Universidad Zaragoza

TRABAJO FIN DE GRADO

Desarrollo de una aplicación cliente-servidor para dispositivos móviles para la monitorización y gestión de baterías de coches híbridos en una flota de taxis

Autor:

Javier Meneses Lobera

Director:

Luis Monzón Jaso

Ponente:

María Canales Compés

Escuela de Ingeniería y Arquitectura

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Septiembre de 2014

Desarrollo de una aplicación cliente-servidor para dispositivos móviles para la monitorización y gestión de baterías de coches híbridos en una flota de taxis

RESUMEN

Este Trabajo Fin de Grado consiste en desarrollar una aplicación en terminales móviles Android para una flota de taxis híbridos que permita tanto la monitorización del estado de la batería del vehículo como la reserva en la central de baterías para su reemplazamiento. El usuario con esta aplicación tendrá acceso al estado de la batería del coche, una estimación de la autonomía de la misma, la distancia y el tiempo a las que se encuentran la central de baterías (situada en la cooperativa de taxis de Zaragoza) y una interacción con el servidor para hacer la reserva de una batería cargada. Para realizar este trabajo se necesita implementar una aplicación nativa en Android, configurar un servidor a través de ficheros PHP para que guarde la información de monitorización de la batería en una base de datos MySQL y procese y gestione la reserva de baterías y programar placas Arduino para poder comunicar las baterías con los teléfonos y el servidor.

El objetivo es conseguir una aplicación estable, usable e intuitiva para el conductor, a la par que potenciar su uso adecuado.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años se ha podido observar un enorme crecimiento de la utilización del teléfono móvil a nivel mundial, tanto de los más pequeños gracias a las plataformas de juegos que ofrecen los diferentes sistemas operativos de móviles como los más mayores con teclados, pantallas e interfaces más aptas para para que se animen a utilizar esta tecnología, pasando por aquellos con todo tipo de dificultades a los que estos sistemas son capaces de adaptarse para satisfacer cualquier necesidad. Gracias a estos dispositivos, junto con las ya famosas redes 3G y 4G se ha revolucionado las comunicaciones en todo el mundo, pudiendo acceder a un nivel de información sin precedentes de cualquier tipo de contenido y en cualquier lugar, y pudiendo establecer una comunicación oral o escrita con cualquier persona del mundo en cuestión de segundos. Por si fuera poco, un teléfono móvil dota actualmente de tales recursos que puede realizar, aparte de las comunicaciones ya citadas, infinidad de procesos, que son agrupados en aplicaciones, y nos permitan, por ejemplo, hacer fotos, ubicarnos en un mapa, tomar notas o ver la televisión.

1.1 Motivación y objetivo final

La idea de realización de este Trabajo Fin de Grado nace de la empresa ZEVNA, más concretamente de su departamento de Ingeniería Aplicada, en el que Luis Monzón Jaso, director del presente Trabajo, definió los objetivos que a continuación se plantean. Uno de los campos que la empresa ZEVNA oferta es la conversión de coches híbridos a híbridos enchufables. Esta conversión implica introducir en el vehículo una batería que se encargue suministrar la energía necesaria al coche para que éste no consuma durante el máximo tiempo posible de su motor de combustible. Desarrollando este concepto nace el proyecto Kit Prius Plug-In: MoreTaxi, finalista del Startup Pirates 2013 y mentorizado durante 4 meses en el IV semillero de empresas Zaragoza Activa. Este proyecto se enfoca al sector de transporte urbano de viajeros, y será comercializada bajo la iniciativa, spin-off de ZEVNA de próxima creación, MoreTaxi. Esta aplicación hace posible este proyecto, y su desarrollo es este Trabajo Fin de Grado. Por lo tanto, e introduciendo los objetivos, tanto del proyecto de ZEVNA como los del trabajo, el propósito es conseguir una aplicación móvil que controle y gestione las baterías instaladas a los taxis, proporcionando a ZEVNA la información de su monitorización y control y gestión de las reservas.

1.2 Materiales y herramientas utilizadas

Para la realización de este trabajo se utilizaron los siguientes materiales y herramientas:

- **Android SDK [1]:** Este kit de desarrollo de Software permitirá la programación de la aplicación nativa.
- **Eclipse IDE [2]:** Se utiliza como entorno de desarrollo de la aplicación.
- **Java [3]:** Como lenguaje de programación complementario a Android.
- **PHP [4]:** Lenguaje de programación utilizado para los ficheros alojados en el servidor.
- **MySql [5]:** Se utiliza para albergar la base de datos y realizar las consultas necesarias de gestión.
- **FileZilla [6]:** Como pasarela para transmitir archivos al servidor a través del protocolo SFTP.
- **Adobe Photoshop CS6 [7]:** Software para la realización del diseño gráfico de la aplicación.
- **Arduino [8]:** Como lenguaje de programación para las placas Arduino presentes en el trabajo.
- **Raspberry Pi [9]:** Se utiliza como servidor durante el desarrollo de la aplicación.
- **Arduino UNO, Bluetooth Kit y Ethernet Shield [10]:** 2 placas Arduino UNO, una con cada kit para realizar las comunicaciones con el teléfono.
- **Xiaomi Red Rice:** Teléfono Android utilizado para realizar la aplicación.

- **HTC Evo 3D:** Teléfono Android utilizado para realizar pruebas del correcto funcionamiento de la aplicación.

1.3 Organización de la memoria

La memoria está organizada de la siguiente forma:

- En el **Capítulo 1** se presenta el Trabajo Fin de Grado, se define cómo ha surgido y se describen los materiales utilizados.
- En el **Capítulo 2** se definen los requisitos del trabajo y sus objetivos, además de situar la aplicación en el contexto actual.
- En el **Capítulo 3** se explica el funcionamiento de la aplicación móvil y la relación de sus diferentes servicios.
- En el **Capítulo 4** se desarrolla el funcionamiento del servidor y los servicios que realiza.
- En el **Capítulo 5** se explica el desarrollo de las placas Arduino y sus funciones.
- En el **Capítulo 6** se detallan las pruebas realizadas para garantizar el funcionamiento del trabajo completo.
- En el **Capítulo 7** se comentan las conclusiones finales y líneas futuras.

En el final de la memoria se pueden encontrar diferentes Anexos indicados a continuación:

- En el **Anexo A** aparecen los diagramas correspondientes a la aplicación realizada que facilitan la comprensión de su funcionamiento.
- En el **Anexo B** se ilustran los diagramas pertenecientes a los diferentes servicios que ofrece el servidor del trabajo.

- En el **Anexo C** se detallan los diagramas de navegación de las placas Arduino y sus correspondientes módulos.
- En el **Anexo D** se explica detalladamente cómo se realiza la estimación del porcentaje de las baterías.

Capítulo 2

Análisis previo y requisitos

2.1 Visión general

Con el uso de esta aplicación se pretende por un lado ofrecer al conductor del vehículo información sobre el estado de su batería, la autonomía de ella y datos de tiempo y distancia hasta el lugar donde él podría recoger y reemplazarlas. Para ello se debe realizar una aplicación intuitiva, fácil y rápida que permita al conductor interactuar con la aplicación e interpretar los resultados de una forma que le resulte cómoda. Por lo tanto, se desglosa a continuación cómo está dividida la aplicación:

- Una parte está dedicada al proceso de Login/Password. De esta forma se puede identificar qué conductor se está conectando a la aplicación para poder gestionar y garantizar las reservas.
- Otra parte se centra en la conexión con la batería del coche del conductor. Así se podrá conocer en la monitorización los datos de la batería y del vehículo. El conductor selecciona el dispositivo al que quiere conectarse e introduce un pin para autenticarse con el vehículo.

- Una última parte de interfaz gráfica de los resultados de monitorización de la batería y posibilidad de reserva de baterías de la central mediante un diálogo de ventanas.

Esta aplicación forma parte del trabajo general, cuyo diagrama está ilustrado en la Figura 2.1. En ella se puede ver que la aplicación está comunicada por Bluetooth con un Arduino Bluetooth, que lee valores de la batería del coche y por Internet a un servidor que gestionará todos los servicios que requiere la aplicación. Por otro lado el servidor está conectado a un Arduino que mediante Ethernet enviará los cambios de baterías producidos en el cargador de baterías en tiempo real, actualizando la base de datos. Además la aplicación incluye una geolocalización GPS que permite al usuario conocer el tiempo y distancia hasta el lugar de recogida de las baterías.

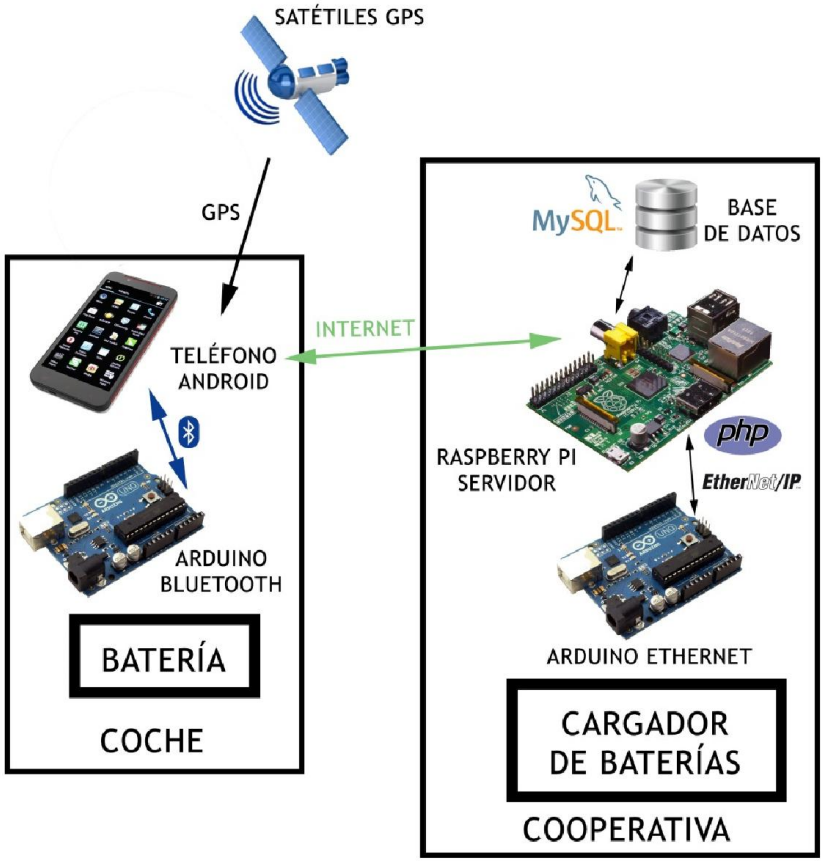

Figura 2.1 Diagrama general del proyecto.

