Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
DOBRE MENCIÓN EN ENXEÑARÍA DE COMPUTADORES
E TECNOLOXÍAS DA INFORMACIÓN

Aplicación móvil y monitorización de un dummy para la formación en emergencias sanitarias

Estudante: Juan González Iglesias
Dirección: Óscar Fresnedo Arias

A Coruña, xuño de 2021.

A mis padres y hermana

Agradecimientos

A Brais por haber puesto la idea inicial y cederme todo el material y asesoramiento necesario.

A Óscar, por todo el apoyo, ayuda y confianza durante toda la realización del proyecto.

Resumen

Cuando se produce un rescate o inmovilización en una emergencia sanitaria o en una situación de riesgo, saber actuar ante esta es primordial para intentar salvar o preservar lo máximo posible la integridad y salud del paciente que es rescatado. Por lo tanto, una buena formación y preparación previa es fundamental, y de ahí la necesidad de incorporar herramientas e instrumentos que nos ayuden en el proceso de aprendizaje y hacer que este sea lo más parecido a la realidad.

La realización de este proyecto consiste en mejorar y perfeccionar una de esas herramientas ya creadas, en nuestro caso, el maniquí de rescate o *dummy*. Monitorizando este elemento y creando una aplicación móvil, podremos visualizar en todo momento los datos más importantes a la hora de realizar un rescate o inmovilización a un paciente, posibilitando así un mayor nivel de aprendizaje. De esta forma, el objetivo es avanzar del tradicional *dummy* estático a un *dummy* tecnológico o *smart dummy*.

Estos datos nos pueden aportar información que van por ejemplo desde la temperatura o humedad que el rescatado presenta, hasta incluso, a la hora de realizar las maniobras de inmovilización, la fuerza que ejercemos en la colocación de un collarín. Otro dato de gran importancia a la hora de realizar un rescate es, por ejemplo, la última posición conocida del rescatado. Para ello, incorporaremos tecnologías de comunicación y geolocalización que nos permitirán transmitir y conocer en todo momento la última posición del *dummy*.

Por último, crearemos una aplicación móvil que se conectará con nuestro *dummy* y la cual nos va a permitir controlar y visualizar todos los datos devueltos de monitorización y posicionamiento. Esta aplicación, además de un control de usuarios y ciertas funcionalidades adicionales, incorporará una pequeña documentación en la que se mostrarán las principales técnicas de primeros auxilios y una guía de como actuar ante ciertas situaciones.

Abstract

When a rescue or immobilization takes place in a health emergency or in a risk situation, knowing how to act is essential to try to save or preserve as much as possible the integrity and health of the patient being rescued. Therefore, good training and prior preparation is essential, hence the need to incorporate tools and instruments that help us in learning and make this as close to reality as possible.

This project consists of improving and perfecting one of these tools already created, in our case the rescue dummy. By monitoring this element and creating a mobile application, we

will be able to visualise the most important data when carrying out a rescue or immobilisation of a patient enabling a higher level of learning.

This data can provide us with information ranging, for example, from the temperature or humidity of the person being rescued, to even, when carrying out immobilisation manoeuvres, the force we exert when placing a neck brace. Another piece of information of real importance when carrying out a rescue is, for example, the last known position of the person being rescued. For this we incorporate communication and geolocation technologies which allow us to transmit and know the last known position of the person being rescued.

On the other hand, we will create a mobile application, which will connect with our *dummy* and which will allow us to control and visualise all the monitoring and positioning data returned. This application, in addition to user control and certain extra functionalities, will incorporate a small documentation in which the main first aid techniques and how to act in certain situations will be listed.

Palabras clave:

- formación
- emergencias
- dummy
- monitorización
- control
- aplicación
- geolocalización
- Sigfox
- BLE

Keywords:

- training
- emergency
- dummy
- monitoring
- control
- application
- geolocation
- Sigfox
- BLE

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introdución

La monitorización y control de un *dummy* para la formación en emergencias sanitarias y el desarrollo de una aplicación móvil para controlar y visualizar estos datos puede suponer una serie de ventajas y alicientes para que estas prácticas sean lo más realistas y controladas posible. Por esa razón, en este trabajo vamos a llevar a cabo la creación de una aplicación móvil y la incorporación de una serie de sensores y módulos a un *dummy* que permitan controlar, visualizar y monitorizar los principales datos o aspectos que son más importantes a la hora de actuar ante una emergencia o rescate sanitario.

1.1 Origen y contexto del proyecto

El ámbito sanitario es uno de los más relevantes e importantes dentro de nuestra sociedad y, en especial, el hecho de saber como actuar ante un rescate o una situación de riesgo y hacerlo de la manera correcta puede definir el futuro del paciente al que se está tratando. Es por ello que existen multitud de acciones de formación y estudios que guardan una estrecha relación con este tema y que cada vez más gente decide encaminar su vida laboral por esta rama.

Uno de estas actividades de formación se centra en la inmovilización y traslado de los pacientes en una situación de emergencia. En este caso, es fundamental realizar una adecuada inmovilización del paciente ya que esto permite estabilizar lesiones existentes y evitar que se produzcan lesiones secundarias que agravarían aún más su estado, además de intentar reducir el dolor y la posibilidad de *shock* del paciente. Estas inmovilizaciones se hacen por personal especializado del equipo de rescate en el mismo lugar donde se produjo el accidente, y es por ello que muchas veces la situación o el entorno no favorecen el rescate, lo que puede llegar a perjudicar o dificultar la extracción del paciente.

Además, la mayor parte de las veces cuando se llega al accidentado aún no se tienen todas las pruebas necesarias para hacer un diagnóstico fiable y, por ello, no hay seguridad sobre

el alcance de las lesiones. Por eso, es importante que el herido se mueva lo menos posible y aplicar bien los distintos elementos de inmovilización para no empeorar su estado hasta que se realicen las pruebas pertinentes en el hospital y llegar así a un diagnóstico final.

Por esta razón, a la hora de realizar los rescates, o mejor dicho, las correspondientes inmovilizaciones, es importante que tengamos en cuenta factores como la inclinación que sufre el paciente al ser levantado o la aceleración del propio movimiento al levantarlo hacia la camilla. A mayores, tenemos que tener en cuenta la presión que podemos llegar a ejercer con un torniquete, una férula o un colchón de vacío en las distintas partes del cuerpo, pues si ejercemos demasiada presión podemos llegar incluso agravar el estado del paciente.

1.2 Motivación

Para el instructor que dirige una práctica de rescate e inmovilización es muy importante tener en todo momento controlado que es lo que está haciendo su alumno y, quizás lo más importante, si este lo está realizando correctamente. Es muy complicado medir el nivel de fuerza que un collarín está ejerciendo sobre el cuello del *dummy* que se utiliza en este tipo de prácticas, o si se está moviendo demasiado rápido al *dummy* en el momento de realizar las tareas de movilización. Por ello, surge la necesidad de intentar controlar todos estos procesos en las actividades de formación y proporcionar, ya no solo al instructor, si no también a quien está realizando esa práctica, una métrica objetiva que permita evaluar lo que está haciendo, si lo hace de forma correcta y si las acciones se hacen con suficiente precisión.

Realizar prácticas de rescate, por ejemplo en montaña, lo más realistas posibles es muy complejo, pues establecer la última posición conocida del rescatado no es del todo fácil o realista. El instructor tiene que llevar al *dummy* a un lugar en concreto y dejarlo allí hasta encontrarlo, por lo que la posición no puede variar. Sin embargo, en un rescate real el paciente se puede estar moviendo y cambiando de zona o encontrarse en otro punto distinto de donde aseguraba estar. Es por ello que, gracias a la utilización de un *smart dummy* adecuadamente equipado, el instructor puede modificar la posición de este para que los alumnos puedan visualizar a través de la aplicación los posibles pasos que siguió, simulando así un verdadero rescate.

Por lo tanto, gracias a la monitorización y control de diferentes parámetros y datos devueltos por el *dummy* y, al mismo tiempo, disponer de herramientas para simular rescates de la forma más realista posible sería muy beneficioso para el instructor, pues le otorgaríamos la capacidad objetiva de evaluar si estas prácticas se están realizando de la forma correcta y así no tener que disponer de otros materiales, herramientas o tiempo para comprobar estas métricas.

Todas estas necesidades que se han detectado en el contexto de las actividades de for-

mación sanitaria o prácticas de rescate usando los tradicionales *dummies*, han motivado el desarrollo de este proyecto, donde se busca dotar al *dummy* de un sistema hardware completo que nos permita monitorizarlo a través de una aplicación que puede ser usada de forma sencilla desde un dispositivo móvil.

1.3 Objetivos

Como se ha introducido en los puntos anteriores, el principal objetivo de este proyecto consiste en el desarrollo de un sistema integral para poder monitorizar y ver el estado de un maniquí de rescate. En definitiva, crear un *smart dummy* que puede ayudar en la instrucción, gestión y monitorización de actividades de formación/prácticas que suelen emplear este tipo de elementos. De esta forma, el proyecto incluye el desarrollo de un sistema hardware capaz de obtener y proporcionar todos los parámetros de interés sobre el *dummy*, además de la creación de una aplicación móvil para la visualización y análisis de los diferentes datos obtenidos. Esta idea la podemos descomponer en los siguientes objetivos parciales:

- **Familiarizarse con las técnicas y acciones requeridas en un rescate.** Esto va a permitir detectar los parámetros principales que se deben tener en cuenta y las tecnologías/hardware más adecuado para obtener esa información.
- **Sensorización completa del dummy**. Se añadirán una serie de sensores y módulos repartidos por todo el cuerpo del maniquí que nos proporcionarán diferente información acerca de este. A mayores, se incluirán una serie de dispositivos de geolocalización y transmisión para poder realizar las diferentes prácticas con nuestro *smart dummy*.
- **Diseñar y crear una aplicación móvil**. Gracias a la cual podremos controlar y visualizar en todo momento el estado y los parámetros devueltos por el *dummy*.
- **Posibilidad de añadir un mayor grado de conocimiento y funcionalidad**, gracias en parte a la falta de herramientas similares en el mercado. El objetivo es ofrecer la posibilidad de saber si un rescate o inmovilización sanitaria se está realizando de una manera correcta desde una perspectiva objetiva. Además, se añadirá un apartado para algunas de las guías de primeros auxilios más básicas, para poder así establecer un conocimiento básico previo de algunas técnicas sanitarias.

1.4 Estado del arte

En el aspecto de monitorización y control de *dummies* **no existen soluciones o alternativas en el mercado** como tal, ya no solo a nivel español, sino que tampoco a nivel internacional. Existe una gran variedad de maniquíes de rescate creados y confeccionados para distintas

situaciones como pueden ser de uso general, de rescate acuático o de rescate en situaciones de fuego, pero no disponen de ningún hardware adicional que permita su sensorización y monitorización.

Una de las empresas más importantes de nuestro país en términos de material de simulacros de emergencia y prácticas sanitarias es **Maxpreven** [4]. Esta empresa proporciona, entre otras cosas, una gran variedad de *dummies* para diferentes finalidades a precios que son verdaderamente altos para las características tan simples que ofrecen. Sin embargo, ninguno de estos *dummies* incorpora ningún tipo de sensor que permita su monitorización o control.

A nivel mundial, otra empresa muy conocida es **LifeTec** [5] que tiene un catálogo muy similar al de la anterior. Esta empresa incorpora alguna novedad como son maniquíes de rescate con capacidad para realizarles la maniobra RCP [6] o un *dummy* que ellos catalogan como el más novedoso y que es llamado *Advanced Water Rescue*. Este *dummy* está diseñado para imitar a una persona ahogada en el agua y con el cual se pueden llevar a cabo diferentes técnicas de primeros auxilios como son la inserción de vías respiratorias, intubación traqueal, ventilación con mascarilla pero, en ningún caso, aportando información de control para saber si todas estas acciones se están realizando de la forma correcta.

A partir de este análisis preliminar, se ha detectado claramente la falta de productos o soluciones similares a la propuesta de este proyecto. No hay ningún maniquí de rescate en el mercado que permita una simulación, monitorización y controles sencillos, prácticos y visuales de los principales aspectos a tener en cuenta a la hora de realizar una inmovilización inicial y posterior movilización en un rescate o situación de emergencia.

1.5 Smart dummy y aplicación móvil

Una vez identificada la necesidad que se busca cubrir en el contexto de los rescates y emergencias sanitarias, puede resultar de gran ayuda definir las líneas principales sobre las que se va asentar el sistema que se va a desarrollar en este proyecto. Esto nos va a permitir tener una visión general de todo el sistema en su conjunto desde un principio, con sus principales módulos y funcionalidades. En primer lugar, el sistema estará formado por dos grandes bloques:

- Un componente hardware de monitorización que incorporará diferentes sensores, sistemas de geolocalización y diferentes módulos de comunicación para transmitir toda la información necesaria.
- Una aplicación móvil desarrollada para sistemas Android que accederá y mostrará los diferentes parámetros relacionados con las acciones que se tengan que realizar sobre el *dummy*. Esta parte incluye la necesidad de implementar un mecanismo para la gestión de los usuarios, el almacenamiento de los datos y la recuperación de los mismos.

De esta forma, las principales funcionalidades a implementar con el sistema desarrollado se podrían agrupar en los siguientes puntos:

- Control de los parámetros más importantes del *dummy* a la hora de realizar un rescate. Entre ellos se encuentran la temperatura, humedad, inclinación, aceleración y fuerza aplicada en diferentes partes del cuerpo.
- Geolocalización e identificación de cada *dummy*. Esto se puede utilizar para realizar prácticas de búsqueda en rescates. Además, sería interesante la incorporación de un histórico de cada uno de los lugares por los que pasó el *dummy*.
- Detección de si se está realizando incorrectamente alguno de los ejercicios en una práctica de rescate.
- Control y reproducción de diferentes sonidos u alertas durante una práctica con el *dummy*. Esto resulta muy útil si queremos reproducir alertas cada vez que nos pasamos de un umbral sobre los parámetros de medición o si queremos incorporar la posibilidad de reproducir diferentes sonidos o frecuencias para prácticas de búsqueda en un rescate.
- Posibilidad de visualizar todos los datos de interés de una manera sencilla e intuitiva, facilitando así el aprendizaje y la mejora de las técnicas utilizadas.
- Sistema de control de usuarios, el cual permite registrar diferentes *dummies* y vincular cada uno de ellos con un usuario administrador que será el encargado de establecer toda la configuración del mismo.
- Posibilidad de consultar la documentación o una breve guía sobre las acciones a realizar durante un rescate para facilitar la aplicación de las técnicas de primeros auxilios más básicas.

1.6 Estructura de la memoria

Esta memoria tiene como objetivo documentar los aspectos más relevantes del proyecto que se ha realizado y consta de la siguiente estructura:

1. **Introducción**. En este capítulo, se ha explicado la idea y motivación principal de este proyecto. Se ha presentado además una revisión del estado del arte para justificar su utilidad, se han comentado los objetivos principales y una breve exposición de las funcionalidades requeridas.
2. **Fundamentos tecnológicos**. Se enumerarán y explicarán las distintas tecnologías usadas para este proyecto. Además, se incluirá un pequeño análisis de las posibles alternativas y justificación de las herramientas elegidas.

3. **Metodología, planificación y estimación de costes**. Capítulo en el que se exponen las características de la metodología usada, así como su aplicación en el proyecto. También se realiza un análisis de la planificación seguida y un estudio de los costes del proyecto.

4. **Visión general del proyecto**. Se presentarán en detalle los diferentes módulos del sistema y se explicarán las diferentes funcionalidades y características del mismo.

5. **Diseño, desarrollo e implementación del hardware**. En este capítulo, se analiza la parte hardware del proyecto, incluyendo el *dummy*, y todos los módulos y sensores que hacen posible la recolección y envío de información acerca del maniquí.

6. **Análisis, diseño e implementación de la aplicación móvil**. Se expone en este capítulo todo el proceso que sigue el desarrollo de nuestra aplicación móvil pasando por la fases de análisis, diseño e implementación.

7. **Pruebas realizadas**. Capítulo en el cual se exponen las diferentes pruebas y tests realizados al proyecto y a las distintas partes que lo componen.

