores ARM de 32 bits y lenguajes C/C++.
    - No se necesita un programador adicional, el hardware está listo desde el primer momento para usarse.
    - EDI y los proyectos disponibles online en cualquier parte (con acceso a internet).
    - Bueno para prototipos y experimentar.

- **Desventajas de Mbed**
    - No cuenta con una interfaz JTAG para la depuración.
    - No se puede acceder a tus proyectos sin acceso a internet.

2.6.2 Altium Designer

En esta sección se van a explicar los motivos de la utilización de Altium Designer para hacer el diseño de la PCB.

Altium Designer proporciona la tecnología de diseño de PCB más avanzada del mercado en un entorno de diseño unificado. Permite sacar el máximo potencial y ahorrar tiempo gracias a sus potentes herramientas de automatización de procesos y de mejora de la productividad. Además, las intuitivas funcionalidades que Altium Designer incluye para el trabajo colaborativo permite a todos los componentes del equipo compartir ideas y diseños, asegurándose así trabajar en la última versión del PCB en desarrollo.

Altium Designer integra:

- Entorno de diseño unificado.
- Rutado interactivo.
- Herramientas automáticas de diseño para alta velocidad.

- Editor integrado de PCB 3D y soporte Rígido-Flexible.

- Cómoda colaboración ECAD/MCAD.

- Fácil creación y control de versiones.

- Fácil reutilización de diseños, componentes y ajustes.

- Creación y actualización automática de la documentación del proyecto.

- Extensa biblioteca editable que permite trabajar desde un entorno unificado y centralizado.

- Reglas de diseño totalmente configurables.

- Enlaces en tiempo real con información de los proveedores (precio, disponibilidad…).

Todo ello, combinado con la capacidad de gestión de datos de diseño, hace de Altium Designer sea la solución o la opción más adecuada para el diseño de la PCB.

Altium Designer incluye los editores y motores de software necesarios para llevar a cabo todos los aspectos del proceso de desarrollo de productos electrónicos. Toda la edición de documentos, recopilación y procesamiento se realiza en el entorno de Altium Designer.

Se integra perfectamente con otras herramientas de apoyo, como el place and route del fabricante de FPGA, o el software de simulación y síntesis HDL de otras compañías.

La base de Altium Designer es su plataforma de integración DXP (Design Explorer Integration Platform) que reúne los diferentes editores y motores de software de Altium Designer y proporciona una interfaz de usuario consistente a través de todas las herramientas y editores.

La interfaz del entorno podría ser la siguiente:

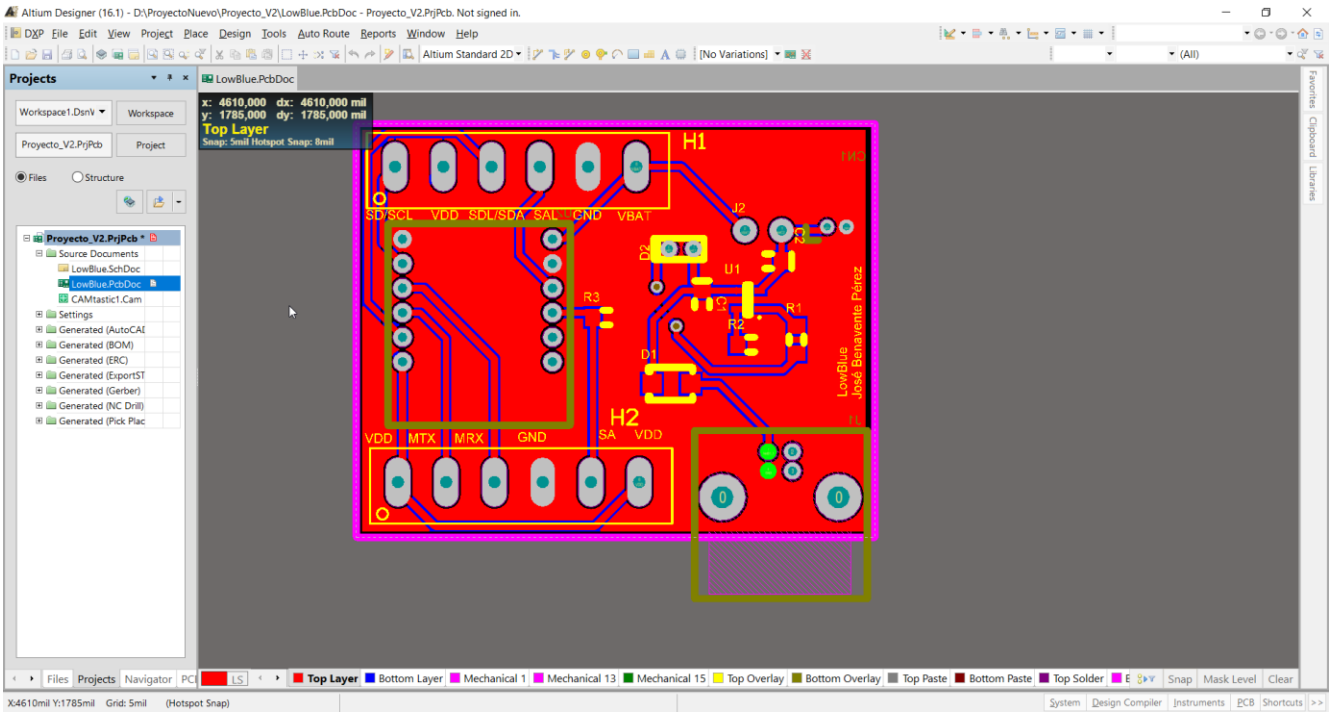


Ilustración 3 Altium Designer.

Otra de las características importantes es que está basado en la Plataforma de Integración DXP, tecnología única de Altium. DXP proporciona una plataforma común para todas las capacidades de Altium Designer y editores de diseño. Esto asegura la consistencia de interfaz a través de los diferentes editores y proporciona la centralización de documentos, proyectos y gestión de bibliotecas en todo el proceso de diseño.

Desde la definición del sistema, la captura de hardware, el diseño de PCB, FPGA, el desarrollo del software embebido y hasta la edición final del archivo CAM, todos los ingenieros con Altium Designer comparten un entorno común, trabajan en proyectos comunes y acceden a editores y capacidades de diseño comunes con el fin de cooperar y hacer frente a todos los desafíos en el diseño de productos electrónicos.

Para terminar, tiene una alta flexibilidad en la reutilización de diseños así como potentes herramientas de ruteado:

- Diseña con facilidad esquemas y ruta las placas más complejas con distintas opciones de rutado interactivo, autorrutado y con soporte para el diseño en alta velocidad.

- Gana tiempo al iniciar un nuevo proyecto gracias a las herramientas de reutilización incluyendo partes usada anteriormente (esquemáticos, librerías de componentes).

- Mantén el control absoluto sobre el proceso de diseño y comparte información con el área de producción de manera clara y sencilla gracias a las potentes herramientas que incluye Altium Deisgner.

- Ahorra tiempo en la creación de componentes gracias al extenso material para el diseño incluido, destacando los más de 300.000 componentes listos para ser usados.

Por todo ello, ha sido el software que ha sido utilizado para la creación de la PCB, ya que es uno de los entornos más profesionales y eficientes en el diseño de PCB.

2.7 Control de versiones

En este apartado, se va a especificar el software que ha sido utilizado para el control de versiones. En el caso de la programación en la plataforma Mbed del hardware, tiene su propio sistema de control de versiones, en la que se pueden crear "commits" y volver a una versión anterior en caso de bugs o errores en las posteriores versiones.

Sin embargo, también se pueden hacer controles de versiones en el diseño del hardware en Altium Designer, además del desarrollo software en la plataforma Android Studio con un software llamado SourceTree.

SourceTree simplifica la forma de interactuar con los repositorios de Git para que puedas centrarte en el código. En él se puede visualizar y administrar los repositorios mediante la sencilla interfaz de usuario de Git de SourceTree.

La interfaz de este programa es la siguiente:

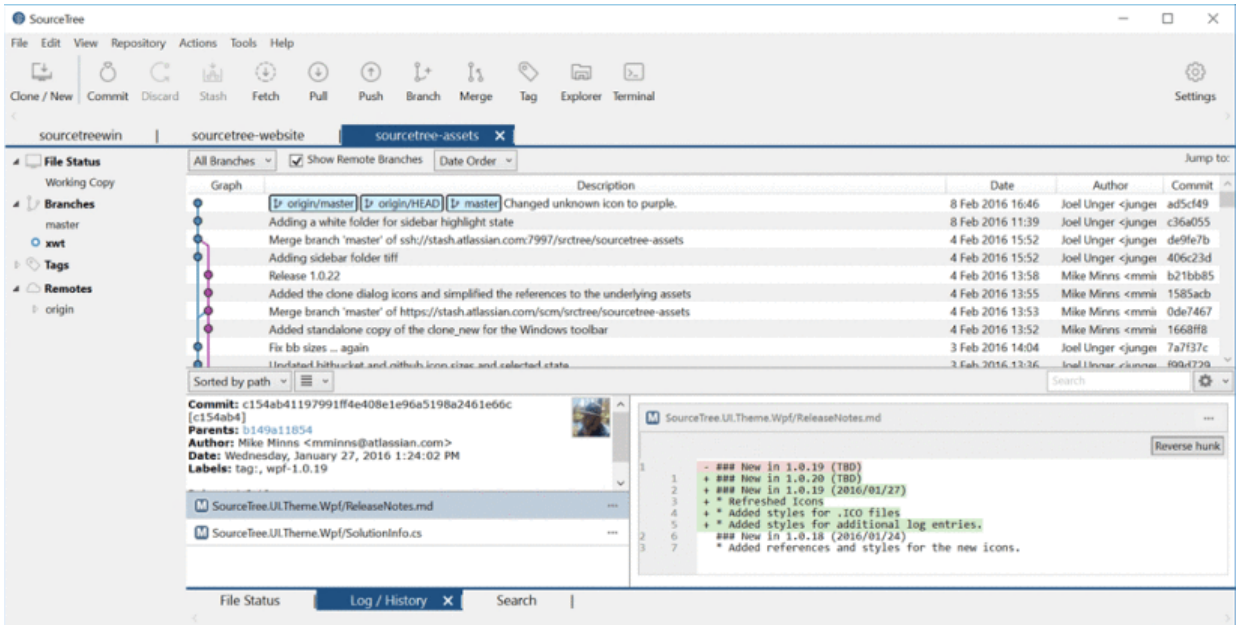


Ilustración 4 SourceTree.

Sin entrar en detalle, podemos destacar estas características principales:

- Anotar y comparar los cambios de código, línea a línea.
- Restablecer ficheros o todo el proyecto a cómo estaba en un momento concreto de su historial.
- Sincronizar el código con los cambios que haya realizado otra persona y viceversa.
- Resolver conflictos cuando se realizan cambios contradictorios.
- Mantener uno o más repositorios de código.
- Establecer flujos de trabajo a modo de ramas paralelas e independientes.

Aunque git suele ser utilizado para desarrollos software, también se pueden hacer controles de versión en el diseño hardware, los archivos de altium aunque no sea útil ver las diferencias entre código entre las versiones sí que se puede volver a una versión anterior, a la que se le pondrá una descripción de los cambios o cualquier información útil de la versión.

Por todo ello, ha sido el software que se ha utilizado para realizar el presente proyecto.

2.8 Desarrollo Software Android Studio.

Para el desarrollo de la aplicación en el SO Android se ha escogido el software "oficial" de programación Android, en concreto Android Studio.

Estas son las principales características de Android Studio:

- Soporte para programar aplicaciones para Android Wear.
- Herramientas Lint. Detecta el código no compatible entre arquitecturas diferentes o código confuso.
- Utiliza ProGuard, para poder optimizar y reducir el código del proyecto al exportar a APK, para dispositivos de gama con limitaciones.
- Nuevo diseño del editor con un soporte para la posible edición de temas.
- Actualizaciones frecuentes (diferentes canales).
- Nueva interfaz específica para el desarrollo en Android.
- Alertas en tiempo real de errores sintácticos, compatibilidad o rendimiento antes de acabar la aplicación.
- Vista previa, en diferentes tipos de proyectos y resoluciones.
- Posibilita la opción del control de versiones accediendo a un repositorio y poder descargar Mercurial, Git, Github o Subversion.
- Permite la importación de los proyectos realizados desde Eclipse.

Además de estas características tiene las siguientes ventajas y, como no, desventajas:

- Compilación rápida.
- Ejecución de la app en tiempo real gracias al emulador.
- Ejecución de la app directamente desde el móvil.

- No soporta el desarrollo para NDK, pero intellij con el plugin Android sí.

- Tiene renderizado en el tiempo real, layouts y puede hacer uso de parámetros tools.

- Funciona bien (sobre todo si usas versiones estables).

- Contiene todo lo necesario para desarrollar cualquier IDE.

- Es capad de asociar automáticamente carpetas y archivos con su papel en la aplicación, la creación de nuevas carpetas, borrado de archivos en values… esto es muy cómodo.

- (Desventaja) Los requisitos son un poco elevados (tendrás que tener una buena máquina para que te funcione bien el emulador). Pero esto hace que sea el mejor entorno para programar en Android, por lo que es necesario. Tira bastante del PC y gasta batería como consecuencia.

Son muchas las ventajas y características de Android Studio. Será la plataforma que se usará para el desarrollo de la aplicación ya que se considera la mejor opción.

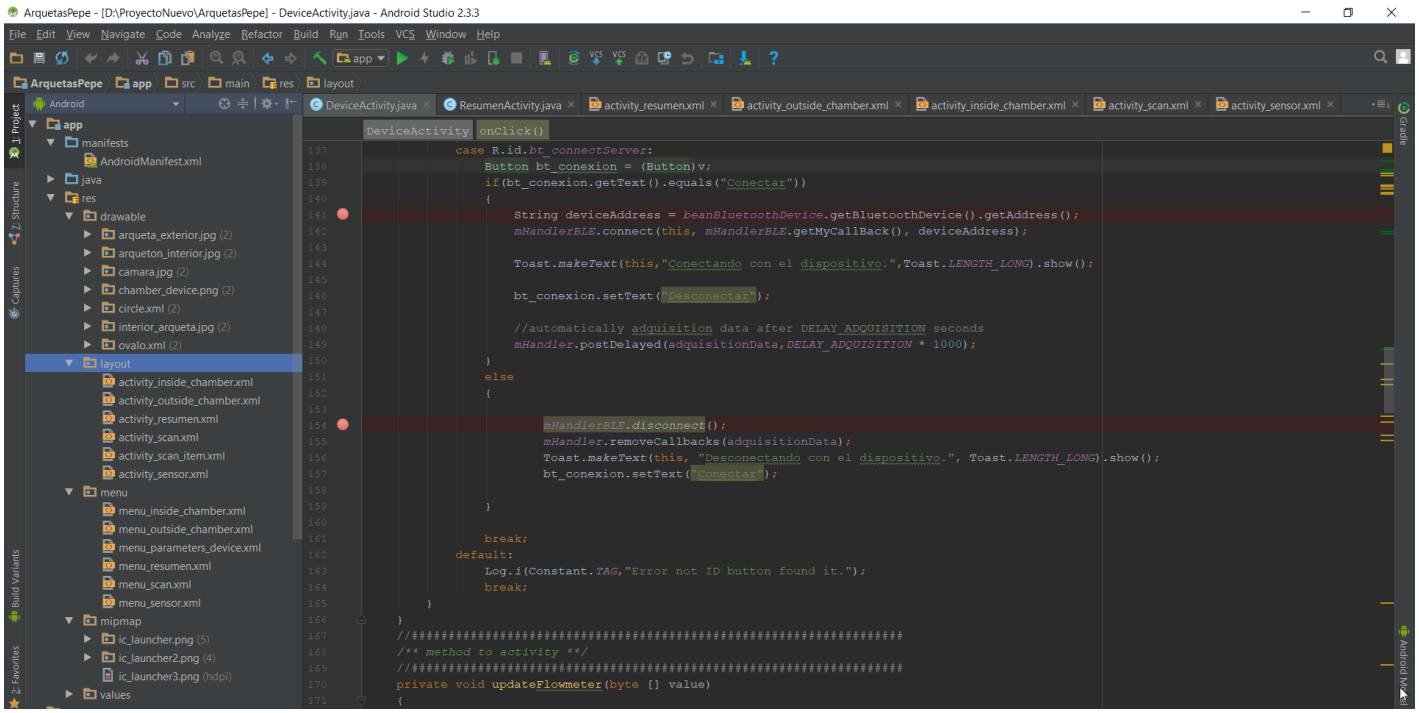


Ilustración 5 Android Studio

2.9 Diseño 3D de la carcasa en Tinkercad.

Para el diseño de la carcasa he utilizado la herramienta online Tinkercad.

Tinkercad es una sencilla herramienta de diseño y modelado 3D basada en navegador con una interfaz muy intuitiva y gracias a sus tutoriales es muy sencilla de utilizar para todo el público. Tinkercad permite a los usuarios diseñar en cuestión de minutos todo lo se pueda imaginar.

Algunas características de tinkercad son las siguientes:

- Se pueden crear formas, las formas son los bloques de construcción básicos de Tinkercad. Puedes utilizar cualquier forma para añadir o quitar material, así como importar o crear tus propias formas.

- Se pueden crear nuevos modelos con los que trabajar agrupando un conjunto de formas. Construye formas complejas y crea modelos altamente detallados.

- Se pueden crear formas vectoriales, además se pueden importar y extruir para crear modelos 3D. También se pueden importar archivos 3D externos, los cuales se convierten en formas de Tinkercad que se pueden editar.

- Tinkercad es compatible con todas las impresoras 3D del mercado que acepten formatos de archivo STL estándares. También se pueden descargar archivos VRML para imprimir en color.

- Gracias a la integración directa con los servicios de impresión de terceros más importantes, se puede diseñar en Tinkercad y luego recibir las impresiones en casa.

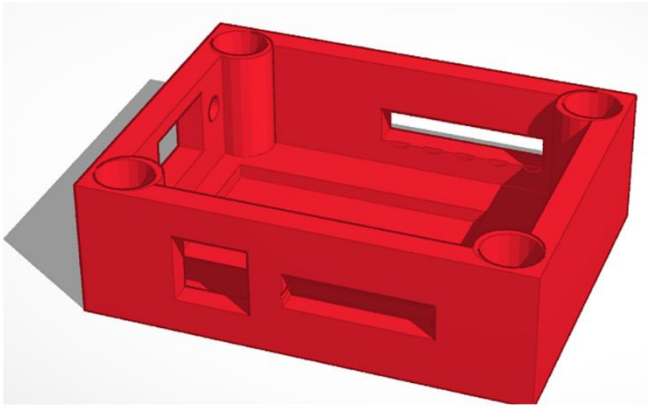
- La exportación de SVG en Tinkercad es fantástica para cortadores láser. El plano de trabajo hará un corte en el modelo, el cual después se puede cortar con láser.

- Tinkercad se ejecuta en cualquier navegador web compatible con HTML5 o WebGL en Windows, Mac o Linux. Los navegadores con el mejor rendimiento son Chrome y Firefox.

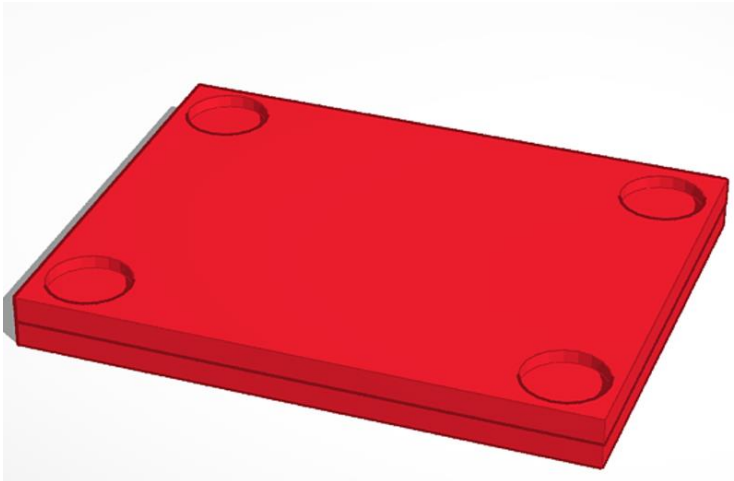
- Los diseños 3D que se crean con Tinkercad se almacenan en la nube, de manera que puedes acceder a ellos con facilidad desde cualquier lugar que tenga una conexión a Internet.

- El kernel de geometría realiza todas las operaciones de edición en un clúster informático de gran tamaño con muchos equipos, lo cual le da una gran velocidad.

- Diseño del cuerpo:



- Diseño de la tapa:



2.10 Arduino.

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

Por otro lado Arduino nos proporciona un software consistente en un entorno de desarrollo (IDE) que implementa el lenguaje de programación de arduino y el bootloader ejecutado en la placa. La principal característica del software de programación y del lenguaje de programación es su sencillez y facilidad de uso

¿Para qué sirve Arduino?

Arduino se puede utilizar para desarrollar elementos autónomos, conectándose a dispositivos e interactuar tanto con el hardware como con el software. Nos sirve tanto para controlar un elemento, pongamos por ejemplo un motor que nos suba o baje una persiana basada en la luz existente es una habitación, gracias a un sensor de luz conectado al Arduino, o bien para leer la información de una fuente, como puede ser un teclado, y convertir la información en una acción como puede ser encender una luz y pasar por un display lo tecleado.

El HW de Arduino es básicamente una placa con un microcontrolador. Un microcontrolador (abreviado µC, UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Características de un Microcontrolador:

- Velocidad del reloj u oscilador.
- Tamaño de palabra.
- Memoria: SRAM, Flash, EEPROM, ROM, etc..
- I/O Digitales.
- Entradas Analógicas.
- Salidas analógicas (PWM).

- DAC (Digital to Analog Converter).
- ADC (Analog to Digital Converter).
- Buses.
- UART.
- Otras comunicaciones.

Un Arduino físicamente es tal que así:

Ilustración 6 Placa Arduino.

2.11 Django.

Django es un framework para aplicaciones web gratuito y open source escrito en Python. Es un WEB framework - un conjunto de componentes que te ayudan a desarrollar sitios web más fácil y rápidamente.

Cuando se construye un sitio web, frecuentemente se necesita un conjunto de componentes similares: una manera de manejar la autenticación de usuarios (registrarse, iniciar sesión, cerrar sesión), un panel de administración para tu sitio web, formularios, una forma de subir archivos, etc.

Por suerte, hace tiempo varias personas notaron que los desarrolladores web enfrentan problemas similares cuando construyen un sitio nuevo, por eso juntaron conocimmientos y crearon frameworks (Django es uno de ellos) que ofrecen componentes listos para usarse.

¿Por qué se necesita un framework?

Para entender para que es Django, necesitamos mirar más de cerca a los servidores. Lo primero es que el servidor necesita saber que quieres que que se sirven en una página web.

Imaginemos un buzón (puerto) el cual es monitoreado por cartas entrantes (peticiones). Esto es realizado por un servidor web. El servidor web lee la carta y envía una respuesta con una página web.

Pero cuando quieres enviar algo tienes que tener algún contenido y Django es algo que ayuda a crear el contenido.

¿Qué sucede cuando alguien solicita una página web de tu servidor?

Cuando llega una petición a un servidor web, ésta es pasada a Django, el cual intenta averiguar lo que realmente es solicitado. Toma primero una dirección de página web y trata de averiguar qué hacer. Esta parte es realizada por el urlresolver de Django (hay que tener en cuenta que la dirección de un sitio web es llamada URL - Uniform Resource Locator - así que el nombre *urlresolver* tiene sentido). Este no es muy inteligente - toma una lista de patrones y trata de encontrar la URL.

Django comprueba los patrones de arriba hacia abajo y si algo coincide entonces Django le pasa la solicitud a la función asociada (que se llama *vista*).

En la función de *vista* se hacen todas las cosas interesantes: podemos mirar a una base de datos para buscar alguna información. ¿Tal vez el usuario pidió cambiar algo en los datos? " La *vista* puede comprobar si tienes permitido hacer eso, entonces actualizar la descripción del trabajo para el usuario y devolver un mensaje: "¡hecho!". Entonces la *vista* genera una respuesta y Django puede enviarla al navegador del usuario.

Capítulo 3

3. Análisis del proyecto, problema – solución planteada.

3.1 Introducción

El presente proyecto, tiene como objetivo mejorar el sistema de control y monitorización de la red de distribución de aguas.

Se han desarrollado una programación del hardware en la plataforma 2.6.1 **Mbed**, además de una aplicación en el SO de Android en el entorno de programación en Desarrollo Software Android Studio y se ha desarrollado un diseño de PCB en 2.6.2 Altium Designer.

El proyecto surge por la necesidad de tener una mayor eficiencia, comodidad y mayor control de los recursos acuíferos para la realización de acciones propias de la distribución del agua y en la medida de la posible para el control más rápido de fallas y controles de calidad de cada una de las arquetas.

El inicio de la idea es gracias al problema continuo de los controles de las arquetas manualmente a manos de los operarios, por lo que se hace especial empeño en la resolución de los problemas actuales de estos operarios de la mano de la tecnología y obteniendo una infinidad de aspectos positivos en su implementación.

Se detallará a continuación la problemática actual con más detalle.

3.2 Descripción de la problemática.

Como en todo dispositivo con intención de venta, primero debe haber un problema que solucionar, o una necesidad que cubrir.

El problema se encuentra en este caso en el sector agrícola, donde se pretende implantar la tecnología actual para optimizar y hacer más eficiente los recursos que se pretenden explotar en dicho sector.

La red de distribución de aguas está compuesta por los siguientes componentes:

- Tuberías principales, secundarias y terciarias.
- Tanques de almacenamiento intermediarios.
- Estaciones de bombeo.
- Válvulas y sensores que permiten obtener la información de las tuberías y permiten operar en la red, además de poder sectorizar los suministros por roturas o por conveniencia según la situación que se aborde.

A estos elementos se accede a través de las llamadas arquetas, las cuales se encuentran a veces soterradas y otras tantas tienen difícil acceso, por tanto, esto genera una de las problemáticas que se desea solucionar.

Un ejemplo de una arqueta que se encuentra soterrada sería la siguiente:

Ilustración 7 Arqueta Soterrada.

También están muchas otras dentro de "casetas" en las que mediante el deterioro de las puertas y su posición en el campo hace que se atasquen a menudo y sean de "difícil" acceso, por lo que cuando un operario se dispone a revisar una a una las arquetas se hace muy tedioso, ya que no siempre se cuentan con un bajo número de ellas.

Ilustración 8 Caseta Arqueta.

Para describir mejor la problemática, se van a detallar los procesos que se llevan a cabo en la inspección de arquetas:

i. Desplazamiento a la localización de las arquetas:

El operario deberá trasladarse a cada una de las arquetas que tiene que inspeccionar, en ocasiones entre ellas tienen separaciones de hasta km.

La posición de la arqueta se la comunica el supervisor de zona, por lo que tendrá que disponer de un vehículo para trasladarse hasta la posición.

ii. Revisión del estado de las instalaciones:

En este segundo procedimiento, se llevan a cabo las revisiones de las piezas de las arquetas, para poder prever las fallas que se puedan ocasionar en el futuro o las roturas y desperfectos que ya hayan ocurrido.

Primero se revisarán los elementos exteriores de la arqueta que son los siguientes:

- Acceso a la ubicación.
- Perímetro arqueta.
- Puerta de acceso.
- Cubierta.
- Parámetros Verticales Internos.
- Parámetros Verticales Externos.
- Ventilación lateral.
- Ventilación superior.
- Escalera.
- Distancia reglamentaría de los elementos.

Y una comprobación visual de los elementos interiores donde se comprobarán los elementos interiores:

- Ventosas.
- Válvulas.
- Juntas de unión.
- Manómetros.
- Contadores.

En este proceso de revisión, se deberá apuntar todo en papel que previamente llevaría el operario. El papel tendrá tres estados en cada uno de los elementos que tiene que revisar, que son: "Buen estado" "Aceptable" y "Necesita Reparación".

Obviamente, según el criterio personal del operador marcará la opción que crea conveniente a cada uno de los elementos.

iii. Toma de datos de los valores leídos en sensores y aparatos de medida.

En este procedimiento de toma de datos, el operador mirará los valores de los sistemas de medida que tenga incorporados la arqueta en cuestión. Apuntará cada uno de los valores en papel, apuntando las referencias de arquetas (para saber qué arqueta es).

Algunos de estos instrumentos de medida pueden ser un caudalímetro, un sensor de presión o barómetro, la posición de las válvulas abiertas o cerradas y otros instrumentos de medida.

iv. Elaboración del informe final de la jornada.

Una vez finalizados los otros tres primeros procedimientos, el operario tendrá que hacer el traspaso de datos que ha tomado en los papeles de anotación de la revisión y toma de datos, al ordenador.

v. Resumen de la problemática y soluciones.

En cuanto a la revisión de cada una de las arquetas y su desplazamiento no hay mucho margen de mejora o implementación de tecnología, pero sí en la recogida de datos y en la elaboración de los informes.

En cada uno de estos procedimientos se pueden sacar varias problemáticas, que se pueden detallar en:

La recogida de datos en papel es una de las problemáticas, ya que, en campo abierto y apuntando a lápiz (para poder rectificar) es donde más fallos se producen. Al transferir los datos al ordenador se pueden apuntar datos erróneos, olvido o pérdida de los papeles, que un número se haya escrito ilegible o directamente se ha olvidado apuntar algún tipo de dato y debería volver físicamente a apuntar los datos olvidados, ocupando por dos veces tiempo y gasto de dinero innecesario. Este problema es solucionado porque los datos se introducirán en la aplicación móvil y ésta no avanzará en el proceso de recogida de datos si hay algún olvido de introducción de datos.

Las arquetas que se tienen un difícil acceso la solución que se aporta es la comodidad de no tener que entrar para ver de cerca los valores de los sensores, sólo hace falta hacer una comprobación visual del estado físico de la arqueta y sus imperfecciones ya que, ahora podemos ver los valores de los sensores en tiempo real gracias a la comunicación bluetooth entre el dispositivo móvil y el dispositivo de recogida de

datos, se puede verificar que es la arqueta que se necesita revisar gracias a su id.

Se pueden hacer fotos, para poder ver posteriormente la avería o la imperfección y evitar errores de olvido o mal interpretación de la avería, además de poder incrustar un comentario.

Al utilizar el dispositivo móvil, se podrán usar los datos a través de servidores en cualquier lugar, ya que, los informes se subirán en cuanto el dispositivo móvil logre conectarse a una red WIFI si en campo abierto no está disponible haciendo más versátil el sistema de verificación de datos.

Se podrán usar estadísticas que podrán ser usadas para ver donde ocurren más averías y poder hacer un examen visual más exhaustivo.

Capítulo 4

4. Plataforma hardware y arquitectura física de la solución.

4.1 Introducción.

En este capítulo se van a describir el diseño y fabricación de la plataforma hardware y la arquitectura física que se ha implementado en el presente proyecto.

4.2 Diagrama de bloques de la arquitectura.

Los sensores de caudal, presión y la válvula serán en principio los sensores de los que se van a recoger los datos para previamente ser procesados en la aplicación móvil.

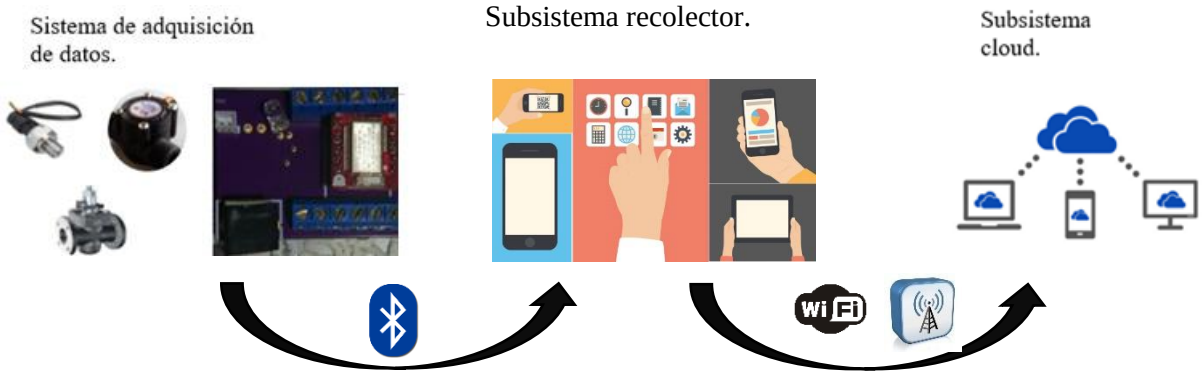


Ilustración 9 Diagrama de bloques.

En este diagrama, el primer subsistema de adquisición de datos son los sensores encargados de transferir la información mediante una comunicación, ya sea uart, modbus o con el procesado de una señal analógica. Los datos que son transferidos por los sensores son procesados y enviados por bluetooth a los dispositivos móviles en los que tengamos disponible la aplicación Android.

En el segundo subsistema, se recogen los datos enviados mediante la aplicación Android. La comunicación se trata de M2M por Bluetooth, es

decir, de los que vienen del subsistema de adquisición y balizamiento BLE. Una vez tramitados al dispositivo móvil los datos son guardados en el dispositivo y enviados por wifi posteriormente.

El tercer y último subsistema, consiste en la comunicación entre el dispositivo móvil y la nube. Los datos serán enviados en el momento en que el dispositivo móvil sea conectado a una conexión WIFI o redes móviles que proporcionen conexión a internet, una vez enviados a la nube los datos estarán en un almacenamiento confiable y con una alta seguridad. Estos datos podrán ser utilizados posteriormente por otros servicios.

Prosiguiendo en el detalle de los subsistemas que componen este proyecto, se explican más detalladamente a continuación.

4.3 Subsistema de adquisición de datos.

En este subsistema, se recolectarán los datos de los sensores que medirán las medidas de presión o caudal reales. Los sensores serán tanto analógicos como digitales.

Estos sensores según los niveles de presión o caudal crearán una señal de una tensión concreta, la cual servirá para medir o cuantificar la medida.

Para el sensor analógico se emplea la tecnología single ended, que consiste en aplicar una intensidad 4-20 mA al sensor, por tanto, el sensor responderá en estos niveles de intensidad. Se emplea esta tecnología por estas razones:

- Se puede detectar la rotura del cable cuando estamos por debajo de 4 mA. Los valores por debajo de 4 y por encima de 20 mA los podemos utilizar para la detección de fallos en la señal. La capacidad de diagnosis será una gran ventaja sin duda.