2.2 Público objetivo

Este trabajo está dirigido al sector público del transporte, en este caso a la flota de taxis de Zaragoza, ya que la idea del proyecto se produjo en la capital aragonesa y el trabajo se realiza en la Escuela de Ingeniería y Arquitectura. Como se puede observar, en los últimos años se ha producido un crecimiento considerable de los coches híbridos en la flota de taxis, especialmente el modelo Toyota Prius. Además, los conductores visitan diariamente la cooperativa. Por ello, este trabajo se orienta y adapta a ellos al instalarles una batería que les permita trabajar sin consumir apenas combustible y facilitarles las reservas de baterías a través de la aplicación. Los conductores dispondrán de un cargador de baterías para poder entregar las baterías que han consumido y recogerlas cargadas para seguir trabajando. Para llevar un control del estado de las baterías se introduce un sistema de monitorización.

2.3 Estado del arte

Si bien es cierto que una aplicación tan concreta para el anteriormente especificado público objetivo es innovadora, en este apartado se describe otras aplicaciones de monitorización o de gestión de reservas que pueden inspirar en el desarrollo de la aplicación. A continuación se detallan una serie de aplicaciones disponibles en el Play Store que han podido inspirar a cómo realizar este trabajo:

- **GSam Battery Monitor:** Esta aplicación permite monitorizar el estado de la batería del teléfono, indicando cuál es el gasto de cada proceso o aplicación.

  https://play.google.com/store/apps/details?id=com.gsamlabs.bbm

- **Battery Widget Reborn (BETA):** Esta aplicación también permite monitorizar el estado de la batería del teléfono, aunque nos lo presenta en forma de Widget en la pantalla principal del teléfono.

  https://play.google.com/store/apps/details?id=net.hubalek.android.reborn.beta

- **Facebook:** Esta famosa aplicación sirve de inspiración para el proceso de Login y mantenimiento de sesión, aunque pueden servir muchas otras, cualquiera de las redes sociales, el trabajo se centra en especial en esta opción.

  https://play.google.com/store/apps/details?id=com.facebook.katana

Para la gestión de reservas, se podría destacar el servicio que se ofrece en la inspección técnica de vehículos, concretamente el de cita previa online, en la que se selecciona una fecha y hora deseada y se observa una propuesta lo más aproximada a nuestra petición, la que se puede confirmar o rechazar de la misma forma que en este trabajo.

http://www.itvcita.com/

Una vez se ha visto otras aplicaciones o servicios del mercado, se procede a realizar el trabajo propuesto juntando todos estos sistemas en una aplicación que englobe los procesos de autenticación, monitorización y gestión de reservas.

2.4 Requisitos del trabajo

Los requisitos del Trabajo Fin de Grado vienen indicados y detallados a continuación:

- **RF1:** La aplicación podrá monitorizar los datos enviados por la placa Arduino y enviarlos al servidor para su posterior análisis.
- **RF2:** La aplicación podrá gestionar las reservas de las baterías, permitiendo elegir al usuario la hora deseada de recogida y proponiendo una respuesta adecuada según las necesidades del conductor.
- **RF3:** La aplicación ofrecerá a través de la posición geográfica del teléfono información sobre la distancia y tiempo restante a la cooperativa de taxis según el tráfico actual que Google elija.
- **RF4:** La aplicación se iniciará con un proceso de autenticación del usuario evitando que cualquier persona no deseada pudiera intervenir en la aplicación.
- **RF5:** La aplicación será capaz de reconocer dispositivos Bluetooth Arduino y conectar con ellos a través de un PIN de autenticación.
- **RF6:** La aplicación mostrará una estimación aproximada de autonomía de la batería para guiar al conductor en el proceso de reserva de la batería, sugiriéndole la hora de recogida de una batería nueva cuando la actual se vaya a agotar.
- **RF7:** El servidor almacenará en una base de datos los datos recogidos de la monitorización de todos los dispositivos que estén conectados a las baterías y se encuentren circulando.
- **RF8:** El servidor será capaz de gestionar las diferentes peticiones de reserva de los usuarios garantizando la exclusión

mutua entre los mismos, y ofreciéndoles un horario acorde a sus necesidades.

- **RF9:** El servidor será capaz de garantizar la autenticación de los usuarios de la aplicación a través de la comparación del hash de la contraseña.
- **RF10:** La placa Arduino que incorpora el módulo de *Ethernet* [13] será capaz de simular un comportamiento del cargador de baterías y enviar al servidor (a través del router donde se conecte) la información de baterías que están siendo cargadas y la hora estimada de recogida.
- **RF11:** La placa Arduino que incorpora el módulo *Bluetooth* [14] podrá simular el comportamiento de una batería y enviar a través de Bluetooth los voltajes de las celdas de la batería de cara a su monitorización.

2.5 Plataforma de desarrollo

La plataforma de desarrollo en la que se desarrolla la aplicación es Android, ya que este sistema operativo móvil es el más común entre los usuarios del público objetivo, y con esta elección se abarca más mercado. Para el diseño del servidor se ha optado por una base de datos MySQL ya que permite hacer consultas SQL, se garantiza la exclusión mutua de peticiones y se puede importar o exportar bases de datos a Access o Excel fácilmente. La inclusión de ficheros PHP permite implementar servicios *HTTP* [15] GET a través de objetos *JSON* [16] (formato de JavaScript para intercambio de datos) que hace posible los requisitos de la aplicación. Las placas Arduino se han elegido por su facilidad en la programación y por ser capaces de leer voltajes de la batería a través de sus entradas analógicas.

Capítulo 3

Desarrollo de la aplicación

3.1 Planteamiento general

Cuando se desarrolla una aplicación móvil, en este caso para el lenguaje Android, hay que estructurarla en las diferentes pantallas o *Activities* que la conforman. Esta aplicación se divide en 3 *Activities*, de las que 2 se utilizan para autenticar al usuario y al taxi, mientras que la restante, la principal, muestra los diferentes servicios que realiza la aplicación con el servidor o con el Arduino Bluetooth.

Dentro del proyecto de Eclipse, la aplicación está dividida en paquetes que contienen clases. Estos paquetes se organizan de la siguiente forma:

- **com.javimeneses.MoreTAXI:** En este paquete se incluyen las clases pertenecientes a las *Activities* nombradas anteriormente. La clase *BeginActivity.java* incluye el inicio de sesión, desarrollado en el apartado 3.2.1. La clase *BluetoothActivity.java* incluye la búsqueda de dispositivos, en este caso, el Arduino conectado al Taxi, y se comenta en el apartado 3.2.2. Finalmente se encuentra la clase *MainActivity*, que contiene la pantalla principal de la aplicación, en la que se producen los servicios de monitorización, autonomía, geolocalización y gestión de reserva.

- **com.javimeneses.MoreTAXI.conexionesbt:** En este paquete se incluyen las clases necesarias para realizar el hilo de conexión del Arduino Bluetooth y su posterior transferencia de datos, previniendo de posibles errores de conexión o de transferencia.
- **com.javimeneses.MoreTAXI.interfazbt:** En este paquete se encuentra la interfaz *Bluetoothuser*, declarada para que se implementen los métodos *Error* y *Result* necesarios en la transferencia realizada por las clases del paquete mencionado anteriormente.
- **com.javimeneses.MoreTAXI.servicios:** En este paquete se incluyen las clases que implementan un servicio *HTTP GET* genérico, de forma que mediante parámetros de sus métodos se pueda introducir la URL y las variables necesarias en cada caso.
- **com.javimeneses.MoreTAXI.globals:** En este paquete se almacenan dos clases. La clase *Globals.java* almacena las posibles variables globales del proyecto, como pueden ser el dispositivo al que conecta, el usuario o el taxi para poder acceder a ellos desde cualquier servicio ejecutado en background. La clase *Operaciones.java* guarda las funciones o métodos estáticos utilizados a lo largo de la aplicación.

En los siguientes apartados se hace referencia a estas clases descritas. Los diagramas de navegación de la aplicación se encuentran en el **Anexo A**. El código completo de la aplicación se puede encontrar en el CD adjunto a la memoria.

3.2 Funcionamiento de la aplicación

Cuando la aplicación se inicia lo primero que se realiza es una búsqueda de variables guardadas, mediante la clase *SharedPreferences*. Esta clase permite almacenar valores incluso cuando la aplicación está cerrada. En este caso las variables interesantes a almacenar son el último usuario registrado y la hora de la última reserva. Una vez recogidos estos valores se inicia la primera pantalla de la aplicación, *BeginActivity.java*, en la que comienza el proceso de login o inicio de sesión.

