Sistema de Control de una Cámara Frigorífica mediante Smartphone

19 / 06 / 2017

TRABAJO FINAL DE GRADO, MODALIDAD B

ESTUDIANTE:

Abella Gassol, Arnau

DIRECTOR:

Cabezas García, Jorge

PONENTE:

Llorente Viejo, Silvia

FACULTAD DE INFORMÁTICA DE BARCELONA

UNIVERSIDAD POLITÉCNICA DE CATALUÑA

Resum

Durant l'últim lustre s'ha produït un gran avanç en l'*Internet of things (IoT)*, que consisteix en la integració de les tecnologies de la informació en eines quotidianes per a l'home per millorar així la seva gestió i ampliar les seves funcionalitats. A l'àmbit industrial, cada vegada més, els diferents sistemes de producció s'estan adaptant a aquestes noves tecnologies per a millorar la seva productivitat i d'aquesta forma l'empresa pot obtenir un major benefici.

Aquest projecte presenta una solució per a poder adaptar un sistema frigorífic industrial. Aquest sistema podrà ser operat a distància a través d'una aplicació per *smartphone* sigui Android o iOS. Un cop adaptat el sistema al IoT, s'obté una gestió energètica més eficient, s'estalvia la necessitat implícita que cada cop que es vulgui modificar el sistema, hi hagi d'assistir un tècnic en persona i es facilita l'ús de tot el sistema entre moltes altres coses que explicarem al llarg d'aquest projecte de final de grau.

Resumen

Durante el último lustro se ha producido un gran avance en el *Internet of things (IoT)* que consiste en la integración de las tecnologías de la información en objetos cotidianos para el hombre para mejorar así su gestión y ampliar sus funcionalidades. En el entorno industrial, cada vez más, los diferentes sistemas de producción se están adaptando a estas nuevas tecnologías para así mejorar su productividad y de esta forma, obtener un mayor beneficio.

Este proyecto presenta una solución para poder adaptar un sistema frigorífico industrial para que este pueda ser operado a distancia a través de una aplicación para *smartphone* ya sea Android o iOS. Al adaptar el sistema frigorífico al IoT, se obtiene una gestión más eficiente de la energía, se ahorra la necesidad de que un técnico asista físicamente al sistema para realizar cambios en éste y se facilita el uso del sistema entre otras muchas cosas que se explicarán a lo largo de este trabajo de final de grado.

Abstract

During the last five years there has been a breakthrough in the Internet of Things (IoT), which consists of integrating Information Technologies (IT) into everyday objects to improve their management and expand their functionalities. In the industrial environment, nowadays, different production systems are adapting to these new technologies to improve their productivity and thus, to obtain a greater benefit.

This project presents a solution to adapt an industrial refrigeration system so that it can be operated remotely through a smartphone application for Android or iOS. By adapting the refrigeration system to the IoT, a more efficient management of energy is obtained. Also, the management of the system is simplified for the final users and the use of the application saves the need for a technical technician to modify the inner state of the system. This final degree project explains in detail how we adapted the whole system in order to be operated with a smartphone.

Índice de tablas

Índice de figuras

Índice de Contenido

1.- Introducción

1.1.- Descripción del proyecto

Este proyecto se realiza como Trabajo Final de Grado (TFG) de Ingeniería Informática, especialidad en Tecnologías de la Información, en la Facultad de Informática de Barcelona, Universidad Politécnica de Catalunya (UPC) conjuntamente con la empresa española BlitWorks, donde trabajo actualmente, especializada en *game porting*. El *game porting*, o mejor conocido como *porting,* es el proceso de adaptar software que pueda ser compilado y ejecutado en un entorno computacional distinto al original por el que fue diseñado (por ejemplo: diferente arquitectura de CPU, sistemas operativos y librerías distintos) [1][2].

Este proyecto es un encargo por parte de la multinacional española Ako. En particular, la subsección de la empresa que nos ha encargado el proyecto es AkoSys especializada en soluciones de control y regulación de temperatura para el sector de la refrigeración industrial y comercial, encargados de preservar la cadena de frío [3]. Debido al auge de de las tecnologías y en especial de las tecnologías portables como sería el *smartphone*, Ako ha querido renovar sus sistemas de refrigeración incorporando en los microcontroladores encargados de esta tarea una interfaz capaz de comunicarse con un control a distancia. En vez de crear un sistema específico para cada uno de los diferentes modelos de controladores de cámaras, Ako apuesta por una solución unificadora que es incorporar este control a distancia en el móvil mediante una aplicación.

1.2.- Formulación del problema

En la actualidad, nos encontramos que los dispositivos de refrigeración de Ako todavía se basan en microcontroladores monolíticos encargados de regular: temperatura, humedad, estado del contenido mediante una serie de alarmas reconfigurables (de gas, personas, componentes químicos del aire, etc). Asimismo, recogen estadísticas y valores de estos datos a lo largo del día, guardando un histórico de valores almacenados durante meses ya que son necesarios para las auditorías de calidad y seguridad. Estas cámaras se configuran durante su puesta en marcha dependiendo de las características de la cámara y del contenido de su interior. Los valores del dispositivo no se pueden modificar, únicamente un técnico puede modificar los parámetros de configuración estando físicamente con el equipo y con un instrumento especial. Exceptuando los modelos más caros, que debido a su gran configurabilidad, pueden ser controlados desde un servidor externo. Debido a esto, estos dispositivos deben estar conectados para poder ser

configurados y en muchos casos no es posible encontrar una entrada a Ethernet debido a la situación geográfica de estos.

Con la llegada de *Internet of Things* (IoT), concepto que se refiere a la interconexión digital de objetos cotidianos con internet con el fin de comunicar y obtener datos de estos [4], Ako decidió dar un paso hacia delante para satisfacer las necesidades de usuarios cada vez con requisitos más complejos y así sobrepasar a la competencia del sector (KIDE, Isotermia...). Pero, ¿cómo se integrarán estas comunicaciones con los microcontroladores de baja potencia? ¿Será posible controlar estos sistemas a distancia sin alterar el correcto funcionamiento de las cámaras? ¿Qué tecnologías actuales se van a usar para poder comunicar estos sistemas monolíticos con el exterior?

Para responder a estas preguntas Ako acudió a BlitWorks para buscar una solución. Este proyecto se centra en la creación de un sistema de control remoto de todos los modelos de microcontroladores de cámaras frigoríficas de Ako mediante el uso de hardware cotidiano que encontraremos en cualquier sistema *smartphone* (tanto iOS como Android) y en software específico diseñado conjuntamente con los ingenieros informáticos de ambas empresas.

1.3.- Objetivos del proyecto

El objetivo de este proyecto es la creación de un sistema de gestión remoto de microcontroladores de cámaras frigoríficas de la empresa Ako integrado en un sistema *smartphone* con un requisitos de hardware mínimos y portable tanto en distribuciones iOS como Android. Debido a que el proyecto se inicializará al mismo tiempo que los ingenieros de hardware y firmware de Ako empiecen a trabajar en los nuevos modelos de microcontroladores parte del diseño y funcionalidades se modificará a medida que el hardware del microcontrolador sea especificado.

La primera versión de este software correrá en una arquitectura iOS utilizando el hardware de iPhones versiones 4 o superior de la empresa Apple. Este trabajo de final de carrera se centrará únicamente en la aplicación para iOS en la cual estamos trabajando en la actualidad. En un futuro, una vez finalizado este proyecto, se realizará el trabajo de cambio de distribución de iOS 9 a Android 7 u 8. Uno de los puntos claves de este proyecto será la comunicación entre el microcontrolador y el smartphone que en secciones posteriores se explicará con detalle. El control remoto integrado en el *smartphone* ha de ser capaz de :

- Comunicarse con el microcontrolador de distintos modelos de cámaras frigoríficas siendo capaz de distinguir el modelo y sus componentes.
- Obtener el estado interno del microcontrolador y de sus diferentes sensores. Por ejemplo, permite visualizar si ha saltado una alerta por baja presión del líquido frigorífico.
- Modificar el estado interno del microcontrolador sin alterar el correcto funcionamiento de este. Esto implica que el dispositivo móvil entiende la configuración del microdispositivo y evita configuraciones de parámetros inválidas o incoherentes.
- Obtener estadísticas y datos históricos del microcontrolador que éste almacena de forma periódica en su memoria persistente. Y permitir al usuario visualizar estos datos en forma de gráficos continuos y tendenciales y comparar los datos con otros datos proporcionados por el usuario.
- Crear configuraciones de arranque del sistema mediante preguntas complejas que definan de forma transparente al instalador parámetros del sistema.
- Programar auditorías a distancia, mediante una interfaz sencilla y fácil de usar.
- Otros, todavía por definir.

1.4.- Contexto

En los puntos anteriores se han introducido los motivos de la creación de este proyecto, el problema general que intenta resolver y los objetivos, es decir, los problemas concretos que tiene que resolver este proyecto. En este punto hablaremos de por qué el uso del *smartphone* como control remoto/visualizador de datos en vez de otros sistemas y también que nos ha llevado a usar las distribuciones iOS y Android en vez de otros sistemas más específicos. Los móviles inteligentes son el dispositivo electrónico más demandado en la actualidad debido a un cambio del estilo de vida de la gente, donde poseer un *smartphone* no es algo opcional sino una necesidad. Un aumento del sueldo medio de los países emergentes ha potenciado la probabilidad de que estos gasten en media, entretenimiento, conectividad y comunicaciones móviles, llevando a los smartphones al primer lugar a nivel de ventas. En la actualidad se calcula que hay alrededor de 2.1 billones de *smartphones* en el mundo pero que este valor podría llegar a subir hasta los 3 billones incluso más con la aparición del IoT [5].

Debido a que los nuevos modelos de *smartphones*, en general, llevan incorporado una serie de elementos hardware diseñados específicamente para comunicar el dispositivo con el exterior los hacen los perfectos candidatos para su uso como control remoto. Es cierto que se podría haber seleccionado otro tipo de sistema más concreto y de menor coste ya que el precio medio de un *smartphone,* en 2017, se sitúa entre los 210$ para una distribución Android [6] y para la distribución iOS en los 700$ en media mundial [6]. Es cierto que estos productos son “caros” pero los usuarios ya dispondrán de *smartphone* antes de comprar el producto de Ako por lo que el hardware necesario para manipular el dispositivo les saldrá a posteriori gratuito. Por estos dos motivos, abundancia del hardware entre los posibles usuarios del producto y falta de necesidad de compra exclusiva para su uso, hace el *smartphone* el hardware ideal para la creación de este control a distancia.

Una vez que hemos entendido el porqué del uso de los *smartphones* como hardware específico para el control de los microcontroladores de las cámaras frigoríficas vamos a hablar de las diferentes distribuciones que pueden ser ejecutadas en estos dispositivos. Una distribución o sistema operativo es un sistema software (código) que se encarga de gestionar el hardware (componentes físicos) y de proporcionar al software de alto nivel una interfaz común para el uso de este hardware. Por ejemplo, las peticiones de entrada/salida o de alocación de memoria han de pasar por el sistema operativo para que el hardware las realice. El sistema operativo también se encarga de realizar de forma eficiente estas funciones y evitar acciones ilegales por parte de los programas [7]. En el caso de los *smartphones* el hardware es muy distinto entre los diferentes modelos y marcas pero gracias a los sistemas operativos el programador puede abstraerse del hardware y que este sea transparente al código. Gracias a la unificación de los sistemas operativos en los móviles, programar para ellos es cada vez más sencillo y genérico.

Antes de empezar el proyecto se realizó un estudio de mercado. En España (mercado inicial del producto) el sistema operativo más usado para móviles inteligentes es Android, sin especificar una versión, seguido por iOS y muy por detrás Windows Phone [8]. La tendencia mundial,como podemos observar en la figura 1, muy similar a la española. Por este motivo se han escogido como target los *smartphone* que corren sistemas operativos iOS y Android.

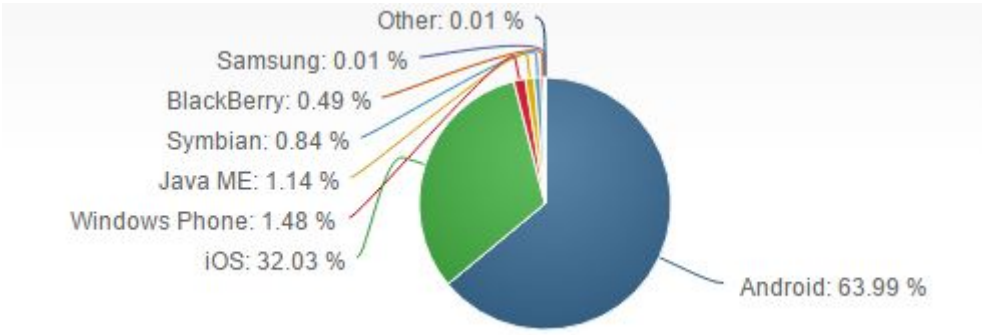


Figura 1. Gráfico de distribución de sistemas operativos en smartphones [9].

En el tercer punto explicaré la versión de sistema operativo para la que se programará el proyecto, tanto para iOS como para Android y los lenguajes de programación en los que se hará esta programación. Una de las grandes ventajas de estos sistemas operativos es que a lo largo de los años se han ido perfeccionando para facilitar a los desarrolladores su uso y se han creado herramientas específicas para programar para estas distribuciones conocidas como entornos de desarrollo integrados o IDE. En el caso de Android encontramos Android Studio y para iOS Xcode.

1.5.- Stakeholders

Los stakeholders son todas aquellas personas, grupos y entidades que tienen intereses de cualquier tipo en una empresa y se ven afectados por su actividad. En nuestro caso nos centraremos en estos tres stakeholders: la audiencia que usará nuestro producto, los usuarios de la aplicación y finalmente en los beneficiarios de esta.

Aquí tenemos que diferenciar entre dos elementos, el microcontrolador de la cámara refrigeradora y el control a distancia (aplicación para *smartphone*). Ambos productos son comprados a la vez pero los stakeholders que tienen son diferentes.

Los microcontroladores de cámaras frigoríficas han sido fabricados para el perfil de empresas con la necesidad de controlar la temperatura de conservación de un género (sea comida, medicamentos, productos frágiles, etc), por ejemplo cadenas de comida rápida como Burger King o farmacéuticas como Abalon Pharma. Son productos caros y que suelen ser vendidos en grandes cantidades ya que producirlos tiene un elevado coste.

Los usuarios de la aplicación suelen ser técnicos, con una educación superior, especializados ya que modificar los parámetros de estos dispositivos tiene un cierto riesgo y podría dañar los productos del interior. El producto se lanzará en diferentes idiomas incluyendo

español, inglés, francés, alemán, italiano, etc. Se va a distribuir en muchos países, de diferentes continentes, por lo que se tiene que evitar iconología que dé lugar a malentendidos y usar un lenguaje técnico y preciso.

Los beneficiarios de este producto son aquellas empresas que usan las cámaras frigoríficas para almacenar sus productos , también las empresas con técnicos encargados del mantenimiento de estas cámaras y finalmente los ingenieros responsables del software.

1.5.- Alcance

1.5.1.- Alcance del proyecto

El alcance de este proyecto es concreto y consiste en diseñar y fabricar un conjunto de hardware (en nuestro caso usaremos un hardware ya creado, los móviles) y software (aplicación iOS/Android) capaz de controlar, manipular y visualizar el estado de un µcrontrolador de una cámara frigorífica industrial para la empresa Ako. Este proyecto en un principio sólo afectará a los clientes directos de Ako y a todos los usuarios del producto pero espero en un futuro que este proyecto, si tiene éxito, anime a las otras empresas del sector a realizar integraciones de tecnologías en sus productos con tal de hacer cada día más plausible la idea de IoT. A medida que las tecnologías de integración avancen más y más sistemas monolíticos se irán incorporando a la red y su coste se reducirá. Es importante realizar un gran trabajo de estandarización hacia el IoT global y aprovechándose de los pasos ya realizados por los *smartphones* podemos conseguir que este camino sea menos arduo.

1.5.2.- Posibles obstáculos

- Pérdida de datos parcial: el contenido del proyecto local de cada trabajador puede perderse debido a fallo del SSD (Solid State Disk), interrupción de write del sistema operativo en el disco, cierre por *crash* del programa de edición, …
    - Solución: guardar de forma periódica (intervalos de 60 minutos) el contenido del trabajo y al finalizar la jornada laboral guardar el contenido en un servidor externo.

- Pérdida de datos total: el contenido del repositorio externo se pierde o quede completamente inutilizable debido a rotura de los discos de almacenaje o debido a destrucción del hardware por incendio/inundación.
    - Solución: se podría hacer *mirroring* del contenido del servidor en otro servidor localizado en una zona geográfica diferente.
- Incumplimiento del contrato por parte de la empresa contratista: la empresa contratista, Ako, cambia de planes o sufre quiebra económica y los fondos para realizar el proyecto desaparecen.
    - Solución: a nivel empresarial el proyecto se cancelaría, pero de forma autónoma podría continuar con él pidiendo a la empresa permiso.

1.6.- Rol personal en el proyecto

Dada la dimensionalidad del proyecto es importante indicar que rol tengo en éste, de qué tareas estoy a cargo y en qué tareas ayudo. Destacar que este proyecto se centra únicamente en la creación de la aplicación y todas las infraestructuras de gestión de ésta como pueden ser los servidores, pero es importante recordar que en el proyecto también se debería incluir los diferentes modelos de microcontroladores.

A nivel de empresas, Ako se encarga del microcontrolador de cada uno de los modelos y del *server-side* de Bluetooth LE. Aparte tiene el rol de líder y se encarga de todas las decisiones, tanto de diseño como de funcionalidad. BlitWorks está a cargo de crear la aplicación para Android e iOS, del *client-side* de Bluetooth LE y también de las infraestructuras auxiliares de gestión de usuarios y del servidor de comunicación entre aplicaciones.

Mi rol principal en este proyecto es el *porting* de la aplicación de iOS a Android. Como *porting engineer* en BlitWorks estoy acostumbrado a trabajar con puertos de juegos a iOS, conozco el funcionamiento básico del sistema operativo y la utilización de librerías gráficas como Metal. Por este motivo se me asignó ayudar en la creación inicial de la aplicación para iOS y posteriormente se me asignó el trabajo de crear toda la aplicación para Android. También se me ha encargado la creación de los servidores Servlet y Node.js y programar el *client-side* para Android e iOS.

1.7.- Beneficios aportados

El proyecto no únicamente ofrece un sistema de control a distancia para unos modelos específicos de microcontroladores de cámaras frigoríficas, ya que si sólo este fuera el objetivo se podría haber llevado a cabo de una forma más sencilla. El proyecto ofrece un sistema software "universal" para cualquier tipo de hardware para la gestión de una cámara frigorífica (siempre y cuando este hardware soporte el sistema operativo Android API 21 / iOS 9.0 o superior, con un chipset de Bluetooth Low Energy y con un módulo de conectividad, ya sea Wi-Fi o 3G/4G/5G). Una de las virtudes de contar con una versión APK para Android es que puede ser instalado en gran multitud de hardware.

Adicionalmente, esta aplicación proporciona una funcionalidad muy parecida a la que todo objeto IoT ofrecerá en el futuro, ser capaz de comunicarse con otros sistemas/dispositivos a través de Internet. En nuestro caso, necesitamos usar un móvil como interfaz para esta operación (funcionalidad de Share) ya que no contamos con la tecnología necesaria para implementar la funcionalidad en el propio microcontrolador. Gracias a este proyecto, se dará un paso en el mundo de IoT para sistemas de refrigeración industrial. Creemos que la mejora que aportará esta aplicación hará abrir los ojos a los principales fabricantes de sistemas de refrigeración para que empiecen a integrar este tipo de tecnología en todos sus sistemas.

A nivel personal, este proyecto me ha ayudado a mejorar mi conocimiento sobre la creación de aplicaciones en Android, iOS y Windows Phone. A como se debe gestionar un proyecto de grandes dimensiones, a como se debe trabajar en equipo en una empresa y como se debe redactar un informe técnico.

2.- Estado del arte

A diferencia de otros proyectos más generales, de los cuales se pueden encontrar proyectos con el mismo propósito pero con distinto enfoque, cuesta encontrar proyectos parecidos a nivel Europeo, se podría considerar que la competencia es inexistente. Existen cámaras frigoríficas que pueden ser controladas a distancia a través de un dispositivo diseñado específicamente para estas cámaras pero en ningún caso un microcontrolador gobernado por una aplicación mediante *smartphone.* Es cierto que existen aplicaciones para controlar televisiones como podría ser LG TV Remote Control [10] o para controlar aires acondicionados como por ejemplo ASmart Remote IR [11].

En el pasado, era común que cada dispositivo industrial tuviera su propio controlador debido a que no existían dispositivos portables con la potencia computacional de los actuales *smartphones*. En el presente, con la industria de los *smartphones* en auge, es impactante encontrar un dispositivo que no se controle a través de una aplicación de ordenador de sobremesa o de una aplicación móvil. Los dispositivos móviles, con la necesidad de integrar tecnologías de la información en las cadenas de producción, han encontrado un hueco en el sector industrial. Los fabricantes de estos dispositivos se han dado cuenta de esto, y en los nuevos modelos de estos dispositivos, observamos que se ha incrementado el número de sensores y de tecnologías de comunicación. Nuestra función en este proyecto es adaptar las tecnologías ya existentes de los controladores frigoríficos para mejorar sus funcionalidades y adaptarlas a los tiempos modernos para incrementar su productividad.

2.1.- Tecnologías usadas

2.1.1.- Bluetooth LE

Bluetooth Low Energy o también conocido como Bluetooth Smart es un subconjunto del estándar de Bluetooth versión 4.0, compuesto de una nueva torre de protocolos que garantiza sencillez y velocidad. Antes de hablar de Bluetooth LE, explicaremos los cimientos de éste.

Bluetooth es una tecnología *wireless* estándar para el intercambio de datos a corta distancia, usa las radiofrecuencias de onda corta UHF (Ultra High Frequency) de la banda ISM (*Industry, Scientific and Medical*) que va desde los 2400 a los 2483.5 MHz. Bluetooth utiliza una tecnología de radio conocida como *frequency-hopping spread spectrum*. Los datos a transmitir se dividen en paquetes, cada paquete se transmite en uno de los 79 canales designados. Cada canal tiene un ancho de banda de 1 MHz. En media, se realizan 800 saltos por segundo si AFH

(*Adaptive Frequency-Hopping*) está activado. La versión LE utiliza 40 canales y cada uno de ellos con el doble de ancho de banda, 2 MHz. Originalmente, se usaba la modulación GFSK (*Gaussian Frequency-Shift Keying*) ya que era la única, pero a partir de la versión 2.0+EDR, se introdujo la modulación π/4-DQPSK (*Differential Quadrature Phase Shift Keying*) y 8-DPSK, también conocidos como EDR (Enhanced Data Rate) con los que se consiguió pasar de una velocidad de 1 Mbps a 2 Mbps y 3 Mbps respectivamente [12].

Bluetooth es un protocolo basado en una comunicación por paquetes con una arquitectura maestro-esclavo. Un dispositivo maestro se puede comunicar con hasta 7 dispositivos esclavos dentro de la misma piconet (nombre denominador de las redes formadas por dispositivos Bluetooth). Todos los dispositivos comparten el *clock* del maestro. El intercambio de paquetes utiliza el *clock* básico de *ticks* con intervalos de 312.5 µs. El *slot* está compuesto de 2 ticks, 625 µs. En el caso más sencillo, el maestro envía en slots pares y recibe en slots impares, al contrario que los esclavos. Los paquetes tienen tamaños variables de 1, 3 y 5 slots. Todo lo explicado es cierto para Bluetooth, pero recordemos que la especificación 4.0 LE es un poco diferente, a continuación explicaremos las diferencias principales que encontramos en esta nueva versión [12].

2.1.1.1.- Bluetooth versión 4.0

La versión 4.0 de Bluetooth Core también conocida como Bluetooth Smart fue publicada en Junio de 2010. Incluye tres nuevas subversiones de Bluetooth: *Classic*, *High-speed* y *Low-energy*.

El primero está basado en las anteriores versiones. El de alta velocidad se basa en Wi-Fi 802.11n. El último, el que usaremos en nuestro proyecto para comunicar el *smartphone* con el microcontrolador, está basado en una torre nueva de protocolos. Éste pretende ser una alternativa de bajo consumo eléctrico. Los chips de Bluetooth LE pueden implementarse de estas dos formas:

- *Single-mode*: únicamente se implementan los circuitos y el *software* correspondiente a la nueva torre de protocolos de baja potencia. Empresas como STMicroelectronics, Nordic Semiconductor and Texas Instruments venden estos chips.
- *Dual-mode:* se integra en un chip, con el ya existente controlador clásico de Bluetooth, el nuevo Bluetooth LE. Empresas como Qualcomm-Atheros y Texas Instruments venden estos tipos de chips ya que el precio de implementar ambas tecnologías es negligible.

Los chips *single-mode* ofrecen, a bajo precio, chips de dimensiones pequeñas fáciles de integrar en dispositivos compactos, una implementación de la capa de transporte (OSI nivel 2) muy ligera que permite operaciones en modo *idle* a muy baja potencia, una fase de *discovery* muy simplificada y finalmente una transferencia de datos *point-to-multipoint* estable y segura con encriptación y opciones de ahorro de energía.

En general, los cambios introducidos en la versión 4.0 son aquellos necesarios para poder implementar el modo de alta velocidad y el modo de baja potencia, implementar GATT (*Generic Attribute Profile*) y los servicios SM (*Security Manager*) con encriptación AES (*Advanced Encryption Standard*) [13][14].

2.1.1.2.- Torre de protocolos Bluetooth LE

Los requisitos de baja potencia de la versión LE de Bluetooth estipulan gran diferencia entre la versión clásica y esta nueva versión. Como ya hemos mencionado anteriormente, los chips pueden seguir el modo *single y dual.* En la figura de abajo podremos observar los bloques principales de la torre del modo clásico (BR/EDR), del *dual-mode* y del *single-mode*.