8. **Conclusiones y lineas futuras**. Se hará un análisis de los resultados obtenidos con respecto a los objetivos iniciales, se conectará el trabajo realizado con las competencias de las dos menciones cursadas por el estudiante y se comentarán las posibles mejoras o líneas futuras del proyecto.

Capítulo 2

Fundamentos tecnológicos

En este capítulo vamos a explicar las tecnologías utilizadas para la realización de este proyecto. Podemos hacer una clasificación general distinguiendo tres grupos:

- **Comunicaciones**. Tanto las realizadas entre el *dummy* y la aplicación móvil, como las utilizadas para obtener la ubicación y enviarla al servidor.
- **Hardware**. Componentes utilizados para medir y obtener distinta información acerca del *dummy*.
- **Software**. Conjunto de tecnologías software utilizadas tanto para el propio desarrollo del proyecto como de apoyo.

2.1 Comunicaciones

Estas tecnologías se utilizarán para la transmisión de datos y para la geolocalización del *dummy*.

2.1.1 Sigfox

En primer lugar tenemos que introducir el concepto de redes Lower Power Wide Area Network (LPWAN). Se trata de un tipo de tecnología de comunicación inalámbrica que permite transmitir datos a larga distancia con un consumo energético muy bajo. Por tanto, presentan tres características fundamentales: un largo alcance geográfico, pequeñas cantidades de datos transmitidos y un bajo consumo de energía. Es por ello que este tipo de tecnología se utiliza principalmente para dispositivos en Internet of Things (IoT) y comunicaciones Machine to machine (M2M).

Existen multitud de tecnologías pertenecientes a LPWAN, pero quizá las más conocidas o las que mejor se podrían adecuar a nuestro proyecto sería LoRa[7] o **Sigfox** [8]. Estas tecnologías de comunicación inalámbrica nos permiten transmitir datos a grandes distancias con

un muy bajo consumo por lo que, para nuestro proyecto, son la mejor opción. Después de analizar estas dos tecnologías, se ha llegado a la conclusión de que la más adecuada para el desarrollo del proyecto es **Sigfox**, pues a diferencia de LoRa, entre otras cosas, no tenemos la necesidad de incorporar una antena receptora en nuestro despliegue.

Sigfox es una red LPWAN patentada y ofrecida por una empresa francesa [9]. Actualmente es una de las redes de estas características más grandes y con mayor cobertura. Además, la red **Sigfox** se basa en cinco principios básicos:

- **Ultra-Narrow Band (UNB) [10]**: **Sigfox** tiene asignada una banda de 192 KHz en el espectro para el envío de mensajes. Para enviar cada mensaje se necesita un ancho de banda de 100Hz y en Europa la banda utilizada está entre los 868 y 868.2MHz.
- **Random access**: La transmisión no está sincronizada entre la red y el dispositivo. Este envía un mensaje en una frecuencia aleatoria y luego envía dos réplicas en diferentes frecuencias y tiempo.
- **Cooperative reception**: Un emisor no se conecta a una estación base específica, sino que el mensaje puede ser captado por cualquier antena que esté en el área de alcance.
- **Small messages**: El tamaño de los mensajes tiene que estar en el rango de entre 0 y 12 bytes.
- **Bi-directional**: Aunque el enlace sea bidireccional, el mensaje de "vuelta" lo inicia el dispositivo Sigfox.

La arquitectura de **Sigfox** es muy simple y presenta la distribución que podemos ver en la figura 2.1. Como se puede apreciar, los distintos dispositivos finales envían los datos por la interfaz del aire a las estaciones base. Estas estaciones base y su correspondiente red de antenas están repartidas por más de 60 países. Las estaciones están compuestas por tres elementos principales: una antena, un low-noise amplifier (LNA) [11] para amplificar la señal y filtrar el ruido, y un punto de acceso que comprende los mensajes **Sigfox** y los envía al *Sigfox Cloud*. Una vez que las antenas reciben el mensaje, estos pasan por el *backhaul* [12] hacia el servidor *back-end*. Este enlace intermedio generalmente usa conectividad Digital Subscriber Line (DSL) [13] y 3G, 4G o conexión por satélite como respaldo.

El *back-end* es el responsable de manejar el procesamiento de los mensajes. Para cada mensaje, se guardan fundamentalmente dos informaciones, los meta-datos que se pueden utilizar para la construcción de servicios y los propios mensajes. Finalmente, podemos acceder a estos datos a través de una interfaz web o utilizar una API Rest que permite el uso de *callbacks*.

Por último, comentar que **Sigfox** nos ofrece una cobertura casi total en España [14]. Casi todo el territorio español goza de buena cobertura, quitando algunas partes que ellos mismos

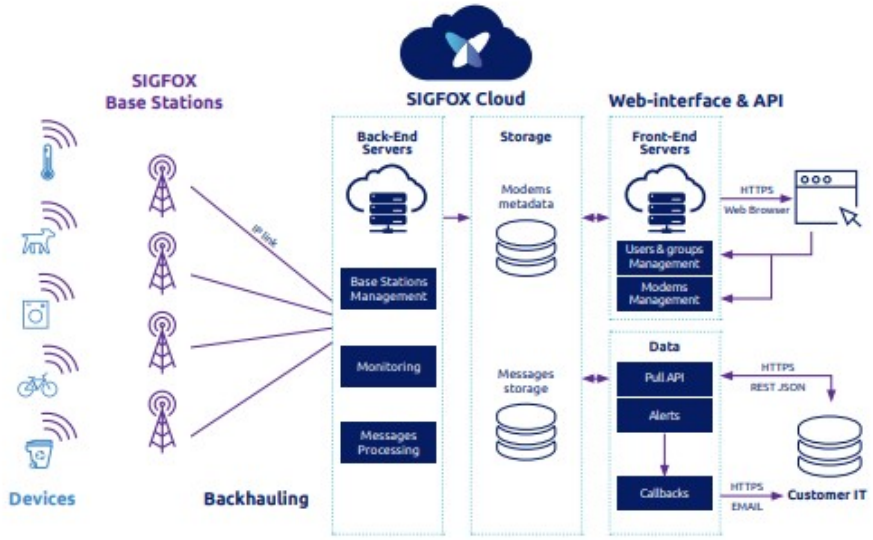


Figura 2.1: Estructura de la red Sigfox

catalogan de "zonas en desenvolvimiento". Esto nos resulta muy beneficioso ya que, a la hora de realizar diversas prácticas de rescate en lugares de complicada accesibilidad, gozaremos de suficiente cobertura para intentar localizar el *dummy*.

2.1.2 GPS

Para poder establecer la posición del *smart dummy* para realizar un simulacro de rescate, una de las posibles tecnologías es Global Positioning Sytem (GPS) [15]. Este sistema de posicionamiento global es un servicio formado por 24 satélites que transmiten señales unidireccionales que están compuestas por 3 partes [16]:

- La primera parte contiene la hora GPS, que es determinada a partir de relojes atómicos.
- La segunda parte se conoce como *ephemeris* y contiene las coordenadas astronómicas del satélite.
- La tercera parte contiene información sobre la posición del satélite con respecto al resto de satélites y se conoce por el nombre de *almanac*.

El procedimiento para realizar el posicionamiento de un objeto con esta tecnología es muy sencillo. El receptor GPS conoce la hora exacta a la que fue enviada la señal procedente de un satélite. El receptor puede utilizar ese dato para calcular la distancia a la que se encuentra el satélite, simplemente restando la hora en la que se transmitió el mensaje de la hora en la que se recibió. A mayores, también conoce la posición exacta en el cielo de los satélites en

el momento en que enviaron la señal por lo que, dado el tiempo de viaje de las señales de 3 satélites distintos y la posición exacta de cada uno, el receptor GPS puede determinar su posición en tres dimensiones: este, norte y altitud.

Existen multitud de módulos GPS con diferentes características, pero todos ellos tienen un funcionamiento muy similar, que se ve reflejado en la figura 2.2. La señal del GPS es recibida a través de la antena del módulo que está conectado a este último mediante un cable de Radio Frequency (RF) [17]. El módulo GPS procesa la señal y la envía mediante una interfaz de conexión de tipo UART [18] al micro-controlador incorporado en el propio módulo.

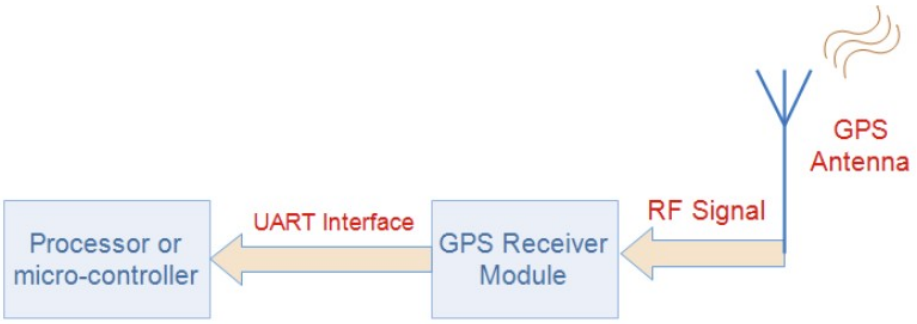


Figura 2.2: Funcionamiento módulo GPS [1].

A mayores, este tipo de módulos GPS suelen utilizar el protocolo National Marine Electronics Association (NMEA) [19] que determina el formato de los mensajes enviados. Este formato está definido por secuencias de líneas de datos llamadas frases. Cada frase contiene una serie de bits de datos organizados en un formato delimitado por comas. Existen diferentes tipos de mensajes, pero quizá el más conocido y el que más nos interesa es el **$GPGGA**. Este mensaje contiene el tiempo (en formato UTC [20]), longitud y latitud, número de satélites vistos y la altitud.

2.1.3 BLE

También conocido como Bluetooth Low Energy [21] o **Bluetooth Smart**, se basa en el estándar Bluetooth 4.0 y, al igual que este, opera en la banda ISM de 2.4 GHz. Fue diseñado y comercializado por el *Bluetooth Special Interested Group* (Bluetooth SIG) y está pensado principalmente para proporcionar un consumo mínimo de energía con un coste computacional muy bajo, por lo que es muy utilizado para aplicaciones de IoT. En nuestro proyecto, no vamos a necesitar estar transmitiendo todo el tiempo y la cantidad de datos transmitidos va a ser muy pequeña, por lo que preservar la duración de la batería será el aspecto más importante. En este sentido, BLE se adapta perfectamente a los requisitos del proyecto en la parte de transmisión de la información.

Existen otras alternativas como son *ZigBee* [22] o *Z-Wave* [23], pero ambas son redes de malla que buscan la interconexión y comunicación entre varios dispositivos, teniendo que usar para ello un dispositivo puente o *hub*, por lo que su propósito e integración no resultan tan adecuados como BLE para este proyecto.

Los dispositivos BLE permanecen más tiempo en modo espera que enviando información y el establecimiento de la conexión es más rápido que en el **Bluetooth** clásico, estableciéndose en unos pocos milisegundos frente a los cerca de 100 milisegundos que ofrece Bluetooth. Además, BLE ofrece un rango de conexión de unos 100 m, una velocidad de trasmisión de datos alrededor de 1 Mbps y un consumo entre 0.01 a 0.5 W. Todas estas características son más que suficientes para cubrir los escenarios de comunicación que tenemos entre el *dummy* y la aplicación.

En cuanto a su funcionamiento, debemos diferenciar dos tipos de servicios o perfiles:

- **Generic Access Profile (GAP)**: Controla las conexiones y hace que el dispositivo sea visible para el resto. Hay dos mecanismos para comunicarse con el mundo exterior, emitir información o conectarse explícitamente a un dispositivo. Dependiendo de esto, podemos establecer dos tipos de roles:
  - Si se trata de una conexión de emisión (*broadcasting*), diferenciamos al dispositivo *broadcaster* y al *observer*.
  - En cambio, si se trata de una operación de conexión simple, distinguimos entre *peripheral* y *central*.
- **Generic Attribute Profile (GATT)**: Una vez los dispositivos estén conectados, el servicio GATT describe detalladamente como se transfieren los datos. Hace uso del protocolo ATT [24] que se utiliza principalmente para almacenar servicios, características y datos relacionados en una tabla. En particular, en esta tabla guardamos información de los siguientes componentes:
  - **Servicios**: Son colecciones de información. Cada servicio tiene un UUID propio y pueden estar formados por una o más características.
  - **Características**: Son el punto principal en el que interactuamos con nuestro periférico BLE. Podemos tanto leer como escribir en ellos, pero solo un valor por característica.

  Para cada entrada de la tabla, se usa un UUID de 16 bits, tanto para las características como para los servicios.

2.2 Hardware

Para la captura, tratamiento y envío de todos los datos obtenidos para el *dummy* es necesario la incorporación de distintos tipos de módulos y dispositivos hardware que nos faciliten esta función.

El componente principal para un propósito así es una placa de desarrollo. Este tipo de elementos son dispositivos que cuentan con un microcontrolador reprogramable, es decir, podemos ejecutar instrucciones para un fin específico. Existen multitud de variantes en el mercado, de las cuales quizás las más conocidas sean *Arduino*, *Heltec* o *Onion*, orientada específicamente para IoT.

2.2.1 Placa de Arduino

Arduino nació en el año 2005, y se caracteriza por ofrecer una plataforma libre, extensible y fácil de utilizar. Su entorno de programación [25] es multiplataforma y el lenguaje de programación está basado en C++, esbozando una serie de *sketches* o bocetos.

Existen gran variedad de placas o modelos, pero la mayoría de estas consisten en un microcontrolador SAMD21 Cortex con cantidades variables de memoria flash, pines y funciones. A mayores y dependiendo de las funcionalidades a desarrollar, estas placas pueden contener chips con funciones *WiFi*, BLE, Global Navigation Satellite System (GNSS) [26], o diferente número de pines de entrada y salida.

Arduino se adapta perfectamente a este proyecto pues nos proporciona las herramientas y capacidades necesarias para la implementación de nuestras funcionalidades en la parte hardware, además de contar con un precio reducido y la existencia de numerosas librerías que nos facilitan el trabajo. Por estas razones, en este proyecto se va a utilizar la placa **Arduino Mkr Wifi 1010** que dispone de conexión BLE y que cuenta con suficientes pines de entrada/salida para la conexión de los diferentes módulos hardware necesarios en el proyecto.

2.2.2 Control de la inclinación-aceleración

Para poder monitorizar la inclinación y aceleración que sufre el *dummy*, tenemos varias alternativas:

- **Sensor Tilt** [27]: este sensor proporciona una señal digital en caso de que su inclinación supere un determinado umbral. Este tipo de sensores no nos permiten saber el grado de inclinación del dispositivo, solamente nos avisan a partir de una cierta inclinación (umbral).

- **Sensor de aceleración lineal**: Está formado por una cápsula hermética que contiene

un contacto normalmente abierto, y al producirse una aceleración o impacto, normalmente superior a una fuerza 5 g, este contacto se cierra.

- **Acelerómetro**: Estos sensores están destinados a medir aceleraciones y podemos dividirlos en tres grupos principales:
  - Acelerómetros capacitivos MEMS [28]: Se utilizan principalmente en dispositivos portátiles, equipos móviles y productos electrónicos de consumo pues es posible su implementación directamente en una placa de tipo Printed Circuit Board (PCB).
  - Acelerómetros piezorresistivos: Su principio de funcionamiento es similar al de un extensómetro. Están equipados con material piezorresistivo que, bajo la influencia de una fuerza externa, este se deforma.
  - Acelerómetros piezoeléctricos: este tipo de acelerómetros son normalmente utilizados para medir el nivel de vibración. Se caracterizan por su alta sensibilidad y precisión.

  La mayoría de estos tres tipos de sensores están compuestos de 3 ejes que pueden medir tanto las fuerzas dinámicas como estáticas (gravedad) de aceleración. Las conexiones básicas que requieren este tipo de sensores pueden ser analógicas, digitales o mediante una interfaz de tipo pulse-width modulation (PWM). Las conexiones digitales son las más utilizadas y pueden utilizar los protocolos de comunicación SPI [29] o I2C [30].