3.2.1 Inicio de sesión

La primera *Activity* de la aplicación es *BeginActivity*, y su propósito principal es el inicio de sesión, que se ilustra en la Figura 3.1. En ella se puede observar una interfaz muy simple con los siguientes elementos: 2 casillas para rellenar los datos de usuario y contraseña, llamadas *EditText*, y un botón de la clase *ImageButton* que ejecuta el servicio de Login, descrito posteriormente en el capítulo 4. Si anteriormente se ha leído en *SharedPreferences* que un usuario ha realizado el inicio de sesión correctamente, se introduce ese usuario en la casilla *Nombre de usuario*, de forma que facilite al usuario en tiempo ese inicio de sesión. La casilla de *contraseña*, por seguridad, se deberá introducir en cada inicio de sesión.

Cuando se hace click en el botón de *inicio de sesión* se comprueba primero que las dos casillas no están vacías. Después se calcula el hash de la contraseña con el algoritmo *SHA-1* [20] y se realiza el servicio de Login, con el método *login* de la clase definida *ServicioApp*. Este método tiene como parámetros dos cadenas de caracteres: el usuario y el hash de la contraseña. La siguiente pantalla que aparece cuando el inicio de sesión es correcto es

BluetoothActivity, en la que se produce la búsqueda de dispositivos Bluetooth. Al acceder a esta pantalla se guarda el usuario conectado a *SharedPreferences*. Si el inicio de sesión no es correcto, se muestra con un mensaje de alerta y no se inicia la nueva *Activity*.

Figura 3.1 Apariencia de la pantalla de inicio de sesión.

3.2.2 Búsqueda de dispositivos

Una vez iniciada la sesión aparece la pantalla *BluetoothActivity*, encargada de realizar una búsqueda de dispositivos Bluetooth a su alrededor y permitiendo elegir a qué dispositivo conectar. La apariencia de esta *Activity* se ilustra en la Figura 3.2. En ella se puede observar en la parte superior el usuario con el que se ha iniciado la sesión y un botón que accede a los ajustes de la aplicación. Estos ajustes están comentados en el apartado 3.2.7 de la memoria. En el centro de la pantalla se encuentra la lista de dispositivos Bluetooth que se actualiza

dinámicamente en el momento en que un nuevo dispositivo es encontrado, gracias a la clase *ListView* y al método *startDiscovery()*. Estos elementos de la lista son seleccionables, aunque al hacer click en uno de ellos se comprueba que el nombre del dispositivo es de la forma TAXIxxxx, siendo estos últimos caracteres 4 dígitos. Si el dispositivo seleccionado no cumple esta condición se muestra un mensaje de alerta indicando que el tipo de dispositivo no es válido. Cuando se selecciona un dispositivo correcto se realiza un cambio de pantalla mediante la clase *Intent* a *MainActivity*, que es la pantalla principal donde se desarrollan el resto de servicios de la aplicación.

Figura 3.2 Apariencia de pantalla de búsqueda de dispositivos.

3.2.3 Monitorización de la batería

La monitorización de la batería del coche es uno de los servicios que se ejecutan desde la pantalla principal, *MainActivity*, aunque son procesos que corren en segundo plano para ofrecer una mejor respuesta al proceso principal, el interfaz con el usuario. En la Figura 3.3 se observa un círculo en el centro de la pantalla que será el elemento visual que muestre el estado de la conexión o el porcentaje de la batería en el caso de una lectura correcta de los datos.

Para realizar una lectura de los datos de la batería, en cuanto se inicia la *Activity* se llama al método *run()* de la clase *ConexionArduino*, que lo que hace es ejecutar un hilo que se conecta al dispositivo Bluetooth seleccionado. Si este dispositivo no ha sido aún enlazado al teléfono se tendrá que escribir un código PIN que se define en el Arduino, descrito en el capítulo 5 de la memoria. El hilo le envía a éste una petición de conexión y escucha lo que le manda periódicamente. Cuando recibe una trama se llama al método *comprobar()*, que verifica que la composición de ésta es correcta. Una vez se reciba la trama verificada el hilo se termina volviendo como resultados el *array* de valores útiles de la trama. Estos valores ya son tratados en el programa principal, *MainActivity*.

Se reciben los voltajes de cada celda y el identificador de la batería, y se lanza con estos datos el servicio de monitorización, explicado en el capítulo 4.3.2. Este servicio es lanzado cada minuto, de forma que al lanzarlo se obtiene el minuto actual del móvil y se pone como condición que ese valor cambie para volver a lanzar el servicio.

Con los datos recibidos de las celdas, siendo voltajes mapeados con una resolución de 1024 valores entre 2,5V y 4,25V, se calcula el

porcentaje aproximado de la batería con la función *porcentajeBateria()*, y explicado detalladamente en el Anexo D.

Una vez conseguido el porcentaje asociado a esos valores enteros, este valor se introduce como parámetro en la función *calculaColor()*, que devuelve el valor hexadecimal proporcional siendo 100% verde y 0% rojo.

Para finalizar se dibuja el círculo, gracias a la clase *Canvas*, introduciendo el porcentaje en el centro y pintándole el color calculado previamente. Una vez dibujado, y pasados 10 segundos, se ejecuta de nuevo el hilo Bluetooth para obtener frecuentemente los valores de la batería. Si no se detecta el dispositivo el propio hilo se detiene y en vez de ejecutar el método *Result()* ejecuta *Error()*, y el círculo se mostrará sin porcentaje y en estado desconectado.

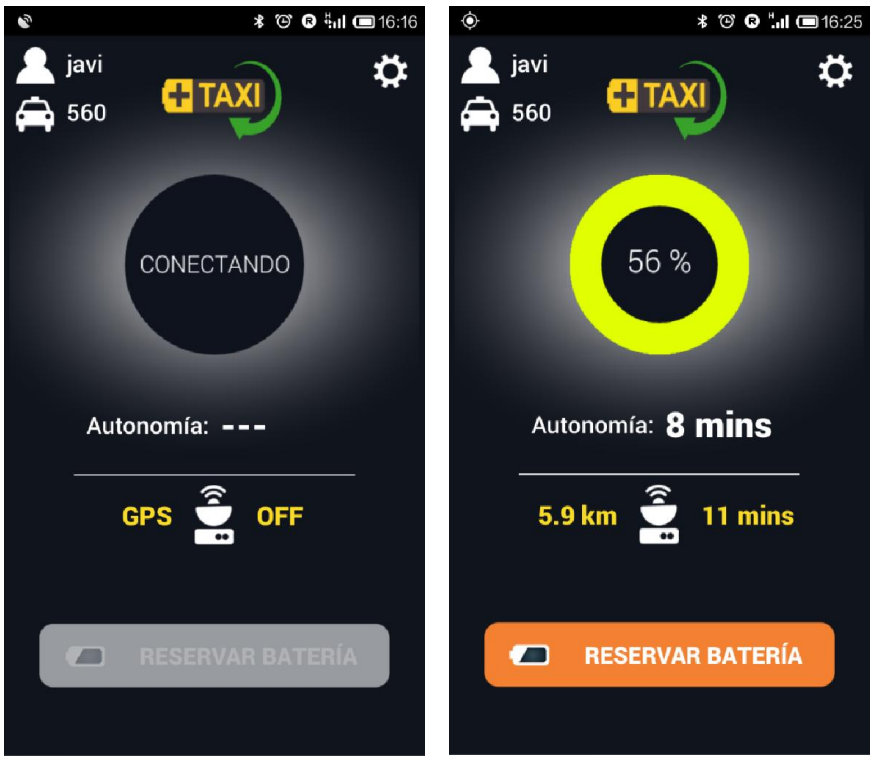

Figura 3.3 Apariencia de la pantalla *MainActivity*.

3.2.4 Autonomía de batería

Una de las informaciones que aparece en el interfaz móvil del usuario es la indicación de la autonomía estimada de la batería. Ésta está presente también en la misma *Activity*, como se puede observar en la Figura 3.3. El cálculo de la autonomía se realiza interpolando los distintos valores del porcentaje restante leído con el tiempo que se ha tomado esa lectura. De esta forma el conductor puede conocer una estimación de la autonomía restante de la batería.

3.2.5 Geolocalización hasta la cooperativa

Otro de los servicios que se realizan en *MainActivity* es la geolocalización del dispositivo y su cálculo de distancia y tiempo hasta la cooperativa. Gracias a *Google Maps* se puede consultar de forma similar a los servicios de la aplicación, descritos en el capítulo 4, qué distancia y tiempo hay hasta un punto determinado del mapa. El punto elegido es la cooperativa de taxis, donde residirá en la puesta en práctica del proyecto los cargadores con las baterías. El servicio se realiza a través de un *HTTP GET* al servidor de Google, introduciendo como parámetros latitud y longitud de origen y de destino y el modo en *driving* para indicar el tiempo restante cuando se va conduciendo. Se recibe un objeto con la notación *JSON* de varios datos, de los que se elige la distancia y el tiempo y se muestra por pantalla. El icono de GPS que se puede ver en la Figura 3.3 es un botón que nos lleva al menú de configuración del GPS de Android, de forma que se puede prescindir independientemente del servicio debido a tener la batería baja o no estar interesado en la información.

3.2.6 Reserva de la batería

Otro de los objetivos del proyecto es la gestión de reserva de baterías. Ésta empieza cuando se pulsa el botón naranja situado en la parte inferior de la pantalla principal, como se ha visto anteriormente en la Figura 3.3. Una vez pulsado, aparece un mensaje de alerta que permite elegir una hora de reserva deseada, como se ve en la Figura 3.4.