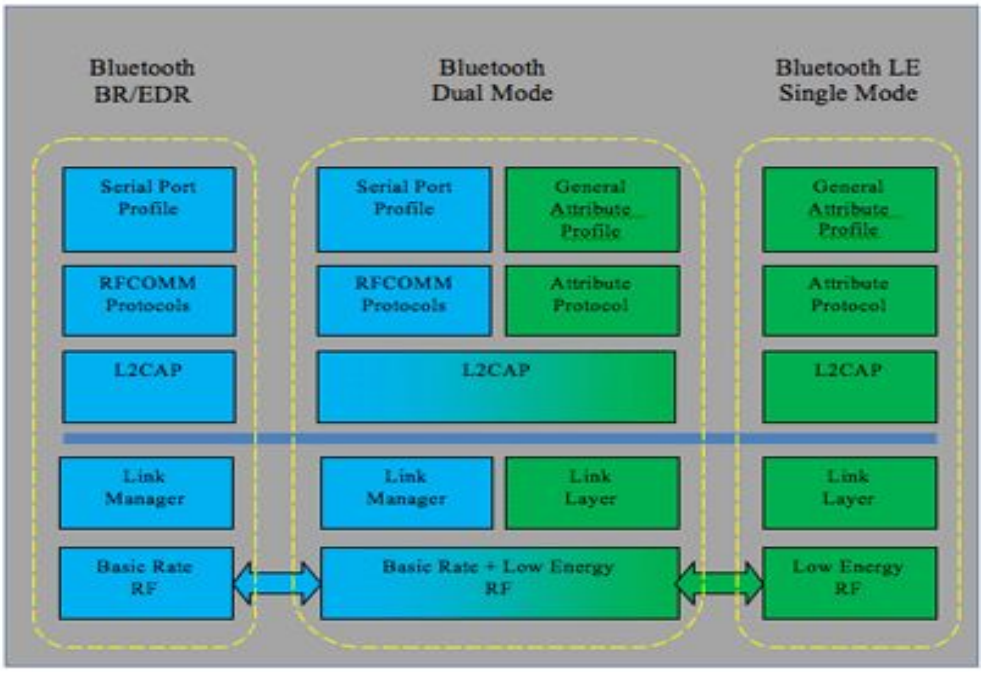


Figura 2. Torre de protocolos Bluetooth v4.0. Fuente:http://www.blueradios.com/hardware_LE4.0-S2.htm

Históricamente, la torre de Bluetooth estaba particionada en 2 piezas independientes: *controller y host*. El *controller* se encarga de ejecutar las partes más bajas de la torre de protocolos, las cuales son necesarias para manejar los paquetes de la capa física. El *controller* tiene que ser muy rápido y se ve limitado por la gran velocidad a la que tiene que realizar las

operaciones. Normalmente se implementa con un circuito integrado SoC (*System on a Chip*) el cual implementa la comunicación radio de Bluetooth.

La porción restante de la torre de protocolos formada por las capas superiores que incluyen *profiles y la API* de la aplicación están en manos del *host.* El *host* se abstrae del *hardware* y de las restricciones de tiempo, las cuales son mucho más relajadas. El *host* suele correr en el procesador junto a la aplicación del usuario. La comunicación entre el *controller* y el *host* está estandarizada mediante el *Hardware Controller Interface* (HCI) mediante UART (*Universal Asynchronous Receiver-Transmitter*), USB (*Universal Serial Bus*) o SDIO (*Secure Digital Input Output*) como interfaz física. Esta capa permite al *host y al controller* abstraerse el uno del otro permitiendo integrar chips de distintos fabricantes.

En la figura 3 vemos la implementación en detalle de la torre de protocolo en *single-mode* de Bluetooth LE. Podemos apreciar los componentes que hemos explicado anteriormente. A continuación, explicaremos las partes restantes de la torre.

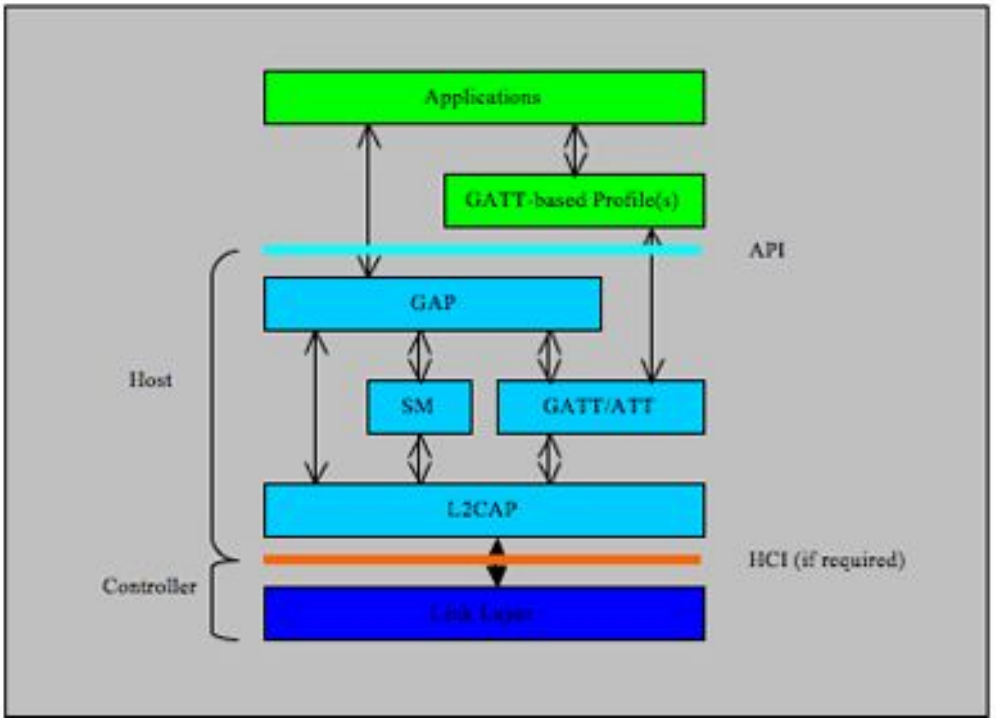


Figura 3: BLE single-mode protocol stack, http://www.blueradios.com/hardware_LE4.0-S2.htm

El *link layer (LL) controller* es responsable de la comunicación en la capa física (PHY). Administra las tramas recibidas y enviadas y se encarga de comunicarse con otros nodos teniendo en cuenta los parámetros de conexión y de gestión de flujo. También es el encargado de administrar los paquetes recibidos y enviados mientras el dispositivo se encuentra en modo

advertising o *scanning*. Además, el controlador LL se encarga de limitar el tiempo de intercambio de datos y de exposición. Si el filtrado está configurado, el LL mantiene una lista actualizada ("white list") de los dispositivos a los que se les permite establecer conexión e ignora a todos aquellos fuera de ésta.

El *logical link control and adaptation layer protocol* (L2CAP) proporciona servicios de datos a las capas superiores como la capa de seguridad o la capa de atributos que explicaremos más adelante. Es responsable del multiplexado de protocolos y de la segmentación de datos en pequeños paquetes para el *LL controller* hacia un sentido.También de de-multiplexar y de las operaciones de reensamblado en el otro sentido. El L2CAP es una interfaz de *backend* para el *generic access profile* (GAP) que define el método genérico relacionado con el descubrimiento de dispositivos BLE y con todos los aspectos relacionados con la conexión entre dispositivos. GAP proporcionado una interfaz para la aplicación con la que puede configurar diferentes modos de operación por ejemplo *advertising* o *scanning* y también iniciar, establecer y controlar la conexión entre dispositivos [4].

El *Security Manager (SM)* es responsable de la fase de emparejado (*pairing*) entre dispositivos y de la distribución de la clave. El *security manager protocol (SMP)* se define como el protocolo de comunicación entre las dos unidades SM de cada uno de los dispositivos que se comunican. El SM proporcionado funcionalidades criptográficas adicionales que pueden ser usadas opcionalmente por diferentes componentes de la torre de protocolos.

La arquitectura del SM usada en BLE está diseñada para minimizar los recursos necesarios en el esclavo, derivando todo el trabajo a los dispositivos maestro que suelen tenen más potencia. BLE utiliza el estándar de algoritmo criptográfico AES-128 y utiliza un mecanismo de *pairing* para distribuir las claves. El SM no únicamente proporciona un mecanismo para encriptar los datos sino también para firmarlos y autenticarlos.

Bluetooth 4.0 introduce un nuevo método de comunicación conocido como *attribute protocol (ATT)* el cual está optimizado para paquetes de tamaño pequeño utilizado comúnmente en Bluetooth Low Energy. El ATT permite a un servidor de atributos exponer un conjunto de atributos y sus correspondientes valores a los clientes. Estos atributos pueden ser descubiertos (primera fase), leídos y escritos por sus dispositivos emparejados.

El *generic attribute profile (GATT)* describe un marco de referencia para los servicios, el cual utiliza el ATT para descubrir servicios y para leer y escribir características en los dispositivos

enlazados con éste. El perfil de la aplicación define el conjunto de atributos que se emplearán para la comunicación y los permisos relacionados con estos atributos.

Para que la comunicación entre dos tecnologías Bluetooth de diferentes fabricantes sea factible no únicamente requiere un protocolo *wireless* estandarizado para la transferencia de bytes de información, sino también una representación de datos correcta, es decir, ambos dispositivos deben enviar y recibir datos usando el mismo formato mediante una interpretación común de los datos. GATT es un puente entre el protocolo de comunicación y la aplicación y sus funcionalidades. BLE utiliza de forma eficiente GATT basados en perfiles que minimizan el tamaño de los datos a intercambiar de esta forma reduce la media de consumo eléctrico. Aparte, los perfiles GATT pretenden ser simples y poder ser implementados con pocas líneas de código para así minimizar la memoria requerida [15].

2.1.1.3.- Ventajas de Bluetooth LE

Después de todo lo explicado, podemos apreciar que BLE abre la puerta a que dispositivos pequeños y baratos puedan comunicarse entre ellos sin la necesidad de baterías recargables. También se ha demostrado que el uso de perfiles GATT simplifica mucho exponer funcionalidades ayudando a los desarrolladores a crear nuevas tecnologías.

En la actualidad podemos encontrar chips BLE a bajos precios. Por ejemplo, TI presentó el chip CC2540 el cual posee un transmisor radio Bluetooth compatible con LE *single-mode* y con un procesador de arquitectura 8051. Tiene una lista extensa de periféricos y una memoria SRAM de 8 KB y una memoria flash reprogramable de 128/256 KB. Este chip viene con una torre implementada de BLE y con soporte al modo de comunicación maestro-esclavo [14].

Gracias a las ventajas explicadas anteriormente y la disponibilidad de chips de buena calidad y bajo precio, usaremos un chip BLE en el microcontrolador para que este pueda comunicarse con cualquier modelo de gama media/alta smartphone del mercado.

2.1.2.- Sistemas operativos: Android e iOS

Los smartphones, como la gran mayoría de sistemas informáticos de media/alta potencia, utilizan el sistema operativo para abstraerse del *hardware* que hay por debajo de estos y facilitar a los programas y al propio programador el uso de este *hardware*. Los sistemas operativos para móviles combinan las características comunes de los sistemas operativos de computadores con las características propias de estos, que son: una pantalla táctil, un sistema de radio, un chip de

Bluetooth, tecnología Wi-Fi, *Global Positioning System* (GPS), videocámara, reconocimiento de voz, *Near Field Communication (*NFZ*),* etc*.*

La gran mayoría de dispositivos móviles contienen 2 sistemas operativos integrados, uno encargado de las aplicaciones, servicios y todo lo relacionado con el usuario y otro de más bajo nivel y con una alta prioridad de ejecución encargado de las operaciones en tiempo real como por ejemplo de la recepción/emisión de radio, gestión de la memoria, *task scheduling* … A diferencia de los ordenadores de sobremesa el mercado de sistemas operativos para *smartphones* está monopolizado por dos grandes sistemas operativos (y sus versiones derivadas). Estos sistemas operativos son Android y iOS.

2.1.2.1.- Android

Android es un sistema operativo para móviles/tablets desarrollado por Google, basado en el *kernel* (núcleo del sistema operativo) de Linux y con una interfaz de usuario basada en la manipulación directa a través de una pantalla táctil. Mediante gestos con los dedos (por ejemplo: deslizamiento, pinza, toque suave, etc) y un teclado virtual el usuario puede interaccionar con los objetos de la pantalla. También podemos encontrar pequeñas modificaciones del sistema operativo como *Android TV*, *Android Auto* para coches, *Android Wear* para *smartwatches*,cada uno con una interacción con el usuario especializada.

El primer modelo comercial de Android fue lanzado en Septiembre de 2008 por el consorcio Open Handset Alliance. El sistema operativo Android cuenta con multitud de versiones y parches. La versión actual, en 2017, es la 7.0 *Nougat*. El código fuente de Android es *open source* eso hace que sea muy popular entre las empresas de tecnología que buscan un sistema operativo de bajo coste, personalizable y rápido de diseñar y por este motivo la gran mayoría de dispositivos Android como los *smartphones* vienen con la combinación de sistema operativo de un kernel de Android y una capa por encima propietaria de cada empresa fabricante.

Las aplicaciones o mejor conocidas como “apps” son ejecutables que extienden las funcionalidades del dispositivo. Normalmente se escriben en Java juntamente con *Android Software Development Kit (SDK)* que ayuda a los desarrolladores a implementar las aplicaciones. En el SDK de Android podemos encontrar entre otras muchas cosas un *debugger,* librerías de sistema / interfaz y un emulador basado en *QEMU.* Se puede combinar Java con otros lenguajes como por ejemplo C/C++ que requieren para comunicarse con la JVM un conjunto de *runtime libraries*. Actualmente Google ha dado soporte a la programación con *Go lang/Kotlin* pero todavía se encuentra en una fase inicial y con una API (Application Programming Interface) limitada.

En los primeros años de Android, el entorno de desarrollo principal (IDE) era Eclipse combinado con el *plugin* Android Development Tool (ADT) pero en Diciembre de 2014 Google sacó su propia plataforma de desarrollo, Android Studio, la cual cada año mejora con nuevas versiones. Otra plataforma de desarrollo digna de mencionar es *Native Development Kit (NDK)* para aplicaciones con extensiones C o C++.

Gracias a las herramientas mencionadas anteriormente hay un gran mercado de aplicaciones de terceros. *Google Play Store* es una aplicación de sistema de Android que permite descargar aplicaciones a través de los contenedores *APK (Android Application Package)*, instalarlas y actualizarlas. Es importante mencionar que *Play Store* no es la única aplicación de mercado, existen otros proveedores de aplicaciones como por ejemplo *Amazon Appstore, GetJar, F-Droid*, entre otros.

Para terminar, mencionar que los dispositivos Android cada vez cuentan con mayor potencia de cálculo aritmético y gráfico. Por este motivo Android incluye soporte para operaciones vectoriales y operaciones gráficas de alto rendimiento tanto para 2D como para 3D mediante la API de OpenGL ES (propiedad de OpenGL). OpenGL es una API multiplataforma de gráficos la cual especifica una interfaz software estándar para el hardware encargado del procesamiento de gráficos 3D. Su otro gran competidor es DirectX.

OpenGL ES es una especificación de OpenGL para dispositivos *embedded*. Android da soporte a varias versiones de la API OpenGL ES:

A. Android API 1 o superior: soporte para OpenGL ES 1.0 & 1.1
B. Android API 8 o superior: soporte para OpenGL ES 2.0
C. Android API 18 o superior: soporte para OpenGL ES 3.0*
D. Android API 21 o superior: soporte para OpenGL ES 3.1*

*Se debe puntualizar que para que un dispositivo Android soporte OpenGL ES 3.0 o superior el *pipeline* de la gráfica debe estar adaptado a esta versione.

2.1.2.2.- iOS

iPhone OS, mejor conocido como iOS, es el otro sistema operativo para dispositivos móviles mayoritario en el mercado por debajo de Android. Es propiedad de Apple Inc. y opera exclusivamente en el hardware de dispositivos de Apple. Lo podemos encontrar en los iPhones, iPads y iPod Touch.

Como en el caso de Android, el sistema operativo se fundamenta en una interfaz de usuario basada en la manipulación directa mediante gestos, más o menos complejos, con los dedos encima de la pantalla. Las versiones *major* de iOS se publican de forma anual, actualmente en 2017 encontramos la versión iOS 10.0 la cual la podremos encontrar en dispositivos iPhone 5 o superior.

El *kernel* the iOS es el *kernel XNU* de Darwin (sistema operativo *open-source* de Apple Inc. basado en los *cores* de NeXTSTEP, BSD y Mach). Desde el iOS 1.0 hasta el 3.1.3 se usó el kernel Darwin 9.0.0d1. En la actualidad, el *kernel* de iOS 10 está basado en la misma arquitectura Darwin pero usa la versión 16.

A nivel de desarrollo, podemos encontrar aplicaciones autorizadas de terceros a través de la aplicación *Apple's App Store*, muy parecida a *Play Store* de Android. Las aplicaciones nativas deben ser escritas en Objectives-C (con algunas partes escritas en C o C++) o en Swift y se compilan para ser ejecutadas en los sistemas operativos de iOS de 32 bits ARM o de 64 bits ARM (a partir del iPhone 5S). La IDE Xcode es la utilizada, comúnmente, para desarrollar en iOS. Una de las características de iOS es que comparte ciertos *frameworks* con el sistema operativo macOS de la misma compañía como por ejemplo *Core Foundation* o *Foundation Kit* aunque estas han sido sustituidas por la interfaz gráfica *Cocoa Touch* y *UIKit*.

El SDK de iOS fue publicado en 2008 y permitió a los desarrolladores independientes crear aplicaciones para iPhone y iPod Touch y testearlas en un simulador. Es importante destacar que para poder ejecutarlas en un dispositivo móvil y no en el simulador o si quieres publicarla en la *Apple Store* tienes que pagar una cuota de 99 $/año.

Como curiosidades, los dispositivos iOS usan una encriptación de la memoria a través del *Secure Enclave* que es un coprocesador encargado de generar números aleatorios. Cada vez que se usa una contraseña en el dispositivo, el contenido de la memoria es encriptado. Esto se hace a través de una unidad donde se encuentra el algoritmo AES 256 bits implementado en hardware por lo que es sumamente rápido el cual se encuentra directamente conectado a la RAM. A nivel de seguridad de red comentar que iOS soporta TLS (*Transport Layer Security*) versión 1.2 o superior. Adicionalmente, cuando el Wi-Fi está activado para evitar técnicas maliciosas como el *sniffing* de tráfico el propio sistema genera una dirección MAC aleatoria para evitar el rastreo de paquetes por MAC.

2.1.3.- Modbus

Modbus nace como un protocolo industrial para la comunicación entre dispositivos de automatización (PLC o PAC). Su finalidad inicial era la transferencia de datos a través de una capa serie, aunque a lo largo de los años se ha expandido para incluir otros protocolos en serie como TCP/IP y UDP. A continuación explicaremos las características más relevantes de este protocolo.

Modbus es un protocolo basado en solicitud-respuesta en una relación maestro-esclavo. Eso implica que el dispositivo maestro debe iniciar la comunicación y esperar la respuesta. Normalmente el maestro es una interfaz humano-máquina (HMI) o un sistema SCADA, y el esclavo es un sensor.

En su implementación inicial el protocolo no soportaba múltiples capas. Con el tiempo, para dar soporte a red, se separó el protocolo principal definido por la unidad de datos de protocolo (PDU) y la capa de red, definida por la unidad de datos de aplicación (ADU).

El formato de Modbus PDU está definido como un código de función seguido por un conjunto de datos asociados. El tamaño y el contenido de estos datos son definidos por el código. La PDU completa (código y datos) no puede exceder los 253 bytes. Cada código de función tiene un comportamiento específico que los esclavos pueden implementar de forma flexible dependiendo del comportamiento del esclavo [16].

Los datos disponibles por medio de Modbus son almacenados en cuatro bancos de datos o rangos de dirección que son almacenados localmente en los dispositivos esclavos. En estos bancos se define el tipo y los derechos de acceso de los datos contenidos. Los datos disponibles por Modbus generalmente son un subconjunto de la memoria principal del dispositivo. El maestro, debe solicitar acceso a estos datos a través de diversos códigos de función. El comportamiento de cada bloque se define en la siguiente tabla. Los bloques son completamente conceptuales, pueden existir como direcciones de memoria separadas pero también pueden solaparse y no tienen porqué existir los 4 bloques.

Bloque de memoria	Tipo de datos	Acceso de maestro	Acceso de esclavo
Salidas digitales o discretas	Booleano	Lectura / Escritura	Lectura / Escritura
Entradas Discretas	Booleano	Solo Lectura	Lectura / Escritura
Registros de Entrada	Palabra	Solo Lectura	Lectura / Escritura
Registros de Salida	Palabra	Lectura / Escritura	Lectura / Escritura

Tabla I. Bloques de modelo de Modbus

El espacio de direcciones es de 16 bits (65,536 direcciones indexables). La dirección inicial es la 0 hasta la 65,535. La diferencia entre las direcciones de memoria y los números de referencia es confusa, por ejemplo, el registro de entrada uno está en la dirección lógica cero. No se requiere que los rangos completos permitidos por la especificación sean implementados por un dispositivo. Aunque la especificación define los diferentes tipos de datos que existen en diferentes bloques y asigna un rango de dirección local para cada tipo, esto no se traduce necesariamente en un esquema intuitivo. En lugar de referir a un elemento como registro de entrada 14 en la dirección 13, se usa la nomenclatura de dirección 400,014, donde 4XXXXX indica que es un registro de salida, con un rango de direcciones de 400,001 a 465,536 y que el valor efectivo de la dirección es el 13. En la siguiente tabla podemos encontrar el prefijo con el bloque asignado.

Bloque de datos	Prefijo
Salidas digitales o discretas	0
Entradas Discretas	1
Registros de Entrada	3
Registros de Salida	4

Tabla 2. Prefijos de rango de datos

El estándar Modbus proporciona un modelo de datos relativamente simple que no incluye tipos de datos adicionales fuera de una palabra sin signo y valor de bit. Insuficiente para los requisitos de algunos sistemas. El módulo NI define un nuevo tipo para valores de datos grandes. Por ejemplo 400,001.2H indica que en el registro de salida hay una secuencia de dos caracteres donde el byte alto corresponde al primer carácter de la secuencia. Esto es necesario ya que Modbus debe conocer los límites exactos de la secuencia en lugar de buscar una longitud o delimitador nulo. Además de permitir el acceso a los datos que cruza un límite de registro, algunos maestros Modbus soportan las referencias para bits individuales. Esto se referencia de la siguiente forma, 400,001.01.

Los datos de coma flotante de precisión simple pueden ser transferidos fácilmente en Modbus al dividir los datos en dos registros. Ya que no está definido en el estándar, el *endianness* es variable. En caso de datos de red, se usa *big-endian* aunque en varios dispositivos se invierte el orden. Se debe consultar la configuración del sistema para conocer el *endianness* del dispositivo.

En general, no se recomienda modificar el protocolo, aunque se puede ajustar el comportamiento de éste. Algunos códigos de función son definidos, pero el estándar Modbus permite desarrollar códigos adicionales. Los códigos del 1 al 64, 73 al 99 y del 111 al 127 son reservados. Los códigos restantes, 65 al 72 y 100 al 110 son definidos por el usuario. Los códigos de función por encima de 127 son reservados para respuestas de excepción.

A nivel de red, Modbus es ideal para UDP. En lugar de requerir una ADU adicional o reutilizar una ADU existente, los paquetes de la PDU de Modbus se pueden enviar usando una API de UDP estándar y ser recibidos completos en el otro extremo. Aunque TCP es ventajoso para algunos protocolos gracias al sistem de confirmación integrado, Modbus realizar confirmación en la capa de aplicación [16].

2.1.4.- Conectividad a internet en los smartphones

Desde la primera generación de redes analógicas celulares, los móviles han sido capaces de transmitir datos a grandes distancias. En un origen, únicamente transportaban la voz codificada de forma simple pero a medida que nuevas necesidades surgían en el mercado, mensajería instantánea, multimedia, etc. estos móviles tuvieron que dar soporte al datagrama o datos en forma de paquetes, incrementar su ancho de banda, es decir, la cantidad de información por banda de frecuencia, reducir la latencia de transporte, soportar múltiples conexiones …

Todos estos cambios fueron introducidos en la segunda generación de móviles y los estándares tecnológicos que los implementaron fueron GSM, GPRS y EDGE. En las siguientes generaciones, actualmente nos encontramos entre la cuarta y la quinta generación, se han ido incorporando nuevas técnicas para mejorar el rendimiento y así poder enviar cada vez más cantidad de información, siempre utilizando una arquitectura / infraestructura, que no explicaremos en este trabajo, muy parecida. En los dos siguientes puntos explicaremos donde nos encontramos ahora a nivel de conectividad a Internet en *smartphones* y que nos espera en la próxima generación, 5G.

2.1.4.1.- 4ª Generación: IMT-Advanced

El estándar 4G define una serie de requisitos [17]:

- Estar basada en transmisión de paquetes y tener un ratio de transmisión de 1Gbit/s en baja movilidad y 100MBit/s en alta movilidad.
- Incrementar el nombre de usuarios por celda respecto a 3G. Con una eficiencia espectral de 15 bit/Hz.

- *Handovers* (se produce cuando un dispositivo celular se desplaza de una estación base, BTS, a otra y todas las acciones asociadas a esto) poco costosos y latencia inferior a 100ms.
- Otros [17].

Las tecnologías que cumplan estos requisitos pueden ser consideradas parte del estándar y reciben una distinción. Encontramos dos tecnologías que cumplen el estándar y son: 4G LTE Advanced del grupo 3GPP y 802.16m de IEEE mejor conocido como WiMAX. Para obtener este salto de rendimiento se usaron las siguientes técnicas físicas[17]:

- MIMO *(Multiple Inputs Multiple Outputs)* juntamente a OFDM *(Orthogonal Frequency-Division Multiple Access).* Estas dos técnicas combinadas mejoran la eficiencia espectral de la señal y también su robustez al ruido.
- *Frequency-domain statistical multiplexing.* Velocidad de bits variable dependiendo de la cantidad de subcanales ocupados.
- *Link adaptation.* Modulación adaptativa y códigos de corrección de errores.