Para este proyecto se ha elegido un acelerómetro de tipo capacitivo que tiene una interfaz de conexión I2C. Estos son los más sencillos de utilizar pues ya vienen montados directamente en una PCB y aportan unos valores lo suficientemente precisos para nuestro propósito. A mayores, la placa **Arduino Mkr Wifi 1010** permite este tipo de conexión I2C.

2.2.3 Control de la temperatura

La temperatura y humedad a la que se encuentra el cuerpo de un rescatado, sobre todo en condiciones de alta montaña, es uno de los factores clave a tener en cuenta a la hora de realizar cualquier rescate. Por ello, es necesario medirla y así comprobar que se están aplicando técnicas de rescate o métodos adecuados. Dentro de las opciones para medir la temperatura, encontramos las siguientes:

- **Termopares**: Son los más empleados, y su funcionamiento se produce por un diferencial de tensión entre dos hilos metálicos de diferentes materiales.
- **Resistance Temperature Detector (RTD)**: Están indicados para la medición de la temperatura y condiciones de humedad en entornos industriales y su funcionamiento se basa en la resistencia a la temperatura de un material específico.

- **Termistores Negative Temperature Coefficient (NTC) y Positive Temperature Coefficient (PTC)**: Están compuestos por materiales semiconductores y su medida proviene de la variación de la resistencia que ejercen estos materiales dependiendo de la temperatura. Los termistores PTC señalizan un aumento de la temperatura cuando se detecta una mayor resistencia, mientras que los NTC señalizan el aumento de la temperatura a partir de una reducción de la resistencia. Estos sensores trabajan en unos rangos de temperatura acordes con las variaciones que se pueden producir en un paciente. Por esta razón, por su facilidad de uso y por su bajo coste, han sido elegidos para este proyecto.

Para medir la humedad existen diversas alternativas como pueden sensores mecánicos, infrarrojos, basados en sales higroscópicas pero, quizás, los más conocidos y utilizados son los sensores de humedad **capacitivos**. Estos sensores son de fácil producción, con unos costes muy bajos y poseen una alta fidelidad, por lo que es el que vamos a utilizar en nuestro proyecto.

2.2.4 Control de la fuerza o presión ejercida

Otro de los aspectos a tener en cuenta, al realizar una inmovilización o rescate, es la presión que es aplicada cuando colocamos un collarín o una férula. Si ejercemos poca presión o fuerza sobre una determinada parte del cuerpo, no conseguiremos mantenerla en la posición adecuada para su correcta inmovilización. Por el contrario, si ejercemos una excesiva fuerza, podemos incluso empeorar la situación del paciente.

Existen pocas alternativas en el mercado para medir este tipo de presión o fuerza. La más conocida son los sensores Force Sensitive Resistor (FSR) o sensores de fuerza resistivos. Estos sensores emplean un material que cambia su resistencia cuando se le aplica una fuerza o presión. En concreto, en este tipo de sensores la resistencia disminuye con el aumento de la presión. Además, presentan una serie de características como son un tamaño pequeño que se puede amoldar a cualquier proyecto, un bajo coste y una buena resistencia a los golpes.

Existe la alternativa de los sensores de celdas de carga. Estos dispositivos son básicamente transductores que convierten la fuerza en una salida eléctrica medible. Son utilizados principalmente para proyectos en los que se requiere una precisión muy elevada y que normalmente cumplen con la función de una báscula.

Teniendo en cuenta las características de cada tipo de sensor, nos hemos decantado por el uso de sensores Force Sensitive Resistor (FSR) para la medición de la fuerza y presión aplicada en zonas específicas del cuerpo del *dummy*, como pueden ser cuello, pierna y brazo. Estos sensores nos proporcionan una buena resistencia ante golpes y, al no ser totalmente rígidos, se pueden acoplar correctamente en las posiciones de interés en el *dummy*, facilitando así su uso y medición.

2.2.5 Sigfox y posición GPS

Si queremos utilizar las tecnologías **Sigfox** (sección 2.1.1) y GPS (sección 2.1.2) en nuestro proyecto, debemos de incorporar una serie de módulos hardware que nos proporcionen estos servicios.

En lo que respecta a **SigFox**, existen multitud de alternativas. Una de las más conocidas y usadas es la proporcionada por Arduino a través de su PCB MKR FOX 1200. Esta placa incorpora un microchip (RF ATA 8250) que funciona en el rango de frecuencia de 868MHz, por lo que no es necesario el uso de ningún otro chip adicional para conectarnos a la red **SigFox**. Otra de las principales alternativas es el uso de módulos hardware independientes que pueden ser usados/integrados en cualquier tipo de placa. Existen diversas propuestas como pueden ser los módulos *BRKWSO1*, *STEVAL* o *DVK*.

Por otra parte, para poder establecer la posición exacta del *dummy*, necesitamos usar algún sistema de posicionamiento como es GPS. De igual manera que para **Sigfox**, también existen diferentes aproximaciones. La más conocida son los módulos NEO-6M de la compañía **u-blox** [31], que se pueden utilizar junto con cualquier placa de desarrollo. Permite la conexión de hasta 22 satélites simultáneos y se caracterizan por un bajo consumo de energía.

Otra posible aproximación sería el uso de módulos hardware que implementasen la tecnología Global System for Mobile communications (GSM) [32]. Aparte de proporcionar un soporte global para las comunicaciones móviles, como su propio nombre indica, GSM también incluye la capacidad de localización (*GSM Location*). A diferencia de GPS, esta tecnología nos ofrece la ubicación con una menor precisión, oscilando el error entre los 300 o 1000 metros. También existe la posibilidad de utilizar productos finales ya montados como, por ejemplo, los localizadores GPS [15]. La mayoría de ellos tienen además capacidad para incluir una tarjeta SIM, lo que nos permite enviar los datos directamente al servidor. Esta alternativa es menos flexible y más costosa que implementar la geolocalización utilizando módulos independientes.

Considerando todos estos aspectos, la aproximación elegida es la utilización de módulos hardware independientes, tanto para la comunicación mediante **Sigfox**, como para la localización mediante GPS. Esta alternativa ofrece una mayor flexibilidad y un menor coste que las opciones anteriormente comentadas.

2.2.6 Sistema de sonido

Para habilitar la reproducción de diversos sonidos por parte del usuario o para la señalización mediante una alerta de algún ejercicio o técnica realizada de forma incorrecta, necesitamos un dispositivo hardware capaz de emitir sonidos. En **Arduino**, esto se consigue mediante la utilización de *buzzers* o de altavoces. La segunda opción es la más interesante, puesto que los *buzzers* producen un tono dependiendo del voltaje aplicado pero, en general, estos sonidos

tienen un volumen demasiado bajo.

Por esta razón, se ha decidido utilizar un par de altavoces para reproducir los diferentes sonidos. Estos sonidos son generados mediante la reproducción de diferentes frecuencias, puesto que los diferentes tipos de alerta se pueden crear mediante la variación y combinación de diferentes frecuencias del tono. Aún así, el volumen del tono producido directamente por los altavoces no es lo suficientemente alto, por lo que es necesario la utilización de un amplificador.

2.3 Software

En cuanto al software utilizado para el proyecto, estas herramientas se han utilizado principalmente para el desarrollo e implementación de la aplicación móvil, aunque siempre relacionándose con las otras partes del proyecto.

2.3.1 Lenguaje de programación

En la actualidad hay multitud de lenguajes para el desarrollo de aplicaciones móviles. Podemos hacer una clasificación global que distinguiría 3 grupos:

- **Lenguajes Android**: **Android** es el sistema operativo para dispositivos móviles más utilizado, pues su presencia no solo se reduce a *smartphones*, sino que también se puede utilizar para *tablets*, *smartwatchs*, televisiones, etc. Dentro de estos lenguajes, los que más destacan y los más utilizados hoy en día son *Java* [33] y *Kotlin* [34]. En concreto, este último es de los que más ha crecido en los últimos años.

- **Lenguajes IOS**: para dispositivos IOS, los lenguajes de desarrollo más utilizados son *Swift* [35] o *Objective-C* [36]. Este último fue el lenguaje de desarrollo original para IOS, mientras que *Swift* ha aparecido más recientemente pero, a día de hoy, es el más empleado.

- **Lenguajes *Cross-Platform***: con ellos, es posible adaptar el desarrollo de aplicaciones tanto para IOS como para **Android** a partir de un mismo código. Los lenguajes más conocidos que permiten hacer este tipo de desarrollo son *JavaScript* [37], *TypeScript* o, quizás el más conocido, *Flutter* [38], que nos permite desarrollar aplicaciones móviles tanto para IOS y **Android** como para entornos web.

En este caso, el lenguaje elegido ha sido **Java** para el desarrollo de aplicaciones **Android**. Gracias a las nociones aprendidas durante varias asignaturas de la carrera, muchos conceptos de este lenguaje resultan familiares para el estudiante. Además, este lenguaje nos ofrece

un paradigma de programación basado en la orientación a objetos, lo que nos permite crear nuestra aplicación de forma más modular.

2.3.2 Base de datos y gestión de la información

La correcta gestión de los datos y de la información en una aplicación es de los aspectos más importantes a tener en cuenta. Existen multitud de herramientas para gestionar estos recursos como pueden ser Parse [39], Back4App [40] y, quizás la más utilizada, Firebase [41].

Firebase se define como un conjunto de herramientas para construir, mejorar y hacer crecer nuestra aplicación, pues ofrece una gran variedad de servicios que resultan muy útiles. Para este proyecto, los servicios que más nos interesan son los siguientes:

- **Realtime database**: también conocida como base de datos en tiempo real, nos permite guardar los datos y toda la información necesaria para la aplicación en tiempo real. Estos datos se almacenan en la nube, en formato JavaScript Object Notation (JSON) y en una base de datos no relacional. Cada vez que un usuario añada, modifique o elimine un dato, **Firebase** enviará automáticamente eventos a las aplicaciones enlazadas.
- **Autenticación de usuarios**: permite tanto el registro como la gestión de usuarios en nuestra aplicación, de una forma segura y protegiendo los datos de los mismos.
- **Almacenamiento en la nube**: disponible tanto en *Firestore Database* como en *Storage*, permite guardar ficheros y datos de las aplicaciones conectadas, además de sincronizarlos, siendo esto personalizable mediante reglas.

Por tanto, **Firebase** nos ofrece una serie de características y utilidades esenciales para cumplir con los principales requisitos que necesita nuestra aplicación con respecto al tratamiento y almacenamiento de toda la información procedente del *dummy* y del propio usuario.

2.3.3 Entornos de desarrollo, software y librerías utilizadas

Para desarrollar la aplicación móvil, y para implementar la configuración de los diferentes módulos hardware y de la placa de **Arduino**, es necesario el uso de diferentes entornos de desarrollo:

- Para escribir, generar, probar y depurar la aplicación móvil se utilizará el entorno de desarrollo de **Android Studio** [42]. Este IDE está basado en **IntelliJ IDEA** y utiliza un sistema de compilación basado en **Gradle** [43].
- Para generar los distintos *mockups* de la aplicación se hizo uso de la herramienta **Adobe XD** [44] la cual, entre otras cosas, nos permite crear y compartir interfaces de aplicaciones móvil y web.

- Por otro lado, para la programación y configuración de la placa **Arduino** se utilizará el entorno de desarrollo **ArduinoIDE**.

En cuanto a las librerías utilizadas para la implementación de la aplicación, podemos destacar las siguientes:

- Librerías de **Firebase** de **Google** [45]. Como pueden ser *firebase-auth*, *firebase-storage*, *firebase-firestores*, ...
- Librería **Camera X** [46]. Se trata de una biblioteca de **Google** creada para que el desarrollo de funcionalidades con cámara resulte más sencillo.
- Librería **ML Kit** [47]. Paquete de funcionalidades de **Google** que, entre otros servicios, proporciona un lector de códigos de tipo Quick Response code (QR) (*Barcode scanning*).
- Librería **Retrofit** [48]. Es un cliente de servidores *REST* para **Android** que nos permite realizar las peticiones a las diferentes APIs.
- Librería **OkHttp** [49]. Nos permite realizar operaciones tanto en HTTP como en SPDY [50] de manera sencilla y eficiente para aplicaciones **Java** y **Android**.
- Librería **Gson** [51]. Es una biblioteca de **Java** que se puede utilizar para convertir tanto objetos **Java** en su representación en JSON, como para convertir una cadena JSON en un objeto **Java** equivalente.
- Librería **Room** [52]. Ofrece una capa de abstracción para poder acceder a la base de datos *Sqlite*.
- Librería **Picasso** [53]. Biblioteca *open-source* para el tratamiento y representación de imágenes en **Android**.
- Librería **GraphView** [54]. Nos permite representar gráficas e histogramas.

Para llevar a cabo un control de todos los cambios y versiones producidas durante todo el ciclo de desarrollo del proyecto hemos utilizado el software de control de versiones **Git** [55]. Para ello, utilizamos el conjunto de extensiones **Git-flow** [56] que nos ayudarán a implementar un ciclo de trabajo basado en las ramas *master*, *develop* y *feature*.

Capítulo 3

Metodología, planificación y estimación de costes

La finalidad principal de aplicar una metodología al desarrollo de un producto software es hacer más eficaz la producción y lograr una alta calidad con el menor coste. Además, nos permite establecer un marco de trabajo y definir una buena planificación del proyecto. Es importante también realizar un análisis y estimación de los costes tanto del proceso de desarrollo como del proyecto en sí, para así poder garantizar la eficiencia, calidad y competitividad.

3.1 Metodología

Para este proyecto se utilizó una metodología **incremental** que complementa y, al mismo tiempo, suple las debilidades del modelo tradicional en cascada. Este modelo incremental nos permite realizar una evolución sostenida del proyecto desde las primeras etapas del mismo. De esta forma, cada incremento da como resultado una versión del producto con una funcionalidad adicional, construyendo así un entorno adecuado para el cumplimiento de los requerimientos que, en ocasiones, pueden ir cambiando a lo largo del tiempo. Esta aproximación tiene dos tipos de enfoque:

- **Cascada o lineal**: Proporciona una visión simple del desarrollo del software, pues representa a los procesos como fases separadas y secuenciales, donde los requerimientos por parte del cliente ya tienen que ser establecidos en las primeras fases del ciclo de vida.
- **Iterativa**: El modelo se repite varias veces a lo largo del desarrollo del proyecto logrando, de esta forma, un sistema más robusto y cercano a las necesidades y expectativas del cliente a medida que se avanza en el sistema.

El modelo en cascada presenta una serie de limitaciones o desventajas con respecto al

modelo incremental. Es muy difícil que un cliente defina explícitamente todos los requisitos del sistema al principio. Además, el cliente no obtendrá ningún resultado visible hasta el final del ciclo de vida del proyecto, por lo que no podrá ir comprobando el desarrollo de este. Debido a estos problemas que presenta el modelo en cascada, se ha decidido seguir un **modelo iterativo incremental** durante el desarrollo del ciclo de vida de este proyecto.

Con respecto a su predecesor, el modelo en cascada, este nuevo modelo incremental ofrece una serie de ventajas:

- Reduce el coste de adaptar nuevos requerimientos o funcionalidades del cliente.
- Se obtiene una retroalimentación inmediata y continua del cliente sobre el trabajo que se está llevando a cabo. Podemos gestionar las expectativas del cliente de manera periódica.
- Es posible que la entrega e implementación de software útil sea más rápida.
- Permite conocer el estado real del proyecto con precisión y, al mismo tiempo, nos permite gestionar la complejidad del mismo, minimizando el número de errores que se pueden generar durante el desarrollo y detectándolos con mayor rapidez.
- En caso de que el cliente no quiera continuar con el proyecto, solo se perderían los recursos de una o varias iteraciones, nunca del proyecto completo.

También podemos identificar una serie de debilidades o desventajas a la hora de utilizar este modelo con respecto a otras metodologías, como pueden ser:

- La disponibilidad del cliente debe de ser alta para permitir así una comunicación fluida con los desarrolladores.
- Requiere de metas claras para conocer en todo momento el estado del proyecto.
- Requiere de mucha planificación, tanto administrativa como técnica.

