Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Sistema de comunicación entre un smartphone y una red aérea desplegada en situaciones de emergencia

Autor: Roberto Espejo Muñoz

Tutor: Jesús Sánchez García

Ponente: Daniel Gutiérrez Reina

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Sistema de comunicación entre un smartphone y una red aérea desplegada en situaciones de emergencia

Autor:
Roberto Espejo Muñoz

Tutor:
Jesús Sánchez García

Ponente:
Daniel Gutiérrez Reina

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera: Sistema de comunicación entre un smartphone y una red aérea desplegada en situaciones de emergencia

Autor: Roberto Espejo Muñoz
Tutor: Jesús Sánchez García
Ponente: Daniel Gutiérrez Reina

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia. A Vanesa. A mis amigos y compañeros. A todos, gracias por el apoyo recibido durante todos estos años, durante toda esta etapa. Y a Jesús, por la paciencia que ha tenido conmigo.

Roberto Espejo Muñoz
Sevilla, 2017

Resumen

El objetivo principal de este proyecto es establecer una comunicación inalámbrica entre un nodo proveedor de una red WiFi y un terminal móvil o tablet Android, de forma que el dispositivo móvil sea capaz de detectar cuál de entre los diferentes nodos que proveen conectividad inalámbrica es el más cercano físicamente para establecer dicha comunicación. La idea es que estos nodos representen dispositivos que dan servicios de comunicación en situaciones en las que no hay disponibles otras redes (por ejemplo, en una misión de rescate en un lugar donde la telefonía móvil no tiene cobertura, o bien donde la infraestructura de comunicaciones ha quedado inutilizada). El dispositivo móvil del usuario se conectará al nodo más cercano con el objetivo de enviar y recibir información útil, como pueden ser las coordenadas GPS de la posición más cercana a la que acercarse para pedir ayuda, o bien para enviar sus propias coordenadas GPS para pedir un rescate.

Abstract

The aim of this project is to stablish a wireless communication between a node providing a WiFi network and an Android mobile phone or tablet, in a way in which the mobile device is able to detect which of the wireless network providing nodes is the nearest one in space in order to stablish the connection. The idea is based on the nodes representing devices which provide communication services in situations where there are no other networks available (e.g, in a rescue mission where there is no mobile network coverage, or maybe the network infrastructure is completely disabled). The user's mobile device will connect with the nearest node in order to send or receive useful information, like the GPS coordinates of the nearest position where to ask for help, or send its own GPS coordinates to ask for a rescue.

Índice

Notación

ARP	*Automatic Repeat Request*, Repetición automática de petición
API	*Application Programming Interface*, Interfaz de programación de aplicaciones
AMPE	*Authenticated Mesh Peering Exchange*, Intercambio de emparejamiento mesh autenticado
AP	*Access Point*, Punto de acceso
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*, Acceso múltiple con detección de portadora y colisión evitable
DAC	*Deployable Aerial Communications*, Comunicaciones aéreas desplegables
DSL	*Domain Specific Language*, Lenguaje de dominio específico
DHCP	*Dynamic Host Configuration Protocol*, Protocolo de configuración dinámica de host
DSSS	*Direct Sequence Spread Spectrum*, Espectro disperso de secuencia directa
FHSS	*Frequency Hopping Spread Spectrum*, Espectro disperso por salto de frecuencia
GPIO	*General Purpose Input/Output*, Entrada/salida de propósito general
GPS	*Global Positioning System*, Sistema de posicionamiento global
IDE	*Integrated Development Environment*, Entorno de desarrollo integrado
IEEE	*Institute of Electrical and Electronics Engineers*, Instituto de ingenieros eléctricos y electrónicos
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*, Protocolo de Internet
JSON	*JavaScript Object Notation*, Notación de objetos de Javascript
LLC	*Logical Link Control*, Control de enlace lógico
OFDM	*Orthogonal Frequency Division Multiplexing*, multiplexación por división de frecuencia ortogonal
MAC	*Medium Acces Control*, Control de acceso al medio
NOOBS	*New Out Of the Box Software*, Software nuevo fuera de la caja
npm	*Node Package Manager*, Gestor de paquetes de Nodejs
nvm	*Node Version Manager*, Gestor de versiones de Nodejs
REST	*REpresentational State Transfer*, La Transferencia de Estado Representacional
RSSI	*Received Signal Strength Indicator*, Indicador de la fuerza de la señal recibida
SAE	*Simultaneous Authentication of Equals*, Autenticación simultánea de iguales
SoC	*System on a Chip*, Sistema en un chip

SSID	*Service Set Identifier*, Identificador de conjunto de servicios
TCP	*Transmission Control Protocol*, Protocolo de control de transmisión
TDMA	*Time Division Multiplexing Access*, Acceso por multiplexión en el tiempo
TDD	*Test-Driven Development*, Desarrollo guiado por pruebas
UAV	*Unmnaned Aerial Vehicle*, Vehículo aéreo no tripulado
UI	*User Interface*, Interfaz de usuario
USB	*Universal Serial Bus*
W3C	*Wide Web Consortium*
WLAN	*Wireless Local Area Network*, Red inalámbrica de área local

Índice de Figuras

Índice de Tablas

Índice de Códigos

1 Introducción y objetivos

En este capítulo se pretende dar una introducción a la materia sobre la que versa el presente proyecto, las comunicaciones inalámbricas en situaciones de emergencia, así como un breve resumen del estado del arte de las mismas para posteriormente introducir los objetivos que se pretenden alcanzar con el desarrollo del proyecto.

1.1 Introducción

El desarrollo vertiginoso de la tecnología electrónica en las últimas décadas, junto con el proceso de digitalización de la información, ha impulsado el desarrollo de diferentes estándares que soportan una infraestructura para las comunicaciones inalámbricas. Dependiendo del escenario de red y el campo de aplicación, puede resultar complicado y muy costoso mantener el correcto funcionamiento de la red. Tras una catástrofe a gran escala, una de las primeras consecuencias -más allá de la propia tragedia- es la interrupción del funcionamiento corriente de las infraestructuras de comunicación debido a la saturación o a daño físico de la propia infraestructura, como ha quedado comprobado en diversas situaciones a nivel mundial (tómese como ejemplo el reciente terremoto en Italia en el año 2012). Esto no hace si no propiciar la creación de sistemas de comunicación de emergencia, que puedan desplegarse por sí mismos en situaciones de urgencia y proveer de servicios de comunicaciones al área afectada.

Una posible solución para este tipo de situaciones son las denominadas DAC (*Deployable Aerial Communications*, Comunicaciones aéreas desplegables) [1], sistemas que permiten dar respuesta a operaciones post-desastre dadas sus ventajas frente a los sistemas terrestres. Los sistemas DAC pueden garantizar más cobertura, dado que los enlaces aéreos son menos propensos al desvanecimiento, y son más apropiados cuando la movilidad terrestre es limitada (como por ejemplo en el caso de inundaciones). El uso de este tipo de soluciones es posible gracias al incremento de la disponibilidad y asequibilidad de dispositivos UAV o drones (**Figura 1.1**), aunque las limitaciones de cobertura aérea de grandes zonas de terreno durante suficiente tiempo hacen que tanto el movimiento coordinado como los problemas energéticos sean los mayores hándicaps para su desarrollo. Existen sin embargo numerosos trabajos que investigan la creación de estos *enjambres* (como se denomina al conjunto de drones) específicamente diseñados para situaciones de emergencia [2].

Este tipo de *enjambres* suelen organizarse en una tolopología de red conocida como *mesh* [3], en la que cada nodo transmite información a los demás, de forma que todos los nodos de la red cooperan en la distribución de datos a través de la misma. Esta topología permite, de una forma eficiente, desplegar grandes redes de comunicaciones e interconectar redes heterogéneas.

Asumiendo este caso de uso de DAC implementadas en *enjambres* de UAVs, a lo largo del proyecto se utilizarán dispositivos Raspberry Pi para simular los drones en una situación de emergencia, con el objetivo de proveer de servicios de comunicaciones a los afectados por el desastre. Se utilizarán dos Raspberries emitiendo WiFi, a diferentes distancias del teléfono, para simular tan sólo una parte del *enjambre*.

El teléfono móvil Android del afectado actuará como nodo de tierra, el cual, haciendo uso de una aplicación, buscará establecer comunicación inalámbrica con el UAV que provea la mayor intensidad de señal en dB (aunque podría no ser el más cercano en distancia física), con el objetivo de intercambiar sus coordenadas

Figura 1.1 UAV de tipo quadcopter.

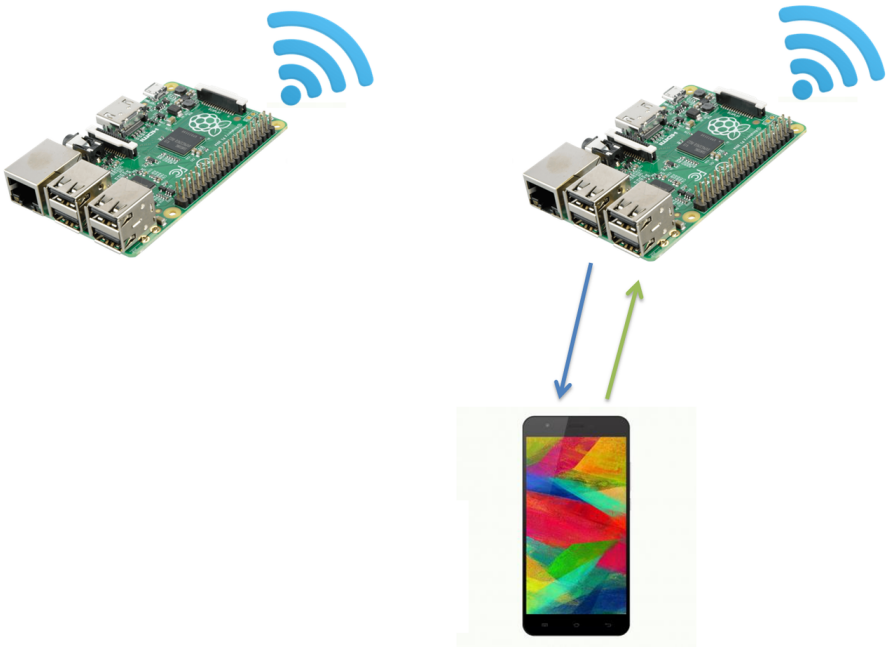


Figura 1.2 Esquema del sistema a realizar.

GPS con el sistema de emergencias (**Figura 1.2**); ya sea enviándolas a un servidor web para pedir un rescate, o pidiendo las coordenadas del punto más cercano al que poder acudir para pedir ayuda.

El dispositivo móvil presentará dos opciones de comunicación con el servidor: en una de ellas, el usuario en situación de emergencia podrá solicitar la localización del punto de ayuda más cercano a su posición, la cual que será enviada por la aplicación como parámetro al servidor (quien se encargará de realizar los cálculos pertinentes); mientras que en la segunda opción, el usuario se encontrará en una situación en la que no podrá acudir por sus propios medios al punto de ayuda, de modo que podrá solicitar un rescate en la posición en la que se encuentre (que de nuevo sería enviada como parámetro en forma de coordenadas GPS al servidor).

1.2 Objetivos

El desarrollo de las ya mencionadas DAC es una materia bastante extensa, debido a que se deben tener en cuenta multitud de escenarios y variables, desde aspectos *hardware* como las limitaciones de autonomía de los drones, su capacidad de vuelo o la capacidad de moverse realmente como un *enjambre*; pasando por aspectos climatológicos dependientes de la zona que se quiera cubrir (que afectan tanto a los UAVs como a

las comunicaciones inalámbricas), aspectos del terreno (altura y número de edificios cercanos, elevaciones del terreno, bosques), hasta las diferentes configuraciones de red para comunicar el *enjambre* con el resto de dispositivos y servicios de emergencias. Es por ello que para el proyecto que nos ocupa se ha decidido abordar únicamente el problema de las comunicaciones entre el nodo de usuario (el dispositivo móvil) y los nodos de la red mesh (drones).

Se supondrá que cada nodo tiene su propia red inalámbrica basada en 802.11, el estándar de comunicaciones sin cable más extendido en los *smartphones* [4], independientemente de cómo esté conectado con el resto de nodos de la red mesh, a la que se podrán conectar los usuarios móviles. Con respecto al desarrollo de la aplicación móvil que pueda conectarse a dichos nodos, se ha optado por el sitema operativo Android, por la sencilla razón de contar con más cuota de mercado que otros sistemas operativos móviles y poder ser ejecutado en dispositivos desde las gamas bajas del mercado hasta la última tecnología.

Se pretende, por tanto, de forma secuencial:

- Estudiar las redes mesh y el estándar 802.11 de comunicaciones inalámbricas.
- Configurar las Raspberry Pis para crear sendas redes WiFi a la que el dispositivo móvil pueda conectarse.
- Desarrollar un servicio web que permita a los dispositivos conectados a la red de la Raspberry enviar y recibir coordenadas GPS.
- Desarrollar una aplicación para el sistema operativo móvil Android que, en función de la intensidad de señal, se conecte a la red de la Raspberry Pi más cercana para poder establecer comunicación con la misma.

2 Fundamentos teóricos

El objetivo de este capítulo es situar el desarrollo del proyecto bajo un contexto teórico que provea las bases para el desarrollo práctico del mismo. Se recogen a continuación los conceptos y protocolos que sustentan (ya sea directa o indirectamente, como en el caso de las redes mesh) el sistema a desarrollar.

2.1 Redes Mesh

Las redes *mesh*, o *de malla*, son sistemas multisalto (*multihop*) [3] en los cuales los nodos se ayudan el uno al otro para transmitir paquetes a través de la red, organizadas en una topología de malla. Este tipo de redes ad hoc se pueden poner en marcha con una preparación mínima, y proveen un sistema fiable y flexible que puede extenderse a miles de dispositivos, lo cual las hace especialmente interesantes en situaciones de emergencia. La fiabilidad, escalabilidad y adaptabilidad son las principales características de las redes mesh inalámbricas.

Las redes mesh son redes *peer-to-peer* (punto a punto), ad hoc y multisalto. Un nodo puede enviar y recibir mensajes, a la vez que hace la función de *router* (encaminador) y reenvía mensajes a sus nodos vecinos. A través del proceso de reenvío, un paquete de red encontrará una ruta hacia su destino, pasando por nodos intermedios con enlaces de comunicación fiables.

Al igual que Internet y otras redes punto a punto basadas en rutas, las redes de malla ofrecen múltiples rutas alternativas redundantes, de forma que si un enlace falla por cualquier razón, la red automáticamente encamina los mensajes a través de caminos alternativos. También, al reducir la distancia entre nodos, se aumenta la calidad del enlace, lo cual aumenta la fiabilidad de las comunicaciones sin un aumento de potencia en el transmisor. De esta forma, en una red mesh, aumentar el alcance, la redundancia y la fiabilidad general de la red es tan sencillo como añadir más nodos.

Una red mesh se organiza de manera automática, y no requiere de un administrador o configuración manual. De este modo, para desplazar un nodo o añadir uno nuevo, basta con simplemente encender el dispositivo. La red descubrirá el nuevo nodo y lo añadirá automáticamente al sistema. Esto hace que las redes mesh no sólo sean fiables, si no también adaptables: el hecho de perder uno o más nodos no debería afectar al

(a) Red de malla completa.

(b) Red de malla parcial.

Figura 2.1 Redes mesh.

funcionamiento de la red (siempre que la configuración de malla de la red no provoque una división en dos de la misma, o el aislamiento total de un nodo).

Acrónimo	Nombre	Funcionalidad
AHCP	Ad Hoc Configuration Protocol	Autoconfiguración
AODV	Ad hoc On-Demand Distance Vector	Encaminamiento
B.A.T.M.A.N	Better Approach To Mobile Adhoc Networking	Encaminamiento
DNVR	Dynamic Nix-Vector Routing	Encaminamiento
DSDV	Destination-Sequenced Distancce-Vector Routing	Encaminamiento
DSR	Dynamic Source Routing	Encaminamiento
DWCP	Dynamic WMN Configuration Protocol	Autoconfiguración
HSLS	Hazy-Sighted Link State	Encaminamiento
HWMP	Hybrid Wireless Mesh Protocol	Encaminamiento
OLSR	Optimized Link State Routing protocol	Encaminamiento
OSPF	Open Shortest Path First Routing	Encaminamiento
PAP	Proactive Autoconfiguration Protocol	Autoconfiguración
PWRP	Predictive Wireless Routing Protocol	Encaminamiento
TORA	Temporally-Ordered Routing Algorithm	Encaminamiento
ZRP	Zone Routing Protocol	Encaminamiento

Tabla 2.1 Algunos protocolos de encaminamiento y autoconfiguración de redes mesh.

Si bien existen numerosos protocolos de encaminamiento (**Tabla 2.1**), el IEEE ha desarrollado un conjunto de estándares bajo el título **802.11s** con el objetivo de definir la arquitectura y los protocolos de las redes mesh [5]. Se trata de una modificación del estándar **802.11** en la que se define cómo los dispositivos inalámbricos se han de comunicar para crear redes WLAN mesh, bien sea para topologías estáticas o redes ad hoc. Este nuevo estándar depende de los anteriores (802.11a, 802.11b, 802.11g o 802.11n) para soportar el tráfico [6]. Con respecto a los protocolos de encaminamiento, 802.11s establece el uso de HWMP. 802.11s también incluye mecanismos para proveer acceso determinista a la red, para control de la congestión y ahorro de energía.

En una red mesh no hay definidos roles: no hay clientes ni servidores. Los protocos de seguridad utilizados en una red mesh deben, por lo tanto, ser protocolos puramente punto a punto donde cualquiera de los extremos puede iniciar la comunicación con el otro. El estándar del IEEE define un protocolo seguro de autenticación basada en contraseña llamado SAE (*Simultaneous Authentication of Equals*, Autenticación simultánea de iguales). Este mecanismo está basado en el intercambio de claves Diffie-Hellman. Cuando los nodos se descubren (y hay seguridad establecida) toman parte en un intercambio SAE. Si este tiene éxito, cada nodo sabe que el otro lado posee la contraseña de la red, y establecen una clave criptográficamente robusta. Esta clave se utiliza con el AMPE (*Authenticated Mesh Peering Exchange*, Intercambio de emparejamiento mesh autenticado) para establecer un emparejamiento seguro y proteger el tráfico de la red.

2.2 IEEE 802.11

El estándar IEEE 802.11 fue definido en el año 1997 como nuevo estándar para las redes de área local inalámbrica, con el objetivo de sustituir a las capas física y de enlace del modelo OSI para redes cableadas (IEEE 802.3) especificando su funcionamiento en redes WLAN, haciendo que ambas redes sean idénticas excepto en la forma en la que los terminales acceden a la red.