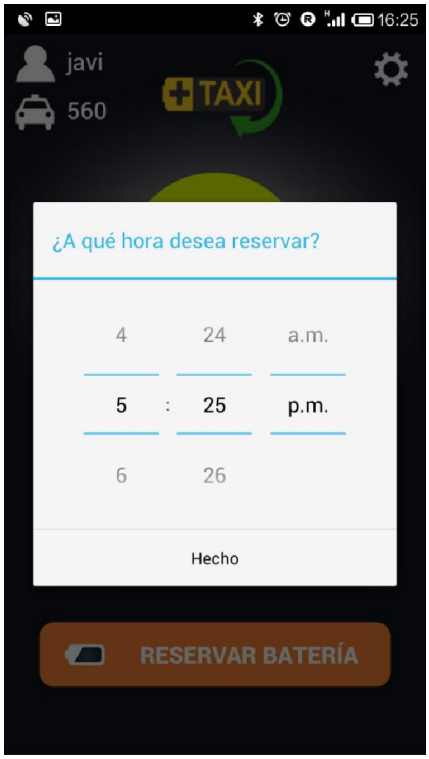

Figura 3.4 Apariencia del mensaje de elección de hora.

Cuando se elige la hora deseada y se acepta se lanza el servicio de pre-reserva, que está descrito en el capítulo 4.3.3 de la memoria. Como respuesta de este servicio se recibe la hora a la que el slot de batería más próximo estará disponible, y lo indica con otro mensaje de alerta que se puede observar en la Figura 3.5. Esta alerta permite confirmar la reserva o por el contrario rechazarla, ya que la hora del siguiente slot disponible no tiene por qué ser la misma que la elegida por el usuario. Si se confirma la reserva, se realiza el servicio de

reserva, descrito también en el capítulo 4.3.3 de la memoria. Con la respuesta afirmativa se cambia el botón de aspecto, mostrando el slot y la hora de reserva y no reaccionando a los clicks del usuario. Una reserva confirmada se ilustra también en la Figura 3.5.

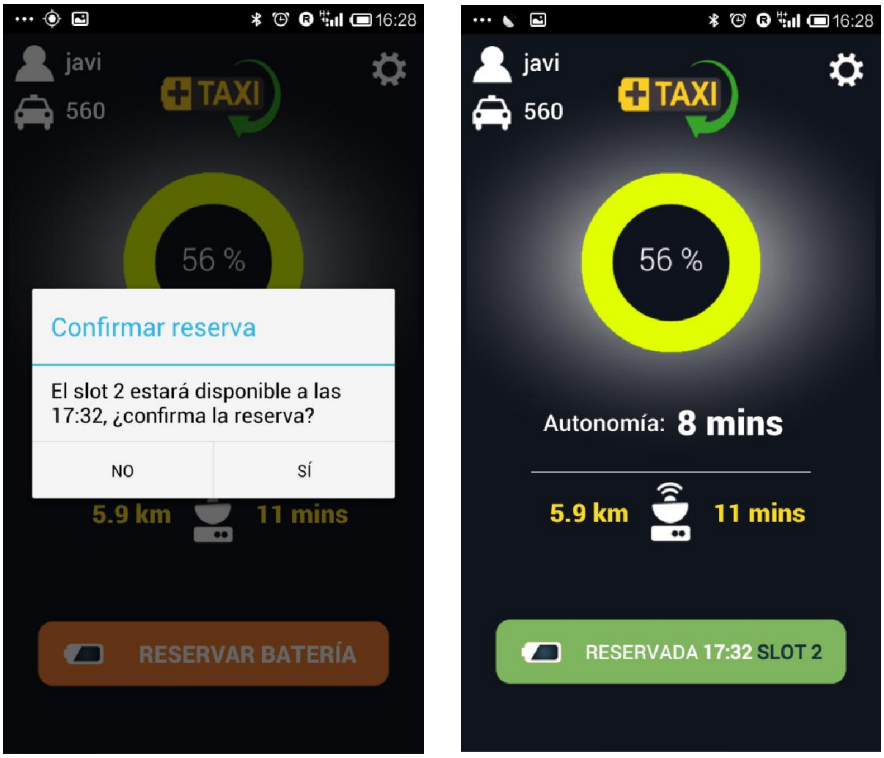

Figura 3.5 Apariencia de la gestión de reservas de una batería.

3.2.7 Ajustes

Finalmente se describe los ajustes de la aplicación, disponibles desde las pantallas de búsqueda de dispositivos y principal, ilustrados anteriormente en la Figura 3.2. La primera opción es cerrar sesión, y realiza un cambio de pantalla hasta *BeginActivity*, donde nos muestra el proceso de inicio de sesión. El usuario almacenado se borra para que otro usuario nuevo pueda iniciar la sesión. El otro posible ajuste de la aplicación es la búsqueda de otro

dispositivo, ya sea por un error al elegir el dispositivo o por necesidad. Este ajuste realiza un cambio de pantalla o *Intent* a *BluetoothActivity*, en la que se actualiza la lista de dispositivos y permite la elección de uno de ellos.

Capítulo 4

Desarrollo del servidor

4.1 Información general

Junto con la aplicación, la pieza más importante de este proyecto es el servidor, éste debe ser capaz de recibir las distintas peticiones que vengan tanto de la aplicación como del Arduino Ethernet a través de Internet. También deberá interactuar con la base de datos para después devolverle a los dispositivos el resultado de las sentencias que devuelve ésta.

En este caso se ha optado por realizar estas comunicaciones a través de ficheros PHP ya que aportan mayor facilidad tanto para programar las peticiones HTTP GET del teléfono y gestionar las respuestas como para interactuar con la base de datos.

Durante el periodo de la realización del proyecto se ha trabajado con una Raspberry Pi tipo B. En este dispositivo se ha instalado Raspbian como sistema operativo, y sobre él se ha configurado un servidor con Apache server, MySQL, PHP y FileZilla Server. Además se le ha dado acceso desde el exterior, configurando el router adecuadamente para permitir los protocolos HTTP, *FTP* [17] y *SSH* [18]. Conectada a éste desde una IP interna fija, se registró en un servicio gratuito de *Dynamic DNS* [19], ya que cada reinicio del router suponía un cambio de IP externa. De esta forma se actualiza

la dirección válida cada 5 minutos y nos previene de caídas prolongadas o cambios manuales de IP en cada reinicio del router. En la Figura 4.1 se observa un esquema general del servidor con los diferentes servicios que ofrece. Los diagramas detallados de cada servicio se encuentran disponibles en el **Anexo B** de la memoria.

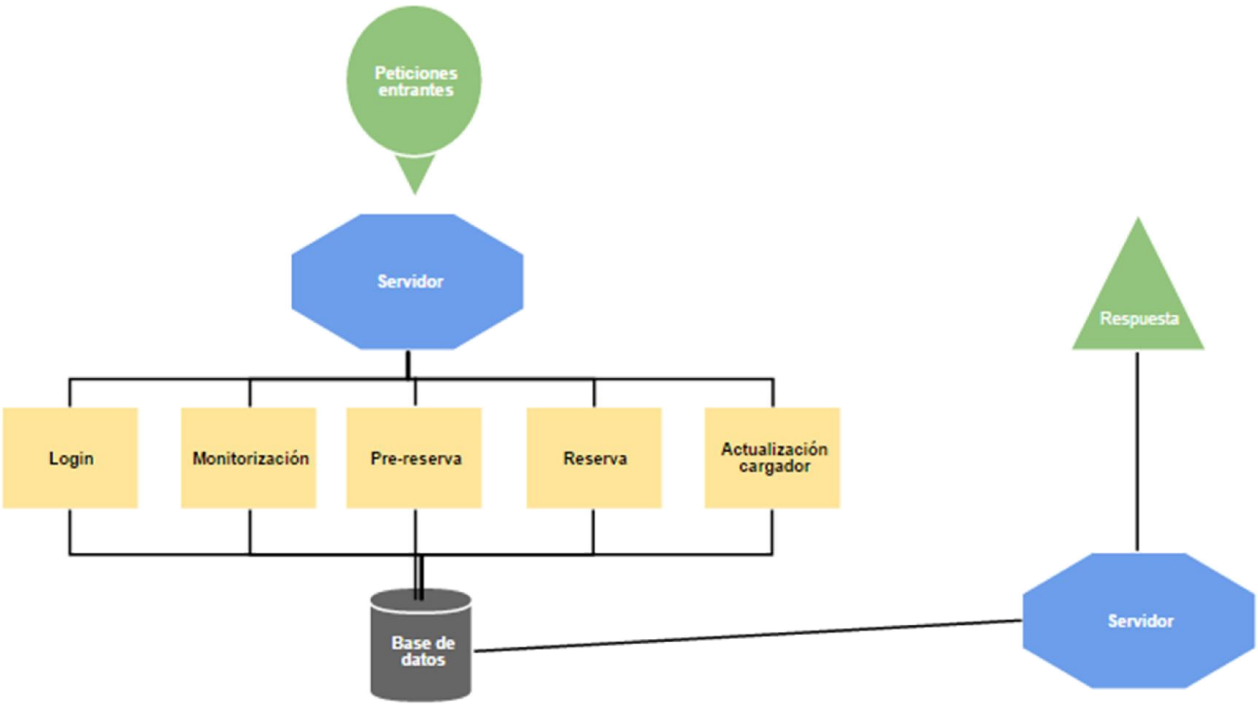

Figura 4.1 Diagrama general del servidor.