3.1.1 Fases de la metodología iterativa incremental

Este modelo define una serie de fases o etapas que compondrán el proceso de desarrollo software:

- **Fase de preparación**: En esta fase se produce la aceptación del proyecto, asignación de la fecha de inicio y la fecha de arranque del proyecto. Además, se define la formación y puesta en marcha del equipo de desarrolladores para establecer los objetivos principales.

- **Fase de definición**: Esta fase está formada por todas las actividades orientadas a conocer y definir las funcionalidades necesarias para nuestro proyecto.

- **Fase de prototipado iterativo**: Fase de desarrollo, ajustes, parametrización y customización incremental. Se llevará a cabo la implementación del código que dará funcionalidad a los requisitos previamente definidos. En cada nueva iteración, se desarrollará un versión funcional del producto con un nuevo incremento definido por los requisitos abordados en esa iteración.

- **Fase de preparación final**: Fase definitiva para la puesta en producción del nuevo sistema. Incluye la aceptación y validación del sistema por parte del cliente, además de la realización de distintos tipos de pruebas.

- **Fase de Go Live & Support**: Detección y corrección de incidencias o nuevas casuísticas no detectadas en fases previas, así como soporte y mantenimiento del proyecto.

3.1.2 Aplicación de la metodología al proyecto

Como se ha introducido en la sección anterior, la metodología escogida para este proyecto es el modelo iterativo **incremental**, puesto que es la que mejor se amolda al mismo. Este modelo nos permite una mejor planificación de las actividades involucradas en el ciclo de vida del proyecto, una mejor definición y ajustes de los requisitos a lo largo del desarrollo tanto por parte del cliente como de los desarrolladores y, además, nos ofrece la posibilidad de obtener una versión funcional desde muy pronto para facilitar la integración de las dos partes del sistema.

Tal y como sugiere la metodología incremental, las primeras fases de esta se centran en establecer los objetivos y funcionalidades básicas del proyecto, así como en definir los roles y equipos que formarán parte del mismo. En nuestro caso, este proyecto se ha desarrollado con la colaboración de una empresa, por lo que la parte del cliente viene determinada por un experto de esta en el ámbito de la formación en emergencias. Por otro lado y debido a la naturaleza de este proyecto, la supervisión y dirección del mismo corresponde al tutor (jefe de proyecto), mientras que el equipo de desarrolladores estará únicamente formado por el alumno (desarrollador).

Siguiendo el marco definido por la metodología incremental, se definieron una serie de objetivos y funcionalidades básicas, siempre teniendo en cuenta los requerimientos del cliente. Para esta fase, se llevaron a cabo una serie de reuniones iniciales tanto con el director como con el cliente. A medida que se avanzaba en el trabajo y se iban diseñando e implantando las funcionalidades correspondiente a cada una de las iteraciones, se programaron una serie de reuniones periódicas para asegurar una correcta comunicación entre los principales miembros

del proyecto y así mantener un *feedback* continuo del mismo, pudiendo modificar, refinar o añadir nuevas funcionalidades o requisitos.

En las últimas fases del proyecto se llevaron a cabo, aunque no completamente, las fases de preparación final y *Go Live & Support*. Se realizaron una serie de pruebas o tests en unas condiciones lo más reales posibles para comprobar el correcto funcionamiento del sistema y, así, determinar si era necesario modificar o mejorar ciertos aspectos. No se llevaron a cabo algunos procesos como son la puesta en producción del sistema o la elaboración de un plan de formación y comunicación del mismo.

En resumen, gracias al uso de la metodología incremental, pudimos tener un *feedback* continuo del cliente que nos ayudó a ser conscientes del estado real del proyecto, del grado de aceptación del cliente y permitió un ajuste continuo de los requisitos que se establecieron en un principio. Esto nos ha permitido conocer el estado actual del proyecto en cada momento y nos ha dado la posibilidad de obtener un sistema funcional lo antes posible.

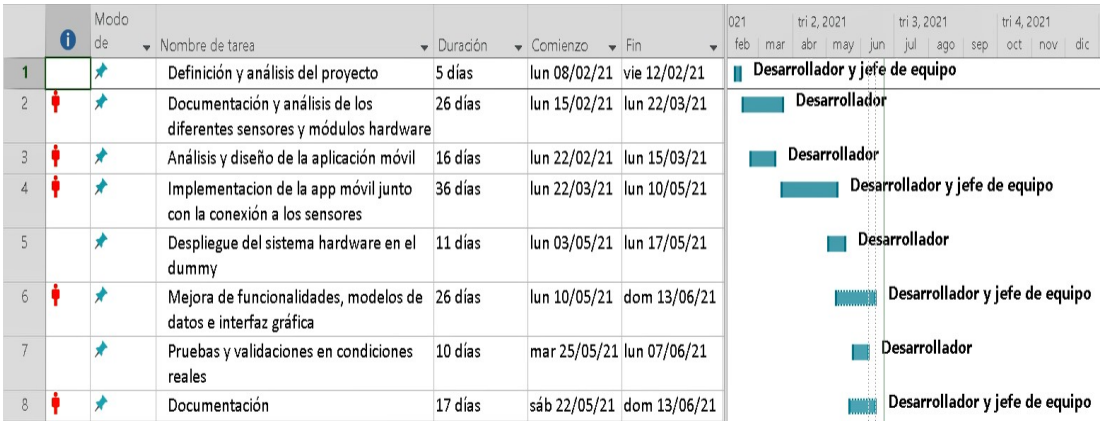
3.2 Planificación inicial

Considerando las directrices establecidas por la **metodología iterativa incremental** y su aplicación en este trabajo, se ha realizado una planificación inicial para el proyecto. En primer lugar, se ha estructurado o dividido el ciclo de vida del proyecto en las siguientes fases o iteraciones:

- Iteración 1: Definición y análisis del problema.
- Iteración 2: Documentación y análisis de los diferentes sensores y módulos hardware.
- Iteración 3: Análisis y diseño de la aplicación móvil.
- Iteración 4: Implementación de la aplicación móvil junto con la conexión a los sensores.
- Iteración 5: Despliegue del sistema hardware en el *dummy*.
- Iteración 6: Mejora de funcionalidades, modelos de datos e interfaz gráfica.
- Iteración 7: Pruebas y validaciones en condiciones reales.

En la imagen 3.1, se muestra el **diagrama de Gantt** con la planificación inicial definida para el proyecto. En él se pueden ver las diferentes tareas o iteraciones a realizar, los recursos humanos asignados a cada iteración y el tiempo empleado para cada una de ellas, en una división por semanas. Como se puede apreciar, la duración total del proyecto va desde el 8 de Febrero hasta el 13 de Junio. Por tanto, se ha estimado que se necesitará un total de 126 días para completar el trabajo. Teniendo en cuenta que el alumno va a trabajar entre 4-5 horas

diarias, nos daría un total aproximado de 630 horas invertidas en el proyecto. Es interesante resaltar que este dato se ajusta razonablemente al número de créditos (24) establecidos para las asignaturas correspondientes al doble Trabajo Fin de Grado (TFG).



	Modo de	Nombre de tarea	Duración	Comienzo	Fin
1		Definición y análisis del proyecto	5 días	lun 08/02/21	vie 12/02/21
2		Documentación y análisis de los diferentes sensores y módulos hardware	26 días	lun 15/02/21	lun 22/03/21
3		Análisis y diseño de la aplicación móvil	16 días	lun 22/02/21	lun 15/03/21
4		Implementacion de la app móvil junto con la conexión a los sensores	36 días	lun 22/03/21	lun 10/05/21
5		Despliegue del sistema hardware en el dummy	11 días	lun 03/05/21	lun 17/05/21
6		Mejora de funcionalidades, modelos de datos e interfaz gráfica	26 días	lun 10/05/21	dom 13/06/21
7		Pruebas y validaciones en condiciones reales	10 días	mar 25/05/21	lun 07/06/21
8		Documentación	17 días	sáb 22/05/21	dom 13/06/21

Figura 3.1: Diagrama Gantt dividido por semanas.

3.2.1 Seguimiento de la planificación

Después de haber completado el ciclo de desarrollo, es necesario comentar que algunas fases o iteraciones sufrieron algún retraso. La fase de **implementación de la aplicación móvil junto con la conexión a los sensores** llevó un tiempo total de 38 días, con respecto a los 36 que estaban preestablecidos. Esto fue debido a una serie de dificultades no previstas en la conexión entre la aplicación móvil y el *dummy* a través de BLE. También hubo un pequeño retraso con respecto al **despliegue del sistema hardware en el dummy** puesto que, durante los primeros días de esta iteración, carecíamos del maniquí por lo que su despliegue tuvo que esperar y comenzó 2 días después de lo establecido.

A pesar de estos retrasos, se consiguió finalizar el proyecto 6 días antes de la entrega final (23 de Junio), con lo cual se acumuló únicamente un retraso de 4 días respecto a lo planificado inicialmente. Por tanto, el margen establecido en la planificación inicial de 10 días fue suficiente y, todavía, hubo cierto margen al final para pulir pequeños detalles relacionados con la documentación del proyecto.

3.3 Estimación de costes

Para la estimación de los costes totales del proyecto, podemos dividirlos en tres grandes grupos: **humanos**, **software** y **hardware**. Estos costes están directamente relacionados con los tipos de recursos necesarios para llevar a cabo con éxito cualquier proyecto de este tipo.

3.3.1 Costes humanos

Dentro de los recursos humanos podemos encontrar dos roles claramente diferenciados: el equipo de desarrollo (el alumno) y el jefe de proyecto o director (tutor del TFG).

Como se comentó en la sección de planificación inicial, se ha estimado que el alumno necesitará emplear aproximadamente unas 630 horas en la realización del proyecto completo. Según el convenio vigente para desarrolladores de software [57], se define un salario bruto anual de 22.993,74 € y un máximo total de horas anuales trabajadas de 1800, por lo que a la hora cobraría 12,77 €/h. Por tanto, podemos calcular lo que debería cobrar el alumno por la realización del proyecto simplemente realizando la siguiente multiplicación: 12,77 € x 630 h = 8045,1 €.

En cuanto al trabajo realizado por el jefe de proyecto o director, este se resume en un total de 8 reuniones de proyecto: una reunión inicial en la que se establecieron los objetivos y características básicas del proyecto, otras seis correspondientes a las iteraciones principales del ciclo de vida y, por último, una revisión final. Estas reuniones tuvieron una duración aproximada de 2 horas, lo que hace un total de 16 horas de trabajo. De esta forma, a partir del salario mínimo de un jefe de proyecto, que asumimos que es aproximadamente 30.000 €, pues no se encontraron referencias explícitas a dicho dato, podemos establecer que debería de cobrar un total de 16 € x 16 h = 256 €.

Por lo tanto, el coste total estimado para los recursos humanos sería de **256 € + 8045,1 € = 8301.1 €**. Sin embargo, como se comentó en la sección de seguimiento del proyecto, ocurrieron una serie de retrasos que ocasionaron que el coste real de los recursos humanos aumentara ligeramente, como se puede apreciar en la tabla 3.1.

	Estimado	**Real**
Fecha inicio	8-02-2021	8-02-2021
Fecha fin	13-06-2021	17-06-2021
Horas totales (desarrollo + reuniones)	630 horas	650 horas
Coste total recursos humanos	8301.1 €	**8555.5 €**

Cuadro 3.1: Coste total recursos humanos

3.3.2 Costes software

Como se comentó en el capítulo 2, para la realización de este proyecto se utilizaron diferentes plataformas o APIs. La utilización de algunas de ellas pueden generar una serie de gastos que debemos tener en cuenta para la estimación de costes.

En primer lugar, tenemos la API correspondiente a **Sigfox**. Para poder utilizar esta red de comunicaciones es necesario poseer una subscripción, la cual tiene una duración de un año.

Dependiendo de la domiciliación de nuestra compañía, el precio puede variar. En el caso de este proyecto, al hacerlo desde España, el importe es de 16.13 € al año.

Otra plataforma utilizada fue **Firebase** que nos ofrece dos tipos de planes de uso: *spark* y *blaze*. En este proyecto, se utilizó el primero de ellos pues es un plan gratuito que tiene una serie de recursos disponibles como 1GB de almacenamiento de datos, 20.000 escrituras por día, 50.000 lecturas por día y 20.000 eliminaciones por día. Por lo tanto, para una primera implementación y despliegue del sistema, estos recursos son más que suficientes. Por otro lado, en este proyecto se utilizaron también las *Cloud Functions* de **Firebase** que el plan *spark* no incluye entre los recursos gratuitos. Sin embargo, **Firebase** sí ofrece gratis 2M/mes de invocaciones, 400.000 GB/segundo, 200.000 GHz-segundo de tiempo de procesamiento y 5 GB de tráfico de salida de Internet al mes, por lo que es más que suficiente para la realización de este proyecto. De esta forma, podemos asumir que el coste asociado al uso de las funcionalidades de **Firebase** durante la realización del proyecto es cero.

Por último, también se utilizaron algunas funcionalidades de **Google**, como son las APIs de *Google Maps SDK for Android* y la API de *Geolocation*. La primera de ellas no tiene ningún coste asociado, mientras que la segunda tiene un coste que varía dependiendo del volumen de peticiones mensuales. Si tenemos un número de solicitudes entre 0-100.000, el precio es de 5$ cada 1.000 mientras que, si nos movemos entre 100.001-500.000 de solicitudes al mes, el precio es de 4$ cada 1.000 solicitudes.

Teniendo en cuenta estos datos, para calcular el coste de los recursos software en este proyecto, al no realizarlo para una mayor producción, solo es necesario tener en cuenta el importe de la licencia de **Sigfox**, es decir, un total de **16.13 €**.

3.3.3 Costes hardware

En este trabajo, la componente hardware del sistema representa una parte muy importante del proyecto ya que es necesario montar todo el sistema de monitorización del *dummy*.

Aunque no es un coste hardware como tal, lo primero que vamos a mencionar es el propio *dummy*. Existen multitud de alternativas en cuanto a maniquíes de rescate, pero los rangos de precios oscilan siempre entre los 900 €-2.500 €. En el caso de este proyecto, se utilizó un *dummy* de la compañía *MaxPreven*, adulto de 70 Kg con un precio de 1.038 €.

En cuanto al resto de hardware utilizado, la tabla 3.2 muestra los costes asociados a cada uno de los recursos empleados en el proyecto.

En esta tabla, no se incluyen gastos como la amortización del ordenador en el que fue desarrollado todo el proyecto, ni el coste de los dispositivos móviles empleados para los diferentes tests, de la corriente, de la conexión a internet o, por ejemplo, de los desplazamientos a la montaña a realizar las diferentes pruebas. De esta forma, considerando el coste total de todo el hardware necesario más el coste real del maniquí, obtenemos un total de 1.038 € +

Producto	Cantidad	Precio unitario	Importe
Arduino Mkr Wifi 1010	1	27,90€	27,90€
Módulo Sigfox BRKWS01-RC1 + 868Mhz Antenna	1	23,88€	23,88€
Módulo GPS GT-U7	1	13,88€	13,88€
Sensor presión SF15-150	3	11,59€	34,77€
Altavoces CQRobot 5 Watt 8 Ohm	1	7,99€	7,99€
Amplificador LM386	1	1,20€	1,20€
Sensor temperatura dht11	1	0,45€	0,45€
Sensor gy-521	1	2,95€	2,95€
Cables de conexión	1	2,99€	2,99€
Batería 2000 mah	1	8,99€	8,99€
Placa de desarrollo	1	0,5€	0,5€
Importe total			125,59€

Cuadro 3.2: Estimación de los costes hardware para el proyecto

125,59 € = **1163.59 €.**

3.3.4 Costes totales

Después de analizar los gastos por separado para cada recurso, podemos unificarlos todos en único valor que representa el coste total para desarrollar el proyecto. La tabla 3.3 resume todos los costes asociados al proyecto, tanto los estimados como los reales tras completar la implementación del sistema.