La capa física de la especificación IEEE 802.11 se ocupa de definir los métodos por los que se difunde la señal. Ofrece 4 tipos de técnicas de transmisión:

- Infrarrojos: usa transmisión de corto alcance, con una velocidad de 1Mbps o 2Mbps.
- FHSS (*Frequency Hopping Spread Spectrum*, Espectro disperso por salto de frecuencia): se transmiten los datos saltando de canal a canal, de acuerdo a una secuencia de salto pseudo aleatoria particular que distribuye uniformemente la señal a través de la banda de frecuencia operativa. Una vez que la secuencia de saltos se configura en un AP, las estaciones se sincronizarán automáticamente según la secuencia de salto correcta. Se consiguen velocidades de transmisión de 1Mbps a 2Mbps. Opera en la banda de los 2,4GHz.

- DSSS (*Direct Sequence Spread Spectrum*, espectro disperso de secuencia directa): utiliza un rango de frecuencia amplio de 22 MHz todo el tiempo. La señal se expande a través de diferentes frecuencias. Cada bit de datos se convierte en una secuencia de chipping que se transmiten en paralelo a través del rango de frecuencia. Se consiguen velocidades de transmisión de 1Mbps a 2Mbps en la versión normal, y hasta 11 Mbps en la versión HR/DSSS. Opera en la banda de los 2,4GHz.
- OFDM (*Orthogonal Frequency Division Multiplexing*, multiplexación por división de frecuencia ortogonal): Divide una portadora de datos de alta velocidad en varias subportadoras de más baja velocidad, que luego se transmiten en paralelo. OFDM utiliza el espectro de manera mucho más eficiente, espaciando los canales a una distancia mucho menor. El espectro es más eficiente porque todas las portadoras son ortogonales entre sí, evitando de esa forma la interferencia entre portadoras muy cercanas. Se consiguen velocidades de transmisión de hasta 54Mbps. Opera en la banda de los 5GHz.

Fecha	Estándar	Banda(GHz)	Ancho de banda (MHz)	Modulación	Tasa binaria máxima
1997	802.11	2.4	20	DSSS, FHSS	2Mbps
1999	802.11b	2.4	20	DSSS	11Mbps
1999	802.11a	5	20	OFDM	54Mbps
2003	802.11g	2.4	20	DSSS, OFDM	54Mbps
2009	802.11n	2.4, 5	20, 40	OFDM	600Mbps

Tabla 2.2 Estándares 802.11.

La capa de enlace de la especificación 802.11 está compuesta por dos subcapas:

- LLC: Capa que se ocupa del control del enlace lógico. Se trata de la capa subcapa superior. Define cómo pueden acceder múltiples usuarios a la capa MAC, pues provee mecanismos de multiplexado que hacen posible que diferentes protocolos (como IP, IPX, Decnet...) puedan coexistir dentro de una red multipunto y se transporten sobre el mismo medio de red. También provee mecanismos de manejo de errores ARQ (*Automatic Repeat Request*, Repetición automática de petición).
- MAC: Conjunto de protocolos que controlan cómo los distintos dispositivos comparten el uso del espectro radioeléctrico. Es más compleja que las de otras especificaciones (802.3, 802.5, etc.). En WLAN se utiliza CSMA/CA (acceso múltiple con detección de portadora y colisión evitable). Sus funciones principales son la exploración (envío de *beacons*, balizas, que incluyen los SSID), la autenticación, asociación (el proceso que da acceso a la red), la seguridad y la fragmentación (la capacidad del punto de acceso de dividir la información en tramas más pequeñas).

2.2.1 Medida de la potencia de la señal WiFi

El denominado por sus siglas en inglés RSSI (*Received Signal Strength Indicator*, Indicador de la fuerza de la señal recibida) es una medida de la potencia presente en una señal radio en un receptor. Se suele medir a frecuencia intermedia (FI), antes del amplificador FI. En sistemas sin FI, se mide antes del amplificador en banda base.

En los sistemas IEEE 802.11, el RSSI es la potencia de señal recibida en un entorno inalámbrico, en unidades arbitrarias. Es un indicador del nivel de potencia recibido por el sistema radio tras las pérdidas de la antena, conectores y cable. Por lo tanto, mientras mayor sea el valor del RSSI, con mayor potencia llegará la señal. Los valores del RSSI se presentan como números negativos, siendo 0 el máximo y representando un 100 % de señal útil recibida. Este valor es calculado normalmente por las tarjetas de red, con frecuencia para determinar si la potencia de la señal está por encima de cierto umbral para poder comenzar la comunicación. No existe, sin embargo, ningún tipo de relación estandarizada entre los parámetros físicos y una lectura del RSSI. El estándar 802.11 no define ninguna relación entre el valor del RSSI y el nivel de potencia en milivatios o decibelios, pero sí que, dentro de los mismos, hay que tener en cuenta que a diferencia de otros sistemas de acceso por multiplexión en el tiempo (*Time Division Multiplexing Access*, TDMA), como GSM, donde para realizar este tipo de medidas de potencia se ajusta a la máscara temportal correspondiente, en 802.11 la medida se realiza durante el preámbulo de la trama, y no durante toda la ráfaga.

En el caso del sistema Android que nos ocupa en este proyecto, el propio sistema operativo cuenta con drivers para comunicarse con el chip BCM4339 de Broadcom (el que utiliza el LG G3), que contiene las

antenas WiFi y Bluetooth del teléfono, cuyo firmware se encarga de realizar la medida del RSSI. La API de Android se encarga de proporcionar a los desarrolladores un objeto `WifiManager` a través del cual obtener información sobre las redes inalámbricas detectadas por la antena, entre las cuales se encuentra la potencia recibida en decibelios. Comparando los distintos valores para las distintas redes, el dispositivo se conectará a la red cuya señal recibida tenga mayor potencia, como se detallará más adelante en el capítulo 4.

2.3 Modos Ad hoc e infraestructura

Los elementos que intervienen en cualquier comunicación inalámbrica son los puntos de acceso (equipos que dan acceso a la red de forma centralizada) y los clientes (dispositivos inalámbricos). Existen dos tipos de topologías:

- Los clientes se conectan directamente entre ellos para comunicarse (modo *Ad Hoc*).
- Los clientes se conectan a un punto de acceso para comunicarse con el exterior o entre ellos mismos (modo Infraestructura).

2.3.1 Modo Ad Hoc

En el modo Ad Hoc los equipos inalámbricos se conectan entre sí para formar una red punto a punto, es decir, los clientes intercambian la información entre ellos directamente sin pasar por ningún equipo intermedio. Las opciones de configuración de seguridad, nombre de red y canal de comunicación se configuran el propio cliente. Una vez que los clientes pertenecen a la misma red, se transmiten los datos al aire y los otros dispositivos reciben y reenvían la información. La configuración que forman los clientes se llama conjunto de servicio básico independiente o IBSS (Independent Basic Service Set).

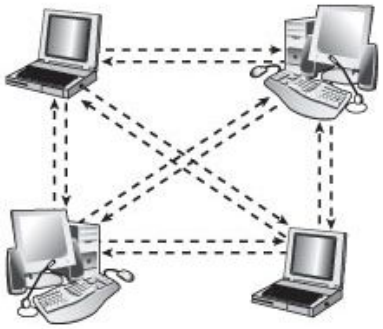


Figura 2.2 Modo ad hoc.

2.3.2 Modo Infraestructura

En el modo de infraestructura, los clientes deben comunicarse con el punto de acceso para poder utilizar la red, ya que el punto de acceso es el encargado de gestionar la autorización. Al grupo formado por el punto de acceso y los clientes que se encuentran dentro de la misma zona de cobertura se le llama conjunto de servicio básico o BSS (*Basic Service Set*). También puede darse el caso de que una red esté formada por varios puntos de acceso y clientes que se conecten a ellos. A este grupo de puntos de acceso y clientes se le llama conjunto de servicio extendido o ESS (Extended Service Set). En este tipo de redes el nombre de la red se define en el punto de acceso con el parámetro ESSID (Extended Service Set ID), lo nos permite diferenciar una red de otra.

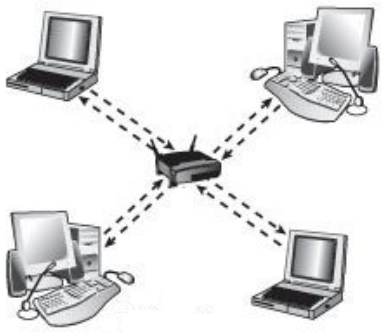


Figura 2.3 Modo infraestructura.

2.4 GPS

El sistema de posicionamiento global, llamado GPS (*Global Positioning System*), es un sistema de navegación espacial por satélite que permite determinar una posición en la Tierra con gran precisión. Desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos, el sistema GPS se sirve de 24 satélites y utiliza la trilateración: para determinar la posición, el receptor deberá localizar al menos tres de los satélites de la red GPS, los cuales enviarán un identificador y una señal de reloj. En base a estas señales, el dispositivo sincronizará el reloj del GPS y calculará el tiempo que han tardado en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante el método de trilateración inversa, el cual se basa en determinar la distancia de cada satélite al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas reales del punto de medición.

2.4.1 Fórmula del haversine

La fórmula del haversine, o fórmula del semiverseno, es una ecuación que determina la distancia de círculo máximo entre dos puntos en una esfera dadas su latitud y su longitud. Se trata de un caso particular de la denominada ley de los semiversenos, que relaciona los lados y ángulos de los triángulos esféricos. La fórmula se basa en la llamada *función haversine*:

$$hav = \sin^2(\frac{\Delta\phi}{2}) + \cos\phi_1 * \cos\phi_2 * \sin^2(\frac{\Delta\lambda}{2}) \tag{2.1}$$

$$c = 2 * atan2(\sqrt{hav}, \sqrt{(1-hav)}); \tag{2.2}$$

Siendo ϕ y λ latitud y longitud respectivamente, en radianes. La distancia en metros entre ambos puntos será:

$$d = R * c \tag{2.3}$$

donde R es el radio de la tierra en metros, $R = 6371 * 10^3$.

2.5 *Test Driven Development*

Se ha decidido usar el enfoque **TDD** (*Test-Driven Development*, Desarrollo guiado por pruebas) con el fin de desarrollar la aplicación del servidor web, una técnica de diseño e implementación de software que se centra en tres pilares fundamentales [7]:

- La implementación de las funciones justas que el cliente necesita y no más.
- La minimización del número de *bugs* (errores) que llegan al software en fase de producción.
- La producción de software modular, altamente reutilizable y preparado para el cambio.

Al contrario que en otras metodologías, donde lo primero es definir la arquitectura, las distintas clases y objetos, patrones, etc, en TDD se deja que la propia implementación de pequeños ejemplos, en constantes iteraciones, haga emerger la arquitectura que es necesario utilizar. Existen numerosas características que hacen que TDD sea una herramienta importante a la hora de diseñar software: la calidad del software aumenta, el código aumenta su reutilización, y el hecho de que escribir el ejemplo (test) antes que el código nos obliga a escribir el mínimo de funcionalidad necesaria, evitando sobrediseñar.

Para llevar a cabo TDD, en primer lugar se debe definir una lista de requisitos (objetivos) y después se ejecuta el siguiente ciclo:

1. Elegir un requisito: Se elige de una lista el requisito que se cree que nos dará mayor conocimiento del problema y que a la vez sea fácilmente implementable.
2. Escribir una prueba (test): Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces.
3. Verificar que la prueba falla: Si la prueba no falla es porque el requisito ya estaba implementado o porque la prueba es errónea.
4. Escribir la implementación: Escribir el código más sencillo que haga que la prueba funcione.
5. Eliminación de duplicación: El paso final es la refactorización, que se utilizará principalmente para eliminar código duplicado. Se hace un pequeño cambio cada vez y luego se corren las pruebas hasta que funcionen.
6. Actualización de la lista de requisitos: Se actualiza la lista de requisitos tachando el requisito implementado. Asimismo se agregan requisitos que se hayan visto como necesarios durante este ciclo y se agregan requisitos de diseño.

Se expone a continuación un sencillo ejemplo de cómo se lleva a cabo este tipo de desarrollo:

Supongamos que se desea implementar una sencilla calculadora que, haciendo uso de un método `add`, sume dos números proporcionados en una cadena de caracteres, separados por comas. El método deberá responder a la forma `Int add(String numbers)` (en este ejemplo se usará la sintaxis del lenguaje Java, pero la metodología es aplicable a cualquier lenguaje de programación). Definimos en primer lugar las especificaciones:

- El método aceptará como parámetros 0, 1 o 2 números.
- El método devolverá 0 si se pasa una cadena vacía como parámetro.
- El método devolverá la suma de los parámetros cuando estos sean válidos.
- El método también deberá aceptar caracteres de retorno de carro como separados válidos, además de las comas.

En este caso, se ha querido limitar el número de requerimientos con el fin de mantener la simplicidad en el ejemplo. A continuación, se elige el primer requisito de la lista, y se escriben los tests:

```
import org.junit.Test;
import com.roberto.example.tdd.StringCalculator;

public class StringCalculatorTest {
    @Test(expected = RuntimeException.class)
    public final void whenMoreThan2NumbersAreUsedThenExceptionIsThrown() {
        StringCalculator.add("1,2,3");
    }
    @Test
    public final void when2NumbersAreUsedThenNoExceptionIsThrown() {
        StringCalculator.add("1,2");
        Assert.assertTrue(true);
    }
    @Test(expected = RuntimeException.class)
    public final void whenNonNumberIsUsedThenExceptionIsThrown() {
        StringCalculator.add("1,X");
    }
}
```

Se está usando en este caso la librería Junit de Java como *framework* para escribir los tests. Se han escrito tres pruebas distintas para comprobar que se cumple la especificación: en la primera, se espera que se lance una excepción del tipo `RuntimeException` cuando se proveen más de dos números como argumentos; en la segunda, se espera que cuando se proveen dos números como argumento, no se lanza ninguna excepción; y en la tercera se comprueba que cuando se provee un argumento que no es un número, se lanza una excepción del tipo `RuntimeException`.

Si intentamos ejecutar estos tests, veremos que en el caso de Java no podemos, pues la clase a la que hacen referencia las pruebas (`StringCalculator`) aún no está implementada. Implementamos entonces la misma para que cumpla los objetivos de las pruebas, de la forma más sencilla posible:

```
public class StringCalculator {
    public static final void add(final String numbers) {
        String[] numbersArray = numbers.split(",");
        if (numbersArray.length > 2) {
            throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
        } else {
            for (String number: numbersArray) {
                Integer.parseInt(number); // If it is not a number, parseInt will throw an exception
            }
        }
    }
}
```

Si ejecutamos los tests, veremos que ahora todos pasarán de forma correcta. Esto nos indica que hemos implementado con éxito el primero de los requisitos. Procedemos entonces al siguiente elemento de la lista de especificaciones: en este caso, que el método devuelve 0 para el caso en el que el argumento es una cadena vacía. Añadimos un nuevo test a los anteriores para cubrir este comportamiento:

```
@Test
public final void whenEmptyStringIsUsedThenReturnValueIs0() {
    Assert.assertEquals(0, StringCalculator.add(""));
}
```

`assertEquals` comprueba la igualdad entre los dos argumentos que se le pasan (en este caso comprueba que el resultado devuelto por la llamada al método con la cadena vacía como parámetros). Si ejecutamos de nuevo el conjunto de pruebas, comprobaremos que las tres anteriores siguen pasando, mientras que la nueva fallará, pues aún no está implementada. Realizamos entonces las modificaciones necesarias en el código:

```
public static final int add(final String numbers) { // Changed void to int
    String[] numbersArray = numbers.split(",");
    if (numbersArray.length > 2) {
        throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
    } else {
        for (String number: numbersArray) {
            if (!number.isEmpty()) {
```

```
                Integer . parseInt (number);
            }
        }
    }
    return 0; // Added return
}
```

Ejecutando las pruebas, ahora las cuatro pasarán correctamente. Podemos comprobar por tanto que no hemos alterado el comportamiento que se había implementado hasta ahora, y que también se ha implementado de forma correcta la nueva especificación. Pasamos entonces a la última especificació en la lista: que el método devuelva la suma de los números. De nuevo escribimos los casos de prueba para este comportamiento, añadiéndolos a los ya existentes:

```
@Test
public final void whenOneNumberIsUsedThenReturnValueIsThatSameNumber() {
    Assert . assertEquals (3, StringCalculator .add("3"));
}

@Test
public final void whenTwoNumbersAreUsedThenReturnValueIsTheirSum() {
    Assert . assertEquals (3+6, StringCalculator .add("3,6"));
}
```

Se quiere comprobar que la función devuelve la suma tanto cuando se pasa un sólo número como cuando se pasan dos, de nuevo haciendo uso del método `assertEquals` para comprobar la igualdad. Siguiendo el ciclo TDD, ejecutamos los tests y observamos cómo los nuevos fallan. Pasamos entonces a escribir la implementación, volviendo a cambiar el código:

```
public static int add( final String numbers) {
    int returnValue = 0;
    String [] numbersArray = numbers. split (",");
    if (numbersArray.length > 2) {
        throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
    }
    for ( String number : numbersArray) {
        if (!number.trim() .isEmpty()) { // After refactoring
            returnValue += Integer . parseInt (number);
        }
    }
    return returnValue ;
```

Ejecutando de nuevo las pruebas podemos comprobar que pasan correctamente, concluyendo por tanto este ejemplo. Se puede observar cómo gracias a TDD y a varias iteraciones, se ha ido escribiendo tan sólo el código que es necesario utilizar, ayudando también al mantenimiento del mismo. Si bien en este ejemplo es difícil apreciarlo, en grandes proyectos *software* no es extraño el poder cambiar una funcionalidad sin que el desarrollador se percate de su error, y es gracias a las pruebas que este tipo de errores no se pasarán por alto.

2.6 Comunicación entre cliente y servidor

La comunicación entre el cliente y el servidor se realizará sobre el protocolo HTTP (*HyperText Transfer Protocol*, Protocolo de transferencia de hipertexto). La estandarización de HTTP corre a cargo de la IETF (*Internet Engineering Task Force*) y el W3C (*Wide Web Consortium*), mediante la publicación de RFC's (*Requests For Comments*), siendo la primera publicada la número 2068 en 1997 estandarizando la primera versión del protocolo, HTTP/1.1.

HTTP es un protocolo de petición-respuesta. El cliente realiza una petición HTTP al servidor, el cual provee recursos (como ficheros HTML, por ejemplo, en la navegación web, entre otros muchos tipos) o realiza operaciones para luego devolver un mensaje de respuesta al cliente. Dicha respuesta contendrá información sobre la petición y el contenido solicitado contenido en el cuerpo (*body*). Se trata de un protocolo de nivel de aplicación, y su definición asume una capa subyacente fiable al nivel de la capa de transporte, siendo normalmente TCP (*Transmission Control Protocol*) el más utilizado. Los recursos HTTP se identifican y localizan en la red gracias a URLs (*Uniform Resource Locators*, localizadores uniformes de recursos).