- Los transmisores de campo de 2 hilos, que suelen ser la mayoría, no necesitan ser alimentados pues lo hacen a través del propio lazo de 4-20 mA. Esta es una de las grandes ventajas sobre todo en instalaciones con muchos instrumentos. Si el punto

cero fuera de 0 mA no podríamos alimentar al transmisor cuando estuviéramos por debajo de unos 3,6 mA.

- El rango 4 – 20 mA puede transmitir los datos digitales HART a través de los mismos cables sin que haya ninguna interferencia entre ambas.

- Una señal de corriente es, en general, más inmune a los ruidos eléctricos que cualquier señal de tensión (0 – 10 VDC, 1 – 5 VDC) y además puede trabajar en distancias largas (más de 1 km si la alimentación nominal es de 24 VDC; podemos siempre subir un poco el nominal si tenemos distancias mayores).

- Detectar fallos o comprobar el lazo 4 – 20 mA es sencillo utilizando un polímetro normal. Además, no hay riesgo personal si tocamos el cable pues la tensión es de 24 VDC y el umbral de corriente peligrosa para el corazón está en 30 mA.

La válvula tendrá una señal digital (0-1) que será procesada por la aplicación Android para representar el valor de cerrado o abierto.

Por último, el manómetro se comunicará con el ARM Cortex M0 del BLE Nano por comunicación modbus, siendo el manómetro el esclavo y maestro el Ble Nano.

A continuación, se muestra un gráfico para entender mejor cómo están los dispositivos conectados:

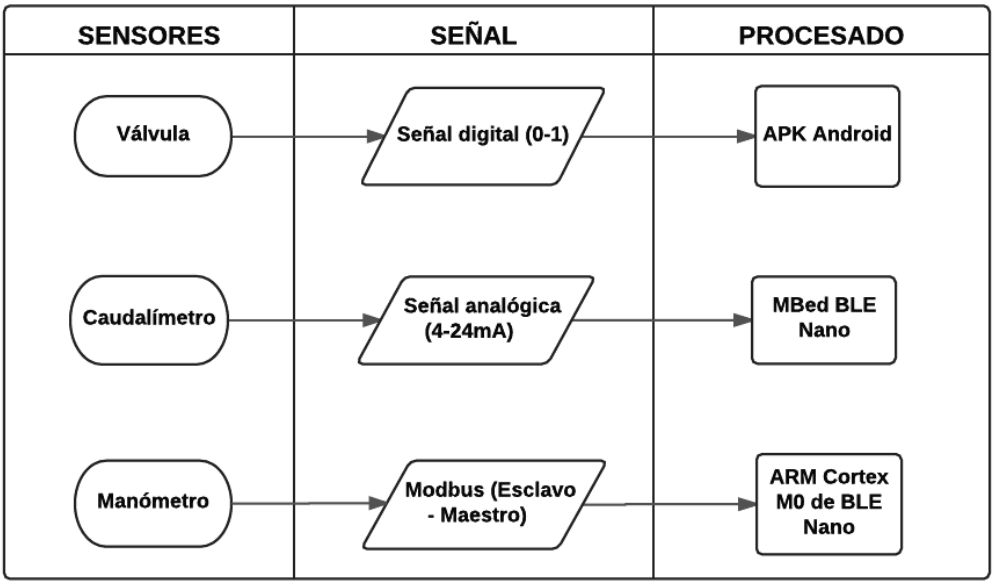


Ilustración 10. Diagrama Sensores/Dispositivos.

Para el acondicionamiento de la señal, los niveles lógicos serán:

Para la válvula tendremos los niveles de 0 o bajo en 0 – 0.8 V y para el 1 o alto 5.4V, el procesado de los datos lo se hace mediante el MBed por lo que no es necesario un circuito de acondicionamiento para este caso.

En el caso del manómetro que se comunica por modbus, tampoco será necesario el acondicionamiento de señal, ya que usa la misma tecnología TTL.

Por último, en el caso de la señal analógica del sensor de caudal, si es necesario un circuito de acondicionamiento, ya que MBed sólo trabajo con niveles de tensión y no de corriente, que sería lo ideal.

Para poder convertir la corriente a tensión y recibirla en el ARM correctamente para cuantificarla mediante MBed se utiliza una resistencia entre los terminales de entrada y tierra. Esta resistencia se calcula dividiendo los valores máximos de tensión y corriente que vamos a manejar, quedando así:

$$R = \frac{3.3\ V}{0.02\ A} = 165\ \Omega\ .$$

Esta medida es necesaria para medir todo el rango posible y tener una mayor precisión de medida en el acondicionamiento. El valor mínimo como se ha dicho anteriormente de corriente es de 4 mA, por lo tanto, la tensión mínima que correspondería a un valor 0 será de:

$$Vmín = 165\ \Omega \text{ x } 0.004 = 0.66 \text{ V}$$

Teniendo por nivel mínimo 0.66 V, los niveles lógicos irán desde este valor mínimo a 3.3 V que corresponde al valor más alto.

La resolución también es interesante calcularla, que en este caso será:

$$Intervalo\ de\ divisiones = \frac{[(3.3\ V - 0.66V)]x1024}{3.3\ V} = 819$$

Los saltos de tensión por tanto corresponderán a:

$$Saltos\ de\ tensión = \frac{(3.3\ V - 0.66\ V)}{819} = 3.22\ mV.$$

Se puede ver por consiguiente que los saltos mínimos de tensión serán de 3.22 mV, por lo que los valores tendrán esta precisión en el acondicionamiento de la señal.

Quedando finalmente así:

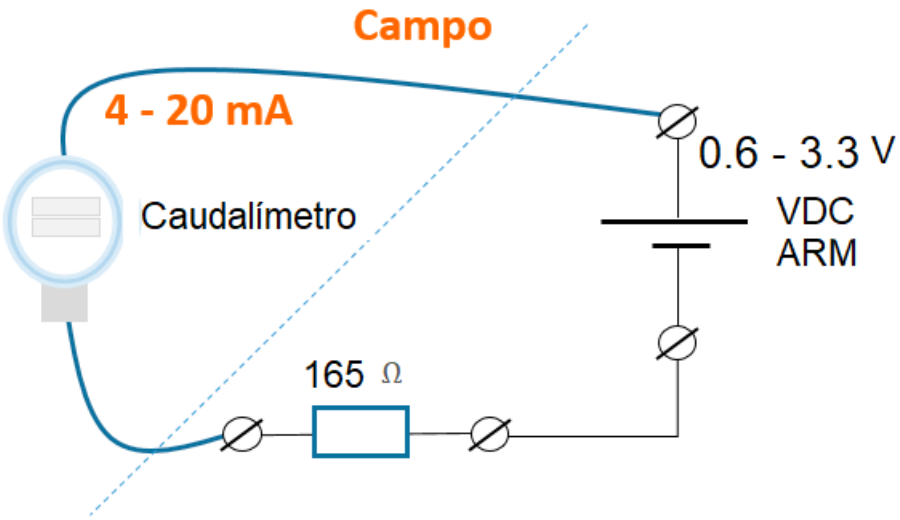


Ilustración 11. Diagrama Acond. Señal.

4.4 Diseño del esquemático y PCB.

Para el diseño de la PCB y el esquemático como se ha comentado antes ha sido en el software de 2.6.2 Altium Designer.

Para el diseño del esquemático se han hecho los siguientes bloques:

- Módulo de carga de batería Li – Po.

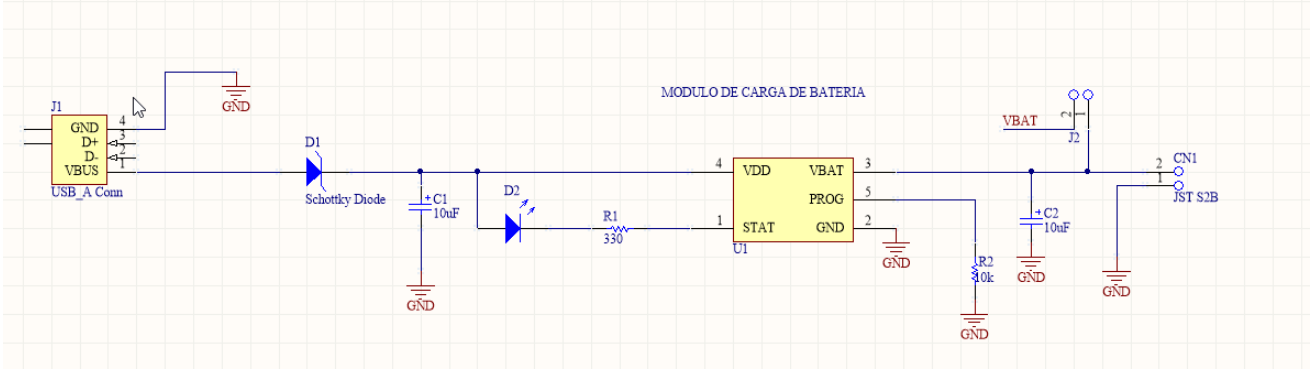


Ilustración 12 Esquemático del módulo de carga.

Como se puede observar, en primer lugar, está el USB de tipo B que es el mismo que utiliza el Arduino, este USB será el encargado de

suministrar los 5 V que da el ordenador o un transformador que tenga de salida 5 V.

A continuación, se coloca un diodo schottky, para evitar corrientes de retorno.

El siguiente bloque está compuesto por dos condensadores de 10uF como indica en su datasheet el controlador de carga MCP73831T-2ACI (para evitar ruidos y suavizar la señal, a la entrada y salida), un diodo led que estará controlado por el pin de estado "STAT", una resistencia de 10 Kohm que será la encargada de controlar los mA de carga de la batería, una referencia a GND y por último la señal VDD o de 5 V.

Según el estado de la patilla "STAT" encenderá o no el led, siguiendo la siguiente tabla sacada de su datasheet:

Charge Cycle State	STAT1	
	MCP73831	MCP73832
Shutdown	Hi-Z	Hi-Z
No Battery Present	Hi-Z	Hi-Z
Preconditioning	L	L
Constant-Current Fast Charge	L	L
Constant Voltage	L	L
Charge Complete – Standby	H	Hi-Z

Ilustración 13 Tabla Pin STAT.

Hay tres estados posibles:

- Alta impedancia (Hi – Z): Este estado estará presente cuando esté apagado el circuito o que no haya presencia de batería.

- Estado de carga: Cuando se detecta que hay batería y se cumplen las condiciones cargará la batería, en un primer momento dejará pasar más corriente para cargar más rápido la batería hasta que por motivos de salud de las baterías se carga a una corriente más baja con un voltaje constante.

- Carga completa: En este caso nuestro dispositivo pone en alto la patilla, encendiendo así el led.

En la parte del final o derecha, tenemos un switch y un "label" que conecta VBAT con Vin del BLE NANO. El switch tiene como función

cortar el suministro de energía al BLE Nano mientras se está produciendo la carga de la batería, para evitar sobretensiones.

- Bloque BLE NANO.

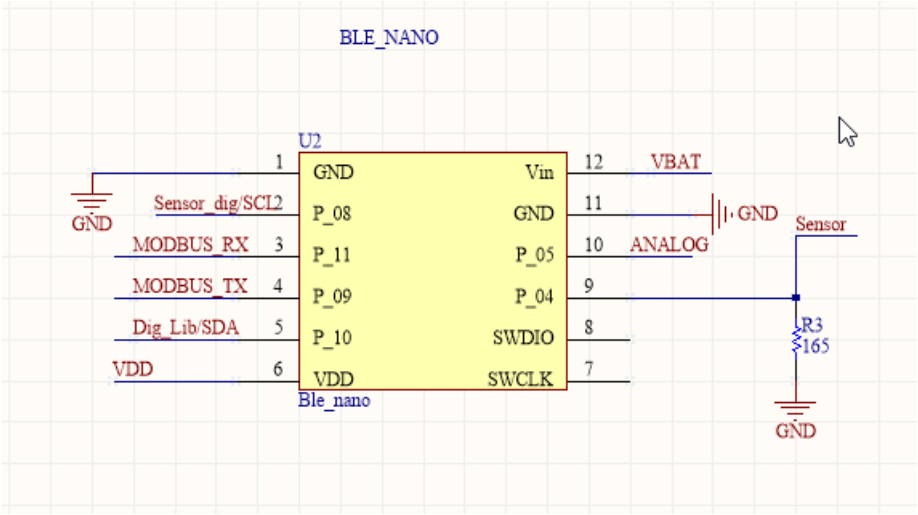


Ilustración 14 Esquemático BLE NANO.

Como se puede observar, este bloque corresponde a las salidas y entradas del dispositivo Ble Nano que se está utilizando. Si vamos enumerando según los números de las I/O tenemos:

El primer Net Label corresponde a una conexión con tierra, el segundo net corresponde a un sensor de carácter digital o SCL, el tercer net es el encargado de recibir los datos de modbus del esclavo (manómetro) y el cuarto es el encargado de enviar las señales al modbus esclavo, el quinto es destinado a un sensor digital o SDA, el sexto da una salida de 3.3 V, el séptimo net corresponder a una señal SWCLK, el octavo a una señal SWDIO, el noveno es el encargado de recibir la señal del caudalímetro y tendrá una resistencia a tierra que hemos calculado anteriormente de 165 Ohm, el décimo net es un sensor analógico libre,

el onceavo conectado a tierra y finalmente el doceavo recibe el suministro de energía mencionado anteriormente llamado VBAT.

- Bloque de disposición de bornas.

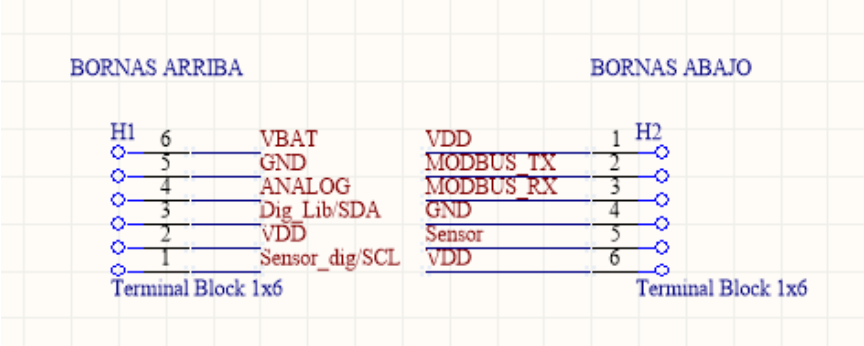


Ilustración 15 Esquemático Bornas.

En este bloque se denominan los "net labels" que podrán ser utilizados externamente.

Para el diseño de PCB:

- Diseño 2D:

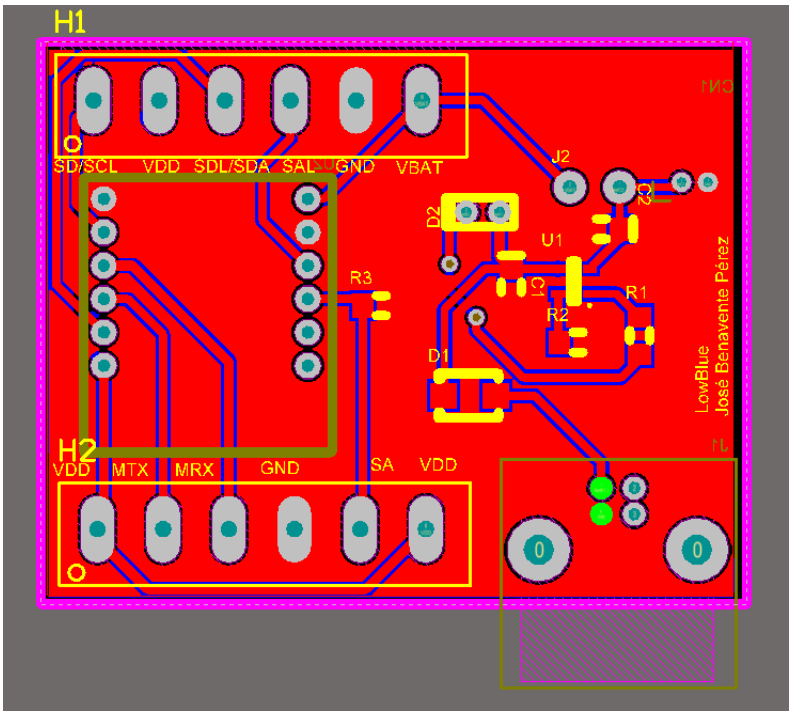


Ilustración 16 Diseño 2D PCB.

- Diseño 3D:

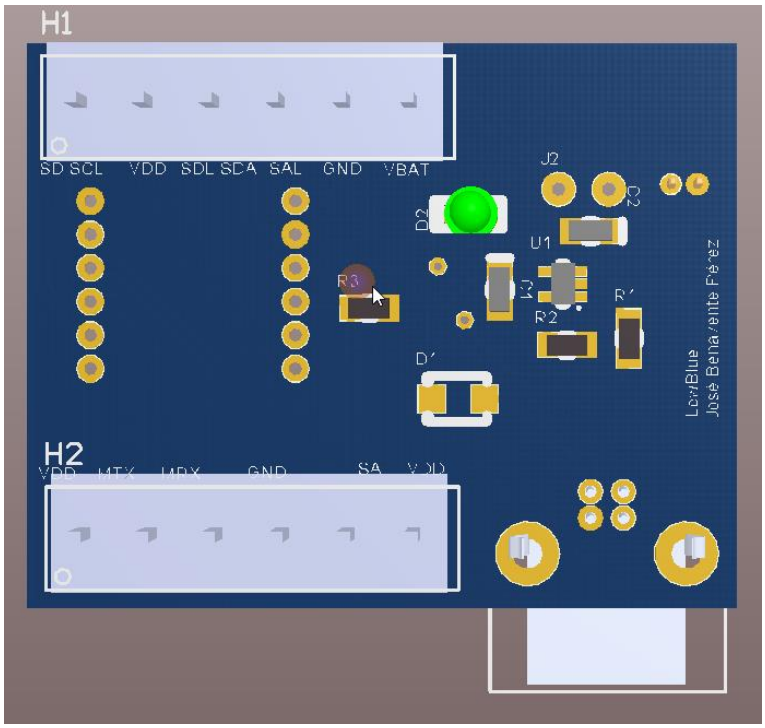


Ilustración 17 Diseño 3D PCB.

4.5 Subsistema de recepción de datos e interfaz con el usuario.

Para recibir los datos del Ble Nano e interactuar con la aplicación Android, se usa concretamente el dispositivo móvil Mi5. Tiene una gran autonomía y se puede portar muy fácilmente gracias a su reducido tamaño y fácil manejo.

Ilustración 18 Dispositivo de recepción de datos.

Las características principales de este dispositivo son:

XIAOMI MI5, CARACTERÍSTICAS TÉCNICAS	
DIMENSIONES FÍSICAS	144,55 x 69,20 x 7,25 mm, 129 gramos
PANTALLA	IPS 5,15 pulgadas
RESOLUCIÓN	1080p (428 ppp)
PROCESADOR	Snapdragon 820 (1,8 Ghz)// Adreno 530
RAM	3 GB
MEMORIA	32 GB (sin ranura microSD)
VERSIÓN SOFTWARE	Android 7.1.2
CONECTIVIDAD	LTE Cat12 compatible con Europa VoLTE, NFC Bluetooth 4.2, Wi-Fi ac GPS
CÁMARAS	Principal de 16 MP (f2.0) con estabilización 4 axis // Vídeo UHD // Secundaria 4 MP
BATERÍA	3000 mAh (no extraíble) Carga rápida 3.0

Ilustración 19 Características Dispositivo Mi5.

Se ha remarcado en rojo las características que han ayudado al desarrollo del proyecto, además de ser necesarias para su correcto funcionamiento.

La versión de Android del dispositivo es una de las últimas que se han desarrollado, comprobando así que funcionará la apk desarrollada en Android como mínimo desde esa versión a las demás anteriores (posteriores sólo estaría Android O, recién sacada al público).

Se puede utilizar también la conectividad LTE compatible con europa (es un dispositivo chino y usan distintas redes), Bluetooth 4.2 uno de los bluetooth más actualizados actualmente y Wi-Fi.

Capítulo 5

5. Desarrollo del Software.

5. 1 Introducción.

Para el desarrollo software se ha utilizado la plataforma Mbed para la programación del hardware y bluetooth del Ble Nano, Desarrollo Software Android Studio [2.8] para la programación de la aplicación móvil y Sourcetree [2.7] que se ha utilizado para el control de versiones del software.

5.2 Desarrollo del software de adquisición y balizamiento BLE.

En esta sección se explicará el desarrollo llevado a cabo de la programación en Mbed para Ble Nano.

Se explicarán los siguientes bloques:

En primer lugar, el flujograma que se aplica a la programación, para tener una idea clara del funcionamiento del mismo.

En segundo lugar, las librerías que se han utilizado para la implementación de la comunicación bluetooth y además las librerías que se han utilizado para la comunicación Modbus, las cuales se pueden encontrar en las librerías libres de Mbed.

5.3 Flujograma del ARM Cortex M0 BLE Nano.

En este apartado, como se ha comentado anteriormente se van a explicar los pasos que se han llevado a cabo para la programación del microprocesador que contiene el BLE Nano.

Un flujograma de la programación del software es la manera más clara y sencilla de mostrar los procesos que van procesándose en el microcontrolador.

El flujograma es el siguiente:

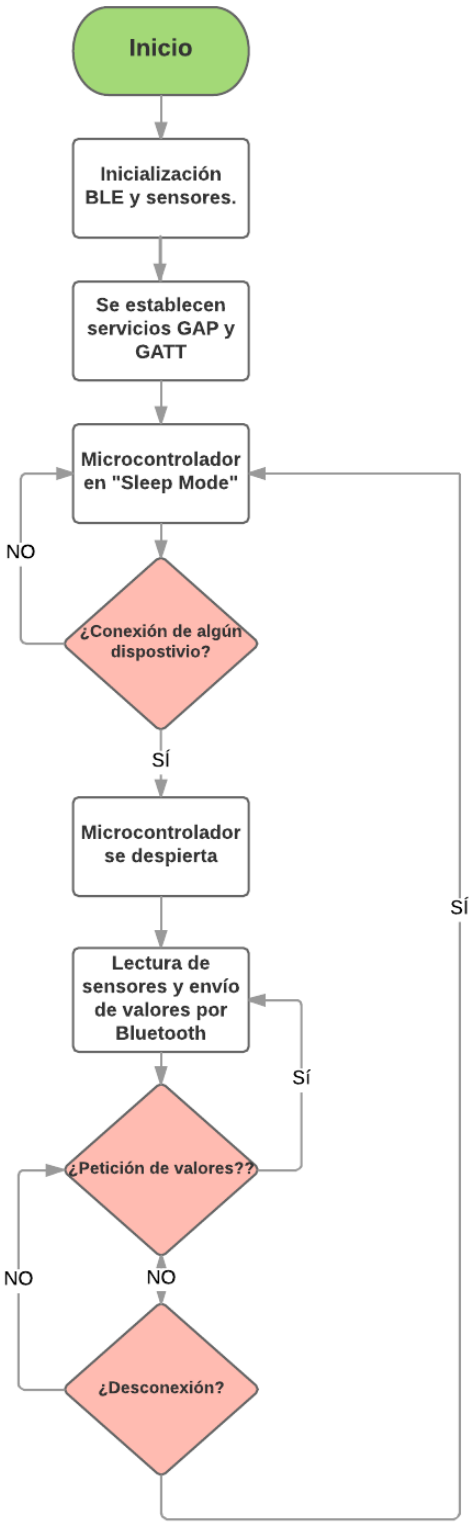


Ilustración 20 Flujograma Microcontrolador.

Como se puede observar, en el inicio del programa se inicializan los sensores, el bluetooth y los servicios GAP y GATT. El servicio GAP (acrónimo para el Generic Access Profile), se encarga de controlar las conexiones y los anuncios en BLE. GAP es lo que permite que el dispositivo sea público hacia el exterior y determina como dos dispositivos pueden (o no) interactuar entre ellos, por lo que es el encargado de la función de balizamiento, enviando una baliza para que los demás dispositivos bluetooth detecten el dispositivo. Por otra parte, los servicios GATT (acrónimo de Generic Attribute Profile), define la manera en que dos dispositivos BLE pueden comunicarse usando los servicios y características. La comunicación se realiza mediante un protocolo conocido como ATT, que se usa para almacenar los servicios, características y datos relacionados en una tabla usando identificadores de 16-bit para cada entrada en la tabla. GATT entra en juego una vez se ha establecido una conexión dedicada entre el dispositivo y el cliente, lo que significa que ya hemos pasado previamente por el GAP.

Se han definido tres atributos que serán de sólo lectura, que corresponderán a los tres sensores de caudal, presión y estado de la válvula. Para la lectura del sensor de caudal, se implementa también una librería Modbus que será posteriormente explicada con más detenimiento.

Los atributos se han definido tal que así:

```
/*------------ BLE variables ---------------------------------------*/
BLE ble;
const static char DEVICE_NAME[]       = "Arqueta";  // NOMBREE DE
DISPOSITIVO BLUETOOTH.

static const uint16_t uuid16_list[]   = {0xFFFF}; //Custom UUID, FFFF
is reserved for development

uint16_t customServiceUUID   = 0xA000;
uint16_t ValveUUID           = 0xA001;  //SENSOR VÁLVULA 0-1.
uint16_t PressureUUID        = 0xA002;  //SENSOR PRESIÓN POR MODBUS.
uint16_t FlowmeterUUID       = 0xA003;  //SENSOR DE CAUDAL POR P_05
ANALÓGICO ADAPTACIÓN 4-20 mA.
```

Ilustración 21 Código Atributos

Como se puede observar están definidos los tres atributos mencionados.

Una vez quedan definidos los protocolos GAP y GATT el microcontrolador entra en modo “sleep” o reposo, para ahorrar la

máxima energía posible, ya que, es suministrada mediante una batería y se precisa el mínimo consumo posible del dispositivo.

El dispositivo quedará mandando una baliza para que sea detectado y esperando una conexión de un cliente.

Cuando se produce una conexión de un cliente, el microcontrolador se "despierta" leyendo el valor de los tres sensores anteriormente declarados, mediante la comunicación digital se leerá el valor de la válvula, una lectura analógica mediante el acondicionamiento de señal anteriormente explicado del caudal y por último se leerá el valor de la presión a través del manómetro por protocolo Modbus.

El dispositivo quedará conectado hasta que se envíe una petición de desconexión y además estará leyendo cada un mínimo tiempo los valores de los sensores y enviándolos por bluetooth (poolling). Cuando se envía una petición de desconexión como marca el flujograma el microcontrolador volverá al estado de Sleep o reposo.

5.1.1 Uso de la librería BLE de Mbed.

Para la mayoría de los desarrolladores, uno de los mayores retos de Bluetooth Smart es la complejidad del controlador y las APIs subyacentes de la pila. Por ello Mbed creó BLE_API para resumir estos detalles. El uso de la API significa que las aplicaciones pueden aprovechar las construcciones de alto nivel y las interfaces ofrecidas por BLE_API, al tiempo que siguen siendo compatibles con todas las plataformas Bluetooth Smart subyacentes soportadas. Esto hace que los programas portátiles, fácilmente se puedan mover de una plataforma soportada a otra. Mediante el uso de BLE_API, los desarrolladores también se benefician implícitamente de todas las optimizaciones de bajo consumo ofrecidas por el hardware.

BLE_API es una capa muy fina y ligera de interfaces y abstracciones C++ que permite a los usuarios desarrollar aplicaciones portátiles para Bluetooth Smart.

El microcontrolador bluetooth del BLE Nano es en concreto el chip Nordic nRF51822, el cual es controlado por la librería de Mbed.
Como todas las librerías, facilitan el trabajo al desarrollador pudiendo utilizar los protocolos y funciones más usadas en comunicación bluetooth.

Para el manejo de las conexiones bluetooth se ha creado un objeto del tipo BLEDevice. Al objeto se le definen los perfiles GATT y GAP mencionados anteriormente para que se pueda comunicar con otros dispositivos.

```
// Set Up custom Characteristics
static uint8_t *flowmeterValue;
ReadOnlyArrayGattCharacteristic<uint8_t, (sizeof(*flowmeterValue)*4)>
flowChar(FlowmeterUUID, flowmeterValue);

static uint8_t valveValue[1] = {0};
ReadOnlyArrayGattCharacteristic<uint8_t, sizeof(valveValue)>
valveChar(ValveUUID, valveValue);

static uint8_t pressureValue [10] = {0};
ReadOnlyArrayGattCharacteristic<uint8_t, sizeof(pressureValue)>
pressureChar(PressureUUID, pressureValue);

// Set up custom service
GattCharacteristic *characteristics[] = {&flowChar, &valveChar,
&pressureChar};
GattService        customService(customServiceUUID, characteristics,
sizeof(characteristics) / sizeof(GattCharacteristic *));
```

Ilustración 22 Protocolos GAP y GATT

Como se puede observar en este caso, se definen los protocolos GAP y GATT de manera sencilla, ya que se disponen de otros ejemplos de librerías como Heart Rate que utilizan los mismos protocolos.

En este proyecto se definen los protocolos de los tres sensores, la válvula (valveValue), el manómetro (pressureValue) y el caudalímetro (flowmeterValue).

También se puede observar como en el payload del protocolo GAP no se inserta código porque se ha utilizado el perfil GATT para transferir la información de los sensores, el cual posee los tres atributos de lectura.

La función main es la siguiente:

```
int main()
{


    led = 0;
        printf("## inicialiazamos y establecemos los parametros ModBUS ms\n");
    //setting Modbus protocol.
    eMBErrorCode    eStatus;

    //eStatus = eMBInit( MB_RTU, SLAVE_ID, 0, 9600, MB_PAR_NONE );
    eMBInit( MB_RTU, SLAVE_ID, 0, 9600, MB_PAR_NONE );

    //setting BLE parameters
    // inicialiazamos y establecemos los parametros BLE

    ble.init();
    ble.gap().onConnection(myConnectionCallback);
    ble.gap().onDisconnection(myDisconnectionCallback);
```

1

```
    /* setup advertising */
    ble.accumulateAdvertisingPayload(GapAdvertisingData::BREDR_NOT_SUPPORTED |
GapAdvertisingData::LE_GENERAL_DISCOVERABLE);    // BLE only, no classic BT
    ble.setAdvertisingType(GapAdvertisingParams::ADV_CONNECTABLE_UNDIRECTED);
// advertising type
    ble.accumulateAdvertisingPayload(GapAdvertisingData::COMPLETE_LOCAL_NAME,
(uint8_t *)DEVICE_NAME, sizeof(DEVICE_NAME));     // add name

ble.accumulateAdvertisingPayload(GapAdvertisingData::COMPLETE_LIST_16BIT_SERVI
CE_IDS, (uint8_t *)uuid16_list, sizeof(uuid16_list)); // UUID's broadcast in
advertising packet

ble.accumulateAdvertisingPayload(GapAdvertisingData::MANUFACTURER_SPECIFIC_DAT
A, chamberData, sizeof(chamberData));
    ble.setAdvertisingInterval(ADVERTISING_TIME);

    // add our custom service
    ble.addService(customService);

        printf("\r START ADVERTISING \n");
    // start advertising
    ble.startAdvertising();

        printf("\r STOP ADVERTISING \n");
    sleep();
```

2

```
    while(true) {
        if(isConnected) {

                *flowmeterValue    = ( int )sensorFlow.read(); //reading
flowmeterValue.
            valveValue[0]     = sensorValve.read();
        }

        ble.waitForEvent(); //Save power

    }
} // end main.
```

3

Ilustración 23 Código Main

El código está dividido en tres fragmentos o partes para poder comprenderlo mejor, son las siguientes:

① Para el primer bloque, se inicializan y se establecen los parámetros Modbus, esta librería será explicada más adelante.

Por otra parte, el objeto BLEDevices permite manejar la librería mediante los eventos que ocurren. Tiene dos métodos llamados onConnection y onDisconnection a los que se les pasará los prototipos de las funciones myConnectionCallback y myDisconnectionCallback que son funciones de recepción.

Estas funciones son inicializadas en este bloque.

② En el segundo bloque, se configura el advertising y el tipo de advertising mediante los payloads de los métodos que contiene el objeto ble mencionado anteriormente.

En este bloque se comienza el advertising del dispositivo y sería visible para los dispositivos externos que estén a su alcance.

③ En el último y tercer bloque, se ha hecho una función while infinito que comprobará si hay alguna conexión con un dispositivo, como indica el flujograma del microcontrolador [0].

Cuando se recibe el evento de conexión se leen los valores de los respectivos sensores.

5.1.2 Uso de la librería BLE de Modbus.

En este punto, se explica la utilización de la librería Modbus para la implementación de este protocolo en el dispositivo Mbed.

Para entender este apartado, primero se va a analizar el protocolo Modbus y posteriormente la implementación que se ha llevado a cabo en este proyecto.

La librería que se ha utilizado gracias a la compatibilidad con el dispositivo Mbed es FreeModbus, que es una librería libre de Mbed.

Modbus es, dicho de forma escueta y sencilla un sistema de comunicación muy utilizado en la industria, sistemas de control, domótica, etc… porque permite establecer una comunicación maestro –esclavo, en la que se envían y se reciben datos (información) que permiten saber que es lo que está sucediendo en estos dispositivos.

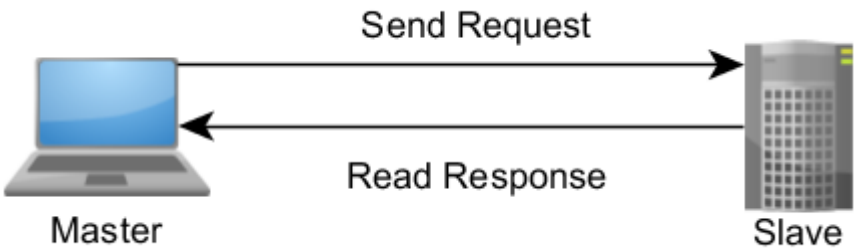


Ilustración 24 Modbus Maestro-Esclavo.

El transporte de datos en la comunicación Modbus es la PDU (protocol data unit) que consta de un código de función de un byte seguido de hasta 252 bytes de datos de funciones específicas.

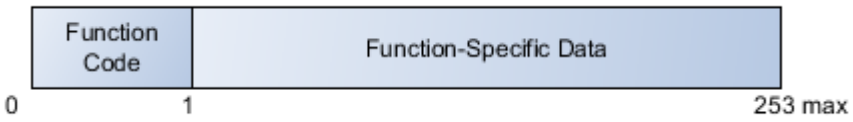


Ilustración 25 PDU Modbus

El código de función (trama que espera el esclavo según el estándar de comunicación) es el primer elemento que será validado. Si el código de función no es reconocido por el dispositivo que recibe la solicitud, responde con una excepción. Si se acepta el código de función, el dispositivo esclavo comienza a descomponer los datos de acuerdo con la definición de la función.