Tipo de coste	Coste Estimado	Coste real
Coste hardware	1163.59 €	1163.59 €
Coste Software	16.13 €	16.13 €
Coste del desarrollador	8045,1 €	8300,5 €
Coste del director	256 €	256 €
Importe total	9.480,82 €	**9.736,22 €**

Cuadro 3.3: Coste total del proyecto

Capítulo 4

Visión general del proyecto

PARA entender mejor toda la estructura y funcionalidades del proyecto, vamos a recapitularlas y explicarlas en este capítulo, donde se ofrecerá una visión general de todo el sistema desarrollado.

4.1 Estructura del sistema

Para tener una visión conjunta de todo el proyecto, podemos hacer uso del esquema presentado en la figura 4.1, en el que se muestran todos los componentes principales del proyecto y la interconexión e intercambio de información que se produce entre ellos.

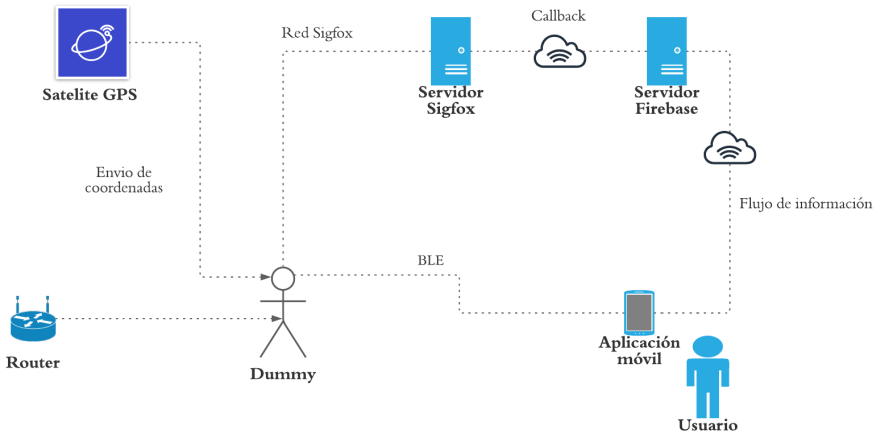


Figura 4.1: Diagrama general del proyecto

Las entidades centrales de este proyecto son el propio *dummy*, encargado de recoger y transmitir su información de monitorización y la **aplicación móvil** encargada de visualizar todos estos datos de una forma adecuada y que pueda resultar útil para el usuario durante su formación. El *dummy* se conecta y relaciona con la red de satélites GPS para poder determinar su posición, aunque también puede escanear las redes *WiFi* más cercanas para obtener las

dos MAC más próximas y hacer así una predicción de la posición si el módulo GPS falla. El maniquí también se interconecta con la red **Sigfox** para la transmisión de los datos de posición hacia el servidor en **Firebase**. En esta comunicación, se hace uso del servicio gratuito **Sigfox Atlas** [58], el cual también estima, aunque de manera menos precisa, la posible ubicación del *dummy*. Por último, el *dummy* se comunica también con el dispositivo móvil a través del protocolo BLE para un intercambio bidireccional de datos.

La aplicación móvil, a su vez, se comunica tanto con el *dummy* como con **Firebase**. Como se ha introducido, esta plataforma permite a la aplicación una correcta gestión de usuarios y proporciona una base datos *online* para almacenar distintos tipos de datos.

Por último, la red **Sigfox** establece la comunicación con **Firebase** a través de su sistema de *callbacks*, permitiendo así el almacenamiento de los diferentes mensajes de posición enviados desde un *dummy*. Los datos de posición recibidos desde **Sigfox** son procesados en el servidor de **Firebase** para ubicarlos en la base de datos *online* correspondiente, dependiendo del dispositivo emisor, y gracias al uso de las *Cloud Functions*. Este procesamiento adicional es importante para asociar en la base de datos las coordenadas recibidas con el *dummy* correspondiente que ha enviado esos datos de posición.

A continuación, se van a presentar aquellos detalles más relevantes de las dos partes del proyecto: el sistema hardware y la aplicación móvil.

4.2 Información proporcionada por el *dummy*

El *smart dummy* ofrece una serie de información que es importante monitorizar a la hora de realizar cualquier práctica o rescate en el ámbito sanitario:

- **Posición del *dummy***. Gracias a la utilización del módulo GPS, mediante al escaneo de dos MAC en una red *WiFi* cercana por parte de la placa **Arduino Mkr Wifi 1010** o mediante el uso del servicio **Sigfox Atlas**. Esta información de posicionamiento se visualizará en la aplicación móvil con la API de **Google Maps**.
- **Inclinación del *dummy*** en los dos ejes x e y. Debido a esto, obtenemos la información de como se está tratando el cuerpo del paciente, y si se está llevando a cabo correctamente la realización de las distintas técnicas de rescate.
- **Aceleración del *dummy*** en los ejes x e y. Podemos saber si el maniquí se está moviendo de forma demasiado brusca o con demasiada rapidez, lo que puede provocar que agravemos la situación del paciente.
- **Temperatura y humedad**. Gracias a controlar y medir la humedad y temperatura de un paciente en el momento de un rescate o inmovilización, podremos determinar el estado de este y actuar en consonancia con ello.

- **Fuerza o presión ejercida** tanto en el cuello, como en el muslo y en el brazo. Esa información permite conocer cuanta presión está aplicando un collarín, un torniquete, una venda o una férula de vacío, siempre en un rango que va desde 1 kg hasta los 10 kg.
- **Sonido y alertas.** Reproducción automática de alertas para informar de si se está realizando alguna de las técnicas de manera incorrecta. Posibilidad de reproducción de sonidos preestablecidos por parte del usuario.

4.3 Funcionalidades aplicación móvil

Teniendo en cuenta el ámbito considerado, podemos establecer la siguiente lista con las principales funcionalidades que debería ofrecer la aplicación móvil:

- **Registro y acceso de usuarios.** Podemos distinguir dos tipos de usuarios:
  - Usuario **normal**: puede utilizar las principales funcionalidades de la aplicación, como pueden ser la conexión con el *dummy*, monitorización del *dummy*, ..., pero no establecer umbrales para los sensores ni modificar el rango de búsqueda. Este usuario puede estar ligado a un administrador, para utilizar las preferencias y umbrales establecidos por este. Este rol corresponde a los estudiantes/personas que participan en una formación o prácticas de actividades de rescate.
  - **Administrador**: ”dirige” a un grupo de usuarios normales y, además de poder acceder a las funcionalidades de un usuario normal, también puede establecer diferentes umbrales o niveles para los diferentes sensores del *dummy*. Esta utilidad le permite modificar las alertas por malas prácticas o modificar el rango de búsqueda en una simulación de rescate.
- **Visualización de todos los datos ofrecidos por el *dummy*.** Posibilidad de ver estos datos individualmente o todos juntos en una pantalla de monitorización completa, ofreciendo además la posibilidad de visualización de manera gráfica y/o numérica.
- **Escaneo y conexión con el *dummy* a partir del protocolo BLE.** El usuario puede ver directamente desde la aplicación los dispositivos BLE disponibles y establecer la conexión o desconexión con estos.
- **Geolocalización del *dummy*.** Esta información puede proceder de los datos devueltos por el módulo de GPS, por la estimación de posición ofrecida por **Sigfox Atlas** o gracias a la API de geolocalización de **Google** a partir de dos MAC. Además, se puede ver un registro de las últimas posiciones del *dummy*, filtrar u ordenar las mismas, así como iniciar la navegación desde tu posición hasta la del *dummy* gracias a **Google Maps**.

- **Visualización de documentación.** Esta información muestra los procedimientos asociados a las técnicas básicas de primeros auxilios. Esta documentación se guarda en una base de datos local.

- **Iniciar simulación de búsqueda.** Una de las funcionalidades más interesantes del proyecto se centra en las prácticas de rescate en alta montaña. El sistema ofrece la posibilidad de enviar periódicamente su información de posición a través de la red **Sigfox**, obteniéndola a partir de los métodos explicados en el punto de **geolocalización del dummy**.

  Podemos ver en la figura 4.2 un diagrama de flujo para el sistema y los diferentes componentes que intervienen en esta funcionalidad. Se parte de una situación en la que la aplicación móvil se encuentra conectada con el *dummy* mediante BLE y se está produciendo un intercambio de datos de monitorización. El usuario en este punto puede iniciar una simulación de rescate y, mediante un *trigger* específico, el sistema avisa al *dummy* para que comience la trasmisión de datos de geolocalización.

  Estos datos son enviados en forma de mensaje a través de la red **Sigfox** a sus servidores desde los cuales, mediante un servicio de *callbacks*, son enviados al servidor de **Firebase**. Este servidor almacena los mensajes que recibe en una base de datos especifica dependiendo del identificador del **dummy** que lo envía. El usuario puede visualizar estos datos desde la aplicación móvil, accediendo para ello a la base de datos *online* de **Firebase**. El resultado de todo este proceso se muestra en un mapa gracias al SDK de **Google Maps**.

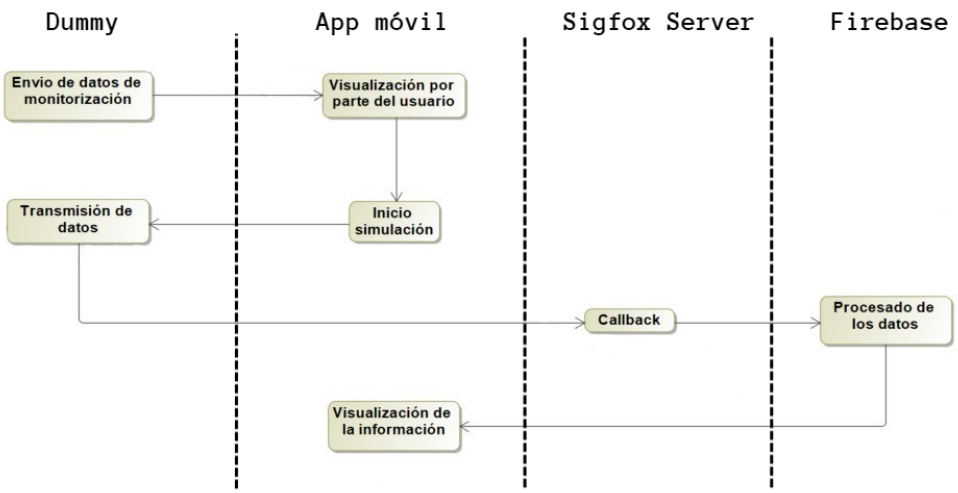


Figura 4.2: Diagrama de actividad para una simulación de rescate en montaña.

Capítulo 5

Smart dummy y sistema hardware

PARA la monitorización y envío de todos los datos por parte del sistema hardware es necesario la integración y configuración de diferentes componentes hardware sobre el propio cuerpo del *dummy*. En una primera aproximación, se fueron probando y configurando todos estos componentes de una forma individual para luego proceder con la integración de todos los elementos en el montaje final.

5.1 Componentes hardware

En esta primera sección, vamos a analizar los componentes y módulos hardware necesarios para la implementación de las principales funcionalidades de esta parte del proyecto.

5.1.1 Monitorización de la temperatura y humedad

Controlar la temperatura y la humedad a la que se encuentra un paciente es una de las primeras acciones que tenemos que realizar. Una temperatura muy baja puede derivar en hipotermia y una excesiva ser sinónimo de otro tipo de enfermedad o de una infección. Por el contrario, una humedad excesivamente baja puede provocar problemas en el sistema respiratorio.

Por esta razón, se ha integrado en el *dummy* un sensor de temperatura y humedad *DHT11*, basado en la tecnología NTC. Este sensor se ha conectado a la placa **Arduino Mkr Wifi 1010** utilizada para recoger, procesar y proporcionar los datos de sensorización. El medidor de temperatura y humedad *DHT11* puede encontrarse en dos formatos distintos: en forma de módulo y en forma de sensor. La diferencia entre uno y otro es que, en el primer caso, incorpora un condensador de filtrado y una resistencia, mientras que en la opción del sensor es necesario añadirlos manualmente para su correcto funcionamiento. En nuestro caso, hemos elegido el uso de la variante en forma de módulo para una mayor comodidad. Este módulo incorpora 3 pines, de los cuales uno es el pin de datos que se utiliza para enviar el valor de la

temperatura y humedad como datos en serie. Para acceder a ese valor desde la placa Arduino, existen diversas librerías que permiten leer estos datos directamente. En nuestro caso, hemos utilizado la librería *DHT.h* [59].

En la tabla 5.1, se detallan las principales características físicas del módulo seleccionado.

Sensor temperatura/humedad DHT11	
Alimentación	3.5V a 5.5V
Consumo en funcionamiento	0.3mA
Consumo en *standby*	60uA
Rango de temperatura	0ºC a 50ºC
Rango de Humedad	20% a 90%
Resolución	16-bit
Precisión	± 1ºC y ± 1%

Cuadro 5.1: Características del sensor DHT11

5.1.2 Control de la inclinación y aceleración

Si al realizar una inmovilización o traslado de un paciente, estas acciones se realizan demasiado rápido o en posiciones incorrectas, pueden provocar que se agraven sus lesiones o que se produzcan otras nuevas. Para monitorizar esta información necesitamos incorporar el módulo *GY-521*. Este módulo está formado por el micro-electromechanical system (MEMS) MPU-6050 que incorpora un giroscopio de 3-ejes, un acelerómetro de 3-ejes, un sensor de temperatura y un procesador de tipo Digital Motion Processor (DMP), que procesa algoritmos complejos directamente en el módulo. Este procesador es capaz de convertir los valores brutos de los sensores en datos de posición estables. Los valores obtenidos son devueltos por el módulo a través de un bus de datos en serie (I2C), que requiere dos cables de conexión con nuestra placa de desarrollo: Serial Data (SDA) y Serial Clock (SCL).

Del mismo modo que en el caso anterior, existen librerías para **Arduino** que nos permiten comunicarnos con este módulo de una forma sencilla. En nuestro caso, utilizamos la librería *Wire.h* [60] que se encarga de la comunicación I2C.

Podemos ver las características principales del módulo utilizado en la tabla 5.2.

Acelerómetro-giroscopio GY-521	
Alimentación	4.3V a 9V
Modo de comunicación	I2C
Rango de giroscopio	+/-250, +/-500, +/-1000, +/-2000 º/s
Rango del acelerómetro	+/-2g, +/-4g, +/-8g, +/-16g
Dimensiones	21.2x16.4x3.3 mm
Peso	2.1g

Cuadro 5.2: Características del acelerómetro-giroscopio GY-521

5.1.3 Control de fuerza o presión

Diversas zonas del cuerpo de una persona son más susceptibles de recibir la colocación de un torniquete, venda o férula para el tratamiento de hemorragias, heridas o fracturas. Estas zonas suelen ser los brazos, los muslos o la parte baja de las piernas. Por eso, se ha decidido colocar 2 sensores de presión en el *dummy*, uno en el muslo, y otro en la parte superior del brazo derecho para poder monitorizar la presión que se ejerce sobre estas zonas. A mayores, una de las zonas más importantes del cuerpo a la hora de realizar una inmovilización es el cuello pues es importante evitar daños neurológicos y físicos en la columna vertebral. Es por ello que en esta parte también se ha colocado un sensor de presión para poder medir la fuerza que aplica la colocación de un collarín.

En todas estas zonas, se colocaron sensores de presión de película resistiva *RP-L-110*. Básicamente, su funcionamiento es parecido al de un resistor, es decir, su valor resistivo va a variar en función de la presión ejercida. Cuando no existe presión, el sensor actúa como una resistencia infinita (circuito abierto), mientras que cuanta más presión ejercemos sobre el sensor menor será la resistencia que este produce.

La forma más común para conectar este tipo de sensores con un microcontrolador es hacerlo a través de un resistencia, para así lograr crear un divisor de tensión. Este circuito seguiría el esquema que se presenta en la figura 5.1.