El protocolo define varios métodos (a veces llamados verbos) para indentificar la acción que se quiere realizar sobre el recurso identificado. La especificación HTTP/1.1 se definen diferentes métodos: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT y PATCH. En el caso de la API que se desarrollará en el presente proyecto, se ofrecen dos rutas diferentes (URLs) que habrán de ser llamadas con el método GET, y se ilustra en la **figura 2.4** un esquema de la comunicación entre el cliente Android y el servidor Javascript:

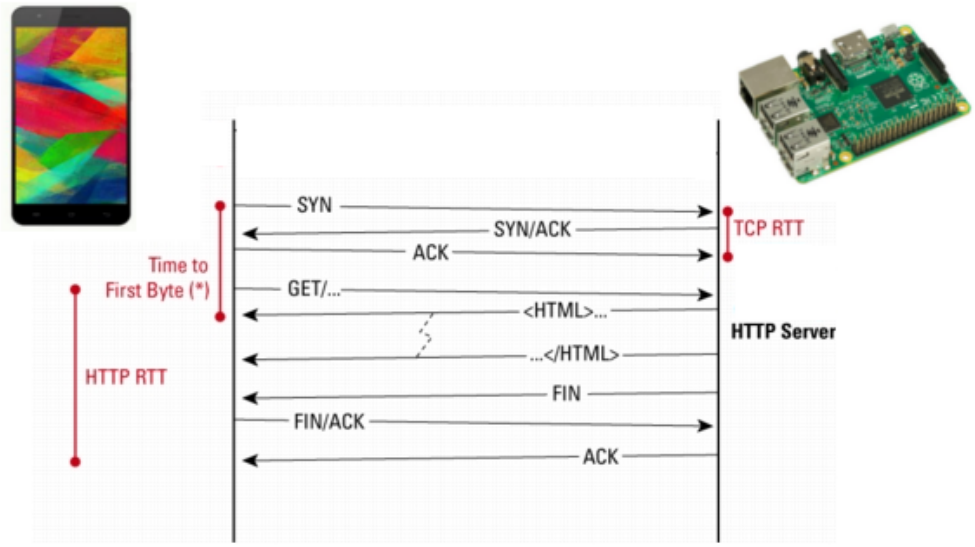


Figura 2.4 Comunicación cliente-servidor.

Se ha dejado la comunicación genérica en la figura, suponiendo respuestas HTML para indicar las diferentes posibles respuestas del servidor (que se detallarán en el capítulo 4 cuando se desarrolle el servidor web).

3 Hardware

El objetivo de este capítulo es enumerar los distintos dispositivos físicos utilizados en el proyecto, así como proporcionar una descripción y describir la configuración y montaje de los mismos. La elección de los dispositivos se ha basado en buscar un sistema que pudiera simular la CPU de los drones con relativa fidelidad y recursos limitados como los que pueden encontrarse en dispositivos empotrados, así como buscando una solución económica y de fácil acceso.

3.1 Dispositivos hardware utilizados

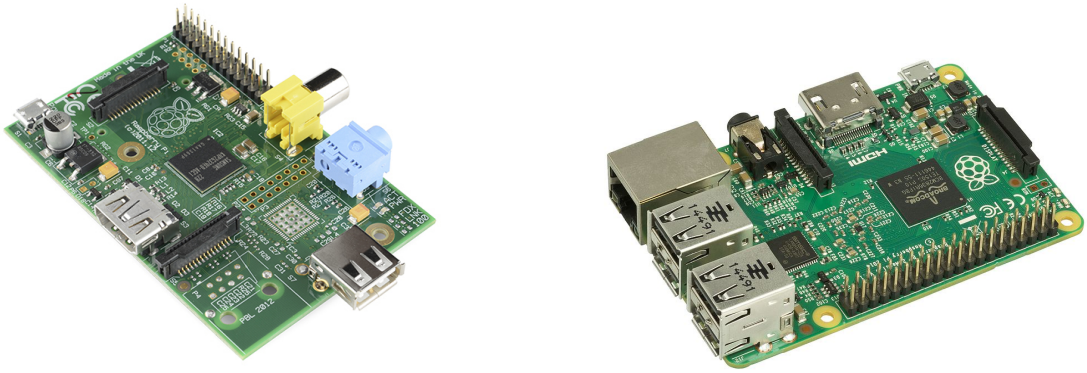
Dentro del sistema se distinguen dos tipos de elementos: los drones y el teléfono móvil del usuario. Con el objetivo de simular los drones se ha decidido utilizar una Raspberry Pi, mientras que para el teléfono móvil Android se ha optado por utilizar el modelo G3 de de la firma coreana LG.

3.1.1 Raspberry Pi

La Raspberry Pi es un ordenador, del tamaño de una tarjeta de crédito, originalmente diseñado para educación e inspirado por el Acorn BBC Micro de 1981 [8]. Fue diseñado y creado por el británico Eben Upton, cuyo objetivo era crear un dispositivo de bajo coste que pudiese ayudar a estudiantes a mejorar sus habilidades de programación y sus conocimientos hardware. Su pequeño tamaño y precio reducido han ayudado a aumentar su popularidad y uso en diferentes proyectos que requieren algo más que un simple microcontrolador (como podrían ser, por ejemplo, los dispositivos Arduino). Se trata de un dispositivo *open hardware*, con la excepción del chip primario, el Broadcomm SoC (*System on a Chip*), el cual se ocupa de muchas de las principales funciones de la placa como la CPU, los gráficos, la memoria, el controlador USB, etc.

Con respecto al software, los dispositivos Raspberry Pi fueron diseñados para el sistema operativo Linux, y muchas distribuciones Linux poseen versiones optimizadas para los mismos. Entre los más populares se encuentran Raspbian (basado en el sistema operativo Debian) y Pidora (basado en el sistema operativo Fedora), aunque existen muchas otras alternativas. OpenELEC y RaspBMC son distribuciones basadas en Linux cuyo objetivo es utilizar la Raspberry Pi como un *media center*. También existen alternativas no basadas en Linux, como RISC OS, aunque su uso no está tan extendido.

Existen en la actualidad distintos modelos de Raspberry Pi, los modelos A y B de primera generación, que fueron los primeros en salir al mercado, y versiones más modernas como la Raspberry Pi 2, la Raspberry Pi 3 y la Raspberry Pi Zero. Centrándonos en la primera generación, el modelo A cuenta con un puerto USB y 256MB de memoria RAM. Es más barato y tiene un menor consumo que el modelo B, el cual posee un segundo puerto USB, un puerto Ethernet para conexión a internet y una memoria RAM de 512MB. Ambos modelos poseen una versión revisada y actualizada, conocidas como modelos A+ y B+ respectivamente, los presentan mejoras tales como un mayor número de puertos USB y una mejora en el consumo de potencia. La Raspberry Pi de segunda generación incluye 1Gb de RAM así como una CPU a 900Mhz respecto al modelo B de la primera generación,, mientras que la Raspberry Pi 3 aumenta la velocidad de la CPU a 1.2Ghz y aporta además módulos de comunicación inalámbrica integrados tanto 802.11 como Bluetooth 4.1. La Raspberry Pi Zero es una versión de tamaño reducido del modelo A+.



(a) Raspberry Pi Modelo A. **(b)** Raspberry Pi Modelo B.

Figura 3.1 Modelos de Raspberry Pi..

Tamaño	86mm x 54mm
Chip	Broadcom BCM2835 SoC
Arquitectura de procesador	ARM11
CPU	700Mhz ARM1176JZFS
GPU	Dual core VideoCore IV Multimedia Co-Processor
Memoria	256MB SDRAM
Adaptador de corriente	Socket micro USB 5V, 2A
Ethernet	N/A
Salida de Video	HDMI, RCA (NTSC y PAL)
Salida de audio	3.5mm Jack, HDMI
USB	1 puerto USB 2.0
GPIO	24 pines 2.54mm
Conector de cámara	N/A
Socket SD	SD

Tabla 3.1 Especificaciones de la Raspberry Pi Modelo A.

Tamaño	85mm x 56mm
Chip	Broadcom BCM2835 SoC
Arquitectura de procesador	ARM11
CPU	700Mhz ARM1176JZFS
GPU	Dual core VideoCore IV Multimedia Co-Processor
Memoria	512MB SDRAM
Adaptador de corriente	Socket micro USB 5V, 2A
Ethernet	10/100 BaseT Ethernet socket
Salida de Video	HDMI, RCA (NTSC y PAL)
Salida de audio	3.5mm Jack, HDMI
USB	4 puertos USB 2.0
GPIO	40 pines 2.54mm
Conector de cámara	15 pines CSI-2
Socket SD	MicroSD

Tabla 3.2 Especificaciones de la Raspberry Pi Modelo B+.

Se ha decidido utilizar para este proyecto una Raspberry Pi modelo B+ y una Raspberry Pi modelo A, con el fin de trabajar con dos modelos diferentes y con distintos recursos accesibles al estar el modelo A más limitado en cuanto a especificaciones.

Con respecto a qué alternativas existen a la Raspberry Pi, existen numerosas opciones entre los ordenadores de tamaño reducido disponibles en el mercado. Una de las alternativas más populares son los dispositivos Arduino. Sin embargo, mientras que los dispositivos de la familia Raspberry Pi son ordenadores completamente funcionales corriendo un sistema operativo, la placa Arduino es tan solo un microcontrolador. Esto implica que no utiliza sistema operativo, si no que en su lugar ejecuta simplemente el código que tenga programado. A pesar de que existen añadidos que dotan de mayor funcionalidad a la placa, no deja de tratarse de un microcontrolador en primer lugar. Otra posible alternativa serían las placas Beaglebone, que a cambio de un mayor precio ofrecen más rendimiento, al igual las placas UDOO, también de mayor potencia y mayor precio. Otra de las ventajas de utilizar la Raspberry frente a sus competidores es el tamaño de la comunidad, pues al ser tan extensa, es mucho más fácil encontrar foros y documentación relativas a la misma. Es por todos estos motivos que se ha elegido a los dispostivos Raspberry frente al resto de opciones disponibles.

Perifericos

Con el fin de acceder y configurar las placas, se hace necesario el uso de distintos periféricos. Se tomará como referencia el modelo A de la Raspberry Pi, al estar más limitada en recursos.

- Al disponer de un solo puerto USB 2.0, se hará necesario el uso de un *hub* USB para poder ampliar el número de periféricos que podemos conectar a la placa al mismo tiempo. Cualquier modelo comercial de *hub* es válido, pues no se requiere ninguna especificación concreta. En cualquier caso, se recomienda utilizar un modelo con alimentación propia, pues dependiendo del número de periféricos conectados al mismo tiempo (y de las especificaciones de consumo de cada uno de ellos), es posible que la Raspberry Pi no sea capaz de alimentar a todos los dispositivos USB al mismo tiempo.
- Será necesario el uso de un teclado (y opcionalmente un ratón) USB 2.0 con el fin de poder utilizar y configurar correctamente las Raspberries. De nuevo no es necesaria ninguna especificación concreta, cualquier dispositivo genérico es válido, y su uso puede limitarse a una primera configuración del dispositivo en exclusiva, pues una vez configurado no será necesario volver a hacerlo.
- El periférico más importante del proyecto es el *dongle* WiFi USB, imprescindible para dotar de conectividad inalámbrica a las placas. Si bien existen modelos de Raspberry Pi con antenas WiFi incorporadas (Raspberry Pi 3), los utilizados en el proyecto carecen de las mismas como se comenta en la sección anterior. Existen numerosas opciones en el mercado a precios económicos. Entre los dispositivos más populares, por citar alguno, se encuentra por ejemlo el TL-WN725N 150Mbps Wireless N Nano USB Adapter de la marca TP-Link. Sin embargo, la elección del dispositivo no es trivial: como es sabido, la Raspberry Pi correrá una distribución Linux, y por tanto deberá disponer de los drivers necesarios para poder utilizar el *dongle*. El TL-WN725N es un dispositivo preparado para trabajar en sistemas operativos Windows, por lo que se ha optado por utilizar otro modelo para el presente proyecto: el EW-7811Un de Edimax Technology. Este dispositivo soporta un bitrate de hasta 150Mbps bajo el estándar 802.11n, soportando los estándares IEEE802.11b/g/n, y dando soporte también para WPS. El dispositivo adapta la salida de transmisión en función de la distancia y la carga de CPU, reduciendo el consumo cuando la conexión inalámbrica está inactiva hasta un 50 %. Además se trata de un dispositivo *plug and play* al no requerir ningún tipo de instalación, pues la distribución de Linux más utilizada en Raspberry Pi (Raspbian) incorpora en su kernel los drivers necesarios para su uso.

Figura 3.2 Edimax EW-7811.

3.1.2 Smartphone

El único requisito para poder ejecutar la aplicación es que el dispositivo Android disponga de conectividad WiFi, algo estándar en inmensa mayoría de terminales disponibles hoy en día en el mercado. Tampoco es necesaria ninguna versión específica del sistema operativo, pues el software es retrocompatible y funcionará tanto en las últimas versiones como en otras más antiguas. En el proyecto que nos ocupa se ha utilizado un teléfono LG G3 de la marca coreana LG por el mero hecho de la disponibilidad del mismo, que ni siquiera debe ser obligatoriamente un smartphone: también podría utilizarse un dispositivo tablet con antena WiFi.

Figura 3.3 LG G3.

3.2 Configuracion y montaje de los dispositivos

El montaje de las Raspberries tan sólo necesita de una fuente de alimentación micro USB de +5V. El amperaje dependerá de los periféricos que conectemos a la Raspberry: el modelo B consume entre 700 y 1000mA, mientras que el modelo A consume poco más de 500mA sin ningún periférico. En el desarrollo de este proyecto se ha utilizado una fuente de 5V y 2A.

Los periféricos (el teclado cuando sea necesario y el *dongle* WiFi) tan sólo necesitan ser conectados al puerto USB, mientras que si queremos disponer de una pantalla deberemos hacer uso del puerto HDMI de la placa. Tanto la pantalla como el teclado tan sólo serán necesarios para el desarrollo y mantenimiento de la Raspberry, y no son indispensables para su correcto funcionamiento como servidor tras la configuración software.

Además de esto, será necesaria la tarjeta SD de la Raspberry (o micro SD, dependiendo del modelo). Para ello se hará uso de NOOBS (*New Out Of the Box Software*, Software nuevo fuera de la caja, o "nada más sacarlo de la caja"), un asistente de instalación de sistemas operativos para Raspberry Pi. NOOBS tiene dos versiones: la versión estándar, que incluye ya el sistema operativo Raspbian (más una selección de otros sistemas operativos y gestores de contenido para Raspberry Pi que se descargan de Internet) y la versión *lite*, que no trae Raspbian preinstalado y simplemente descargará el sistema operativo que elijamos de Internet. Tras formatear la tarjeta SD, tan sólo hemos de descomprimir el contenido de NOOBS y copiarlo en la

tarjeta, que quedará lista en ese momento para usar en la Raspberry. NOOBS se ejecuta de forma automática, y muestra al usuario una ventana donde se puede elegir el sistema operativo que se desea instalar (desde el menú también se accede a un menú inferior donde cambiar de idioma y de configuración de teclado). Bastará entonces con seleccionar Raspbian (en este proyecto se ha elegido Raspbian por ser la distribución más extendida en Raspberry Pi y por tanto la que goza de más y además más constantes actualizaciones y mantenimiento, aunque otras distribuciones también son válidas) de la lista para comenzar la instalación.

4 Software

A lo largo de esta sección se pretende explicar el código que se ha desarrollado para la aplicación, comenzando por la estructura del proyecto, y comentando posteriormente las clases y las actividades (*activities*). Los códigos fuente completos de las mismas pueden encontrarse en el Anexo B de esta misma memoria, siendo el objetivo de esta sección comentar las responsabilidades de cada clase/actividad a un nivel de abstracción mayor, sin pretender entrar con demasiada profundidad en el código de las mismas, en la medida de lo posible.

4.1 Aplicación smartphone

Como se ha comentado anteriormente, el sistema operativo elegido para el desarrollo de la aplicación será **Android**, el sistema operativo móvil desarrollado por Google, basado en el kernel de linux, y principalmente diseñado para dispositivos táctiles como teléfonos inteligentes o *tablets*. Se trata del sistema operativo más instalado del mundo [9], siendo el sistema operativo más vendido en *tablets* desde 2013 [10], y predominante en el mundo de los *smartphones* sin ninguna duda, extendiéndose de los dispositivos de gama baja hasta dispositivos de lujo, el cual es uno de los principales motivos para su elección como sistema operativo para el que desarrollar la aplicación.

Para el desarrollo de la aplicación se ha elegido utilizar **Android Studio**, el IDE (entorno de desarrollo integrado) oficial de Google. Si bien existen otras posibilidades en el mercado (siendo otra de las opciones más populares Eclipse), Android Studio ofrece ciertas ventajas sobre los demás al tratarse del único al que Android soporta de forma oficial: por ejemplo, con respecto al sistema de *build*, el IDE de Google nos ofrece **Gradle**, un sistema basado en Apache Maven que introduce su propio DSL (*Domain Specific Language*, Lenguaje de dominio específico) con el fin de obtener una mayor versatilidad en los scripts (al contrario que Eclipse, por ejemplo, que utiliza Apache And, basado en xml y por tanto más rígido y menos versátil), así como mejores herramientas de autocompletado de código y refactorizado, desarrolladas específicamente para el código de Android, que se añaden a las ya presentes en IntelliJ IDEA, el IDE de JetBrains en el que se basa Android Studio. Ofrece también herramientas para previsualizar (e incluso editar de manera visual) las distintas actividades.

Android Studio provee además la estructura base de una aplicación Android con cada nuevo proyecto que se inicialice, reduciendo la laboriosa tarea de tener que crearla a mano por el desarrollador y tener que escribir toda la configuración para Gradle. Es por ello que, a lo largo de esta sección, se ignorará esta configuración (así como otros muchos ficheros) generados de forma automática por el IDE -llamados normalmente por su denominación inglesa *boilerplate*, código necesario para el funcionamiento de la aplicación pero usualmente generado automáticamente y que no forma parte de la aplicación como tal-, pues no son realmente relevantes para el objetivo de esta memoria. Nos centraremos, por tanto, en el código desarrollado específicamente para este proyecto, dejando de lado tanto el proceso de *build* como otros aspectos básicos de la estructura de un proyecto de aplicación Android.

4.1.1 Estructura del proyecto

La estructura del proyecto se representa en el siguiente árbol de directorios:

```
/
├── .gradle/
├── .idea/
├── app/
│   ├── build/
│   ├── libs/
│   ├── src/
│   │   └── main/
│   │       ├── java/
│   │       │   └── com.pfc.app.iHelp/
│   │       │       ├── activities/
│   │       │       │   ├── HelpMessageActivity.java
│   │       │       │   ├── MainActivity.java
│   │       │       │   ├── MainMenuActivity.java
│   │       │       │   └── RescueMessageActivity.java
│   │       │       └── classes/
│   │       │           ├── GPSHandler.java
│   │       │           ├── RestClientTask.java
│   │       │           ├── ToastHandler.java
│   │       │           └── WifiHandler.java
│   │       ├── res/
│   │       │   ├── drawable/
│   │       │   ├── layout/
│   │       │   │   ├── activity_help_message.xml
│   │       │   │   ├── activity_main.xml
│   │       │   │   ├── activity_main_menu.xml
│   │       │   │   └── activity_rescue_message.xml
│   │       │   ├── mipmap-hdpi/
│   │       │   ├── mipmap-mdpi/
│   │       │   ├── mipmap-xdpi/
│   │       │   ├── mipmap-xxdpi/
│   │       │   ├── mipmap-xxxdpi/
│   │       │   ├── values/
│   │       │   │   ├── colors.xml
│   │       │   │   ├── dimens.xml
│   │       │   │   ├── strings.xml
│   │       │   │   └── styles.xml
│   │       │   ├── values-v21/
│   │       │   └── values-w820dp/
│   │       └── AndroidManifest.xml
│   ├── app.iml
│   ├── build.gradle
│   └── proguard-rules.pro
├── build/
│   ├── generated/
│   └── intermediates/
├── gradle/
├── build.gradle
├── gradle.properties
├── gradlew
├── gradlew.bat
├── local.properties
├── MyApplication.iml
└── settings.gradle
```

Con el objetivo de mantener una separación entre actividades y clases, se han creado sendos directorios dentro del paquete iHelp. Aunque realmente no todo el código está contenido aquí (pues, como se verá más adelante, el layout de las actividades está definido dentro de la carpeta *res*), se utilizará esta separación conceptual a lo largo del capítulo para detallar la implementación de la aplicación.