Debido a que el tamaño del paquete está limitado a 253 bytes, los dispositivos están limitados a la cantidad de datos que pueden ser transferidos. Los códigos de función más comunes pueden transferir entre 240 y 250 bytes de datos del modelo de datos de esclavos, dependiendo del código.

Se pueden usar varios protocolos de red. Los protocolos más comunes son serial y TCP/IP, pero se pueden usar otros como UDP también. Para transmitir los datos necesarios para Modbus a través de estas capas,

Modbus incluye un conjunto de variantes PDU que son diseñadas para cada protocolo de red.
Los tres formatos PDU estándares son TCP, unidad terminal remota (RTU), y ASCII. RTU y ASCII ADUs normalmente son usados a través de una línea serial, mientras que el TCP es usado a través de redes TCP/IP o UDP/IP modernas.

Las variantes de los protocolos son entre otros:

Modbus RTU: Es la implementación más común disponible para Modbus. Se utiliza en la comunicación serie y hace uso de una representación binaria compacta de los datos para el protocolo de comunicación. El formato RTU sigue a los comandos/datos con una suma de comprobación de redundancia cíclica (CRC) como un mecanismo de comprobación de errores para garantizar la fiabilidad de los datos. Un mensaje Modbus RTU debe transmitirse continuamente sin vacilaciones entre caracteres. Los mensajes Modbus son entramados (separados) por períodos inactivos (silenciosos).

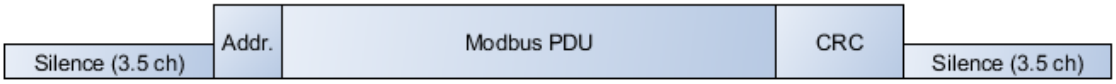


Ilustración 26 Modbus RTU

Modbus ASCII: Se utiliza en la comunicación serie y hace uso de caracteres ASCII para el protocolo de comunicación. El formato ASCII utiliza un checksum de control de redundancia longitudinal (LRC). Los mensajes Modbus ASCII están entramados por los dos puntos principales (":") y la nueva línea (CR/LF).

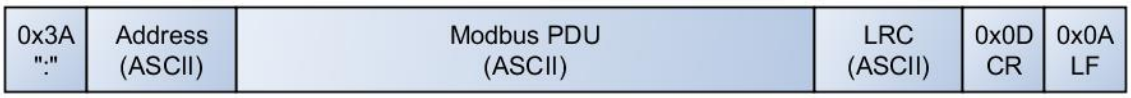


Ilustración 27 Modbus ASCII

Modbus TCP/IP o Modbus TCP: Se trata de una variante Modbus utilizada para comunicaciones a través de redes TCP/IP, conectándose a través del puerto 502.2 No requiere un cálculo de suma de verificación (checksum), ya que las capas inferiores ya proporcionan protección de checksum.

Ilustración 28 Modbus TCP/IP

Para este proyecto el protocolo más sencillo y eficaz para la comunicación es el Modbus RTU por lo que se ha escogido este protocolo.

Primero, la dirección (Addr.) es usada para definir para qué esclavo está diseñada una PDU. En la mayoría de las redes, una dirección 0 define la dirección de "broadcast". Es decir, un maestro puede enviar un comando de salida a la dirección 0 y todos los esclavos deben procesar la solicitud, pero ningún esclavo debe responder. Además de esta dirección, el CRC es usado para asegurar la integridad de los datos.

Para la comunicación Modbus RTU se hace mediante las conexión de RS-232. Esta conexión requiere de 4 terminales, que son los siguientes:

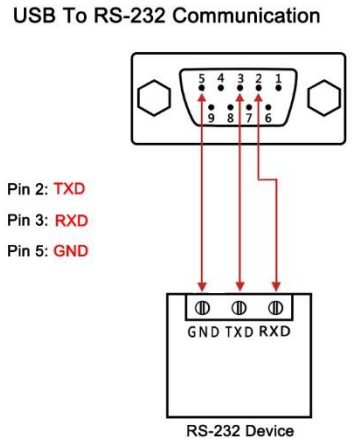


Ilustración 29 Comunicación RS-232.

Si recordamos en el Ilustración 15 Esquemático Bornas. están las 3 señales necesarias para la comunicación, en los números 2 (TX), 3 (RX) y 4 (GND).

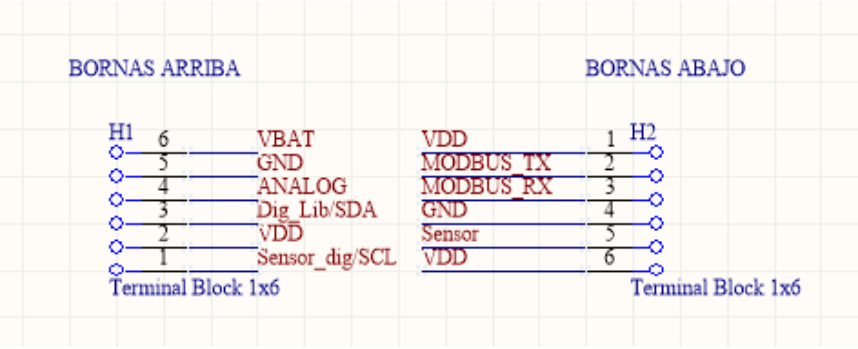


Ilustración 30 Esquemático RS-232

La biblioteca FreeModbus por tanto, recoge los datos de estas señales y los procesa teniendo en cuenta los conceptos anteriormente explicados, en este caso sólo usamos los registros de lectura del esclavo, esto ahorrará memoria en el dispositivo.

Dirección MODBUS	Dirección Usada en el protocolo	Nombre de la Tabla de Datos
1 - 9999	0000 – 9998	Output Coils (Lectura/escritura)
10001 - 19999	0000 – 9998	Inputs Contact (Lectura)
30000 - 39999	0000 – 9998	Inputs Registers (Lectura)
40001 - 49999	0000 – 9998	Holding Registers (Lectura/Escritura)

Ilustración 31 Datos y direcciones Modbus.

Por último, los procesos de comunicación Modbus tienen tres casuísticas posibles que se muestran a continuación:

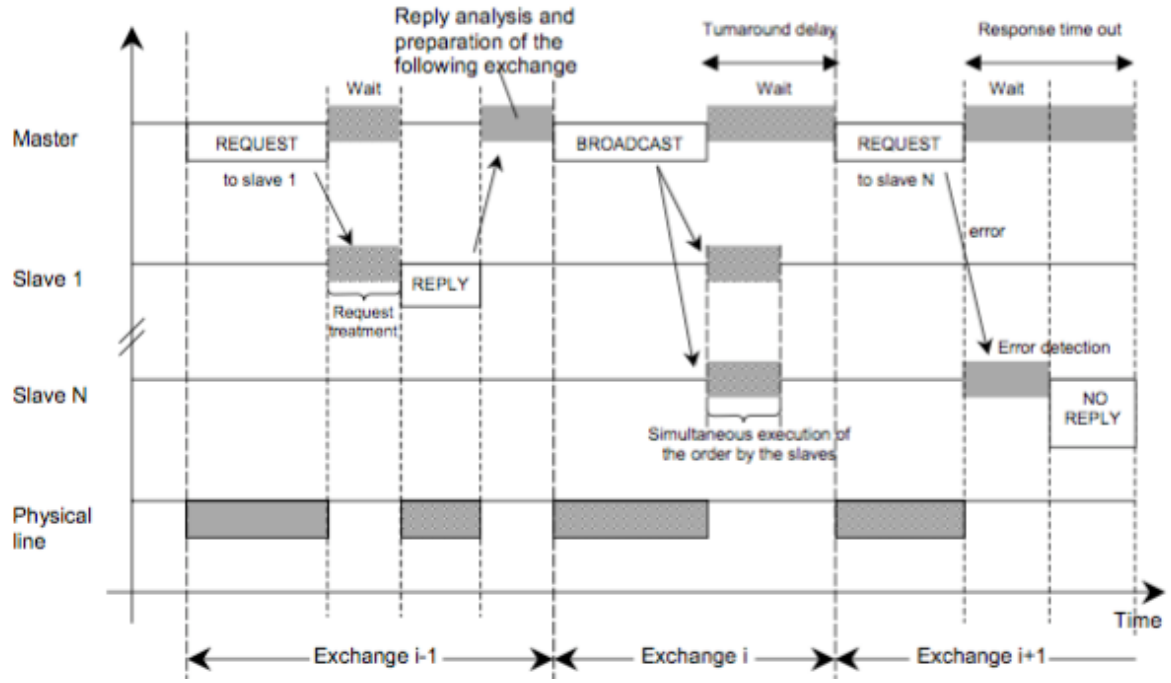


Ilustración 32 Diagrama de comunicación Modbus.

La primera se da cuando el maestro envía una petición a un esclavo y este esclavo recibe dicha petición de información, el esclavo procesará la información que se le ha pedido y la envía de vuelta al maestro. En este caso es una comunicación exitosa.

En la segunda casuística, como se ha comentado anteriormente el maestro puede enviar un mensaje "broadcast" que llegará a todos los esclavos, éstos deben procesar la información requerida y enviarla al maestro.

En el último caso, se produce un error de comunicación y el cliente no envía nada al maestro esperando que salte un time out en el maestro que retransmitirá de nuevo la petición a dicho esclavo.

Para terminar en esta sección, el microcontrolador del dispotivio BLE será el dispositivo maestro y el sensor de caudal el esclavo.

5.3 Interfaz y arquitectura software con el operador de la aplicación Android.

En esta sección, se va a explicar de una manera más concreta la composición de la arquitectura de funcionamiento de la aplicación Android.

La aplicación Android tiene distintas funcionalidades, que para entender mejor se puede ver la siguiente figura o ilustración:

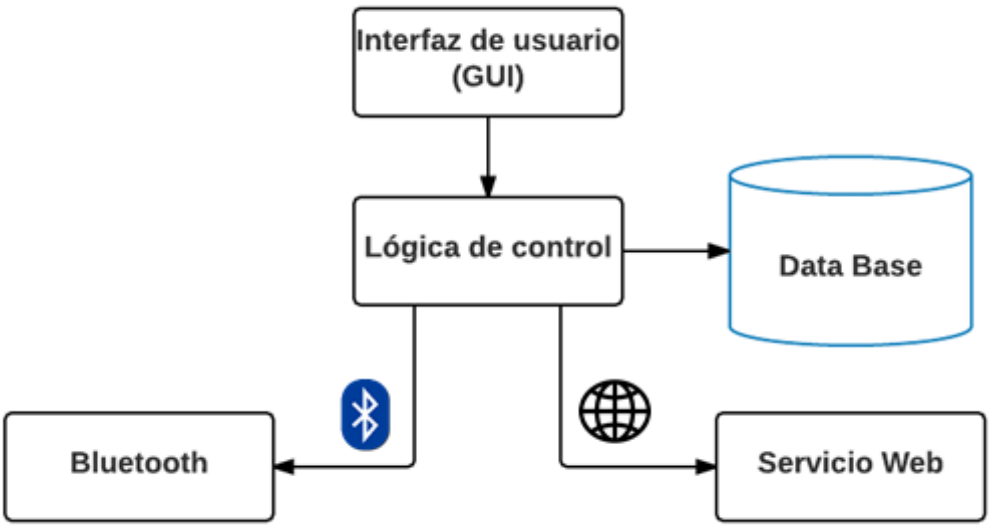


Ilustración 33 Diagrama funciones aplicación Android

Cada una de las funciones son:

- **GUI:** Tiene como función interactuar con el operador de modo visual, en la que muestra la información de los sensores y poder insertar datos que son pedidos en dicha interfaz.

- **Lógica de Control:** Esta funcionalidad tiene como objetivo la decisión de llevar a cabo una funcionalidad u otra. Por ejemplo, si retransmitir datos por bluetooth, enviar datos al servicio web o guardarlos en la data base correspondiente.

- **Data Base:** Esta es la base de datos de la aplicación, por tanto, es donde se guardarán los datos o información de los datos introducidos en la interfaz.

- **Bluetooth:** Realizará el control de funcionalidades del bluetooth y manejará el comportamiento del mismo.

- **Servicio Web:** Esta función será la encargada del manejo de peticiones y las respuestas del servicio web.

5.3.1 Módulos software de la aplicación Android.

La aplicación Android cuenta con unos módulos software. Cada uno de estos módulos tiene una función concreta, se ha hecho así para simplificar al máximo el software de dicho software.

Los módulos que se han desarrollado son los siguientes:

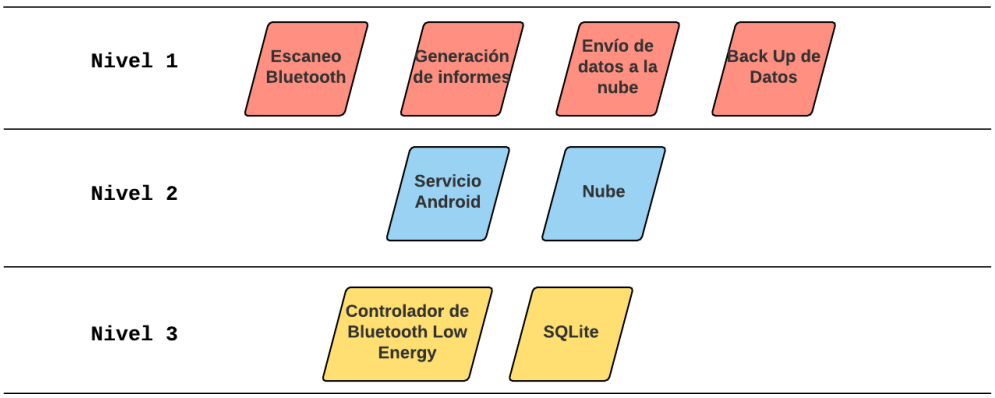


Ilustración 34 Niveles de los módulos

Como se puede ver en la ilustración anterior, se han propuesto tres niveles de estructura de los módulos:

El tercer nivel, es el nivel más bajo del software. Se encarga del manejo de los datos en la aplicación y además incorpora un controlador del Bluetooth LE desarrollado bajo el patrón Singletón y un módulo que maneja los datos en la comunicación con la base de datos SQLite que tienen interna los dispositivos Android que además también tiene un patrón Singletón.

El segundo nivel, es el nivel intermedio. Este nivel tiene como función de dar funcionalidad a la aplicación. En este nivel, se encuentra el módulo de servicio Android que sirve para mandar notificaciones al dispositivo y además hacer posible un escaneo de dispositivos Bluetooth en un proceso background. Por otra parte, también está el módulo de la Nube que se encarga de la comunicación con el Cloud y en concreto la serialización de los objetos que se envían mediante Json.

El primer y más alto nivel, es la capa de presentación con el operador. Sirve como método de comunicación con el operador y hacer sencillo la comunicación con él. En esta interfaz se pueden escanear los dispositivos bluetooth, hacer los test de estado de las arquetas, guardar una foto de la misma, enviar informes o ver un resumen de los datos de los sensores.

5.3.2 Clases de la aplicación Android.

En este apartado se van a comentar las funcionalidades de cada una de las clases que se han implementado en la aplicación Android.

Las clases están agrupadas en paquetes y son las siguientes:

- **BEAN:** Este paquete contiene las clases contenedoras de la información de la base de datos y clases encargadas del paso de información en el sistema de Android.
    - **BeanBluetoothDevice:** En esta clase se encapsula la información del objeto BeanBluetoothDevice de la librería de Android y además se serializa el objeto.
    - **BeanInformesDB:** Serializa los datos de cada una de las activitys que va recorriendo la aplicación. Estos datos son

introducidos por el usuario o son recogidos por medio de la comunicación Bluetooth.

- **BLE:** Este paquete es el encargado del manejo del Bluetooth BLE.
    - **AdRecord:** Clase para tratar la trama de balizamiento que se envía para extraer los datos.
    - **BLEBroadcasterReceiver:** Recibe toda la información recibida por Bluetooth y la envía a las demás clases que soliciten dicha información.
    - **HandlerBLE:** Se encarga del manejo del Bluetooth del dispositivo móvil, tiene un patrón Singletón necesarios para la clase Handler.
    - **MyCallBack:** Implementa las acciones que puede ofrecer el Bluetooth, como la conexión o la desconexión.
    - **ServiceData:** Contiene los servicios a los que se puede conectar con Bluetooth.
    - **ServiceType:** Clase contenedora que tiene como función manejar los tipos de servicios que están implementados que son necesarios para el objeto Handler.
- **CORE:** Es el paquete que contiene las clases que son el núcleo de la aplicación.
    - **BLE_Application:** Contiene parte del patrón Singletón para hacer más fácil su uso y tener el objeto HandlerBLE disponible en todas las activitys que se consideren necesarias.
    - **MyNotificationHandler:** Tiene como función crear un tipo específico de notificación, así como la estética de la misma y la función que puede realizar el usuario.
    - **ServiceDetectionTag:** En esta clase, se implementa un servicio Android. Este servicio se ejecuta en segundo plano para realizar operaciones. El servicio que se implementa es el escaneo de dispositivos sin tener que abrir la aplicación necesariamente.

- **DB_SQLite:** Este paquete tiene como función el manejo de la base de datos interna de Android SQLite.

    - **InformesSQLiteDataSource:** Esta clase implementa las sentencias SQL de la base de datos además de los métodos utilizados para realizar las operaciones de insertar, eliminar y modificar. Esta base de datos tiene la misma estructura que la base de datos que se encuentra en el servicio web.

    - **MySQLiteOpenHelper:** Hereda de la anterior, proporcionada por la librería Android. Su objetivo es el manejo de la base de datos y que se encuentre en el dispositivo móvil.

- **GUI:** Este paquete tiene las clases que corresponden al manejo de la interfaz gráfica y los componentes que la componen.

    - **ScanActivity:** Implementa la primera de las activitys de la aplicación. En ella el usuario puede iniciar el escaneo de los dispositivos Bluetooth que se encuentren al alcance. Se mostrarán en pantalla los dispositivos cercanos detectados en una lista.

    - **ScanArrayAdapter:** Tiene como función adaptar la vista de la lista de los elementos que son mostrados en la activity de scaneo.

    - **DeviceActivity:** Recoge los datos por Bluetooth, como también inserta información extra, en este caso un comentario o una foto.

    - **InsideChamberActivity:** En esta clase se recogen los parámetros del interior de la arqueta, mediante un tipo test de los parámetros más comunes.

    - **OutsideChamberActivity:** En esta clase se recogen los parámetros del exterior de la arqueta, mediante un tipo test de los parámetros más comunes.

    - **ResumeChamberActivity:** Muestra toda la información que ha sido recogida a modo de un informe para una comprobación visual del operador. Contiene un botón de envío de datos el cual crea un informe para enviarlo al servicio web. En caso de haya fallo se guardará en la memoria interna.

- **UTIL:** Este paquete es un paquete auxiliar, ya que contiene una clase que facilita el uso de las demás clases.
    - **Constant:** Contiene valores constantes de la aplicación.
- **WEBService:** Paquete que contiene clases que se encargan del manejo de las peticiones y respuestas del webservice.
    - **BeanArqueta:** Recoge la información de la base de datos la tabla arqueta.
    - **BeanInforme:** Recoge la información de la base de datos de la tabla informe.
    - **BeanSector_trabajo:** Recoge la información de la base de datos de la tabla sector_trabajo.
    - **VolleySingleton:** Esta clase maneja las peticiones y las respuestas del servicio web mediante la librería Volley.
    - **WebServiceConstant:** Clase que contiene las constantes del servicio web, la dirección del servidor y otras rutas.

5.4 Arquitectura software de la gestión online.

En esta sección se explica cómo se ha desarrollado el servicio web y también las herramientas que se han utilizado para la gestión de la información y la perseverancia de datos.

Esta arquitectura se va a componer de una aplicación web como es django y una base de datos que tiene integrada de MySQLite.

El esquema gráfico de la arquitectura podría ser este:

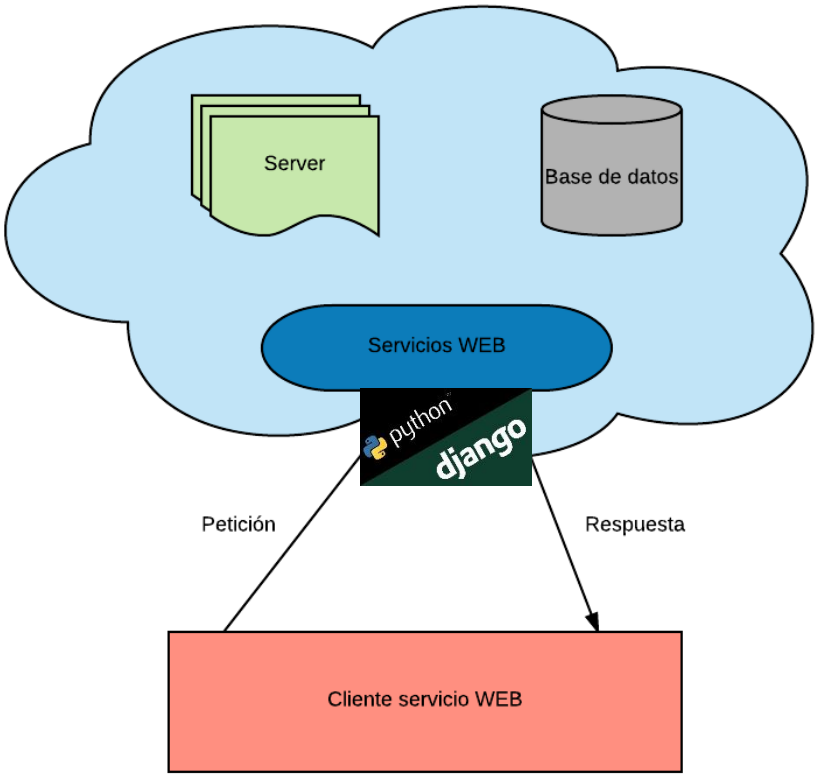


Ilustración 35 Esquema Gráfico Gestión Online

El servicio web, esperará una petición del cliente para responder atendiendo a la base de datos y al servidor.

Se ha desarrollado en lenguaje python, que se ha escogido porque el lenguaje tiene una gran comunidad que hace más fácil el desarrollo, con una evolución constante y una cantidad de ejemplos y librerías considerable adecuándose correctamente a las necesidades del presente proyecto.

Actualmente la mayor parte de servidores tienen una arquitectura LMAP generando una gran competencia de precios entre las plataformas para adquirir el servicio.

Por otro lado, en cuanto a la base de datos se ha escogido una base de datos que pudiera ser escalable en un futuro, por ello se ha escogido una base de datos relacional SQL.

5.4.1 Base de datos.

Para la base de datos, se han tenido en cuanta la estabilidad, la escalabilidad y la seguridad principalmente.

En la base de datos, hay una relación de 3 tablas:

- Una tabla llamada arqueta que es donde se guardará la información de la arqueta.

- La segunda tabla, que guardará la información suministrada por el dispositivo móvil del cliente.

- La tercera tabla guardará la información que no es modificable ni insertable desde el dispositivo móvil, por lo que es una tabla estática.

- Una cuarta tabla con los usuarios registrados en la aplicación web.

- Y la última y quinta tabla, que tendrá los usuarios suscritos, para la recepción de emails.

A continuación, se van a describir las informaciones que contienen dichas tablas.

En la primera tabla, tabla arqueta contiene los siguientes parámetros introducidos por el cliente:

- **ID:** Es la calve de identificación universal de la tabla autoincremente que hace de índice de la tabla.

- **Dirección_arqueta:** Es un registro que denomina la dirección bluetooth de la arqueta, este valor es alfanumérico y hace la función de clave primaria.

- **Inser_time:** Es el tiempo que la arqueta de da de alta en el sistema.

- **Nombre_Arqueta:** Es el nombre de arqueta que se le ha designado.

- **UUID_Sensor1:** Es el identificador del primer sensor.

- **UUID_Sensor2:** Es el identificador del segundo sensor.

- **UUID_Sensor3:** Es el identificador del tercer sensor.

En la segunda tabla o tabla informe tiene los siguientes parámetros:

- **ID:** Es la clave primaria del identificador universal de la tabla de autoincremento. Hace de índice de la tabla.

- **Acceso_ubicación:** Este dato es un byte que almacena si el estado de la arqueta que se ha introducido es bueno, aceptable o necesita reparación.

- **Comentario:** Es un tipo varchar que recoge un comentario del cliente desde la aplicación Android.

- **Contadores:** Se trata de un byte especial para definir el rango en bueno, aceptable o necesita reparación según en este caso del estado de los contadores.

- **Cubierta:** Es un byte especial para definir el estado de la cubierta.

- **Dirección_Arqueta:** Es un registro que almacena la dirección bluetooth de la arqueta y hace de función de clave primaria.

- **Disctancia_Reglamentaria:** Es un byte que define el rango del estado de la distancia de los elementos.

- **Juntas_Escaleras:** Es un bute que define el rango del estado de las juntas.

- **Manómetros:** Es un bute que define el rango del estado del manómetro.

- **Param_verticales_ext:** Es un bute que define el rango del estado de los parámetros verticales.

- **Param_verticales_int:** Es un bute que define el rango del estado de los parámetros verticales.

- **Partes_escalera:** Es un bute que define el rango del estado de las partes de la escalera.

- **Perímetro_arqueta:** Es un bute que define el rango del estado del perímetro de la arqueta.

- **Puerta_acceso:** Es un bute que define el rango del estado de la puerta o puertas de acceso.

- **Válvulas:** Es un bute que define el rango del estado de las válvulas.

- **Ventilación_lateral:** Es un bute que define el rango del estado de la ventilación lateral.

- **Ventilación_superior:** Es un bute que define el rango del estado de la ventilación superior.

- **Ventosas:** Es un bute que define el rango del estado de las ventosas.

Para la tercera, la tabla sector de trabajo:

- **ID:** Es la clave primaria de identificador universal de la tabla de autoincremento y hace de índice de la tabla.

- **Dirección_arqueta:** Es un registro alfa numérico que corresponde a la dirección bluetooth de la arqueta y hace de función primeria.

- **Fecha_control:** Es la fecha en la que se ha dado de alta el registro del sector de trabajo.

- **Hm:** Es un parámetro de la posición de la arqueta.

- **Tm:** Es un parámetro de la posición de la arqueta.

- **Ramal:** Es la definición del ramal que pertenece a la arqueta.

- **Responsable_control:** Define a la persona que se encuentra a cargo del control de la inspección de las arquetas.

- **Responsable_zona:** En este caso definde el responsable de inspección zonal.

- **Zona:** Define la zona o área donde se desarrolla la inspección de las arquetas.

Para la cuarta y quinta tabla, la de usuarios registrados y suscritos tendremos los mismos valores:

- **Nombre:** Es el nombre del usuario registrado.

- **Email:** Consta del email del usuario.

- **Empresa:** Es la empresa de la que forma parte el operario.

- **TimeStamp:** Es la hora en la que se ha registrado dicho operario en la aplicación web mediante los formularios.

5.4.2 Servicio Web.

Los servicios web son un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas intercambian datos entre sí con el objetivo de ofrecer servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios los solicitan llamando a estos procedimientos a través de la Web. A su vez proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

Las características principales de los servicios web son las siguientes:

- Utilización de estándares de internet. La única forma para que los servicios Web sean utilizados por la cantidad de sistemas heterogéneos existentes en Internet es el empleo del protocolo

de transferencia de datos HTTP utilizado por todos los navegadores Web y XML.

- Basados en tecnologías de paso de mensajes. La interacción entre el cliente y el proveedor del servicio es empaquetada en unidades autodesciptivas denominadas mensajes. Dicha interacción se describe en función de los mensajes intercambiados.

- Combinan lo mejor de la tecnología de componentes y de la tecnología Web. Los servicios Web presentan una funcionalidad de caja negra que puede ser reutilizada sin preocuparse de cómo es implementada y ello proporciona interfaces bien definidas.

Para proporcionar la interoperabilidad se ha utilizado el formato Json para la trasmisión de mensajes entre las aplicaciones. Se ha decidido utilizar el método get del protocolo http para enviar y recibir mensajes. Se creará una url que llamará a un script que le enviará la información.

La url es la siguiente:

- /servicioweb/scriptN.php?name1=value1&name2=value2

En la primera parte se tiene la dirección del script que está dentro del servidor. Le sigue el símbolo de interrogación "?" que es el que indica que se pasará variables al script en un combinado clave-valor separando las variables con el aspersar "&" como muestra la tabla.

Por último, para entender mejor el proceso que lleva el servidor web, se van a explicar las capas diseñadas y cómo se relacionan entre ellas.

Las capas son las siguientes:

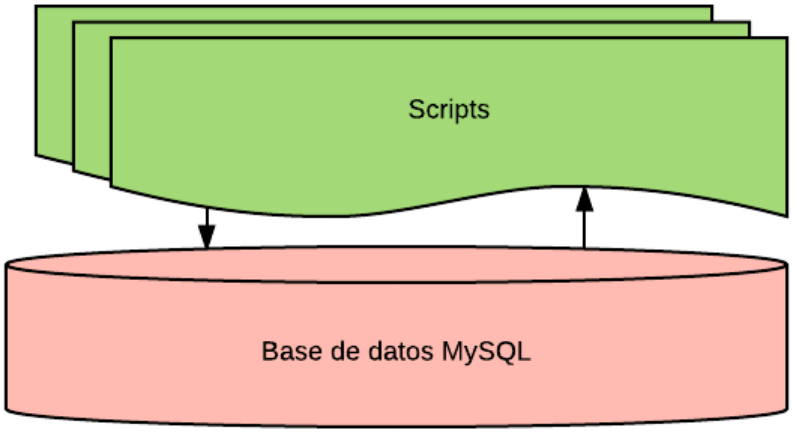


Ilustración 36 Capas Servicio WEB

La base de datos MySQLite es la encargada de crear una instancia de la base de datos por una petición enviada desde el dispositivo móvil para el manejo de la misma. La capa superior está compuesta por scripts, que dan la funcionalidad al software. Estos scripts serán los que realizan la tarea de recoger la información de las peticiones y procesarla, comprobando también que los datos no llegan vacíos o que los datos que se han proporcionado son los esperados. También serán los encargados de solicitar a la capa de base de datos las operaciones que el script tiene autorizados para hacer.

5.4.3 Aplicación Web Django.

Django tiene la siguiente estructura:

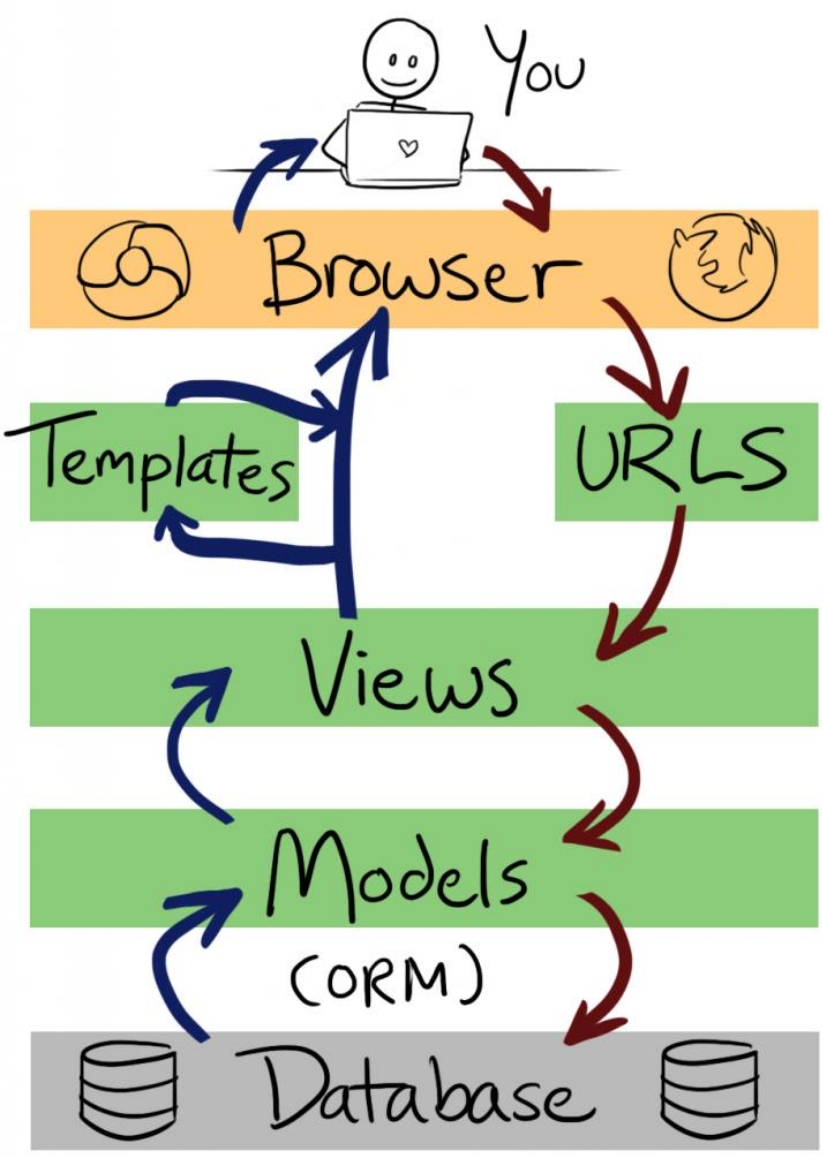


Ilustración 37 Estructura Django General

Como se muestra en la ilustración anterior, el usuario puede hacer una petición a través del navegador a nuestra aplicación web mediante una url y nuestra aplicación web le mostrará los templates que hayamos creado, cogerá la información de las vistas que a su vez las forman los modelos y por último los modelos cogerán la información actualizada de la base de datos.

Para explicar mejor estos pasos, tenemos la siguiente estructura de carpetas .

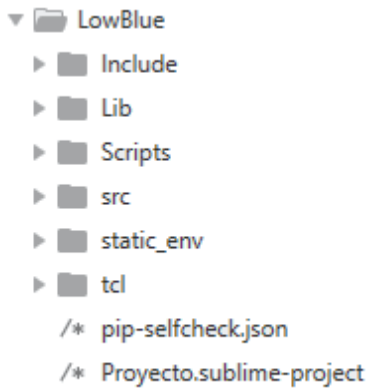


Ilustración 38 Estructura Carpeta Django

Esta estructura es la estructura que se crea cuando se empieza un proyecto en django. Lo recomendable es hacer un "virtual environment" de Python, porque si se trabaja en equipo en la aplicación web, que los cambios en versiones no afecten a este entrono si no lo deseamos.