El primer modelo de *smartphone* que incorporaba 4G fue el HTC Evo 4G. Utilizaba la infraestructura WiMAX y fue lanzado en EE.UU. en Junio de 2010 [18]. Uno de los requisitos para el correcto funcionamiento del proyecto es que los *smartphones* donde se ejecutarán las aplicaciones de control deben incorporan un *chipset* que implemente el hardware y el software necesario de 3G y/o 4G ya que parte de las operaciones de la aplicación requieren conectividad a Internet.

2.1.4.2.- 5ª Generación

La próxima generación, aunque no sea necesaria para nuestro proyecto ya que con las capacidades de la actual generación el proyecto se puede llevar a cabo, es interesante tenerla en cuenta ya que ofrece un gran abanico de posibilidades para próximos proyectos. La 5ª Generación está enfocada al Internet de las Cosas (IoT) y permitiría al microcontrolador dar un salto cualitativo y no tener que depender de Bluetooth para comunicarse ya que podría utilizar esta nueva red para hacerlo.

Las características distintivas de la 5ª generación serán [17]:

- Velocidades de transmisión de 1-10 Gbps con una latencia de *round trip time* en el plano de control de 10ms.

- Gran cantidad de dispositivos interconectados en celdas de tamaños muy pequeños para dar soporte al IoT.
- Disponibilidad de la red del 99.9 % y con una cobertura del 100% en cualquier región del mundo.
- Una reducción de energía del 90% y con un menor impacto a nivel energético y ambiental.
- Otros.

Las técnicas, entre otras muchas, que se van a emplear para llevar a cabo estos avances serán las siguientes [17] :

- Partición del plano de control y del plano de datos mediante *Software Designed Networks (SDN)* y *Heterogeneous Networks (HetNets)*.
- *Massive-MIMO*. Para mejorar el ancho de banda, la eficiencia espectral y permitir gran cantidad de dispositivos interconectados en región espaciales muy pequeñas.
- Reestructuración del protocolo MAC para reducir el *overhead* de comunicación.

Este proyecto se hubiera enfocado distinto si la quinta generación ya fuese una realidad pero tendremos que esperar un años más a poder beneficiarnos de ella.

2.2.- Conocimientos aplicados

2.2.1.- Integración de conocimientos

Este proyecto, a diferencia de otros proyectos de carácter más académico, pretende a través de múltiples tecnologías maduras dar solución a un problema. Para hacer esto posible es importante que estas tecnologías se integren en un único sistema. Una de las ventajas de los nuevos modelos de smartphone es que ya vienen preparados con el hardware y el firmware necesario para el uso de estas tecnologías.

En este proyecto se usarán las siguientes tecnologías. Algunas de ellas han sido aprendidas en el ámbito universitario y otras de forma autónoma.

- iOS, Xcode y swift/Objective-C
- Android, Android Studio/Eclipse IDE y Java/Android SDK
- Servidores HTTP Servlet (Java) y Node.js (Javascript)
- Bases de datos relacionales y SQL
- Modbus y Arquitectura ARM-32 bits
- Bluetooth Low Energy (Master-side y Slave-side)

- Librerías de soporte para Web-services en JSON-HTTP

- Edición gráfica con Photoshop
- Gestión de proyectos

2.2.2.- Análisis de alternativas

Antes de iniciar el proyecto, hubo una fase donde se plantearon posibles tecnologías. Es importante mencionar que existió una versión anterior a este proyecto, donde el microcontrolador se gobernaba a través de una aplicación de sobremesa. Esta versión no fue popular y acabó siendo un fracaso. Por este motivo era obligatorio darle otro enfoque. Se buscaba un sistema portable capaz de manipular la cámara frigorífica con un precio de producción ínfimo. Por este motivo se descartaron hardware especializados ya que obligaba al usuario a comprar este hardware para poder operar el microcontrolador. Esto provocaba que el coste total del producto se incrementara drásticamente y la ventaja de bajo precio respecto a la competencia se perdía.

La selección de Bluetooth Low Energy para la comunicación no fue casual. Ako había trabajado anteriormente con un Chipset de Bluetooth integrado en uno de sus microcontroladores por lo que ya se conocía la tecnología. La elección de la versión Low Energy fue casi algo obligatorio debido a las necesidades de bajo consumo eléctrico que tienen los microcontroladores. Con la integración de BLE en los nuevos modelos de *smartphones* se abrió un camino llano para este proyecto. Se podría integrar en una aplicación móvil el controlador a distancia sin la necesidad de coste adicional ya que en la actualidad gran proporción de la población cuenta con un *smartphone.*

A nivel de servidor, se plantearon otras tecnologías como simpleServer de Python o Casablanca de C++ pero debido a que ambas empresas dominaban Java y la creación de API en REST mediante las librerías Jersey y Spring nos decantamos por las últimas. En relación a la base de datos, nos decantamos por una base de datos tradicional basada en el modelo relación por columnas. Y para el envío de correos electrónicos se optó por el uso del servicio de Amazon de SMTP en vez del de Gmail por las ventajas del primero aunque éste fuera de pago.

En relación a las versiones API de sistema operativo para las que se ha programado (Target API SDK) se intentó dar soporte a versiones lo más antiguas posibles. En el caso de Android se da soporte a la versión 5.0 del sistema operativo (API 21) y para iOS damos soporte hasta la versión iOS 8.0.

3.- Desarrollo del proyecto

Debido a que la dimensionalidad de la aplicación es grande, ésta se ha dividido en diferentes tareas que se realizarán de forma paralela por los diferentes integrantes del equipo. Cada parte de la aplicación o módulo seguirá este patrón de realización: diseño gráfico del módulo, implementación del módulo, conexión del módulo con los otros módulos y finalmente comprobación del correcto funcionamiento del módulo. Encontraremos un apartado destinado exclusivamente al servidor de *sharing*, con el framework node.js y al de registro, a través de un Apache Tomcat. Todo el código de la aplicación que se mostrará en el documento es de la versión de iOS si no se indica lo contrario. Se hará una mención especial en este apartado a como se ha realizado el *porting* de iOS a Android. La aplicación ha sufrido variaciones a lo largo de su desarrollo, tanto a nivel de diseño como a nivel de funcionalidad, aunque el flujo de pantallas es muy parecido al inicial. Encontraremos el *wireframe* de la aplicación en los anexos del trabajo.

A nivel de clases podemos diferenciar dos grandes grupos debido a su comportamiento, inspirado en el diseño Modelo-Vista-Controlador donde el plano de control y el de datos está dividido. Encontraremos las clases de control, mayoritáriamente Singletons, que se encargan de realizar las operaciones de comunicación con ambos servidores. También de la comunicación con el microcontrolador a través de solicitudes codificadas y decodificadas de Modbus a cadenas de bytes. Y finalmente, de comunicar estos datos a las clases de la interfaz gráfica. Por otro lado, encontramos las clases de UI, especializadas en mostrar al usuario los datos de forma comprensible para el ser humano y de interactuar con estos datos. Por ejemplo, modificar su valor, visualizar los datos en gráficos y diagramas, etc. Esta división de planos, nos ha permitido modularizar mejor la aplicación y trabajar de forma paralela en los diferentes módulos sin conflictos entre sí ya que mientras que un programador se encargaba , por ejemplo, de crear el comportamiento visual de los parámetro, el otro se encargaba de programar las solicitudes y las respuestas para que el microcontrolador enviase estos parámetros y pudieran ser modificados. Empecemos con las explicaciones detalladas de los módulos.

3.1.- Login

3.1.1.- Funcionalidades

El módulo de login ofrece a la aplicación la funcionalidad de autenticación. Es decir, permite, a través de un servidor externo (en este caso el servidor de Tomcat), permitir o denegar el acceso a los usuarios de la aplicación. Todo usuario de la aplicación debe registrarse a través de la aplicación (no existe otra forma de registrarse) y crear una cuenta en nuestro servidor. Los usuarios de la aplicación gratuita Fit deben proporcionar únicamente un correo electrónico que será validado mediante un código de confirmación para poder acceder a ésta. Los usuarios de la aplicación de instalador Tool deben proporcionar información adicional como son el nombre, la empresa y un teléfono de contacto. La aplicación de la posibilidad al usuario de cambiar la contraseña en caso de haberla olvidado. Todos los datos de los usuarios serán almacenados en una base de datos SQL orientada a fila. Se usará un Web Service de Amazon (Amazon SES) para enviar los correos electrónicos para la confirmación y recuperación de cuenta.

3.1.2.- Diseño gráfico

El diseño del login es simple e intuitivo. La primera pantalla que encontraremos en la aplicación será una *Splash Screen* o pantalla de carga. En esta pantalla encontraremos el logo de AKO y su utilidad principal es permitir a la aplicación cargar todo lo necesario para su funcionamiento sin que el usuario sea advertido. En la siguiente pantalla encontraremos el login. El usuario que quiera entrar en la aplicación deberá introducir el correo electrónico de registro y su contraseña. La aplicación se encargará de gestionar tokens de sesión con el servidor para validar temporalmente a un usuario. Un usuario una vez conectado será capaz de indicar a la aplicación que mantenga su sesión abierta y no tenga que escribir constantemente sus credenciales para autenticarse. En caso de no recordar la contraseña, el usuario puede modificarla a través de un menú de diálogos y un código de confirmación que se le enviará a su correo.

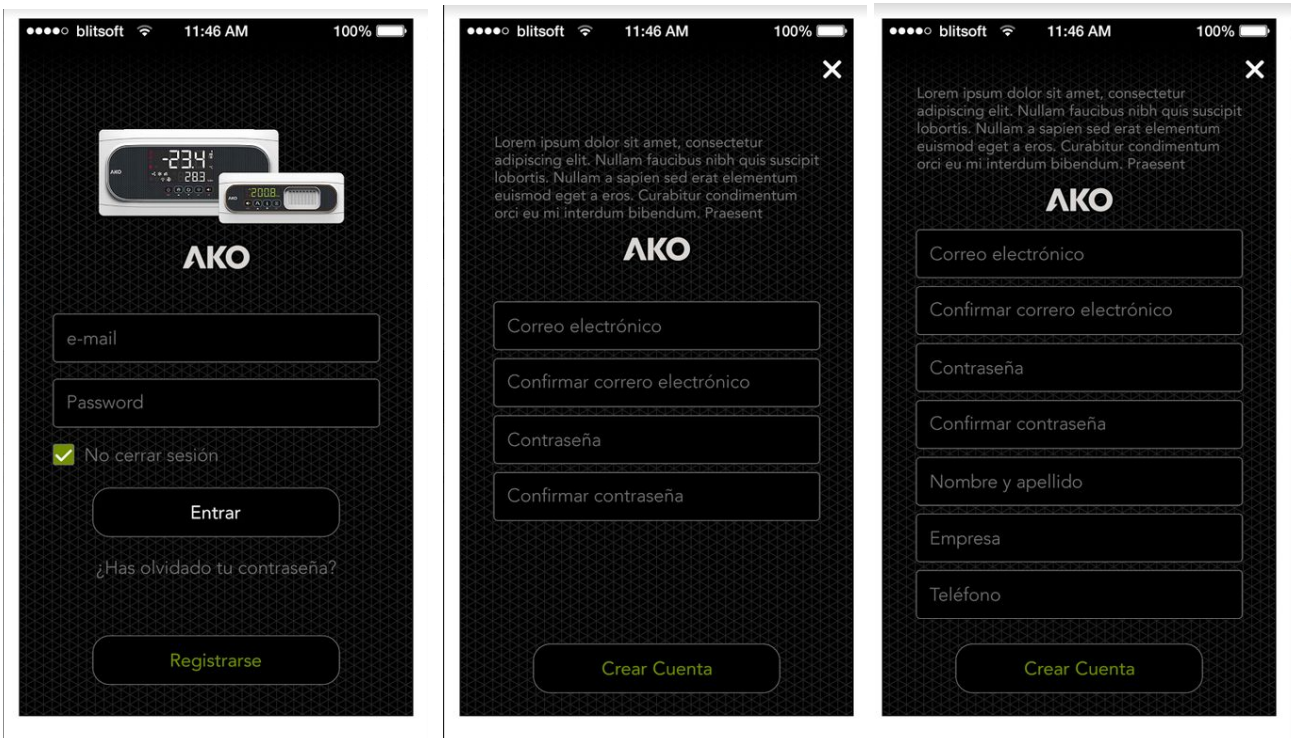


Figura 4. Pantallas actualizadas de Login para las dos versiones de la aplicación

3.1.3.- Implementación

A nivel de implementación las clases gráficas no tienen ningún tipo de dificultad, son un conjunto de campos de texto con un par de funcionalidades interesantes añadidas. Como saltar saltar al campo de texto inferior una vez escrito el superior o un desplazamiento vertical para evitar que el teclado bloquee la visión al usuario de los campos. Además, se han añadido pequeñas comprobaciones de seguridad tanto en el lado del cliente como en el servidor.

La clase que gestiona la comunicación con el servidor y que se encarga de organizar las peticiones de autenticación, registro y cambio de contraseña es LoginConnection. La comunicación con el servidor de registro es a través del protocolo HTTP mediante JSON. Se usa un *Web Service* en el servidor para interceptar las peticiones POST con los datos del registro y se responde a estas con un confirmación en caso positivo o un código con el error en caso negativo. Para gestionar las peticiones en iOS se usa la clase *URLRequest* de la librería Foundation que es *core*. En cambio en Android para simplificar el proceso ya que la API es un poco más rebuscada hemos usado la librería *Volley* que podréis encontrar en los anexos.

```
private func makeHTTPPostRequest(path: String, body: NSDictionary, onCompletion: @escaping ServiceResponse) {
    var request = URLRequest(url: URL(string: path)!)
    request.httpMethod = "POST"
    request.setValue("application/json; charset=utf-8", forHTTPHeaderField: "Content-Type")

    request.httpBody = try? JSONSerialization.data(withJSONObject: body)

    let task = URLSession.shared.dataTask(with: request, completionHandler: {data, response, error -> Void in
        var json = NSDictionary()
        if error == nil {
            json = (try? JSONSerialization.jsonObject(with: data!)) as! NSDictionary
        }
        OperationQueue.main.addOperation {
            onCompletion(json, error)
        }

    })
    task.resume()
}
```

Figura 5. La función makeHTTPostRequest de iOS usada para comunicarse con el servidor.

3.2.- Equipos

3.2.1.- Funcionalidades

Una vez superado el login encontraremos la pantalla de gestión de equipos. Esta pantalla está formada por dos secciones, la pantalla en sí que ahora explicaremos y un menú lateral desplegable que permite al usuario acceder a la gestión de equipos conectados por Bluetooth, a un catálogo de equipos, a la gestión de cuentas, desconexión de la cuenta actual e información general de la aplicación. Se ha añadido el acceso al modo share que explicaremos más adelante en esta sección. Se ha eliminado la funcionalidad de backup ya que no se le encontraba ninguna utilidad y la de Reset de la aplicación.

La pantalla de gestión de equipos es algo compleja. En esta pantalla encontraremos una lista actualizada de los equipos cercanos que se encuentren haciendo *advertising* por Bluetooth pero también encontraremos listados todos aquellos equipos que alguna vez se conectaron por Bluetooth con la aplicación. En un principio, los datos almacenados en los registros de cada dispositivo serán copiados a la aplicación y podrán ser consultados *offline*. Otra funcionalidad es el acceso a la mochila de los equipos. La mochila es como se le denomina al conjunto hardware formado por el chipset de Bluetooth Low Energy, a los registros de datos y al firmware del microcontrolador. Más adelante explicaremos que controla la mochila y que funcionalidades tiene. Las pantallas de sincronización de datos con el server han sido eliminadas, ya que en un principio la idea era almacenar los datos tanto en el servidor como una copia local en el dispositivo móvil pero esta idea fue suspendida ya que la gestión de estos datos era una tarea compleja y que no aportaba mucho al proyecto. Tanto en la pantalla de gestión de equipos como en la opción de

catálogo, si presionamos un dispositivo con Bluetooth activado nos conectaremos a éste y accederemos a su pantalla de emulación.

3.2.2.- Diseño gráfico

La interfaz de este menú es sencilla y es importante entender la simbología de los diferentes colores para entender su significado. El menú lateral se despliega pulsando el icono de tres rayas horizontales. En este menú encontraremos las opciones de : equipos, catálogo, conexión remota, gestión de cuenta, cerrar sesión y acerca de. La lista, que podemos observar en la imagen de debajo, está divida en dos: equipos nuevos y equipos ya registrados. Los equipos registrados son aquellos en los que nos hemos conectado y hemos podido descargarnos parte de su memoria (no tiene porque estar actualizada). El icono de la batería nos indica la batería útil del equipo. El icono de alarma nos indica una previsualización de la cantidad de alarmas activas en ese equipo (para obtener más detalles de las alarmas nos tendremos que conectar). El icono verde o rojo nos indica si el equipo es accesible o no. En el wireframe de los anexos podemos encontrar toda la información adicional de esta pantalla.

Figura 6. Pantalla principal de gestión de equipos a través de BLE

3.2.3.- Implementación

La pantalla de equipos donde se produce el escaneo de los dispositivos tiene su complejidad a nivel de código ya que debe encargarse de configurar el Bluetooth LE (con códigos distintos en ambas versiones de sistema operativo) para así poder localizar los dispositivos. Será el la siguiente pantalla, la de emulación, donde el dispositivo se enlazará con la mochila conectada al microcontrolador y se intercambiará información a través de los servicios GATT de escritura y de notificación. La fase de enlace se conoce como *pairing*. Algunos servicios requieren autenticación por lo que es necesario un PIN. Este PIN puede ser modificado a través de la aplicación.

La clase BluetoothConnection, tanto en iOS como en Android, es la encargada de gestionar la conexión Bluetooth Low Energy con el microcontrolador. Ambas versiones de sistema operativo ofrecen una API para la gestión de BLE. En el caso de iOS la interfaz es mucho más simple pero a su vez limitada. En el caso de Android, la documentación relacionada con la API [19] es aceptable pero algunos detalles como por ejemplo la intercepción del mensaje de solicitud de BLE Pin (muy común en aplicaciones comerciales) no están bien documentados. Las fases del proceso de conexión en Android son las siguientes:

Activar Bluetooth y la localización GPS (es un bug de la API v21 o superior). Escanear todos los dispositivos que hacen *advertising.* Usaremos la función:

```
mBluetoothAdapter.startLeScan(mLeScanCallback);
```

Esta función nos devolverá una callback con este formato:

```
onLeScan(final BluetoothDevice device, int rssi, final byte[] scanRecord)
```

Para identificar los mensajes de los dispositivos AKO lo que hacemos es filtrar todos aquellos paquetes que su nombre de dispositivo no sea M_BLE y que el tamaño del campo *Manufacturer Data* no sea de 23 bytes. El campo de *Manufacturer Data* se usa adicionalmente para transmitir información como por ejemplo el submodelo, el número de alarmss activas en el dispositivo, la batería de la mochila … entre otros.

Una vez identificado el dispositivo al que queremos conectarnos (el dispositivo se identifica por la dirección MAC) nos conectaremos a él. Para conectarnos a un dispositivo con la capa de autenticación activada necesitaremos primero emparejarnos con él a través del PIN del dispositivo. Para este proceso lo que haremos primero será comprobar si el dispositivo se

encuentra *Bonded* (después de la fase de *Pairing* ambos dispositivos almacenan las claves utilizadas para la comunicación encriptada y facilitan la próxima comunicación). Si el dispositivo no se encuentra *Bonded* tendremos que pasar por la fase de *Pairing y Bonding* antes de seguir con el proceso. Todo el código necesario para esta fase se encuentra en los anexos. Para conectarnos:

```
mBluetoothGatt = dispositivo.connectGatt(mContext, false, mGattCallback);
```

La callback de esta función, mGattCallback, nos indicará si se establecido la comunicación y qué servicios implementa el dispositivo al que nos acabamos de conectar. Cada servicio está compuesta por una o más características GATT y cada una de estas tiene una funcionalidad concreta. En nuestro caso encontraremos una característica de escritura y una de suscripción de lectura (cada vez que se produzca un cambio en la característica nos avisará y podremos leer de esta). Es muy importante, a la hora de subscribirnos, activar los descriptores de suscripción antes de solicitar la suscripción o se nos denegará el acceso.

A partir de este momento, el proceso de comunicación es simple. Si queremos modificar un parámetro del dispositivo sólo tenemos que escribir en la característica de escritura a través de la trama Modbus y esperar a que este nos devuelva una *callback* de que el proceso se ha realizado correctamente (puede fallar si la trama no existe o el CRC es incorrecto). Y, en el caso de lectura, tenemos que escribir en la característica de suscripción de lectura y esperar a que el dispositivo nos advierta de que el dato está listo para leer. Es muy importante que al final el proceso de comunicación con el dispositivo se cierre la comunicación a través de:

```
mBluetoothGatt.disconnect();
mBluetoothGatt.close();
```

Si se realiza únicamente el *disconnect()* el dispositivo se queda esperando a que se reabra la conexión en modo baja potencia. Esto provoca que el dispositivo se quede bloqueado.

3.3.- Emulación

3.3.1.- Funcionalidades

La pantalla de emulación, como su nombre indica, pretende emular la pantalla LED que todo controlador lleva incorporada. Estas pantallas LED difieren dependiendo del modelo, en la de la imagen inferior podemos observar uno de los antiguos diseños del dispositivo AD1. Podemos observar que ésta muestra la temperatura actual de la cámara con sus respectivas unidades de temperatura, la humedad, las alarmas activas, la batería disponible, el estado de los ventiladores, etc. Podemos apreciar que en la parte inferior hay una serie de iconos, todos estos iconos pueden

ser pulsados. Las funcionalidades de estos, de izquierda a derecha, son reiniciar el microcontrolador de control, activar el modo desescarche, activar las funcionalidades auxiliares, activar la iluminación de la cámara y silenciar las alarmas acústicas. Es importante destacar que tanto los LEDS como los botones inferiores a los LED dependen mucho de cada dispositivo. Podemos encontrar las pantallas de emulación de los diferentes dispositivos en los anexos. En la esquina superior derecha encontramos dos iconos que nos permiten acceder a las funcionalidades de Share y datos de la mochila. Los botones situados en la parte inferior de la pantalla son de navegación exceptuando en de refrescar los datos (que en las nuevas versiones ha sido sustituido por la funcionalidad de Share) y el de Set Point que permite regular la temperatura máxima/mínima de la cámara.

3.3.2.- Diseño gráfico

El diseño de esta pantalla pretendía parecerse lo más posible a las pantallas de los dispositivos. Por este motivo la pantalla superior, tanto LEDs como botones, es una copia exacta de la pantalla que encontraremos en el dispositivo. En la parte inferior primero encontraremos el valor actual de la temperatura máxima/mínima y también una etiqueta rotativa indicando el valor de las distintas alarmas activadas. Los botones de navegación principal se han situado en la parte inferior y de un color distinto ya que son comunes para todos los dispositivos. Son funcionalidades no asociadas a los diferentes modelos. Y encima de éstos, las funcionalidades particulares de cada dispositivo.

3.3.3.- Implementación

A nivel de implementación todas las pantallas se generan a partir de una clase base con el nombre de Equipo Base para agrupar funcionalidades compartidas y evitar la duplicación de código. Una de las gracias de la aplicación es la generalización que se ha hecho del código. Todo lo relacionado con los diferentes modelos de dispositivos, como por ejemplo los parámetros, ha sido generalizado a través de ficheros de representación JSON. A través de varios ficheros JSON se puede representar un dispositivo. Esto nos permite hacer el código transparente a los dispositivos y de esta forma, en un futuro, si se decide añadir nuevos dispositivos estos pueden ser representados a través de sus correspondientes JSON. Los formatos de cada JSON están incluídos en los anexos.

A nivel gráfico no ha sido posible crear una interfaz de traducción de JSON a elemento en pantalla ya que requería mucho tiempo por lo que se ha optado por crear un *layout* gráfico para cada dispositivo con algunos elementos gráficos compartidos como por ejemplo la barra superior de estado (*Status bar / Action Bar*). Esta pantalla a nivel de código no es muy compleja ya que se encarga únicamente de pedir a través de Bluetooth el estado actual de los LEDs y de las alarmas y gracias a las diferentes clases de flujo de control.

Figura 7. Pantalla de emulación de dispositivo

3.3.- Share

3.3.1.- Funcionalidades

La funcionalidad de Share de la aplicación es una de las más interesantes después de la comunicación con el microcontrolador. Esta funcionalidad permite a un usuario operar el microcontrolador a distancia como si éste estuviera conectado directamente. Para ello necesitaremos dos usuarios, uno que esté conectado con el dispositivo a través de Bluetooth LE y que los dos dispositivos tengan acceso a Internet (ya sea a través de Wi-Fi o de la red 3G/4G). El dispositivo móvil conectado a través de Bluetooth, esclavo, tiene el papel de *bridge*. Este dispositivo únicamente se encarga de comunicarse, mediante Internet, con la aplicación maestra y de enviar/recibir datos entre la aplicación maestra y el dispositivo. Para iniciar esta comunicación primero se establece una sala en el servidor remoto, con el framework Node.js, entre la aplicación maestra y el slave. Una vez establecido el *handshake* entre estos dispositivos, el dispositivo maestro se encarga de enviar peticiones a través de las clases de datos al dispositivo esclavo que a su vez se encarga de pedir estos datos al controlador frigorífico. La forma de enviar datos es a través de esquemas JSON con los que se conceptualizan todas las peticiones y los datos. Estos envíos se hacen a través de la API Socket.io que añadi un nivel superior de abstracción al protocolo HTTP. Es importante que ambos dispositivos tengan acceso a Internet. La velocidad de comunicación no ha de ser elevada ya que el cuello de botella es la comunicación Bluetooth. La aplicación gráfica se encarga de gestionar la conexión con el servidor de share de forma transparente al usuario.