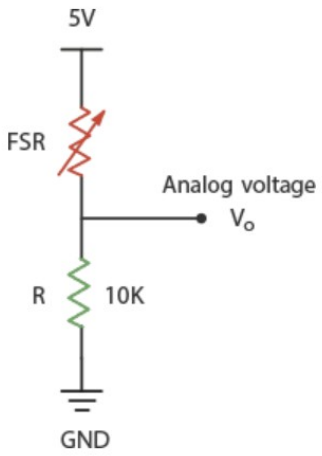


Figura 5.1: Esquema de conexión de un sensor FSR

Como se puede apreciar en la figura, necesitamos alimentar nuestro sensor con V_{cc} = 5 V y colocar una resistencia con un valor adecuado. En ese caso, el voltaje de salida (V_o) dependerá de la presión que se ejerza sobre el sensor ya que el nivel de resistencia FSR varía en función de esa presión. Este voltaje de salida será leído por nuestro **Arduino MKR Wifi 1010** a través

de un pin analógico, y su valor vendrá dado por la siguiente función:

$$V_o = \frac{V_{\text{cc}} R}{R + \text{FSR}}.$$

De esta forma, si no aplicamos ninguna presión, la salida será cercana a 0 V mientras que, si ejercemos la máxima presión, la salida será próxima a los 5 V. Normalmente, el valor de la resistencia que se utiliza en estos casos es del orden de 10 KΩ.

La tabla 5.3 muestra el valor de voltaje analógico aproximado en función de la fuerza/ resistencia que se aplica sobre el sensor con una alimentación de 5 V y una resistencia de 10 KΩ.

Fuerza (kg)	**Resistencia FSR**	**Voltaje de salida**
0 kg	infinita	0 V
0.018 kg	30 KΩ	1.3 V
0.1 kg	6 KΩ	3.1 V
1 kg	1 KΩ	4.5 V
10 kg	250 Ω	4.9 V

Cuadro 5.3: Relación entre la fuerza ejercida, resistencia FSR y voltaje de salida

Por último, podemos ver las características más importantes de este modelo de sensor FSR en la tabla 5.4.

Sensor FSR RP-L-110	
Rango de detección	20g a 10kg
Longitud	11.5cm
Espesor	0.35mm
Tiempo de respuesta	<10ms

Cuadro 5.4: Características de sensor FSR RP-L-110

5.1.4 Sonido en el dummy

Poder reproducir diferentes sonidos como sirenas u otro tipo de alertas es una función muy útil cuando se están realizando prácticas de rescate. Avisar mediante un sonido preestablecido cuando se está realizando incorrectamente alguna práctica o movimiento sobre el maniquí también resulta muy atractivo y ayuda a los alumnos a la hora de realizar las prácticas. A mayores, se puede notificar al usuario mediante la reproducción de un sonido característico, cuando el dispositivo móvil se conecta-desconecta con el *dummy*. Por esta razón, se han incorporado en el *dummy* dos altavoces de 5 Watt - 8 Ohm de la marca *CQRobot* [61]. Además, debido a la baja potencia que es capaz de suministrar la placa de Arduino, es necesario la

utilización de un amplificador *EK1236* para aumentar el volumen de sonido que emiten los altavoces.

Las características más importantes de estos dos componentes se pueden consultar en las tablas 5.5 y 5.6.

Altavoces CQRLB8O5W-B	
Impedancia	8 Ohm
Potencia nominal	5.0 W
Distorsión	10%
Tamaño	70x31x16 mm
Peso	24g

Cuadro 5.5: Características de los altavoces CQRLB8O5W-B

Amplificador EK1236	
Voltaje de trabajo	5 a 12 V
Resistencia	10 KΩ ajustable
Tamaño	8x7x6 cm
Peso	25g

Cuadro 5.6: Características del amplificador EK1236

Estos sonidos se pueden reproducir directamente usando la función *tone()* de Arduino, la cual genera una onda cuadrada con una frecuencia especifica. De esta forma, se ha creado un vector con diferentes frecuencias (que corresponderían a cada una de las notas) y una duración determinada para cada una de ellas. A partir de estos dos parámetros, podemos formar cualquier tipo de sonido o melodía.

5.1.5 Conexión con Sigfox

Para poder realizar el envío de datos de forma remota, como es en el caso de los mensajes de posición del *dummy*, necesitamos estar conectados a alguna red para habilitar la comunicación. Como se comentó en la sección 2.1.1, **Sigfox** es la tecnología que mejor se amolda a nuestro proyecto debido a su facilidad de uso, su rango de transmisión, su bajo consumo y a que no es necesario incorporar un elemento extra como es un *gateway*. Existen multitud de opciones a la hora de elegir hardware que implemente esta tecnología pero, en nuestro caso, hemos elegido el módulo *BRKWS01-RC1*, debido a su sencillez y bajo coste. Este módulo incluye una antena que radia en la banda de 868 Mhz y se acompaña de un año de licencia para utilizar la red **Sigfox**, con un máximo de 140 mensajes de subida al día.

El módulo se controla mediante el uso de comandos ATT en serie enviados a través de los pines RX/TX de la placa Arduino. Los más importantes son *AT$I=10* para obtener el identifi-

cador del módulo, *AT$I=11* para obtener el código PAC y *AT$SF* seguido del mensaje a enviar de 8 bits.

Las características principales de este módulo se encuentran resumidas en la tabla 5.7.

Módulo BRKWS01-RC1	
Voltaje de trabajo	3 V
Comunicación	Serial UART
Protocolo	Comandos ATT
Comunicación Serial	9600 bauds
Dimensiones	23.3 x 21.3 mm

Cuadro 5.7: Características del módulo Sigfox BRKWS01-RC1

5.1.6 Posicionamiento GPS

Como comentábamos en el capítulo anterior, conocer la posición relativa de una persona en una búsqueda de rescate es uno de los factores más importantes y a tener en cuenta. Para conseguir esto, una de las posibilidades que tenemos es obtener la posición través de la tecnología GPS.

Para ello, se ha usado el módulo *GT-U7* que incluye una antena cerámica con interfaz *IPEX*. De la misma manera que en el módulo de **Sigfox**, este se conecta a través del puerto UART con los pines RX/TX a nuestra placa **Arduino MKR Wifi 1010**. Además, se necesita hacer uso de la librería *Tiny.gps* [62] para utilizar y visualizar de una manera más sencilla los datos obtenidos a través del protocolo NMEA.

Este módulo de GPS presenta las características que se recogen en la tabla 5.8.

Módulo GT-U7	
Voltaje	3.6 - 5 V
Velocidad de transmisión	9600 baudios
Protocolo entrada/salida	NMEA
Dimensiones	27.6 x 26.6 mm

Cuadro 5.8: Características módulo GPS GT-U7

5.1.7 Alternativas al posicionamiento GPS

Cuando nos encontramos en lugares urbanos con edificios grandes o cuando un objeto o edificación representa un obstáculo que impide que la señal GPS llegue con suficiente potencia hasta el dispositivo, podemos encontrarnos en la situación de que el módulo GPS no consigue obtener una ubicación precisa.

Por esa razón, se han valorado y utilizado otras alternativas para conseguir dicha posición en estas situaciones:

- **Utilizar la funcionalidad *Atlas Native* de Sigfox**. Simplemente a través de la red **Sigfox** y usando lo que ellos llaman un **Geomodule**, el módulo de **Sigfox** es capaz de estimar la posición de un emisor con un margen de error de alrededor de 500 metros. Esta posición es calculada e incluida automáticamente en cada mensaje enviado.

- **Utilización de la API de geolocalización de Google Maps**. Esta API permite estimar la ubicación a partir de las MAC de dos *routers* de redes *WiFi* cercanas. Por tanto, el hardware del *dummy* puede escanear las redes cercanas y enviar esta información a través de la red **Sigfox** con el objetivo de habilitar esta estrategia de posicionamiento.

  La placa **Arduino Mkr Wifi 1010** tiene conectividad *WiFi* por lo que no tenemos la necesidad de incluir ningún módulo a mayores para escanear las dos MAC más cercanas. Esta operación de escaneo se puede realizar en **Arduino** de la siguiente manera:

```
    Serial.println("** Scan Networks **");
    int numSsid = WiFi.scanNetworks();
    uint8_t msg1[12];

    // print the list of networks seen:
    Serial.print("number of available networks:");
    Serial.println(numSsid);

    byte bssid[6];

    int j= 0;
    for (int i = 5; i >= 0; i--) {
        msg1[j]=(WiFi.BSSID(0, bssid))[i];
        j++;
    }

    j=0;
    for (int i = 5; i >= 0; i--) {
        msg1[j+6]=(WiFi.BSSID(1, bssid))[i];
        j++;
    }

```

5.1.8 Arduino Mkr Wifi 1010

Para obtener los datos proporcionados por los diferentes sensores, acceder de forma cómoda a ellos y enviarlos para su visualización, necesitamos incorporar una placa de desarrollo

que integre y optimice todos estas funciones. La placa elegida es una evolución de la **MKR 1000 Wifi**, la cual está formada por un microcontrolador **ARM Cortex** y equipada con un módulo *ESP32* desarrollado por la compañía **U-BLOX** [63]. Esta placa de **Arduino** nos proporciona además conectividad *WiFi* y *Bluetooth*. Otra característica interesante es que necesita un consumo bajo de energía, a mayores de ofrecer la posibilidad de alimentar la tarjeta a través de una batería de Lithium Polymer Battery (LiPo) que será cargada de forma automática mientras el **Arduino MKR WiFi 1010** reciba alimentación por el puerto USB.

Esta placa incorpora también una gran variedad de pines para la conexión con los diferentes sensores y módulos hardware. Entre ellos, dispone de 8 entradas/salidas digitales, 12 canales de PWM, 7 entradas analógicas y 1 salida analógica. Además, proporciona 256 Kb de memoria *flash* para programas y 32 Kb de memoria RAM.

Son por todas estas características que esta placa es la opción predilecta y más utilizada para la realización de aplicaciones de IoT. En nuestro proyecto, además de las características vistas, nos proporciona la capacidad necesaria para la comunicación BLE con la aplicación móvil, por lo que la placa de **Arduino Mkr Wifi 1010** nos resulta una opción ideal para la creación del *smart dummy*.

5.2 Dummy

Esta clase de maniquíes buscan simular con el mayor realismo posible situaciones extremas o de emergencia que puede sufrir un paciente real. Existen multitud de clases y de tipos de *dummies*, desde los creados específicamente para rescate acuático, hasta los pensados para situaciones con fuego pero, en nuestro caso, se trata de uno de uso general que simula a una persona inconsciente, es decir, la espalda no se mantiene recta. Este maniquí se denomina *maniquí de uso general* y es de la marca **Maxpreven**.

Presenta una altura de 1,80 metros y un peso de 70 kg repartidos por todo el cuerpo del muñeco en forma de sacos o cilindros de arena y recubierto por una funda construida por el mismo material utilizado en los chalecos balísticos/punzantes de la policía.

En este proyecto, vamos a convertir este *dummy* básico en un *smart dummy* mediante todo el hardware de sensorización y posicionamiento que se ha presentado en la sección anterior 5.1, y con el que podemos comunicarnos e interaccionar a través de los elementos que se van a explicar en las siguientes secciones.

5.3 Comunicación mediante Sigfox

Como se ha comentado anteriormente, la conexión del módulo **Sigfox** se realiza mediante los pines RX/TX a nuestro **Arduino Mkr Wifi 1010**. Para que el envío de mensajes requiera de poca energía, estos mensajes están diseñados para ser muy pequeños y optimizados para

ser utilizados con sensores, y es por eso que el *payload* está limitado a un máximo de 12 bytes, excluyendo las cabeceras. **Sigfox** soporta comunicación bidireccional, es decir, podemos enviar mensajes *uplink* desde el módulo al servidor de **Sigfox**, o solicitar datos a este servidor, mensajes *downlink*. Para solicitar este tipo de mensajes, es el dispositivo **Sigfox** quien tiene que iniciar la comunicación con un mensaje *uplink* y esperar aproximadamente 30 segundos para recibir un mensaje *downlink* con un *payload* de 8 bytes. Este tipo de mensajes *downlink* están limitados a 4 mensajes al día por la regulación ETSI [64], mientras que los mensajes *uplink* como ya comentamos anteriormente son 140 al día, que resultan más que suficientes para nuestro propósito.

En nuestro caso, es necesario llevar a cabo dos pasos para poder acceder a los datos de posicionamiento enviados desde el módulo **Sigfox** en el *dummy*: enviar los mensajes desde el módulo al servidor y acceder a esos mensajes en el lado del servidor. Un mensaje *uplink* tarda en enviarse aproximadamente 6 segundos, a una velocidad de 100 bits/seg, y es enviado 3 veces distintas en 3 bandas de 100 Hz, seleccionadas de forma aleatoria del canal principal. Luego, el servidor **Sigfox** se encarga de eliminar estos duplicados. Para enviar estos mensajes desde el **Arduino Mkr Wifi 1010** conectado a través de los pines TX/RX, tenemos que utilizar la función *Serial1.println()* con el mensaje a enviar, precedido por una cabecera ATT de la forma *AT$SF=*. Se puede ver un ejemplo del envío de mensajes a través de la red **Sigfox** con la placa de desarrollo **Arduino Mkr Wifi 1010** en el anexo A.

Por otro lado, podemos acceder al servidor al que se envían todos los datos a través de la siguiente URL: https://backend.sigfox.com/. Este *backend* almacena todos los datos enviados por los dispositivos registrados para un usuario, permite su visualización y la creación de *callbacks* para redirigir esos mensajes a servidores o aplicaciones externas del usuario. En la imagen 5.2, podemos ver un ejemplo con la lista de mensajes recibidos, cada uno con su *payload* correspondiente en formato hexadecimal. Además, nos ofrece la posibilidad de acceder a la posición estimada del dispositivo que envió cada mensaje, calculada mediante el servicio **Sigfox Atlas**.

Podemos observar además que, para cada mensaje, se generó un *callback* que, en nuestro caso, permite reenviar los mensajes a nuestra base de datos en **Firebase**. Estos *callbacks* se pueden asociar a un dispositivo en concreto cuando se crean para actuar sobre todos los mensajes que se reciban procedentes de ese dispositivo.

En la imagen 5.3, podemos ver un ejemplo detallado de un *callback* creado en el proyecto. En él, se puede observar como se trata de una operación *POST*, a la que le pasamos la URL de la base de datos *Realtime* de **Firebase** y una serie de datos proporcionados directamente por **Sigfox**, así como el *payload* del mensaje recibido desde dispositivo.