En este proyecto se ha creado un entonrno virtual llamado LowBlue y se ha instalado el entorno de Django en la versión "2.0.5", la versión más actual.

Para la creación de las carpetas de la ilustración anterior hay que crear el proyecto mediante la instrucción por consola de "Python .\scripts\django-admin.py startproject LowBlue". Se creará dentro del directorio donde se haya ejecutado la scipt.

Posteriormente para iniciar la aplicación web en servidor local, se ejecuta dentro de la carpeta src "Python manage.py runserver" y mediante la url del servidor local veremos nuestra web:

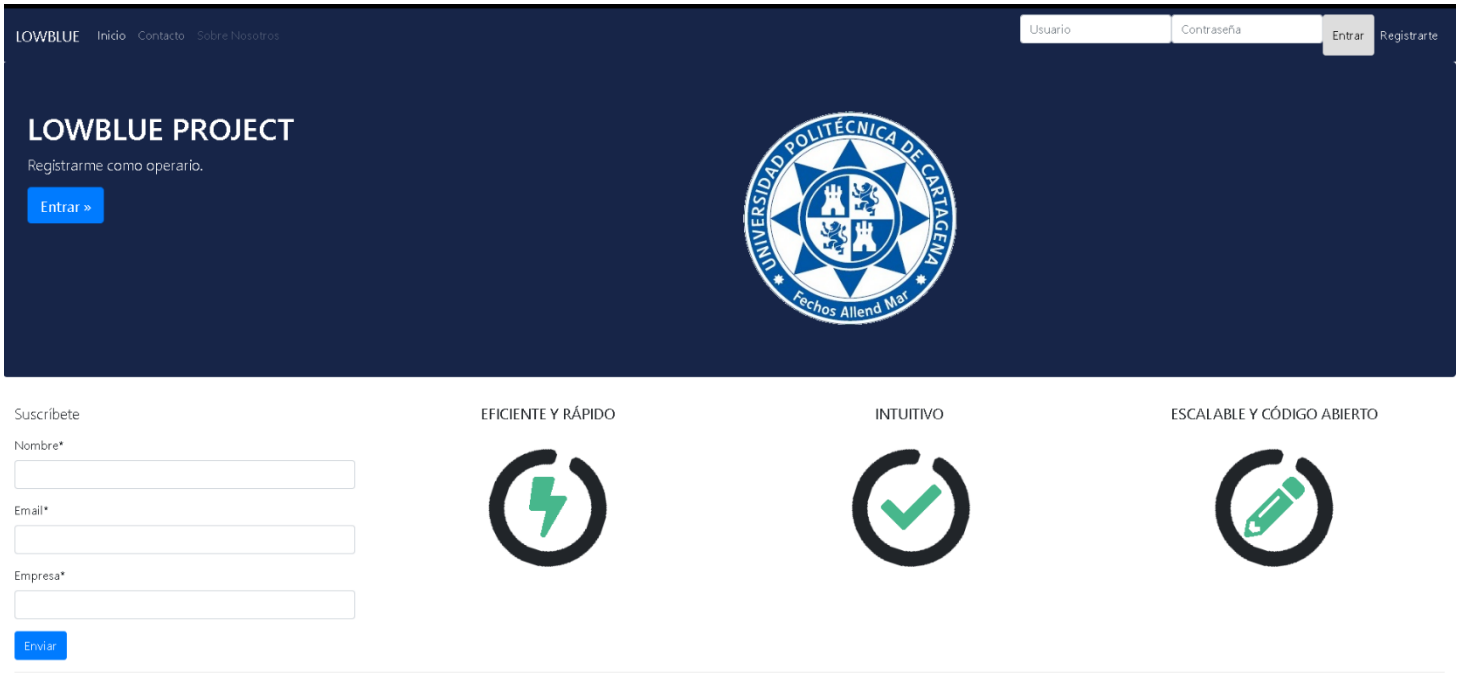


Ilustración 39 Aplicación Web Principal

En este caso, este es el resultado final de la aplicación web creada en este proyecto, sin embargo, si no tuviéramos nada se vería una página por defecto que crea django.

La carpeta que es más importante es la de src que es en la que se hacen los cambios deseados de la aplicación.

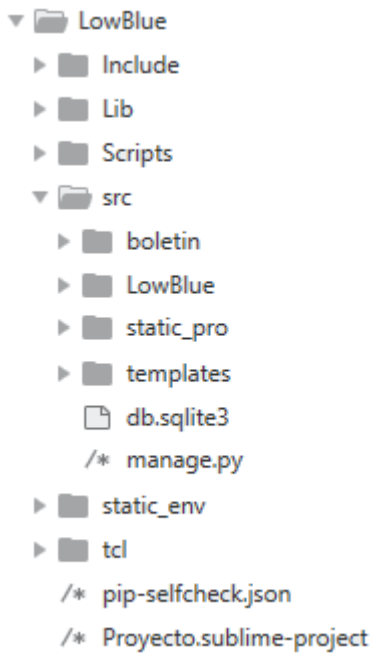


Ilustración 40 Carpetas Proyecto Django

En esta carpeta se pueden ver las carpetas boletín, LowBlue, static_pro, templates y los archivos db_sqlite3 y manage.py.

Para comenzar a explicar el funcionamiento que tiene cada uno tenemos:

- **Manage.py**: Este archivo de Python es una utilidad de la línea de comandos que permite interactuar con este proyecto Django de diferentes formas. Si se ejecuta la siguiente instrucción "Python manage.py" se muestran las diferentes opciones que permite hacer este scipt. Entre las más importantes son:

    - **Runserver:** Como se ha comentado anteriormente este comando arrancará nuestro servidor.
    - **Createsuperuser:** Este comando servirá para la creación de super usuarios, los super usuarios tienen derecho a entrar en la administración de la web.
    - **Changepassword:** Ofrece la posibilidad de cambiar la contraseña de super usuario desde la consola.
    - **Inspectdb:** Muestra las tablas que se han creado en la base de datos a partir de los modelos que se hayan creado. Django tiene ya unas tablas o modelos creados por defecto, como "authuser".
    - **Makemigrations:** Se aplican los cambios que se han creado en los modelos del proyecto, sin migrar a la base de datos.
    - **Migrate:** Se aplican los cambios de los modelos en la base de datos.
    - **Startapp:** Crea una aplicación para nuestro proyecto.
    - **Startproject:** Este comando es el que se ha comentado anteriormente para la creación del proyecto, en este caso se crea desde la carpeta scripts de django.
    - **CollectStatic:** Este comando se utiliza para aplicar los cambios en los archivos estáticos del proyecto, en este

caso la barra de navegación forma parte de un documento estático porque está presente en todas las vistas.

- **Dbsqlite3:** Por defecto Django utiliza este tipo de base de datos, aunque se pueden utilizar otros relacionales como MYSQL, Oracle, PostgreSQL… En otros casos como MongoDB también se puede implementar, aunque oficialmente no gastan recursos en ello.

- **Boletin:** Es la aplicación que se ha creado para este proyecto, dentro tenemos estos archivos:

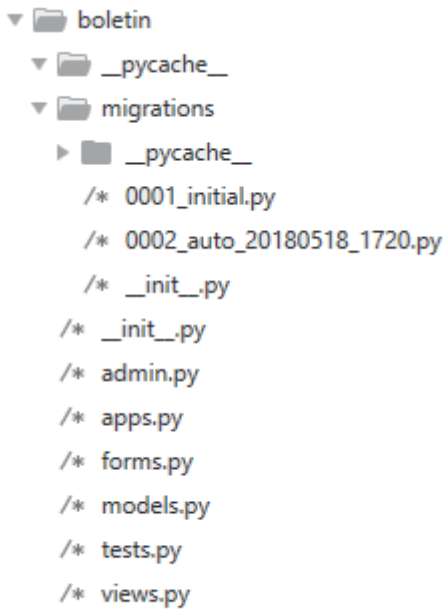


Ilustración 41 App Boletin

Como se puede observar, tiene varios archivos de Python. Los más importantes son los de admin, forms, models y views.

  - **Models:** Aquí se ha creado el modelo de "Registrados", en él tendremos la información de nombre, email, empresa y el timestamp de cada uno de los usuarios que se registren en la suscripción de la aplicación.

  - **Admin:** Aquí se pondrán las características del modelo Registrado que se podrán ver en el apartado de administración, además de la edición de ellos, ya sea la eliminación o creación de usuarios (pero solo para el superuser).

- **Forms:** En este caso se tienen los formularios que se utilizan en la aplicación de boletín, como son el de la suscripción y el de contacto.

- **Views:** En este archivo de Python se definen las vistas, en este caso están las de inicio y contacto.

- **LowBlue:** En este caso se tienen los mismos archivos que en la aplicación, pero serán generales de la aplicación web.

- **Static_Pro:** En este caso se crean los archivos estáticos, estos archivos son los que se contendrán los estilos de la aplicación, como son los css las imágenes y los javascripts. En este caso se ha usado Bootstrap.

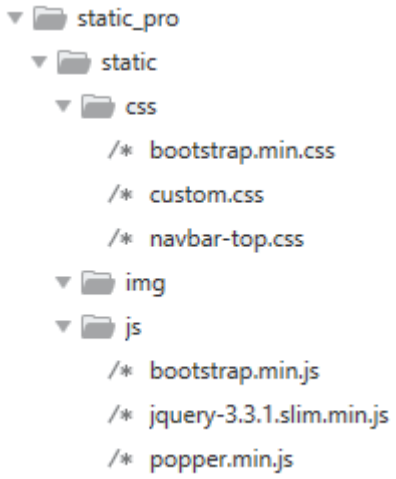


Ilustración 42 Archivos estáticos.

- **Templates:** Por último, los templates, estos son los archivos html que son los que mostrarán al usuario la interfaz visual.

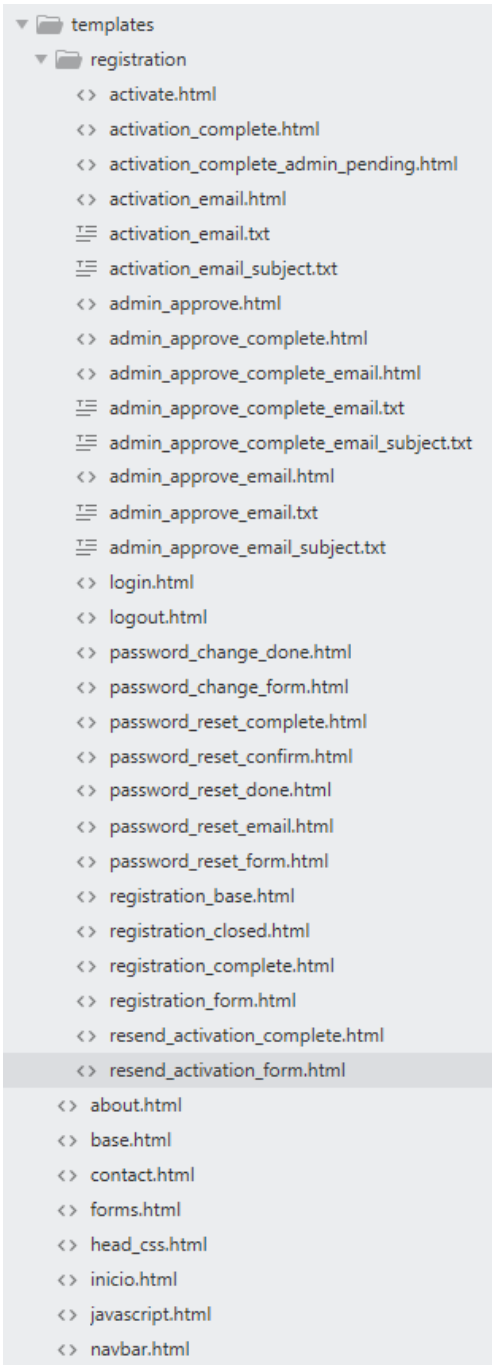


Ilustración 43 Templates.

Todos los html anteriores son los que dan forma a la aplicación web, se tienen las siguientes vistas de la web.

- La de inicio, sin tener una sesión iniciada:

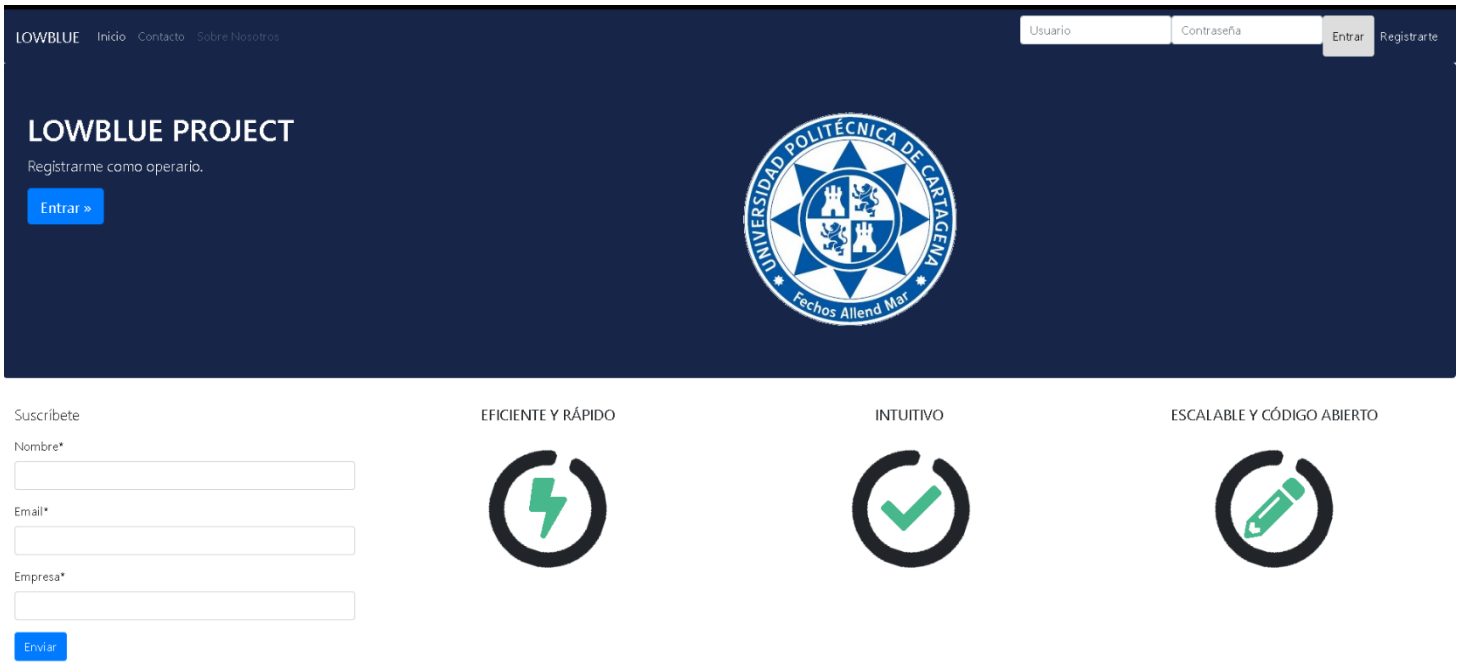


Ilustración 44 Web inicio sin tener iniciada la sesión.

- La de inicio con sesión iniciada de un operario sin permisos de staff o superusuario:

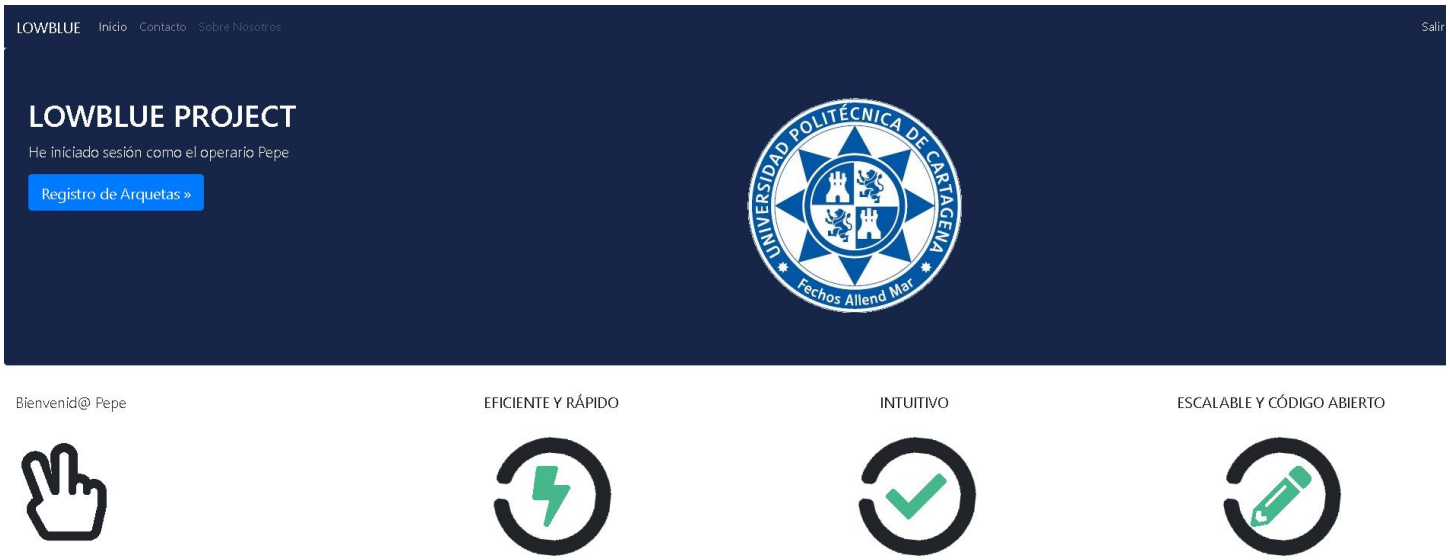


Ilustración 45 Web inicio con sesión iniciada.

- Inicio con sesión iniciada con permisos de staff o superusuario:

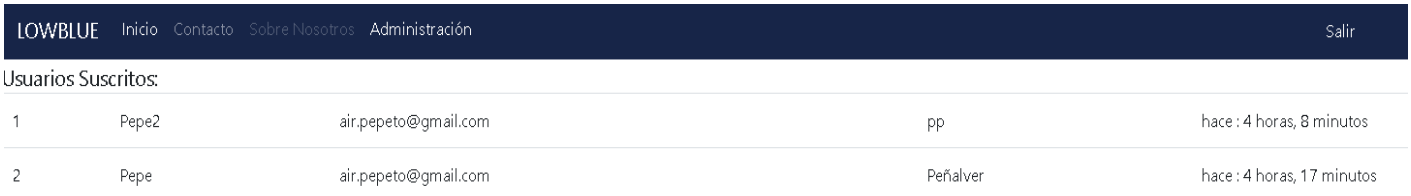


Ilustración 47 Web Inicio sesión iniciada de staff.

- Contacto, que enviará un mensaje via email al administrador:

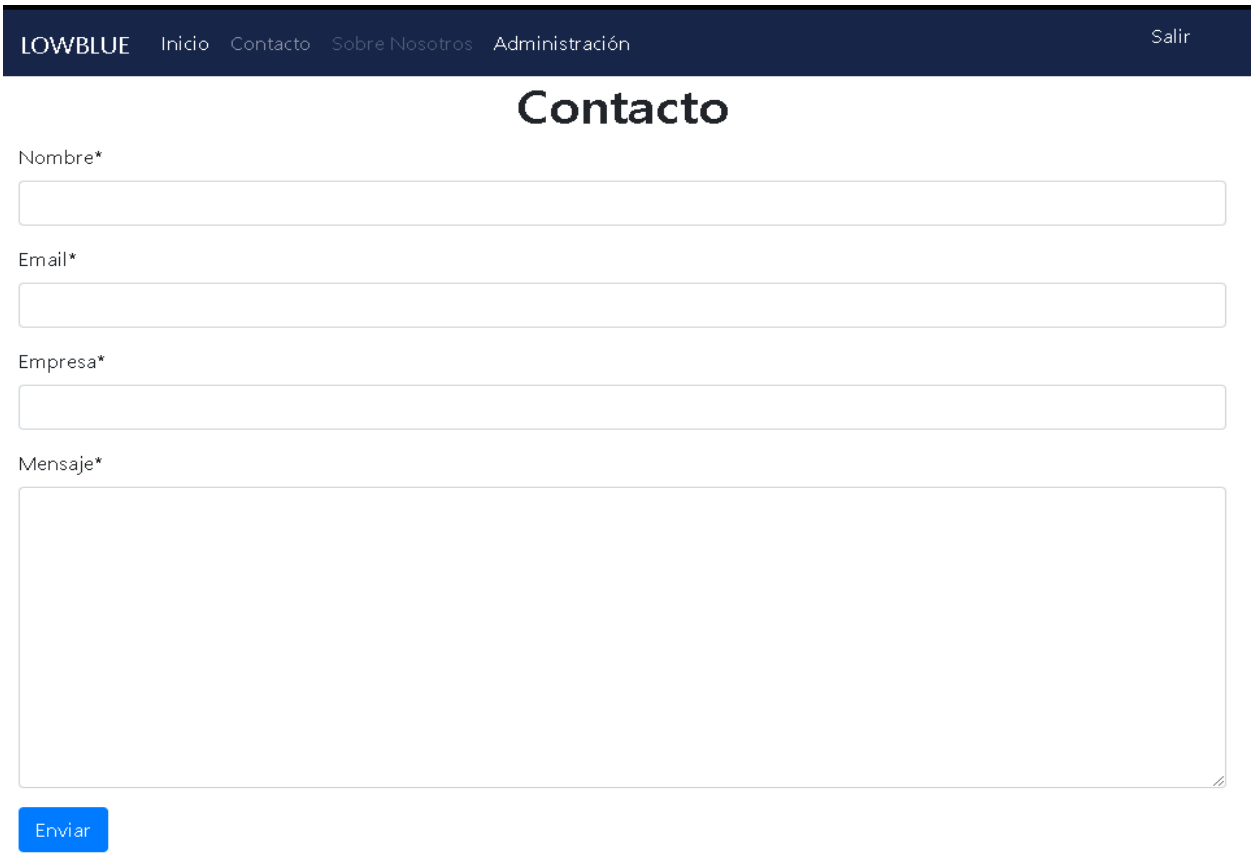


Ilustración 46 Web Contacto.

- Sobre nosotros:

Ilustración 48 Web Sobre Nosotros.

- Registro:

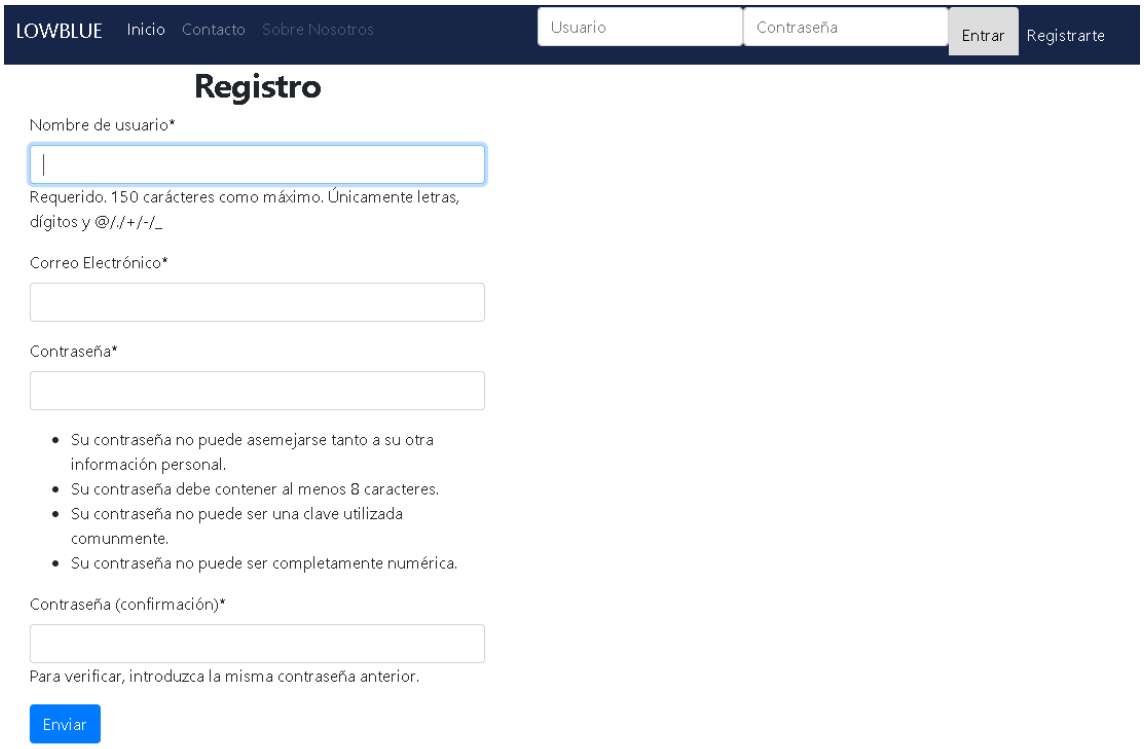
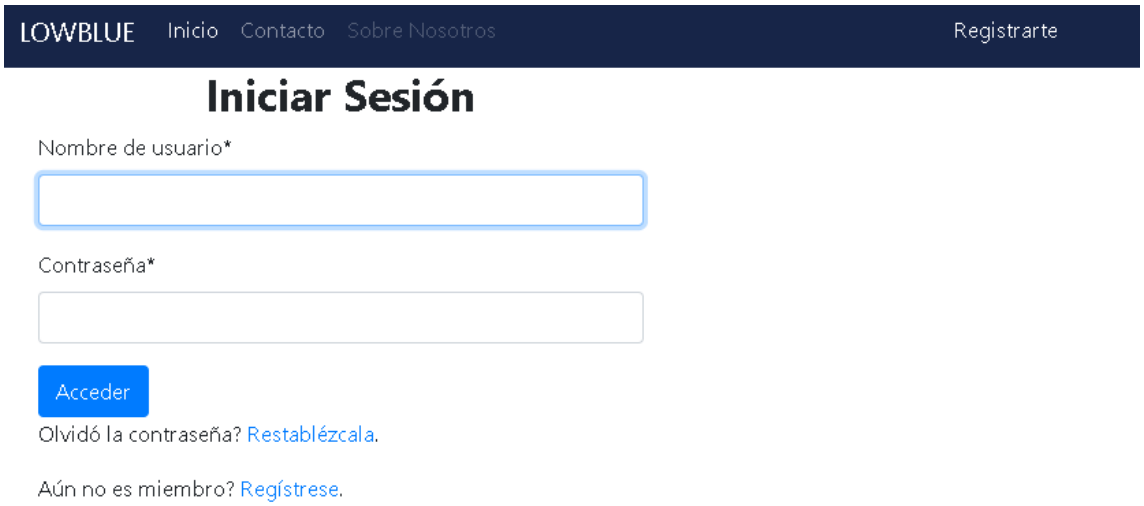


Ilustración 49 Web Registro.

- Inicio de sesión:



Capítulo 6

6. Pruebas realizadas.

6.1. Batería y rendimiento.

Este último capítulo se dedica a las pruebas que se han realizado físicamente en el dispositivo.

Uno de los apartados más importantes en este proyecto es la duración de la batería que se va a implementar en el dispositivo.

La batería que actualmente se ha instalado es la siguiente:

Ilustración 50 Batería 400 mAh

Esta batería tiene un tamaño muy reducido (26.5mm x 36.9mm x 5mm), con un voltaje de salida nominal de 3.7 V (perfecto para la alimentación del dispositivo BLE Nano) y una máxima corriente de 400 mAh, que es más que suficiente para el consumo del dispositivo.

La carga normal de esta batería es de 80 mAh, por lo que debería completarse en 5 horas si estuviera totalmente descargada la batería.

La carga máxima sería de 400 maH pero no es lo recomendado, por las posibles altas temperaturas y deterioro de la batería.

Estos son los datos teóricos de la batería, pero se han estudiado los datos en la práctica arrojando los siguientes resultados.

El consumo del dispositivo mientras de anuncia (con el led encendido) es de 2.25 mAh. El consumo cuando se conecta el dispositivo móvil para ver los datos, se produce un pico de 3 mAh y vuelve a bajar a unos 2.15 mAh. Por tanto, si despreciamos los picos de consumo producidos puntualmente en la conexión del dispositivo tenemos el siguiente consumo.

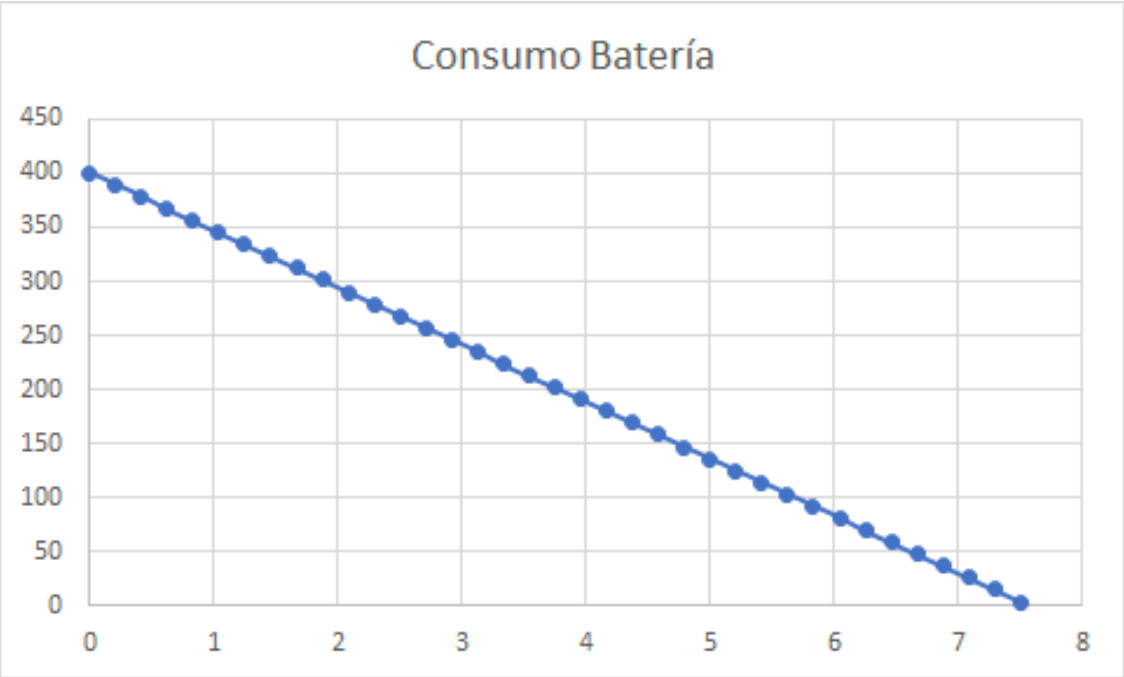


Ilustración 51 Consumo Batería

Se comprueba que con un consumo constante de 2.2 mAh, la batería debería durar alrededor de una semana y efectivamente este es el resultado final.

En cuanto a la carga de la batería, se ha usado el módulo MCP73831T-2ACI/OT.

En el datasheet del módulo, se puede apreciar la siguiente gráfica:

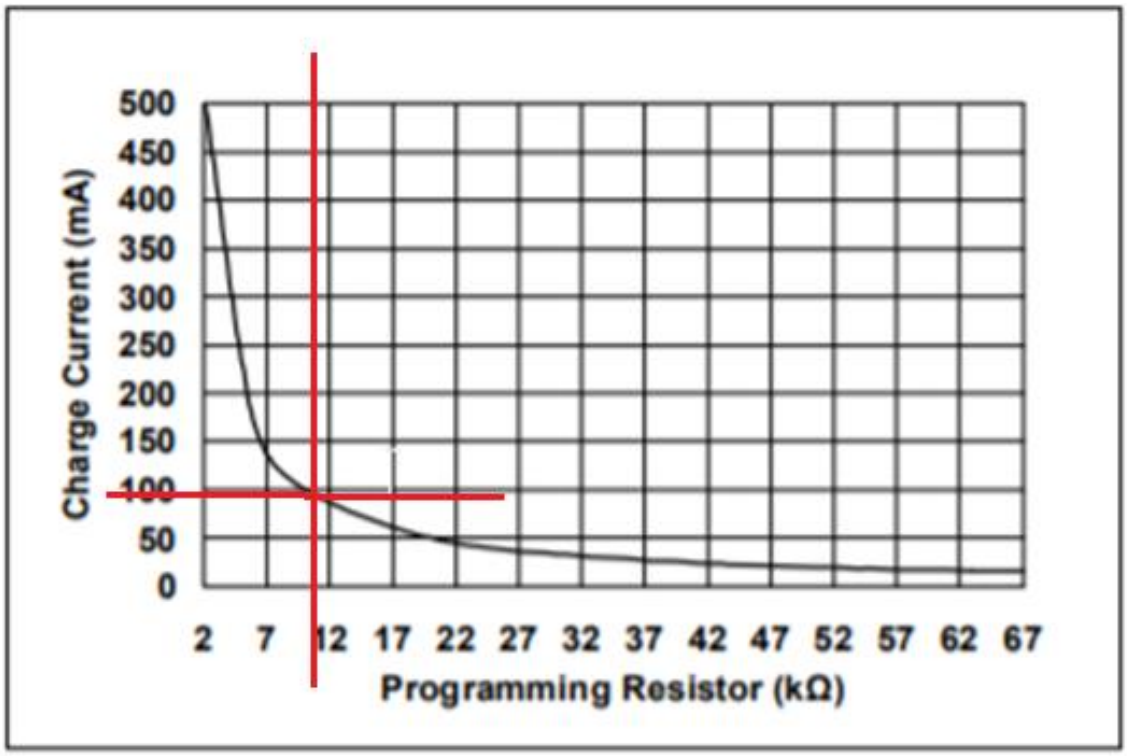


FIGURE 2-4: *Charge Current (I_{OUT}) vs. Programming Resistor (R_{PROG}).*

Teniendo en cuenta que se ha usado una resistencia de programación de 11 KOhm, en la gráfica sale como resultado que la corriente de carga es de alrededor de casi 100 mAh. Por tanto, se tiene que la carga se completa en:

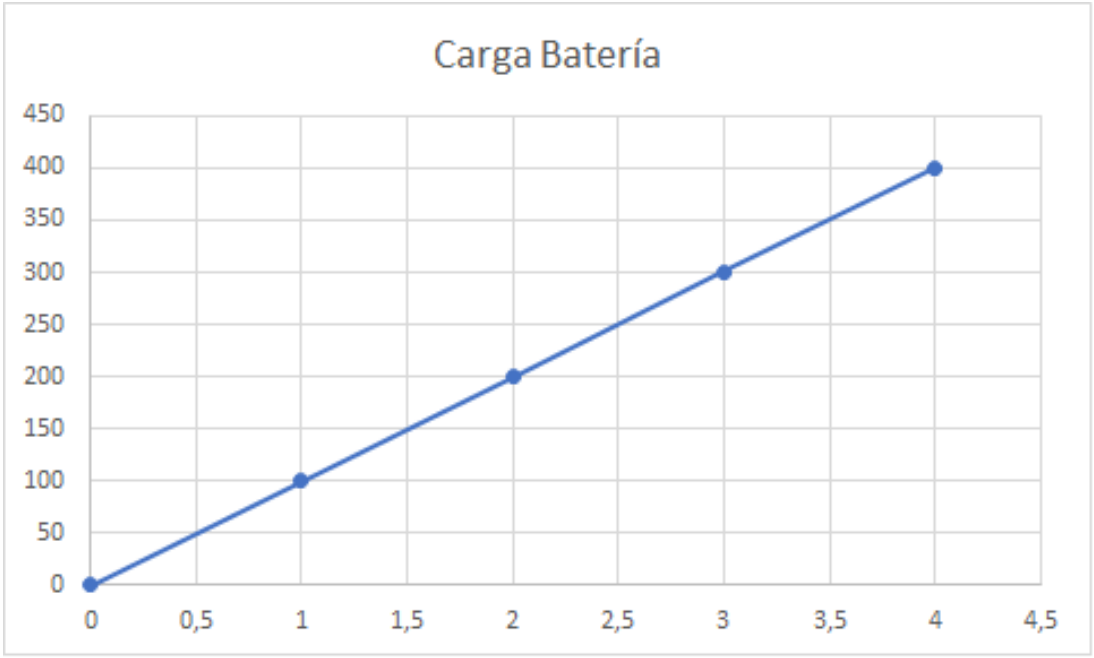


Ilustración 52 Carga Batería

La carga de la batería se debería completar en unas cuatro horas, teniendo en cuenta una carga de 100 mAh aproximadamente de corriente de carga. En la realidad se ha comprobado que tarda un poco más la carga de la batería.