3.3.2.- Diseño gráfico

La funcionalidad, como ya hemos mencionado antes, requiere que dos aplicaciones tengan acceso a Internet y estén conectadas a la misma sala del servidor Node. Una debe estar en modo *Share Master* y la otra en modo *Share Slave*. En modo maestro solo se puede acceder a través de la versión *Tool* de la app. En el menú lateral de la pantalla de equipos encontramos la opción. Tendremos que introducir el identificador de la sala a la que queremos conectarnos. En este caso, el identificador es el correo electrónico del usuario que esté compartiendo sus datos en modo esclavo. Una vez conectados esperaremos a que el usuario esclavo se conecte. Para compartir el dispositivo en modo esclavo se puede hacer tanto en la versión *Tool* como en la versión *Fit* de la aplicación. Una vez nos conectemos al dispositivo con Bluetooth accederemos a la pantalla de emulación. En la pantalla de emulación encontramos el icono de Bluetooth. Al presionarlo, el dispositivo pasará a modo compartiendo. En este modo el dispositivo se quedará bloqueado haciendo de puente entre el usuario maestro y el dispositivo. El usuario maestro, una

vez conectado al usuario esclavo, entrará en modo *Share Master* y su aplicación cambiará el tema de fondo a rojo y se comportará como si estuviera conectado directamente con Bluetooth pero con todos los privilegios que tiene la aplicación en versión *Tool*. Para desconectarse solo debe presionar otra vez el icono de Bluetooth o la barra superior con el texto “Desconectar”.

3.3.3.- Implementación

Respecto a la parte visual, el modo Share se ha podido simplificar mucho gracias a la herencia de la Action Bar. El *feedback* visual del modo Share, rojo, se consigue en todas las pantallas gracias a que la Action Bar, tanto en Android como en iOS, es customizada. Esta barra tiene asociada una clase que se encarga de gestionar el estilo dependiendo si se encuentra en modo compartir o no. La gestión de las ventanas que se necesitan para establecer el modo Share es complejo ya que se ha pretendido usar los *popUps* por defecto de ambas distribución y esto ha generado un gran *overhead* de código. Por otro lado, la automatización del modo Share, es decir, que el maestro una vez conectado al dispositivo sea enviado directamente a su pantalla de emulación también ha resultado laborioso.

Respecto a la parte de datos, el *pipeline* del proceso de obtención de datos permite que las clases de gestión de datos como ModelData no deban saber la procedencia de los datos. La clase ProxyConnection es la encargada de gestionar el origen de los datos y de identificar a qué clase debe pedir éstos. Si el dispositivo se encuentra conectado a un dispositivo estos datos se pedirán a la clase BluetoothConnection con la ayuda de ModbusProtocolHelper y si el dispositivo se encuentra conectado a otro a través del servidor HTTP Node será la clase ShareConnection la encargada de hacer peticiones al servidor para obtener los datos almacenados en la otra aplicación.

Toda la comunicación con el servidor en el cliente se realiza a través de la librería Socket.IO-client [20]. Esta librería, de licencia abierta del MIT (Massachusetts Institute of Technology), se encuentra soportada en múltiples plataformas incluyendo iOS y Android. Las diferencias entre las diferentes plataformas son mínimas aunque tangibles. Socket.io-server proporciona una API en javascript para el lado del servidor. Las siguientes dos figuras son extractos del código de iOS necesario para enviar/recibir peticiones.

```
private func receiveData(data_type: String, data: Any) {
    if let req = requestQueue.first() as? RequestGeneral {
        if data_type == type(of: req).toString(){
            //Decodifica los datos
            let myString = data as? String
            let myData = myString?.data(using: String.Encoding.utf8)
            let JSONObj = try? JSONSerialization.jsonObject(with: myData!, options: [])
            req.fillRequestWith(json: JSONObj as! NSDictionary)
            ProxyConnection.sharedInstance.receiveData(data: req)
            _ = requestQueue.dequeue()
            emitNextRequest()
        }
        else {
            emitNextRequest()
        }
    }
}

private func receiveRequest(data_type: String, data: Any) {
    if let req = getRequestFrom(string: data_type) {
        req.device = ModelData.sharedInstance.modelo_dispositivo.toString
        let myString = data as? String
        let myData = myString?.data(using: String.Encoding.utf8)
        let JSONObj = try? JSONSerialization.jsonObject(with: myData!, options: [])
        req.fillRequestWith(json: JSONObj as! NSDictionary)
        ProxyConnection.sharedInstance.sendRequest(request: req)
    }
}
```

Figura 8. Funciones de recepción de datos de ShareConnection.swift

```
func sendData(data: RequestGeneral) {
    let JSONObj = data.requestToJSON()
    let myData = try? JSONSerialization.data(withJSONObject: JSONObj, options: [])
    let myString = String.init(data: myData!, encoding: String.Encoding.utf8)
    socket?.emit("senddata", user, "data", type(of: data).toString(), myString!)
}

func sendRequest(request: RequestGeneral) {
    requestQueue.enqueue(obj: request)
    if(requestQueue.size() == 1) {
        emitNextRequest()
    }
}

private func emitNextRequest() {
    if(requestQueue.size() != 0) {
        let request = requestQueue.first() as! RequestGeneral
        let JSONObj = request.requestToJSON()
        let myData = try? JSONSerialization.data(withJSONObject: JSONObj, options: [])
        let myString = String.init(data: myData!, encoding: String.Encoding.utf8)
        socket?.emit("senddata", user, "request", type(of: request).toString(), myString!)
    }
}
```

Figura 9. Funciones de envío de datos de ShareConnection.swift

Es importante entender que todos los datos transferidos a través del servidor de Share han de poder traducirse de sus respectivas clases a un representación basada en JSON y viceversa. Todas estas traducciones y el código completo de ShareConnection se encuentra en los anexos de sus respectivas versiones de iOS y Android.

3.4.- Parámetros

3.4.1.- Funcionalidades

La funcionalidad de parámetros, como su nombre indica, permite modificar el estado de los parámetros del microcontrolador y por tanto, su funcionamiento interno. Los parámetros permiten configurar multitud de eventos como por ejemplo: el tiempo de espera antes de parar la cámara una vez la puerta está abierta, la frecuencia de desescarche, temperatura de parada de los ventiladores o las unidades de temperatura (ºC / ºF). Una de las particularidades de los parámetros es que tienen dependencias entre sí, es decir, en algunos casos hay parámetros que solo se pueden modificar si existe otro activado o en otros casos los valores máximos y mínimos dependen de otros parámetros.

La aplicación nos proporcionará las herramientas necesarias para modificar los parámetros del dispositivo que se encuentra conectado a nosotros a través de Bluetooth o del modo Share. También se encarga, de forma transparente al usuario, de filtrar los parámetros y de mostrar sólo aquellos que son visibles en la configuración actual del microcontrolador y de mostrar las opciones válidas dependiendo del estado de otros parámetros.

3.4.2.- Diseño gráfico

Visualmente la interfaz ha sufrido cambios desde su versión original. La primera pantalla que encontramos en el proceso de modificación de parámetros es una lista dinámica dividida por categorías de parámetros. Cada categoría es expandible y dentro de cada categoría encontraremos una celda por cada parámetro y cada celda consta del nombre del parámetro, una descripción del parámetro y su correspondiente valor. Es importante saber que los parámetros se han representado de forma conceptual como 5 tipos básicos: enumeración (conjunto de opciones predeterminadas), string (cadena de 16 bytes), binario, coma flotante simple y entero. Una vez seleccionado un parámetro, accederemos a su pantalla de modificación.

Existen 5 pantallas de modificación dependiendo del tipo básico del parámetro. Si el parámetro es de sólo lectura, la pantalla de modificación toma un tema más oscuro y las opciones de modificación y de envío quedan deshabilitadas. Si el parámetro es de lectura y escritura puede ser modificado. Para el elemento binario usamos un *switch*, para los elementos string, float y entero un campo de texto y para el elemento enumeration usamos un selector en forma de rodillo.

3.4.3.- Implementación

Cada modelo de controlador tiene asociado un documento JSON, por ejemplo para el modelo AD_X el fichero correspondiente es parameters_X. Un parámetro está formado por los siguientes valores: nombre, tipo básico, código, descripción, descripción abreviada, valor por defecto, valor mínimo, valor máximo, permisos, tipo p (parámetros de arranque que no pueden ser modificados de forma manual), valores (en el caso que sea de tipo enumeración), valor dependiente , tipo de dependencia y parámetros del que depende. Es indispensable que se rellene de forma manual este fichero para cada dispositivo así de forma automática el programa convertirá este modelo conceptual en datos que la aplicación podrá utilizar.

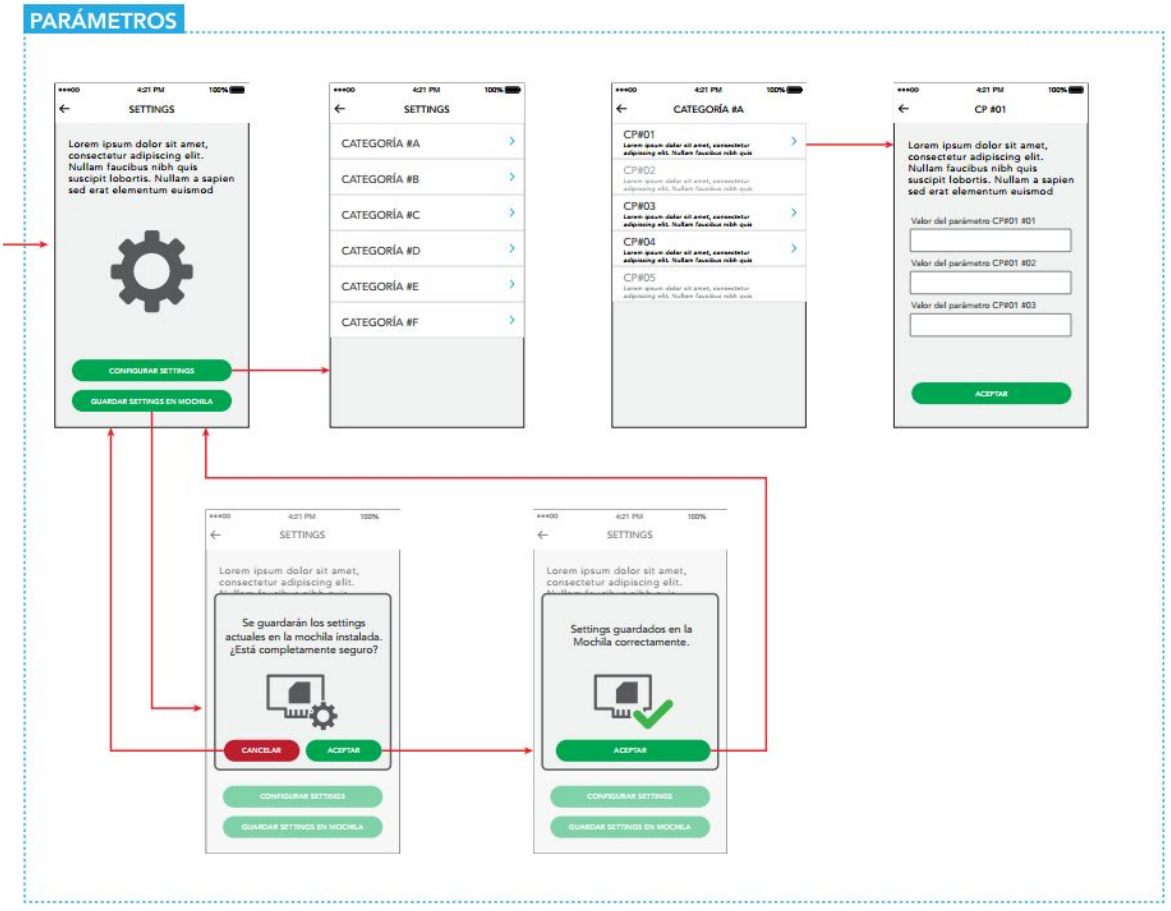


Figura 10. Diagrama de secuencia de las pantallas de parámetros

Para implementar las diferentes pantallas de modificación de parámetros se ha utilizado una superclase, *ModificarParametro*, que se encarga de gestionar la acción de envío del valor del parámetro por Bluetooth y del aspecto visual de las pantallas. Después se han creado 5 subclases, una para cada tipo de parámetro, que se encargan de gestionar la entrada del valor en el parámetro dependiendo del tipo de éste.

3.5.- Configuraciones

3.5.1.- Funcionalidades

Cada dispositivo, como hemos explicado en el punto 3.4, tiene asociado una serie de dispositivos. Una configuración es simplemente un subconjunto de los parámetros de un dispositivo que se guarda en forma de elemento atómico y que se puede transferir entre cuentas y a su vez enviar al dispositivo. Existen dos tipos de configuraciones, las normales y las que son de puesta en marcha. Una configuración de puesta en marcha es aquella que contiene parámetros de puesta en marcha. Los parámetros de puesta en marcha no se pueden modificar a través del menú de parámetros. Sólo se puede modificar a través de un proceso de arranque que es gestionado a través del menú de configuraciones. Para crear una configuración de arranque primero debemos acceder al menú de emulación del dispositivo y luego al menú de configuración. Una vez aquí, tendremos que indicar a la aplicación que queremos crear una nueva configuración de puesta en marcha. Después saltaremos a la pantalla *wizard* que es simplemente una serie de preguntas que al finalizarlas nos crean automáticamente una configuración de arranque con los parámetro de puesta en marcha en el valor adecuado respecto a las respuestas del *wizard*.

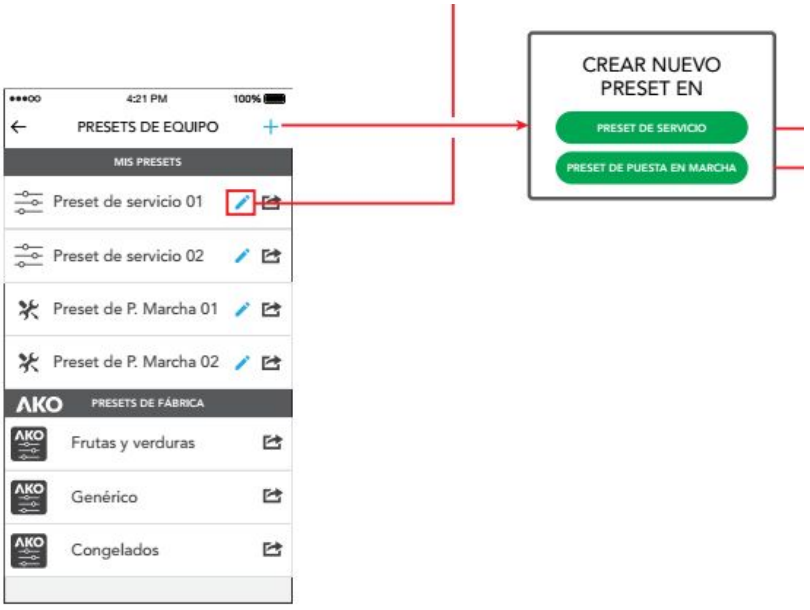


Figura 11. Listado de configuraciones

Las configuraciones una vez creadas pueden ser modificadas. Los parámetros normales pueden ser añadidos o eliminados a través de un menú parecido al de parámetros. Los parámetros de puesta en marcha también se pueden modificar pero este proceso se lleva a cabo pasando otra

vez por el *wizard*. Una configuración puede ser creada desde cero, copiada de otra configuración o creada como copia del estado actual del microcontrolador.

3.5.2.- Diseño gráfico

La pantalla inicial es un listado de configuraciones. El botón en forma de cruz permite crear una nueva configuración. Luego tenemos un listado de nuestras configuraciones. En la figura de arriba podemos observar que existe otra categoría dentro de la lista que es configuraciones de fábrica. Esta categoría ha sido eliminada en las versiones actuales de la aplicación. Una configuración se guarda en memoria interna y puede ser eliminada mediante el gesto de *swipe* de la celda y aceptando el diálogo de confirmación. El botón de enviar, oculto si el dispositivo no se encuentra en modo compartir o conectado a Bluetooth, nos permite enviar toda la configuración al dispositivo. Es una operación lenta. Si intentamos crear una nueva configuración se nos abrirá una pantalla con un menú de selección. Si estamos conectados a la mochila podemos acceder a la opción de copia desde mochila. En esta pantalla se nos mostrarán todos los parámetros de la mochila y podremos seleccionar aquellos que nos interesa. Los parámetros de arranque se han de seleccionar todos o ninguno y destacan por la tuerca al lado de su nombre. La otras dos opciones son copia desde configuración o crear una nueva. En la lista de la pantalla inicial podemos clicar una celda para modificar su configuración. Un menú muy parecido al de parámetros nos aparecerá y podremos añadir o quitar parámetros.

3.5.3.- Implementación

A nivel de código, la gestión de las configuraciones es un poco costosa. El código de esta parte del proyecto es el menos limpio y menos usable de todos. Esto es debido a que la idea original, como podemos observar en los esquemas conceptuales de los anexos, fue una y a lo largo del proyecto se fue modificando esta idea por parte de la empresa AKO resultando en una cambio gradual de las clases que lo implementan. En los anexos podemos encontrar el conjunto de pantallas definitivo de las configuraciones. Lo más complicado a nivel de código de esta parte ha sido en iOS las listas expandibles con elementos seleccionables.

3.6.- Dashboards

3.6.1.- Funcionalidades

La funcionalidad del dashboard consiste en mostrar al usuario los datos más importantes de cada uno de los dispositivos. El usuario puede seleccionar si los datos que se muestran son diarios, semanales o mensuales. Una vez seleccionado el tipo de rango de fechas, el usuario puede avanzar o retroceder la fecha de la cual se muestran los datos a través del gesto de *swipe*. Un *swipe* hacia la izquierda, retrocede la fecha un día, una semana o un mes, dependiendo del modo de fecha. Y un *swipe* hacia la derecha avanza la fecha, nunca sobrepasando la fecha actual ya que el microcontrolador sólo guarda datos hasta el periodo actual. Adicionalmente el usuario, manteniendo el dedo pulsado en uno de los valores, es capaz de establecer un límite virtual en este valor. De esta forma, si el valor sobrepasa ese límite, se alerta al usuario con una flecha roja sobre en el panel y con una alarma en la sección de alarmas. En la parte inferior de este panel de datos, podremos encontrar los botones para acceder a las funcionalidades de eventos, gráficos continuos y de tendencias y auditorías.

3.6.2.- Diseño gráfico

A nivel visual esta funcionalidad está implementada en una única pantalla formada por diferentes fragmentos o *ViewControllers*. En todos los dispositivos, encontramos en común el selector de día/semana/mes y la barra inferior con los botones de eventos, auditorías … Después para cada tipo de modelo de dispositivo encontramos un fragmento gráfico diferente ya que el tipo de dato que se visualiza en la pantalla es diferente. Cada elemento va relacionado con su unidad de medida y encontramos en la esquina superior derecha de cada elemento unas flechas de color verde o rojo que indican si los objetivos se han superado. Si mantenemos presionado un elemento, nos aparecerá una pantalla que nos permitirá modificar el actual objetivo.

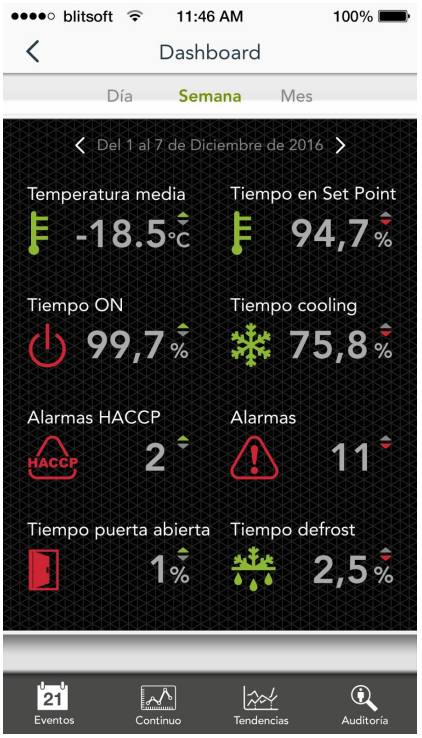


Figura 12. Dashboards específico del dispositivo AD1-4R

3.6.3.- Implementación

Primero encontramos la pantalla exterior con el selector de fecha y las opciones inferiores gestionada por la clase ActividadAlarmasAD. Esta pantalla es la encarga de gestionar la fecha y de dar órdenes a los controladores adjuntos para que muestren los datos en las fechas correctas. Los controladores dependiendo del modelo al que se esté conectado están gestionados por una clase diferente con el nombre DashboardTopADx. Una vez seleccionada una fecha, se crean en la fecha actual, 3 controladores de pantalla, cada uno con una fecha anterior y posterior, que se encargan de pedir los datos de la fecha asignada. De esta forma siempre habrá en pantalla un controlador cargado por si el usuario quiere desplazarse a un día anterior o posterior, de esta forma se evitan tiempos de carga lentos. Cada vez que se realiza un cambio de fecha, se generan 3 nuevos controladores para las correspondientes fechas. A nivel de objetivos (las barras rojas y verdes), son los controladores de pantalla los encargados de gestionar estos eventos pero los objetivos se encuentran almacenados en la clase de datos.

3.7.- Gráficas

3.7.1.- Funcionalidades

Permite visualizar un dato a lo largo de un periodo de tiempo indicado por el usuario. También comparar un dato con otros datos en ese mismo periodo de tiempo. Se distinguen dos tipos de gráficas: las gráficas lineales (Continuo) y los diagramas de barras (Tendencias). Los datos de estas gráficas se generan y almacenan en el microcontrolador, cada tipo de datos (continuo o lineal) se almacena en un registro diferente ya que los datos en forma de gráficos lineales tienen mayor granularidad que los de barras, es decir, la cantidad de datos necesarios para generar las gráficas lineales es mayor. Éste último tiene una granularidad a escala de minutos, en cambio, los de tendencias son a nivel de día.

3.7.2.- Diseño gráfico

La API del generador de gráficas es muy completa y permite al usuario un gran nivel de customización. El área generada entre la línea de datos del valor principal y el eje de coordenadas se pinta siempre para resaltar ese dato. Sucesivamente, se dibujan las líneas de los valores de los elementos a comparar. Como podemos ver en la imagen de la figura 13 de una gráfica continua, el dato principal, la temperatura de la cámara, se resalta en verde y se compara con la humedad de la cámara y la temperatura en el registro.

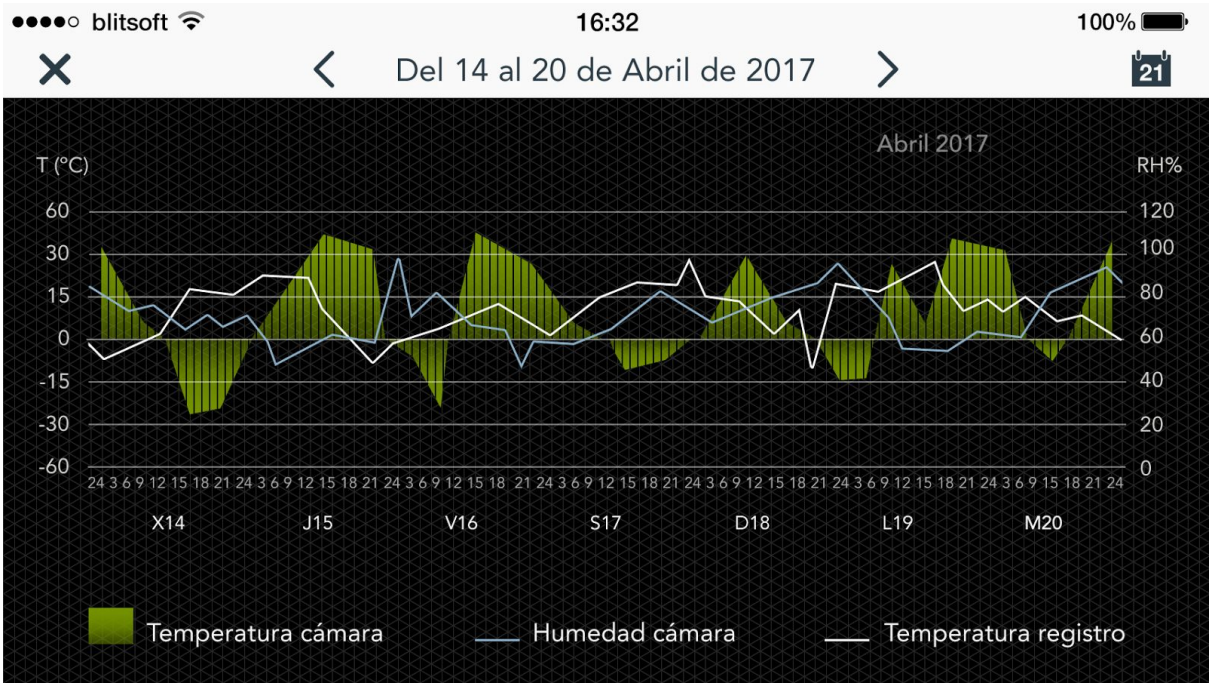


Figura 13. Gráfica continua, comparando 3 valores de temperatura de diferentes sondas.

La gráfica de tendencia permite comparar varios valores a lo largo de la semana, con menor granularidad que una gráfica lineal. Algunos datos se visualizan con mayor claridad en éste tipo de gráficas, como por ejemplo los datos acumulativos.

Figura 14. Gráfica de tendencias versión alpha

3.7.3.- Implementación

Las gráficas en iOS las gestionamos a través de las llamadas que ofrece el sistema operativo sin ninguna librería de soporte. En el caso de Android, todavía no está definido el procedimiento que usaremos para crearlas. Es muy probable que las gráficas generadas visualmente no tengan el mismo aspecto en cada una de las plataformas. Las clases usadas para la generación de gráficas se encuentran en el subdirectorio: ios/Android -> source -> Classes -> UI -> Graficas -> Grafica/Tendencia/Continuo.

3.8.- Alarmas/Eventos/Auditorías & Mochila

3.8.1.- Funcionalidades

Las pantallas de Alarmas, Eventos y Auditorías forman parte de las pantallas de visualización gráficas de datos. Este trío de pantallas muestra en forma de listado una serie de sucesos que se producen en el microcontrolador. En el caso de alarmas se muestran las alarmas divididas en tres categorías. En el caso de eventos y auditorías se muestran ciertos cambios en algunos parámetros del microcontrolador. Por ejemplo, activar el modo de arranque genera un evento de auditoría.