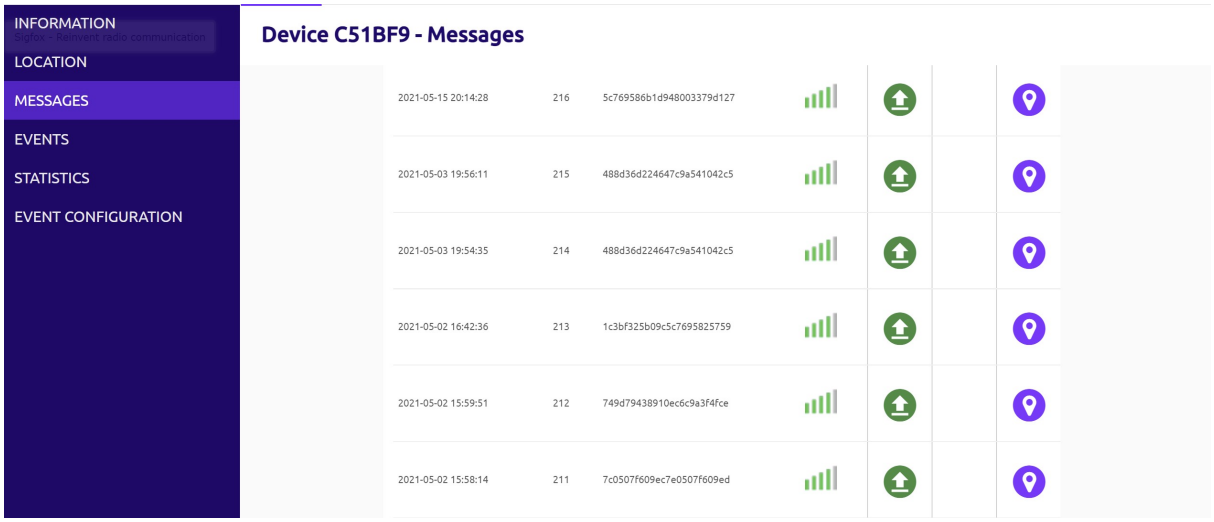


Figura 5.2: Lista de mensajes recibidos en el *Sigfox backend*

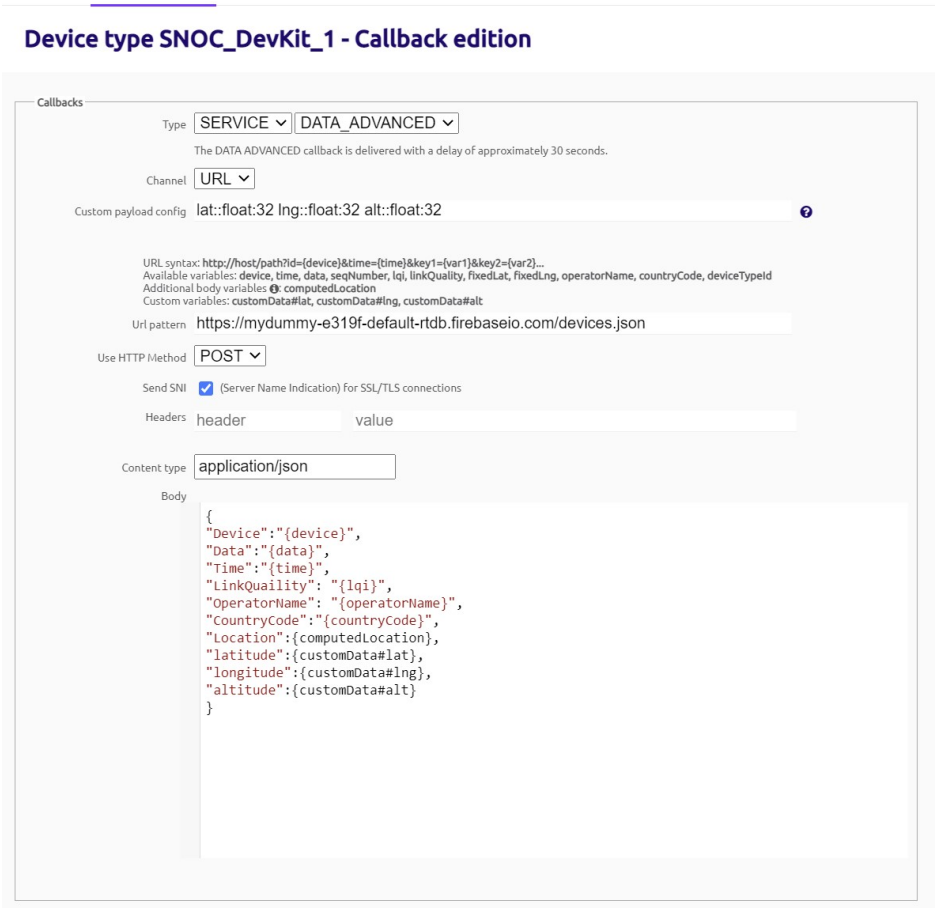


Figura 5.3: Ejemplo de *callback* en Sigfox

5.4 Conexión BLE

La placa **Arduino Mkr Wifi 1010** posee la capacidad de utilizar los protocolos de comunicación tanto de *Bluetooth* como de BLE. Para minimizar el consumo de energía y garantizar una mayor eficiencia, elegimos el segundo puesto que, además, es el más idóneo para aplicaciones de IoT. También proporciona un alcance y una velocidad de transmisión de datos más que aceptable para nuestro propósito.

Para poder establecer la conexión BLE en **Arduino** es necesario seguir una serie de pasos:

1. Incluir la librería *ArduinoBLE* [65]. Esta librería soporta tanto BLE como *Bluetooth 4.0* y superiores.

2. Establecer el nombre de nuestro dispositivo. Este será el nombre que verán los dispositivos que se conecten a él. Se establece usando la función *BLE.setLocalName( )*, indicando el nombre que queremos.

3. Crear un nuevo servicio para este dispositivo. Esto se puede realizar mediante la función *newService( )*, pasándole el nombre del nuevo servicio.

4. Crear una *characteristic*. Para ello, debemos indicar si vamos a poder escribir en ella, leer su contenido o notificar si este cambia mediante los *flags BLEWrite*, *BLERead* y *BLENotify*.

5. Añadir las características creadas al servicio. Una vez añadidas deberemos de añadir también el servicio.

6. Hacer visible (alcanzable) nuestro dispositivo mediante la función *BLE.advertise( )*.

7. Por último, comprobar si algún otro dispositivo se conectó a él, usando las funciones *BLE.central()* y *BLE.central( ).connected( )*.

En total, en el proyecto se han necesitado declarar 4 características. Dos de ellas se han definido con los *flags* de *BLERead* y *BLENotify*, mientras que el resto utilizarán el *flag BLEWrite*.

Una de las características declarada para lectura permitirá acceder a los valores de los diferentes sensores del *dummy*, los cuales son: valores del sensor de temperatura y humedad, valores de los sensores de fuerza, valores del sensor de inclinación y valores del sensor de aceleración. Se define también con el *flag BLENotify* para que la aplicación móvil sea notificada cuando se modifica el valor de esta característica. Otra de las características será la utilizada para transmitir el identificador del *dummy* nada más se establezca la conexión entre la aplicación y el *dummy* para que, de esa forma, el usuario pueda registrarlo automáticamente en su base de datos.

Las características que se han definido con el *flag BLEWrite* serán de escritura y son utilizadas para: 1) que el usuario pueda silenciar/de-silenciar el *dummy* a mayores de reproducir sonidos en el *dummy* a través de la aplicación móvil, y 2) avisar al *dummy* para que comience el envío de datos de su ubicación en la simulación de rescate. Por tanto, el hardware del *dummy* necesitará comprobar el valor de dichas características para realizar las tareas y acciones necesarias.

5.5 Integración de todos los componentes

Una vez que probamos y configuramos todos los módulos y sensores por separado, el siguiente paso es integrarlos todos juntos en el *smart dummy* para que cumplan el objetivo final: monitorizar y proporcionar distinta información acerca del estado del *dummy*.

Teniendo en cuenta el propósito de este proyecto y las situaciones que puede vivir un *dummy* (caídas, golpes, movimientos bruscos...), es necesario dotar a los componentes hardware de la mayor robustez y solidez posibles para que no sufran ningún fallo. Para ello, se han integrado y juntado todos los componentes en una placa de desarrollo.

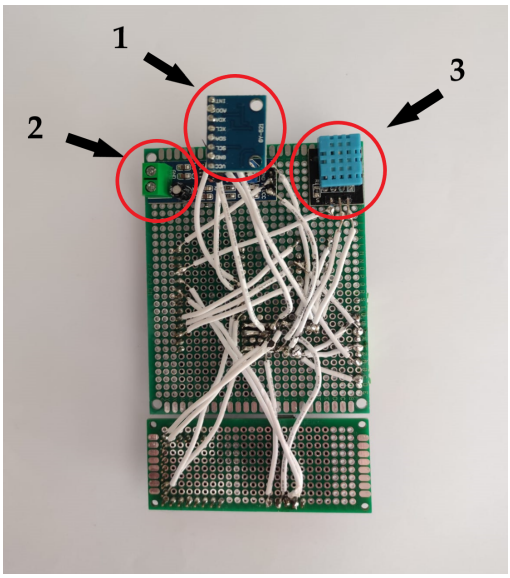
Para conectar y unir todos los componentes a través de esta placa, podríamos haber utilizado los típicos *jumpers* o conectores de **Arduino** pero estos, si se llegan a producir movimientos bruscos o golpes, corren el riesgo de desconectarse y provocar fallos en el hardware. Es por ello que se ha decidido soldar a mano uno por uno todo los componentes hardware y conexiones necesarias para dotar así de un mayor nivel de robustez y seguridad al *smart dummy* durante la realización de las diferentes prácticas y técnicas sanitarias. Para ello, se han cortado a medida todos los cables necesarios para la interconexión de los módulos y sensores, soldando estos a los diferentes conectores o pines del **Arduino Mkr Wifi 1010** y a los diferentes módulos y sensores. Para este proceso se ha utilizado un soldador de estaño, cable sólido de 22 AWG y alambre de estaño.

En la figura 5.4, se puede apreciar el resultado final del montaje sobre la placa de desarrollo y podemos ver los diferentes componentes que se han necesitado integrar en la placa PCB, numerados del 1 al 8. Estos números hacen referencia a los siguientes componentes:

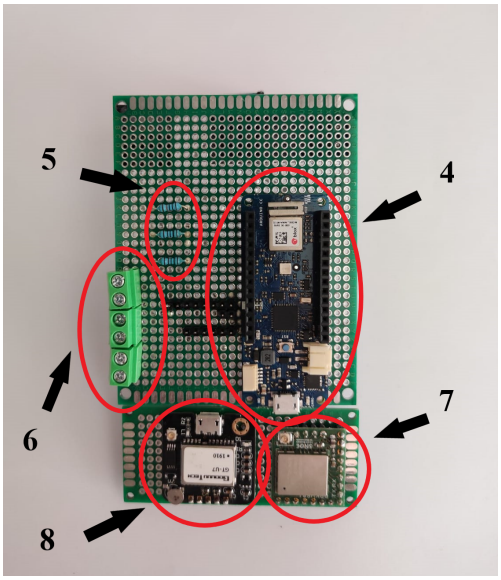
1. Módulo acelerómetro-osciloscopio.
2. Amplificador.
3. Módulo temperatura-humedad.
4. Arduino Mkr Wifi 1010.
5. Resistencias 10KΩ para los sensores de fuerza.
6. Conectores para los tres sensores de fuerza.

7. Módulo **Sigfox**.

8. Módulo GPS.



(a) Distribución PCB por detrás.



(b) Distribución PCB por delante.

Figura 5.4: Distribución PCB

Uno de los principales problemas que nos surgió al unir todos los componentes fue como conectar tanto el módulo GPS como el módulo de **Sigfox** al puerto TX/RX del **Arduino MKR Wifi 1010**. Esta placa solo dispone de un puerto UART, por lo que es necesario declarar uno nuevo, asignando los pines digitales 1 y 0 para dicho fin. En la figura 5.5, se puede ver la instrucción que nos permite hacer esto en Arduino.

```
// Instantiate the Serial3 class
Uart Serial3(&sercom3, 1, 0, SERCOM_RX_PAD_1, UART_TX_PAD_0);
```

Figura 5.5: Declaración de un nuevo puerto UART

Una vez integrados todos los componentes en la placa de desarrollo, se llevó a cabo la recuperación de los datos proporcionados por parte de los diferentes sensores y módulos hardware para su posterior envío a través de las características BLE. En el siguiente código, podemos ver un ejemplo de como se pueden recuperar los valores de inclinación y aceleración sobre el *dummy*, los cuales se recogen utilizando la librería *Wire*.

```
#include <Wire.h>

Wire.begin();
Wire.beginTransmission(MPU);
Wire.write(0x6B);
Wire.write(0);
Wire.endTransmission(true);

//Leemos los valores del Acelerometro
Wire.beginTransmission(MPU);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU,6,true);
AcX=Wire.read()<<8|Wire.read();
AcY=Wire.read()<<8|Wire.read();
AcZ=Wire.read()<<8|Wire.read();

//Leemos los valores del Giroscopio
Wire.beginTransmission(MPU);
Wire.write(0x43);
Wire.endTransmission(false);
Wire.requestFrom(MPU,4,true);
GyX=Wire.read()<<8|Wire.read();
GyY=Wire.read()<<8|Wire.read();

```

Otro ejemplo es la utilización de la librería **dht.h** para el acceso a los datos del sensor de temperatura y humedad.

```
#include "DHT.h"

#define DHTTYPE DHT11

DHT dht(2, DHTTYPE); // digital pin connected

dht.begin();

float h = dht.readHumidity();
float t = dht.readTemperature();

```

En las dos imágenes de la figura 5.6, podemos ver la distribución de todo el hardware utilizado en el proyecto para ser colocado sobre el *dummy*. Esta distribución de los componentes hardware se ha diseñado de forma medida para adaptarse a las peculiaridades que presenta la constitución física del *dummy*, pues el interior de este como ya comentamos está relleno de

diferentes tipos de sacos y cilindros de arena.

Se decidió colocar toda la placa base, módulos hardware, antenas de comunicación y batería en un armazón prefabricado que tiene la forma del torso de una persona, extraído de un maniquí *RCP-AED* [66]. Este armazón irá alojado dentro del propio *dummy* y nos permite lograr un mayor nivel de protección de todos los componentes, pues el *dummy* puede sufrir golpes, movimientos bruscos o caídas durante la realización de las prácticas sanitarias.

A mayores, para así poder conseguir una mayor realismo y nivel de precisión, se decidió colocar un listón de metal de manera transversal, para simular así una columna vertebral. En este listón va fijado el sensor de inclinación para conseguir que los datos que nos proporcione sean lo más reales posible. El resultado completo, incluyendo el listón de metal, se puede ver en la parte derecha de la figura 5.6.

(a) Integración de todos los componentes sin columna.

(b) Integración de todos los componentes con columna.

Figura 5.6: Integración de todos los componentes hardware

En la imagen 5.7, se muestra como uno de los sensores de fuerza va colocado en los sacos de arena que forman el *dummy*, en este caso el del cuello. Este tipo de sacos son los mismos que se utilizan para las extremidades, por lo que la idea para los sensores que van en el brazo y en el muslo es la misma. Tanto los sensores y cableado de las extremidades como del cuello van fijados y pegados al *dummy* por lo que, a la hora de retirar el torso con los componentes hardware, no tendremos que desmontar completamente el circuito, sino que bastará con destornillar los conectores, facilitando así las tareas de mantenimiento o reparación de los componentes hardware.

La imagen 5.8 representa como el torso con todos los componentes hardware va montado

Figura 5.7: Representación de la colocación del sensor de fuerza en el cuello.

en el interior del *dummy*, conectado a su vez con los sensores de presión repartidos por las extremidades y cuello. El interior del *dummy* está formado básicamente por dos láminas de sacos de tierra, por lo que resulta idóneo colocar el torso con los componentes hardware en medio de estas, proporcionando así un mayor nivel de protección.

Figura 5.8: *Dummy* con todos los componentes hardware montados.

Capítulo 6

Análisis, diseño e implementación de la aplicación móvil

PARA la monitorización y control de todos los datos proporcionados por el *smart dummy*, es necesario la creación de una aplicación móvil que permita al usuario interactuar con el sistema. Para el desarrollo de cualquier aplicación es importante seguir alguna especificación que nos permita organizar todas las etapas del ciclo de desarrollo del software. En este sentido, se debe comprender el problema (fase de análisis) para luego plantear una posible solución (fase de diseño) y, por último, llevar a cabo la solución planteada (fase de implementación). En este capítulo, vamos abordar cada una de estas fases por separado.

6.1 Actores del sistema

Existen dos tipos de usuarios que pueden interactuar y utilizar las diferentes funcionalidades del sistema. Estos son los usuarios **normales** y los usuarios **administradores**.

1. **Usuario normal**: Cualquier usuario que se registre en la aplicación correctamente adquiere este rol. Este tipo de usuario puede hacer uso de casi todas las funcionalidades del sistema, excepto de la modificación de los umbrales de los sensores y de la modificación del rango de búsqueda en una simulación. Puede registrar o vincularse con un usuario administrador mediante el escaneo del código QR de este, para así incorporar los parámetros que estableció ese usuario administrador.

2. **Usuario administrador**: Sólo el administrador del sistema puede crear este tipo de usuarios, directamente en la base de datos de **Firebase**. Este usuario aparte de tener acceso a las mismas funcionalidades y servicios que un usuario normal, puede modificar y establecer los umbrales de los diferentes sensores, así como la modificación de la distancia del rango de búsqueda en una simulación de rescate. Puede generar y visualizar su código QR para que un usuario normal pueda vincularse con él.

6.2 Análisis de requisitos

Estos requisitos son una condición o capacidad que necesita el desarrollador para solventar el problema propuesto y son capturados o recogidos a través de un trabajo conjunto entre los analistas o desarrolladores de software y los clientes.