4.1.2 *Activities*

Una ***activity***, o actividad, es una abstracción de Android que se define sencillamente como una sola acción que el usuario puede llevar a cabo. Casi todas las actividades interactúan con el usuario, de modo que la clase `Activity` se encarga de crear una ventana en la cual se coloca la UI. Normalmente las actividades se presentan al usuario como ventanas a pantalla completa, aunque pueden anidarse múltiples actividades dentro de otras. Es importante destacar que todas las actividades, para poder comenzar, han de declarar la etiqueta `<activity>` dentro del fichero `AndroidManifest.xml`.

Las actividades se manejan mediante una pila. Cuando una nueva actividad empieza, se pone "encima" de la pila y se convierte en la aplicación que está corriendo, quedando la que estuviese corriendo anteriormente en segundo plano, y no volverá al primer plano mientras exista otra actividad por encima de ella en la pila. El ciclo de vida de una actividad, o *lifecycle*, es el conjunto de estados en los que una actividad puede encontrarse desde el momento de su creación hasta el momento de su destrucción.

El siguiente diagrama muestra los diferentes caminos que puede seguir el *lifecycle* de una actividad en Android:

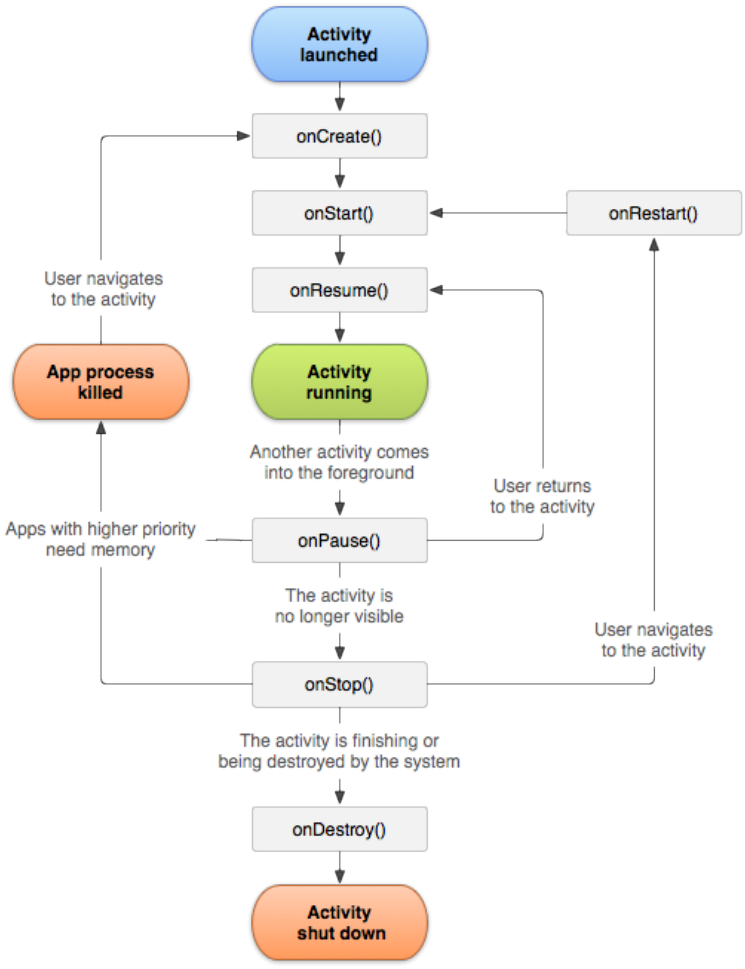


Figura 4.1 Ciclo de vida de una actividad en Android.

Existen por tanto cuatro estados:

- Si una actividad se muestra en la pantalla, está encima de la pila, y es por tanto la actividad activa.
- Si una actividad pierde el foco pero sigue estando visible (es decir, que otra actividad que no está a pantalla completa está por encima), entonces la actividad está pausada. Esto quiere decir que está viva (mantiene su estado y la información que estuviese en proceso sigue ahí), pero puede ser destruida por el sistema en caso de recursos memoria extremadamente baja.
- Si una actividad está complemente oculta por otra actividad, está parada. Mantiene el estado y la información, pero no es visible y será destruida por el sistema operativo en cuando la memoria haga falta para cualquier otro recurso.
- Si una actividad está pausada o parada, el sistema puede quitarla de la memoria bien "pidiéndole" que termine o directamente matando el proceso. Cuando se muestre de nuevo, deberá ser completamente reseteada.

MainActivity

Esta sencilla actividad es el punto de entrada de la aplicación desarrollada en este proyecto. Visualmente se ofrece al usuario un botón con el texto *Start scanning*, que será quien, mediante el método `onClick`, llame a la función `startScanning` de nuestra actividad. Este método tan sólo lanza una nueva actividad, llevándonos al menú principal, representado por `MainMenuActivity`.

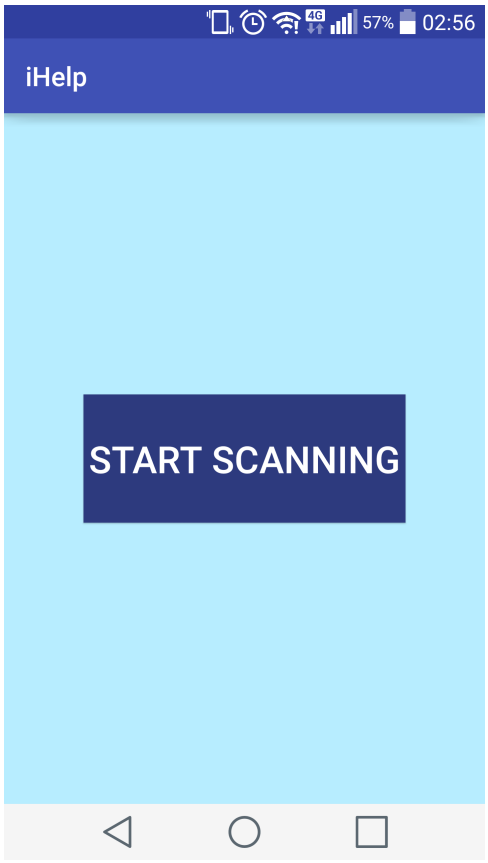


Figura 4.2 Punto de entrada de la aplicación.

MainMenuActivity

Esta actividad es, visualmente, el menú principal de la aplicación, desde el que se da la posibilidad al usuario de solicitar la localización del punto de ayuda más cercano o bien de pedir un rescate.

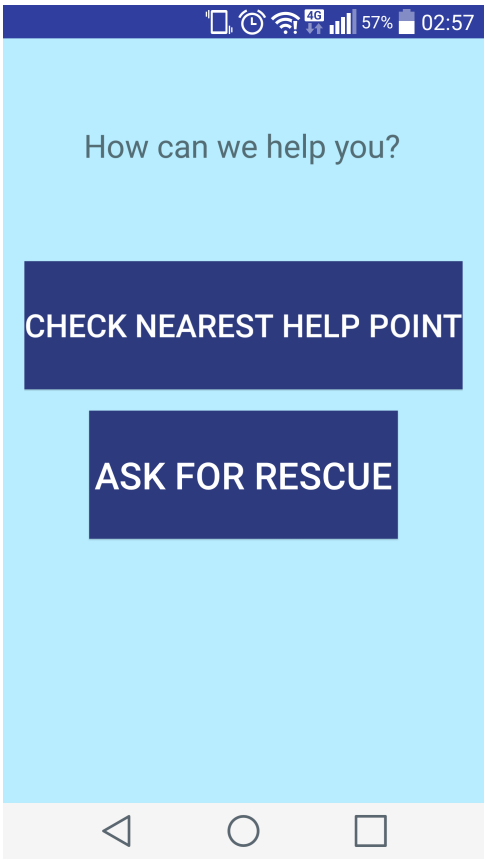


Figura 4.3 Menú principal de la aplicación.

Sin embargo, a nivel lógico, se trata de la actividad más importante de la aplicación. En primer lugar, en el método `onCreate` (el cual es invocado al instanciar la actividad), se instancian los distintos objetos del apartado clases, esto es, `toastHandler`, `wifiHandler` y `gpsHandler`. Acto seguido, es este menú principal quien se encarga de llamar al método `scanNetworksAndConnectToBest` del objeto wifiHandler, y también de asegurarse de que la aplicación tiene los permisos necesarios para acceder al GPS del teléfono antes de, utilizando un *ThreadPoolExecutor*, lanzar un hilo con la tarea del GpsHandler.

En el caso en el que el servicio de localización no esté activado, la aplicación mostrará un mensaje al usuario indicándole que deberá activarlo para poder utilizar la aplicación, proveyendo un acceso directo a la sección de ajustes del teléfono que permite hacerlo (la función `showAlert` se encargará de esto). En caso de que no se pueda obtener una localización GPS, la aplicación mostrará un mensaje indicándolo e impedirá el uso de los botones para realizar las llamadas a la API. Esta lógica se implementa en el método `disableButtonsAndShowMessage`.

Esta actividad tiene también, por supuesto, los *callbacks* asociados a los eventos *click* de la interfaz, llamados respectivamente `help` y `rescue`, quienes, haciendo uso de la tarea `RestClient`, se encargarán de realizar la llamada correspondiente a la API, de manejar el error, y de lanzar las actividades correspondientes para informar al usuario con la respuesta del servidor. En caso de un error de red causado por la pérdida de conexión, la aplicación se encargará de relanzarse (mediante el método `handleNetworkError`).

HelpMessageActivity

El objetivo de esta actividad es simplemente mostrar al usuario un mensaje con la respuesta del servidor, de forma que pueda acudir, dentro del escenario afectado por el desastre ocurrido, a un punto de ayuda. Para ello, además de mostrar la información que provee la Raspberry (un nombre identificable de la localización, las coordenadas GPS y la distancia a la que se encuentra), esta actividad cuenta con el método privado `getDirection`, del cual hace uso para poder indicar al usuario en qué dirección se encuentra su destino.

```
private String getDirection (Double deviceLat, Double deviceLon, Double lat, Double lon) {
    if (lat > deviceLat) {
        if(lon >= deviceLon + 0.1) {
            return "North-East";
        } else if ( deviceLon > lon - 0.1 && deviceLon < lon + 0.1 ) {
            return "North";
        } else {
            return "North-West";
        }
    } else {
        if(lon >= deviceLon + 0.1) {
            return "South-East";
        } else if (deviceLon > lon - 0.1 && deviceLon < lon + 0.1) {
            return "South";
        } else {
            return "South-West";
        }
    }
}
```

Código 4.1 getDirection.

El objetivo de esta actividad es tratar de dar la máxima información posible al usuario de forma que pueda localizar el punto de ayuda más cercano a su posición de la forma más sencilla posible, dada la imposibilidad de utilizar servicios más completos como Google Maps al no disponer de conexión a internet en una situación de emergencia, debido al daño recibido por la infraestructura de comunicaciones o bien por la saturación de la misma debido al uso masivo de un elevado número de dispositivos al mismo tiempo. La red de los drones no provee de servicios de conexión a internet, si no que provee comunicación inalámbrica a un sistema específico de ayuda para las víctimas.

RescueMessageActivity

De nuevo, el fin de esta actividad es simplemente mostrar al usuario la respuesta del servidor tras llamar a la ruta `/rescue`. En este caso es tan sencillo como simplemente asignar el valor del texto al elemento correspondiente en la vista, tomándolo directamente de los parámetros con los que se ha invocado a la actividad, sin requerir lógica adicional como en el caso anterior. Esto se puede apreciar en el siguiente fragmento de código extraído diretamente del código de la actividad:

```
        (...)

        textView = (TextView) findViewById(R.id.textView2);

        String text = getIntent().getStringExtra("message");

        changeText(text);
    }

    public void changeText(String text) {
        textView.setText(text);
    }
```

Código 4.2 Fragmento de RescueMessageActivity.

4.1.3 Clases

GPSHandler

`GPSHandler` implementa la interfaz *Callable* con el fin de poder ejecutarse en segundo plano en un *ThreadPoolExecutor*, de forma que pueda realizar operaciones sin bloquear el hilo principal de ninguna actividad. Para ello, implementa el método `call()`. Este método se encargará de utilizar una de las mútliples abstracciones que el sistema operativo Android provee para el manejo de la localización de usuario, `LocationListener`, con el objetivo de obtener la última localización del dispositivo. Este objeto se encarga de ejecutar ciertos

métodos cuando haya cambios en el estado de la localización: *onLocationChanged*, *onStatusChanged*, *onProviderEnabled* y *onProviderDisabled*.

El *LocationListener* correrá en un hilo y se encargará de monitorizar el estado de la localización, mientras que el método `getLocation` proveerá la última localización conocida en el momento en que es llamado.

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

Código 4.3 Declaración de los permisos necesarios para GPSHandler en AndroidManifest.xml.

RestClientTask

Esta clase extiende a la abstracción de Android *AsyncTask*, que provee una forma de realizar operaciones en *background* con el fin de publicar los resultados en el hilo de UI sin necesidad de manejar hilos o concurrencia a más bajo nivel. En realidad, no es más que una clase construida sobre *Thread* y *Handler*, que no constituye un verdadero *framework* de concurrencia: la idea es utilizar `AsyncTask` para operaciones cortas, que no duren más de unos cuantos segudos.

`RestClientTask` sobreescribe el método `doInBackground` de `AsyncTask` donde se encarga de realizar una llamada http al servidor de la Raspberry, para posteriormente procesar la respuesta y devolver un objeto JSON al hilo principal de interfaz de usuario desde el que se invocó la tarea. Es responsabilidad también de la tarea el configurar la conexión antes de realizar cualquier llamada al servidor.

De nuevo es importante señalar que el permiso `INTERNET` será necesario para que nuestra aplicación pueda realizar llamadas http:

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

Código 4.4 Declaración de los permisos necesarios para RestClientTask en AndroidManifest.xml.

ToastHandler

Un *toast* es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento).

El objetivo de esta clase es proveer una pequeña capa de abstracción sobre la clase Toast de Android, pudiendo tener una sola instancia en toda la aplicación desde la que poder mostrar mensajes al usuario sin necesidad de acceder directamente a Toast en cada ocasión en que sea necesario. Provee además una forma de centralizar la configuración de los *toasts*, pues en caso de necesitar cambiar la configuración, tan sólo habremos de acudir a esta clase para editarla, y no tener que cambiar cada *toast* de manera individual.

WifiHandler

La clase WifiHandler pretende albergar toda la lógica relacionada con la conexión inalámbrica (relacionada con la conexión, no con la comunicación, esto es, no se ocupará de la comunicación entre la aplicación Android y el servidor web, si no de la conectividad inalámbrica). El punto de entrada de la lógica de la clase es el método `scanNewtorksAndConnectToBest`, el cual es llamado desde la actividad del menú principal.

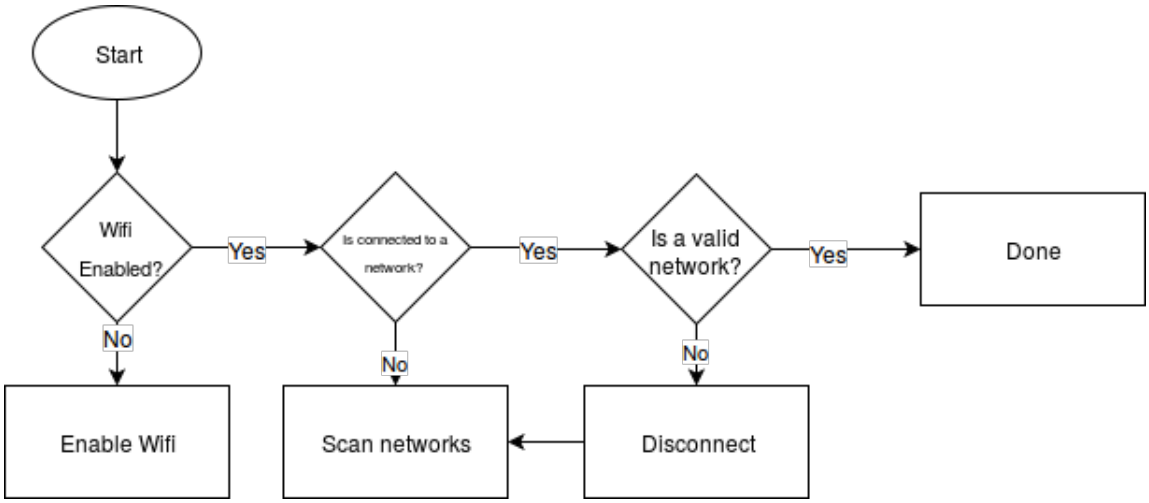


Figura 4.4 Lógica de conexión inalámbrica.

En la figura se explica el funcionamiento del algoritmo para la conexión a una red inalámbrica. El criterio para elegir si una red es válida o no se basa en el SSID, pues se asume que todas las redes inalámbricas estarán anotadas con las siglas "RP" (Raspberry Pi). La referencia de la figura a *ScanNetworks* se refiere al método `scanNetworksAndConnectToBest`. Este método utiliza, al igual que el resto del WifiHandler, la abstracción que provee la API de Android para manejar las conexiones inalámbricas, WiFiManager. Haciendo uso del mismo es capaz de iterar sobre las conexiones disponibles tras el escaneo, para seleccionar aquella cuya intensidad recibida en dB sea mayor. Es importante destacar que, para que la clase WifiHandler pueda llevar a cabo todas estas tareas, hemos de declarar en el manifiesto de nuestra aplicación que hará uso de los permisos `ACCESS_WIFI_STATE` y `CHANGE_WIFI_STATE`:

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

Código 4.5 Declaración de los permisos necesarios para WifiHandler en AndroidManifest.xml.

Una vez detectada la red inalámbrica con mayor potencia, WifiHandler comprueba si se trata de una red conocida (es decir, configurada pues el dispositivo se ha conectado a ella en algún momento en el pasado), con el fin de crear la configuración correspondiente si esta no está presente.

4.2 Configuración de la Raspberry Pi

Para que la Raspberry Pi sea capaz de emitir WiFi, será necesaria la instalación y configuración de dos servicios: *hostapd* y *udhcpd*, además de la instalación del entorno node.js para poder correr la aplicación que hará de servidor.