Las funcionalidades asociadas a la mochila son las siguientes: consultar el nivel de carga de la batería de la mochila, consultar la versión del firmware (opcionalmente actualizar el firmware), modificar el PIN de Bluetooth LE que se usa durante la fase de *pairing&bonding*, guardar los datos de la mochila en el dispositivo móvil, borrar los registros de la mochila y acceder a los datos actualizados de eventos y auditorías de la mochila.

3.8.2.- Diseño gráfico

A nivel de diseño las pantallas de de Auditoría/Eventos/Alarmas son un simple listado con todos los valores que representan cada uno de estos elementos. En posición vertical las pantallas muestran N valores y en posición horizontal, al disponer de más espacio, las pantallas muestran M valores siempre M ≥ N. Por defecto se muestra siempre los datos en la semana actual, el usuario a través del icono del calendario puede modificar el intervalo de tiempo. Como la carga de los valores necesarios para estas pantallas puede ser muy larga, se bloquea la interacción con el usuario hasta que los datos hayan cargado.

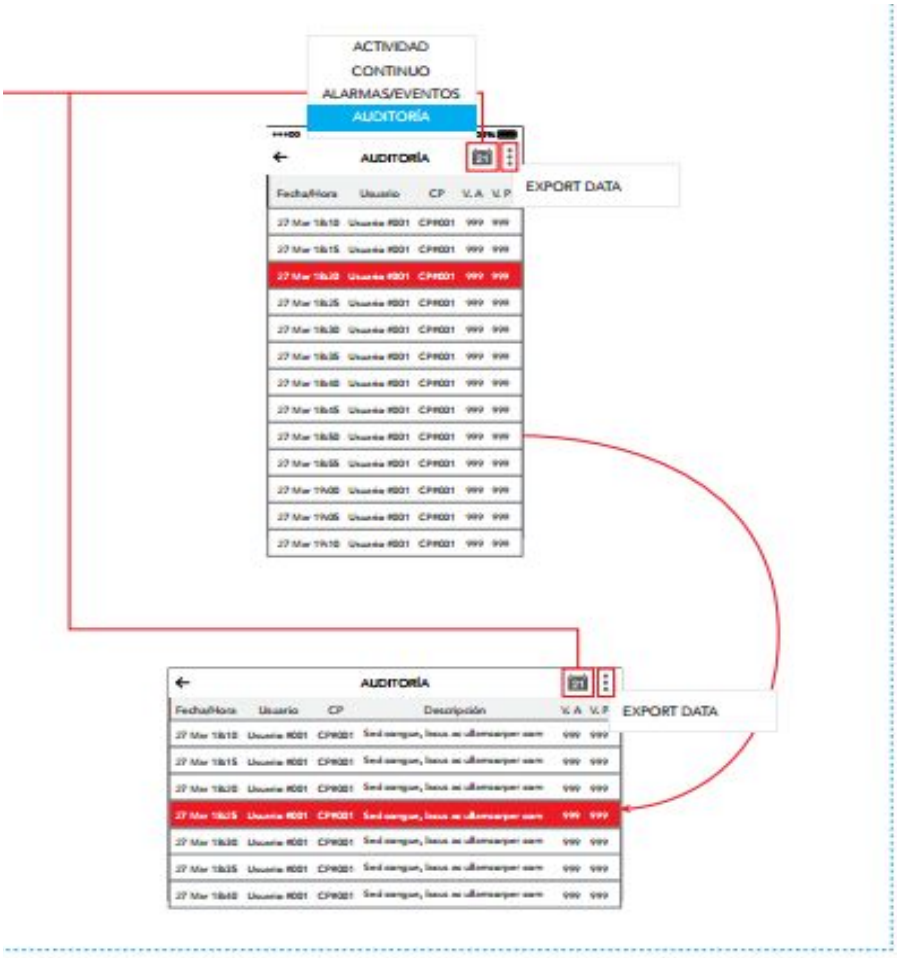


Figura 15. Pantalla de Auditoría en orientación *portrait y landscape*

La pantalla de mochila es únicamente una interfaz gráfica para visualizar los datos de la mochila y acceder a las opciones que ofrece ésta. Se indica a través de iconos el nivel de carga de la batería, la memoria total ocupada por los datos del registro del microcontrolador, la versión del firmware, etc.

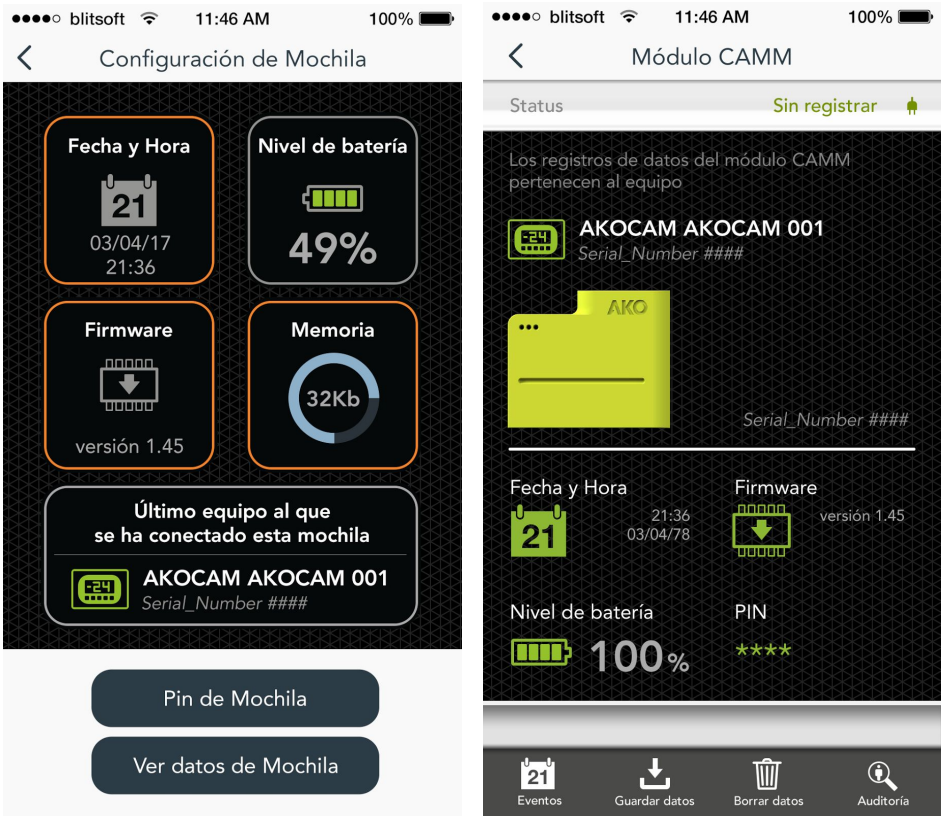


Figura 16. Versiones alpha de la pantalla de mochila

3.8.3.- Implementación

El módulo de mochila se ha definido a finales de Mayo por lo que ha provocado que las clases de datos se tengan que adaptar a éste módulo. Para evitar tener que adaptar toda la clase ProtocolModbusHelper.swift/java que era la encargada de gestionar las funciones y los códigos Modbus, se ha añadido una clase aparte únicamente encargada de gestionar las peticiones a la mochila ya que los comandos y las direcciones de memoria son completamente diferente. Aparte, la mochila tiene un conjunto de parámetros propios de ésta que se acceden de una forma completamente diferente a la habitual. De momento se ha implementado la consulta de batería, el cambio de pin y guardar/borrar datos del registro continuo del microcontrolador. Todavía falta por determinar si se hará la opción de actualizar el firmware y cómo se hará.

3.9.- Diagrama de clases

El diagrama de clases de la aplicación Android/iOS consta de más de 100 clases distintas por lo que no es visualizable en un documento de texto. Para hacerse una idea, la figura 17 muestra el diagrama de clases de la aplicación en Android. Algunas cumplen la función de una estructura de datos, otras son herencia de la clase Activity/Fragment de Android, clases instanciables gráficamente. También encontramos clases de comunicación como pueden ser ModelData o ProxyConnection, clases de mensajería como son todas las clases que heredan de RequestGeneral. Finalmente, encontramos las clases misceláneas con operaciones estáticas, por ejemplo Utils.

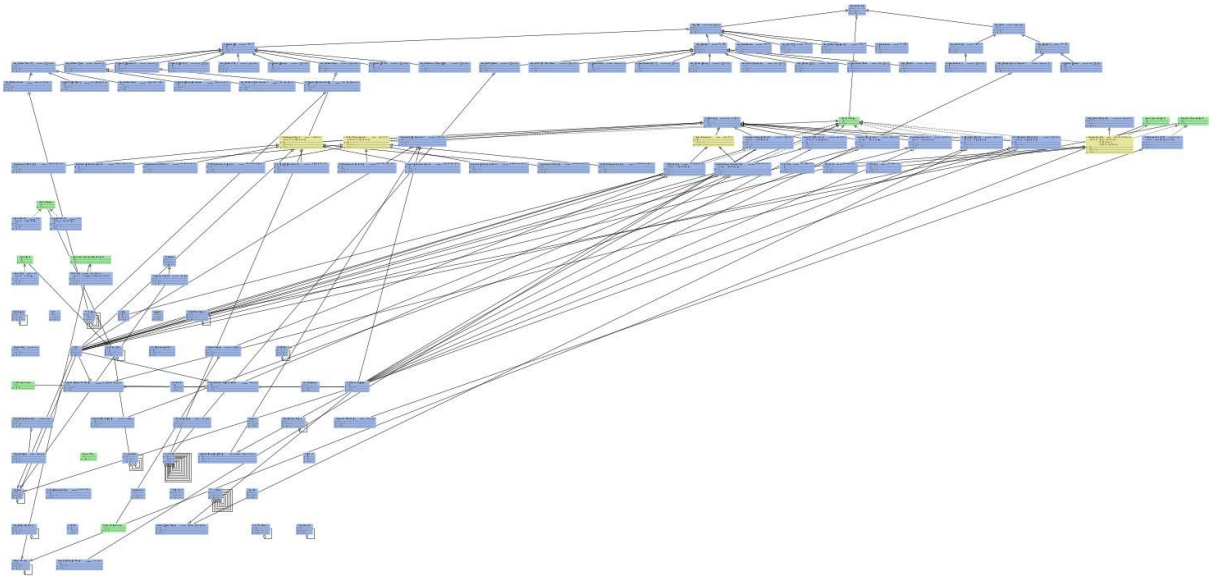


Figura 17. Diagrama de clases Android generado con SimpleUML

Dentro de esta gran multitud de clases, encontramos un conjunto reducido de clases que son las clases de control. Las clases de control son: ModelData, ProxyConnection, ShareConnection, BluetoothConnection, LoginConnection, FileConnection. Para la gestión de tramas Modbus encontramos la clase auxiliar ModbusProtocolHelper. En la versión final del proyecto, encontramos también la clase ReadFileRecordHelper que nos ayuda a gestionar la lectura del registro continuo del microcontrolador ya que se accede de una forma completamente distinta a todos los otros registro de parámetros y se requiere un sistema de colas para gestionar los datos. La figura 18 muestra el conjunto de clases de datos. Al ser Singletons vemos que tienen una referencia a ellas mismas y aunque entre ellas se llamen, al ser referencias estáticas, no se muestra una relación en el esquema UML.

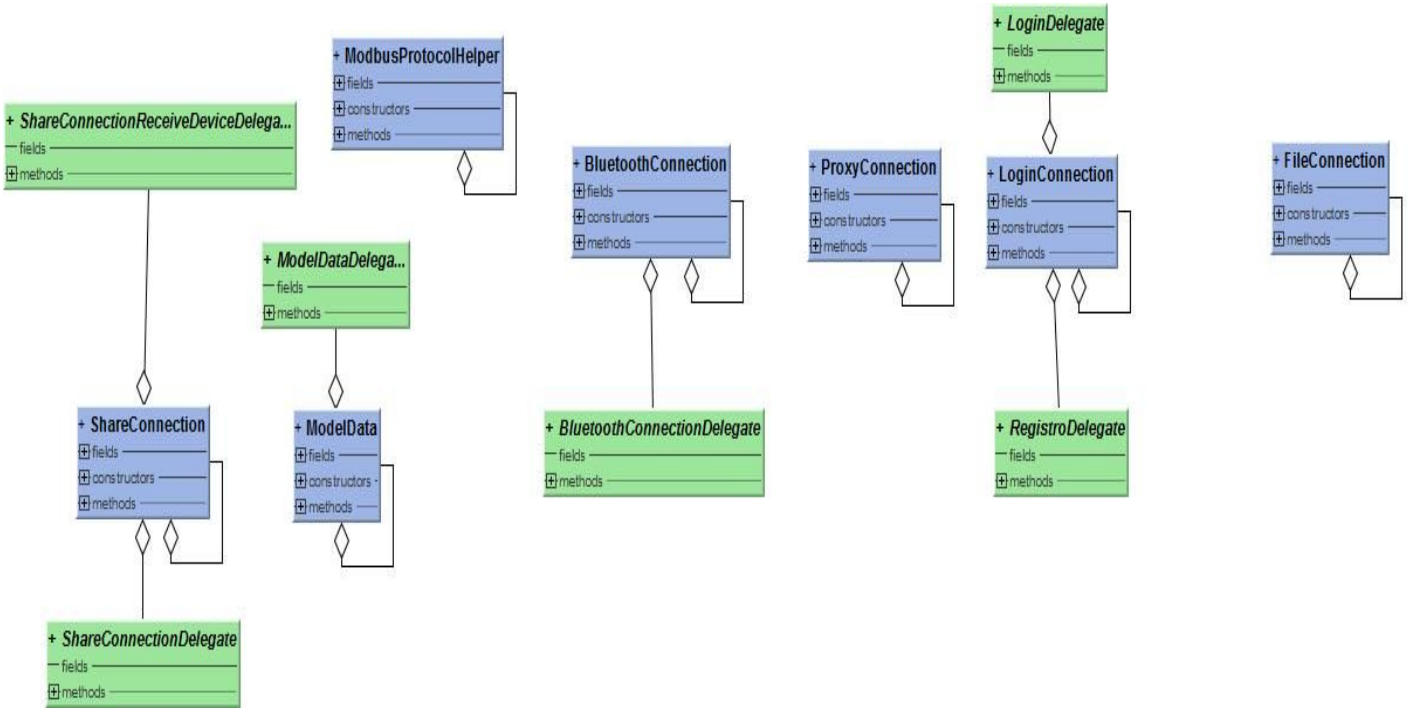


Figura 18. Diagrama de clases de control de la aplicación Android.

A continuación se explicarán las funcionalidades principales de cada clase:

- ModelData: todo dato que las clases de interfaz necesitan se solicita a esta clase. Esta clase no distingue el origen de los datos. Para obtener los datos recurre a ProxyConnection. ModelData se encarga de guardar en memoria los datos de forma persistente para evitar perder tiempo buscándolos de nuevo en caso que se vuelvan a solicitar. Para comunicarse con ProxyConnection utiliza el sistema de mensajes basado en la clase Request.

- ProxyConnection: es la clase encargada de identificar el tipo de dato solicitado por el mensaje de ModelData y comunicarse con la clase BluetoothConnection en caso de tener conexión directa con el microcontrolador por BLE o de comunicarse con el servidor de sharing a través de la clase ShareConnection o de obtener los datos de un fichero, ya sea en *cloud* o local a través de la clase FileConnection.

- BluetoothConnection: es la clase encargada de comunicarse con el microcontrolador a través de la API de Bluetooth LE de Android y de iOS. Como la comunicación con el microcontrolador es a través de tramas Modbus formadas por cadenas de bytes utiliza la clase ModbusProtocolHelper para crear estas tramas y solicitar los datos correspondientes.

- ShareConnection: es la clase encargada de comunicarse con el servidor Node.js de *sharing* a través del protocolo HTTP y de la librería Socket.io. Se encarga de establecer una comunicación con el servidor bidireccional para enviar y recibir datos de otro usuario conectado directamente al microcontrolador por Bluetooth.

- FileConnection: esta clase se encarga de solicitar los datos a un fichero. Este fichero puede estar almacenado de forma local en los ficheros internos de la aplicación o en un servidor remoto.

- LoginConnection: esta clase se encarga de comunicarse con el servidor de registro a través de los servicios web abiertos en el servidor. A través de la API HTTP/TCP que ofrece Android e iOS un usuario puede autenticarse en el servidor, registrarse, renovar la contraseña, etc.

- ModbusProtocolHelper: clase auxiliar encargada de traducir los mensajes Request en java a mensajes escritura y lectura Modbus en cadenas de bytes y viceversa. También se encarga de calcular el código de corrección de errores de las tramas (CRC) y de transformar los datos binarios a tipos básicos de Java como son Integer, Float y String UTF-8.

- ReadFileRecordHelper: clase auxiliar encargada de gestionar las tramas procedentes de Modbus del registro continuo mediante de colas de recepción. También se encarga de gestionar una pequeña cache en el dispositivo de todos los datos almacenados del registro continuo.

Estas son las principales clases de comunicación y de gestión de datos. Estas clases utilizan gran multitud de clases auxiliares para hacer posible la comunicación con las clases de interfaz. Todos las peticiones de datos de las clases gráficas son devueltas con los datos a través de una interfaz común que es la *interface*/*protocol* ModelDataDelegate a través de un *Hashmap* donde se almacena el dato con su correspondiente clave.

3.10.- Servidor de registro: Java Servlet y PostgreSQL

Para la compilación del servidor en Java se ha utilizado la plataforma integrada Eclipse Neon para facilitar la importación de librerías y también la compilación y exportación del fichero .war para el servidor Apache Tomcat 7. Para instalar el servidor Tomcat necesitamos la máquina virtual de java en la que correrá todo el código del servidor (importante que la versión de la JVM sea soportada por la versión instalada de Tomcat). En este link [21] encontraremos una tabla con las diferentes versiones de Tomcat y de JVM. Recordar que necesitamos el compilador de Java para compilar el servidor. Una vez tengamos la máquina virtual de Java instalada podemos empezar con la instalación de Tomcat. Ésta se puede hacer a través de gestor de paquetes como APT o compilando los binarios. En este proyecto hemos usado un servidor (virtualizado seguramente) de Amazon con este Kernel de sistema operativo: *Linux version 3.10.0-327.36.3.el7.x86_64* y hemos compilado a través del código fuente el servidor Tomcat 7. La configuración de un servidor Tomcat es algo compleja por lo que no está dentro del abasto de este proyecto. Una vez tenemos el servidor configurado falta hacer el deployment del código Java que se encargará de solicitar al sistema los mensajes HTTP hacia la IP/puerto del servidor y de hacer las operaciones oportunas dependiendo del tipo de mensaje.

Para la creación del Servlet en sí, nos hemos apoyado en las siguientes librerías:

- Jersey: API de JAX-RS para Java. Permite la gestión del protocolo HTTP y la configuración de Web Services REST.
- Postgresql: *drivers* para la comunicación con la base de datos y librerías de soporte para consultas SQL. Es importante que la versión de la librería y la versión del driver sean compatibles con la versión de Postgres instalada.
- Javax.mail: *framework* que usaremos para gestionar el cliente de SMTP (Simple mail Transfer Protocol) para la comunicación con el servidor SMTP de correos de Amazon.
- Jackson: librería de *parsing* de JSON de las request HTTP.

Para instalar todas estas librerías hemos usado el gestor de librerías integrado en Eclipse Maven, excepto para Jackson que se añadió a través del ejecutable compilando los códigos fuente. Se podría haber usado Gradle como gestor de librerías.

Una vez tenemos el código preparado con todos los servicios REST listos para la comunicación con la aplicación, debemos compilar los ficheros y hacer el *deployment* en el servidor Tomcat. Para facilitar la compilación nos hemos ayudado de Eclipse que permite exportar el fichero comprimido .war que el propio servidor Tomcat es capaz de descomprimir y ejecutar. Nuestro servidor para almacenar de forma eficiente los datos de los usuarios utiliza una base de

datos, en este caso, PostgreSQL. Para instalar PostgreSQL en el servidor lo hicimos a través de los binarios proporcionados en la página oficial [22]. Como ya he mencionado antes, es importante descargarse los *drivers* correctos de Java para la versión instalada de PostgreSQL. Los datos se almacenan en dos tablas, una para los usuarios registrados donde se guarda información de estos y otra para los usuarios todavía sin confirmar. Para confirmar un usuario, este debe introducir el código enviado a su correo electrónico. La generación de este código es pseudoaleatoria como podemos ver en la figura inferior. El envío de correos electrónicos lo hacemos a través de los servicios Amazon SES con la cuenta privada de AKO ya que es un servicio de pago. Una vez ya tenemos el servidor configurado y la base de datos configurada hemos de abrir los puertos del ordenador y de los routers conectados para que acepten el tráfico TCP/IP para el puerto 22 (HTTP) y el puerto 25 (STARTTLS para SMTP de Amazon). Finalmente, de forma adicional, configuraremos los servicios de arranque de Linux con *init.d* para arrancar el servicio de Tomcat y de PostgreSQL durante el boot del sistema.

```
public class RandomString {

    private static final char[] symbols;

    static {
      StringBuilder tmp = new StringBuilder();
      for (char ch = '0'; ch <= '9'; ++ch)
        tmp.append(ch);
      for (char ch = 'a'; ch <= 'z'; ++ch)
            tmp.append(ch);
      symbols = tmp.toString().toCharArray();
    }

    private final Random random = new Random();

    private final char[] buf;

    public RandomString(int length) {
      if (length < 1)
        throw new IllegalArgumentException("length < 1: " + length);
      buf = new char[length];
    }

    public String nextString() {
      for (int idx = 0; idx < buf.length; ++idx)
        buf[idx] = symbols[random.nextInt(symbols.length)];
      return new String(buf);
    }
}
```

Figura 19. Generador de códigos pseudoaleatorio del servidor.

3.11.- Servidor de sharing: Node.js

El propósito de este servidor es comunicar datos entre una aplicación conectada por Bluetooth al microcontrolador con otra aplicación remota. La aplicación remota se comportará como si estuviera conectada de forma directa al microcontrolador ya que se han organizado todas las clases de datos del proyecto para, en caso de no tener conexión por Bluetooth pero sí conexión con otra aplicación en modo esclavo, seguir funcionando de forma transparente al usuario. El servidor de share se ejecuta en un servicio diferente al servidor Tomcat para reducir la carga de trabajo y en caso de que sea necesario poder pasar este servicio a otra máquina. Se eligió el entorno de ejecución Node.js ya que es un entorno asíncrono orientado a eventos, escrito en javascript y con un gestor de paquetes, npm, fácil de usar y con gran cantidad de librerías públicas de buena calidad. Las librerías usadas en este proyecto son las siguientes:

- Socket.io: librería de gestión TCP/IP y HTTP para la creación de un canal de comunicación bidireccional. Capa de abstracción por encima de HTTP para facilitar la creación de servicios REST.
- HTTP: librería para el uso del protocolo HTTP en Node.js. Es una dependencia directa de Socket.io.
- Express: librería de infraestructuras de aplicaciones web. También es una dependencia directa de Socket.io.

Todas las dependencias se han instalado a través del gestor de paquetes npm. El código del servidor es simple. El servidor actúa como un puente entre el cliente maestro sin conexión al dispositivo y el cliente esclavo con conexión al dispositivo.

La primera fase de la comunicación consiste en la creación de una sala, que puede ser creada por ambos usuarios. Una vez ambos usuarios se registran en la sala con sus direcciones, se inicia el proceso de comunicación. La aplicació maestra solicita datos a través de *request* a la aplicación esclava. Una vez la aplicación esclava recibe esta solicitud, esta se encarga de buscar estos datos al microcontrolador y de comunicar estos datos a la aplicación maestra. El servidor únicamente comunica todos los datos que le llegan a todos los usuarios de la sala excepto al emisor.

En la figura inferior podemos ver la función de Socket.io que se ejecuta una vez los usuarios se conectan a la sala. Esta función crea una sala con un identificador único. Posteriormente, los usuarios podrán comunicarse con los otros miembros de la sala a través de la

función *senddata* de la imagen. La gracia de este proceso no se encuentra en el servidor sino en la aplicación. Ya que esta se debe encargar de transformar las solicitudes a JSON y posteriormente decodificar estas solicitudes, obtener los datos que piden, codificarlos a JSON y devolver una respuesta con los datos. Todo el código del servidor lo podréis encontrar en los ficheros adjuntos.

```
io.sockets.on('connection', function (socket) {

    // when the client emits 'adduser', this listens and executes
    socket.on('adduser', function(username, device, room){
        // store the username in the socket session for this client
        socket.username = username;
        // store the room name in the socket session for this client
        socket.room = room;
        // add the client's username to the global list
        if (typeof rooms[room] != 'undefined') {
            rooms[room][username] = username
        }
        else {
            rooms[room] = {}
            rooms[room][username] = username
        }
        devices[username] = device;
        // send client to room 1
        socket.join(room);

        for (var user in rooms[room]) {
            console.log('username: %s, room', user, room);
        }
        // echo to client they've connected
        socket.emit('senddata', 'SERVER', 'connected', JSON.stringify(rooms[room]), JSON.stringify(devices));
        // echo to room 1 that a person has connected to their room
        socket.broadcast.to(room).emit('senddata', 'SERVER', 'usernames', JSON.stringify(rooms[room]), JSON.stringify(devices));
        socket.emit('updaterooms', rooms, room);
    });

    // when the client emits 'senddata', this listens and executes
    socket.on('senddata', function (sender, message, data_type, data) {
        io.sockets.in(socket.room).emit('senddata', sender, message, data_type, data);
    });
```

Figura 20. Fragmento de código del servidor Node.js

4.- Metodología

4.1.- Metodología de trabajo

La metodología de trabajo utilizada en la empresa para el desarrollo del proyecto ha sido el desarrollo ágil de software. En el cual se han utilizado iteraciones (una iteración se considera el software desarrollado en una unidad de tiempo [21]) de aproximadamente 1 semanas en las que se han planificado pequeños objetivos a conseguir durante la iteración, se han añadido nuevas funcionalidades, se ha realizado corrección de errores de iteraciones pasadas y se ha generado la documentación necesaria para completar la memoria final del proyecto.