6.1.1 Funcionamiento del led.

Para saber si la batería está totalmente cargada se ha incorporado un led azul, este led se encenderá en cuanto la batería se detecte que ya está cargada.

Probando la carga completa de la batería se puede ver como efectivamente el led se enciende:

Ilustración 53 Led Batería.

6.2 Simulación de sensores con Arduino.

Por otro lado, las simulaciones de las lecturas de los sensores, tanto de caudal como de la válvula han sido simuladas mediante Arduino.

En primer lugar, para la simulación de la señal analógica se ha utilizado una salida digital PWM de Arduino, concretamente la salida 3.

Para la conexión de ordenador - DAP link USB Board – RedBearLab Nano – Ardino se ha hecho lo siguiente:

Primero, se conecta mediante el USB la placa del ReadBearLab Nano al ordenador, creando una conexión COM.

En esta placa se conectará la señal de GND, Tx y Rx desde la placa que se ha diseñado en el proyecto, para poder ver mediante conexión UART los resultados de la comunicación con los sensores.

Finalmente, se conectará Arduino la masa o GND y la salida de Arduino PWM se conectará a la placa que se ha diseñado en la conexión de SA (sensor analógico).

Quedando como en esta ilustración real:

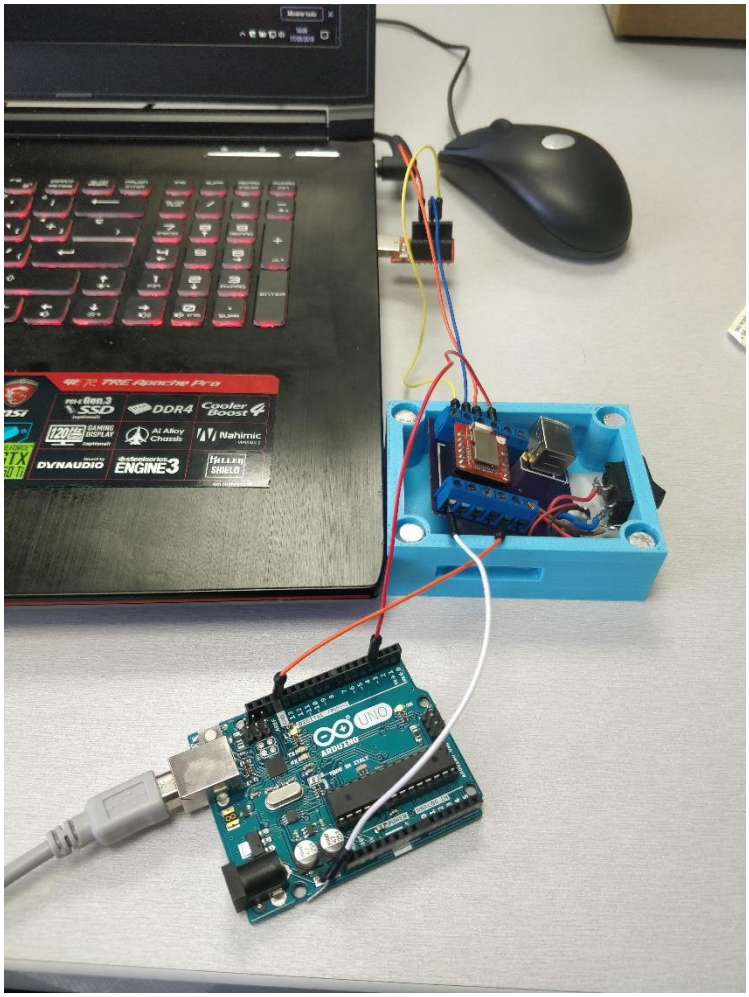


Ilustración 54 Conexión PC-USBDAP-Placa-Arduino.

Utilizando Putty en el ordenador se generan los siguientes resultados:

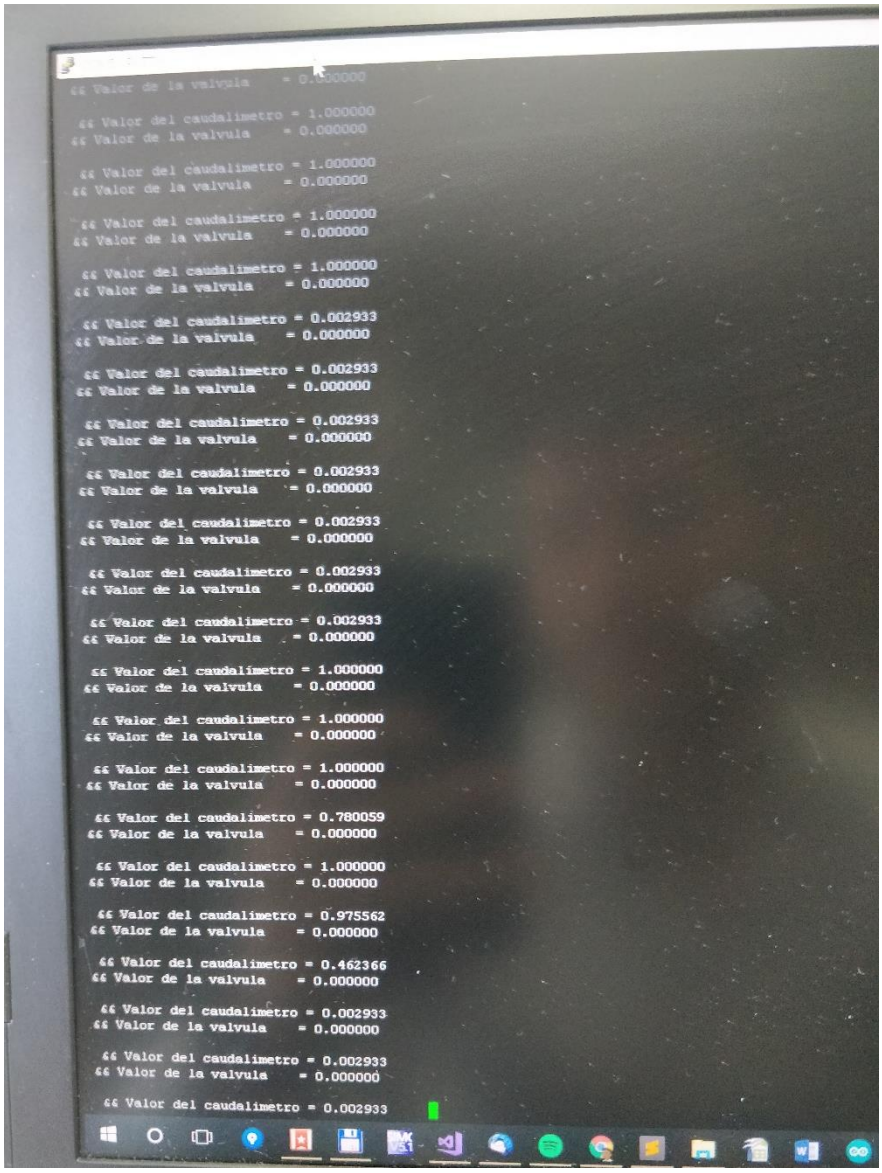


Ilustración 55 Resultados PWM Sensor Caudal Putty.

Como se puede observar, el valor de la válvula será 0 y el valor de la señal de caudal va variando, ya que se ha simulado en un sketch de Arduino como va variando con un delay determinado para acercar la señal digital a una analógica.

El sketch sería el siguiente:

```
const int Flow = 3;



void setup() {
  // put your setup code here, to run once:
pinMode (Flow, OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
   for (int i=0; i <= 168; i++){
      analogWrite(Flow, i);
      delay(100);
   }

}
```

Ilustración 56 Sketch PWM Arduino

Vemos en la anterior imagen, como se ha configurado una variable global constante, la cual será el valor de la salida digital PWM, en este caso la 3.

Seguidamente en el bloque Setup se configura como salida esta señal PWM y en el bloque Loop o de repetición se va a repetir infinitas veces un bucle for que dará a la señal un valor de voltaje. Si 5v corresponde al valor 255 y queremos que varíe entre 3.3 y 0 voltios, haciendo una sencilla regla de tres 3.3 voltios sería aproximadamente 168, por ello, se varía la tensión desde 0 a 3.3 voltios incrementando el contador desde 0 a 168 para conseguir la señal deseada.

Para la simulación de la válvula es más sencillo, habría que configurar una salida de voltaje constante, es decir una señal digital del Arduino. Para este caso, el sketch es el siguiente:

```
const int Flow = 3;
const int Valve = 4;


void setup() {
  // put your setup code here, to run once:
pinMode (Flow, OUTPUT);
pinMode (Valve, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
   for (int i=0; i <= 168; i++){
      analogWrite(Flow, i);
      delay(100);
   }
  digitalWrite (Valve, 1);
}
```

Ilustración 57 Señal Válvula Digital Arduino

Como se puede observar para este caso se ha añadido la salida cuatro, que es una salida digital. Arduino por una salida digital sólo puede sacar 5 voltios, en realidad un poco menos por las pérdidas que se producen en su trayecto.

Una foto que ilustra la prueba realizada sería la siguiente:

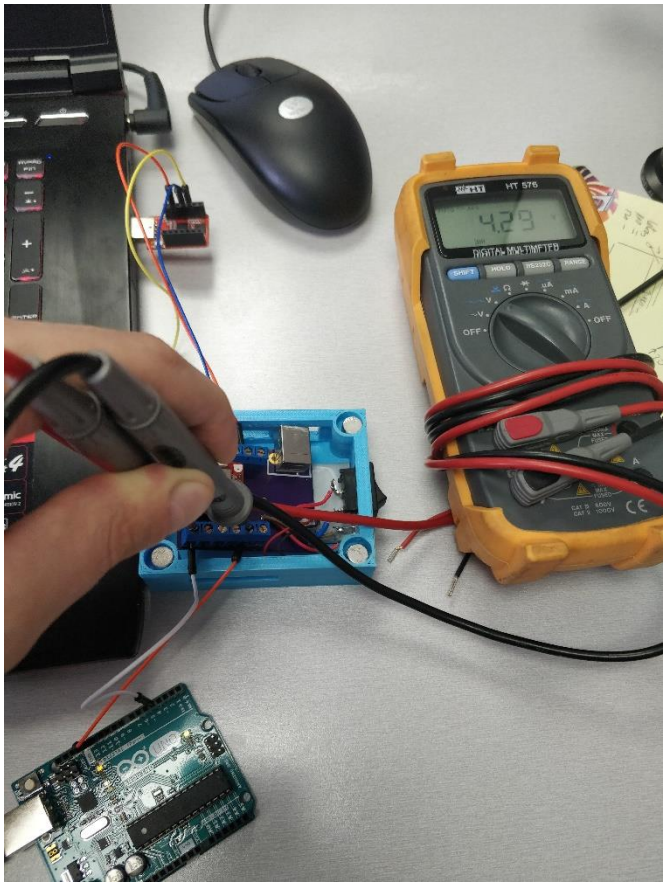


Ilustración 58 Conexión y prueba Válvula Arduino.

Como se puede observar no llega a los 5 voltios que deberían, aun así, para las pruebas podemos observar que efectivamente el valor cambia a 1.

Capítulo 7

7. Conclusiones y líneas futuras.

En este último capítulo se van a desarrollar las conclusiones y las líneas futuras del proyecto.

7.1 Conclusiones.

Durante este trabajo de fin de grado se ha pretendido desarrollar un prototipo con un dispositivo bluetooth de bajo coste basado en el dispositivo BLE Nano.

Se ha desarrollado una plataforma de aplicación en el campo de la agricultura para la transmisión de datos de forma inalámbrica mediante bluetooth, en este caso de los sensores de caudal, presión y estado de la válvula.

Para cumplir con los objetivos de este proyecto se ha estudiado la plataforma BLE Nano para dar los mejores resultados, usando las mejores plataformas de desarrollo software y hardware explicadas a lo largo de la memoria.

En este dispositivo se ha desarrollado en diferentes plataformas. En el caso del desarrollo hardware se ha utilizado la plataforma Mbed, programando el sistema de bluetooth y modbus. Para el desarrollo hardware se ha utilizado una de las mejores herramientas actuales, Altium Designer y para la aplicación de la interfaz con el operador se ha desarrollado en Android Studio, completando así el prototipo deseado y logrando los objetivos a cumplir al principio del desarrollo.

Sobre este prototipo se han realizado una serie de pruebas simuladas para la obtención de datos útiles y la obtención de información que ayudará a mejoras futuras y a posibles imprevistos.

Las conclusiones obtenidas para cada una de las secciones o aspectos de este trabajo son:

- La identificación del problema actual de recogida de datos por parte de los operadores de campo, obteniendo así una necesidad de mejora en el sector. Una vez identificado el problema se ha propuesto una solución en forma de este prototipo, concluyendo con éxito dicha solución.

- Se han afianzado conocimientos que se han dado a lo largo de la carrera Ingeniería Electrónica Industrial y Automatización, con el uso de electrónica como BLE Nano y los acondicionamientos de señal.

- La fabricación del prototipo PCB con un módulo de carga de batería ha pulido los conocimientos adquiridos teóricos, ya que, es un sector en el que se tiene poca experiencia a nivel físico y práctico y no solamente teórico como se trata en el grado.

- El desarrollo de la programación en Android y en Mbed proporciona conocimientos medios de programación en java y además en el uso del protocolo bluetooth, que actualmente tiene mucha presencia como método de transmisión de datos.

- Por último, la elección de los componentes que se han usado en la fabricación de la PCB también ha conferido conocimientos en el estudio de los datasheets y la capacidad de decisión de elección de componentes acorde a sus características.

En conclusión, se ha concluido que se han cumplido la mayoría de los objetivos iniciales dando por satisfactorio la realización del proyecto.

7.2 Líneas Futuras.

En este apartado se explicará las posibles líneas futuras que se pueden afrontar para continuar con el desarrollo del proyecto y crear una solución robusta, estable y funcional.
Pero para esta tarea hace falta distinguir entre varios frentes, los cuales debemos atajar para las posibles mejoras futuras.

- **Hardware:** En este caso, el hardware está bien conseguido, las mejoras deberían ser a través del uso continuado y ver las faltas y necesidades que vaya requiriendo el sistema.

- **Encapsulado:** El encapsulado está hecho mediante una impresora 3D, por lo que podría no ser muy recomendable en lugares que sea indispensable una protección oficial de IP67-IP68.

- **Software:** Al igual que el hardware, para este primer prototipo está bien conseguido, pero obviamente necesita de constante desarrollo para las necesidades que va abriendo el proyecto y facilidades al operador.

Capítulo 8

8. Bibliografía.

- Documentación del internet de las cosas:

http://www.muylinux.com/2017/08/01/internet-de-las-cosas-claves/

http://www.computerworld.es/negocio/el-boom-del-internet-de-las-cosas

- Documentación del datasheet del módulo MCP73831T:

https://www.sparkfun.com/datasheets/Prototyping/Batteries/MCP73831T.pdf

- Información sobre la tecnología inalámbrica y WiFi.

https://www.adslzone.net/2016/01/25/wifi-abgnacahad-que-significan-estas-siglas/

https://www.lifewire.com/g00/wireless-standards-802-11a-802-11b-g-n-and-802-11ac-816553?i10c.referrer=https%3A%2F%2Fwww.google.es%2F

- Información sobre la tecnología actual y estudios.

http://www.madrimasd.org

- Documentación técnica del Ble Nano Kit.

http://redbearlab.com/blenano/

- Información sobre el uso de Mbed y compilador.

https://www.mbed.com/en/

- Documentación acerca de Altium Designer.

http://www.altium.com

- Documentación sobre Android Studio.

https://androidstudiofaqs.com

- Página de información y uso de tinkercad.

https://www.tinkercad.com/

- Documentación acerca del modbus.

http://www.modbus.org/

- Protocolo Modbus.

http://www.ni.com/white-paper/52134/es/

- Django.

https://www.djangoproject.com/

Anexo I

Modelo de negocio CANVAS.

El modelo de negocio canvas es una manera de expresar la idea de negocio del proyecto, este sería el modelo canvas que se ha pensado para este proyecto:

Business Model Canvas

Diseñado para:	Universidad Politécnica de Cartagena
Diseñado por:	José Benavente Pérez

Relaciones Clave

* Relación con la universidad de agrónomos, identificado el problema de los clientes potenciales.

* Relaciones con agrícolas que identifican el problema alrededor de Cartagena.

Actividades Clave

* Programación de objetivos en la mejora de software y hardware de la solución.

* Programación de objetivos trimestrales para logros de objetivos ejecutada por el líder de operaciones.

Recursos Clave

* Equipo de desarrollo que mantenga la plataforma en la nube.

* Equipo de atención al cliente para atender a los usuarios

* Plataformas tecnológicas, servidores y herramientas de trabajo (estaciones de trabajos).

Propuesta de Valor

* Novedad y empleo de la tecnología en un ambiente poco actualizado tecnológicamente.

* Diseño sencillo y escueto, posible instalación en prácticamente cualquier lugar.

* Facilidad de uso, mediante un móvil o tablet y larga duración de batería (poco mantenimiento).

* Fácil instalación, cualquier persona podría instalarlo y ponerlo en funcionamiento.

* Solución a fallos de recogida datos manual y el añadido de poder repasar los datos en la nube en cualquier momento.

Relaciones con los clientes

* Ofrecer un servicio de atención al cliente que sea cercano al cliente, con respuestas efectivas y eficaces.

* Seguimiento del agente de ventas para validar el servicio.

Canales de Distribución

* En primera instancia tendrá poca presencia en el mercado para tener una amplia experiencia en el funcionamiento y posibles fallos no esperados.

* Primeramente se podrán hacer pruebas para el comprador y poder hacerse un hueco en el mercado.

* El soporte se hará en el mismo momento de la venta y post-venta, mediante un servicio online y telefónico.

Mercado Meta - Clientes

* Los clientes y el mercado está únicamente en el ámbito agrícola, por geolocalización hay mucha demanda de este sector.

* Mercado reacio a tecnologías, por lo que hay que hacer pruebas con el cliente y ver la sencillez y funcionamiento del

* Una vez instalados una cantidad respetable, grandes empresas podrían interesarse en este dispositivo y ampliar el mercado.

Estructura de Costos

* Costos del mantenimiento de servidores y programación de los mismos.

* Fabricación hardware de la PCB.

* Costo en primera instancia del dispositivo BLE Nano, posterirmente fabricarlo incrustado en la PCB.

PUNTO DE EQUILIBRIO

Flujos de Ingresos

* Pago del mantenimiento de servidores y mejoras en el software anualmente.

* Pago por cada uno de los dispositivos físicos instalados.

* Pago anual según el número de dispositivos instalados para su mantenimiento en campo abierto.

Anexo II

Código de Aplicación Android.

El código que se ha usado en las distintas clases de la aplicación Android que se han mencionado anteriormente, es el siguiente:

- Para Android manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.Jose.arquetanatureble" >

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="26" />

    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
/>
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="true" >
    </uses-feature>

    <application
        android:name=".CORE.BLE_Application"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <service android:name=".CORE.ServiceDetectionTag"
                 android:process=":Nature_ScannerBLE"/>

        <activity
            android:name=".GUI.ScanActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
        <activity
            android:name=".GUI.DeviceActivity"
            android:label="@string/title_activity_sensor"
            android:screenOrientation="portrait" >
        </activity>

        <activity
            android:name=".GUI.ResumenActivity"
            android:label="@string/title_activity_resumen"
            android:screenOrientation="portrait" >
        </activity>

        <activity
            android:name=".GUI.OutsideChamberActivity"
            android:label="@string/title_activity_outside_chamber"
            android:screenOrientation="portrait">
        </activity>

        <activity
            android:name=".GUI.InsideChamberActivity"
            android:label="@string/title_activity_inside_chamber"
            android:screenOrientation="portrait">
        </activity>

        <receiver
            android:name=".BLE.BLEBroadcastReceiver"
            android:enabled="true" >
            <intent-filter>
                <action
android:name="es.Jose.arquetanatureble.DEVICE_FOUND" />
            </intent-filter>
        </receiver>

        <receiver

android:name=".CORE.ServiceDetectionTag$ServiceBroadcastReceiver"
            android:enabled="true" >
            <intent-filter>
                <action
android:name="es.Jose.arquetanatureble.NOTIFY_NEW_DEVICE" />
            </intent-filter>
        </receiver>

    </application>
</manifest>
```

- **Para BeanBluetoothDevice:**

```
package es.Jose.arquetanatureble.BEAN;

import android.bluetooth.BluetoothDevice;
import android.os.Parcel;
import android.os.Parcelable;
import android.text.TextUtils;
import android.util.Log;

import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.List;

import es.Jose.arquetanatureble.UTIL.Constant;

//serializable class object to move between 2 Bluetooth activities ..
public class BeanBluetoothDevice implements Parcelable
{
    private BluetoothDevice mdevice;
    private int mRssi;
    private byte[] mScanRecord;

    public BeanBluetoothDevice() {
        super();
    }


//#########################################################################

    protected BeanBluetoothDevice(Parcel in) {
        mdevice     =
in.readParcelable(BluetoothDevice.class.getClassLoader());
        mRssi       = in.readInt();
        mScanRecord = in.createByteArray();
    }

    public static final Creator<BeanBluetoothDevice> CREATOR = new
Creator<BeanBluetoothDevice>() {
        @Override
        public BeanBluetoothDevice createFromParcel(Parcel in) {
            return new BeanBluetoothDevice(in);
        }

        @Override
        public BeanBluetoothDevice[] newArray(int size) {
            return new BeanBluetoothDevice[size];
        }
    };


//#########################################################################
    /****************** gets and sets methods
*********************/

//#########################################################################

    public void setBluetoothDevice(BluetoothDevice device)
    {mdevice = device;}

    public BluetoothDevice getBluetoothDevice()
    {return mdevice;}

    public int getmRssi() {
        return mRssi;
    }

    public void setmRssi(int mRssi) {
        this.mRssi = mRssi;
    }

    public byte[] getmScanRecord() {
```

```
        return mScanRecord;
    }

    public void setmScanRecord(byte[] mScanRecord) {
        this.mScanRecord = mScanRecord;
    }


    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeParcelable(mdevice, flags);
        dest.writeInt(mRssi);
        dest.writeByteArray(mScanRecord);
    }
```

- **Para BeanInformes.**

```
package es.Jose.arquetanatureble.BEAN;

import android.bluetooth.BluetoothDevice;
import android.os.Parcel;
import android.os.Parcelable;
import android.text.TextUtils;
import android.util.Log;

import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import es.Jose.arquetanatureble.UTIL.Constant;

//serializable class object to move between 2 Bluetooth activities ..
public class BeanBluetoothDevice implements Parcelable
{
    private BluetoothDevice mdevice;
    private int mRssi;
    private byte[] mScanRecord;

    public BeanBluetoothDevice() {
        super();
    }


//#####################################################################

    protected BeanBluetoothDevice(Parcel in) {
        mdevice     =
in.readParcelable(BluetoothDevice.class.getClassLoader());
        mRssi       = in.readInt();
        mScanRecord = in.createByteArray();
    }
```

```
    public static final Creator<BeanBluetoothDevice> CREATOR = new
Creator<BeanBluetoothDevice>() {
        @Override
        public BeanBluetoothDevice createFromParcel(Parcel in) {
            return new BeanBluetoothDevice(in);
        }

        @Override
        public BeanBluetoothDevice[] newArray(int size) {
            return new BeanBluetoothDevice[size];
        }
    };


//######################################################################
    /****************** gets and sets methods
*********************/

//######################################################################

    public void setBluetoothDevice(BluetoothDevice device)
    {mdevice = device;}

    public BluetoothDevice getBluetoothDevice()
    {return mdevice;}

    public int getmRssi() {
        return mRssi;
    }

    public void setmRssi(int mRssi) {
        this.mRssi = mRssi;
    }

    public byte[] getmScanRecord() {
        return mScanRecord;
    }

    public void setmScanRecord(byte[] mScanRecord) {
        this.mScanRecord = mScanRecord;
    }


    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeParcelable(mdevice, flags);
        dest.writeInt(mRssi);
        dest.writeByteArray(mScanRecord);
    }
```

- **Para AdRecord.**

```
package es.Jose.arquetanatureble.BLE;

import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.List;


public class AdRecord {

    /* An incomplete list of the Bluetooth GAP AD Type identifiers */
    public static final int TYPE_FLAGS                                  =
0x1;
    public static final int TYPE_UUID16_INC                             =
0x2;
    public static final int TYPE_UUID16                                 =
0x3;
    public static final int TYPE_UUID32_INC                             =
0x4;
    public static final int TYPE_UUID32                                 =
0x5;
    public static final int TYPE_UUID128_INC                            =
0x6;
    public static final int TYPE_UUID128                                =
0x7;
    public static final int TYPE_NAME_SHORT                             =
0x8;
    public static final int TYPE_NAME                                   =
0x9;
    public static final int TYPE_TRANSMITPOWER                          =
0xA;
    public static final int TYPE_DEVICE_ID                              =
0x10;
    public static final int TYPE_CONNINTERVAL                           =
0x12;
    public static final int TYPE_SERVICEDATA                            =
0x16;
    public static final int TYPE_APPEARANCE                             =
0x19; /**< \ref Appearance */
    public static final int TYPE_ADVERTISING_INTERVAL                   =
0x1A; /**< Advertising Interval */
    public static final int TYPE_MANUFACTURER_SPECIFIC_DATA             =
0xFF;  /**< Manufacturer Specific Data */


    /*
     * Read out all the AD structures from the raw scan record
     */
    public static List<AdRecord> parseScanRecord(byte[] scanRecord) {
        List<AdRecord> records = new ArrayList<AdRecord>();

        int index = 0;
        while (index < scanRecord.length)
        {
            int length = scanRecord[index++] & 0xFF;
            //Done once we run out of records
            if (length == 0) break;

            int type = scanRecord[index] & 0xFF;
            //Done if our record isn't a valid type
            if (type == 0) break;


            byte[] data = Arrays.copyOfRange(scanRecord, index + 1,
index + length);
```

```
            records.add(new AdRecord(length, type, data));
            //Advance
            index += length;
        }

        return records;
    }

    /* Helper functions to parse out common data payloads from an AD
structure */

    public static String getName(AdRecord nameRecord) {
        return new String(nameRecord.mData);
    }

    public static int getServiceDataUuid(AdRecord serviceData) {
        if (serviceData.mType != TYPE_SERVICEDATA) return -1;

        byte[] raw = serviceData.mData;
        //Find UUID data in byte array
        int uuid = (raw[1] & 0xFF) << 8;
        uuid += (raw[0] & 0xFF);

        return uuid;
    }

    public static byte[] getServiceData(AdRecord serviceData) {
        if (serviceData.mType != TYPE_SERVICEDATA) return null;

        byte[] raw = serviceData.mData;
        //Chop out the uuid
        return Arrays.copyOfRange(raw, 2, raw.length);
    }

    /* Model Object Definition */

    private int mLength;
    private int mType;
    private byte[] mData; //unsigned 8 bits

    public AdRecord(int length, int type, byte[] data)
    {
        mLength = length;
        mType = type;
        mData = data;
    }

    public int getLength()
    {
        return mLength;
    }

    public int getType()
    {
        return mType;
    }

    public byte[] getData(){return mData;};

    /*
```

```
public static void printScanRecord (int[] scanRecord)
    {
        // Simply print all raw bytes
        try {
            String decodedRecord = new String(scanRecord,"UTF-8");
            if(Constant.DEBUG)
                Log.i(Constant.TAG, "AdRecord -- printScanRecord ->
decoded String : " + ByteArrayToString(scanRecord));
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

        // Parse data bytes into individual records
        List<AdRecord> records = AdRecord.parseScanRecord(scanRecord);


        // Print individual records
        if (records.size() == 0)
        {
            if(Constant.DEBUG)
                Log.i(Constant.TAG, "AdRecord -- printScanRecord ->
Scan Record Empty");
        } else {

            if(Constant.DEBUG)
                Log.i(Constant.TAG, "AdRecord -- printScanRecord ->
Scan Record: " + TextUtils.join(",", records));
        }

    }*/

    public static String ByteArrayToString(byte[] ba)
    {
        StringBuilder hex = new StringBuilder(ba.length * 2);
        for (byte b : ba)
            hex.append(b + " ");

        return hex.toString();
    }

    @Override
    public String toString() {
        switch (mType) {
            case TYPE_FLAGS:
                return "Flags";
            case TYPE_NAME_SHORT:
            case TYPE_NAME:
                return "Name";
            case TYPE_UUID16:
            case TYPE_UUID16_INC:
            case TYPE_UUID32:
            case TYPE_UUID32_INC:
            case TYPE_UUID128:
            case TYPE_UUID128_INC:
                return "UUIDs";
            case TYPE_DEVICE_ID:
                return "Device ID";
            case TYPE_TRANSMITPOWER:
                return "Transmit Power";
            case TYPE_CONNINTERVAL:
```

```
                return "Connect Interval";
            case TYPE_SERVICEDATA:
                return "Service Data";
            case TYPE_APPEARANCE:
                return "Appearance";
            case  TYPE_ADVERTISING_INTERVAL:
                return "Advertising interval";
            case TYPE_MANUFACTURER_SPECIFIC_DATA:
                return "Manufacturer specific data";
            default:
                return "Unknown Structure: "+mType;
        }
    }
}
```

- **Para BLEBroadcastReceiver.**

```
package es.Jose.arquetanatureble.BLE;

import android.app.Activity;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;
import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.BLE.HandlerBLE;
import es.Jose.arquetanatureble.CORE.ServiceDetectionTag;
import es.Jose.arquetanatureble.GUI.ScanArrayAdapter;
import es.Jose.arquetanatureble.UTIL.Constant;


public class BLEBroadcastReceiver extends BroadcastReceiver
{
    private static Activity mActivity;
    private static ScanArrayAdapter mAdapter;
    private static Context mContext;

    public BLEBroadcastReceiver(Activity activity, ScanArrayAdapter
adapter)
    {
        super();
        mAdapter  = adapter;
        mActivity = activity;
        mContext  = activity.getApplicationContext();
    }

    public BLEBroadcastReceiver(Context context)
    {
        super();
        mContext = context;
    }

    public BLEBroadcastReceiver()
    {
        super();
    }

    @Override
    public void onReceive(Context context, Intent intent)
```

```
    {
        if(Constant.DEBUG)
            Log.i(Constant.TAG, "BLEBroadcastReceiver(Listener) --
OnReceive() -> BroadcastReceiver new device found.");

        String action = intent.getAction();
        final BeanBluetoothDevice beanDeviceFound =
intent.getExtras().getParcelable(Constant.EXTRA_BEAN_BLUETOOTHDEVICE);
        //byte[] scanRecord = beanDeviceFound.getmScanRecord();
        //get signal and add new device into MyarrayAdapter

        //if(action.equals(HandlerBLE.ACTION_BLE_FOUND))
        try
        {

            //mActivity = ScanActivity.GetScanActivity();

//((ScanActivity)mActivity).addNewDevice2MySimpleArrayAdapter(beanDevi
ceFound);

            final BluetoothDevice deviceFound =
beanDeviceFound.getBluetoothDevice();


            /*if(Constant.DEBUG)
                Log.i(Constant.TAG, "BLEBroadcastReceiver -- onReceive
-> ScanRecord value: "
                        +byteArray2Hexadecimal(scanRecord));*/

//((ScanActivity)mActivity).addNewDevice2MySimpleArrayAdapter(beanDevi
ceFound);

            mActivity.runOnUiThread(new Runnable() {
                @Override
                public void run() {

                    BeanBluetoothDevice oldDevice;
                    oldDevice =
mAdapter.findElement(beanDeviceFound.getBluetoothDevice().getName()
                            ,
beanDeviceFound.getBluetoothDevice().getAddress());

                    if (oldDevice != null) {
                        //delete old one and inset new one.
                        mAdapter.deleteElement(oldDevice);
                    }

                    mAdapter.addElement(beanDeviceFound);

                    if (Constant.DEBUG)
                        Log.i(Constant.TAG, "mActivity.runOnUiThreadr
-- runnable() -> Added new device "
                                +
beanDeviceFound.getBluetoothDevice().getAddress()
                                + " --- Name: " +
beanDeviceFound.getBluetoothDevice().getName());
                }
            });
        } catch (Exception e) {
```

```
            Log.i(Constant.TAG, "[Error(BLEBroadcastReceiver)]: \n
Message: " + e.getMessage() + "\n Cause:" + e.getCause() + "\n
StackTrace" + e.getStackTrace() + "\n" + e.getLocalizedMessage());
        }