4.2 Funcionamiento global

El objetivo global del servidor es almacenar la información de las monitorizaciones de las distintas baterías que estén conectadas a la aplicación y gestionar adecuadamente las reservas que se produzcan por parte de los conductores. Para ello habrá que contar también con una base de datos que guarde dicha información, y

consultar ésta desde el servidor. Dentro del Apache Server instalado se encuentran los ficheros PHP que hacen esto posible a través de una petición HTTP GET cuando se inicia una petición al servidor y una respuesta de tipo objeto JSON.

Cuando se recibe un paquete HTTP GET a un fichero PHP determinado se realiza una extracción de los parámetros de la URL enviada y se guardan en variables. En este punto se realiza la consulta SQL con la base de datos y con los resultados que se obtienen se construye el objeto JSON que se envía como respuesta.

De esta forma se obtiene un método sencillo para realizar servicios muy variados, además de ser la forma más empleada de realizar este tipo de peticiones. Como ejemplo y ya citado anteriormente, así se realizan las peticiones de distancia y tiempo a la cooperativa y se recibe la respuesta del servidor de Google de la misma forma.

4.2.1 Comunicación con la base de datos

Para comunicarse con la base de datos el fichero se conecta primero con la base de datos a través de la función *mysql_connect*, en la que se indica la dirección (en este caso será *localhost* ya que comparten la misma ubicación física), el nombre de usuario y la contraseña. Después se llama a la función *mysql_select_db*, en la que se selecciona la base de datos MoreTAXI.

Una vez el servidor está conectado a una base de datos específica, se realiza una sentencia SQL con lenguaje MySQL, y se guarda en una variable *String*. A través de la función *mysql_query* se realiza la consulta, y se incluye como parámetros la sentencia y la base de datos seleccionada nombrada anteriormente. Para recoger el

resultado de la sentencia se recurre a la función *mysql_result* y se guardan en las variables oportunas.

Finalmente, para construir la respuesta se crea un array en el que se almacenan las variables o parámetros que interesen y se realiza un echo del array codificado con la función *json_encode*.

4.3 Servicios del servidor

Una vez explicado el funcionamiento básico del servidor y cómo se realizan las peticiones con la base de datos en este apartado se concreta cuáles son los servicios concretos que soporta el servidor y cómo los gestiona.

El trabajo consta de 5 servicios diferentes, sintetizados en los 3 siguientes subapartados. Los servicios de login, de monitorización, de pre-reserva y de reserva son los que están relacionados con la aplicación móvil, mientras que el de actualización de cargador se implementa en el Arduino Ethernet, detallado en el apartado 5 de la memoria. Estos 3 últimos servicios están relacionados y agrupados por ello con la gestión de reservas de baterías. Todos ellos tienen características similares aunque se diferencian en los parámetros, la sentencia y la respuesta. A continuación se definen todos los servicios que presta el servidor de este trabajo.

4.3.1 El servicio de login

Como se ha podido ver anteriormente, en la primera pantalla descrita en la aplicación se observa un proceso de Login. Cuando se pulsa el botón iniciar sesión se procede a este servicio. Los parámetros incluidos son usuario y contraseña, siendo ésta última un hash calculado con el algoritmo *SHA-1*. De esta forma se evita posibles robos de contraseña con ataques al servicio o a la base de datos.

Para conectarse al servidor se envían los parámetros al fichero *acces.php*, que se obtienen mediante el método GET. Una vez guardado el usuario y el hash de la contraseña se ejecuta la función *login*, que se encuentra en el fichero *funciones_bd.php*. Ésta ejecuta una sentencia SQL de tipo SELECT en la que se busca cuántos resultados satisfacen tener ese usuario con ese hash asociado. En el caso de no obtener ningún resultado se envía mediante un objeto *JSON* la variable *logstatus* con valor 0, y en caso afirmativo *logstatus* cambiará su valor a 1.

La tabla encargada de guardar la información de este servicio es *usuarios*, en la que se guardan dos variables: el nombre de usuario y el hash de la contraseña. Además se aconseja que sea esta misma tabla la que guarde información personal y de contacto de los diferentes usuarios.

Como complemento a este servicio también se implementa la combinación de ficheros *adduser.php* y su asociado *adduser.html*. Esto permitirá el ingreso de nuevos usuarios a través de un sencillo formulario en el que se utiliza la función *adduser* dentro de *funciones_bd.php*. En este caso la sentencia SQL es de tipo

INSERT, dentro de la misma tabla usuarios, con una previa comprobación de que ese usuario no está ya registrado. Esta comprobación se realiza mediante la función *isuserexist*.

Con este complemento se puede dar de alta fácilmente a usuarios a través de cualquier dispositivo con conexión a Internet o dentro de la intranet del servidor.

4.3.2 El servicio de monitorización

Uno de los objetivos principales del trabajo es la monitorización de los niveles de batería. Una vez que los diferentes voltajes de las celdas se convierten en enteros y se envían por Bluetooth del Arduino al terminal móvil se produce otra emisión de los datos hasta el servidor.

Las variables que engloba este servicio son los 16 enteros equivalentes al voltaje de cada celda, el número de serie de la batería, el identificador del taxi y el nombre del usuario que lo envía.

El fichero que guarda las variables y ejecuta la sentencia SQL asociada a este servicio es el de *monitorizacion.php*. Se realiza un GET de los parámetros y se almacenan los valores. A continuación se realizan 2 sentencias de tipo SELECT que sirven para obtener el identificador de las diferentes tablas con las que se trabaja. Por ejemplo, el usuario *admin* equivale en la tabla usuarios al identificador número 1. Trabajando con esto, se consigue optimizar las tablas y tenerlas relacionadas, aunque visualmente sin relacionar no parezcan tan intuitivas. Por lo tanto, con las sentencias se consiguen los identificadores del número de taxi y del nombre de

usuario. A continuación se envía la sentencia SQL que insertará una nueva fila en la tabla *monitorización* de la base de datos *MoreTAXI*, por lo que el tipo de sentencia será INSERT. En ésta se incluyen los 16 valores de celda y los 3 índices de los datos mencionados anteriormente.

En la base de datos se incluye un sello de tiempo que se actualiza automáticamente cuando se inserta la fila. Estos datos sirven para analizar los tiempos de descarga de las baterías, detecciones de celdas dañadas y diferentes patrones de consumo de los taxis y usuarios.

4.3.3 La gestión de reservas

El otro objetivo principal del trabajo es saber gestionar bien las reservas, y para ello en este punto se describen 3 servicios diferentes que funcionan mediante los ficheros *prerreserva.php*, *reserva.php* y *updateCargador.php*.

El servicio de pre-reserva se utiliza para garantizar a los usuarios la batería más próxima a la hora que ellos elijan. Así se evita que 2 o más usuarios puedan reservar la misma batería, ya que aunque MySQL garantiza que no haya concurrencia entre peticiones, este tiempo de posible conflicto se incrementa al tener que hacer una confirmación de la reserva.

Los parámetros utilizados para pre-reservar una batería son el número de licencia de taxi, el nombre de usuario y la hora que se quiere hacer la reserva. Este último dato es un entero largo que

indica el número de segundos desde el 1 de enero de 1970, una unidad común en formatos de sellos de tiempos.

Como ya se describe en el servicio anterior se deben obtener los índices equivalentes en sus respectivas tablas del número de licencia del taxi y del nombre de usuario, así que con las mismas sentencias SQL de tipo SELECT se consiguen y almacenan. A continuación se realiza una actualización de los estados de las baterías que están recargándose en la tabla *cargador*. Con la sentencia UPDATE se actualiza la fila *estado* liberándose aquellas baterías que se marcan como reservadas o pre-reservadas y que no han sido recogidas a tiempo con un margen de 15 minutos o por algún error de la red no han actualizado su estado. Una vez están actualizados los estados se busca cuál es la batería que mejor satisface las necesidades del usuario, observando en la hora de carga cuál es la más próxima al instante solicitado por el taxi. Esto se realiza a través de una sentencia SQL de tipo SELECT en la que se elige el índice de la tabla *cargador* cuyo estado sea libre y su tiempo de carga restado al solicitado por el usuario sea mínimo.

Una vez obtenido el índice se procede a pre-reservar la batería. Para ello se realiza una sentencia UPDATE para actualizar el estado de ese índice a pre-reservado. Además se incluye cada pre-reserva en un registro en la tabla *prerreservas* con otra sentencia de tipo INSERT en la que se incluye el slot reservado, el número de licencia de taxi y el nombre de usuario, además de un sello de tiempo automático.

Como resultado del servicio se devuelve el slot pre-reservado, la hora más cercana en la que la batería de ese slot estará disponible y un parámetro error que indica si el servicio ha ido correctamente.

El usuario recibe la información del slot pre-reservado y debe confirmar la reserva. Tanto si la respuesta es positiva como negativa

se realiza el servicio de reserva. En el caso de confirmarla, el parámetro *respuesta* vale *true*, y para volver a liberar la batería el valor es *false*. También se envía los parámetros comunes de reserva, que son el número de licencia de taxi y el nombre de usuario. El fichero correspondiente a este servicio alojado en el servidor es *reserva.php*. Éste obtiene los parámetros y traduce el booleano *respuesta* en el estado correspondiente; si se confirma *estado* vale 4 y si se libera 2. Después se obtienen los índices de las tablas *taxis* y *usuarios* con las sentencias SQL de tipo SELECT de la misma manera que en la pre-reserva.

Una vez obtenidos estos índices se actualiza la tabla *cargador* con el estado calculado previamente con la sentencia UPDATE y también se añade un registro en la tabla *reserva* con los datos del taxi, usuario, slot y un sello de tiempo automático.

En la respuesta del servicio de reserva se indica el slot reservado y la confirmación de que la comunicación ha sido correcta.