Debido a cambios constantes de la empresa contratista por motivos de desarrollo del hardware y por cambios de requisitos y funcionalidades se ha requerido una constante comunicación entre los ingenieros de la empresa Ako y los ingenieros de nuestra empresa BlitSoftware mediante la herramienta Ryver, llamadas telefónicas y visitas semanales y mensuales por parte de directivos de la empresa Ako.

Para la comunicación dentro del grupo de ingenieros de nuestra empresa hemos usado Slack la cual ofrece salas de chat organizadas por temas, así como grupos privados y mensajes directos. Para trabajar con el código usábamos el sistema de repositorios Git con nuestro propio servidor local y la herramienta SourceTree que facilita el uso y evita el mal uso de Git. Las tareas se repartían de forma modular y debido a las dependencias entre ellas era muy importante llevar el trabajo al día y constante comunicación.

Para gestionar el proyecto de una forma eficiente y facilitar el feedback entre el alumno y el director del proyecto se ha utilizado una herramienta web para planificar tareas, llamada Trello. De esta forma el alumno ha tenido una guía de cuál es el siguiente paso a seguir, ya que durante las reuniones con el director se han definido cada una de las tareas a realizar durante esa iteración. Gracias a esta herramienta, el director ha sido conocedor en cada momento de que trabajo estaba realizando el alumno, además de las reuniones presenciales que se han realizado durante el desarrollo del producto. También ha sido una herramienta muy potente porque en el contexto del proyecto, ha establecido un canal de comunicación bidireccional entre los integrantes del proyecto y cualquiera de los integrantes ha podido hacer una aportación directa al planificador para que el resto de miembros hayan podido observar los cambios en tiempo real.

El ponente se encarga de la parte administrativa del proyecto y de validar que el trabajo se realizaba de forma correcta, respetando la normativa de formato de la UPC. En el tramo final del trabajo, una vez superada la fase de GEP, el ponente se va ha encargar de guiar al alumno durante lo que quede de trabajo y ayudarle a preparar la presentación de éste delante del tribunal.

Importante destacar que no se han producido cambios en la metodología de trabajo ni con la forma de comunicación con el ponente hasta la fecha de hoy.

4.2.- Validación del proyecto

Todo los módulos en los que se ha dividido este proyecto han sido probados y validados antes de ser incorporados al proyecto final para así asegurar su correcto funcionamiento. La validación del proyecto no ha sido tan meticulosa como podría haber sido en un proyecto de carácter científico divulgativo. La metodología que se ha seguido para validar los diferentes módulos ha sido a través de numerosas pruebas y test de calidad. Lo primero que se hacía al empezar un nuevo módulo era crear una rama de Git para ese módulo, se programaba el módulo y una vez el módulo estaba operativo (no por tanto validado) se aplicaban las diferentes pruebas. Las pruebas eran diferentes dependiendo del tipo de módulo:

- Módulos relacionados con Bluetooth LE: se usaba un programa *sniffer* para interceptar los paquetes y ver el contenido de las tramas. También se comprobaba que las tramas llegaran correctamente al *smartphone*/microcontrolador a través de modos de *debugging* de ambos sistemas.

- Módulos relacionados con el comportamiento de la aplicación: a través de test unitarios y pruebas por un grupo de QA formado por personas ajenas al proyecto.

- Módulos relacionados con el servidor: primero se hacían pruebas locales en un servidor Tomcat. Una vez comprobado el correcto funcionamiento del módulo, se recompilaba el servidor con el módulo añadido.

Adicionalmente, a los test explicados, también se ha testeado en iOS el correcto funcionamiento en los iPhone 5,6 y 7 en las diferentes versiones (SE, normal y Plus) y en Android se han probado los móviles con mayor porcentaje de uso de los usuarios finales de la aplicación. Éstos han sido: Samsung Galaxy en sus diferentes versiones a partir de la versión 4, distintos modelos de Huawei, One Plus, LG, Motorola … En el caso de Android era importante hacer estas pruebas ya que las capacidades hardware de los diferentes móviles son algo distintas y en algún caso el chipset de Bluetooth dependiendo si era versión 4.0 o 4.1 daba problemas. Se ha

priorizado dar soporte a los modelos de *smartphone* con mayor porcentaje de uso entre los usuarios de AKO.

Otro punto crítico es el servidor. La velocidad del servidor no es un factor clave pero sí su disponibilidad. Es muy importante que el servidor tenga una alta disponibilidad, es decir, que esté operativo gran parte del día, sobretodo en horas punta. Cualquier problema que detenga el proceso del servidor o que pare la propia máquina del servidor ha de ser solucionado. Ya que la aplicación bloquea a todo usuario sin autentificar. Sin el servidor de registro la autenticación no es posible y por lo tanto, los usuarios no podrán usar la aplicación. Otro factor a tener en cuenta es la gestión de la carga de trabajo del servidor. Todavía, medianos de Junio de 2017, no se ha testeado como se comporta el servidor en casos de elevada carga de trabajo ni de saturación de la red de entrada/salida. Una vez terminada la aplicación en Android es muy importante simular y probar situaciones de estrés en el servidor para comprobar cuántos usuarios puede soportar simultáneamente. Finalmente, es clave que el servidor sea seguro. Como es imposible protegerlo de todas las vulnerabilidades, el proyecto se centrará en cubrir las vulnerabilidades básicas, aquellas que un sistema automático (robot) pueda encontrar.

5.- Planificación temporal

5.1.- Fases del proyecto

Para la realización del proyecto hemos tenido en cuenta 3 factores fundamentales:

- Recursos (materiales y humanos).
- Actividades (divididas en tareas granulares).
- Tiempo (tiempo dedicado a cada tarea).

Como recurso material se ha contado por parte de la empresa BlitSoftware con un Mac mini de 2.8GHz dual-core Intel i5, con una capacidad de almacenaje de 1TB Fusion Drive, 8GB memoria RAM y gráfica integrada Intel Iris Graphics [22] con los programas necesarios para el desarrollo de la aplicación para iOS que han sido Xcode 8 (entorno de desarrollo y compilación para iPhone, iPad, Mac, Apple Watch y Apple TV [23]), Photoshop CS6 y Adobe Illustrator CC con el que se han diseñado los gráficos tanto para la versión de iOS como para la versión de Android. Además, se han utilizado los dispositivos iPhone 5, iPhone 6, iPhone 6 +, iPhone 6 SE, iPhone 7 para testear que la aplicación tenía el mismo comportamiento en el simulador de Xcode que en los dispositivos reales.

Los recursos necesarios para programar la versión de la aplicación para Android han sido: el ordenador de sobremesa mencionado anteriormente con la herramienta de desarrollo Android Studio versión 1.5.1 para MAC OS X (sistema operativo propietario de Apple Company instalado en todos los ordenadores y dispositivos de Apple desde 2002 [24]).

Por parte de la empresa Ako, se han proporcionado los µcontroladores necesarios para testear el correcto funcionamiento de las aplicaciones y también el hardware necesario para la conexión Bluetooth entre la aplicación y el dispositivo. Ako también ha sido responsable de proporcionar servidores, en este caso servidores alquilados a AWS (Amazon Web Services), necesarios para el desarrollo del servidor HTTP encargado de gestionar las cuentas de los usuarios y los datos de estos.

En relación a los recursos humanos utilizados se ha de contar con muchos ya que estamos hablando de un proyecto de dimensiones grandes. Los roles con mayor implicación directa han sido los siguientes: 2 programadores de aplicaciones móviles por parte de la empresa BlitSoftware (donde me incluyo), uno encargado de la parte de iOS y otro encargado de la parte

de Android, ambos también encargados de testear las aplicaciones, un equipo de programadores de *firmware* por parte de la empresa Ako, un equipo de diseño de hardware (encargados del microcontrolador), dos artistas gráficos, un director de proyecto de la empresa Blit Software, un subdirector y un director de proyectos de la empresa Ako, un director de hardware de la empresa Ako, un director de marketing y finalmente una ponente de la FIB-UPC encargada de supervisar la realización del proyecto y de la memoria de éste. Adicionalmente, ambas empresas tendrán, en caso de baja temporal de algún trabajador, trabajadores sustitutos para evitar posibles retrasos en el proyecto.

Este proyecto se ha dividido en 4 fases principales para completar su ejecución, todas las partes se han de completar con éxito para finalizar este proyecto. Las fases en las que se divide este proyecto son:

- **Fase 1**: Documentación y búsqueda de información de Bluetooth, Modbus, Node.js , etc. La información relacionada con los dispositivos de Ako fue proporcionada por la empresa. Los programadores implicados en la creación de la aplicación ya tenían conocimiento previos de iOS como de Android aunque se han tenido que documentar con mayor profundidad para funcionalidades concretas.

- **Fase 2:** Diseño de la aplicación, tanto de las funcionalidades como de los gráficos. Durante esta fase ambas empresas negociaron como debía ser la aplicación, en qué dispositivos debía poder ejecutarse, qué funcionalidades iba a soportar, etc. Juntamente a la implementación de las versiones para iOS y Android de la aplicación, desarrollo del código y escritura de la memoria provisional.

- **Fase 3**: Revisión del proyecto, desarrollo de documentación técnica y generación de la memoria final.

A continuación se describe con detalle las tres fases mencionadas anteriormente y las tareas en las que se han dividido estas fases. Las fases son secuenciales y las tareas son secuenciales en la mayoría de los casos, aunque algunas se harán de forma paralela. Consultar el diagrama de Gantt en anexos para más información.

5.1.1.- Fase 1: Documentación

La ***fase 1*** contiene las siguientes tareas:

1. **Definición del proyecto:** Durante aproximadamente una semana, ambas empresas (Ako y BlitSoftware) estuvieron pactando y definiendo el proyecto a nivel teórico para comprobar si su implementación era posible. En esta fase también se especificó el contrato y las posibles modificaciones que podía sufrir en el futuro.
2. **Búsqueda de información e investigación acerca de Bluetooth en iOS y en Android y cómo implementarlo.**
3. **Búsqueda de información e investigación acerca de Modbus y de los dispositivos de Ako.**
4. **Búsqueda de información e investigación acerca de servidores HTTP para distribuciones Linux.**
5. **Búsqueda de información e investigación acerca socketIO y BluetoothCore.**
6. **Investigación de aplicaciones en la actualidad que utilizan sistemas de comunicación Bluetooth y cómo lo implementan.**
7. **Investigación de aplicaciones en la actualidad que utilizan sistemas de comunicación Modbus y cómo lo implementan.**
8. **Investigación de aplicaciones en la actualidad que utilizan los FrameWork socketIO y BluetoothCore.**
9. **Búsqueda de información e investigación acerca de servidores HTTP nodeJS.**

Esta fase se completó en el tiempo esperado, compaginada con la elaboración de la documentación de GEP. No únicamente se ha buscado información en esta fase, durante todo el proyecto la documentación ha sido importante para implementar las funcionalidades de ambas aplicaciones y emular el comportamiento deseado en los sistemas. Debido a que los dispositivos de Ako se han diseñado al mismo tiempo que las aplicaciones la constante documentación del funcionamiento de estos ha sido necesaria.

5.1.2.- Fase 2: Diseño e implementación

La ***fase 2*** contiene las siguientes tareas:

1. **Realizar cada uno de los entregables correspondientes a la documentación de GEP:** esta tarea se ha realizado durante el primer mes de la segunda fase, al mismo tiempo que se estaba implementando la aplicación para iOS.

2. **Creación del repositorio donde se almacena el proyecto y diseño conceptual de las clases:** primero para gestionar el proyecto entre varios usuarios se construyó un repositorio de Git en el servidor de la empresa BlitSoftware. Luego se diseñó un UML (Unified Modeling Language) con las diferentes clases que constaría el proyecto y sus funcionalidades.

3. **Diseño, implementación y desarrollo de la classe BluetoothConnection:** posiblemente esta fue la parte más difícil a nivel teórico del proyecto ya que se tuvo que buscar información y entender el funcionamiento del Bluetooth tanto en iOS como en Android y cómo implementarlo y utilizarlo. Esta fue la tarea que llevó más tiempo de todo el proyecto juntamente con la creación del servidor HTTP y con la clase shareProxy.

4. **Diseño, implementación y desarrollo de la clase shareProxy y ModelData:** Model Data es la clase encargado de gestionar todos los datos que utiliza la aplicación. Esta clase se encarga únicamente de gestionar los datos dentro de la aplicación, no de obtenerlos ni de modificarlos. La clase encargada de esto último es Share Proxy, esta clase conoce si un dato debe ser leído/escrito en el servidor HTTP mediante JSON (JavaScript Object Notation) y/o en el dispositivo mediante Bluetooth y/o en los archivos locales de la aplicación. Esta fue la tarea que llevó más tiempo de todo el proyecto juntamente con la creación del servidor HTTP y con la clase BluetoothConnection.

5. **Diseño, implementación y desarrollo del servidor HTTP nodeJS:** esta tarea consistía en diseñar e implementar el servidor nodeJS encargado de cargar los datos de precarga de la aplicación en formato JSON, tanto posibles parámetros de los dispositivos como configuraciones creadas por los usuarios o por las empresas. También se encarga de gestionar las bases de datos con los usuarios y finalmente gestionar la comunicación de datos del mode "*share*" de la aplicación.

6. **Diseño, implementación y desarrollo de las pantallas principales de la aplicación:** esta tarea consistió en planificar la distribución de pantallas y generarlas sin contenido. Durante el proyecto debido a cambios constantes por parte de la empresa Ako la

planificación y distribución de pantallas fue variando por lo que el diseño inicial no se siguió.

7. **Diseño, implementación y desarrollo de la funcionalidad de modificación de parámetros:** esta tarea se realizó en el tiempo estimado, aunque tuvo que ser modificada a medida que nuevos dispositivos con parámetros complejos se iban añadiendo a la aplicación.

8. **Diseño, implementación y desarrollo de la funcionalidad de Configuraciones:** esta tarea se hizo y rehizo más de tres veces debido a cambios subjetivos (para llamarlos de algún modo), de la empresa Ako. Como en la tarea anterior debido a cambios no planificados se tuvo que ir mejorando a lo largo del proyecto.

9. **Diseño, implementación y desarrollo de la funcionalidad de Gráficas (Actividad, Tendencias y Continuo):** ésta posiblemente es la única tarea que se realizó en el tiempo estimado y que se conservó sin ningún tipo de cambio durante todo el proyecto.

10. **Diseño, implementación y desarrollo de las funcionalidades de Auditorías, Alarmas y Eventos:** todavía por definir.

11. **Diseño, implementación y desarrollo de las funcionalidades de BackUp y About:** todavía por definir.

12. **Diseño, implementación y desarrollo de la funcionalidad de Cuentas:** todavía por definir**.**

13. **Porting de la aplicación de iOS a Android, siguiendo los mismos pasos y fases que en la aplicación para iOS adaptando el código al lenguaje de programación y al sistema operativo objetivo:** esta tarea consiste en realizar todas las anterior tareas pero implementándolas para móviles Android en vez de iOS. La realización de esta tarea comporta gran número de desafíos que se tendrán que resolver durante su ejecución.

5.1.3.- Fase 3: Revisión

La ***fase 3*** contiene las siguientes tareas:

1. **Realizar las comprobaciones necesarias para validar que los productos se ajustan a los objetivos planteados.**
2. **Revisar errores de programación "*bugs*" complejos y validar todos los posibles estados de la aplicación.**
3. **Comprobar que el servidor soporta cargas de trabajo reales.**
4. **Redactar la documentación correspondiente a la parte técnica del proyecto.**
5. **Componer la memoria final a partir de la documentación generada durante todo el proyecto.**
6. **(Opcional) Implementar funcionalidad de gráficas para Android.**
7. **(Opcional) Proteger el servidor contra posible ataques como denegación de servicios, cross-site request forgery, API rates limits …**
8. **(Opcional) Mantenimiento y mejora de la aplicación durante un periodo indefinido determinado por la empresa.**

Todas las fases se han realizado con éxito excepto las opcionales. Se ha redactado la memoria técnica y se ha generado la documentación final. La aplicación se ha testeado en diversas ocasiones y se comprueba que responde a todas las funciones y no produce errores. En los anexos podremos encontrar los diagrama de Gantt correspondientes a la planificación inicial y final.

5.2.- Desviaciones y planes alternativos

Las posibles desviaciones que puede sufrir el proyecto son muchas por este motivo solo contemplaré las desviaciones que afecta de forma directa al proyecto por parte de la empresa BlitSoftware donde trabajo. Utilizaré la misma estructura de 3 fases para explicar las posibles desviaciones y los planes alternativos en caso que se produzca una o varias de ellas.

Las desviaciones propias de la ***fase 1*** son:

- **Problemas de documentación de BLE (Bluetooth Low Enegery) y Modbus debido a falta de documentación técnica.** La empresa cuenta con trabajadores con experiencia que han trabajo en proyectos similares lo que permitiría obtener la ayuda necesaria para completar esta tarea. Esto provocaría retraso en otros proyectos pero no afectaría de forma directa a éste.

Las desviaciones propias de la fase 2 son:

- **Pérdida parcial o total de datos del repositorio principal.** La empresa cuenta con un repositorio mirror que se actualiza de forma periódica, se pueden recuperar los datos perdidos pero este supone hacer una migración de datos de repositorios distintos. Esto provocaría un retraso de unas horas.

- **Problemas de compatibilidad de BLE de móvil con el BLE implementado en los dispositivos de Ako.** Este problema se ha encontrado durante la primera fase de implementación donde se probó, con los diferentes sistemas operativos (iOS y Android) y con sus correspondientes implementaciones de Bluetooth, chipset de Bluetooth y firmware de Bluetooth, si estos se podían conectar y enviar información a través del sistema de telecomunicaciones BLE. Debido a las incompatibilidades de la implementación de los dos primeros niveles de la torre OSI (capa física y capa de transporte) de BLE en los diferentes hardware los móviles eran incapaces de comunicarse con los dispositivos. Esto retrasó el proyecto más de dos semanas, una para identificar el problema y otra para solucionar-lo.

- **Falta de acuerdo en el diseño interno y visual de la aplicación.** A lo largo del proyecto, la aplicación era revisada por la empresa Ako. Debido a “nuevas funcionalidades” del producto no contempladas y pedidas por Ako se tuvo que rehacer parte del código de diferentes partes de la aplicación. Esto ha provocado

un retraso muy grande, con más del 20% de horas de trabajo perdidas en rehacer partes ya acabadas de la aplicación.

- **Porting de iOS a Android.** Una vez acabada la aplicación en iOS, ésta debe reescribirse para dispositivos Android. Algunas partes son triviales, ya que muchas clases tienen su equivalente en Android, pero en otras, como por ejemplo los módulos de Bluetooth y de conexiones HTTP los cambios han supuesto un gran esfuerzo de documentación y reimplementación.

Las desviaciones propias de la ***fase 3*** son:

- **Gran cantidad de bugs en las aplicaciones.** Se espera encontrar algún que otro problema una vez acabada la aplicación, lo que no se contempla es encontrar bugs complejos o gran cantidad de ellos que ralentizan el proceso de finalización de la aplicación.

- **Problema de sobrecarga de trabajo de los servidores.** Si la carga de trabajo en los servidores es muy elevada se debería distribuir el trabajo en varios servidores. Este problema supondría un retraso de un par de días.

- **Baja temporal de los trabajadores.** Ambas empresas han contemplado la posibilidad que los trabajadores del proyecto puedan sufrir una baja temporal con una duración mayor a una semana por enfermedad. En este caso, ambas empresas constant con un equipo de ingenieros informáticos suficientemente extenso como para sustituir temporalmente ese ausencia.

5.3.- Desviaciones sufridas durante el proyecto

A día de hoy, 22 de Mayo de 2017, nos encontramos con la siguiente planificación de tareas pendientes. La aplicación para la versión iOS ha sufrido un retraso de aproximadamente 1 mes de duración que se ve reflejado en el actual diagrama de Gantt y también en el cambio de presupuestos, no a nivel material ya que se calculó una posible desviación en recursos de aproximadamente un mes, pero si a nivel de recursos humanos ya que todos los trabajadores implicados recibirán ingresos por las horas extraordinarias del proyecto y esto se verá reflejado en el presupuesto. En el caso de Android, la desviación es mayor, aproximadamente de 2 meses, sin fecha prevista para las funcionalidades de gráficas de tendencias y de gráficas continuas. En la tabla inferior podemos observar las diferentes entregas de hoy hasta la fecha de finalización del proyecto junto a su contenido.

Entrega	Fecha	Definición
BETA iOS v0.6	19/05/2017	● Registro Contínuo (lectura y ajustar) ○ Leer los datos de la mochila ○ Contextualizar los datos ○ Pasar a las gráficas ○ Adaptar el Share ● Mochila ○ ~~Pantalla de mochila~~ ○ Restricciones según estado de la mochila
BETA iOS v0.7	26/05/2017	● Pantalla de escaneo con más opciones (datos del advertising) ● Mochila ○ Descarga y carga de datos de la mochila
iOS v1.0	9/06/2017	● Exportar datos (selección calendario) ● Enviar/importar presets ● Localización textos ● Manuales en el catálogo ● Diferenciación FIT ● Pantalla I/O ● Visualización correcta en distintos tamaños ● Actualización Firmware*
BETA Android v0.1	9/06/2017	● Parámetros ● Configuraciones ● Login ● Emulación
BETA Android v0.2	16/06/2017	● Dashboard ● Eventos ● Alarmas ● Auditoría

		● Mochila
Android v1.0	23/06/2017	● Exportar datos(selección calendario) ● Enviar/importar presets ● Localización textos ● Manuales en el catálogo ● Diferenciación FIT ● Pantalla I/O ● Visualización correcta en distintos tamaños ● Actualización Firmware* *Si no surge ningún problema técnico
Android v1.1	??/06/2017	● Gráfica Contínuo ● Gráfica Tendencias

Tabla 3. Tareas pendientes hasta la fecha de entrega.

Las primera versiones para salir a la calle serían iOS v1.0 y la Android v1.0. Estos cambios en la planificación afectan de forma directa el presupuesto, ya que se consideró una posible desviación de costes de 96 horas laborables para problemas técnicos, pero no se consideró un retraso total del proyecto de aproximadamente 2 meses. El retraso se ha producido por problemas con el hardware del microcontrolador. Debido a que este hardware se está produciendo al mismo tiempo que la aplicación móvil, la aplicación no puede avanzar si el hardware no lo hace. También el retraso ha sido provocado por el añadido de funcionalidades que no estaban contempladas en la planificación final. En los anexos encontraremos la actual planificación en forma de diagrama de Gantt en el apartado Anexo 2: Diagrama de Gantt fase seguimiento.

6.- Gestión Económica

6.1.- Presupuesto

6.1.1.- Identificación de los costes

Teniendo en cuenta que este proyecto se ha realizado en la empresa barcelonesa de software BlitSoft, se ha desarrollado un presupuesto en el que se ha considerado los siguientes elementos:

- Costes directos por actividad
    - Mano de obra de los programadores y *testers*
    - Mano de obra de los artistas gráficos
    - Mano de obra del jefe de proyecto
    - Herramientas de trabajo: Mac Mini Pro
    - Software de trabajo: Xcode, Android Studio, Photoshop CS6 y Modbus Poll
    - Herramientas de *testing iOS*: Iphone 5, 6, 6S y 6SE
    - Herramientas de testing Android: Sony Xperia XZ y Samsung Galaxy s6
- Costes indirectos
    - Consumo eléctrico de la oficina (luz, aire acondicionado) y de los ordenadores
    - Conexión a internet
    - Alquiler del local
- Amortizaciones
    - Hardware
    - Software amortizado por ser gratuito (con la compra de Mac Mini)
- Contingencias
    - Desviaciones de las actividades de: módulo Bluetooth, módulo servidor NodeJs y módulo de *testing/debugging*
- Imprevistos
    - Modificación tecnológica de la API de Android 22 a 23 [25].

6.1.2.- Estimación de los costes

En la figura siguiente se muestran los salarios pertenecientes a cada rol. Estos salarios se basan en una aproximación del estudio de remuneración de Michael Page del año 2015 [26].