4.2.1 udhcpd

udhcpd es un servidor DHCP ligero, pensado para sistemas embebidos. Su nombre proviene de la abreviación de micro (μ, **u**), el protocolo DHCP (**dhcp**) y su propia naturaleza como *daemon* (**d**). Se trata de un programa completamente funcional, que cumple la RFC 2131. Fue originalmente desarrollado en 1999 por Matthew Ramsay y distribuido bajo licencia GNU GPL por Moreton Bay.

Instalación, configuración y puesta en marcha

El método de instalación más común y sencillo es utilizar los paquetes incluidos en los repositorios configurados en el gestor de paquetes de cualquier distribución Linux, pues udhcpd está incluído en la gran mayoría. En Raspbian, por ejemplo, bastaría con introducir en una terminal:

```
sudo apt-get install udhcpd
```

La configuración del servicio es muy sencilla. *udhcpd* utiliza un solo fichero principal de configuración, `/etc/udhcpd.conf`. Con cualquier editor de texto y permisos de superusuario, el fichero debe quedar como sigue:

```
start 192.168.42.2
end 192.168.42.20
interface wlan0
remaining yes
opt router 192.168.42.1
opt lease 864000
```

Código 4.6 udhcpd.conf.

`start` y `end` delimitan el rango de IPs que el dispositivo podrá proporcionar a los clientes. En este caso, se dispone de 18 direcciones IP. `interface` indica la interfaz de red a utilizar por el servicio. Las líneas que comienzan con `opt` son parámetros opcionales, pero necesarios para la configuración deseada. `router` indica la IP de la Raspberry Pi en la interfaz wlan0, y `opt lease` indica el tiempo, en segundos, de "préstamo" que cada cliente tendrá asignada la dirección IP (en este caso, 864000 segundos equivalen a diez días).

Una vez configurado, aún nos queda otro paso, y es activar DHCP. Para ello, editamos el fichero bajo la ruta `/etc/default/udhcpd`, de forma que su contenido quede:

```
# Comment the following line to enable
# DHCPD_ENABLED="no"
```

Código 4.7 /etc/default/udhcpd.

Se puede observar que tan sólo hemos comentado (añadiendo, al principio de la línea, el símbolo #) la configuración que deshabilita DHCPD, habilitándolo de esta forma.

4.2.2 hostapd

hostapd es un *daemon* (demonio) de espacio de usuario para servidores de punto de acceso y autenticación. Implementa puntos de acceso IEEE 802.11, autenticadores IEEE 802.1X/WPA/WPA2/EAP, cliente RADIUS, servidor EAP y servidor de autenticación RADIUS. La última versión soporta Linux (driver Host AP entre otros) y FreeBSD (driver net80211). Está diseñado para ser un programa *daemon* que corra en segundo plano (*background*) y actúe como un componente backend controlando la autenticación, aunque también existen programas cliente frontend que permiten su configuración como *hostapd_cli*, incluido en la distribución de *hostapd*.

Se enumeran a continuación las características WPA/IEEE 802.11i/EAP/IEEE 802.1X soportadas por hostapd:

- WPA-PSK ("WPA-Personal")
- WPA con servidor EAP o servidor RADIUS de autenticación externo ("WPA-Enterprise")
- Gestión de claves CCMP, TKIP, WEP104, WEP40
- WPA y IEEE 802.11i/RSN/WPA2 al completo
- RSN: caché PMKSA, pre-autenticación
- IEEE 802.11r
- IEEE 802.11w
- Wi-Fi Protected Setup (WPS)

Instalación, configuración y puesta en marcha

Al igual que con udhcp, podemos utilizar los repositorios de nuestra distribución para instalar el software. En Raspbian bastaría con introducir en un terminal:

```
sudo apt-get install hostapd
```

Para el caso concreto que nos ocupa, utilizando el *dongle* WiFi Edimax EW7811, el proceso no es tan sencillo. Si bien el dispositivo es *plug and play* en cuanto a conectividad inalámbrica (bastará conectarlo a un puerto USB de la Raspberry para que ésta lo detecte y sea capaz de conectarse a redes inalámbricas,

pues el kernel Raspbian incluye los drivers necesarios), para conseguir que se comporte como un punto de acceso, el proceso es algo más complicado, y será necesario disponer del driver nl80211. El por qué es el siguiente: el EW7811 está basado en el chip Realtek RTL8188CUS. Este chip no soporta el driver estándar nl80211 de *hostapd*, y sus drivers *in-kernel* están limitados en características, por lo que la propia Realtek publicó en su web los drivers para este chip bajo licencia GPL. Será necesario entonces utilizar una versión parcheada de hostapd para incluir estos drivers, o bien hacerlo nosotros mismos. Gracias a la gran comunidad *open source*, no es difícil encontrar dichas versiones ya parcheadas de *hostapd* publicadas por reconocidos ingenieros *senior* expertos en la materia, como Pritam Baral (https://github.com/pritambaral) o Jens Segers (https://github.com/jenssegers).

Si tomamos este último, por ejemplo, el procedimiento de instalación de *hostapd* es sencillo. En primer lugar, y dado que vamos a instalar una versión modificada de *hostapd*, hemos de eliminar cualquier instalación anterior. En una terminal, bastará con ejecutar:

```
sudo apt-get autoremove hostapd
```

El siguiente paso es descargar el código fuente en la Raspberry. Bien sea clonando el repositorio de git, o descargando directamente una versión comprimida del proyecto (pues no nos interesa modificar el código), comenzamos con la instalación. Si nos decantamos por la opción de la descarga directa, los siguientes comandos de terminal descargarán el fichero y lo descomprimirán:

```
wget https://github.com/jenssegers/RTL8188-hostapd/archive/v2.0.tar.gz
tar -zxvf v2.0.tar.gz
```

Una vez descomprimido, tan solo hemos de movernos al directorio que contiene el código y hacer uso del fichero Make para compilar *hostapd*.

```
cd RTL8188-hostapd-2.0/hostapd
sudo make
```

Y, una vez tengamos de nuevo control de la terminal, instalar el programa:

```
sudo make install
```

Tras la instalación, se creará el binario en la ruta `/usr/local/bin`, se creará un script de arranqe y se creará un fichero de configuración en `/etc/hostapd/hostapd.conf`. Este es el fichero de configuración principal de *hostapd*, el cual deberemos abrir con el editor de texto de nuestra preferencia para editarlo y configurar el servicio.

Hostapd nos ofrece la posibilidad de crear redes seguras protegidas por contraseña, pero el objetivo del proyecto es conseguir comunicar fácilmente el smartphone con la Raspberry, por lo que se configurará para utilizar una red abierta. Los contenidos del fichero quedan, entonces:

```
interface=wlan0
ssid=Raspberry-Pi
hw_mode=g
channel=6
auth_algs=1
wmm_enabled=0
```

Código 4.8 hostadp.conf.

En la primera linea, `interface` representa la interfaz de red a utilizar, en este caso será la interfaz inalámbrica de la Raspberry Pi, wlan0. `ssid` es el nombre de la red inalámbrica, que será incluido en todos los paquetes de la misma para identificarlos como parte de esa red, `hw_mode` configura la frecuencia de la señal (el valor `g` la fija a 2.4Ghz), `channel` indica el canal a utilizar, `auth_algs` indica los algoritmos de autenticación a utilizar (1 = wpa, 2 = wep, 3 = ambos) y `wmm_enabled` configura el soporte para QoS en Wifi, que en este caso no será necesario.

Por último, habremos de editar también el fichero `/etc/default/hostapd`, para indicar que el fichero `/etc/hostapd/hostapd.conf` será el que contiene la configuración del servicio:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Código 4.9 /etc/default/hostadp.

4.2.3 node.js

node.js es un entorno (*runtime*) de código abierto de Javascript, construido sobre el motor V8 de Chrome, con la finalidad de construir diferentes herramientas y aplicaciones en el lado servidor. Utiliza una arquitectura basada en eventos no bloqueantes, que provee la posibilidad de entrada/salida asíncronas, lo cual permite optimizar el tráfico y escalabidad de aplicaciones web con numerosas operaciones de entrada/salida.

Los desarrolladores, gracias a node, pueden crear servidores fácilmente escalables sin necesidad de acudir a la programación multi-hilo, utilizando el modelo de eventos, que utiliza *callbacks* para indicar la finalización de una tarea. De hecho, uno de los motivos para la creación de node.js fue el hecho de que la concurrencia es complicada en un gran número de lenguajes de programación y suele conllevar un pobre rendimiento. Es por ello que, unido a la ventaja del extendido uso de Javascript en el mundo de la programación web, la comunidad de node.js le ha hecho extremadamente popular en los últimos años.

Esta gran comunidad ha desarrollado cientos de aplicaciones y librerías de código abierto para node.js, la gran mayoría de ellas alojadas en los servidores de npm (*Node Package Manager*), el gestor de paquetes de node.js que nos permitirá de forma sencilla instalar y desinstalar librerías y resolver dependencias.

Con el fin de instalar y administrar de forma sencilla el entorno node.js (esto es, node.js y npm), haremos uso de **nvm** (*Node Version Manager*). Se trata de un script de bash que tiene la capacidad de manejar múltiples versiones activas de node.js, y permite instalar, ejecutar comandos en una versión de node.js determinada, asignar el valor a la variable PATH para usar una versión determinada de node.js, entre otras muchas funcionalidades. Para instalar nvm, hemos de obtener el código fuente en primer lugar. Para ello hacemos uso del software de control de versiones git, y ejecutando los siguientes comandos, obtendremos nvm en el directorio `/.nvm`:

```
git clone https://github.com/creationix/nvm.git ~/.nvm
cd ~/.nvm
git checkout v0.25.4
```

Habremos ahora de editar los ficheros `/.bashrc` y `/.profile`, añadiendo la línea:

```
source ~/.nvm/nvm.sh
```

al final de ambos. Para ello necesitaremos permisos de super usuario, y podemos hacer uso de cualquier editor de texto. Tras esto, será necesario reiniciar el dispositivo para que los cambios se apliquen:

```
sudo reboot
```

Una vez tengamos la terminal disponible de nuevo, podemos comprobar que la instalación se ha llevado a cabo de forma correcta ejecutando el comando:

```
nvm --version
```

que deberá retornar el valor de la versión que se ha instalado (en nuestro caso, `v0.25.4`).

Una vez instalado correctamente `nvm`, es hora de instalar node.js. Para ello, bastará con ejecutar:

```
nvm install 0.10.26
npm update -g npm
```

Se trata de un proceso sencillo, pues `nvm` es quien se encargará de la instalación por el usuario, teniendo tan sólo que especificar qué versión de node.js se desea instalar. Para utilizar de ahora en adelante la versión recién instalada de node.js, habremos de ejecutar:

```
nvm use 0.10.26
```

A partir de ahora, con tal sólo utilizar el comando `node`, tendremos accesible una consola de node.js, en la que podremos ejecutar comandos o correr nuestro código Javascript.

4.2.4 Configuración adicional y arranque de servicios

No hemos de olvidarnos de configurar, desde la terminal, la interfaz inalámbrica para utilizar la dirección IP configurada en el fichero `/etc/udhcpd.conf`. Para ello, ejecutamos el comando:

```
sudo ifconfig wlan0 192.168.42.1
```

Esto configura la interfaz la interfaz wlan0 para tener la IP que deseamos. Sin embargo, no es un cambio permanente, y se perderá en el próximo reinicio del sistema. Para hacer este cambio permanente y no tener que preocuparnos de la configuración cada vez que la Raspberry se arranque, editaremos el fichero de configuración de interfaces de red, `/etc/network/interfaces` para que incluya:

```
iface wlan0 inet static
  address 192.168.42.1
  netmask 255.255.255.0
  network 192.168.42.0
  broadcast 192.168.42.255

#allow-hotplug wlan0
#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet manual
```

Código 4.10 Cambios en /etc/network/interfaces.

Es importante notar que estamos comentando las tres últimas lineas. Comentar `allow-hotplug` previene el inicio de la interfaz cuando el kernel detecte el evento *hotplug* en la interfaz, mientras que comentar la linea `wpa-roam` evita la configuración de de wpa_supplicant (que queda fuera de los objetivos del proyecto).

Tan sólo nos queda ahora arrancar ambos servicios, de nuevo haciendo uso de una terminal:

```
sudo service hostapd start
sudo service udhcpd start
```

Estos cambios pueden también hacerse persistentes, de forma que ambos servicios se inicien al arrancar el sistema operativo, ejecutando:

```
sudo update-rc.d hostapd enable
sudo update-rc.d udhcpd enable
```

`update-rc.d` actualiza los enlances de scripts de arranque sistema `/etc/rc`runlevel`.d/NN`nombre cuyo objetivo es el script `/etc/init.d/`nombre ("NNnombre" es una cadena genérica que puede tomar varios valores, tal y como se explica en los párrafos que siguen).

Será necesario hacer una última modificación. Para que *udhcpd* arranque de manera correcta es necesario que las interfaces de red estén configuradas de forma adecuada (esto es, en el caso que nos ocupa, que `wlan0` tenga asignada la IP estática correspondiente). Para ello, debemos asegurarnos que el servicio `networking` está arrancado cuando lo haga *udhcpd*. Con el fin de hacerlo, deberemos realizar una pequeña modificación a nuestro sistema, creando *symlinks* (enlaces simbólicos) a los scripts de arranque correspondientes a los servicios (localizados en `/etc/init.d`). En primer lugar podemos averiguar el *run level* en el que nuestro sistema opera normalmente haciendo uso del comando:

```
runlevel
```

En el caso de las Raspberries, el resultado es 2. En consecuencia, habremos de acudir al directorio `/etc/rc2.d/` (el directorio cambiará dependiendo del *run level*, si este por ejemplo fuese 3, habríamos de acudir a `/etc/rc3.d/`) y crear los links simbólicos mencionados anteriormente. En una terminal, bastará con ejecutar:

```
sudo ln -s /etc/init.d/networking /etc/rc2.d/K01networking
```

Los enlaces de este directorio siguen una nomenclatura particular, como se puede apreciar. Todos van precedidos por K (*kill*) o S (*start*), y un número, indicando el orden en el que se ejecutarán. Con la ejecución del anterior comando nos aseguramos de que el servicio `networking` será detenido en primer lugar, y tan sólo nos queda asegurarnos de que el script para udhpcd (que ya debería estar presente en el directorio) está precedido de S y un número mayor que 01. En último lugar, volvemos a arrancar el servicio `networking` actualizando el fichero `/etc/rc.local` añadiendo la línea `service networking start`.

Habremos de ocuparnos también de que el servidor arranque al hacerlo la Raspberry. Si bien existen diferentes formas de automatizar este proceso en Linux, en este caso se ha optado por la opción de utilizar `upstart`, un reemplazo para el sistema tradicional de iniciar procesos en Linux basado en eventos, que de hecho es de uso extendido en gran parte de las distribuciones Linux basadas en Debian. Fue escrito y desarrollado por Scott James Remnant.

El proceso tradicional de arranque en Linux era responsable de asegurar el estado normal de la máquina tras el encendido, o de terminar correctamente los procesos antes de su apagado. Como resultado, el diseño era estrictamente síncrono, bloqueando las tareas futuras hasta que termine la que se esté procesando en ese momento. Además, las tareas debían definirse *a priori*, lo que limitaba ciertas tareas no relacionadas con el arranque que son indispensables en un ordenador moderno, como la adición de nuevos discos o dispositivos USB, o incluso el firmware de un dispositivo. El modelo de eventos de `upstart` le permite responder a los eventos de manera asíncrona conforme se generan, manejando el arranque de las tareas y servicios durante el arranque y deteniéndolas en el apagado, pero también supervisándolos mientras el sistema está corriendo.

Para instalar `upstart` en nuestra máquina, habremos de ejecutar:

```
sudo apt-get install upstart
```

Una vez instalado, habremos de escribir la configuración de arranque de nuestro servicio. Para ello, creamos el fichero `/etc/init/gpsapi.conf`:

```
start on filesystem and started networking
stop on [06]

setuid pi
setgid pi

respawn

script
  chdir /home/pi/dev/pfc/node
  exec /home/pi/.nvm/v0.10.26/bin/node server.js
end script
```

Código 4.11 /etc/init/gpsapi.conf.

En este fichero estamos indicando a `upstart` bajo qué condiciones debe arrancar y bajo cuáles no, con las dos primeras líneas (en la primera indicamos que el servicio debe arrancar una vez se haya montado el sistema de ficheros y el servicio de red, y en la segunda le indicamos que no debe arrancar para los *runlevels* 0 y 6, *halt* y *reboot*, respectivamente, pues no tiene sentido). A continuación indicamos el usuario y el grupo con el que ejecutar el script (en este caso, se trata del usuario por defecto de Raspbian, Pi), y finalmente le indicamos qué debe ejecutar.

4.3 Servidor web

Para simular la lógica de los drones, se ha decidido desarrollar un pequeño servidor web que correrá en la Raspberry Pi y ofrecerá una sencilla API REST mediante la cual la aplicación de Smartphone podrá solicitar las coordenadas GPS del puesto de auxilio más cercano, así como enviar las suyas propias para solitar ayuda. El *runtime* elegido es **node.js**, construido sobre el motor V8 de Chrome. Utiliza un modelo de entrada/salida no bloqueante, basado en eventos, ligero y eficiente. El *framework* utilizado es **Express**. Se trata de una infraestructura de aplicaciones web node.js mínima y flexible que proporciona un conjunto

sólido de características para las aplicaciones web. Con miles de métodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla, proporcionando además una delgada capa de características de aplicación web básicas, permitiendo un gran rendimiento.

Con el objetivo de escribir los tests, se utilizarán dos de las herramientas más extendidas en Javascript para tal fin: **Mocha**, como *framework* de testing, y **chai**, una librería para comprobaciones (*assertions*).

4.3.1 Desarrollo

Especificaciones

Comencemos, en primer lugar, como se explicó en los fundamentos teóricos, a escribir las especificaciones de nuestro proyecto a alto nivel, sin tener en cuenta de los detalles de la futura implementación. En primer lugar, teniendo en cuenta los objetivos de diseño, parece tener sentido utilizar dos *endpoints* para nuestra API: uno para solicitar las coordenadas GPS del punto de ayuda más cercano (`/help`) y otro para solicitar un rescate (`/rescue`).

El primero de ellos, `/help` recibirá como parámetros las coordenadas GPS del cliente, y en función de las mismas, devolverá las coordenadas GPS del punto de ayuda más cercano, tras realizar los cálculos pertinentes. En caso de que no se manden las coordenadas GPS, o de que estas no sean válidas, el servidor deberá devolver un error.

El otro *endpoint*, `/rescue`, también recibirá como parámetros las coordenadas GPS del cliente. De nuevo, si estas no existen o son erróneas, deberá devolver un código de estado de error, y en caso contrario, deberá devolver un código de estado "OK" así como un pequeño mensaje para el usuario indicando que la ayuda está en camino. En caso de que dicho usuario ya haya realizado una petición (donde se identificará al usuario por su dirección IP), devolverá un mensaje de OK añadiendo también la fecha y hora en las que realizó la petición, así como el tiempo que ha transcurrido desde entonces.