Por último, para finalizar la gestión de las reservas falta describir el servicio de actualización del cargador. Éste se realiza desde el Arduino Ethernet, que monitoriza los movimientos que se producen en el cargador como extracciones de batería reservadas o inserciones de baterías desgastadas. La tabla *cargador* de la base de datos se encarga de indicar a tiempo real cómo se encuentra el cargador de baterías situado en la cooperativa de taxis.

El Arduino envía como parámetros el número de slot donde encuentra un cambio, el identificador de batería que se inserta o extrae, y el estado al que se actualiza. El fichero al que envía esa información es *updatecargador.php*.

En él se obtienen las variables *slot*, *batería* y *estado* y se envía una sentencia de tipo UPDATE a la tabla *cargador* en la que se actualiza el nuevo estado del slot del mismo. El Arduino Ethernet está programado para que al introducir una nueva batería se actualice el la marca de tiempo o timestamp a 2 horas después de la inserción, de forma que en la base de datos se indique a qué hora estará disponible.

Con estos servicios se engloba la gestión de las baterías desde el punto de vista del servidor, y su conexión con los terminales móviles y con el cargador de baterías.

4.4 Base de datos

Un pilar importante para el funcionamiento del trabajo es la base de datos ya que en ella se almacenan todos los movimientos de la plataforma. Su estructura requiere ser analizada ya que hay distintas opciones de construcción. Tras valorar distintos formatos se ha optado por utilizar varias tablas relacionadas indicadas en la Figura 4.2.

Se han realizado 8 tablas que indican los diferentes elementos o registros de servicio que figuran en el trabajo.

- La tabla *taxis* guarda los números de licencia dados de alta y los asocia con un identificador. Esta tabla está relacionada con los registros de monitorización, pre-reserva y reserva, en los que coge ese identificador.

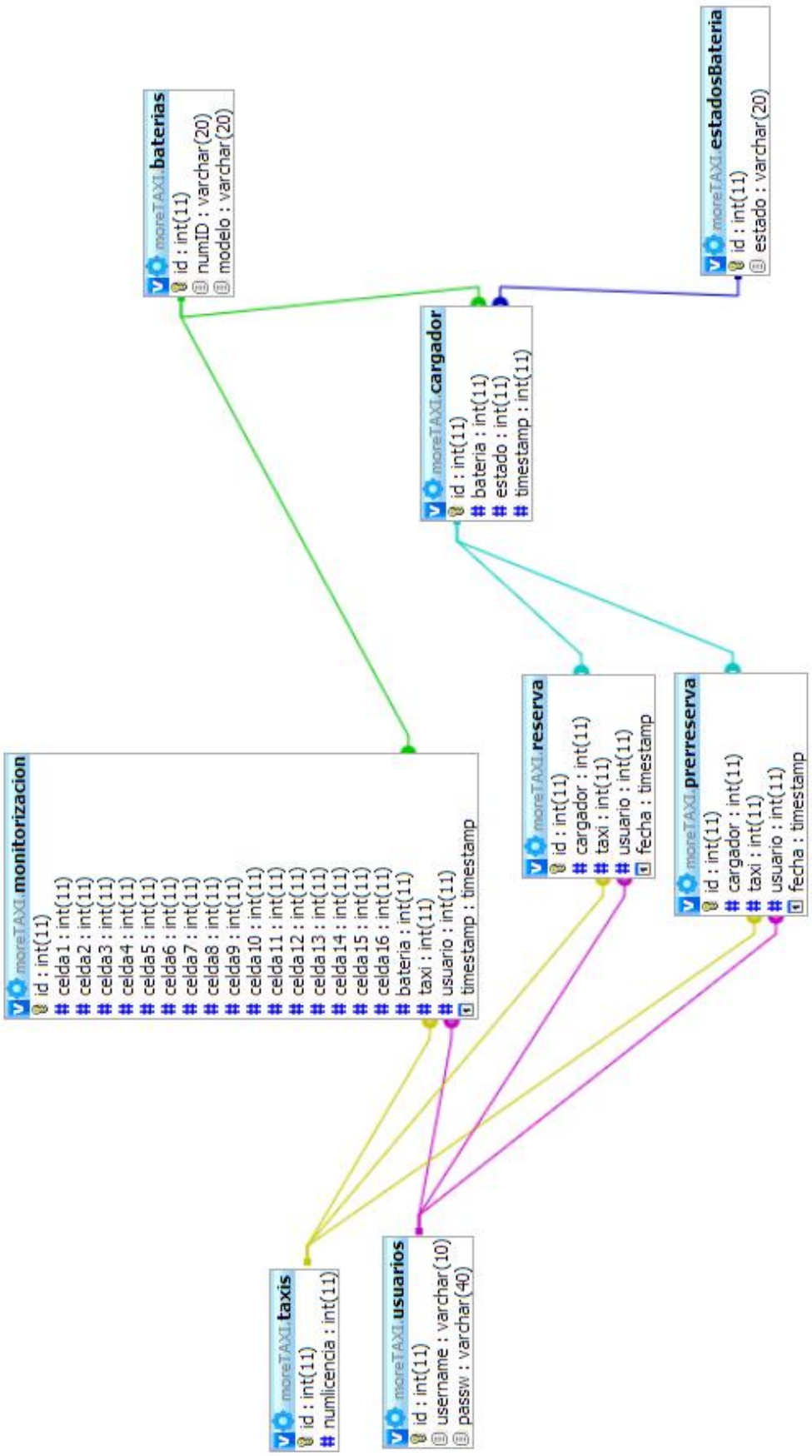

Figura 4.2 Diagrama de relaciones de la base de datos.

- La tabla *usuarios* almacena los nombres de los usuarios dados de alta y su contraseña firmada con el algoritmo SHA-1, además de otro identificador de tabla asociado. Esta tabla realiza el servicio de login además de estar relacionada también con los registros de monitorización, pre-reserva y reserva.
- En la tabla *monitorización* se recogen los datos que envía el Arduino Bluetooth al teléfono, que periódicamente son enviados al servidor. Está formada por 16 columnas con valores de voltajes de cada celda, además de recoger los identificadores de tabla de batería, de taxi y de usuario. Se le añade un timestamp automático cuando una fila es insertada.
- Las tablas *prerreserva* y *reserva* almacenan un registro de los cambios que se producen en la gestión de reserva de baterías. Están formadas por los índices de tabla de slot de cargador, taxi y usuario además de un timestamp automático al insertar una fila.
- La tabla *cargador* es la que representa a tiempo real cómo se encuentran los diferentes slots del cargador de baterías. Consta de un identificador de tabla que coincide con el número de slot, el identificador de batería, otro de estado de la batería, y un sello de tiempo que indica el momento en el que la batería se encuentra o encontrará disponible.
- En la tabla *batería* se guarda información sobre el modelo concreto de batería y se le asigna un identificador que permita reconocerla.
- Por último, en la tabla *estadosBatería* se guardan las posibles situaciones de una batería frente a la tabla *cargador*, en la que se han predefinido 4 estados: 1 equivale a hueco de cargador libre, 2 a que la batería está insertada en el cargador y está disponible, 3 en situación pre-reservada y 4 en reservada. Esta

tabla es necesaria cuando en un futuro se quiera añadir un estado nuevo como celda dañada, ya que con añadir una nueva fila a esta tabla se evitan problemas de cambios de todo el sistema.

Con esta estructura se consigue una base de datos escalable en la que más adelante la empresa Zevna pueda insertar alguna fila más si quisiera almacenar más información de los usuarios, taxis, baterías o estados de ellas.

Capítulo 5

Desarrollo de los Arduino

5.1 Planteamiento general

En este capítulo se describen las funcionalidades que desempeñan las placas Arduino utilizadas en este trabajo y cómo se ha realizado su programación.

Para la realización del proyecto se han necesitado 2 placas de Arduino UNO, en las que se han montado sobre ellas 2 módulos diferentes. Primero se describe el funcionamiento con el módulo Ethernet, que es la placa encargada de monitorizar las baterías del cargador. A continuación se describe el trabajo del Arduino con módulo Bluetooth, situado en el proyecto real en cada taxi y encargado de enviar las lecturas de la batería al terminal móvil.

Es importante aclarar que estos elementos son los que acotan el proyecto, y por tanto su programación es diferente al proyecto real. Al no tener acceso a la batería física se ha realizado una simulación de su comportamiento en la propia placa, así que cuando el proyecto se pusiera a la venta sería necesario cambiar estos comportamientos de simulación por lecturas de la batería. Esto sería aún más sencillo de realizar ya que Arduino permite leer a través de sus entradas analógicas con la función *read*.

Los diagramas de funcionamiento de las placas Arduino se encuentran disponibles en el **Anexo C** de la memoria.

5.2 Funcionamiento con módulo Ethernet

Como se ha comentado anteriormente, la placa Arduino encargada de monitorizar el estado del cargador es sobre la que se conecta este módulo. Como parte de simulación, se han conectado botones a una placa de prototipos para indicar los comportamientos de extracciones e inserciones de baterías. En las siguientes figuras se ilustran los diferentes elementos utilizados para desarrollar esta parte del trabajo.