Rol	Salario
Jefe de Proyecto (J)	20 € / h
Programador (P) y *Tester* (T)	10 € / h
Artista Gráfico (A)	9 € / h

Tabla 4. Asignación de salarios

6.1.2.1.- Costes directos

- Coste recursos humanos:

	Nombre de la tarea	Duración (horas)	Recursos	Coste (euros)
1	Definición del proyecto y su alcance.	70	P (42h) A (20h) J (8h)	760
2	Investigación acerca de Bluetooth, Modbus, HTTP Server y socketIO.	56	P (56h)	560
3	Aplicaciones que utilizan el estándar BLE (Bluetooth Low Energy) y el protocolo de comunicación Modbus.	32	P (32h)	320
4	Aplicaciones que utilizan la librería socketIO. Servidores HTTP implementados en NodeJS.	32	P (32h)	320
5	Creación del repositorio y diseño UML (Unified Modeling Language) de la arquitectura de la aplicación y diseño gráfico.	110	P (62h) A (48h)	1.052
6	Implementar la clase Bluetooth, ProxyShare, ModelData y Request encargadas de la comunicación entre dispositivos.	114	P (114h)	1.140
7	Implementar el servidor HTTP mediante NodeJS y la creación de los servicios web con éste.	30	P (30h)	300
8	Implementar las pantallas principales de la aplicación en formato *placeholder* (sin funcionalidad).	62	P (42h) A (20h)	600
9	Implementar la clase Parametros encargada de la gestión de los parámetros modificables del dispositivo.	84	P (84h)	840
10	Implementar la clase Configuracion e implementar la clase Graficas.	84	P (84h)	840
11	Implementar la clase Auditorias, la clase Alarmas y la clase Eventos.	60	P (60h)	600
12	Implementar las funcionalidades de *Backup*. Implementar la gestión de Cuentas en la aplicación y en el servidor NodeJS.	60	P (60h)	600
13	*Porting* de la aplicación en iOS a Android.	988	P (900h) A (88h)	9.792

14	Revisión de la aplicación y de los servidores, corregir fallos y pulir funcionalidades.	96	P (40h) T (56h)	960
15	Redactar la documentación correspondiente a la parte técnica del proyecto	56	P (56h)	560
16	Componer la memoria final a partir de la documentación generada	32	P (32h)	320
18	Realizar las comprobaciones necesarias para validar el producto	86	P (18h) T (60h) J (8h)	(no incluída) 940
19	**Total**	**2.052**	**P (1744h)** **A (176h)** **T (116h)** **J (16h)**	**19.694€**

Tabla 5. Coste de recursos humanos

- Coste recursos materiales:

    - Mac Mini: 799€ (comprado 2016 [27])
    - Iphone 5: 398€ (comprado 2015, actualmente descalogado)
    - Iphone 6, 6S, 6SE: 3*600€ (media de los 3 dispositivos [28])
    - Sony Xperia XZ: 499€ (precio en España 2017)
    - Samsung Galaxy s6: 380€ (precio en España 2017)

  El coste de los recursos materiales se obtiene de realizar el cálculo de amortización del equipo personal, el hardware:
  *(Precio de compra del equipo personal / (años de amortización * días laborables al año * horas de uso diario)) * Horas de uso del equipo durante el proyecto* = (3880 / 4*230*8)) * 1346 = **709,60 €**

6.1.2.2.- Costes indirectos

- Consumo eléctrico

  A partir de la factura eléctrica de la empresa se ha calculado un coste fijo de 0,126€/h y un coste por energía consumida de 0,14€/KWh. Teniendo en cuenta que el equipo personal tiene un consumo de 120W/h y que las horas totales de uso son 1346, el coste del consumo de electricidad derivado por el equipo persona es:
  *(Coste fijo + (Coste energía consumida * consumo equipo))* Horas de consumo* = (0,126+ (0,14*0,120))*1346 = **192,20 €**

- Conexión a internet:

  La cuota de acceso a Internet es de 55€/mes, la empresa cuenta con 2 líneas de acceso, por tanto, el coste de conexión a Internet es:
  *(2*cuota acceso Internet/días mes/horas día) * Horas de consumo = (2*55/30/24) * 1346* = **205,60 €**

- Alquiler del local:

  El coste mensual del alquiler del local es de 1000€/mes, la empresa cuenta con 12 trabajadores, por tanto, el coste de alquiler del local para este proyecto es de:
  (Costa del local / Número de trabajadores / Horas laborables al mes) Número de personas trabajando en el proyecto* Horas de trabajo* = (1000/12/240) * 1 *1346 = **467,40 €**

- Contingencias e imprevistos:

  Se ha asignado un período de 4 días laborables, es decir 24 horas del proyecto para imprevistos y desviaciones de las tareas. Estas horas se han contabilizado como una tarea en la asignación de recursos humanos, en concreto la última tarea 18 de *"Realizar las comprobaciones necesarias para validar el producto"*. El coste para el cálculo del presupuesto es de **960 €**.

- Amortizaciones:

  En este trabajo se ha considerado una amortización del hardware de 4 años, ya incluida en el cálculo de costes de recursos materiales. Así mismo también el software queda amortizado, ya que se trata de software propietario de Apple (Xcode) incluído con la compra del Mac Mini, Photoshop CS6 amortizado en anteriores proyectos y software libre (Android Studio) y por tanto su coste es **0€**.

6.1.3.- Coste total

Costes	Concepto	Precio
Costes directos	Coste recursos humanos	20.100,00
	Coste recursos materiales	709,60
Costes indirectos	Consumo eléctrico	192,20
	Conexión a Internet	205,60
	Alquiler del local	467,40
Contingencias	Tarea nº18 Recursos Humanos	960
Amortizaciones	-	0,00
Coste Total	-	**22.429,20€**

Tabla 6. Coste total del proyecto

7.- Sostenibilidad

7.1.- Evaluación de sostenibilidad

En esta parte del informe, evaluaremos cómo afecta nuestro proyecto a las diferentes dimensiones de las que está compuesta la sostenibilidad:

- Dimensión económica
- Dimensión social
- Dimensión ambiental

7.1.1.- Dimensión económica

- ¿Existe una evaluación de los costes, tanto recursos materiales como humanos? ¿Se ha tenido en cuenta el coste de actualización y reparación de los equipos durante la vida útil del proyecto ?¿ Se podría realizar un proyecto similar en menos tiempo/recursos y por lo tanto menor coste ?

En el punto uno se estima el coste de los recursos materiales y humanos y sus implicaciones en el proyecto. No se ha tenido en cuenta, pero la reparación de estos equipos es casi imposible por lo que se deberían comprar de nuevos. Antes de empezar el proyecto se dedicó 2 semanas enteras a plantear el proyecto y cómo minimizar los costes de éste, por lo tanto es difícil encontrar una forma más rápida de realizarlo.

7.1.2.- Dimensión social

- ¿Hay una necesidad del proyecto ?¿ En qué mejora la calidad de vida de los consumidores este proyecto ?¿En qué cambiará la vida del usuario ? Alguien se ve perjudicado por este proyecto ?

Este proyecto ofrece un salto a nivel cualitativo de los operarios y técnicos de las cámaras frigoríficas industriales. Evita que el técnico necesite hardware especializado y costoso para realizar su tarea. También evita en algunos casos que el técnico tenga que desplazarse hasta el lugar donde se encuentra la cámara reduciendo así las emisiones de CO_2. Posiblemente reducirá el número de técnicos necesarios para manipular las cámaras, quitando puestos de trabajo.

A nivel personal, este proyecto me ha ayudado a valorar las tecnologías de integración, a entender el comportamiento de Bluetooth y finalmente a aprender mucho sobre los sistemas operativos iOS y Android y programar para ellos. También ha ser capaz de resolver problemas que no tienen solución a priori buscando alternativas que definitivamente es para lo que nos preparan en la carrera.

7.1.3.- Dimensión ambiental

- ¿Qué consumo tendrán los recursos durante la fase de desarrollo del proyecto y posteriormente durante su vida útil ? ¿Qué consumo e impacto ambiental tendría realizar la misma actividad sin la existencia de tu proyecto ? ¿Se han tenido en cuenta criterios para facilitar su posterior reciclaje ?¿Las fuentes de extracción de los materiales usados en tu proyecto tienen alguna implicación ética?¿Con tu proyecto, reducirás o aumentarás la pisada ecológica ?

El principal consumo que tendrá nuestro proyecto será eléctrico, en España la electricidad proviene mayoritariamente de centrales nucleares o térmicas. Sin este proyecto el 100% de los problemas con la configuración de las cámaras frigoríficas implicaría un desplazamiento por parte del técnico a la fábrica, esto provocaría, a causa del transporte, una emisión de CO_2 a la atmósfera. En este proyecto se han utilizado como dispositivos de comunicación móviles, teóricamente la Unión Europea garantiza el reciclaje de estos si se llevan a depósitos especiales ya que cuando compras un electrodoméstico (móvil) pagas una tarifa para que cuando finalice su vida útil éste sea reciclado. No se ha tenido en cuenta ningún criterio para facilitar el reciclaje de los móviles. Los materiales usados para fabricar móviles son comunes, excepto por el coltán que es bien conocido por sus implicaciones éticas en países como la República Democrática del Congo. En un principio este proyecto pretende reducir el coste económico y a su vez ecológico de la actividad de operar cámaras frigoríficas.

7.2.- Matriz de sostenibilidad

	PPP	Vida Útil	Riesgos
Ambiental	Consumo del diseño	Huella ecológica	Riesgos ambientales
	6	15	-5
Económico	Factura	Plan de viabilidad	Riesgos económicos
	5	12	-5
Social	Impacto personal	Impacto social	Riesgos sociales
	7	10	-10
Rango Sostenibilidad	18	37	-20
	35		

Tabla 7: Matriz de sostenibilidad

8.- Identificación de leyes y regulaciones

Es importante saber que los datos de los usuarios están protegidos por un conjunto de leyes y se han de manipular teniendo en cuenta una serie de regulaciones para preservar los derechos de los usuarios. A continuación explicaremos, de más general a específico, estas leyes y regulaciones. El artículo 12 de la Declaración Universal de los Derechos Humanos (DUDH) y en el artículo 17 del Pacto Internacional de los Derechos Civiles y Políticos se cita:

"Nadie será objeto de injerencias arbitrarias en su vida privada, su familia, su domicilio o su correspondencia, ni de ataques a su honra y su reputación. Toda persona tiene derecho a la protección de la ley contra tales injerencias o ataques."

A pesar de ser una cita muy genérica esta se podría aplicar a los datos privados que almacenaremos en nuestra base de datos. De forma más específica, encontramos el artículo 8 de la Carta de los Derechos Fundamentales de la Unión Europea (CDFUE) que dice [29]:

Artículo 8. Protección de datos de carácter personal.

1. Toda persona tiene derecho a la protección de los datos de carácter personal que le concierne.
2. Estos datos se tratarán de modo leal, para fines concretos y sobre la base del consentimiento de la persona afectada o en virtud de otro fundamento legítimo previsto por la ley. Toda persona tiene derecho a acceder a los datos recogidos que le conciernen y a obtener su rectificación.
3. El respeto de estas normas estará sujeto al control de una autoridad independiente.

Aunque el artículo 8 no se refiera específicamente a las bases de datos con datos de usuarios, podemos incluirlas en esta regulación. El primer punto obliga a la base de datos a ser segura, es decir, a ofrecer protecciones contra accesos no autorizados. Uno de los puntos importantes es la forma en que se almacenan los datos (sobretodo las contraseñas) en estas bases de datos. Es importante almacenar los datos de forma parcialmente encriptada para evitar, en el caso de posibles *leaks* de memoria, que los atacantes se adueñen de la menor cantidad de datos posible.

En el marco nacional español hallamos el artículo 18 de la Constitución Española y también la Ley Orgánica de 1982 que establece una serie de puntos que extienden el artículo 18 de la Constitución. Estos artículos son el artículo 1, 2 y 8 [30]. En caso del no cumplimiento de estos artículos las penas se regulan a través del código penal en los siguientes artículos: el

artículo 197.1 del Código Penal [31] que trata sobre el descubrimiento y revelación de secretos y que castiga con penas de prisión de 1 a 4 años y multa “el apoderamiento de documentos o efectos personales y la interceptación de comunicaciones”. El delito se produce cuando un tercero se hace con documentos o efectos personales de alguien. Por otro lado, también se podría aplicar el artículo 197.2 del Código Penal que trata sobre el delito contra los secretos informáticos, y que castiga con entre 2 y 5 años de cárcel por apoderarse, utilizar o modificar, sin autorización y en perjuicio de un tercer, datos de carácter personal o familiar con el propósito de vulnerar y descubrir la intimidad de otro.

En parte somos responsables de que usuarios terceros no se adueñen de los datos que enviamos al servidor ni al controlador de la cámara frigorífica. Nuestro proyecto tiene en cuenta este aspecto y por este motivo encripta toda comunicación punto a punto. En el caso de la comunicación con la cámara frigorífica el propio Bluetooth al establecer durante la fase de *pairing* genera un canal seguro de comunicación por el cual se intercambia una clave simétrica para comunicarse con la aplicación. En el caso de la comunicación con el servidor a través de Internet, el protocolo Transport Layer Security (TLS) nos proporciona la capa necesaria de encriptación entre el protocolo TCP/IP y el protocolo HTTP.

No únicamente los propietarios de las bases de datos tienen obligaciones, también es importante saber que derecho tienen sobre los datos almacenados en éstas y cómo pueden utilizarlos para su beneficio. La Ley 30/1992, de 26 de noviembre, de régimen jurídico, de las administraciones públicas y del procedimiento administrativo común, dedica 10 puntos en el Artículo 37 al desarrollo del derecho de los ciudadanos al acceso a la información. Incluso para ciertas bases de datos existe una normativa reguladora, como en el caso de las fiscales (Orden de 30 de julio de 1982 sobre limitaciones de acceso a la información contenida en las bases de datos fiscales), las del Impi (Orden de 3 de abril de 1992 por la que se autorizan los precios de las consultas a las bases de datos del Instituto de la Pequeña y Mediana Empresa Industrial), o las del Registro de la Propiedad Industrial (Resolución de 15 de noviembre de 1987, del Registro de la Propiedad Industrial -hoy Oficina Española de Patentes y Marcas-, por la que se establecen las condiciones a las que han de someterse las consultas de sus bases de datos). En definitiva, el usuario legítimo de una base de datos podrá hacer libre uso de su contenido siempre que éste no cause un perjuicio injustificado a los intereses legítimos del titular del derecho (art. 135.2) [32].

Teniendo en consideración todas las leyes y regulaciones mencionadas anteriormente, creemos que nuestro proyecto hace un uso legítimo de los datos del usuario y mantiene estos datos seguros de manipulaciones de terceros.

9.- Conclusiones

9.1.- Dificultades

Las principales dificultades de este proyecto no han sido tanto a nivel técnico sino a nivel organizativo. La planificación inicial del proyecto por parte de ambas empresas era correcta y los plazos eran plausibles. El problema principal se debe al hardware. Este hardware se iba diseñando y construyendo a medida que se iba creando la aplicación. Cada cambio en el hardware comportaba un cambio en la aplicación. Por ejemplo, el hardware de la mochila no estuvo acabado hasta finales de Mayo, esto no únicamente implicó que el módulo de la mochila no se pudo implementar y acoplar hasta medianos de Junio sino que para acoplarlo se tuvo que adaptar código de otras clases. Otro factor que ralentizó todo el proceso fueron los constantes cambios de diseño de la aplicación.

A nivel técnico, los dos principales motivos de desviaciones en la planificación fueron la comunicación a través de Bluetooth Low Energy y la generación de gráficas. En iOs, como el hardware lo limita la propia compañía, el sistema operativo ya está adaptado para un hardware específico y los problemas de compatibilidad son escasos, no hubo problemas con la comunicación BLE entre dispositivos iOS. Todo lo contrario que en Android, que al dar soporte a tal gran cantidad de dispositivos, no se puede evitar encontrar problemas, problemas que se tardó tiempo en descubrir el motivo. En segundo lugar la generación de gráficas con los valores del registro continuo. Teóricamente, tendría que haber sido algo relativamente fácil ya que ambos sistemas ofrecen API's para la generación de gráficas. El problema fue que las API no están diseñadas para ser 100% customizables y muchos detalles que AKO quería eran difíciles de hacer.

9.2.- Integración de conocimientos

Durante la carrera los estudiantes hemos aprendido gran multitud de conocimientos para, al finalizar la carrera, aplicarlos en situaciones reales. Este proyecto es un claro ejemplo de esto, ya que no aporta ningún tipo de conocimiento nuevo a la sociedad, sino que adapta e integra técnicas ya existentes para dar soluciones. Los conocimientos aplicados son los siguientes:

- Bluetooth: ha permitido la comunicación entre el smartphone y el microcontrolador sin la necesidad de implementar toda la torre de protocolos hasta llegar al nivel de transporte UDP/TCP en el dispositivo. Este protocolo se ha dado en la asignatura de Tecnologies de Xarxes de Comunicació (TXC).

- HTTP/REST: ha permitido de forma simple comunicar la aplicación con un servidor externo a través de Internet sin la necesidad directa del uso de sockets IP/TCP. Este protocolo se ha dado en la asignatura de Aplicaciones Distribuidas (AD).

- Paralelismo: ambas implementaciones de Bluetooth LE en Android e iOS utilizan threads para gestionar de forma asíncrona la comunicación. Para poder gestionar los threads ha sido necesario conocer el comportamiento de éstos y los mecanismos de control como por ejemplo las barreras y consistencia de variables con locks atómicos. Este concepto se ha dado en la asignatura de Paralelismo (PAR).

- Modelo-Vista-Controlador: ha permitido separar mediante este patrón software lo que es la lógica del sistema, de la interfaz gráfica (las pantallas) y la comunicación. Este concepto se ha explicado en la asignatura de Proyectos de Programación (PROP).

- Servidor Tomcat en Java y servidor Node.js: ha permitido gestionar un servidor de registro con una base de datos PostgresSQL a través de una máquina virtual de Java. Y un servidor puente de comunicación entre dos dispositivos remotos. Estas tecnologías se han explicado superficialmente en las asignaturas de Aplicaciones Distribuidas (AD) y Proyectos de las Tecnologías de la Información (PTI).

- Arquitectura de computadores: ha permitido trabajar a nivel de bytes con las tramas de Bluetooth para comunicar información. Este concepto lo hemos estado trabajando a lo largo de toda la carrera, especialmente en Arquitectura de Computadores (AC).

- Gestión de proyectos: ha permitido que un proyecto grande y complicado de organizar se completara con éxito. Se ha tratado especialmente en Proyectos de las Tecnologías de la Información (PTI).

- Arquitectura iOS/Android: ha permitido generar las aplicaciones para los sistemas operativos Android e iOS con todos los conocimientos previos integrados. Y diseñar interfaces de usuarios usables. Este concepto se ha tratado de forma superficial en Interacción y Diseño de Interfaces (IDI).

Gracias a todos estos conocimientos aplicados a sido posible crear este sistema de control a distancia para una cámara refrigeradora industrial.

9.3.- Conclusiones

Este proyecto se centraba inicialmente en la creación de un sistema de control remoto para los microcontroladores de cámaras frigoríficas, modelos propietarios de la empresa AKO. La idea inicial era poder modificar el estado interno del microcontrolador a través de tramas Bluetooth. Un sistema muy parecido a un mando a distancia de un televisor con un poco más de complejidad. A medida que el hardware del microcontrolador se iba materializando se fueron añadiendo nuevas tecnologías como Bluetooth Low Energy y nuevas funcionalidades como lectura del dispositivo, lectura del registro contínuo, modo *Share* de la aplicación. Cuando se determinó que estas funcionalidades eran viables, se incorporó el registro de clientes y gestión de cuentas, la generación de gráficas, actualización del firmware a través de Bluetooth , etc.

Todas las funcionalidades mencionadas anteriormente fueron implementadas con las distintas tecnologías que se han explicado a lo largo de este documento. Encontramos tecnologías tan diferentes como una aplicación en Java/C++ para Android y un servidor HTTP/Socket.io en Javascript o una aplicación en swift 3.0/Objective-C para iOS y un servidor HTTP con una base de datos funcionando en una máquina virtual de Java.

La principal dificultad de este proyecto ha sido aprender y adaptarse a tecnologías distintas. Nunca antes se había trabajado con muchas de las tecnologías mencionadas como por ejemplo Bluetooth Low Energy ni sus respectivas API's de Android e iOS. Tampoco antes había trabajado con el protocolo Modbus ni había realizado un *port* de iOS a Android.

Para concluir, se puede comprobar que este proyecto cumple con su cometido, es decir, manipular un microcontrolador a distancia a través de una aplicación móvil iOS o Android. Además, esta aplicación realiza todas las comprobaciones necesarias para mantener la consistencia de los parámetros modificados en el microcontrolador. Adicionalmente, el proyecto ha cumplido con todas las funcionalidades complementarias como son la gestión de clientes, la visualización de los datos en forma de gráficas, la manipulación a través de Internet del microcontrolador, etc. En un futuro, este proyecto servirá de base para todas las aplicaciones de gestión de cámaras frigoríficas industriales y permitirá crear sistemas automáticos capaces de, no únicamente operar las cámaras frigoríficas de forma autónoma, sino también de mejorar la eficiencia energéticas de las cámaras frigoríficas.

9.4.- Futuro del proyecto y posibles mejoras

Una vez finalizado el proyecto en iOS, faltaría por terminar el *porting* a Android con el módulo de gráficas que implementaremos una vez la aplicación esté operativa en el mercado. También se llevarían a cabo las tareas de corrección de bugs durante la beta de la aplicación y mejorar la seguridad y la capacidad del servidor. Todo este trabajo se estima que tenga una duración de dos a tres meses.

A nivel de control a distancia, la aplicación ya es capaz de gestionar todos los eventos del microcontrolador. Cualquier mejora en este sentido implicaría un cambio en el hardware del microcontrolador. Una mejora ya mencionada en anteriores puntos es la integración de IoT en el microcontrolador para así, de forma adicional, este pueda comunicarse en tiempo real con algún servicio web para hacer una gestión inteligente de los datos y para tener un backup continuo de estos.

La aplicación podría ofrecer servicios cloud ya planteados durante la planificación del proyecto que por motivos de carga de trabajo fueron excluidos durante la realización del proyecto. Estos servicios serían:

- Actualización del firmware del controlador a través de una descarga directa de un servidor de descargas.
- Gestión de backups externos de las configuraciones y de los datos del registro continuo del microcontrolador.

Finalmente, se puede aplicar gran cantidad de mejoras en el *back-end* (servidores) del proyecto ya que son también una parte crítica de las funcionalidades que ofrece la aplicación. Ejemplo de mejoras podrían ser:

- Blindar el servidor contra ataques que permitan aprovechar errores de corrupciones de memoria para obtener datos sensibles de los usuarios.
- Repartición de carga de trabajo (*load-balancing*) en varios servidores para evitar saturaciones en un único servidor. A través de técnicas como DNS delegation.
- Seguridad en la comunicación punto a punto a través de SSL/TLS en Apache Tomcat a través de un certificado *trusted* por una *Certificate Authority (CA)*.

10.- Referencias

[1] Blit Software "*About Blit Software*", consultada el 26 de Febrero de 2017. Fuente: http://www.blitsoftware.com/about-us/

[2] Tanenbaum, "*Languages, levels, and virtual machines"*, section 1.1, p.3 1984.

[3] Ako: compañía, "*Nuestra esencia, nuestra historia, modelo de crecimiento",* consultada el 26 de Febrero de 2017. Fuente:http://www.Ako.com/es/empresa

[4] D. Raggett, "The Web of Things: Challenges and opportunities," IEEE Comput., vol. 48, no. 5, pp. 26–32, May 2015.

[5] Statista: The Statistics Portal , "*Number of smartphone users worldwide from 2014 to 2020 (in billions)".*
*Fuente:*https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[6] Statista: The Statistics Portal , "*Apple iPhone and Android smartphone average selling price from 2008 to 2016 (in U.S. dollars)".*
*Fuente:*https://www.statista.com/statistics/612937/smartphone-average-selling-price-iphone-and-android/

[7] Stallings (2005). Operating Systems, Internals and Design Principles. Pearson: Prentice Hall. p. 6.

[8] Statista: The Statistics Portal , "*Distribution of leading smartphone operating systems in Spain in 2014".*
*Fuente:*https://www.statista.com/statistics/432961/leading-smartphone-operating-systems-in-spain/

[9] NetMarketShare, "*Mobile/Tablet Operating System Market Share*".
Fuente:https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

[10] Google Play, "*LG TV Remote Application*", consultada el 26 de Febrero de 2017.
Fuente:https://play.google.com/store/apps/details?id=com.lge.tv.remoteapps&hl=es

[11] Google Play, "*ASmart Remote IR*", consultada el 26 de Febrero de 2017.
Fuente:https://play.google.com/store/apps/details?id=com.raon.aremotefreegalaxy&hl=es_419

[12] Bluetooth SIG, "*Bluetooth: how it works".* Consultado el 10/04/2017. *Fuente:https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works*

[13] J.C. Haartsen, "*The Bluetooth Radio System*", IEEE Pers. Commun., pp. 28-36, Feb. 2000. Fuente: http://ieeexplore.ieee.org/document/824570/

[14] N. Gunasekaran, et al., "*Bluetooth in wireless communication"*, IEEE Comm. Mag., pp. 90-96, Aug. 2002. Fuente: http://ieeexplore.ieee.org/document/1007414/

[15] M. Galeev. "*Bluetooth 4.0: An introduction to Bluetooth Low Energy—Part II"*, EETimes, Jul. 2011. Fuente: http://www.eetimes.com/document.asp?doc_id=1278966

[16] National Instruments, "*Información detallada sobre el protocolo Modbus",* National Instruments Papers, Oct. 2014. Fuente: http://www.ni.com/white-paper/52134/es/

[17] A. Abella, "*5G Wireless Networks*", FIB-UPC, pp. 2-4, Dic. 2016. Fuente: No pública.

[18] SmartGSM, *"HTC EVO 4G: specifications"*, Consultado el 01/05/2017. Fuente: http://www.smart-gsm.com/moviles/htc-evo-design-4g

[19] Android Developers, "*Documentation: Bluetooth Low Energy*", Consultado el 09/01/2017. Fuente: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html

[20] MIT, *"Socket.io-client*", Socket.io-client API source code, Consultado el 16/05/2017. Fuente : https://github.com/socketio/socket.io-client

[21] Apache Tomcat, *"Apache Tomcat versions compatibility*", Apache Foundation. Consultado el 08/06/2017. Fuente : http://tomcat.apache.org/whichversion.html

[22] PostgreSQL, "*Linux downloads from source code*", consultado el 08/06/2017. Fuente: https://www.postgresql.org/download/linux/

[23] Asierra, 2015, August 30, "*Desarrollo ágil de software"*. Fuente: https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software

[24] Apple Inc, "*About Mac mini*", consultada el 4 de Marzo de 2017. Fuente: http://www.apple.com/mac-mini/

[25] Apple Inc, "Xcode 8: how to ?", consultada el 4 de Marzo de 2017. Fuente: https://developer.apple.com/xcode/

[26] J. Siracusa, *"Five years of MAC OS X"*, Ars Technica, Condé Nast Digital, 24 de Marzo de 2006, pg 10-11.

[27] Apple Inc, "*Android API Differences Report*", consultada el 11 de Marzo de 2017. Fuente: https://developer.android.com/sdk/api_diff/23/changes.html

[28] M. Page, "*Los estudios de remuneración de Michael Page 2015*" (2015), consultada el 11 de Marzo de 2017. Fuente: http://www.michaelpage.es/content.html?subsectionid=10166

[29] Apple Inc, "*Android Store: Mac Mini*", consultada el 11 de Marzo de 2017.

Fuente: http://www.apple.com/es/shop/buy-mac/mac-mini

[30] Apple Inc, "*Android Store: Iphone 6*", consultada el 11 de Marzo de 2017.