Preparación

Una vez definidas las especificaciones, antes de comenzar con la implementación, utilizaremos el gestor de paquetes de node.js **npm** para inicializar el proyecto.

```
npm init
```

De esta manera podremos crear, con una sencilla interfaz, el contenido básico del fichero `package.json`. Este fichero, que debe seguir el formato JSON, contiene información sobre el proyecto (tales como el nombre y la versión, una descripción, un enlace al sistema de seguimiento de error o *bug-tracker* del proyecto o una dirección de correo a la que reportar los errores, información sobre la licencia con la que se publica el módulo, un campo para indicar el punto de entrada del programa, una referencia al repositorio donde se aloja el código, scripts, etc), así como la lista de dependencias del mismo (nombre y versión de las mismas).

El siguiente paso es instalar las dependencias necesarias. Npm distingue entre dos tipos de dependecias: *dependencies*, las cuales son indispensables para poder ejecutar el programa, y *devDependencies* (dependencias de desarrollo), las cuales sólo son necesarias durante la fase de desarrollo del proyecto. Por lo tanto, los paquetes `mocha` y `chai` serán dependencias de desarollo, mientras que `express` y `underscore` serán dependencias necesarias para que el programa funcione correctamente.

```
npm install --save-dev mocha
npm install --save-dev chai
npm install --save express
npm install --save underscore
```

Los parámetros `-save` y `-save-dev` sirven para que npm incluya de forma automática las dependencias al fichero `package.json`, el cual ahora deberá tener un contenido como el que sigue (tras añadir un script para ejecutar los tests utilizando mocha):

```
{
  "name": "gps-api",
```

```
  "version": "1.0.0",
  "description": "simple api to return fake gps coordinates",
  "main": "index.js",
  "scripts": {
    "test": "./node_modules/.bin/mocha test/*.spec.js"
  },
  "author": "Roberto Espejo",
  "license": "",
  "devDependencies": {
    "chai": "^3.5.0",
    "mocha": "^3.2.0",
    "request": "^2.79.0"
  },
  "dependencies": {
    "express": "^4.14.0",
    "body-parser": "^1.15.2",
    "underscore": "^1.8.3"
  }
}
```

Código 4.12 package.json.

La estructura del directorio del proyecto será la siguiente:

```
/
├── scripts/
├── node_modules/
├── test/
│   └── server.spec.js
├── package.json
└── server.js
```

Creamos por tanto los ficheros necesarios y nos disponemos a comenzar con la implementación.

Implementación

Ahora que está todo preparado, podemos comenzar con la implementación del código, como se ha comentado anteriormente, siguiendo la metodología TDD:

- En primer lugar, elegimos una especificación. Por ejemplo, tomemos el primero de los endpoints, `/help`. Se especifica que, ante una llamada sin parámetros, el servidor deberá devolver un código de error.
- A continuación, debemos escribir una prueba (test). Utilizaremos los bloques `describe` para dar contexto a los tests, y la directiva `it` para describir cada uno de los propios casos de prueba (*test cases*). En este caso se pretende comprobar que, ante una llamada sin parámetros, el servidor responde con un código de error http 400 (*Bad request*, Petición errónea). Haremos uso por tanto de la librería *request* para realizar esta llamada, y, mediante las comprobaciones de *chai*, nos aseguraremos de que la respuesta del servidor es la esperada.

```
describe('GPS API', function() {

  describe('Help', function() {

    var url = 'http://localhost:3000/help';

    it('should return an error if no parameters provided', function(done) {
    request(url, function(error, response, body) {
          expect(response.statusCode).to.equal(400);
          expect(body).to.equal(JSON.stringify({
          msg: 'GPS coordinates need to be provided. Please try again.'
            }));
          done();
        });
  });
});
```

```
});
```

- Verificamos ahora que la prueba falla. Como nuestro código aún no está implementado, es imposible que el test pase. Ejecutamos los tests utilizando el comando que provee npm (y que configuramos en el fichero `package.json`):

```
npm test
```

Obtendremos la siguiente salida en la consola:

```
> gps-api@1.0.0 test /home/rob/pfc/node
> mocha test/*.spec.js

  GPS API
    Help
      1) should return an error if no parameters provided

  0 passing (60ms)
  1 failing

  1) GPS API Help should return an error if no parameters provided:
     Uncaught TypeError: Cannot read property 'statusCode' of undefined
      at Request._callback (test/server.spec.js:13:25)
      at self.callback (node_modules/request/request.js:186:22)
      at Request.onRequestError (node_modules/request/request.js:845:8)
      at Socket.socketErrorListener (_http_client.js:310:9)
      at emitErrorNT (net.js:1276:8)
      at _combinedTickCallback (internal/process/next_tick.js:74:11)
      at process._tickCallback (internal/process/next_tick.js:98:9)

npm ERR! Test failed. See above for more details.
```

Esto se debe a que aún no tenemos implementado el servidor, y por lo tanto, el test no es capaz de recibir ninguna respuesta.

- Escribir la implementación: Escribir el código más sencillo que haga que la prueba funcione. En este caso, editamos el fichero `server.js` e incluimos la lógica.

```
var express = require('express');
var app = express();

app.get('/help', function(req, res) {

  var keys = Object.keys(req.query);

  if (!_validateArguments(keys)) {
    res.status(400).send({
      msg: 'GPS coordinates need to be provided. Please try again.'
    });
  }

});

app.listen(3000);

_validateArguments = function (Arguments) {
  return Arguments.length !== 0;
};
```

Se ha definido un método `_validateArguments`. En Javascript no existen métodos o atributos privados como en otros lenguajes de programación orientados a objetos, por lo que, por convención de código, los nombres de los métodos o atributos que deberían ser privados y no utilizados fuera del objeto donde están definidos se preceden con una barra baja (_) para así indicarlo. Si ahora ejecutamos el servidor:

```
node server.js
```

podemos comprobar que el test pasará y por tanto ya tenemos nuestro código implementado. Bastaría con repetir este proceso hasta conseguir implementar todas las especificaciones. El hecho de tener tests también nos ayuda a la hora de mantener el código: si durante el proceso de implementación del resto de las mismas los tests anteriores dejan de pasar, sabremos que algo no está comportándose como es debido.

El resto del código sigue este mismo patrón de desarrollo, y se adjunta en su totalidad en los apéndices. Con el fin de agilizar el comentario del mismo, se ofrecerá a continuación una descripción más general del resto del código, sin entrar en tanto detalle sobre cómo ha afectado TDD al desarrollo del mismo.

GpsCalculator.js Antes de hablar del resto del servidor, haremos mención al módulo GpsCalculator. Un módulo en Javascript no es más que una pequeña unidad de código reutilizable, que suele utilizarse para agrupar métodos con un fin común. En este caso, el propósito de nuestro módulo GpsCalculator es agrupar toda la lógica relacionada con el cálculo y validación de las coordenadas GPS. El módulo exporta (mediante la directiva `exports`) dos métodos: *getNearestPoint* y *validateCoordinates*, y posee otros dos métodos privados que no exporta (como indica su nombre prefijado de _). Para el cálculo de la distancia del punto de ayuda más cercano, la función `getNearestPoint` acepta como parámetros las coordenadas originales y una lista de los puntos de ayuda, sobre los cuales itera llamando al método privado `getNearestPoint`, quien aplica la llamada **fórmula de Haversine** (que ya se explicó en los fundamentos teóricos) para calcular la distancia entre dos puntos definidos por coordenadas de latitud y longitud. Finalmente, getNearestPoint devuelve aquel lugar cuya distancia sea menor respecto a las demás al punto dado.

server.js El código del servidor, **server**, hace uso del módulo GpsCalculator mediante la directiva `require`. Esto hace que en el contexto de server.js esté disponible un objeto GpsCalculator que provee los métodos que nuestro módulo exporta, como se ha comentado anteriormente, lo cual nos permite utilizarlo en las rutas `/help` y `/rescue` para validar las coordenadas GPS. El diseño de estos dos *endpoints* ya ha sido explicado cuando se dieron sus especificaciones en cuanto a qué devolver, pero merece la pena destacar el hecho de que `/help` se encarga también de guardar en un vector de objetos las peticiones recibidas cuyo resultado ha sido 200, con el fin de saber si se ha procesado ya una petición desde dicha dirección IP. De este modo, la próxima vez que se reciba una petición, se podrá comprobar si existe ya en dicho vector para cambiar el mensaje de respuesta que el servidor ofrecerá al cliente. Otro aspecto a destacar en el desarrollo del servidor es la implementación de otra ruta de la que no se había hablado hasta ahora: `/requests`. Este *endpoint* se ha implementado con dos fines: para monitorizar el estado de las peticiones de rescate recibidas hasta el momento, con propósitos de depuramiento, así como para que cualquier equipo de rescate pueda acceder a esta información y dirigirse a ayudar al usuario. De nuevo con propósitos de depuramiento se ha implementado una funcionalidad extra, y es que esta ruta acepta un parámetro *flush* para vaciar la lista de peticiones de ayuda recibidas.

4.4 Puesta en marcha y pruebas

Tras la configuración de las Raspberries, tan sólo será necesario encenderlas para poner todo en marcha, pues los scripts de arranque configurados se encargarán de iniciar todos los procesos necesarios en la máquina una vez esta esté en marcha. En cuanto a la aplicación Android, será suficiente con instalar el archivo `.apk` generado desde el IDE para tener la aplicación disponible y funcionando.

Con respecto a la distribución de las Raspsberries, se ha situado una a 2 metros del dispositivo móvil (Raspberry B, cuyo SSID es Rob-RPB) y otra a una distancia de 10 metros (Raspberry A, cuyo SSID es Rob-RPA).

Se realizarán a continuación una serie de pruebas en las que se pretende comprobar el correcto funcionamiento tanto de la aplicación Android como del servidor web:

- Prueba 01: Se arrancará la aplicación con el WiFi desconectado y el servicio de localización desactivado.

- Prueba 02: Se arrancará la aplicación con el WiFi desconectado y el servicio de localización activado, y se realizará una petición de rescate.
- Prueba 03: Se arrancará la aplicación con el WiFi conectado y el servicio de localización activado, y se realizará una petición para obtener el punto de ayuda más cercano.
- Prueba 04: Se desconectará el WiFi tras realizar al menos una petición.

4.4.1 Prueba 01

Para esta prueba se iniciará la aplicación con el servicio WiFi desconectado y el servicio de localización también desactivado. El usuario será bienvenido con el botón de "Start Scanning" y, tras hacer click en él, será informado de que el servicio WiFi está desactivado y está en proceso de ser activado mediante un *toast*.

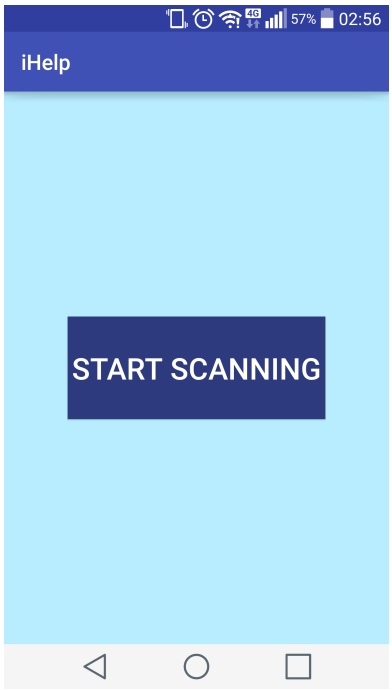


Figura 4.5 Prueba 01. Botón para empezar el escaneo de redes inalámbricas.

En este momento, el usuario recibirá un mensaje diciendo que el servicio de localización debe ser activado para poder utilizar la aplicación. Al hacer click en *Location settings* el usuario será redirigido a los ajustes de localización del teléfono, donde podrá activar dicho servicio.

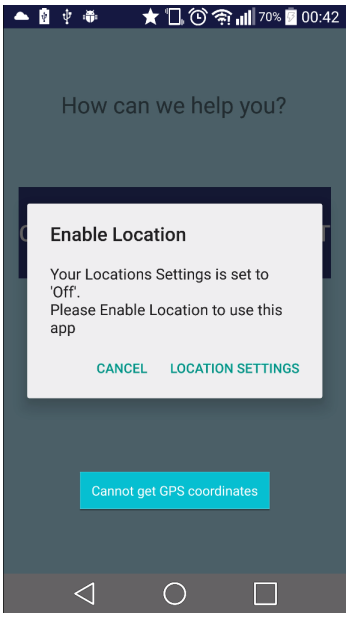


Figura 4.6 Prueba 01. Alerta para activar el servicio de localización.

Inmediatamente al regresar a la aplicación, la interfaz de usuario estará deshabilitada y un mensaje nos informará de que no tenemos coordenadas GPS, pues el dispositivo aún no ha sido capaz de obtener la posición del satélite.

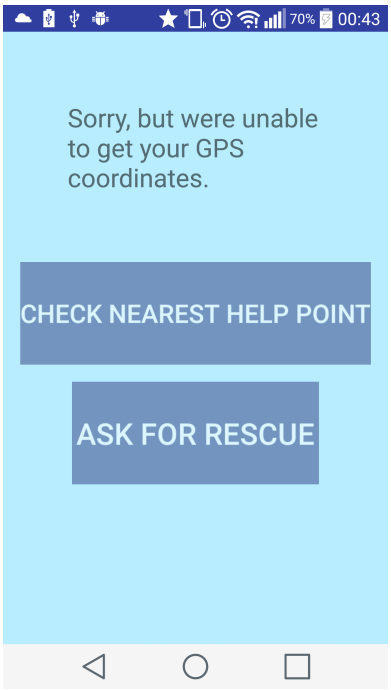


Figura 4.7 Prueba 01. Interfaz de usuario dehabilitada.

4.4.2 Prueba 02

Para esta prueba se iniciará la aplicación con el servicio WiFi desconectado y el servicio de localización activado. El usuario será de nuevo bienvenido con el botón de "Start Scanning" y, tras hacer click en él, como en el caso anterior, será informado de que el servicio WiFi está desactivado y está en proceso de ser activado mediante el mismo *toast* de la prueba anterior.

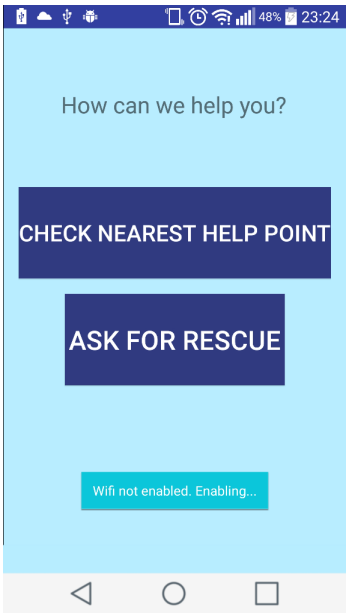


Figura 4.8 Prueba 02. *Toast* activando el Wifi.

Acto seguido, tras conectar a la red que se recibe con mayor intensidad (la que proviene de la Raspberry B), podemos observar la primera diferencia con el caso anterior. Los botones en este caso aparecen activados. Pasamos entonces a hacer click en el botón *Ask for rescue*, y la aplicación nos mostrará el mensaje del servidor informándonos de que la ayuda está en camino.

Figura 4.9 Prueba 02. Mensaje del servidor tras pedir un rescate.

Las coordenadas GPS han llegado al servidor y éste nos ha devuelto el mensaje de ayuda. Podemos comprobar en el servidor que el fichero log contiene una entrada correspondiente a nuestra llamada:

```
[RESCUE] Received request on /RESCUE from 192.168.42.14, params: 51.55721578 −0.98982799
[RESCUE] Returning 200: { msg: Worry not, help is on the way.}
```

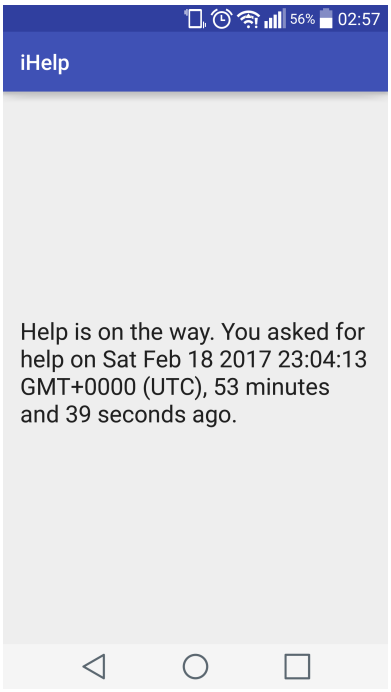


Figura 4.10 Prueba 02. Mensaje del servidor tras pedir un rescate por segunda vez.

Si volvemos ahora a la pantalla anterior, haciendo uso de la navegación de Android, y volvemos a utilizar el botón, obtendremos esta vez una respuesta diferente, y es que el servidor habrá registrado la llamada anterior, de forma que ahora indicará cuándo se realizó la llamada anterior y el tiempo que ha transcurrido desde entonces. De nuevo podemos comprobar en el log del servidor:

```
[RESCUE] Received request on /RESCUE from 192.168.42.14, params: 51.55721578 −0.98982799
[RESCUE] Returning 200 to an already existing request
```

4.4.3 Prueba 03

Para esta prueba se iniciará la aplicación con el servicio WiFi conectado y el servicio de localización activado. Al igual que en los casos anteriores, el usuario será de nuevo bienvenido con el botón de "Start Scanning". Tras hacer click en él, será informado de la conexión a la red inalámbrica (de nuevo RPB), y al contrario que en los casos anteriores, no verá el *toast* informado de que el WiFi está desactivado. Una vez de nuevo en el menú principal, haremos click esta vez en el botón de obtener el punto de ayuda más cercano.

La aplicación nos mostrará la respuesta del servidor, indicando un nombre identificativo del punto de ayuda más cercano junto a sus coordenadas GPS, así como una indicación de la distancia a la que se encuentra y en qué dirección.

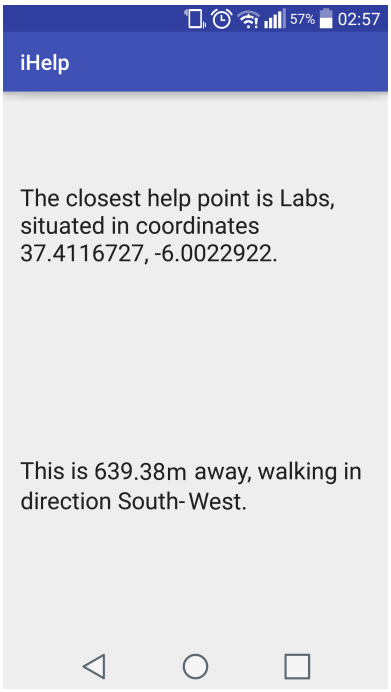


Figura 4.11 Prueba 03. Mensaje del servidor tras solicitar el punto de ayuda más cercano a nuestra posición.

4.4.4 Prueba 04

Para esta última prueba se iniciará de nuevo la aplicación con el servicio WiFi conectado y el servicio de localización activado. Tras escanear redes como en las pruebas anteriores, se procederá a desconectar el WiFi conel fin de simular un error en la red, o bien una desconexión real por parte del usuario. A continuación, al tratar de hacer click en alguno de los dos botones del menú principal, se puede observar cómo la aplicación se reinicia mostrando un *toast* al usuario informándole de la pérdida de conexión.