    }
}
```

- **Para HandlerBLE.**

```
package es.Jose.arquetanatureble.BLE;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothAdapter.LeScanCallback;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothManager;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.CORE.ServiceDetectionTag;
import es.Jose.arquetanatureble.UTIL.Constant;



public class HandlerBLE implements LeScanCallback
{
    public static final String ACTION_BLE_FOUND           =
"es.Jose.arquetanatureble.BLE_FOUND";

    private static HandlerBLE mHandlerBLE;
    private static List<ServiceType> mServices;
    private static Context mContext;
    private static MyCallBack mCallBack;
    private static BluetoothDevice mDevice;
    private static String mDeviceAddress;
    private static BluetoothGatt mGatt;
    private static BluetoothAdapter mBlueAdapter = null;

    public static boolean isScanning = false;


//#####################################################################
    /****************** Constructor
*********************/

//#####################################################################
    public HandlerBLE(Context context)
    {
        mContext        = context;
```

```
        mDeviceAddress  = null;
        mServices       = new ArrayList<ServiceType>();
        mCallBack       = new MyCallBack(context, this);
    }


//#####################################################################
    /****************** Getters & setters
*********************/

//#####################################################################

    public MyCallBack getMyCallBack() {return mCallBack;}
    public List<ServiceType> getServices()
    {
        return mServices;
    }

    public ServiceType getService(UUID uuid)
    {
        for (ServiceType service:mServices)
        {

if(service.getService().getUuid().toString().equals(uuid.toString()))
                return service;
        }
        return null;
    }
    public void addService(ServiceType serviceType)
    {
        mServices.add(serviceType);
    }
    public void setGatt(BluetoothGatt gatt){mGatt = gatt;}
    public BluetoothGatt getGatt(){return mGatt;}

//#####################################################################
    /*********************  statics methods
************************/

//#####################################################################
    public static HandlerBLE getInstance(Context context) {
        if(mHandlerBLE==null){
            mHandlerBLE = new HandlerBLE(context);
            setup();
        }
        return mHandlerBLE;
    }

    public static void  resetHandlerBLE()
    {
        mDeviceAddress     = null;
        mGatt              = null;
        if (mServices!=null) mServices.clear();
        disconnect();
    }

    public static void setup()
    {
        BluetoothManager manager = (BluetoothManager)
mContext.getSystemService(Context.BLUETOOTH_SERVICE);
        mBlueAdapter = manager.getAdapter();
```

```
    }


//#########################################################################
    /********************* methods bluetooth
************/

//#########################################################################

    public boolean IsEnabledBlue()
    {
        if(mBlueAdapter != null)
            return mBlueAdapter.isEnabled();

        return false;
    }

    public BluetoothAdapter getBlueAdapter()
    {
        return mBlueAdapter;
    }

    public void startLeScan()
    {
        try
        {
            mBlueAdapter.startLeScan(this);//deprecated
            isScanning = true;
            //mBlueAdapter.startDiscovery();
        }
        catch (Exception e)
        {

Log.i(Constant.TAG,"(HandlerBLE)[Error]:"+e.getStackTrace()+"
"+e.getCause()+" "+e.getMessage()+
                    " "+e.getLocalizedMessage());
        }

    }

    public void stopLeScan()
    {
        try
        {
            mBlueAdapter.stopLeScan(this); //deprecated
            //mBlueAdapter.startDiscovery();
            isScanning = false;
        }
        catch(Exception e)
        {

Log.i(Constant.TAG,"(HandlerBLE)[Error]:"+e.getStackTrace()+"
"+e.getCause()+" "+e.getMessage()+
                    " "+e.getLocalizedMessage());
        }
    }


//#########################################################################
    /********************* methods GATT bluetooth
************/
```

```
//#####################################################################
    public void discoverServices()
    {
        if (Constant.DEBUG)
            Log.i(Constant.TAG, "(HandlerBLE)Scanning services and
caracteristics");
        mGatt.discoverServices();
    }

    public void connect()
    {
        mDevice    = mBlueAdapter.getRemoteDevice(mDeviceAddress);
        mServices.clear();
        if(mGatt!=null)
        {
            mGatt.connect();
        }else
        {
            mDevice.connectGatt(mContext, false, mCallBack);
        }
    }

    public void connect(Context context, MyCallBack callBack,String
deviceAddress)
    {
        mDevice = mBlueAdapter.getRemoteDevice(deviceAddress);
        mDevice.connectGatt(context, false, callBack);
    }

    public static void disconnect()
    {
        if (mGatt!=null)
        {
            try
            {
                mGatt.disconnect();
                mGatt.close();
                if (Constant.DEBUG)
                    Log.i(Constant.TAG, "(HandlerBLE)Disconnecting
GATT");
            } catch(Exception ex)
            {
                if (Constant.DEBUG)
                    Log.i(Constant.TAG, "(HandlerBLE) Error
disconnecting :"+ex.getMessage());
            }
        }
        mGatt = null;
    }

    public boolean isConnected(){
        return (mGatt!=null);
    }

    public void getDataFromSensors()
    {
        if(mCallBack != null && mGatt != null)
        {
            mCallBack.resetStateMachine();
            mCallBack.readDataFromSensor(mGatt);
```

```
        }
    }


//#####################################################################
    /********************* methods Scan bluetooth
*************/

//#####################################################################
    /*
    * this method is used to receive devices which were found durind a
scan*/
    @Override
    public void onLeScan(BluetoothDevice device, int rssi, byte[]
scanRecord)
    {

        if(Constant.DEBUG)
            Log.i(Constant.TAG,"(HandlerBLE) -- onLeScan -> throwing
information to the listener.");

        BeanBluetoothDevice beanBlue = new BeanBluetoothDevice();
        beanBlue.setBluetoothDevice(device);
        beanBlue.setmRssi(rssi);
        beanBlue.setmScanRecord(scanRecord);

        //create the packet wich will be sent to listener.
        Intent intent = new Intent();
        intent.setAction(HandlerBLE.ACTION_BLE_FOUND);
        intent.putExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE,beanBlue);
        mContext.sendBroadcast(intent);

        //listener in background
        Intent newIntent = new Intent();

newIntent.setAction(ServiceDetectionTag.ServiceBroadcastReceiver.ACTIO
N_NOTIFY_NEW_DEVICE_FOUND);

newIntent.putExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE,beanBlue);
        mContext.sendBroadcast(newIntent);
    }
}
```

- **Para MyCallback.**

```
package es.Jose.arquetanatureble.BLE;

import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothProfile;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

import java.util.List;
```

```
import java.util.UUID;

import es.Jose.arquetanatureble.GUI.DeviceActivity;
import es.Jose.arquetanatureble.UTIL.Constant;

public class MyCallBack extends BluetoothGattCallback {


    private HandlerBLE mHandlerBLE;
    private Context mContext;
    // State machine
    private int mState;
    private boolean nextSensor;

    public MyCallBack(Context mContext, HandlerBLE mHandlerBLE)
    {
        this.mContext    = mContext;
        this.mHandlerBLE = mHandlerBLE;
        mState           = 0;
        nextSensor       = true;
    }

    protected void resetStateMachine()
    {
        mState           = 0;
        nextSensor       = true;
    }

    protected void sendBroadCast(Context context,String action,String
parameter,byte [] value)
    {
        Intent broadcastIntent = new Intent();
        broadcastIntent.setAction(action);
        broadcastIntent.putExtra(parameter, value);
        context.sendBroadcast(broadcastIntent);
    }

    protected void advanceStateMachine()
    {
        mState++;
    }

    protected void readDataFromSensor(BluetoothGatt gatt)
    {

        BluetoothGattCharacteristic characteristic;
        //BluetoothGattService service = new
BluetoothGattService(ServiceData.CHAMBER_SERVICE
        //        ,BluetoothGattService.SERVICE_TYPE_PRIMARY);

        BluetoothGattService service =
mHandlerBLE.getService(ServiceData.CHAMBER_SERVICE).getService();

        switch (mState)
        {
            case 0:
                characteristic =
service.getCharacteristic(ServiceData.FLOWMETER_CHARACTERISTIC);
                break;
            case 1:
```

```
                characteristic =
service.getCharacteristic(ServiceData.PRESSURE_CHARACTERISTIC);
                break;
            case 2:
                characteristic =
service.getCharacteristic(ServiceData.VALVE_CHARACTERISTIC);
                nextSensor     = false;
                break;
            default:
                return;
        }
        gatt.readCharacteristic(characteristic);
    }
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int
status, int newState) {
        super.onConnectionStateChange(gatt, status, newState);

        if(Constant.DEBUG){
            if (status!=0) {
                Log.i(Constant.TAG, "MyCallBack -- Error status
received onConnectionStateChange: " + status + " - Newstate: " +
newState);
            } else {
                Log.i(Constant.TAG, "MyCallBack --
onConnectionStateChange received. status = " + status +
                        " - NewState: " + newState);
            }
        }
        if(status != BluetoothGatt.GATT_SUCCESS)
        {
            gatt.disconnect();
            return;
        }

        //Connection established
        if (newState == BluetoothProfile.STATE_CONNECTED) {

            if(Constant.DEBUG)
                Log.i(Constant.TAG, "MyCallBack --
onConnectionStateChange -- discovering services.");

            mHandlerBLE.setGatt(gatt);
            //Discover services
            //gatt.discoverServices();
            mHandlerBLE.discoverServices();

        } else if (newState == BluetoothProfile.STATE_DISCONNECTED)
        {
            //Handle a disconnect event
            Log.d(Constant.TAG,"MyCallBack -- onConnectionStateChange
-- device disconnecting.");
        }

    }


    @Override
    public void onReadRemoteRssi(BluetoothGatt gatt, int rssi, int
status) {
        super.onReadRemoteRssi(gatt, rssi, status);
```

```
    }


    @Override
    public void onCharacteristicWrite(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic, int status) {
        super.onCharacteristicWrite(gatt, characteristic, status);
    }


    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic, int status) {
        super.onCharacteristicRead(gatt, characteristic, status);

        String uuid_charac = characteristic.getUuid().toString();
        byte[] value = characteristic.getValue();

        if(Constant.DEBUG)
            Log.i(Constant.TAG, "MyCallBack -- onCharacteristicRead: "
+ uuid_charac);


if(uuid_charac.equals(ServiceData.FLOWMETER_CHARACTERISTIC.toString())
)
        {
            //to send the information UI.
            sendBroadCast(mContext,
DeviceActivity.BroadcasterDataSensor.DATA_FLOW_ACT
                    ,Constant.MSG_FLOW,value);
        }
        else
if(uuid_charac.equals(ServiceData.PRESSURE_CHARACTERISTIC.toString()))
        {

sendBroadCast(mContext,DeviceActivity.BroadcasterDataSensor.DATA_PRESS
U_ACT
                    ,Constant.MSG_PRESSU,value);
        }
        else if
(uuid_charac.equals(ServiceData.VALVE_CHARACTERISTIC.toString()))
        {

sendBroadCast(mContext,DeviceActivity.BroadcasterDataSensor.DATA_VALVE
_ACT
                    ,Constant.MSG_VALVE,value);
        }
        else
        {

sendBroadCast(mContext,DeviceActivity.BroadcasterDataSensor.DATA_UNKNO
WN_ACT
                    ,Constant.MSG_UNKNOWN,value);
        }

        if(nextSensor)
        {
            advanceStateMachine();
            readDataFromSensor(mHandlerBLE.getGatt());
        }
    }
```

```
    @Override
    public void onCharacteristicChanged(BluetoothGatt
gatt,BluetoothGattCharacteristic characteristic) {
        super.onCharacteristicChanged(gatt, characteristic);

        String uuid_charac = characteristic.getUuid().toString();
        byte[] value = characteristic.getValue();

        if(Constant.DEBUG)
            Log.i(Constant.TAG, "MyCallBack --
onCharacteristicChanged: " + uuid_charac);


if(uuid_charac.equals(ServiceData.FLOWMETER_CHARACTERISTIC.toString())
)
        {
            //to send the information UI.
            sendBroadCast(mContext,
DeviceActivity.BroadcasterDataSensor.DATA_FLOW_ACT
                    ,Constant.MSG_FLOW,value);
        }
        else
if(uuid_charac.equals(ServiceData.PRESSURE_CHARACTERISTIC.toString()))
        {

sendBroadCast(mContext,DeviceActivity.BroadcasterDataSensor.DATA_PRESS
U_ACT
                    ,Constant.MSG_PRESSU,value);
        }
        else if
(uuid_charac.equals(ServiceData.VALVE_CHARACTERISTIC.toString()))
        {

sendBroadCast(mContext,DeviceActivity.BroadcasterDataSensor.DATA_VALVE
_ACT
                    ,Constant.MSG_VALVE,value);
        }
    }


    @Override
    public void  onServicesDiscovered(BluetoothGatt gatt, int status)
{
        super.onServicesDiscovered(gatt, status);

        //adding new services
        if(Constant.DEBUG)
            Log.i(Constant.TAG, "MyCallBack -- onServicesDiscovered
status: " + status);

        String serviceUUID;
        ServiceType serviceType;
        List<BluetoothGattCharacteristic> characteristics;
        for(BluetoothGattService serviceInList: gatt.getServices())
        {
            serviceUUID = serviceInList.getUuid().toString();
            serviceType = new ServiceType(serviceInList);
            characteristics = serviceType.getCharacteristics();
```

```
            if(Constant.DEBUG)
                Log.i(Constant.TAG, "MyCallBack --
onServicesDiscovered -- New service: " + serviceUUID);

            for(BluetoothGattCharacteristic characteristicInList :
serviceInList.getCharacteristics())
            {
                if(Constant.DEBUG)
                    Log.i(Constant.TAG, "MyCallBack --
onServicesDiscovered -- New characteristic: " +
characteristicInList.getUuid().toString());
                characteristics.add(characteristicInList);
            }

            serviceType.setCharacteristics(characteristics);
            mHandlerBLE.addService(serviceType);

            characteristics.clear();
        }

        ////////////////////////////////////
        //ServiceType serviceType1 =
mHandlerBLE.getService(ServiceData.CHAMBER_SERVICE);
        //for(BluetoothGattCharacteristic characteristicInList :
serviceType1.getService().getCharacteristics())
        //{
        //    if(Constant.DEBUG)
        //        Log.i(Constant.TAG, "MyCallBack -- test -- service
characteristic: " + characteristicInList.getUuid().toString());
        //
        //}
        ////////////////////////////////////

        resetStateMachine();
        readDataFromSensor(gatt);
    }


    @Override
    public void onDescriptorWrite(BluetoothGatt gatt,
BluetoothGattDescriptor descriptor, int status) {
        super.onDescriptorWrite(gatt, descriptor, status);
    }

}
```

- **Para Service Data:**

```
package es.Jose.arquetanatureble.BLE;

import java.util.UUID;

public final class ServiceData {
    /* mbed definition
    uint16_t customServiceUUID   = 0xA000;
    uint16_t ValveUUID           = 0xA001;
    uint16_t PressureUUID        = 0xA002;
    uint16_t FlowmeterUUID       = 0xA003;
     */
```

```
    public static final UUID CHAMBER_SERVICE           =
UUID.fromString("0000a000-0000-1000-8000-00805f9b34fb");
    public static final UUID FLOWMETER_CHARACTERISTIC  =
UUID.fromString("0000a001-0000-1000-8000-00805f9b34fb");
    public static final UUID PRESSURE_CHARACTERISTIC   =
UUID.fromString("0000a002-0000-1000-8000-00805f9b34fb");
    public static final UUID VALVE_CHARACTERISTIC      =
UUID.fromString("0000a003-0000-1000-8000-00805f9b34fb");

    private ServiceData()
    {super();}

    public static int []workFlometerData(byte[] value)
    {
        int[] data = new int[value.length];
        for(byte i=0;i < value.length;i++)
            data[i] = value[i] & 0xff;

        return data;
    }

    public static int[] workPressureData (byte[] value)
    {
        int[] data = new int[value.length];
        for(byte i=0;i < value.length;i++)
            data[i] = value[i] & 0xff;

        return data;
    }
    public static boolean workValveData (byte[] value)
    {
        int[] data = new int[value.length];
        for(byte i=0;i < value.length;i++)
            data[i] = value[i] & 0xff;

        return data[0] == 1? true:false;
    }

}
```

- **Para Service Type:**

```
package es.Jose.arquetanatureble.BLE;

import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;

import java.util.ArrayList;
import java.util.List;

public class ServiceType {
    private BluetoothGattService mService;
    private List<BluetoothGattCharacteristic> mCharacteristics;

    ServiceType(BluetoothGattService service)
    {
        mService = service;
        mCharacteristics= new
ArrayList<BluetoothGattCharacteristic>();
    }
```

```
    public BluetoothGattService getService() {return mService;}
    public List<BluetoothGattCharacteristic> getCharacteristics ()
{return mCharacteristics;}
    public void setCharacteristics(List<BluetoothGattCharacteristic>
characteristicList)
    {
        mCharacteristics = characteristicList;
    }
}
```

- **Para BLE_Application:**

```
package es.Jose.arquetanatureble.CORE;

import android.app.Application;
import android.content.Context;

import es.Jose.arquetanatureble.BLE.HandlerBLE;

public class BLE_Application extends Application
{
    static HandlerBLE mHandlerBLE;

    public void resetHandlerBLE()
    {
        HandlerBLE.resetHandlerBLE();
    }

    public HandlerBLE getmHandlerBLEInstance(Context context)
    {
        return mHandlerBLE = HandlerBLE.getInstance(context);
    }

    public static HandlerBLE getmHandlerBLE()
    {
        return mHandlerBLE;
    }

}
```

- **Para MyNotificationHandler:**

```
package es.Jose.arquetanatureble.CORE;



import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.app.TaskStackBuilder;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;


import es.Jose.arquetanatureble.GUI.ScanActivity;
import es.Jose.arquetanatureble.R;
```

```
public class MyNotificationHandler
{
    //private Notification mNotification;
    private static NotificationManager mNotificationManager;

    private static NotificationCompat.Builder mNotification;
    private static PendingIntent pIntent;
    private static Intent resultIntent;
    private static TaskStackBuilder stackBuilder;

    private static Context mContext;

    public static final String NOTIFICACION_MESSAGE   = "New device
found.";
    public static final String NOTIFICACION_TITLE     = "Device :
Chamber";
    public static final String NOTIFICACION_TICKER    = "Nueva Arqueta
detectada.";
    public static final int NOTIFICATION_ID           = 1100;

    public MyNotificationHandler(Context context)
    {
        super();
        //mNotification        = new Notification();
        mContext               = context;
    }

    public void SendNotify()
    {
        //old implementation
        /*Notification notification                =
CreateNotification();
        mNotificationManager =(NotificationManager)
mService.getSystemService(Context.NOTIFICATION_SERVICE);
        mNotificationManager.notify(NOTIFICATION_ID, notification);*/

        startNotification();
        mNotificationManager = (NotificationManager)
mContext.getSystemService(Context.NOTIFICATION_SERVICE);
        mNotificationManager.notify(NOTIFICATION_ID,
mNotification.build());
    }


    protected void startNotification()
    {
        //Creating Notification Builder
        mNotification = new NotificationCompat.Builder(mContext);
        //Title for Notification
        mNotification.setContentTitle(NOTIFICACION_MESSAGE);
        //Message in the Notification
        mNotification.setContentText(NOTIFICACION_TITLE);
        //Alert shown when Notification is received
        mNotification.setTicker(NOTIFICACION_TICKER);
        //Icon to be set on Notification
        mNotification.setSmallIcon(R.mipmap.ic_launcher);

        //Creating new Stack Builder
        stackBuilder = TaskStackBuilder.create(mContext);
        stackBuilder.addParentStack(ScanActivity.class);
        //Intent which is opened when notification is clicked
```

```
        resultIntent = new Intent(mContext, ScanActivity.class);
        stackBuilder.addNextIntent(resultIntent);
        pIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
        mNotification.setAutoCancel(true);
        mNotification.setContentIntent(pIntent);
    }
```

- **Para ServiceDetectionTag:**

```
package es.Jose.arquetanatureble.CORE;

import android.app.IntentService;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.util.Log;

import java.util.Timer;
import java.util.TimerTask;

import es.Jose.arquetanatureble.BLE.HandlerBLE;
import es.Jose.arquetanatureble.UTIL.Constant;


public class ServiceDetectionTag extends Service {
    private static final String NAMECLASS = "ServiceDetectionTag";
    private static final long STOP_SCAN_TIMER = 25 * 1000;
    private static final long START_SCAN_TIMER = 30 * 1000;

    private static Context mContext;
    private static HandlerBLE mHandlerBLE;
    private static ServiceBroadcastReceiver mServiceBroadcastReceiver;
    private static Thread workerThread = null;
    private static Boolean alive;

    public ServiceDetectionTag()
    {
        super();
    }

    /*private volatile Timer mTimerStart;
    private volatile Timer mTimerStop;*/

    public static Boolean isAlive()
    {
        return alive;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        alive = true;
```

```
        mHandlerBLE = ((BLE_Application)
getApplication()).getmHandlerBLEInstance(this.getApplicationContext())
;
        ((BLE_Application) getApplication()).resetHandlerBLE();
        mContext = getApplicationContext();

        mServiceBroadcastReceiver = new ServiceBroadcastReceiver();
        IntentFilter i = new
IntentFilter(ServiceBroadcastReceiver.ACTION_NOTIFY_NEW_DEVICE_FOUND);
        registerReceiver(mServiceBroadcastReceiver, i);

        if (Constant.DEBUG)
            Log.i(Constant.TAG, NAMECLASS + " ## -- Init Proccess");

            workerThread = new Thread(new Runnable() {
                public void run()
                {
                        while (isAlive())
                        {
                            try
                            {
                                if (Constant.DEBUG)
                                    Log.i(Constant.TAG, NAMECLASS + "
## -- onCreate -> Start scanning");

                                mHandlerBLE.startLeScan();
                                workerThread.sleep(START_SCAN_TIMER);

                                if (Constant.DEBUG)
                                    Log.i(Constant.TAG, NAMECLASS + "
## -- onCreate -> Stop scanning");
                                mHandlerBLE.stopLeScan();
                                workerThread.sleep(STOP_SCAN_TIMER);

                            }
                            catch (Exception e)
                            {
                                e.printStackTrace();
                            }
                        }
                }));
                workerThread.start();
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onDestroy ()
    {
        alive = false;

        if (Constant.DEBUG)
            Log.i(Constant.TAG, NAMECLASS + " ## -- onDestroy -> Stop
scanning");
        mHandlerBLE.stopLeScan();

        try {
```

```
            workerThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        unregisterReceiver(mServiceBroadcastReceiver);
        super.onDestroy();
    }

//##################################################################################
#########################

//##################################################################################
#########################
    public static class ServiceBroadcastReceiver extends
BroadcastReceiver {
            public static final String ACTION_NOTIFY_NEW_DEVICE_FOUND
= "es.Jose.arquetanatureble.NOTIFY_NEW_DEVICE";
            public ServiceBroadcastReceiver(){}
            @Override
            public void onReceive(Context context, Intent intent) {
                if (Constant.DEBUG)
                    Log.i(Constant.TAG, " ##
ServiceBroadcastReceiver(Listener) -- onReceive ");
                String action = intent.getAction();
                if
(action.equals(ServiceBroadcastReceiver.ACTION_NOTIFY_NEW_DEVICE_FOUND
)) {

                    if(mContext != null )
                    {
                        if (Constant.DEBUG)
                            Log.i(Constant.TAG, " ##
ServiceBroadcastReceiver -- onReceive -> sending
notication(statusBar)");

                        MyNotificationHandler myNotificationHandler;
                        myNotificationHandler = new
MyNotificationHandler(mContext);
                        myNotificationHandler.SendNotify();
                    }
                }
            }

        }
}
```

- **Informes SQLiteDataSource:**

```
package es.Jose.arquetanatureble.DB_SQLITE;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import java.util.ArrayList;
import java.util.List;

import es.Jose.arquetanatureble.BEAN.BeanInformesDB;
```

```
public class InformeSQLiteDataSource
{
    //Metainformación de la base de datos
    public static final String INFORMES_TABLE_NAME = "INFORMES";
    public static final String VARCHAR_TYPE        = "varchar";
    public static final String INT_TYPE            = "integer";
    public static final String SMALLINT_TYPE       = "smallint";
    public static final String BLOB_TYPE           = "blob";
    public static final String DATETIME_TYPE       = "datetime";


    public static final class ColumnInformes
    {
        public static final String ID_INFORME           =
"acceso_ubicacion";
        public static final String ACCESO_UBICACION     =
"acceso_ubicacion";
        public static final String PERIMETRO_ARQUETA    =
"perimetro_arqueta";
        public static final String PUERTA_ACCESO        =
"puerta_acceso";
        public static final String CUBIERTA             = "cubierta";
        public static final String PARAM_VERTICALES_INT =
"param_verticales_int";
        public static final String PARAM_VERTICALES_EXT =
"param_verticales_ext";
        public static final String VENTILACION_LATERAL  =
"ventilacion_lateral";
        public static final String VENTILACION_SUPERIOR =
"ventilacion_superior";
        public static final String PATES_ESCALERA       =
"pates_escalera";
        public static final String DISTANCIA_REGLA_ELEMENTOS =
"distancia_reglamentaria_elementos";
        public static final String VENTOSAS             = "ventosas";
        public static final String VALVULAS             = "valvulas";
        public static final String JUNTAS_UNION         =
"juntas_carretes";
        public static final String MANOMETROS           =
"manometros";
        public static final String CONTADORES           =
"contadores";
        public static final String FECHA                = "fecha";
        public static final String DIRECCION_ARQUETA    =
"direccion_arqueta";
        public static final String COMENTARIO           =
"comentario";
        public static final String FOTO                 = "foto";
    }

    //Script de Creación de la tabla Quotes
    public static final String CREATE_INFORME_SCRIPT =" ( " +
                    ColumnInformes.ID_INFORME+" "+INT_TYPE+"(11)
unsigned NOT NULL AUTO_INCREMENT," +
                    ColumnInformes.ACCESO_UBICACION+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.PERIMETRO_ARQUETA+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.PUERTA_ACCESO+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
```

```
                    ColumnInformes.CUBIERTA+" "+SMALLINT_TYPE+"(6)
DEFAULT NULL,"+
                    ColumnInformes.PARAM_VERTICALES_INT+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.PARAM_VERTICALES_EXT+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.VENTILACION_LATERAL+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.VENTILACION_SUPERIOR+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.PATES_ESCALERA+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.DISTANCIA_REGLA_ELEMENTOS+"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.VENTOSAS+" "+SMALLINT_TYPE+"(6)
DEFAULT NULL,"+
                    ColumnInformes.VALVULAS+" "+SMALLINT_TYPE+"(6)
DEFAULT NULL,"+
                    ColumnInformes.JUNTAS_UNION +"
"+SMALLINT_TYPE+"(6) DEFAULT NULL,"+
                    ColumnInformes.MANOMETROS+" "+SMALLINT_TYPE+"(6)
DEFAULT NULL,"+
                    ColumnInformes.CONTADORES+" "+SMALLINT_TYPE+"(6)
DEFAULT NULL,"+
                    ColumnInformes.FECHA+" "+DATETIME_TYPE+" NOT
NULL,"+
                    ColumnInformes.DIRECCION_ARQUETA+"
"+VARCHAR_TYPE+"(30) DEFAULT NULL,"+
                    ColumnInformes.COMENTARIO+" "+VARCHAR_TYPE+"(30)
DEFAULT NULL,"+
                    ColumnInformes.FOTO+" "+BLOB_TYPE+" )";

    private MySQLiteOpenHelper openHelper;
    private SQLiteDatabase database;

    public InformeSQLiteDataSource(Context context)
    {
        //getting instance of database
        openHelper  = MySQLiteOpenHelper.getInstance(context);
        database    = openHelper.getWritableDatabase();
    }

    public void insertInforme(BeanInformesDB informeDB)
    {
        ContentValues cont = new ContentValues();


cont.put(ColumnInformes.ACCESO_UBICACION,informeDB.getAccesoUbicacion(
));

cont.put(ColumnInformes.PERIMETRO_ARQUETA,informeDB.getPerimetroArquet
a());

cont.put(ColumnInformes.PUERTA_ACCESO,informeDB.getPuertaAcceso());
        cont.put(ColumnInformes.CUBIERTA,informeDB.getCubierta());

cont.put(ColumnInformes.PARAM_VERTICALES_INT,informeDB.getParVertInt()
);

cont.put(ColumnInformes.PARAM_VERTICALES_EXT,informeDB.getParVertExt()
);
```

```
cont.put(ColumnInformes.VENTILACION_LATERAL,informeDB.getVentilacionLa
teral());

cont.put(ColumnInformes.VENTILACION_SUPERIOR,informeDB.getVentilacionS
uperior());

cont.put(ColumnInformes.DISTANCIA_REGLA_ELEMENTOS,informeDB.getDistanc
iaRegElementos());
        cont.put(ColumnInformes.VENTOSAS,informeDB.getVentosas());
        cont.put(ColumnInformes.VALVULAS,informeDB.getValvulas());

cont.put(ColumnInformes.JUNTAS_UNION,informeDB.getJuntasUnion());
        cont.put(ColumnInformes.MANOMETROS,informeDB.getManometros());
        cont.put(ColumnInformes.CONTADORES,informeDB.getContadores());
        cont.put(ColumnInformes.FECHA,informeDB.getFecha());

cont.put(ColumnInformes.DIRECCION_ARQUETA,informeDB.getDireccion_arque
ta());
        cont.put(ColumnInformes.COMENTARIO,informeDB.getComentario());
        cont.put(ColumnInformes.FOTO,informeDB.getFoto());

        database.insert(INFORMES_TABLE_NAME, null, cont);
    }

    public boolean deleteInforme(int id)
    {
        boolean flag = true;
        String selection = ColumnInformes.ID_INFORME + " = ?";
        String[] selectionArgs = { String.valueOf(id) };

        int state = database.delete(INFORMES_TABLE_NAME, selection,
selectionArgs);

        if(state < 0)
            flag = false;

        return flag;
    }


    public int numbersInformesInsideDB()
    {
        int id = -1;
        Cursor c = database.rawQuery("SELECT Count(*) FROM
"+INFORMES_TABLE_NAME,null);

        if(c.moveToFirst())
            id = c.getInt(0);

        return id;
    }

    public List<BeanInformesDB> getAllInformes()
    {
        List<BeanInformesDB> informesList = new
ArrayList<BeanInformesDB>();
        BeanInformesDB beanInformesDB;

        String query = "SELECT * FROM "+INFORMES_TABLE_NAME;
```

```
        Cursor c= database.rawQuery(query, null);

        if(c.moveToFirst())
        {
            do{
                beanInformesDB = new BeanInformesDB();


beanInformesDB.setIDDataBase(c.getInt(c.getColumnIndex(ColumnInformes.
ID_INFORME)));

beanInformesDB.setDireccion_arqueta(c.getString(c.getColumnIndex(Colum
nInformes.DIRECCION_ARQUETA)));

beanInformesDB.setPerimetroArqueta(c.getInt(c.getColumnIndex(ColumnInf
ormes.PERIMETRO_ARQUETA)));

beanInformesDB.setPuertaAcceso(c.getInt(c.getColumnIndex(ColumnInforme
s.PUERTA_ACCESO)));

beanInformesDB.setCubierta(c.getInt(c.getColumnIndex(ColumnInformes.CU
BIERTA)));

beanInformesDB.setParVertExt(c.getInt(c.getColumnIndex(ColumnInformes.
PARAM_VERTICALES_EXT)));

beanInformesDB.setParVertInt(c.getInt(c.getColumnIndex(ColumnInformes.
PARAM_VERTICALES_INT)));

beanInformesDB.setVentilacionLateral(c.getInt(c.getColumnIndex(ColumnI
nformes.VENTILACION_LATERAL)));

beanInformesDB.setVentilacionSuperior(c.getInt(c.getColumnIndex(Column
Informes.VENTILACION_SUPERIOR)));

beanInformesDB.setPatesEscalera(c.getInt(c.getColumnIndex(ColumnInform
es.PATES_ESCALERA)));

beanInformesDB.setDistanciaRegElementos(c.getInt(c.getColumnIndex(Colu
mnInformes.DISTANCIA_REGLA_ELEMENTOS)));


beanInformesDB.setVentosas(c.getInt(c.getColumnIndex(ColumnInformes.VE
NTOSAS)));

beanInformesDB.setValvulas(c.getInt(c.getColumnIndex(ColumnInformes.JU
NTAS_UNION)));

beanInformesDB.setManometros(c.getInt(c.getColumnIndex(ColumnInformes.
MANOMETROS)));

beanInformesDB.setContadores(c.getInt(c.getColumnIndex(ColumnInformes.
CONTADORES)));


beanInformesDB.setComentario(c.getString(c.getColumnIndex(ColumnInform
es.COMENTARIO)));

beanInformesDB.setFoto(c.getBlob(c.getColumnIndex(ColumnInformes.FOTO)
));
```

```
                informesList.add(beanInformesDB);

            }while(c.moveToNext());

            return informesList;
        }
        return informesList;
    }
}
```

- **Para MySQLiteHelper:**

```
package es.Jose.arquetanatureble.DB_SQLITE;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import es.Jose.arquetanatureble.UTIL.Constant;

public class MySQLiteOpenHelper extends SQLiteOpenHelper
{
    private static String DATABASE_NAME    = "informeBackup.db";
    private static int    DATABASE_VERSION = 1;

    private static MySQLiteOpenHelper myInstance;
    private static Context mContext;

    public MySQLiteOpenHelper(Context context)
    {
        super(context,DATABASE_NAME,null,DATABASE_VERSION);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        //Create table informes
        Log.d(Constant.TAG,"Query
:"+InformeSQLiteDataSource.CREATE_INFORME_SCRIPT);
        db.execSQL(InformeSQLiteDataSource.CREATE_INFORME_SCRIPT);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion)
    {
        db.execSQL("DROP TABLE " +
InformeSQLiteDataSource.INFORMES_TABLE_NAME);
        db.execSQL(InformeSQLiteDataSource.CREATE_INFORME_SCRIPT);
    }

    ////////////////////////////////////////////
    //////////////singleton/////////////////////
    ////////////////////////////////////////////
    public static MySQLiteOpenHelper getInstance(Context context){
        if (myInstance == null)
            myInstance = new
MySQLiteOpenHelper(context.getApplicationContext());
```

```
        return myInstance;
    }
    ////////////////////////////////////////////////
}
```

- **Para DeviceActivity:**