Fuente: http://www.apple.com/es/shop/buy-iphone/iphone6s/pantalla-de-4.7-"-32gb-oro-rosa

[31] European Union Law, "*Carta de los Derechos Fundamentales de la Unión Europea*", Art. 8, traducción del inglés. Fuente: http://www.europarl.europa.eu/charter/pdf/text_es.pdf

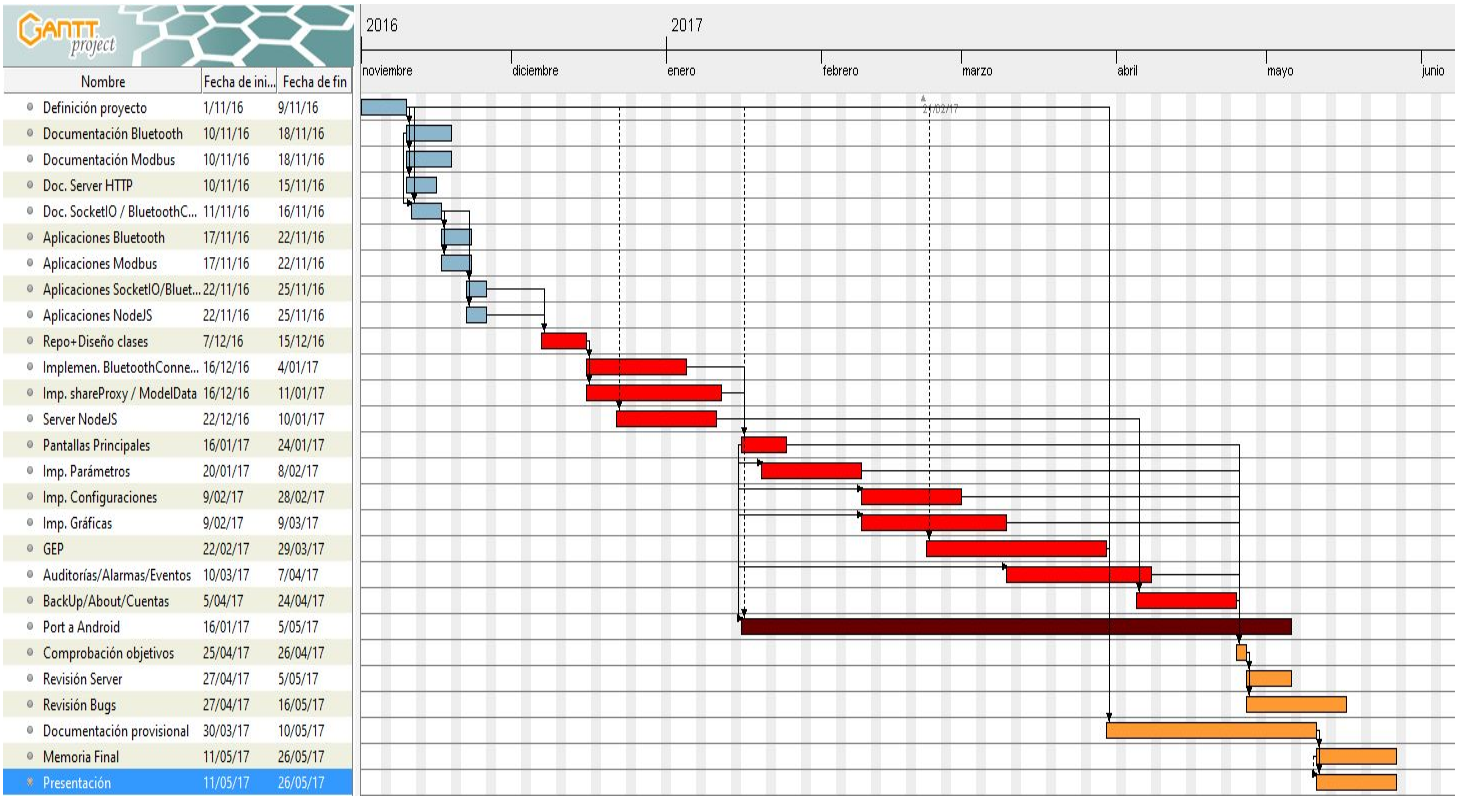
[32] Notícias Jurídicas, "*Ley Orgánica 9/1983*", Jefatura del Estado, publicado en el BOE de 18 de Julio de 1983. Fuente: http://noticias.juridicas.com/base_datos/Admin/lo9-1983.html

[33] Agencia Estatal Boletín Oficial del Estado, "Código Penal y legislación complementaria" Consultado el 19/05/2017. Fuente: https://www.boe.es/legislacion/codigos/codigo.php?id=38&modo=1&nota=0&tab=2

[34] M. Estrella, "*Legislación y control de la propiedad intelectual en bases de datos"*, Julio de 1998. Fuente: http://www.elprofesionaldelainformacion.com/contenidos/1998/julio/legislacion_y_control_de_la_propiedad_intelectual_en_bases_de_datos.html

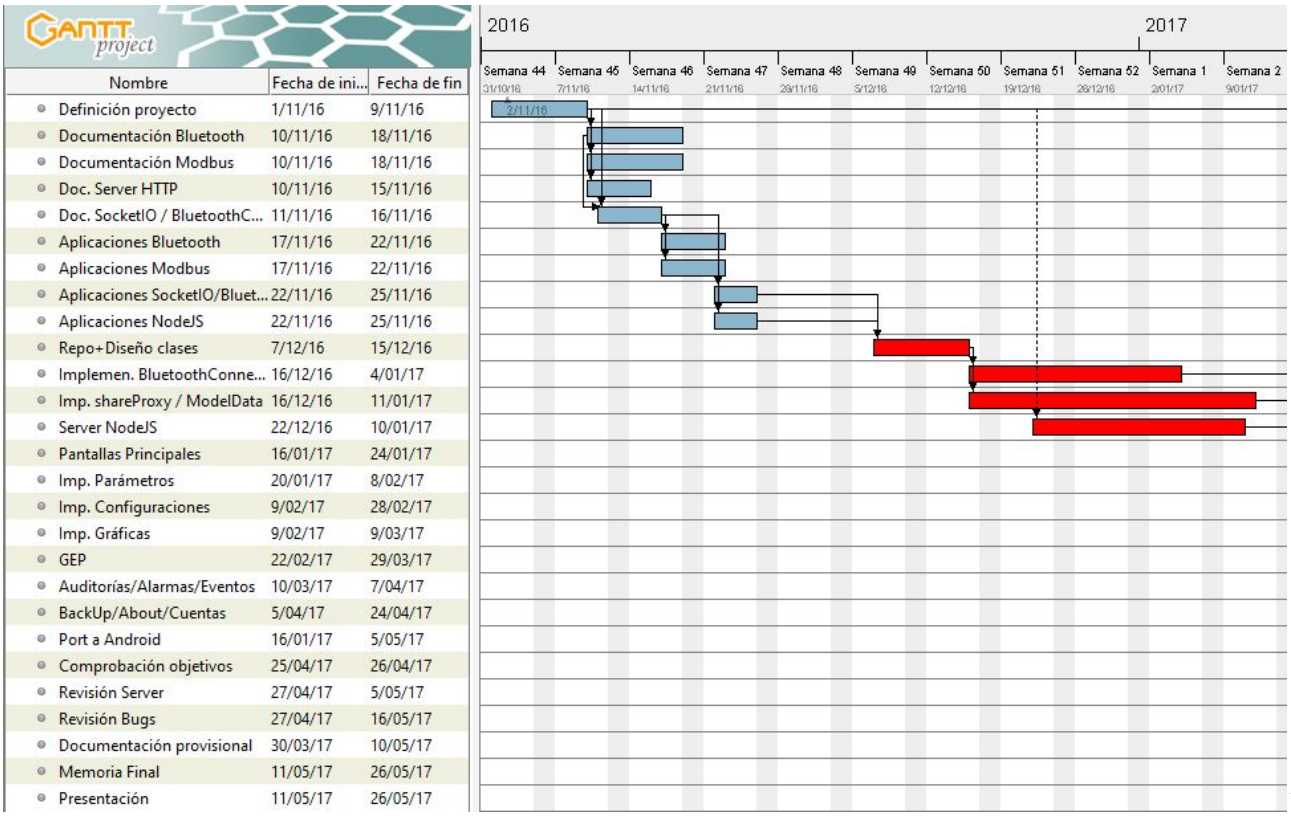
11.- Anexos

Anexo 1: Diagrama de Gantt Inicial



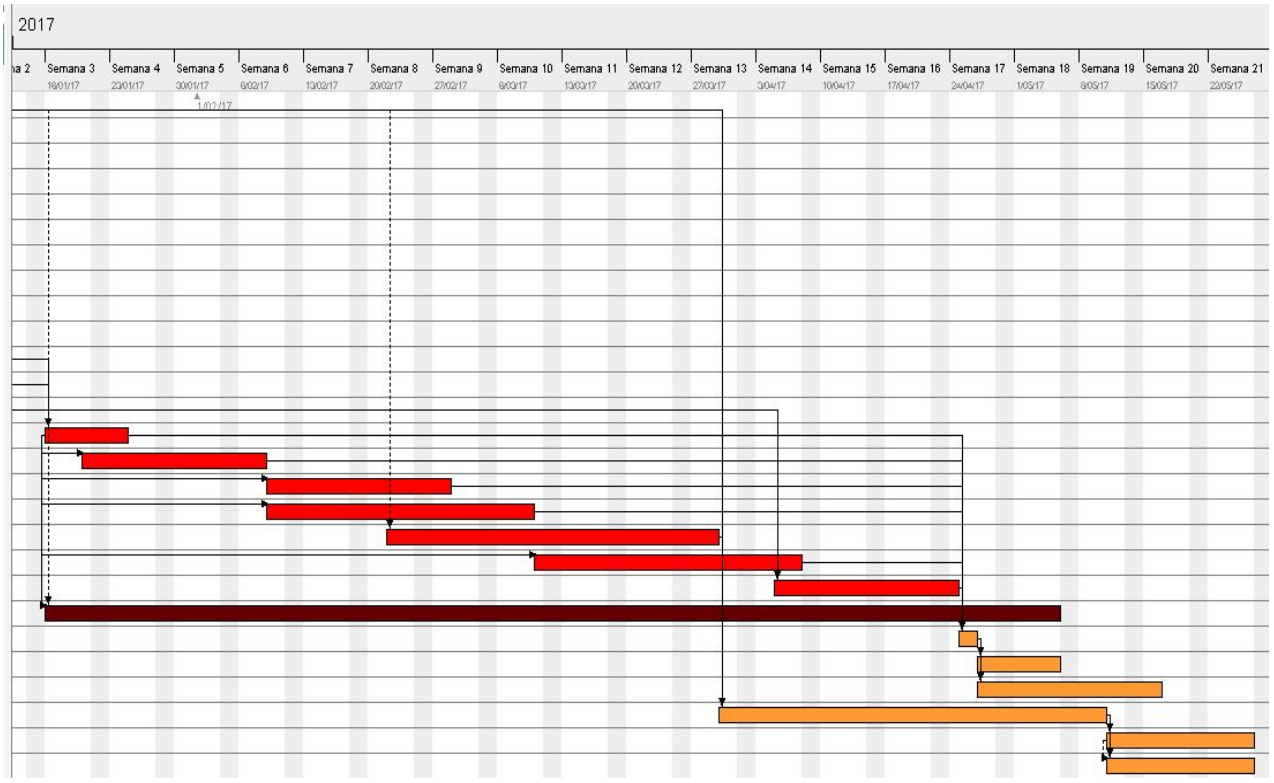
Nombre	Fecha de ini...	Fecha de fin
Definición proyecto	1/11/16	9/11/16
Documentación Bluetooth	10/11/16	18/11/16
Documentación Modbus	10/11/16	18/11/16
Doc. Server HTTP	10/11/16	15/11/16
Doc. SocketIO / BluetoothC...	11/11/16	16/11/16
Aplicaciones Bluetooth	17/11/16	22/11/16
Aplicaciones Modbus	17/11/16	22/11/16
Aplicaciones SocketIO/Bluet...	22/11/16	25/11/16
Aplicaciones NodeJS	22/11/16	25/11/16
Repo+Diseño clases	7/12/16	15/12/16
Implemen. BluetoothConne...	16/12/16	4/01/17
Imp. shareProxy / ModelData	16/12/16	11/01/17
Server NodeJS	22/12/16	10/01/17
Pantallas Principales	16/01/17	24/01/17
Imp. Parámetros	20/01/17	8/02/17
Imp. Configuraciones	9/02/17	28/02/17
Imp. Gráficas	9/02/17	9/03/17
GEP	22/02/17	29/03/17
Auditorías/Alarmas/Eventos	10/03/17	7/04/17
BackUp/About/Cuentas	5/04/17	24/04/17
Port a Android	16/01/17	5/05/17
Comprobación objetivos	25/04/17	26/04/17
Revisión Server	27/04/17	5/05/17
Revisión Bugs	27/04/17	16/05/17
Documentación provisional	30/03/17	10/05/17
Memoria Final	11/05/17	26/05/17
Presentación	11/05/17	26/05/17

Vista General del Diagrama



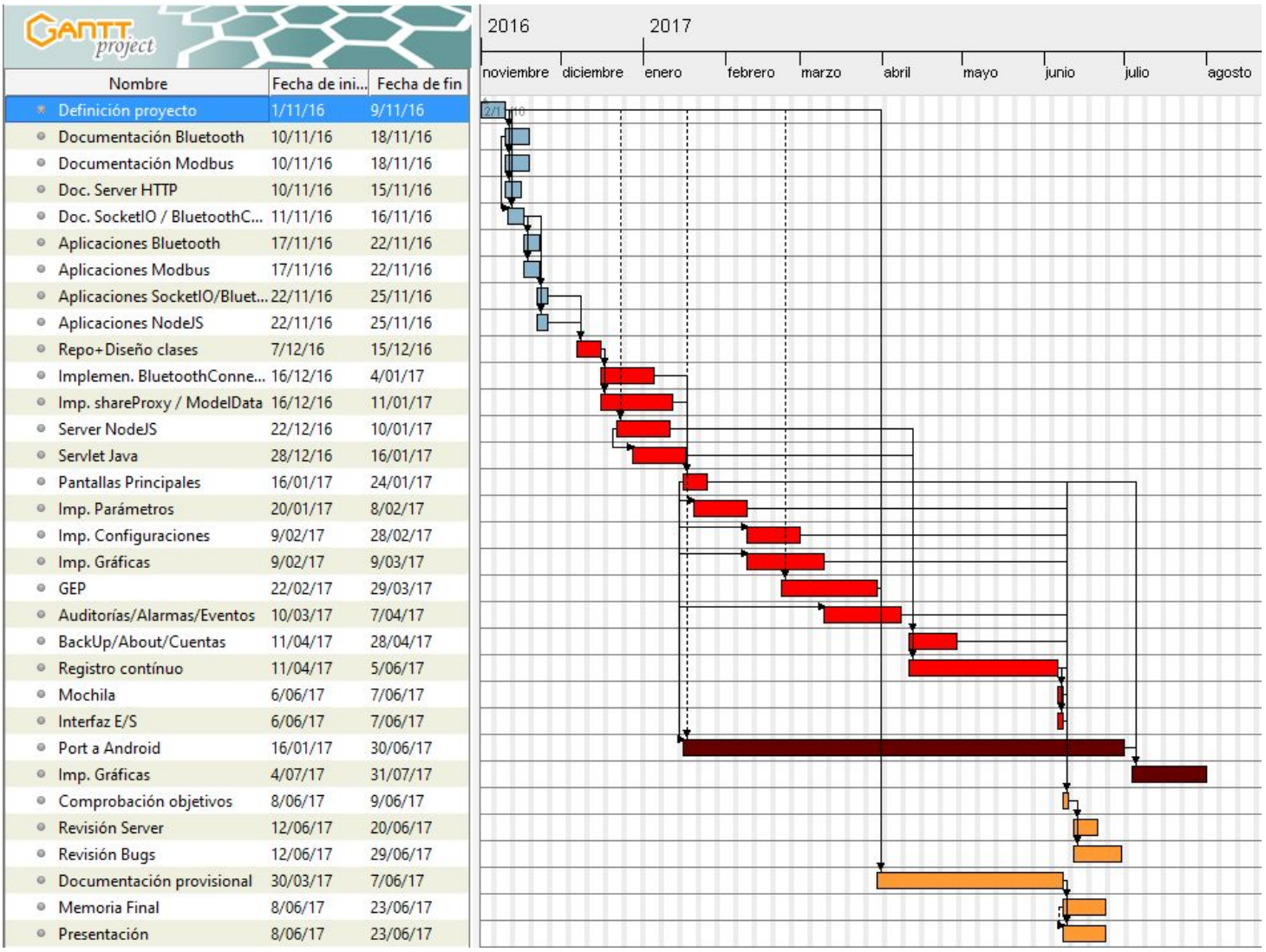
Nombre	Fecha de ini...	Fecha de fin
Definición proyecto	1/11/16	9/11/16
Documentación Bluetooth	10/11/16	18/11/16
Documentación Modbus	10/11/16	18/11/16
Doc. Server HTTP	10/11/16	15/11/16
Doc. SocketIO / BluetoothC...	11/11/16	16/11/16
Aplicaciones Bluetooth	17/11/16	22/11/16
Aplicaciones Modbus	17/11/16	22/11/16
Aplicaciones SocketIO/Bluet...	22/11/16	25/11/16
Aplicaciones NodeJS	22/11/16	25/11/16
Repo+Diseño clases	7/12/16	15/12/16
Implemen. BluetoothConne...	16/12/16	4/01/17
Imp. shareProxy / ModelData	16/12/16	11/01/17
Server NodeJS	22/12/16	10/01/17
Pantallas Principales	16/01/17	24/01/17
Imp. Parámetros	20/01/17	8/02/17
Imp. Configuraciones	9/02/17	28/02/17
Imp. Gráficas	9/02/17	9/03/17
GEP	22/02/17	29/03/17
Auditorías/Alarmas/Eventos	10/03/17	7/04/17
BackUp/About/Cuentas	5/04/17	24/04/17
Port a Android	16/01/17	5/05/17
Comprobación objetivos	25/04/17	26/04/17
Revisión Server	27/04/17	5/05/17
Revisión Bugs	27/04/17	16/05/17
Documentación provisional	30/03/17	10/05/17
Memoria Final	11/05/17	26/05/17
Presentación	11/05/17	26/05/17

Vista Detallada del Diagrama 1



Vista Detallada del Diagrama 2

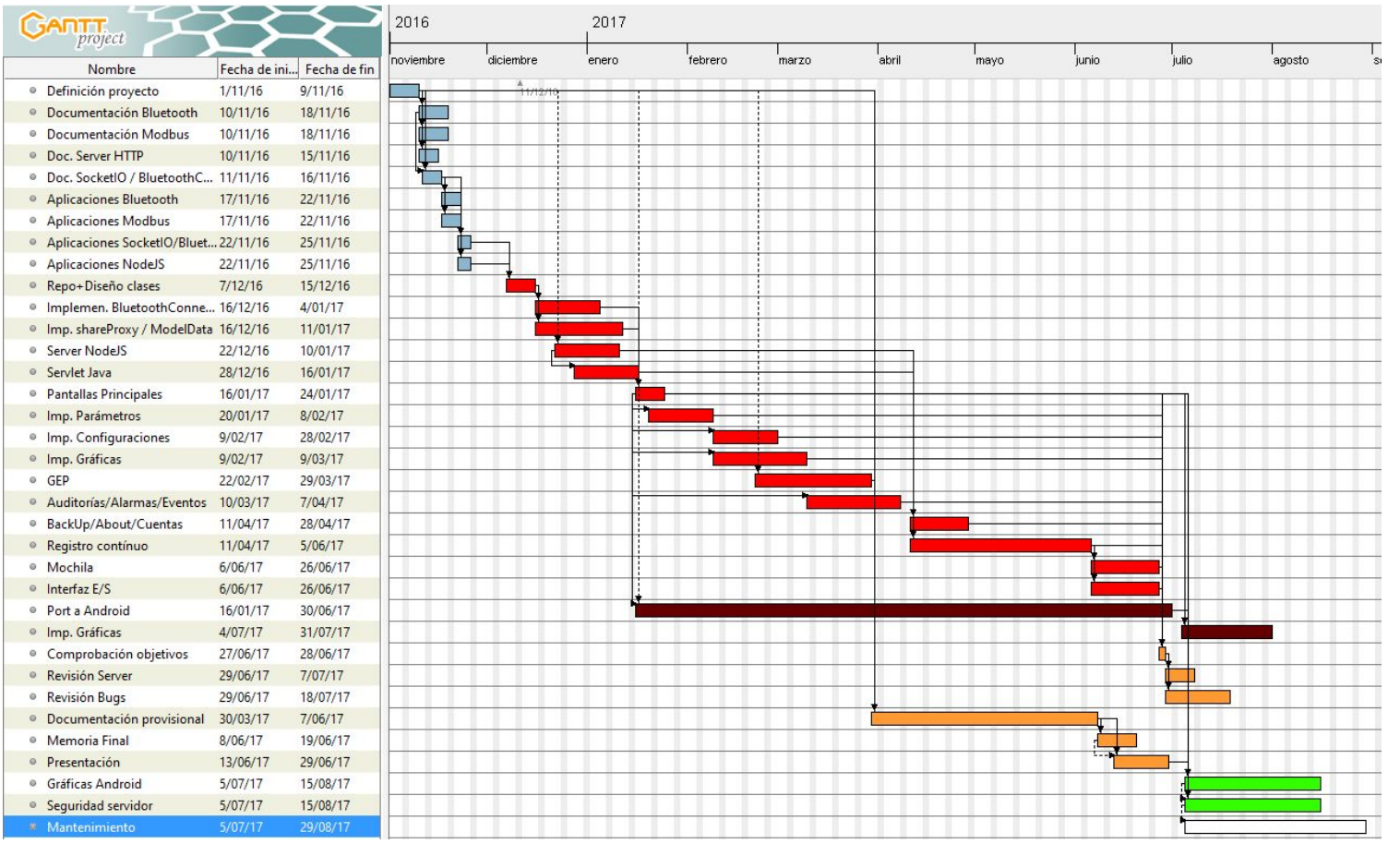
Anexo 2: Diagrama de Gantt fase seguimiento



Nombre	Fecha de ini...	Fecha de fin
Definición proyecto	1/11/16	9/11/16
Documentación Bluetooth	10/11/16	18/11/16
Documentación Modbus	10/11/16	18/11/16
Doc. Server HTTP	10/11/16	15/11/16
Doc. SocketIO / BluetoothC...	11/11/16	16/11/16
Aplicaciones Bluetooth	17/11/16	22/11/16
Aplicaciones Modbus	17/11/16	22/11/16
Aplicaciones SocketIO/Bluet...	22/11/16	25/11/16
Aplicaciones NodeJS	22/11/16	25/11/16
Repo+Diseño clases	7/12/16	15/12/16
Implemen. BluetoothConne...	16/12/16	4/01/17
Imp. shareProxy / ModelData	16/12/16	11/01/17
Server NodeJS	22/12/16	10/01/17
Servlet Java	28/12/16	16/01/17
Pantallas Principales	16/01/17	24/01/17
Imp. Parámetros	20/01/17	8/02/17
Imp. Configuraciones	9/02/17	28/02/17
Imp. Gráficas	9/02/17	9/03/17
GEP	22/02/17	29/03/17
Auditorías/Alarmas/Eventos	10/03/17	7/04/17
BackUp/About/Cuentas	11/04/17	28/04/17
Registro contínuo	11/04/17	5/06/17
Mochila	6/06/17	7/06/17
Interfaz E/S	6/06/17	7/06/17
Port a Android	16/01/17	30/06/17
Imp. Gráficas	4/07/17	31/07/17
Comprobación objetivos	8/06/17	9/06/17
Revisión Server	12/06/17	20/06/17
Revisión Bugs	12/06/17	29/06/17
Documentación provisional	30/03/17	7/06/17
Memoria Final	8/06/17	23/06/17
Presentación	8/06/17	23/06/17

Vista General del Diagrama Gantt de Seguimiento

Anexo 3: Diagrama de Gantt fase final



Nombre	Fecha de ini...	Fecha de fin
Definición proyecto	1/11/16	9/11/16
Documentación Bluetooth	10/11/16	18/11/16
Documentación Modbus	10/11/16	18/11/16
Doc. Server HTTP	10/11/16	15/11/16
Doc. SocketIO / BluetoothC...	11/11/16	16/11/16
Aplicaciones Bluetooth	17/11/16	22/11/16
Aplicaciones Modbus	17/11/16	22/11/16
Aplicaciones SocketIO/Bluet...	22/11/16	25/11/16
Aplicaciones NodeJS	22/11/16	25/11/16
Repo+Diseño clases	7/12/16	15/12/16
Implemen. BluetoothConne...	16/12/16	4/01/17
Imp. shareProxy / ModelData	16/12/16	11/01/17
Server NodeJS	22/12/16	10/01/17
Servlet Java	28/12/16	16/01/17
Pantallas Principales	16/01/17	24/01/17
Imp. Parámetros	20/01/17	8/02/17
Imp. Configuraciones	9/02/17	28/02/17
Imp. Gráficas	9/02/17	9/03/17
GEP	22/02/17	29/03/17
Auditorías/Alarmas/Eventos	10/03/17	7/04/17
BackUp/About/Cuentas	11/04/17	28/04/17
Registro contínuo	11/04/17	5/06/17
Mochila	6/06/17	26/06/17
Interfaz E/S	6/06/17	26/06/17
Port a Android	16/01/17	30/06/17
Imp. Gráficas	4/07/17	31/07/17
Comprobación objetivos	27/06/17	28/06/17
Revisión Server	29/06/17	7/07/17
Revisión Bugs	29/06/17	18/07/17
Documentación provisional	30/03/17	7/06/17
Memoria Final	8/06/17	19/06/17
Presentación	13/06/17	29/06/17
Gráficas Android	5/07/17	15/08/17
Seguridad servidor	5/07/17	15/08/17
Mantenimiento	5/07/17	29/08/17

Vista General del Diagrama Gantt de Seguimiento

Anexo 4: Esquema UML de las clases de Control expandido

Esquema UML de las clases de datos generado con simpleUML de Android Studio