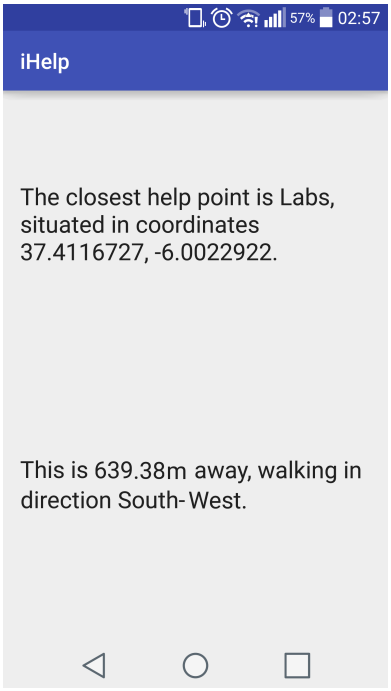


Figura 4.12 Prueba 04. *Toast* informando al usuario de la pérdida de conexión.

5 Conclusiones

5.1 Conclusiones finales

Durante el proyecto se ha podido estudiar la versatilidad que provee una Raspberry Pi como sistema empotrado, permitiéndonos crear una red inalámbrica a la que usuarios móviles pueden conectarse y además utilizar un servicio web de forma sencilla. Se ha estudiado en profundidad la configuración de red de un sistema Linux, así como el proceso de arranque de servicios durante el encendido de la máquina, con el objetivo de automatizar el proceso y que la Raspberry se haya convertido en un servidor *plug and play*, teniendo tan sólo que iniciar el sistema operativo para tener todo funcionando.

Durante el desarrollo de la aplicación Android se ha trabajado con el sistema tanto a nivel visual, editando los ficheros XML que conforman la interfaz de usuario, como a nivel de programación y como a nivel de modificación de parámetros del dispositivo, tales como activar y desactivar la conexión WiFi. Se ha podido estudiar tan sólo una parte de la inmensa API que provee Android a los desarrolladores, así como distintos enfoques con respecto la concurrencia (como tareas asíncronas e hilos).

Se ha podido comprobar también cómo Nodejs y express proveen potentes herramientas para crear APIs REST de manera sencilla, y cómo el proceso de TDD ayuda al desarrollador tanto durante la fase de desarrollo como durante la fase de mantenimiento del software.

En resumen, se han cumplido los objetivos establecidos en la introducción a este mismo texto, permitiendo finalmente que un usuario en una situación de emergencia, sin acceso a redes móviles, pueda solicitar ayuda a través de unos cuantos toques en su dispositivo Android.

5.2 Futuras líneas de trabajo

Si bien el alcance de este proyecto ha intentado cubrir tan sólo la comunicación nodo-usuario de la situación de desastre en la que nos enmarcamos al comenzar, existen aún muchas mejoras por implementar en el sistema, como las que se listan a continuación:

- Por un lado, la ampliación del alcance para trabajar en la comunicación nodo a nodo. Esto implicaría la implementación de la red mesh que comunique a todos los drones del enjambre utilizando una segunda interfaz inalámbrica, así como la configuración de la misma, al igual que en este proyecto se ha realizado con la red nodo-usuario.
- Dentro de la red mesh, sería necesaria una base de datos para sincronizar la información entre los distintos nodos de la red. De este modo, existirá persistencia de las llamadas recibidas (pues ahora mismo los servidores tan sólo guardan las peticiones en memoria), de forma que se eviten peticiones duplicadas si un usuario cambia por algún motivo de nodo. Además, la base de datos proveería a terceras entidades la posibilidad de actualizar en tiempo real los lugares de ayuda, bien sea incluyendo nuevos, o cambiando la información sobre los ya existentes.
- Otra posible mejora, una vez implementada la red mesh, sería el proveer de acceso a internet al usuario (y no sólo a una pequeña red local).

Bibliografía

[1] L. Reynaud, R. Tinku. *Deployable aerial communication networks: challenges for futuristic applications*. PE-WASUN'12, Octubre de 2012.

[2] J. Sánchez-García, J. M. García-Campos, S. L. Toral, D. G. Reina, F. Barrero. *An Intelligent Strategy for Tactical Movements of UAVs in Disaster Scenarios*. Sage Journals, Marzo de 2016.

[3] M. Conti, S. Giordiano. *Multihop Ad Hoc Networking: The Reality*. IEEE Communications Magazine, Abril de 2007.

[4] F. Manjoo. *A Murky Road Ahead for Android, Despite Market Dominance*. The New York Times, Mayo de 2015

[5] J. Henry. *802.11s Mesh networking*. CWNP, Noviembre de 2011.

[6] Institute of Electrical and Electronics Engineers. *Official IEEE 802.11 Working Group Project Timelines*. Noviembre de 2011

[7] K. Beck. *Test Driven Development*. The Addison-Wesley Signature Series. Noviembre de 2002.

[8] P. Lomas. *Case study: Challenges in incarnating a credit card sized SBC*. Embedded Computing, Junio de 2013. http://embedded-computing.com/articles/case-card-sized-sbc/

[9] F. Manjoo. *A Murky Road Ahead for Android, Despite Market Dominance*. The New York Times, Mayo de 2015. https://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html

[10] statcounter.com. *Top Mobile Operating Systems Per Country*. Enero de 2016. http://gs.statcounter.com/os-market-share

Apéndice A
Código del servidor en node.js

A.1 Archivos del proyecto

```
{
  "name": "gps-api",
  "version": "1.0.0",
  "description": "simple api to return fake gps coordinates",
  "main": "index.js",
  "scripts": {
    "test": "./node_modules/.bin/mocha test/*/*.spec.js"
  },
  "author": "Roberto Espejo",
  "license": "",
  "devDependencies": {
    "chai": "^3.5.0",
    "mocha": "^3.2.0",
    "request": "^2.79.0"
  },
  "dependencies": {
    "body-parser": "^1.15.2",
    "express": "^4.14.0",
    "underscore": "^1.8.3"
  }
}
```

Código A.1 package.json.

```
var express = require('express');
var app = express();
var _ = require('underscore');

var gpsCalculator = require('./scripts/GpsCalculator');

var _acceptedRequests = [];

var _knownPlaces = [
  {
    lat: 37.4116727,
    long: -6.0022922,
    name: 'Labs'
  },
  {
    lat: 37.4086703,
    long: -5.9971619,
    name: 'Parking Isla Magica'
  },
  {
    lat: 37.4043522,
    long: -5.9888324,
```

```
    name: 'Parliament'
  }
]

app.get('/help', function(req, res) {

  _logRequest('HELP', req);

  var keys = Object.keys(req.query);

  if (!_validateArguments(keys)) {
    console.log(new Date() + ' [HELP] Returning 400: { msg: GPS coordinates need to be provided.
        Please try again.}');
    res.status(400).send({
      msg: 'GPS coordinates need to be provided. Please try again.'
    });
  } else if (!gpsCalculator.validateCoordinates(parseFloat(req.query.lat), parseFloat(req.query.long))) {
    console.log(new Date() + ' [HELP] Returning 422: { msg: Sorry, but we were not able to get your
        position. Please try again.}');
    res.status(422).send({
      msg: 'Sorry, but we were not able to get your position. Please try again.'
    });
  } else {
    console.log(new Date() + ' [HELP] Returning 200');
    var closestPlace = gpsCalculator.getNearestPoint(_knownPlaces, parseFloat(req.query.lat), parseFloat(req.query.
        long));

    res.status(200).send( closestPlace );
  }

});

app.get('/rescue', function(req, res) {

  _logRequest('RESCUE', req);

  var keys = Object.keys(req.query);

  if (!_validateArguments(keys)) {
    console.log(new Date() + ' [RESCUE] Returning 400: { msg: GPS coordinates need to be provided.
        Please try again.}');
    res.status(400).send({
      msg: 'GPS coordinates need to be provided. Please try again.'
    });
  } else if (!gpsCalculator.validateCoordinates(parseFloat(req.query.lat), parseFloat(req.query.long))) {
    console.log(new Date() + ' [RESCUE] Returning 400: { msg: Sorry, but we were not able to get your
        position. Please try again.}');
    res.status(422).send({
      msg: 'Sorry, but we were not able to get your position. Please try again.'
    });

  } else if (_alreadySentHelp(req, _acceptedRequests)) {
    console.log(new Date() + ' [HELP] Returning 200 to an already existing rescue request}');

    var originalDate = _.findWhere(_acceptedRequests, {ip: req.ip}).date;
    var timeDiff = new Date() - originalDate;

    timeDiff /= 1000;

    var seconds = Math.round(timeDiff % 60);
    timeDiff = Math.floor(timeDiff / 60);
    var minutes = Math.round(timeDiff % 60);

    res.status(200).send({
      msg: 'Help is on the way. You asked for help on ' + originalDate + ', ' + minutes + ' minutes and ' +
          seconds + ' seconds ago.'
    });

  } else {
    console.log(new Date() + ' [HELP] Returning 200: { msg: Worry not, help is on the way.}');
```

```
      _acceptedRequests.push( {
        ip: req.ip,
        date: new Date()
      });

      res. status (200).send({
        msg: 'Worry not, help is on the way.'
      });
    }

});

app.get('/requests', function(req, res) {

  var keys = Object.keys(req.query);

  _logRequest('REQUESTS', req);

  if (keys.length === 1 && keys.indexOf('flush') !== 1) {
    _acceptedRequests = [];
    console.log(new Date + ' [REQUESTS] Flushing already received requests!');
  }

  res. status (200).send(_acceptedRequests);

});

app. listen (3000);

_validateArguments = function (Arguments) {
  return Arguments.length !== 0 && (Arguments.indexOf('lat') !== -1 || Arguments.indexOf('long') !== -1);
};

_logRequest = function (endpoint, req) {
  console.log(new Date() + ' ' + '[' + endpoint + '] Received request on /' + endpoint + ' from ' + req.ip + ',
      params: ' + req.query.lat + ' ' + req.query.long);
};

_alreadySentHelp = function (req, acceptedRequests) {
  if (_.findWhere(acceptedRequests, {ip: req.ip})) {
    return true;
  } else {
    return false;
  }
}
```

Código A.2 server.js.

A.2 Tests

```
var request = require('request');
var chai = require('chai');
var expect = chai.expect;

describe('GPS API', function () {

  describe('Help', function () {

    var url = 'http://localhost:3000/help';

    describe('Error', function () {

      it('should return an error if no parameters provided', function (done) {
        request(url, function(error, response, body) {
              expect(response.statusCode).to.equal(400);
              expect(body).to.equal(JSON.stringify({
              msg: 'GPS coordinates need to be provided. Please try again.'
                 }));
              done();
            });
      });

      it('should return an error if the parameters are not GPS coordinates', function(done) {

        request(url + '?abc=123&123=abc', function(error, response, body) {
              expect(response.statusCode).to.equal(400);
              expect(body).to.equal(JSON.stringify({
              msg: 'GPS coordinates need to be provided. Please try again.'
                 }));
              done();
            });

      });

      it('should return an error if the GPS coordinates are wrong', function(done) {

        request(url + '?lat=123&long=abc', function(error, response, body) {
              expect(response.statusCode).to.equal(422);
              expect(body).to.equal(JSON.stringify({
              msg: 'Sorry, but we were not able to get your position. Please try again.'
                 }));
              done();
            });

      });
    });

    describe('Success', function () {

      it('should return status 200 if the GPS coordinates are ok', function(done) {

        request(url + '?lat=45&long=73', function(error, response, body) {
              expect(response.statusCode).to.equal(200);
              done();
            });

      });

      it('should return the closest help point if the GPS coordinates are ok', function(done) {

        request(url + '?lat=37.392894&long=-5.994779', function(error, response, body) {
              expect(response.statusCode).to.equal(200);
              expect(body).to.equal(JSON.stringify({
             lat: 37.4043522,
            long: -5.9888324,
            name: 'Parliament',
             distance: 1378.135254316005
```

```
            }));
                done();
              });

      });

    });

});

describe('Rescue', function () {

  var url = 'http://localhost:3000/rescue';

  describe('Error', function () {

    it('should return an error if no parameters provided', function (done) {
      request(url, function(error, response, body) {
            expect(response.statusCode).to.equal(400);
            expect(body).to.equal(JSON.stringify({
            msg: 'GPS coordinates need to be provided. Please try again.'
              }));
            done();
          });
    });

    it('should return an error if the parameters are not GPS coordinates', function(done) {
      request(url + '?abc=123&123=abc', function(error, response, body) {
            expect(response.statusCode).to.equal(400);
            expect(body).to.equal(JSON.stringify({
            msg: 'GPS coordinates need to be provided. Please try again.'
              }));
            done();
          });
    });

    it('should return an error if the GPS coordinates are wrong', function(done) {
      request(url + '?lat=123&long=abc', function(error, response, body) {
            expect(response.statusCode).to.equal(422);
            expect(body).to.equal(JSON.stringify({
            msg: 'Sorry, but we were not able to get your position. Please try again.'
              }));
            done();
          });

    });

  });

  describe('Success', function () {

    it('should return status 200 if the GPS coordinates are ok', function(done) {
      request(url + '?lat=45&long=73', function(error, response, body) {
            expect(response.statusCode).to.equal(200);
            expect(body).to.equal(JSON.stringify({
            msg: 'Worry not, help is on the way.'
              }));
            done();
          });
    });

    it('should return status 200 if a second request arrives', function(done) {
      request(url + '?lat=45&long=73', function(error, response, body) {
            expect(response.statusCode).to.equal(200);
            expect(body).not.to.equal(JSON.stringify({
            msg: 'Worry not, help is on the way.'
              }));
            done();
          });
    });
```

```
    });

  });

  describe('Requests', function () {
    var url = 'http://localhost:3000/requests';

    it('should return the received requests', function(done) {
      request(url, function(error, response, body) {
        expect(response.statusCode).to.be.equal(200);
        done();
      })
    });

    it('should flush the request if asked to', function(done) {
      request(url + '?flush', function(error, response, body) {
        expect(response.statusCode).to.be.equal(200);
        done();
      })
    });
  });

});
```

Código A.3 test/server.spec.js.

```
var gpsCalculator = require('../../scripts/GpsCalculator');

var request = require('request');
var chai = require('chai');
var expect = chai.expect;

describe('gpsCalculator', function () {

  describe('Validate coordinates', function () {

    describe('error', function () {
      it('should return false if gps coordinates are not present', function () {
        expect(gpsCalculator.validateCoordinates()).to.be.false;
      });

      it('should return false if gps coordinates are not numbers', function () {
        expect(gpsCalculator.validateCoordinates(20,'cbahhadhadhada')).to.be.false;
      });

      it('should return false if gps coordinates are not valid (positive numbers)', function () {
        expect(gpsCalculator.validateCoordinates(95, 190)).to.be.false;
      });

      it('should return false if gps coordinates are not valid (negative numbers)', function () {
        expect(gpsCalculator.validateCoordinates(-95, -190)).to.be.false;
      });

    });

    describe('success', function () {
      it('should return true if gps coordinates are valid (positive numbers)', function () {
        expect(gpsCalculator.validateCoordinates(85, 170)).to.be.true;
      });

      it('should return true if gps coordinates are valid (negative numbers)', function () {
        expect(gpsCalculator.validateCoordinates(-85, -170)).to.be.true;
      });
    });

  });

  describe('Get nearest point', function () {

    var knownPlaces = [
        {
          lat: 37.4116727,
          long: -6.0022922
        },
        {
          lat: 37.4086703,
          long: -5.9971619
        }
    ];

    describe('error', function () {
      it('should return an error if no arguments are provided', function () {
        expect(gpsCalculator.getNearestPoint()).to.be.equal(-1);
      });

      it('should return an error if no knownPlaces are provided', function () {
        expect(gpsCalculator.getNearestPoint(null,84,127)).to.be.equal(-1);
      })

      it('should return an error if no coordinates are provided', function () {
        expect(gpsCalculator.getNearestPoint(knownPlaces)).to.be.equal(-1);
      })
    });

    describe('success', function () {
      it('should calculate the nearest point properly', function() {
        expect(gpsCalculator.getNearestPoint(knownPlaces,37.4043522,-5.9888324)).to.deep.equal({
```

```
            lat: 37.4086703,
            long: -5.9971619,
            distance: 878.5390121762819
          });
        })
      });
    });

});
```

Código A.4 scripts/GpsCalculator.spec.js.

A.3 Scripts

```
var _ = require('underscore');

exports.getNearestPoint = function(knownPlaces, lat, long){
  if(!knownPlaces || !lat || !long) {
    return -1;
  }

  var locationsWithDistance =_.map(knownPlaces, function(location) {
    location.distance = _getDistanceBetweenTwoPoints(location.lat, location.long, lat, long);
    return location;
  });

  var result = _.where(locationsWithDistance, {distance: _.min(_.map(locationsWithDistance, function(location) {
      return location.distance; }))});

  return result[0];
};

exports.validateCoordinates = function(lat, long) {

    if (!lat || !long) {
      return false;
    } else if (isNaN(lat) || isNaN(long)) {
      return false;
    } else {
      return ((Math.abs(lat) < 90) && (Math.abs(long) < 180));
    }
};

_getDistanceBetweenTwoPoints = function (lat1,long1,lat2,long2) {
  var R = 6371e3;
  var phi1 = _toRadians(lat1);
  var phi2 = _toRadians(lat2);
  var inc_phi = _toRadians(lat2-lat1);
  var inc_lambda = _toRadians(long2-long1);

  var a = Math.sin(inc_phi/2) * Math.sin(inc_phi/2) +
          Math.cos(phi1) * Math.cos(phi2) *
          Math.sin(inc_lambda/2) * Math.sin(inc_lambda/2);
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

  return R * c;
}

_toRadians = function (degrees) {
  var pi = Math.PI;
  return degrees * pi / 180;
}
```

Código A.5 scripts/GpsCalculator.js.

Apéndice B
Código de la aplicación Android

B.1 *manifests*

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

    <uses-feature android:name="android.hardware.location.gps" />

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name="com.pfc.app.iHelp.activities.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.pfc.app.iHelp.activities.MainMenuActivity" />
        <activity android:name="com.pfc.app.iHelp.activities.RescueMessageActivity"></activity>
    </application>

</manifest>
```

Código B.1 manifest.xml.

B.2 *java*

B.2.1 *activities*

```
package com.pfc.app.iHelp. activities ;

import android.content. Intent ;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;

import com.pfc.app.iHelp.R;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        setContentView(R.layout. activity_main );
    }

    public void startScanning (View view) {
        Intent intent = new Intent( this , MainMenuActivity.class);
         startActivity ( intent );
    }

}
```

Código B.2 MainActivity.java.