```
package es.Jose.arquetanatureble.GUI;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.google.gson.Gson;

import org.json.JSONException;
import org.json.JSONObject;

import android.os.Handler;

import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.BEAN.BeanInformesDB;
import es.Jose.arquetanatureble.BLE.HandlerBLE;
import es.Jose.arquetanatureble.BLE.ServiceData;
import es.Jose.arquetanatureble.CORE.BLE_Application;


import es.Jose.arquetanatureble.R;
import es.Jose.arquetanatureble.UTIL.Constant;
import es.Jose.arquetanatureble.WEBSERVICE.BeanArqueta;
import es.Jose.arquetanatureble.WEBSERVICE.VolleySingleton;
import es.Jose.arquetanatureble.WEBSERVICE.WebServiceConstant;


public class DeviceActivity extends Activity implements
View.OnClickListener
{
    private static Button bt_resumen;
    private static Button bt_conect2serverBLE;
    private static HandlerBLE mHandlerBLE;
    private static BeanInformesDB beanInformesDB;
    private static BeanBluetoothDevice beanBluetoothDevice;
    private static BroadcasterDataSensor broadcasterDataSensor;
    private static Handler mHandler;

    private static TextView flowmeter;
    private static TextView valve;
```

```
    private static TextView pressure;

    private static Gson gson = new Gson();
    private static JsonObjectRequest jsonObjectRequest;

    private static final int DELAY_ADQUISITION = 1;


//#####################################################################
    /****************** metodos del flujo Android.
*********************/

//#####################################################################
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensor);

        //get information about informes
        beanInformesDB      =
getIntent().getParcelableExtra(Constant.EXTRA_INFORMESDB);
        beanBluetoothDevice =
getIntent().getParcelableExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE);

        //handler BLE
        mHandlerBLE = ((BLE_Application)
getApplication()).getmHandlerBLEInstance(this);
        ((BLE_Application) getApplication()).resetHandlerBLE();

        bt_conect2serverBLE = (Button)
findViewById(R.id.bt_enviarInforme);
        bt_resumen          = (Button)
findViewById(R.id.bt_connectServer);

        bt_conect2serverBLE.setOnClickListener(this);
        bt_resumen.setOnClickListener(this);

        pressure  = (TextView) findViewById(R.id.tV_sensor1);
        flowmeter = (TextView) findViewById(R.id.tV_sensor2);
        valve     = (TextView)findViewById(R.id.tV_sensor3);

        //get data from WebService
        loadFromWebService(1);

        broadcasterDataSensor = new BroadcasterDataSensor();
        mHandler = new Handler();
    }

    @Override
    protected void onResume()
    {
        super.onResume();

        IntentFilter i = new
IntentFilter(BroadcasterDataSensor.DATA_FLOW_ACT);
        registerReceiver(broadcasterDataSensor, i);

        i = new IntentFilter(BroadcasterDataSensor.DATA_PRESSU_ACT);
        registerReceiver(broadcasterDataSensor, i);

        i = new IntentFilter(BroadcasterDataSensor.DATA_VALVE_ACT);
```

```
        registerReceiver(broadcasterDataSensor,i);

        i = new IntentFilter(BroadcasterDataSensor.DATA_UNKNOWN_ACT);
        registerReceiver(broadcasterDataSensor,i);
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(broadcasterDataSensor);
    }

    @Override
    protected void onStop()
    {
        super.onStop();
        mHandler.removeCallbacks(adquisitionData);
    }

//#####################################################################
    /****************** metodos implements Android.
*********************/

//#####################################################################
    //here events or action from usar are taken.
    @Override
    public void onClick(View v)
    {
        Intent intentActivity;
        switch (v.getId())
        {
            case R.id.bt_enviarInforme:
                //falta enviar datos.
                intentActivity = new Intent(this,
ResumenActivity.class);
                startActivityForResult(intentActivity, 0);
                break;
            case R.id.bt_connectServer:
                Button bt_conexion = (Button)v;
                if(bt_conexion.getText().equals("Conectar"))
                {
                    String deviceAddress =
beanBluetoothDevice.getBluetoothDevice().getAddress();
                    mHandlerBLE.connect(this,
mHandlerBLE.getMyCallBack(), deviceAddress);

                    Toast.makeText(this,"Conectando con el
dispositivo.",Toast.LENGTH_LONG).show();

                    bt_conexion.setText("Desconectar");

                    //automatically adquisition data after
DELAY_ADQUISITION seconds

mHandler.postDelayed(adquisitionData,DELAY_ADQUISITION * 1000);
                }
                else
                {

                        mHandlerBLE.disconnect();
                        mHandler.removeCallbacks(adquisitionData);
```

```
                        Toast.makeText(this, "Desconectando con el
dispositivo.", Toast.LENGTH_LONG).show();
                        bt_conexion.setText("Conectar");

                }

                break;
            default:
                Log.i(Constant.TAG,"Error not ID button found it.");
                break;
        }
    }

//#########################################################################
    /** method to activity **/

//#########################################################################
    private void updateFlowmeter(byte [] value)
    {
        int [] data = ServiceData.workFlometerData(value);
        if(Constant.DEBUG)
            Log.i(Constant.TAG,"DeviceActivity -- updateFlowmeter --
updating value = "+ new String(String.valueOf(data)));
        //read data from sensor.
        //adapter
        flowmeter.setText(new String(String.valueOf(data)));
    }

    private void updatePressure(byte [] value)
    {
        int [] data = ServiceData.workPressureData(value);
        if(Constant.DEBUG)
            Log.i(Constant.TAG,"DeviceActivity -- updatePressure --
updating value = "+ new String(String.valueOf(data)));
        //read data from sensor.
        //adapter
        pressure.setText(new String(String.valueOf(data)));
    }

    private void updateValve(byte [] value)
    {
        boolean flag = ServiceData.workValveData(value);
        if(Constant.DEBUG)
            Log.i(Constant.TAG,"DeviceActivity -- updateValve --
updating value = "+ flag);

        valve.setText(flag ? "On" : "Off");

    }

    //Handle automatic
    private Runnable adquisitionData = new Runnable() {
        @Override
        public void run() {

            if (Constant.DEBUG)
                Log.i(Constant.TAG, "Stop scanning");

            mHandler.postDelayed(adquisitionData,DELAY_ADQUISITION *
1000);
        }
```

```
    };

    /**
     * Carga el adaptador con las metas obtenidas
     * en la respuesta
     */
    public void loadFromWebService(int id)
    {
        final String newURL = WebServiceConstant.GET_ARQUETA_BYID +
"?idArqueta = "+id ;
        jsonObjectRequest = new JsonObjectRequest(
                Request.Method.GET,
                newURL,
                new Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject jsonObject) {
                        // Procesar la respuesta Json
                        procesarRespuesta(jsonObject);
                        Log.d(Constant.TAG, "Procesando respuesta");
                        Log.d(Constant.TAG, "URL : "+newURL);
                    }
                },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError
volleyError) {
                        Log.d(Constant.TAG, "Error Volley: " +
volleyError.getMessage());
                    }
                });

        // Petición GET

VolleySingleton.getInstace(this).addToRequestQueue(jsonObjectRequest);
    }

    /**
     * Interpreta los resultados de la respuesta y así
     * realizar las operaciones correspondientes
     *
     * @param response Objeto Json con la respuesta
     */
    private void procesarRespuesta(JSONObject response) {
        try {
            // Obtener atributo "estado"
            String estado = response.getString("estado");

            switch (estado) {
                case "1": // EXITO
                    Log.d(Constant.TAG, "respuesta exitosa.");
                    // Obtener array "metas" Json
                    JSONObject mensaje =
response.getJSONObject("arqueta");
                    // Parsear con Gson
                    Log.d(Constant.TAG, "ParsearGson mensaje:
"+mensaje.toString());
                    gson = new Gson();
                    BeanArqueta arqueta =
gson.fromJson(mensaje.toString(), BeanArqueta.class);
                    Log.d(Constant.TAG, "arqueta:
"+arqueta.toString());
```

```
                    //presentar en activity
                    updateDatosActivity(arqueta);
                    break;
                case "2": // FALLIDO
                    Log.d(Constant.TAG, "respuesta fallida.");
                    String mensaje2 = response.getString("mensaje");
                    Toast.makeText(
                            this,
                            mensaje2,
                            Toast.LENGTH_LONG).show();
                    break;
                default:
                    Log.d(Constant.TAG, "respuesta error
estado:"+estado);
                    break;
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    private void updateDatosActivity(BeanArqueta arquetas)
    {

        /*
        tvId.setText(arquetas.getId());
        tvFechaAlta.setText(arquetas.getInsert_time());
        tvNombre.setText(arquetas.getNombre_arqueta());
        tvDireccion.setText(arquetas.getDireccion_arqueta());
        tvUuid1.setText(arquetas.getUuid_sensor1());
        tvUuid2.setText(arquetas.getUuid_sensor2());
        tvUuid3.setText(arquetas.getUuid_sensor3());*/

    }


//#####################################################################
    /** Class to update value of several sensor in UI **/

//#####################################################################
    public class BroadcasterDataSensor extends BroadcastReceiver
    {
        public static final String DATA_FLOW_ACT     =
"DATA_FLOW_ACT";
        public static final String DATA_PRESSU_ACT   =
"DATA_PRESSU_ACT";
        public static final String DATA_VALVE_ACT    =
"DATA_VALVE_ACT";
        public static final String DATA_UNKNOWN_ACT  =
"DATA_UNKNOWN_ACT";

        @Override
        public void onReceive(Context context, Intent intent)
        {
            String action = intent.getAction();
            byte[] value;

            if(Constant.DEBUG)
                Log.i(Constant.TAG,"BroadcasterDataSensor -- onReceive
-- action :" +action);
```

```
            if(action.equals(DATA_FLOW_ACT))
            {
                value  = intent.getByteArrayExtra(Constant.MSG_FLOW);
                updateFlowmeter(value);
            }
            else if(action.equals(DATA_PRESSU_ACT))
            {
                value  =
intent.getByteArrayExtra(Constant.MSG_PRESSU);
                updatePressure(value);
            }
            else if(action.equals(DATA_VALVE_ACT))
            {
                value  = intent.getByteArrayExtra(Constant.MSG_VALVE);
                updateValve(value);
            }
            else //message unknown
            {
                if(Constant.DEBUG)
                    Log.i(Constant.TAG,"BroadcasterDataSensor -- Error
msg unknown.");
            }
        }
    }
}
```

- **Para InsideChamberActivity:**

```
package es.Jose.arquetanatureble.GUI;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.provider.MediaStore;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;

import java.io.ByteArrayOutputStream;

import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.BEAN.BeanInformesDB;
import es.Jose.arquetanatureble.R;
import es.Jose.arquetanatureble.UTIL.Constant;

public class InsideChamberActivity extends Activity implements
View.OnClickListener {


    private static final int NUMBER_PARAMS = 5;
    private static BeanInformesDB beanInformesDB;
    private static BeanBluetoothDevice beanBluetoothDevice;
```

```
    private static Bitmap bmp;
    private static ImageView img;
    private static RadioGroup RG_params[];
    private static TextView tV_comment;

    public static Button bt_nextDevice;
    public static Button bt_photo;



    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inside_chamber);

        //get information about informes
        beanInformesDB      =
getIntent().getParcelableExtra(Constant.EXTRA_INFORMESDB);
        beanBluetoothDevice =
getIntent().getParcelableExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE);

        RG_params       = new RadioGroup[NUMBER_PARAMS];
        RG_params[0]    = (RadioGroup)findViewById(R.id.GrbGrupoEI1);
        RG_params[1]    = (RadioGroup)findViewById(R.id.GrbGrupoEI2);
        RG_params[2]    = (RadioGroup)findViewById(R.id.GrbGrupoEI3);
        RG_params[3]    = (RadioGroup)findViewById(R.id.GrbGrupoEI4);
        RG_params[4]    = (RadioGroup)findViewById(R.id.GrbGrupoEI5);

        img             = (ImageView) findViewById(R.id.IV_camera);

        tV_comment      = (TextView) findViewById(R.id.tB_comentario);

        bt_nextDevice  = (Button)
findViewById(R.id.bt_siguienteDevice);
        bt_photo        = (Button) findViewById(R.id.bt_camera);

        bt_nextDevice.setOnClickListener(this);
        bt_photo.setOnClickListener(this);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_inside_chamber, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
```

```
        return super.onOptionsItemSelected(item);
    }

    //func to control diferents button actions
    @Override
    public void onClick(View view)
    {
        Intent intentActivity;
        switch (view.getId())
        {
            case R.id.bt_siguienteDevice:
                //read whole parameters selected.
                boolean nextActivity = true;
                int count;
                int IDs[];
                int j = 0;
                int result[];
                int resultSTR;
                for(int i = 0; i < RG_params.length-1 ;i++)
                {
                    count = RG_params[i].getChildCount();
                    IDs = new int[count];

                    j = 0;
                    while(j <= count -1)
                    {
                        IDs[j] = RG_params[i].getChildAt(j).getId();
                        j++;
                    }

                    result = ChekingRadioButton(RG_params[i], IDs[0],
IDs[1], IDs[2]);
                    if(result[0] == 0)
                    {
                        nextActivity = false;
                        Toast.makeText(this, "No están todos los check
seleccionados", Toast.LENGTH_LONG).show();
                        break;
                    }

                    switch (result[1])
                    {
                        case 1:
                            resultSTR = beanInformesDB.BUENO;
                            break;
                        case 2:
                            resultSTR = beanInformesDB.ACEPTABLE;
                            break;
                        case 3:
                            resultSTR =
beanInformesDB.NECESITA_REPARACION;
                            break;
                        default:
                            resultSTR = 0;
                            break;
                    }

                    beanInformesDB.gnrlSetEi(resultSTR,i);

                }if(nextActivity)
```

```
                {
                    //add comment and photo.
                    if(bmp != null)
                    {
                        ByteArrayOutputStream stream = new
ByteArrayOutputStream();
                        bmp.compress(Bitmap.CompressFormat.JPEG, 100,
stream);
                        byte[] photoData = stream.toByteArray();

                        beanInformesDB.setFoto(photoData);
                    }

beanInformesDB.setComentario(tV_comment.getText().toString());

                    intentActivity= new Intent(this,
DeviceActivity.class);
                    intentActivity.putExtra(Constant.EXTRA_INFORMESDB,
beanInformesDB);

intentActivity.putExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE,beanBlueto
othDevice);
                    this.startActivity(intentActivity);
                }
                break;

            case R.id.bt_camera:
                intentActivity = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(intentActivity, 0);
                break;

            default:
                Log.i(Constant.TAG, "InsideChamberActivity -- Error
not button found.");
        }
    }

    //checker radioButton active
    private int[] ChekingRadioButton(RadioGroup radioGroup, int
IDButton1, int IDButton2, int IDButton3)
    {
        int out[] = new int[2];
        out[0] = 0;
        out[1] = 0;
        int radioButtonSelected  =
radioGroup.getCheckedRadioButtonId();

        if(IDButton1 == radioButtonSelected)
        {
            out[0] = 1;
            out[1] = 1;//bueno
        }
        else if(IDButton2 == radioButtonSelected)
        {
            out[0] = 1;
            out[1] = 2;//aceptable
        }
        else if(IDButton3 == radioButtonSelected)
        {
            out[0] = 1;
```

```
            out[1] = 3;//necesita reparacion
        }

        return out;
    }


// this method gives us the result of picture taken with tha camera.
    @Override
    protected void onActivityResult(int requestCode,int
resultCode,Intent data)
    {
        Log.i(Constant.TAG, "InsideChamberActivity --
onActivityResult: taking the image wich has been taking with the
camera");
        super.onActivityResult(requestCode, resultCode, data);

        if(resultCode == Activity.RESULT_OK)
        {
            Bundle ext = data.getExtras();
            bmp        = (Bitmap)ext.get("data");
            img.setImageBitmap(bmp);
        }
        else
        {
            Log.i(Constant.TAG,"InsideChamberActivity --
onActivityResult: It couldn't possible take the image");
        }
    }
}
```

- **Para OutsideChamberActivity:**

```
package es.Jose.arquetanatureble.GUI;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.BEAN.BeanInformesDB;
import es.Jose.arquetanatureble.R;
import es.Jose.arquetanatureble.UTIL.Constant;

public class OutsideChamberActivity extends Activity implements
View.OnClickListener {

    private static final int NUMBER_PARAMS = 10; //number of
parameters

    private static Button bt_next;
    private static RadioGroup RG_params[];
```

```
    private static BeanBluetoothDevice beanBluetoothDevice;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_outside_chamber);

        bt_next = (Button) findViewById(R.id.bt_inside);
        bt_next.setOnClickListener(this);

        RG_params       = new RadioGroup[NUMBER_PARAMS];

        //get data from ScanActivity
        beanBluetoothDevice =
getIntent().getParcelableExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE);

        RG_params[0]    = (RadioGroup)findViewById(R.id.GrbGrupo1);
        RG_params[1]    = (RadioGroup)findViewById(R.id.GrbGrupo2);
        RG_params[2]    = (RadioGroup)findViewById(R.id.GrbGrupo3);
        RG_params[3]    = (RadioGroup)findViewById(R.id.GrbGrupo4);
        RG_params[4]    = (RadioGroup)findViewById(R.id.GrbGrupo5);
        RG_params[5]    = (RadioGroup)findViewById(R.id.GrbGrupo6);
        RG_params[6]    = (RadioGroup)findViewById(R.id.GrbGrupo7);
        RG_params[7]    = (RadioGroup)findViewById(R.id.GrbGrupo8);
        RG_params[8]    = (RadioGroup)findViewById(R.id.GrbGrupo9);
        RG_params[9]    = (RadioGroup)findViewById(R.id.GrbGrupo10);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_outside_chamber, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings)
        {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    //function to control e button
    @Override
    public void onClick(View view)
    {
        switch (view.getId())
        {
```

```
            case R.id.bt_inside:
                boolean nextActivity = true;
                BeanInformesDB beanInformesDB = new BeanInformesDB();

                //read whole parameters selected.
                int count;
                int IDs[];
                int j = 0;
                int result[];
                int resultSTR;
                for(int i = 0; i < RG_params.length-1 ;i++)
                {
                    j = 0;
                    count = RG_params[i].getChildCount();
                    IDs = new int[count];

                    while(j <= count-1)
                    {
                        IDs[j] = RG_params[i].getChildAt(j).getId();
                        j++;
                    }

                    result = ChekingRadioButton(RG_params[i], IDs[0],
IDs[1], IDs[2]);
                    if(result[0] == 0)
                    {
                        nextActivity = false;
                        Toast.makeText(this,"No están todos los check
seleccionados",Toast.LENGTH_LONG).show();
                        break;
                    }

                    switch (result[1])
                    {
                        case 1:
                            resultSTR = beanInformesDB.BUENO;
                            break;
                        case 2:
                            resultSTR = beanInformesDB.ACEPTABLE;
                            break;
                        case 3:
                            resultSTR =
beanInformesDB.NECESITA_REPARACION;
                            break;
                        default:
                            resultSTR = 0;
                            break;
                    }

                    beanInformesDB.gnrlSetExt(resultSTR, i);
                }
                if(nextActivity)
                {
                    Intent intentActivity = new Intent(this,
InsideChamberActivity.class);
                    intentActivity.putExtra(Constant.EXTRA_INFORMESDB,
beanInformesDB);

intentActivity.putExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE,beanBlueto
othDevice);
                    this.startActivity(intentActivity);
```

```
                }
                break;
            default:
                Log.i(Constant.TAG, "OutsideChamberActivity -- Error
not button found.");
        }

    }

    //checker radioButton active
    private int[] ChekingRadioButton(RadioGroup radioGroup, int
IDButton1, int IDButton2, int IDButton3)
    {
        int out[] = new int[2];
        out[0] = 0;
        out[1] = 0;
        int radioButtonSelected  =
radioGroup.getCheckedRadioButtonId();

        if(IDButton1 == radioButtonSelected)
        {
            out[0] = 1;
            out[1] = 1;//bueno
        }
        else if(IDButton2 == radioButtonSelected)
        {
            out[0] = 1;
            out[1] = 2;//aceptable
        }
        else if(IDButton3 == radioButtonSelected)
        {
            out[0] = 1;
            out[1] = 3;//necesita reparacion
        }

        return out;
    }
}
```

- **Para ResumenActivity:**

```
package es.Jose.arquetanatureble.GUI;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.google.gson.Gson;

import org.json.JSONException;
import org.json.JSONObject;
```

```
import java.util.HashMap;
import java.util.Map;

import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.BEAN.BeanInformesDB;
import es.Jose.arquetanatureble.DB_SQLITE.InformeSQLiteDataSource;
import es.Jose.arquetanatureble.R;
import es.Jose.arquetanatureble.UTIL.Constant;
import es.Jose.arquetanatureble.WEBSERVICE.WebServiceConstant;
import es.Jose.arquetanatureble.WEBSERVICE.VolleySingleton;


public class ResumenActivity extends Activity {

    private static BeanInformesDB beanInformesDB;
    private static BeanBluetoothDevice beanBluetoothDevice;

    private static Context mContex;
    private static InformeSQLiteDataSource mInformeBD;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_resumen);

        mContex = this;

        //get information about informes
        beanInformesDB      =
getIntent().getParcelableExtra(Constant.EXTRA_INFORMESDB);
        beanBluetoothDevice =
getIntent().getParcelableExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE);

        //Setting the data in text to check everything is ok.


        //Sending the data to the webservice.
        saveInfoOnCloud();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_resumen, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
```

```
        return super.onOptionsItemSelected(item);
    }


    private void proccessReplyFromWebService(JSONObject response) {
        try {
            // Obtener atributo "estado"
            String estado = response.getString("estado");

            switch (estado) {
                case "1": // EXITO
                    Log.d(Constant.TAG, "respuesta exitosa.");
                    Toast.makeText(this, "Insercion correcta",
Toast.LENGTH_LONG).show();
                    break;
                case "2": // FALLIDO
                    Log.d(Constant.TAG, "respuesta fallida.");
                    String mensaje2 = response.getString("mensaje");
                    Toast.makeText(this, mensaje2,
Toast.LENGTH_LONG).show();
                    break;
                default:
                    Log.d(Constant.TAG, "respuesta error
estado:"+estado);
                    break;
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }


    public void saveInfoOnCloud() {
        // Obtener valores actuales de los controles

        Log.d(Constant.TAG, WebServiceConstant.INSERT);

        String newURL = WebServiceConstant.INSERT
                +"?acceso_ubicacion="+
beanInformesDB.getAccesoUbicacion()
                +"&perimetro_arqueta="+
beanInformesDB.getPerimetroArqueta()
                +"&puerta_acceso="+ beanInformesDB.getPuertaAcceso()
                +"&cubierta="+ beanInformesDB.getCubierta()
                +"&param_verticales_int="+
beanInformesDB.getParVertInt()
                +"&param_verticales_ext="+
beanInformesDB.getParVertExt()
                +"&ventilacion_lateral="+
beanInformesDB.getVentilacionLateral()
                +"&ventilacion_superior="+
beanInformesDB.getVentilacionSuperior()
                +"&pates_escalera="+ beanInformesDB.getPatesEscalera()

+"&distancia_reglamentaria_elementos="+beanInformesDB.getDistanciaRegE
lementos()
                +"&ventosas="+ beanInformesDB.getVentosas()
                +"&juntas_carretes="+ beanInformesDB.getJuntasUnion()
                +"&manometros="+ beanInformesDB.getManometros()
                +"&contadores="+ beanInformesDB.getContadores()
```

```
                +"&fecha="+ beanInformesDB.getFecha()
+"&direccion_arqueta="+beanInformesDB.getDireccion_arqueta()
                +"&comentario="+ beanInformesDB.getComentario()
                +"&foto="+ beanInformesDB.getFoto();

        Log.d(Constant.TAG, newURL);

        // Actualizar datos en el servidor
        VolleySingleton.getInstace(this).addToRequestQueue(
                new JsonObjectRequest(
                        Request.Method.GET,
                        newURL,
                        new Response.Listener<JSONObject>() {
                            @Override
                            public void onResponse(JSONObject
response) {
                                // Procesar la respuesta del servidor
                                proccessReplyFromWebService(response);
                            }
                        },
                        new Response.ErrorListener() {
                            @Override
                            public void onErrorResponse(VolleyError
error) {
                                Log.d(Constant.TAG, "Error Volley
(response insert): " + error.getMessage());
                                Toast.makeText(mContex, "No se
comunicó con el Cloud.", Toast.LENGTH_LONG).show();
                                //aqui guardamos en la SQLite
                                mInformeBD = new
InformeSQLiteDataSource(getApplicationContext());

mInformeBD.insertInforme(beanInformesDB);
                                Toast.makeText(mContex, "Guardado
informe para posterior envio.", Toast.LENGTH_LONG).show();
                            }
                        }

                ) {
                    @Override
                    public Map<String, String> getHeaders() {
                        Map<String, String> headers = new
HashMap<String, String>();
                        headers.put("Content-Type", "application/json;
charset=utf-8");
                        headers.put("Accept", "application/json");
                        return headers;
                    }

                    @Override
                    public String getBodyContentType() {
                        return "application/json; charset=utf-8" +
getParamsEncoding();
                    }
                }
        );

    }
}
```

- **Para ScanActivity:**

```
package es.Jose.arquetanatureble.GUI;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.Toast;
import android.os.Handler;

import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;

import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.BEAN.BeanInformesDB;
import es.Jose.arquetanatureble.BLE.BLEBroadcastReceiver;
import es.Jose.arquetanatureble.BLE.HandlerBLE;

import es.Jose.arquetanatureble.CORE.BLE_Application;
import es.Jose.arquetanatureble.CORE.MyNotificationHandler;
import es.Jose.arquetanatureble.CORE.ServiceDetectionTag;
import es.Jose.arquetanatureble.DB_SQLITE.InformeSQLiteDataSource;
import es.Jose.arquetanatureble.R;
import es.Jose.arquetanatureble.UTIL.Constant;
import es.Jose.arquetanatureble.WEBSERVICE.BeanInforme;
import es.Jose.arquetanatureble.WEBSERVICE.VolleySingleton;
import es.Jose.arquetanatureble.WEBSERVICE.WebServiceConstant;
```

```
public class ScanActivity extends Activity implements
OnItemClickListener { //}, LeScanCallback {

    private static final int SCAN_ITEM = 1;
    private static final int UPDATE_INFO_INTO_CLOUD = 2;
    private static ListView mListView;
    private static Context mContext;
    private static HandlerBLE mHandlerBLE;
    private Handler mHandler;
    private static ScanArrayAdapter mAdapter;
    private static List<BeanBluetoothDevice> mDeviceList;
    private Menu mMenu;
    private static Activity mActivity;
    private static InformeSQLiteDataSource informesSQLite;
    private BLEBroadcastReceiver mScanBroadcastReceiver;
    public static boolean isOncreate = false;


//#####################################################################
    /****************** metodos del flujo Android.
*********************/

//#####################################################################
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_scan);

        isOncreate = true;

        mActivity    = this;
        mContext     = this;
        mHandler     = new Handler() ;
        mDeviceList = new ArrayList<BeanBluetoothDevice>();
        mAdapter     = new ScanArrayAdapter(this, mDeviceList);

        //manejador BLE
        mHandlerBLE = ((BLE_Application)
getApplication()).getmHandlerBLEInstance(this);
        ((BLE_Application) getApplication()).resetHandlerBLE();

        mScanBroadcastReceiver = new BLEBroadcastReceiver(this,
mAdapter);

        /*IntentFilter i = new
IntentFilter(HandlerBLE.ACTION_BLE_FOUND);
        registerReceiver(mScanBroadcastReceiver, i);*/

        //mListView = getListView();
        mListView = (ListView) findViewById(R.id.listView);
        mListView.setVisibility(View.VISIBLE);

        mListView.setAdapter(mAdapter);
        mListView.setOnItemClickListener(this);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        super.onCreateOptionsMenu(menu);
```

```
        mMenu = menu;

        String menuTitle = getResources().getString(R.string.scan);
        menu.add(0,SCAN_ITEM,0,menuTitle);

        menuTitle = getResources().getString(R.string.sendInfo2Cloud);
        menu.add(0,UPDATE_INFO_INTO_CLOUD,0,menuTitle);

       /*
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_scan, menu);*/
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId()){
            case SCAN_ITEM:
                scan();
                Toast.makeText(this, "Escaneando arquetas.",
Toast.LENGTH_LONG).show();
                break;
            case UPDATE_INFO_INTO_CLOUD:
                //updateInfoCloud();
                break;
        }
        return true;
        /*
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);*/
    }


    @Override
    protected void onResume()
    {
        super.onResume();

        //restart broadcaster
        mScanBroadcastReceiver = new BLEBroadcastReceiver(this,
mAdapter);
        IntentFilter i = new
IntentFilter(HandlerBLE.ACTION_BLE_FOUND);
        registerReceiver(mScanBroadcastReceiver,i);

        if
(!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOT
H_LE))
        {
```

```
            Toast.makeText(this, "Tecnología BLE no está soportado por
este dispositivo.", Toast.LENGTH_SHORT).show();
            finish();
        }

        //checking if bluetooth is enable.
        if(!mHandlerBLE.IsEnabledBlue())
        {
            Intent enableBlue = new
Intent(mHandlerBLE.getBlueAdapter().ACTION_REQUEST_ENABLE);
            startActivity(enableBlue);
            finish();
            return;
        }

        Log.d(Constant.TAG,"#>> onResume : Stopping service.");
        //stop service
        Intent service = new Intent(this, ServiceDetectionTag.class);
        stopService(service);

        mAdapter.clear();
        HandlerBLE.setup();
    }

    @SuppressLint("MissingSuperCall")
    @Override
    protected void onPause() {
        super.onPause();

        //Make sure that there is no pending Callback
        mHandler.removeCallbacks(mStopRunnable);

        if(mScanBroadcastReceiver != null)
            unregisterReceiver(mScanBroadcastReceiver);

        mAdapter.clear();
        mAdapter.notifyDataSetChanged();

        isOncreate = false;

        if (mHandlerBLE.isScanning) {
            mHandlerBLE.stopLeScan();
        }
    }

    @Override
    protected void onDestroy() {

        //stop service
        Intent service = new Intent(this, ServiceDetectionTag.class);
        stopService(service);

        super.onDestroy();
    }
    @Override
    protected void onStop()
    {
        super.onStop();

        Log.d(Constant.TAG, "#>> onStop : Running service.");
```

```
        //Run service
        Intent service = new Intent(this, ServiceDetectionTag.class);
        startService(service);

    }


//#####################################################################
    /****************** metodos manejo Tag
*********************/

//#####################################################################
    @Override
    //recogemos los metodos del tag seleccionado y recogemos los
datos.
    public void onItemClick(AdapterView<?> parent, View view, int
position, long id)
    {
        //stop service
        Intent service = new Intent(this, ServiceDetectionTag.class);
        stopService(service);

        if (Constant.DEBUG)
            Log.i(Constant.TAG, "Selected device " +
mDeviceList.get(position).getBluetoothDevice().getAddress());

        if (mHandlerBLE.isScanning)
        { //stop scanning
            configureScan(false);
            mHandlerBLE.stopLeScan();
            if (Constant.DEBUG)
                Log.i(Constant.TAG, "Stop scanning");
        }

        String address  =
mDeviceList.get(position).getBluetoothDevice().getAddress();
        String name     =
mDeviceList.get(position).getBluetoothDevice().getName();

        if (name==null)
            name="unknown";

        BeanBluetoothDevice beanBlue = (BeanBluetoothDevice)
mAdapter.getItem(position);

        Intent intentActivity= new Intent(this,
OutsideChamberActivity.class);

intentActivity.putExtra(Constant.EXTRA_BEAN_BLUETOOTHDEVICE,beanBlue);
        //intentActivity.putExtra(Constant.EXTRA_ADDRESS, address);
        //intentActivity.putExtra(Constant.EXTRA_NAME, name);
        this.startActivity(intentActivity);

    }

    //Handle automatic stop of LEScan
    private Runnable mStopRunnable = new Runnable() {
        @Override
        public void run() {
            configureScan(false);
            if (Constant.DEBUG)
```

```
                Log.i(Constant.TAG, "Stop scanning");

            }
        };

        public void configureScan(boolean flag)
        {
            String itemText = null;

            if (!flag)
            {
                itemText = getResources().getString(R.string.stopScan);
                mHandlerBLE.stopLeScan();

                if (Constant.DEBUG)
                    Log.i(Constant.TAG, "ScanActivity -- StopScan");

            }
            else
            {
                itemText= getResources().getString(R.string.scan);
                mHandlerBLE.startLeScan();

                if (Constant.DEBUG)
                    Log.i(Constant.TAG, "ScanActivity -- StartScan");
            }

            mMenu.findItem(SCAN_ITEM).setTitle(itemText);

            if (Constant.DEBUG)
                Log.i(Constant.TAG, "Updated Menu Item. New value: " +
itemText);
        }

        // Metodo para iniciar el scaneo cuando te llaman manualmente.
        private void scan() {
            Boolean flag =
mMenu.findItem(SCAN_ITEM).getTitle().equals(R.string.stopScan)?true:fa
lse;
            if (flag) { //stop scanning

                configureScan(false);

                if (Constant.DEBUG)
                    Log.i(Constant.TAG, "Stop scanning");

            } else {
                mAdapter.clear();
                mAdapter.notifyDataSetChanged();
                configureScan(true);

                if (Constant.DEBUG)
                    Log.i(Constant.TAG, "Start scanning for BLE
devices...");

                //automatically stop LE scan after 5 seconds
                mHandler.postDelayed(mStopRunnable, 30000);
            }
        }

        private void updateInfoCloud()
```

```
    {

        try
        {
            informesSQLite = new
InformeSQLiteDataSource(getApplicationContext());
        }
        catch(Exception e )
        {
            Toast.makeText(this, "Error enviando informes."+
e.getMessage()+e.getStackTrace(), Toast.LENGTH_LONG).show();
        }

        List<BeanInformesDB> informes2Upload =
informesSQLite.getAllInformes();
        if(informes2Upload.isEmpty())
        {
            Toast.makeText(this, "No hay ningún informe para enviar.",
Toast.LENGTH_LONG).show();
            return;
        }

        String newURL;

        for (final BeanInformesDB bean:informes2Upload )
        {
            newURL = "";
            newURL = WebServiceConstant.INSERT
                    +"?acceso_ubicacion="+ bean.getAccesoUbicacion()
                    +"&perimetro_arqueta="+ bean.getPerimetroArqueta()
                    +"&puerta_acceso="+ bean.getPuertaAcceso()
                    +"&cubierta="+ bean.getCubierta()
                    +"&param_verticales_int="+ bean.getParVertInt()
                    +"&param_verticales_ext="+ bean.getParVertExt()
                    +"&ventilacion_lateral="+
bean.getVentilacionLateral()
                    +"&ventilacion_superior="+
bean.getVentilacionSuperior()
                    +"&pates_escalera="+ bean.getPatesEscalera()