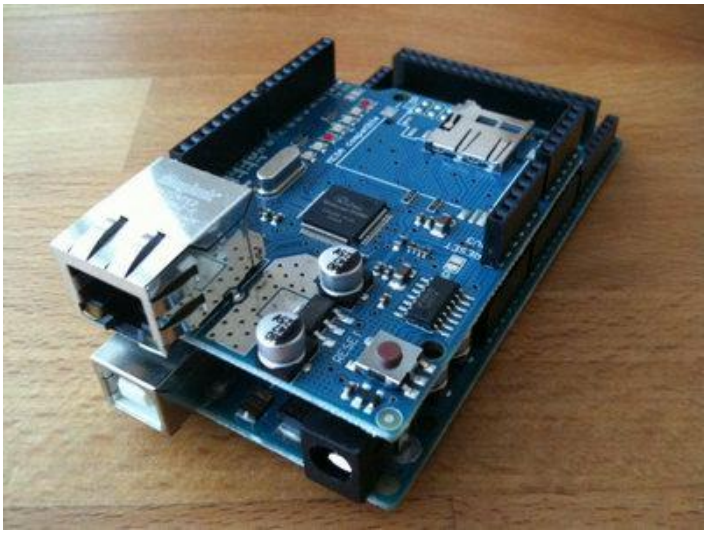

Figura 5.1 Arduino UNO con módulo Ethernet.

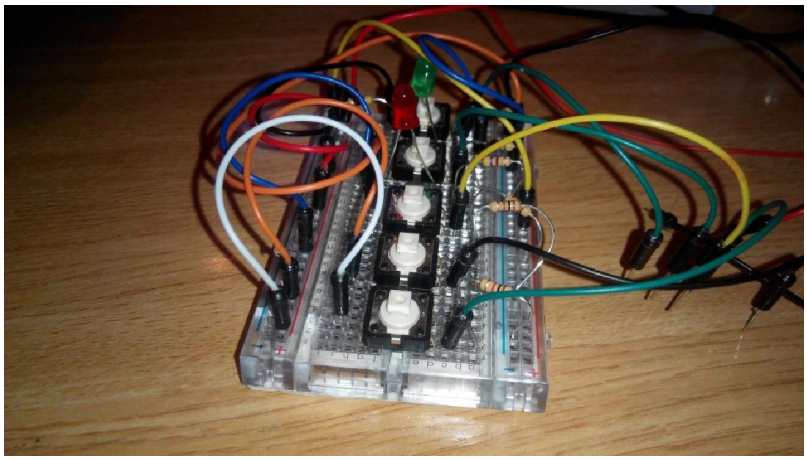

Figura 5.2 Placa de prototipos para simulación mediante botones.

5.2.1 Información del programa interno

El fichero programado que lleva cargada la placa es *SimBotones.ino*, que coge como referencia una conexión de cliente Web en su librería Ethernet. Además de incluir esta librería también se añade la del interfaz puerto serie, que será por donde se escriba la URL a la que se quiere acceder.

Después de inicializar las variables necesarias, en la función *setup()* del programa se inicia la conexión puerto serie a 9600 baudios y se definen los pines de entrada y salida. Para este entorno de simulación, y por limitaciones de tamaño de la placa de prototipos, se han definido 5 pines de entrada correspondientes a los botones del prototipo, y dos LEDs por los pines de salida.

Una vez se ha definido la configuración se escribe la función *loop()*, que se ejecutará como bucle infinito mientras la placa se encuentre encendida. El estado de los pines es leído con la función *digitalRead(n)*, donde n es el número de pin. Se detecta si hay algún botón pulsado, y si lo hay se espera 400 milisegundos. Pasado ese tiempo se vuelve a leer el estado de los botones, y de esta forma se detecta una pulsación corta de una larga. Se asocia una pulsación larga de botón a la extracción de una batería y una corta a una inserción de batería, siempre en el slot correspondiente a la pulsación. Cuando se detecta qué tipo de pulsación ha sido y en qué slot se empieza la conexión Ethernet con el router, utilizando la función *Ethernet.begin(mac)* con DHCP activado y con *Ethernet.begin(mac, ip, dns, gateway, subnet)* para la configuración manual.

Una vez se produzca la conexión con la función *client.connect(server,80)* se escribe con *print()* la URL y los parámetros que previamente se han establecido según la pulsación,

Finalmente se ejecuta la función *client.stop()* para finalizar la conexión de cliente web.

5.2.2 Información de la comunicación

Para comunicar correctamente con el servidor hay que tener en cuenta varios aspectos. En la variable *server*, que es de tipo *IPAddress* se almacena la dirección IP del servidor que almacena el fichero al que se accede, en este caso será *updateCargador.php*. Los parámetros enviados son el slot que ha sido pulsado en la simulación, el identificador de batería que se ha introducido y el nuevo estado al que se actualiza.

La forma de enviar los datos al servidor es similar en todos los servicios del trabajo, al realizar un cliente Web con conexión del puerto 80 al servidor, se realiza una petición GET a una URL con los parámetros recogidos en el fichero.

5.3 Funcionamiento con módulo Bluetooth

La placa Arduino que se instala en cada taxi se conecta al módulo Bluetooth, y su objetivo es realizar las lecturas de la batería que está suministrando energía al coche y enviarla por este protocolo inalámbrico al teléfono. Al realizarse una simulación en vez de una lectura de batería, el programa realiza un comportamiento de desgaste de batería en bucle, detallado en el próximo apartado. En la siguiente figura se muestra la placa con el módulo conectado.

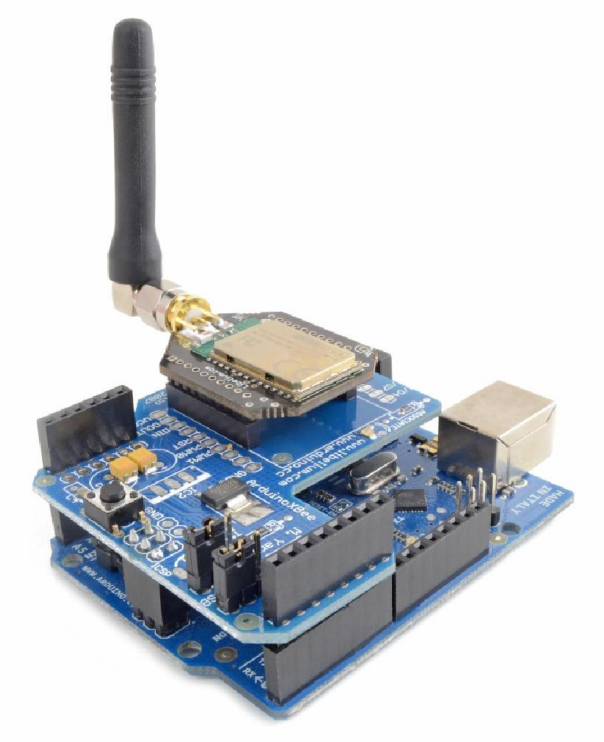

Figura 5.3 Arduino UNO con módulo Bluetooth.

5.3.1 Información del programa interno

El fichero de programación cargado en la placa es *SimBateria.ino*, que realizará un envío periódico de la información que lea.

En el programa se inicializan las variables entre las que destaca el vector de celdas y el identificador de batería que serían leídos.

La parte más importante del fichero es la función de configuración *setup()*, en la que se utiliza el puerto serie para configurar el módulo Bluetooth mediante comandos AT. Las características que más destacan son la configuración del nombre del dispositivo, el código PIN, indicar que acepte peticiones automáticamente y poner el dispositivo visible. Una vez configurado lo se conecta a través del comando *AT+JSCR*.

En la función *loop()* se construye el *String* que es enviado por Bluetooth. La cadena está separada por comas con las palabras clave INICIO y FIN delimitando la trama. Ésta se deposita directamente en el buffer de salida ya que se escribe en el puerto serie. Después se realiza un cambio de valores en las celdas como parte de la simulación, que decrementan su valor progresivamente y se inicializan cuando llegan a un valor mínimo. Para finalizar el bucle se realiza una espera con la función *delay()* que permite controlar el envío periódico.

5.3.2 Información de la comunicación

El envío de información por el protocolo Bluetooth en esta placa es muy sencillo. El propio módulo está preparado para conectarlo con el Arduino y poder escribir la información con el puerto serie. Es importante realizar un vaciado del buffer periódico con la función *Serial.flush()* entre la configuración y el envío de información para poder realizar la extracción correctamente. La comunicación entre la placa y el módulo con el puerto serie es de 115200 baudios, parámetro esencial de configuración ya que es el soportado por el módulo Bluetooth.

Capítulo 6

Pruebas realizadas

6.1 Pruebas de la aplicación

Las pruebas del correcto funcionamiento de la aplicación se realizaron a través de dos terminales móviles reales, un **Xiaomi Red Rice** y un **HTC Evo 3D**, con las siguientes características:

Versión Android	4.2.2
Resolución	1280x720
Memoria RAM	1Gb
Procesador	Quad-Core 1,5GHz

Cuadro 6.1: Características del terminal Xiaomi Red Rice

Versión Android	2.3.6
Resolución	960x540
Memoria RAM	1Gb
Procesador	Dual-Core 1,2GHz

Cuadro 6.2.: Características del terminal HTC Evo 3D

Además de estos terminales, para comprobar que el diseño de la aplicación no contiene errores en las diferentes resoluciones que ofrece Android se ha ejecutado la aplicación en diversos dispositivos simulados gracias al entorno Eclipse. Este emulador permite ejecutar la aplicación, aunque no permitirá comprobar su correcto

funcionamiento, ya que el simulador no permite hacer transferencias Bluetooth.

En cuanto a pruebas durante el desarrollo de la aplicación, Eclipse permite exportar rápidamente ésta para poder comprobar resultados, por lo que este período y el de programación están muy unidos. Estas pruebas se han realizado con el terminal Xiaomi Red Rice, por comodidad al ser móvil personal.

6.2 Pruebas del entorno completo

Además de realizar una serie de pruebas de la aplicación es necesario probar el esquema global del trabajo, tanto del servidor como de las placas Arduino.