```
package com.pfc.app.iHelp. activities ;

import android.Manifest;
import android. annotation .TargetApi;
import android.app. Activity ;
import android.app.AlertDialog;
import android.content.Context;

import android.content. DialogInterface ;
import android.content. Intent ;
import android.content.pm.PackageManager;
import android. location .Location;
import android. location .LocationManager;
import android.net. wifi .WifiManager;
import android.os.Build;
import android.os.Bundle;
import android. provider . Settings ;
import android.support.v4.content.ContextCompat;
import android. util .Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.pfc.app.iHelp. classes .GPSHandler;
import com.pfc.app.iHelp.R;
import com.pfc.app.iHelp. classes . RestClientTask ;
import com.pfc.app.iHelp. classes .ToastHandler;
import com.pfc.app.iHelp. classes .WifiHandler;

import org.json.JSONException;
import org.json.JSONObject;

import java.net.MalformedURLException;
import java.net.URL;
import java. util . concurrent .ExecutionException;
import java. util . concurrent .ExecutorService;
import java. util . concurrent .Executors;
```

```
public class MainMenuActivity extends Activity {

    private static final String TAG = "MainMenuActivity";

    private Context context;

    private ToastHandler toastHandler;

    private WifiHandler wifiHandler;
    private GPSHandler gpsHandler;

    private static final String[] LOCATION_PERMS = {
            Manifest.permission.ACCESS_FINE_LOCATION
    };

    private Location location;

    @TargetApi(Build.VERSION_CODES.M)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_menu);

        context = getApplicationContext();

        toastHandler = new ToastHandler(context);

        wifiHandler = new WifiHandler((WifiManager) getSystemService(Context.WIFI_SERVICE), toastHandler);
        gpsHandler = new GPSHandler((LocationManager) getSystemService(Context.LOCATION_SERVICE), this);

        wifiHandler.scanNetworksAndConnectToBest();

        if (!canAccessLocation()) {
            Log.d(TAG, "GPS Permission: requesting ...");
            requestPermissions(LOCATION_PERMS, 1);
        }

        checkIfGpsIsEnabled();

        ExecutorService fixedThreadPoolExecutor = Executors.newFixedThreadPool(1);
        fixedThreadPoolExecutor.submit(gpsHandler);

        location = gpsHandler.getLocation();

        if(location != null) {
            Log.d(TAG, location.toString());
        } else {
            toastHandler.showUserMessage("Cannot get GPS coordinates");
            disableButtonsAndShowMessage();
        }
    }

    public void setLocation(Location location) {
        this.location = location;
    }

    private boolean canAccessLocation() {
        Log.d(TAG, "GPS Permission: Permission is " + String.valueOf(ContextCompat.checkSelfPermission(this,
                android.Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED));
        return ContextCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_FINE_LOCATION) ==
                PackageManager.PERMISSION_GRANTED;
    }

    public void help(View view) throws ExecutionException, InterruptedException, JSONException {
        RestClientTask restClientTask = new RestClientTask();

        restClientTask.execute(composeURL("help", location));
        JSONObject result = restClientTask.get();
        if(result != null) {
            Log.d(TAG, result.toString());
```

```
            Intent intent = new Intent(this, HelpMessageActivity.class);
            intent.putExtra("locationName", result.get("name").toString());
            intent.putExtra("distance", (Double) result.get("distance"));
            intent.putExtra("lat", (Double) result.get("lat"));
            intent.putExtra("lon", (Double) result.get("long"));
            intent.putExtra("deviceLat", location.getLatitude());
            intent.putExtra("deviceLon", location.getLongitude());
            startActivity(intent);
        } else {
            handleNetworkError("Connection lost. Restarting app...");
        }
    }

    public void rescue(View view) throws ExecutionException, InterruptedException, JSONException {
        RestClientTask restClientTask = new RestClientTask();

        restClientTask.execute(composeURL("rescue", location));
        JSONObject result = restClientTask.get();
        if (result != null) {
            Log.d(TAG, result.toString());
            Intent intent = new Intent(this, RescueMessageActivity.class);
            intent.putExtra("message", result.get("msg").toString());
            startActivity(intent);
        } else {
            handleNetworkError("Connection lost. Restarting app...");
        }
    }


    private URL composeURL(String endpoint, Location location) {
        try {
            return new URL("http://192.168.42.1:3000/" + endpoint + "?lat=" + location.getLatitude() + "&long=" +
                    location.getLongitude());
        } catch (MalformedURLException e) {
            e.printStackTrace();
            return null;
        }
    }

    public void handleNetworkError(String message) {
        toastHandler.showUserMessage(message);
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }

    private void checkIfGpsIsEnabled() {
        LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        if (!locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            Log.d(TAG, "GPS is not enabled");
            showAlert();
        } else {
            Log.d(TAG, "GPS is enabled");
        }
    }

    public void showAlert() {
        Log.d(TAG, "ShowAlert");
        final AlertDialog.Builder dialog = new AlertDialog.Builder(this);
        dialog.setCancelable(true);
        dialog.setTitle("Enable Location")
                .setMessage("Your Locations Settings is set to 'Off'.\nPlease Enable Location to " +
                        "use this app")
                .setPositiveButton("Location Settings", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                        Intent myIntent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                        startActivity(myIntent);
                    }
                })
                .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
```

```
                    @Override
                    public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                    }
                });
        AlertDialog alertDialog = dialog.create();
        Log.d(TAG, "ShowAlert after create");

        alertDialog.show();
    }

    private void disableButtonsAndShowMessage() {
        Button btn1 = (Button) findViewById(R.id.button2);
        btn1.setEnabled(false);
        btn1.setAlpha(.5f);

        Button btn2 = (Button) findViewById(R.id.button3);
        btn2.setEnabled(false);
        btn2.setAlpha(.5f);

        TextView text = (TextView) findViewById(R.id.textView);
        text.setText("Sorry, but were unable to get your GPS coordinates.");
    }

}
```

Código B.3 MainMenuActivity.java.

```
package com.pfc.app.iHelp.activities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

import com.pfc.app.iHelp.R;

import java.text.DecimalFormat;

public class HelpMessageActivity extends AppCompatActivity {

    private TextView textTopView;
    private TextView textBottomView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help_message);

        textTopView = (TextView) findViewById(R.id.textView3);
        textBottomView = (TextView) findViewById(R.id.textView4);

        Bundle bundle = getIntent().getExtras();

        String name = bundle.getString("locationName");
        double lat = bundle.getDouble("lat");
        double lon = bundle.getDouble("lon");
        double distance = bundle.getDouble("distance");

        double deviceLat = bundle.getDouble("deviceLat");
        double deviceLon = bundle.getDouble("deviceLon");

        changeTopText(name, lat, lon);
        changeBottomText(distance, getDirection(deviceLat,deviceLon, lat,lon));
    }

    private String getDirection(Double deviceLat, Double deviceLon, Double lat, Double lon) {
        if (lat > deviceLat) {
            if (lon >= deviceLon + 0.1) {
                return "North-East";
            } else if (deviceLon > lon - 0.1 && deviceLon < lon + 0.1) {
```

```
                return "North";
            } else {
                return "North-West";
            }
        } else {
            if (lon >= deviceLon + 0.1) {
                return "South-East";
            } else if (deviceLon > lon - 0.1 && deviceLon < lon + 0.1) {
                return "South";
            } else {
                return "South-West";
            }
        }
    }

    private void changeTopText(String name, double lat, double lon) {
        textTopView.setText("The closest help point is " + name + ", situated in coordinates " + Double.toString(
            lat) + ", " + Double.toString(lon) + ".");
    }

    private void changeBottomText(double distance, String direction) {
        DecimalFormat twoDForm = new DecimalFormat("#.##");
        Double roundedDistance = Double.valueOf(twoDForm.format(distance));

        textBottomView.setText("This is " + roundedDistance + "m away, walking in direction " + direction + ".");
    }
}
```

Código B.4 HelpMessageActivity.java.

```
package com.pfc.app.iHelp.activities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

import com.pfc.app.iHelp.R;

public class RescueMessageActivity extends AppCompatActivity {

    private TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rescue_message);

        textView = (TextView) findViewById(R.id.textView2);

        String text = getIntent().getStringExtra("message");

        changeText(text);
    }

    public void changeText(String text) {
        textView.setText(text);
    }
}
```

Código B.5 RescueMessageActivity.java.

B.2.2 *classes*

```
package com.pfc.app.iHelp.classes;

import android.location.Location;
```

```
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;

import com.pfc.app.iHelp.activities.MainMenuActivity;

import java.util.concurrent.Callable;

public class GPSHandler implements Callable<Integer> {

    private static final String TAG = "GPSHandler";

    private MainMenuActivity mainMenu;
    private LocationManager locationManager;

    public GPSHandler(LocationManager locationManager, MainMenuActivity mainMenu) {
        this.locationManager = locationManager;
        this.mainMenu = mainMenu;
    }

    public Integer call() {
        try {

            // Define a listener that responds to location updates
            LocationListener locationListener = new LocationListener() {
                public void onLocationChanged(Location location) {
                    Log.d(TAG, location.toString());
                    mainMenu.setLocation(location);
                }

                public void onStatusChanged(String provider, int status, Bundle extras) {
                }

                public void onProviderEnabled(String provider) {
                    Log.d(TAG, "GPS enabled");
                }

                public void onProviderDisabled(String provider) {
                    Log.d(TAG, "GPS disabled");
                }
            };

            // Register the listener with the Location Manager to receive location updates
            locationManager.requestLocationUpdates(
                    LocationManager.GPS_PROVIDER, 5000, 10, locationListener);
        } catch (SecurityException e) {
            Log.d("GPS Location: ", e.toString());
        }
        return new Integer(1);
    }

    public Location getLocation() {
        Log.d(TAG, "Getting location ... ");
        try {
            Log.d(TAG, "Returning location: " + locationManager.getLastKnownLocation(LocationManager.
                GPS_PROVIDER));
            return locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        } catch(SecurityException e) {
            Log.e(TAG, "Exception getting GPS location: " + e.toString());
            return null;
        }
    }
}
```

Código B.6 GPSHandler.java.

```
package com.pfc.app.iHelp.classes;
```

```
import android.os.AsyncTask;
import android.util.Log;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.ProtocolException;
import java.net.URL;
import java.util.Scanner;

public class RestClientTask extends AsyncTask<URL, Integer, JSONObject> {

    private static final String TAG = "Rest client";

    @Override
    protected JSONObject doInBackground(URL... urls) {
        HttpURLConnection urlConnection = null;
        try {
            URL url = urls[0];

            Log.d(TAG, "Calling url: " + url);
            urlConnection =
                    (HttpURLConnection) url.openConnection();

            Log.d(TAG, "Instantiated connection");

            configureConnection(urlConnection);

            int statusCode = urlConnection.getResponseCode();

            Log.d(TAG, "Response code from the server: " + statusCode);

            if (statusCode != HttpURLConnection.HTTP_OK) {
                Log.d(TAG, "Connection failed: statusCode: " + statusCode);
                return new JSONObject();
            }

            InputStream in = new BufferedInputStream(
                    urlConnection.getInputStream());

            String responseText = getResponseText(in);
            Log.d(TAG, "responseText: " + responseText);

            return new JSONObject(responseText);

        } catch (IOException | JSONException e) {
            e.printStackTrace();
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect();
            }
        }

        return null;
    }

    private HttpURLConnection configureConnection(HttpURLConnection urlConnection) {
        try {
            urlConnection.setConnectTimeout(20000);
            urlConnection.setReadTimeout(20000);
            urlConnection.setRequestMethod("GET");
            Log.d(TAG, "Configured connection");
        } catch (ProtocolException e) {
            Log.d(TAG, "Something went wrong configuring the connection");
            e.printStackTrace();
        }
```

```
            return urlConnection;
        }

        private static String getResponseText(InputStream inStream) {
            return new Scanner(inStream).useDelimiter("\\A").next();
        }
    }
```

Código B.7 RestClientTask.java.

```
package com.pfc.app.iHelp.classes;

import android.content.Context;
import android.widget.Toast;

public class ToastHandler {

    private static int duration = Toast.LENGTH_SHORT;
    private Context context;

    public ToastHandler(Context context) {
        this.context = context;
    }

    public void showUserMessage(String message) {
        Toast toast = Toast.makeText(context, message, duration);
        toast.show();
    }

}
```

Código B.8 ToastHandler.java.

```
package com.pfc.app.iHelp.classes;

import android.net.wifi.ScanResult;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.util.Log;

import java.util.List;

public class WifiHandler {

    private static final String TAG = "WifiHandler";

    private WifiManager wifiManager;
    private ToastHandler toastHandler;

    public WifiHandler(WifiManager wifiManager, ToastHandler toastHandler) {
        this.wifiManager = wifiManager;
        this.toastHandler = toastHandler;
    }

    public boolean isWifiEnabled() {
        return wifiManager.isWifiEnabled();
    }

    public boolean isAlreadyConnectedToANetwork() {
        WifiInfo wifiInfo = wifiManager.getConnectionInfo();
        if(wifiInfo.getNetworkId() < 0) {
            Log.d(TAG, "Not connected to any network.");
            return false;
        } else {
            return true;
        }
```

```
}

public boolean isConnectedNetworkValid() {
    WifiInfo wifiInfo = wifiManager.getConnectionInfo();
    Log.d(TAG, "WIFI INFO: " + wifiInfo.toString());
    if (wifiInfo.getSSID().contains("RP")) {
        Log.d(TAG, "Already connected to " + wifiInfo.getSSID());
        return true;
    } else {
        return false;
    }
}

public void scanNetworksAndConnectToBest() {

    Log.d(TAG, "Scanning networks. Checking if Wifi is enabled ...");

    if (!isWifiEnabled()) {
        Log.d(TAG, "Wifi not enabled. Enabling ...");

        toastHandler.showUserMessage("Wifi not enabled. Enabling ...");

        wifiManager.setWifiEnabled(true);
        Log.d(TAG, "Wifi enabled.");
    } else {
        Log.d(TAG, "Wifi was already enabled. Checking if it's already connected to a network ...");
        if (isAlreadyConnectedToANetwork()) {
            Log.d(TAG, "Already connected to a network.");
            if (isConnectedNetworkValid()) {
                Log.d(TAG, "Connected network is valid.");
                return;
            } else {
                Log.d(TAG, "Connected network is not valid - disconnecting ...");
                WifiInfo wifiInfo = wifiManager.getConnectionInfo();
                wifiManager.disableNetwork(wifiInfo.getNetworkId());
                wifiManager.disconnect();
            }
        }
    }

    wifiManager.startScan();
    Log.d(TAG, "Scanning ...");

    List<ScanResult> connections = wifiManager.getScanResults();
    Log.d(TAG, "List of connections: " + connections.toString());

    ScanResult bestWifi = getBestWifi(connections);

    int networkId = findExistingNetworkConfig(bestWifi.SSID);

    if (networkId < 0) {
        Log.d(TAG, "Unknown network. Adding it now...");
        addNewAccessPoint(bestWifi);
    } else {
        boolean b = wifiManager.enableNetwork(networkId, true);
        Log.d("WifiPreference", "enableNetwork returned " + b);
    }

    toastHandler.showUserMessage("Connected to Network " + bestWifi.SSID);
}

private ScanResult getBestWifi(List<ScanResult> connections) {
    Log.d(TAG, "Getting the best WiFi connection ...");

    ScanResult best = null;
    for (ScanResult scanResult : connections) {
        if (best == null || (wifiManager.compareSignalLevel(best.level, scanResult.level) < 0 && scanResult.
            SSID.contains("RP"))) {
            best = scanResult;
        }
    }
```

```
        Log.d(TAG, "Returning best WiFi SSID: " + best.SSID);

        return best;
    }

    private int findExistingNetworkConfig(String ssid) {

        Log.d(TAG, "Checking if the configuration for best WiFi already exists ...");

        if (ssid != null && !ssid.isEmpty()) {
            for (WifiConfiguration wifiConfig : wifiManager.getConfiguredNetworks()) {
                if (ssid.equals(wifiConfig.SSID)) {
                    Log.d(TAG, "It does.");
                    return wifiConfig.networkId;
                }
            }
        }
        // Didn't find a matching network ssid
        Log.d(TAG, "It doesn't.");
        return -1;
    }

    public void addNewAccessPoint(ScanResult scanResult) {

        WifiConfiguration wc = new WifiConfiguration();
        wc.SSID = '\"' + scanResult.SSID + '\"';
        wc.priority = 10000;
        wc.status = WifiConfiguration.Status.ENABLED;
        wc.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
        int res = wifiManager.addNetwork(wc);
        Log.d(TAG, "WifiPreference: add network returned " + res);
        boolean b = wifiManager.enableNetwork(res, true);
        wifiManager.saveConfiguration();

        Log.d(TAG, "WifiPreference: enableNetwork returned " + b);
    }
}
}
```

Código B.9 WifiHandler.java.

B.3 *res*

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.pfc.app.iHelp.activities.HelpMessageActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/textView3"
        android:layout_marginTop="68dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height ="wrap_content"
        android:textAppearance ="? android:attr /textAppearanceLarge"
        android:text ="Large Text"
        android:id ="@+id/textView4"
        android:layout_alignParentBottom ="true"
        android:layout_alignLeft ="@+id/textView3"
        android:layout_alignStart ="@+id/textView3"
        android:layout_marginBottom="81dp" />
</RelativeLayout>
```

Código B.10 activity_help_message.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:background="@color/lightBlue"
    android:layout_width ="match_parent"
    android:layout_height ="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context =". activities .MainActivity">

    <Button
        android:onClick="startScanning"
        android:padding="2dp"
        android:scaleX="2"
        android:scaleY="2"
        android:background="@color/darkBlue"
        android:textColor ="@color/white"
        android:layout_width ="wrap_content"
        android:layout_height ="wrap_content"
        android:text ="Start scanning"
        android:id ="@+id/button"
        android:layout_centerVertical ="true"
        android:layout_centerHorizontal ="true" />
</RelativeLayout>
```

Código B.11 activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation =" vertical " android:layout_width ="match_parent"
    android:background="@color/lightBlue"
    android:layout_height ="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:textAlignment ="center"
    android:touchscreenBlocksFocus ="true"
    android:weightSum="1">

    <TextView
        android:layout_width ="239dp"
        android:paddingTop="64dp"
        android:paddingBottom="64dp"
        android:layout_height ="wrap_content"
        android:text ="How can we help you?"
        android:textSize ="24sp"
        android:id ="@+id/textView"
        android:layout_gravity =" center_horizontal "
        android:layout_weight ="0.13" />

    <Button
        android:onClick="help"
        android:padding="2dp"
```

```
            android:scaleX="2"
            android:scaleY="2"
            android:background="@color/darkBlue"
            android:textColor="@color/white"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_marginBottom="64dp"
            android:text="Check nearest help point"
            android:textSize="12sp"
            android:id="@+id/button2"
            android:layout_gravity=" center_horizontal "
            android:breakStrategy="simple" />

        <Button
            android:onClick="rescue"
            android:padding="2dp"
            android:scaleX="2"
            android:scaleY="2"
            android:background="@color/darkBlue"
            android:textColor="@color/white"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:layout_centerHorizontal="true"
            android:text="Ask for rescue"
            android:id="@+id/button3"
            android:layout_gravity=" center_horizontal " />

    </LinearLayout>
```

Código B.12 activity_main_menu.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".activities.RescueMessageActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/textView2"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="193dp" />
</RelativeLayout>
```

Código B.13 activity_rescue_message.xml.