+"&distancia_reglamentaria_elementos="+bean.getDistanciaRegElementos()
                    +"&ventosas="+ bean.getVentosas()
                    +"&juntas_carretes="+ bean.getJuntasUnion()
                    +"&manometros="+ bean.getManometros()
                    +"&contadores="+ bean.getContadores()
                    +"&fecha="+ bean.getFecha()
                    +"&direccion_arqueta="+bean.getDireccion_arqueta()
                    +"&comentario="+ bean.getComentario()
                    +"&foto="+ bean.getFoto();

            // Actualizar datos en el servidor
            VolleySingleton.getInstace(this).addToRequestQueue(
                    new JsonObjectRequest(
                            Request.Method.GET,
                            newURL,
                            new Response.Listener<JSONObject>() {
                                @Override
                                public void onResponse(JSONObject
response)
                                {
```

```
                                    // Procesar la respuesta del
servidor
                                    try {
                                        // Obtener atributo "estado"
                                        String estado =
response.getString("estado");

                                        switch (estado)
                                        {
                                            case "1": // EXITO
                                                Log.d(Constant.TAG,
"respuesta exitosa.");

informesSQLite.deleteInforme(bean.getIDDataBase());

Toast.makeText(mContext, "Insercion correcta",
Toast.LENGTH_LONG).show();
                                                break;
                                            case "2": // FALLIDO
                                                Log.d(Constant.TAG,
"Error al enviar los datos.");
                                                String mensaje2 =
response.getString("mensaje");

Toast.makeText(mContext, mensaje2, Toast.LENGTH_LONG).show();
                                                break;
                                            default:
                                                Log.d(Constant.TAG,
"respuesta error estado:"+estado);
                                                break;
                                        }
                                    } catch (JSONException e) {
                                        e.printStackTrace();
                                    }
                                }
                            },
                            new Response.ErrorListener() {
                                @Override
                                public void
onErrorResponse(VolleyError error) {
                                    Log.d(Constant.TAG, "Error Volley
(response insert): " + error.getMessage());
                                    Toast.makeText(mContext, "No se
comunicó con el Cloud intentelo más tarde.",
Toast.LENGTH_LONG).show();
                                }
                            }

                    ) {
                        @Override
                        public Map<String, String> getHeaders() {
                            Map<String, String> headers = new
HashMap<String, String>();
                            headers.put("Content-Type",
"application/json; charset=utf-8");
                            headers.put("Accept", "application/json");
                            return headers;
                        }

                        @Override
                        public String getBodyContentType() {
```

```
                            return "application/json; charset=utf-8" +
getParamsEncoding();
                        }
                    }
                );
            }
        }
        public void addNewDevice2MySimpleArrayAdapter(final
BeanBluetoothDevice beanDeviceFound) {

            final BluetoothDevice deviceFound =
beanDeviceFound.getBluetoothDevice();
            this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    mActivity.runOnUiThread(new Runnable() {
                        @Override
                        public void run()
                        {
                            mAdapter.addElement(beanDeviceFound);

                            if (Constant.DEBUG)
                                Log.i(Constant.TAG, "ScanActivity --
addNewDevice2MySimpleArrayAdapter() -> Added new device "
                                        + deviceFound.getAddress() + " ---
Name: " + deviceFound.getName());
                        }
                    });
                }
            });
        }
}
```

- **Para ScanArrayAdapter:**

```
package es.Jose.arquetanatureble.GUI;

import android.app.Activity;
import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.List;

import es.Jose.arquetanatureble.BEAN.BeanBluetoothDevice;
import es.Jose.arquetanatureble.R;



/*
    Clase para crear el adaptador de dispositos Bluetooh
     */
public class ScanArrayAdapter extends ArrayAdapter {
    private final Context mContext;
```

```
    private static ScanActivity mScanActivity;
    private List<BeanBluetoothDevice> listBeanBluetoothDevices;

    public ScanArrayAdapter(Context context, List<BeanBluetoothDevice>
deviceList)
    {
        super(context, R.layout.activity_scan_item, deviceList);
        this.mContext = context;
        listBeanBluetoothDevices = deviceList;
    }

    public ScanArrayAdapter(Activity activity,
List<BeanBluetoothDevice> deviceList)
    {
        super(activity.getApplicationContext(),
R.layout.activity_scan_item, deviceList);
        mScanActivity = (ScanActivity)activity;
        mContext = mScanActivity.getApplicationContext();
        listBeanBluetoothDevices = deviceList;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup
parent)
    {
        LayoutInflater inflater = mScanActivity.getLayoutInflater();
        View item = inflater.inflate(R.layout.activity_scan_item,
null);

        TextView nameDevice         = (TextView)
item.findViewById(R.id.deviceName);
        TextView addressDevice      = (TextView)
item.findViewById(R.id.deviceAddress);
        TextView rssilevelTV        = (TextView)
item.findViewById(R.id.rssiLevel);
        ImageView imageView         = (ImageView)
item.findViewById(R.id.chamberPicture);

        final BluetoothDevice bluetoothDevice =
listBeanBluetoothDevices.get(position).getBluetoothDevice();

        nameDevice.setText(bluetoothDevice.getName());
        addressDevice.setText(bluetoothDevice.getAddress());
        int rssi = listBeanBluetoothDevices.get(position).getmRssi();
        rssilevelTV.setText(String.valueOf(rssi));

        imageView.setImageResource(R.drawable.chamber_device);

        return item;
    }

    public void addElement(BeanBluetoothDevice beanBluetoothDevice)
    {
        this.add(beanBluetoothDevice);
        this.notifyDataSetChanged();
    }

    public BeanBluetoothDevice findElement(String name, String
address)
    {
```

```
        for (BeanBluetoothDevice beanBluetoothDevice :
listBeanBluetoothDevices)
        {

if(beanBluetoothDevice.getBluetoothDevice().getName().equals(name)
                        &&
beanBluetoothDevice.getBluetoothDevice().getAddress().equals(address))
                { return beanBluetoothDevice;}
        }

        return null;
    }

    public void deleteElement(BeanBluetoothDevice beanBluetoothDevice)
    {
        listBeanBluetoothDevices.remove(beanBluetoothDevice);
    }
}
```

- **Para Constant:**

```
package es.Jose.arquetanatureble.UTIL;


public final class Constant
{
    public static boolean DEBUG = true;
    public static String TAG    ="IoT-APP";

    //Data to move between activities.
    public static final String EXTRA_ADDRESS                =
"address";
    public static final String EXTRA_NAME                   = "name";
    public static final String EXTRA_BEAN_BLUETOOTHDEVICE   =
"beanbluetoothdevice";
    public static final String EXTRA_INFORMESDB             =
"extra_informesDB";

    //to caugth data from sensors.
    public static final String MSG_FLOW     = "MSG_FLOW";
    public static final String MSG_PRESSU   = "MSG_PRESSU";
    public static final String MSG_VALVE    = "MSG_VALVE";
    public static final String MSG_UNKNOWN  = "MSG_UNKNOWN";
}
```

- **Para BeanArqueta:**

```
package es.Jose.arquetanatureble.WEBSERVICE;


public class BeanArqueta
{
    private  String id;
    private  String insert_time;
    private  String nombre_arqueta;
    private  String direccion_arqueta;
    private  String uuid_sensor1;
    private  String uuid_sensor2;
    private  String uuid_sensor3;
```

```
public BeanArqueta()
{
    super();
}

public  String getId() {
    return id;
}

public  void setId(String id) {
    this.id = id;
}

public  String getInsert_time() {
    return insert_time;
}

public  void setInsert_time(String insert_time) {
    this.insert_time = insert_time;
}

public  String getNombre_arqueta() {
    return nombre_arqueta;
}

public  void setNombre_arqueta(String nombre_arqueta) {
    this.nombre_arqueta = nombre_arqueta;
}

public  String getDireccion_arqueta() {
    return direccion_arqueta;
}

public  void setDireccion_arqueta(String direccion_arqueta) {
    this.direccion_arqueta = direccion_arqueta;
}

public  String getUuid_sensor1() {
    return uuid_sensor1;
}

public  void setUuid_sensor1(String uuid_sensor1) {
    this.uuid_sensor1 = uuid_sensor1;
}

public  String getUuid_sensor2() {
    return uuid_sensor2;
}

public  void setUuid_sensor2(String uuid_sensor2) {
    this.uuid_sensor2 = uuid_sensor2;
}

public  String getUuid_sensor3() {
    return uuid_sensor3;
}

public  void setUuid_sensor3(String uuid_sensor3) {
    this.uuid_sensor3 = uuid_sensor3;
}
```

```
    public String toString()
    {
        return "id="+id+" insert_time = "+insert_time+" nombre_arqueta
="+nombre_arqueta+" direccion_arqueta= "
                +direccion_arqueta+" uuidSensor1= "+uuid_sensor1+"
uuidSensor2= "+uuid_sensor2+
                " uuidSensor3= "+uuid_sensor3;
    }
}
```

- **Para BeanInforme:**

```
package es.Jose.arquetanatureble.WEBSERVICE;

public class BeanInforme
{
    private static String id;
    private static String acceso_ubicacion;
    private static String perimetro_arqueta;
    private static String puerta_acceso;
    private static String cubierta;
    private static String param_verticales_int;
    private static String param_verticales_ext;
    private static String ventilacion_lateral;
    private static String ventilacion_superior;
    private static String pates_escalera;
    private static String distancia_reglamentaria_elementos;
    private static String ventosas;
    private static String juntas_carretes;
    private static String manometros;
    private static String contadores;
    private static String fecha;
    private static String direccion_arqueta;
    private static String comentario;
    private static String foto;

    public BeanInforme()
    {
        super();
    }

    public static String getId() {
        return id;
    }

    public static void setId(String id) {
        BeanInforme.id = id;
    }

    public static String getAcceso_ubicacion() {
        return acceso_ubicacion;
    }

    public static void setAcceso_ubicacion(String acceso_ubicacion) {
        BeanInforme.acceso_ubicacion = acceso_ubicacion;
    }

    public static String getPerimetro_arqueta() {
        return perimetro_arqueta;
```

```
    }

    public static void setPerimetro_arqueta(String perimetro_arqueta)
{
        BeanInforme.perimetro_arqueta = perimetro_arqueta;
    }

    public static String getPuerta_acceso() {
        return puerta_acceso;
    }

    public static void setPuerta_acceso(String puerta_acceso) {
        BeanInforme.puerta_acceso = puerta_acceso;
    }

    public static String getCubierta() {
        return cubierta;
    }

    public static void setCubierta(String cubierta) {
        BeanInforme.cubierta = cubierta;
    }

    public static String getParam_verticales_int() {
        return param_verticales_int;
    }

    public static void setParam_verticales_int(String
param_verticales_int) {
        BeanInforme.param_verticales_int = param_verticales_int;
    }

    public static String getParam_verticales_ext() {
        return param_verticales_ext;
    }

    public static void setParam_verticales_ext(String
param_verticales_ext) {
        BeanInforme.param_verticales_ext = param_verticales_ext;
    }

    public static String getVentilacion_lateral() {
        return ventilacion_lateral;
    }

    public static void setVentilacion_lateral(String
ventilacion_lateral) {
        BeanInforme.ventilacion_lateral = ventilacion_lateral;
    }

    public static String getVentilacion_superior() {
        return ventilacion_superior;
    }

    public static void setVentilacion_superior(String
ventilacion_superior) {
        BeanInforme.ventilacion_superior = ventilacion_superior;
    }

    public static String getPates_escalera() {
        return pates_escalera;
```

```
    }

    public static void setPates_escalera(String pates_escalera) {
        BeanInforme.pates_escalera = pates_escalera;
    }

    public static String getDistancia_reglamentaria_elementos() {
        return distancia_reglamentaria_elementos;
    }

    public static void setDistancia_reglamentaria_elementos(String
distancia_reglamentaria_elementos) {
        BeanInforme.distancia_reglamentaria_elementos =
distancia_reglamentaria_elementos;
    }

    public static String getVentosas() {
        return ventosas;
    }

    public static void setVentosas(String ventosas) {
        BeanInforme.ventosas = ventosas;
    }

    public static String getJuntas_carretes() {
        return juntas_carretes;
    }

    public static void setJuntas_carretes(String juntas_carretes) {
        BeanInforme.juntas_carretes = juntas_carretes;
    }

    public static String getManometros() {
        return manometros;
    }

    public static void setManometros(String manometros) {
        BeanInforme.manometros = manometros;
    }

    public static String getContadores() {
        return contadores;
    }

    public static void setContadores(String contadores) {
        BeanInforme.contadores = contadores;
    }

    public static String getFecha() {
        return fecha;
    }

    public static void setFecha(String fecha) {
        BeanInforme.fecha = fecha;
    }

    public static String getDireccion_arqueta() {
        return direccion_arqueta;
    }
```

```
    public static void setDireccion_arqueta(String direccion_arqueta)
{
        BeanInforme.direccion_arqueta = direccion_arqueta;
    }

    public static String getComentario() {
        return comentario;
    }

    public static void setComentario(String comentario) {
        BeanInforme.comentario = comentario;
    }

    public static String getFoto() {
        return foto;
    }

    public static void setFoto(String foto) {
        BeanInforme.foto = foto;
    }
```

- **Para BeanSector_trabajo:**

```
package es.Jose.arquetanatureble.WEBSERVICE;


public class BeanSector_trabajo
{
    private static String id;
    private static String fecha_control;
    private static String zona;
    private static String localizacion;
    private static String ramal;
    private static String responsable_control;
    private static String responsable_zona;
    private static String tm;
    private static String th;
    private static String direccion_aqueta;

    public BeanSector_trabajo()
    {
        super();
    }

    public static String getId() {
        return id;
    }

    public static void setId(String id) {
        BeanSector_trabajo.id = id;
    }

    public static String getFecha_control() {
        return fecha_control;
    }

    public static void setFecha_control(String fecha_control) {
        BeanSector_trabajo.fecha_control = fecha_control;
    }
```

```
    public static String getZona() {
        return zona;
    }

    public static void setZona(String zona) {
        BeanSector_trabajo.zona = zona;
    }

    public static String getLocalizacion() {
        return localizacion;
    }

    public static void setLocalizacion(String localizacion) {
        BeanSector_trabajo.localizacion = localizacion;
    }

    public static String getRamal() {
        return ramal;
    }

    public static void setRamal(String ramal) {
        BeanSector_trabajo.ramal = ramal;
    }

    public static String getResponsable_control() {
        return responsable_control;
    }

    public static void setResponsable_control(String
responsable_control) {
        BeanSector_trabajo.responsable_control = responsable_control;
    }

    public static String getResponsable_zona() {
        return responsable_zona;
    }

    public static void setResponsable_zona(String responsable_zona) {
        BeanSector_trabajo.responsable_zona = responsable_zona;
    }

    public static String getTm() {
        return tm;
    }

    public static void setTm(String tm) {
        BeanSector_trabajo.tm = tm;
    }

    public static String getTh() {
        return th;
    }

    public static void setTh(String th) {
        BeanSector_trabajo.th = th;
    }

    public static String getDireccion_aqueta() {
        return direccion_aqueta;
    }
```

```
    public static void setDireccion_aqueta(String direccion_aqueta) {
        BeanSector_trabajo.direccion_aqueta = direccion_aqueta;
    }
}
```

- **Para VolleySingleton:**

```
package es.Jose.arquetanatureble.WEBSERVICE;

import android.content.Context;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.toolbox.Volley;


public final class VolleySingleton
{
    private static VolleySingleton singleton;
    private RequestQueue requestQueu;
    private static Context mContext;

    private VolleySingleton(Context context)
    {
        this.mContext = context;
        requestQueu   = getRequestQueu();
    }

    public static synchronized VolleySingleton getInstace(Context
context)
    {
        if(singleton == null)
            singleton = new
VolleySingleton(context.getApplicationContext());

        return singleton;
    }

    public RequestQueue getRequestQueu()
    {
        if(requestQueu == null)
            requestQueu =
Volley.newRequestQueue(mContext.getApplicationContext());

        return requestQueu;
    }

    public <T> void addToRequestQueue(Request<T> req)
    {
        getRequestQueu().add(req);
    }
}
```

- **Para WebServiceConstant:**

```
package es.Jose.arquetanatureble.WEBSERVICE;

public final class WebServiceConstant
{
    public static final int CODIGO_DETALLE       = 100;
    public static final int CODIGO_ACTUALIZACION = 101;

    private static final String PUERTO_HOST      = "63343";
    private static final String IP               =
"http://185.2.130.82/~Jose";

    /*
     WEB SERVICE URLs
     */
    //public static final String ADREESS            = IP + PUERTO_HOST
+ "/webService";
    public static final String ADREESS            = IP +
"/webService";
    public static final String ARQUETA_DIR        = "/arqueta/";
    public static final String INFORME_DIR        = "/informes/";
    public static final String SECTOR_TRABAJO_DIR =
"/sector_trabajo/";

    public static final String GET_ARQUETA_BYID          = ADREESS +
ARQUETA_DIR +"Obtener_arquetaById.php";
    public static final String GET_SECTOR_TRABAJO_BYID   = ADREESS +
SECTOR_TRABAJO_DIR +"Obtener_sector_trabajoById.php";

    public static final String INSERT             = ADREESS +
INFORME_DIR +"Insertar_informe.php";
}
```

Código de Aplicación Web Django.

En el caso de la Aplicación Web de Django, se ha usado el siguiente código, que se ha mencionado cada una de las clases anteriormente:

Para la aplicación creada de boletín, tenemos que :

- ## Script Admin.py:

```
from django.contrib import admin

# Register your models here.
from .models import Registrado
from .forms import RegModelForm


class AdminRegistrado(admin.ModelAdmin):
    list_display=["email", "nombre", "empresa", "timestamp"]
    form = RegModelForm
    list_filter = ["timestamp"]
    list_display_links = ["timestamp"]
    list_editable = ["email", "nombre", "empresa"]
    list_fields = ["email", "nombre", "empresa"]
    #class Meta:
        #model = Registrado



admin.site.register(Registrado, AdminRegistrado)
```

- ## Script App.py:

```
from django.apps import AppConfig


class BoletinConfig(AppConfig):
    name = 'boletin'
```

- ## Script forms.py:

```
from django import forms

from .models import Registrado


class RegModelForm(forms.ModelForm):

    class Meta:

        model = Registrado
        fields = ["nombre","email","empresa"]

    def clean_email(self):

        email =self.cleaned_data.get("email")
```

```
        email_base, proveedor = email.split ("@")
        dominio, extension = proveedor.split(".")

        if not (extension == "es" or extension == "com"):
            raise forms.ValidationError ("Por favor introduzca una
extensión válida")
        return email

class ContactForm(forms.Form):
    nombre = forms.CharField(max_length = 100)
    email = forms.EmailField(max_length = 100)
    empresa = forms.CharField(max_length = 100)
    mensaje = forms.CharField(max_length = 255, widget =
forms.Textarea)
    def clean_email(self):

        email =self.cleaned_data.get("email")
        email_base, proveedor = email.split ("@")
        dominio, extension = proveedor.split(".")

        if not (extension == "es" or extension == "com"):
            raise forms.ValidationError ("Por favor introduzca una
extensión válida")
        return email
```

- **Script model.py:**

```
from django.db import models

# Create your models here.


class Registrado(models.Model):

    nombre = models.CharField(max_length = 100, null = True)
    empresa = models.CharField(max_length = 100, null = True)
    email = models.EmailField()
    timestamp = models.DateTimeField(auto_now_add = True, auto_now =
False)


    def __unicode__(self): #Python 2
        return self.email

    def __str__(self):  #Python 3
        return "El email es : {}, El nombre es: {}, La empresa es :
{}".format(self.email, self.nombre, self.empresa)
```

- **Script views.py:**

```
from django.conf import settings
from django.core.mail import send_mail
from django.shortcuts import render

from .forms import RegModelForm, ContactForm
from .models import Registrado
# Create your views here.

def inicio(request):
    titulo = "Bienvenidos."
    operario = request.user
    form = RegModelForm(request.POST or None)
    if request.user.is_authenticated:
        titulo = " Bienvenid@ {}".format(request.user)


    context = {

        "el_titulo" : titulo,
        "el_form": form,
        "el_operario": operario

    }

    if form.is_valid():

        instance = form.save(commit = False)
        instance.save()
        print (instance)
        print (instance.timestamp)

        context = {

            "el_titulo" : "Gracias {} !".format(instance.nombre)
        }
        """form_data = form.cleaned_data

        print ("Nombre De Registrado : ", form_data.get("nombre"))
        print ("Empresa Del Registrado: ", form_data.get("empresa"))

        nombre_registrado = form.data.get("nombre")
        empresa_registrado = form.data.get("empresa")
        email_registrado = form.data.get("email")
        obj = Registrado.objects.create(nombre = nombre_registrado,
empresa = empresa_registrado,  email = email_registrado)"""

    if request.user.is_authenticated and request.user.is_staff:
        queryset = Registrado.objects.all().order_by("-timestamp")
#.filter(nombre__iexact="karlita")
        context = {
            "queryset": queryset,
        }

    return render(request, "inicio.html", context)

def contact(request):
    titulo = "Contacto"
```

```
    form = ContactForm(request.POST or None)
    context = {

        "form" : form,
        "titulo" : titulo
    }
    if form.is_valid():

        print ("Nombre De Contacto : ",
form.cleaned_data.get("nombre"))
        print ("Email De Contacto : ", form.cleaned_data.get("email"))
        print ("Empresa Del Contacto: ",
form.cleaned_data.get("empresa"))
        print ("Mensaje Del Contacto: ",
form.cleaned_data.get("mensaje"))

        form_nombre = form.cleaned_data.get("nombre")
        form_empresa = form.cleaned_data.get("empresa")
        form_email = form.cleaned_data.get("email")
        form_mensaje = form.cleaned_data.get("mensaje")

        asunto = 'Form de Contacto {}'.format(form_nombre)
        email_from = settings.EMAIL_HOST_USER
        email_to = [email_from]
        email_mensaje = "Mensaje enviado de {} :
{}".format(form_nombre, form_mensaje)

        send_mail(asunto,
            email_mensaje,
            email_from,
            email_to,
            fail_silently = False
            )


        context = {

            "titulo" : "Gracias {} !".format(form_nombre)
        }
        print(form.cleaned_data)

    return render(request, "contact.html", context)
```

Para el código de la configuración de la Aplicación Web, tenemos:

- **Script Settings.py:**

```
"""
Django settings for LowBlue project.

Generated by 'django-admin startproject' using Django 1.10.

For more information on this file, see
https://docs.djangoproject.com/en/1.10/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.10/ref/settings/
"""
```

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR,
...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See
https://docs.djangoproject.com/en/1.10/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'c9$*z1$$ro6w_d1)mapn18_ydabtna1cif$^ezb3a2tze&)5e&'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

EMAIL_HOST = "smtp.gmail.com"
EMAIL_HOST_USER = "air.pepeto@gmail.com"
EMAIL_HOST_PASSWORD = "hcjosdyfwbpqzaxu"
EMAIL_PORT = 587
EMAIL_USE_TLS = True



# Application definition

INSTALLED_APPS = [
    #apps django
    'django.contrib.sites',
    'registration', #should be immediately above
'django.contrib.admin'
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',

    'django.contrib.messages',
    'django.contrib.staticfiles',

    #app terceros
    'crispy_forms',


    #mis apps
    'boletin',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

```
ROOT_URLCONF = 'LowBlue.urls'

CRISPY_TEMPLATE_PACK = 'bootstrap3'
ACCOUNT_ACTIVATION_DAYS = 7
REGISTRATION_AUTO_LOGIN = True
LOGIN_REDIRECT_URL = ('/')
SITE_ID = 1

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, "templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },

]

WSGI_APPLICATION = 'LowBlue.wsgi.application'


# Database
# https://docs.djangoproject.com/en/1.10/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/1.10/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValida
tor',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
```

```
]


# Internationalization
# https://docs.djangoproject.com/en/1.10/topics/i18n/

LANGUAGE_CODE = 'es-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.10/howto/static-files/

STATIC_URL = '/static/'
MEDIA_URL = '/media/'
#/static/imagenes/img1.jpg


STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static_pro", "static"),
    #'/var/www/static/',
]

STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env",
"static_root")
MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_env",
"media_root")
```

- **Script urls.py:**

```
from django.urls import path
from django.conf.urls import url, include
from django.contrib import admin
from django.conf import settings
from django.conf.urls.static import static
from django.contrib.auth import views as auth_views
from boletin import views
from .views import about

"""urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^contact/$', views.contact, name='contact'),
    url(r'^about/$', about, name='about'),
    url(r'^$', views.inicio, name='inicio'),
    url(r'^accounts/', include('registration.backends.default.urls')),

]"""

urlpatterns = [
    path('admin/', admin.site.urls),
    path('contact/', views.contact, name='contact'),
    path('about/', about, name='about'),
```

```
    path('', views.inicio, name='inicio'),
    url(r'^accounts/', include('registration.backends.default.urls')),

]

if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

- **Script Views.py:**

```
from django.shortcuts import render


def about(request):

    return render(request, "about.html", {})
```

Finalmente, para los templates de la Aplicación Web:

- **Código html de about:**

```
{% extends "base.html" %}

{% block content %}
<div class='row'>
    <div class='col-sm-6 col-sm-offset-3'>
        <h2 class="text-align-
center"><strong>Historia</strong></h2><hr/>
        <p class="lead text-align-center">¡Es una historia muy larga!
¿   Tienes tiempo?</p>
        <p class="text-align-center">Este proyecto es fruto del
proyecto</p>
        <p class="text-align-center">final de grado de la Universidad
</p>
        <p class="text-align-center">Politécnica de Cartagena, hecho
por</p>
        <p class="text-align-center">José Benavente Pérez y Miguel
Alarcón.</p>
        <p class="text-align-center">Agradecimientos a todos los
familiares</p>
        <p class="text-align-center">y cercanos.
</p>

    </div>
</div>

{% endblock %}
```

- **Template de la base:**

```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="../../../../favicon.ico">

    <title>{% block head_title %}{% endblock %} LOWBLUE
PROJECT</title>
<style>
{% block style %}{% endblock %}
</style>


   {% include "head_css.html" %}
  </head>

  <body>
{% include "navbar.html" %}


{% block jumbotron %}

{% endblock %}
    <main role="main" class="container-fluid">
{% block content %}
{% endblock %}

    </main>

    <!-- Bootstrap core JavaScript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster --
>
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
    <script>window.jQuery || document.write('<script
src="../../assets/js/vendor/jquery-slim.min.js"><\/script>')</script>
    <script src="{% static 'js/popper.min.js' %}"></script>
    <script src="{% static 'js/bootstrap.min.js' %}"></script>
  </body>
</html>
```

- **Para el contact:**

```
{% extends "base.html" %}
{% load crispy_forms_tags  %}

{{ el_titulo }}
<hr/>
<br/>


{% block content %}
<div class='row'>
    <div class='col-sm-6'>

{% if titulo %}

<h1 class = 'text-align-center'>{{ titulo }}</h1>

{% endif %}

<form method="POST" action = "">{% csrf_token %}

{{form|crispy}}

<input class='btn btn-primary' type="submit" value="Enviar"/>

</form>

    </div>
</div>
{% endblock %}
```

- **En el caso de forms:**

```
{% extends "base.html" %}
{% load crispy_forms_tags %}


{% block content %}
<div class='row'>
    <div class='col-sm-6 col-sm-offset-3'>
{% if titulo %}
<h2 class='text-align-center'><strong>{{ titulo }}</strong></h2><hr/>
{% endif %}
<form method="POST" action="">{% csrf_token %}
{{ form|crispy }}
<input class='btn btn-primary' type='submit' value='Registrame' />
</form>
</div>
</div>
{% endblock %}
```

- **Los links de Bootstrap para css:**

```
{% load static %}
 <!-- Bootstrap core CSS -->
    <link href="{% static 'css/bootstrap.min.css' %}"
rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="{% static 'css/navbar-top.css' %}" rel="stylesheet">

    <link href="{% static 'css/custom.css' %}" rel="stylesheet">

    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.0.13/css/all.css"
integrity="sha384-
DNOHZ68U8hZfKXOrtjWvjxusGo9WQnrNx2sqG0tfsghAvtVlRW3tvkXWZh58N9jp"
crossorigin="anonymous">
```

- **Para el template de inicio:**

```
{% extends "base.html" %}
{% load crispy_forms_tags %}
{% block head_title %} Bienvenidos |{% endblock %}
<style>
{% block style %}
.jumbotron {
    background-color: #172548 !important;
    color: white !important;
}
{% endblock %}
</style>
{% block jumbotron %}

{% if request.user.is_staff %}
<table class= "table">
<tr><strong><div class = 'lead'>Usuarios
Suscritos:</strong></tr></div>
{% for instance in queryset %}
 <tr><td>{{ forloop.counter }}</td><td>{{ instance.nombre }}</td><td>
{{ instance.email }}</td> <td>{{ instance.empresa }}<td> hace : {{
instance.timestamp|timesince }}</td> </tr></td>
{% endfor %}
</table>
{% else %}

<div class="jumbotron">
{% if not request.user.is_authenticated %}
<div class = "row">
    <div class="col-sm-6">
        <h1>LOWBLUE PROJECT</h1>
        <p class="lead">Registrarme como operario.</p>
        <a class="btn btn-lg btn-primary" href="/accounts/login"
role="button">Entrar &raquo;</a>
    </div>
```

```
    <div class="col-sm-6"><img src =
'http://girtel.upct.es/projects/fierro/img/logos/logo_upct.png' class
= 'img-responsive'/> </div>

</div>
</div>

<div class = "row"></div>
</div>

{% else %}
<div class = "row">
    <div class="col-sm-6">
        <h1>LOWBLUE PROJECT</h1>
        <p class="lead">He iniciado sesión como el operario {{
el_operario }}</p>
        <a class="btn btn-lg btn-primary" href="/accounts/login"
role="button">Registro de Arquetas &raquo;</a>
    </div>
    <div class="col-sm-6"><img src =
'http://girtel.upct.es/projects/fierro/img/logos/logo_upct.png' class
= 'img-responsive'/> </div>

</div>
</div>

<div class = "row"></div>
</div>

{% endif %}
{% endif %}
{% endblock %}
<strong>{{ el_titulo }}</strong>
<hr/>
<br/>

{% block content %}
{% if not request.user.is_staff %}


<div class="row">
    <div class =" col-sm-3 pull-right">

        {% if not user.is_authenticated %}
        <p class = 'lead'>Suscríbete</p>
        <form method="POST" action = "">{% csrf_token %}
        {{ el_form|crispy }}
        <input class="btn btn-primary" type="submit" value="Enviar"/>
        </form>
        {% else %}
        <p class = 'lead'>{{ el_titulo }}</p>
        <br/><span class="fa-stack fa-4x">
      <i class="fa fa-circle-thin fa-stack-2x"
style="color:#47b78c;"></i>
      <i class="far fa-hand-peace fa-stack-2x"></i>
    </span></p>
        {% endif %}
    </div>
<div class='col-sm-3'>
   <strong> <p class = 'lead text-align-center'> EFICIENTE Y
RÁPIDO</strong><br/><br/><span class="fa-stack fa-4x">
```

```
  <i class="fa fa-circle-notch fa-spin fa-stack-2x"></i>
  <i class="fa fa-bolt fa-stack-1x" style="color:#47b78c;"></i>
</span></p>
</div>
<div class='col-sm-3'>
    <strong><p class = 'lead text-align-center'> INTUITIVO
</strong><br/><br/><span class="fa-stack fa-4x">
  <i class="fa fa-circle-notch fa-spin fa-stack-2x"></i>
  <i class="fa fa-check fa-stack-1x" style="color:#47b78c;"></i>
</span></p>
</div>
<div class='col-sm-3'>
    <strong><p class = 'lead text-align-center'> ESCALABLE Y CÓDIGO
ABIERTO</strong><br/><br/><span class="fa-stack fa-4x">
  <i class="fa fa-circle-notch fa-spin fa-stack-2x"></i>
  <i class="fa fa-pencil-alt fa-stack-1x" style="color:#47b78c;"></i>
</span></p>


</div>
</div>


<hr/>
{% endif %}
{% endblock %}
```

- **Links javascript:**

```
{% load static %}
    <!-- Bootstrap core JavaScript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster --
>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js
"></script>
    <script>window.jQuery || document.write('<script
src="../../assets/js/vendor/jquery.min.js"><\/script>')</script>
    <script src="{% static 'js/bootstrap.min.js' %}"></script>
    <!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
    <script src="{% static 'js/ie10-viewport-bug-workaround.js'
%}"></script>
```

- **Plantilla de NavBar de Bootstrap:**

```
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
      <a class="navbar-brand" href="/">LOWBLUE</a>
      <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarCollapse" aria-
controls="navbarCollapse" aria-expanded="false" aria-label="Toggle
navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarCollapse">
        <ul class="navbar-nav mr-auto">
          <li class="nav-item active">
```

```
            <a class="nav-link" href="/">Inicio <span class="sr-
only">(current)</span></a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="{% url 'contact'
%}">Contacto</a>
          </li>
          <li class="nav-item">
            <a class="nav-link disabled" href="{% url 'about'
%}">Sobre Nosotros</a>
{% if request.user.is_staff %}
          </li>
        <li class="nav-item">
            <a class="nav-link active"
href="admin/">Administración</a>
          </li>
 {% endif %}
        </ul>
{% if not request.user.is_authenticated and not "/accounts/login/" in
request.get_full_path %}
    <form class='navbar-nav navbar-right' method='POST' action="{% url
'auth_login' %}">{% csrf_token %}

        <div class='form-group'>
            <input type='text' class='form-control' name='username'
placeholder="Usuario" />
        </div>

        <div class='form-group'>
            <input type='password' class='form-control'
name='password' placeholder="Contraseña" />
        </div>
    <button type="submit" class='btn btn-default'>Entrar</button>
    </form>
     {% endif %}
    <ul class="nav navbar-nav navbar-right">
        {% if request.user.is_authenticated %}
        <li class="nav-item">
            <a class="nav-link active" href="{% url 'auth_logout'
%}">Salir</a>
          </li>
        {% else %}
          <li class="nav-item">
            <a class="nav-link active" href="{% url
'registration_register' %}">Registrarte</a>
          </li>

        {% endif %}
    </ul>

<!--          <form class="form-inline mt-2 mt-md-0">
          <input class="form-control mr-sm-2" type="text"
placeholder="Search" aria-label="Search">
          <button class="btn btn-outline-success my-2 my-sm-0"
type="submit">Search</button>
        </form> -->

      </div>
    </nav>
```