En primer lugar, el servidor ha estado ubicado en dos lugares diferentes durante el desarrollo del trabajo, comprobando su correcto funcionamiento. En cuanto a la gestión del número de peticiones, ésta dependerá de las características del servidor donde se aloje el proyecto final, aunque en su desarrollo se ha probado las reservas con varios dispositivos simultáneamente y la respuesta ha sido óptima.

Las pruebas realizadas con las placas Arduino se han producido durante el desarrollo del proyecto. El módulo Bluetooth se ha probado en el interior de un coche con la alimentación de éste. El módulo Ethernet, que realiza la función de cargador, ha sido probado en varias ubicaciones diferentes del servidor con resultados satisfactorios.

Capítulo 7

Conclusiones y líneas futuras

7.1 Opinión personal

La realización de este trabajo me ha supuesto un reto en el sentido de que, además de lo cursado en la carrera, ha supuesto un trabajo de aprendizaje de algunos lenguajes de programación, introducirme en el mundo Arduino e ir buscando documentación sobre cómo realizar los servicios del servidor. Teniendo en cuenta también que no había realizado todavía ninguna aplicación móvil, ha sido difícil y ha costado más tiempo por hacerlo todo por primera vez. Además he tenido que adaptarme a las necesidades que han surgido aunque es cierto que he tenido mucha libertad para decidir cómo trabajar tanto de requerimientos de la aplicación como de tiempo de organización.

En el trabajo ha habido dos principales escollos, el primero ha sido el aprendizaje de programación de aplicaciones Android, ya que aunque se basa en Java y este lenguaje se aprendió en el grado, tiene una estructura de programación diferente. El otro obstáculo a destacar fue la monitorización Bluetooth del Arduino, tanto la configuración de la placa como la preparación del hilo receptor en el teléfono.

En conclusión, este trabajo me ha aportado unos conocimientos sobre toda la tecnología actual que me van a ser muy útiles en mi vida profesional.

7.2 Desglose de tiempos

En la Figura 7.1, mostrada a continuación, se muestra el reparto del tiempo de los distintos ámbitos que forman el desarrollo del trabajo. Se han separado en un proceso análisis del trabajo a realizar, aprendizaje de las distintas plataformas o lenguajes que se iban a utilizar, diseño de la aplicación, implementación de la misma, pruebas y corrección de errores y redacción de la memoria. Todas estas fases se sintetizan mediante el siguiente diagrama de Gantt:

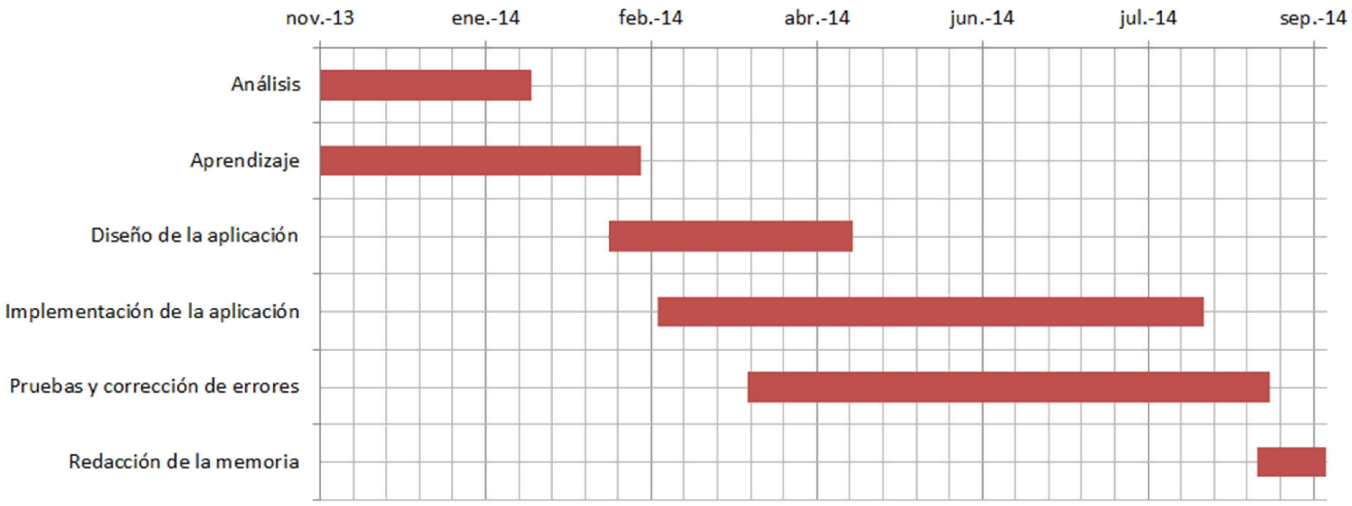

Figura 7.1: Diagrama de Gantt del proyecto.

7.3 Conclusiones

En este trabajo se han alcanzado los objetivos y requisitos finales e indicados al principio de este documento. Se ha realizado una aplicación que cuenta con inicio de sesión y una posterior búsqueda de dispositivos Bluetooth. A través de la conexión a Internet del teléfono y la conexión Bluetooth con el Arduino del coche se monitoriza el estado de la batería. A partir de los valores que se reciben, se calcula su autonomía. Además, la aplicación indica la geolocalización de la cooperativa de taxis, proporcionando la distancia y tiempo restante hasta ella. El usuario también puede realizar desde la aplicación la reserva de baterías, permitiéndole elegir la hora deseada de recogida. En cuanto a la plataforma, se ha desarrollado un servidor capaz de ofrecer el servicio de login, de monitorización con el registro de datos y la correcta gestión de reservas de las baterías. Como parte de la plataforma se han desarrollado también dos dispositivos Arduino. El primero es capaz de simular el comportamiento de una batería y enviar la información correspondiente al teléfono a través de Bluetooth. El segundo simula el comportamiento de un cargador de baterías y a través de una placa prototipo se puede insertar o liberar baterías, comunicándose con el servidor a través de Ethernet. Por lo tanto, se puede concluir que este trabajo ha concluido satisfactoriamente.

7.4 Líneas futuras

Una vez desarrollada la base de la aplicación este apartado se centra en las posibles mejoras que se harían tanto a la plataforma en general como a la aplicación en particular. A continuación se indican los primeros frentes a mejorar:

- Realizar la aplicación en otros sistemas operativos móviles: Aunque este trabajo se ha enfocado en realizar la aplicación en Android, que es el que mayor cuota de mercado tiene (un 84.6% en el segundo cuatrimestre de 2014 según Strategy Analytics), se podría realizar la aplicación para Apple iOS y Microsoft (Windows Phone), abarcando un 99,2% de la cuota de mercado mundial.
- Implementar una interfaz oral en la aplicación: Para un uso adecuado de la aplicación por parte de los conductores resulta adecuado facilitarles aún más la aplicación a través de una interfaz oral para que puedan hacer las reservas a través de esta vía.
- Detectar automáticamente fallos en las baterías: Este trabajo se centra en el envío de datos periódico de los voltajes de celdas de las baterías, teniendo como objetivo su almacenamiento para que posteriormente se hiciera un análisis manual de los datos. A través de scripts en el servidor se podría implementar notificaciones de fallos en las baterías cuando por ejemplo alguna celda falle.
- Detección automática de la batería: Tal y como se realiza este trabajo el conductor tiene que iniciar sesión e indicar a qué batería se conecta para empezar la monitorización. Esta

operación podría realizarse automáticamente si se guarda la sesión de conexión y se detecta la batería automáticamente.

- Implementar el proyecto completo: La línea futura más importante es implementar de forma completa el proyecto colocando realmente las baterías en los coches híbridos y terminando de programar los Arduinos para que lean los valores.

Bibliografía

[1] Página oficial de desarrolladores de Android:

http://developer.android.com/index.html

[2] Página oficial del entorno de desarrollo Eclipse:

http://projects.eclipse.org/

[3] Documentación de Java:

http://www.java.com/es/download/faq/whatis_java.xml

[4] Documentación de PHP:

http://php.net/manual/php4.php

[5] Documentación de MySQL:

http://www.mysql.com/

[6] Página oficial de FileZilla:

https://filezilla-project.org/

[7] Página oficial de Adobe Photoshop:

http://www.adobe.com/es/products/photoshop.html

[8] Documentación Arduino:

http://arduino.cc/en/pmwiki.php?n=Reference/HomePage

[9] Tutorial para crear el Servidor Web en Raspberry Pi:

http://www.penguintutor.com/linux/raspberrypi-webserver

[10] Tutorial Arduino Bluetooth: http://www.cooking-hacks.com/documentation/tutorials/arduino-bluetooth

[11] Foro de desarrolladores StackOverFlow:

http://stackoverflow.com/

[12] Foro de Cooking Hacks:

http://www.cooking-hacks.com/forum/

[13] Estándar Ethernet:

http://standards.ieee.org/about/get/802/802.3.html

[14] Estándar Bluetooth: https://www.bluetooth.org/en-us/specification/adopted-specifications

[15] Hypertext Transfer Protocol. RFC 2616 Junio 1999:

http://www.ietf.org/rfc/rfc2616.txt

[16] Documentación de la notación JSON:

http://json.org/

[17] File Transfer Protocol. RFC 959 Octubre 1985:

https://www.ietf.org/rfc/rfc959.txt

[18] Secure Shell Transport Layer Protocol. Rfc 4253 Enero 2006:

http://tools.ietf.org/html/rfc4253

[19] Dynamic DNS. RFC 2136 Abril 1997:

https://www.ietf.org/rfc/rfc2136.txt

[20] Secure Hash Algorithm 1 RFC 3174 Septiembre 2001:

http://tools.ietf.org/html/rfc3174