En el caso concreto de este proyecto, el análisis de requisitos y definición de funcionalidades se hizo en colaboración con profesionales expertos en actividades de formación para primeros auxilios y rescates (de la empresa **Efesgal**), que nos ayudaron a establecer las necesidades y requisitos que debería cumplir la aplicación.

Antes de detallar los requisitos, cabe destacar que, para acceder a la mayoría de funcionalidades de la aplicación, existe un precondición general y es que el usuario debe tener iniciada una sesión en la aplicación. Esta precondición no es necesaria solo en el caso del registro de usuario, para hacer *login* y para acceder a la opción de recuperación de la contraseña.

6.2.1 Requisitos funcionales

Este tipo de requisitos hacen referencia al conjunto de funciones que debe proporcionar o incluir el software que se va a desarrollar. Se han definido los siguientes:

- **Login de usuario**
  - **Entrada**: El usuario introduce su correo electrónico seguido de su contraseña.
  - **Comportamiento**: Realizamos una consulta a **Firebase** para comprobar si el usuario está registrado y los datos introducidos son correctos.
  - **Salida**: Si todo es correcto, el usuario accede a su cuenta en la aplicación. Si se produce un error, se muestra un mensaje indicando el motivo de error.
- **Registro de usuario**
  - **Entrada**: El usuario debe cubrir un formulario en el que se le pedirán una serie de datos.
  - **Comportamiento**: Se comprobará que los datos introducidos son correctos.
  - **Salida**: Si los datos son correctos, se registra al nuevo usuario en la base de datos de usuarios de **Firebase** y se crea una entrada en *Firestore Database*. En caso contrario, se produce un mensaje de error indicando el motivo de fallo.
- **Visualizar perfil**
  - **Entrada**: Accesible mediante un botón en la barra de navegación superior.
  - **Comportamiento**: Se extraen los datos de la colección del usuario de **Firebase**.

  - **Salida**: Se representan los datos del usuario junto con la imagen que este haya establecido para su perfil.

- **Modificar datos de usuario**
  - **Entrada**: El usuario puede modificar su nombre de perfil o la foto de perfil, de forma que debe proporcionar estos datos.
  - **Comportamiento**: Se comprueba que los datos introducidos son correctos.
  - **Salida**: Se almacena el nuevo nombre en la colección del usuario en *Firestore Database* y la imagen en la base de datos *Storage* de **Firebase**. Si se produce un error durante la actualización de los datos, se muestra el mensaje de error.

- **Recuperar contraseña de usuario**
  - **Entrada**: El usuario solicita recuperar la contraseña en la ventana de *login* e introduce su correo electrónico.
  - **Precondiciones**: El usuario debe tener una cuenta registrada con ese correo electrónico.
  - **Comportamiento**: Se comprueba que el correo tiene el formato correcto y está registrado.
  - **Salida**: Si todo es correcto, se envía un mensaje de recuperación propio de **Firebase** al correo del usuario, permitiéndole así el cambio de contraseña. En caso contrario, se muestra un mensaje de error.

- **Contactar con el equipo de desarrollo**
  - **Entrada**: Mediante la vista del perfil, en el desplegable de la barra de navegación superior, el usuario puede contactar mediante correo electrónico con los encargados de la aplicación.
  - **Comportamiento**: Se solicita al usuario que aplicación de mensajería quiere abrir para completar dicha acción.
  - **Salida**: Se abre la aplicación correspondiente, estableciendo como asunto el identificador del usuario y como destinatario la cuenta de correo del proyecto.

- ***Overview* de la aplicación**
  - **Entrada**: Se ejecutará la primera vez que un usuario inicie la aplicación.
  - **Precondiciones**: Debe ser el primer inicio de la aplicación.
  - **Comportamiento**: Se comprueba mediante las *SharedPreferences* que la aplicación no había sido iniciada anteriormente.

  - **Salida**: Se muestran las principales características de la aplicación y se actualizan las *SharedPreferences* para que no vuelva aparecer la *overview*.

- **Añadir nuevo dummy**

  - **Entrada**: Cuando se establece la conexión BLE con un nuevo *dummy*, el sistema guarda/registra el identificador de dicho *dummy* en la información de usuario de **Firebase**.
  - **Precondiciones**: Que se establezca la conexión mediante BLE.
  - **Comportamiento**: Se comprueba que ese *dummy* no está ya registrado para el usuario.
  - **Salida**: Se añade el nuevo dispositivo registrado en la colección correspondiente de **Firebase** del usuario.

- **Cambiar umbrales de sensor (administrador)**

  - **Entrada**: Dentro de la vista detallada de cada sensor, el administrador puede cambiar los umbrales máximos y mínimos para ese sensor desde la barra de navegación superior.
  - **Comportamiento**: Se comprueba en la base de datos de **Firebase** los últimos valores establecidos y si es necesario actualizarlos.
  - **Salida**: Los nuevos valores son almacenados en la base de datos de **Firebase** del usuario administrador. En caso de que se produzca algún fallo, se muestra un mensaje de error.

- **Vincular administrador (usuario normal)**

  - **Entrada**: Mediante el escaneo del código QR de un administrador, el usuario puede "vincularse" a ese administrador. Accesible mediante la vista perfil.
  - **Precondiciones**: El usuario debe tener los permisos necesarios para que la aplicación acceda a la cámara.
  - **Comportamiento**: Se comprueba de que el administrador existe y que dicho administrador no está ya vinculado con ese usuario.
  - **Salida**: Si el proceso finaliza de forma correcta, se guarda esta asociación en la colección correspondiente del usuario en *Firestore Database*. De esta forma, quedan fijados para el usuario los umbrales establecidos por el administrador para los diferentes sensores. En caso de que no exista vinculación con ningún administrador, se establecen unos valores por defecto para el usuario.

- **Mostrar QR (administrador)**
  - **Entrada**: Al acceder a la vista de perfil, el administrador puede obtener su código QR.
  - **Comportamiento**: Se genera el código QR a partir del identificador de usuario administrador.
  - **Salida**: Se muestra el código QR a pantalla completa.
- **Configurar preferencias**
  - **Entrada**: Mediante la pantalla de perfil, un usuario puede acceder a modificar sus preferencias.
  - **Comportamiento**: Se comprueban las *SharedPreferences* guardadas y si está conectado con el *dummy*.
  - **Salida**: Las preferencias son modificadas en la entrada correspondiente de la base de datos *SharedPreferences*.
- **Ver vista completa de sensores**
  - **Entrada**: Mediante la barra de navegación inferior, el usuario puede acceder a una vista global de todos los sensores con los valores proporcionados por *dummy* al que está conectado.
  - **Comportamiento**: Se comprueba que está conectado el *dummy* mediante BLE.
  - **Salida**: Se muestran los datos devueltos por el *dummy* de una manera visual e intuitiva, así como la posibilidad de interactuar con el maniquí. Si no está conectado, se muestra un mensaje de error.
- **Ver sensores y módulos por separado**
  - **Entrada**: Podemos acceder mediante el botón de la barra de navegación inferior.
  - **Comportamiento**: Se comprueban los sensores que están declarados y configurados. Cuando se selecciona un sensor, se comprueba que existe conexión con el *dummy*, se recuperan los datos para ese sensor y se presenta la información correspondiente a este.
  - **Salida**: Se obtiene una lista de todos los posibles sensores y módulos conectados al *dummy* para visualizar dentro de cada uno de ellos la información obtenida por estos. Si no existe conexión con el *dummy*, se muestra un mensaje de error.

- **Iniciar simulación rescate**
  - **Entrada**: Mediante el botón *Iniciar simulación* ubicado en la pantalla general de monitorización o bien mediante la lista que muestra todos los sensores.
  - **Precondiciones**: Debe existir conexión BLE con el *dummy*.
  - **Comportamiento**: Se comprueba que tenemos conexión con el *dummy* mediante BLE. A continuación, se envía un *trigger* al *dummy* para que empiece a transmitir datos acerca de su posición cada 2 minutos mediante la red **Sigfox**, utilizando para ello o bien el módulo GPS o las distintas MAC cercanas captadas.
  - **Salida**: En la pantalla de ubicación se puede visualizar un mapa con las posiciones enviadas por el *dummy*.
- **Modificar rango de búsqueda (administrador)**
  - **Entrada**: En el mapa, el administrador puede pulsar el botón de configuración para modificar el rango de búsqueda para un *dummy*.
  - **Comportamiento**: Se obtiene el último valor establecido del rango de búsqueda de *Firebase* y se comprueba si es necesario actualizarlo.
  - **Salida**: El valor de rango establecido por el administrador se almacena en la colección especifica de *Firestore Database*. En caso de fallo, se muestra un mensaje de error.
- **Escanear dispositivos BLE**
  - **Entrada**: El usuario accede a la pantalla de escaneo de dispositivos mediante la barra de navegación inferior.
  - **Precondiciones**: La aplicación dispone de los permisos *Bluetooth* necesarios.
  - **Comportamiento**: Se comprueba que la interfaz de *Bluetooth* del dispositivo móvil está conectada. En caso de que no esté activa, se le da la posibilidad al usuario de activarlo. Además, se comprueba que no existe ninguna conexión BLE ya establecida.
  - **Salida**: Se muestra una lista con todos los nombres e identificadores de dispositivos que están disponibles para conectarse. Existe también el botón de recargar y actualizar la lista, o directamente arrastrando la pantalla desde el borde superior hacia abajo.
- **Conectar dispositivo al *dummy* por BLE**
  - **Entrada**: El usuario, al hacer *click* en la lista de dispositivos disponibles, inicia la conexión mediante BLE con el dispositivo.

  - **Precondiciones**: Escaneo previo de los dispositivos BLE disponibles.
  - **Comportamiento**: Se inicia el servicio de conexión BLE.
  - **Salida**: Se establece la conexión con el *dummy* y se muestra información acerca de este enlace. El *dummy* queda registrado para ese usuario si no lo estaba ya.

- **Desconectar dispositivo BLE**

  - **Entrada**: Mediante la barra de navegación inferior, el usuario puede terminar la conexión con el *dummy*.
  - **Precondiciones**: Que exista una conexión BLE.
  - **Comportamiento**: Se comprueba que existe dicha conexión con el *dummy* y se procede a desactivarla.
  - **Salida**: Se elimina la conexión BLE con el *dummy*.

- **Consultar localización**

  - **Entrada**: El usuario podrá consultar la última ubicación conocida del *dummy* seleccionando el mapa en la barra de navegación inferior. Esta ubicación se trata de la última ubicación enviada por alguno de los dispositivos que tiene registrados. Si tiene varios *dummies* registrados, puede seleccionar cual quiere visualizar.
  - **Precondiciones**: Que el *dummy* haya enviado como mínimo un mensaje de localización.
  - **Comportamiento**: Se comprueba la conexión con internet del dispositivo, los permisos de ubicación y GPS que tiene establecidos la aplicación y si existe algún *dummy* registrado para dicho usuario.
  - **Salida**: Se muestra mediante la API de **Google Maps** la última posición conocida del *dummy* junto con un posible rango de búsqueda en el mapa. Si se produce algún fallo, se muestra un mensaje de error.

- **Consultar historial de localizaciones**

  - **Entrada**: Mediante un botón flotante encima del mapa el usuario, se puede consultar la lista completa de las posiciones que envió el *dummy*.
  - **Precondiciones**: El *dummy* debe haber enviado al menos un mensaje de localización.
  - **Comportamiento**: Se accede a la base de datos *Realtime* en **Firebase** correspondiente a los *dummies* que tiene registrado el usuario.

  - **Salida**: Se muestra una lista con todos los mensajes de todos los *dummies* registrados para ese usuario. El usuario luego puede ordenar dicha lista por fecha ascendente o descendente, buscar directamente por fecha o filtrar por *dummy*.

- **Navegar hasta una ubicación**

  - **Entrada**: El usuario puede iniciar la navegación hasta la localización del *dummy* mediante el botón flotante del mapa.
  - **Comportamiento**: Se comprueba que el dispositivo contiene la aplicación **Maps** instalada.
  - **Salida**: Se inicia la aplicación de **Maps** con la mejor opción de ruta para llegar a la ubicación del *dummy*.

- **Mostrar últimas 5 localizaciones**

  - **Entrada**: Si el usuario despliega el botón flotante sobre el mapa, puede encontrar un botón para acceder a las 5 últimas posiciones conocidas del *dummy*.
  - **Precondiciones**: El *dummy* debe haber enviado al menos cinco mensajes de localización.
  - **Comportamiento**: Se comprueba que el *dummy* está registrado para dicho usuario y se extraen de la base de datos de **Firebase** las últimas 5 posiciones.
  - **Salida**: Se muestran estas posiciones sobre el mapa, utilizando la API de **Google Maps**.

- **Mostrar información de mensaje de ubicación**

  - **Entrada**: Cualquier usuario puede *clickar* sobre cualquier elemento de la lista de ubicaciones, para obtener información a acerca de ese elemento.
  - **Comportamiento**: Se obtienen de **Firebase** los datos correspondientes a dicho mensaje.
  - **Salida**: Se muestra una colección de datos, como son el *country code*, el texto del mensaje enviado desde el *dummy* (dos MAC o coordenadas obtenidas a través del módulo GPS) en formato hexadecimal, la fecha y hora en la que se envió el mensaje, el identificador del dispositivo, la calidad del enlace de red, el nombre del operador que atendió el mensaje, y la localización y error estimado por **Sigfox**.

- **Eliminar mensaje de ubicación**

  - **Entrada**: El usuario puede eliminar cualquier mensaje de ubicación emitido por el *dummy* mediante el botón *eliminar* en la vista detallada de cada mensaje.

  - **Comportamiento**: Se realiza una operación de borrado en la base de datos de **Firebase**.
  - **Salida**: Se elimina el mensaje, se muestra un mensaje de que todo ha ido bien y se vuelve a la lista que contiene todos los mensajes de ubicación. Si algo falla, se produce un mensaje de error.

- **Estimar posición mediante MAC**
  - **Entrada**: Si el mensaje obtenido por parte de **Sigfox** se corresponde con las MAC de dos *routers WiFi*, se habilitará un botón en la vista detallada del mensaje para hacer uso de la API de geolocalización del *dummy*.
  - **Comportamiento**: Se realiza una petición asíncrona a la API de **Google** de la que se obtiene un conjunto de coordenadas a partir de la información dada.
  - **Salida**: Se muestra en un mapa este conjunto de coordenadas, indicando así la posición precisa del *dummy*.

- **Mostrar posición a través de la información obtenida del GPS**
  - **Entrada**: Si el mensaje obtenido por parte de **Sigfox** se corresponde con las coordenadas obtenidas del módulo GPS, se habilita un botón en la vista detallada del mensaje para mostrar dichas coordenadas en el mapa utilizando **Google Maps**.
  - **Comportamiento**: Se comprueba que dichas coordenadas tienen el formato correcto y tiene similitud con las estimadas por **Sigfox**.
  - **Salida**: Se muestra en un mapa la localización seleccionada.

- **Consultar documentación**
  - **Entrada**: El usuario puede consultar una serie de guías básicas de primeros auxilios a través de la barra de navegación inferior.
  - **Comportamiento**: Se extrae de una base de datos interna toda la documentación disponible.
  - **Salida**: Se obtiene una lista con las diferentes guías disponibles para que puedan ser consultadas por parte de un usuario.

6.2.2 Requisitos no funcionales

Este tipo de requisitos se pueden entender como aquellos requisitos que imponen ciertas restricciones en el diseño o en la implementación de la aplicación, siempre teniendo en cuenta estándares de calidad. En nuestro caso, se han considerado los siguientes:

- **Conectividad**. La aplicación requiere tanto de conexión a internet, sistema GPS para posicionamiento y conectividad BLE.
- **Facilidad de uso**. Debemos proporcionar a los usuarios un sencillo y fácil manejo de la aplicación con interfaces amigables.
- **Integración**. La aplicación está pensada para funcionar en cualquier sistema **Android** cuya versión sea posterior a la versión *5.1 Lollipop*.
- **Accesibilidad**. La aplicación debe seguir los patrones de accesibilidad establecidos por **Google**.
- **Robustez de datos**. El sistema debe garantizar que no hay usuarios duplicados y debe controlar la correcta entrada de datos.
- **Mensaje de información y errores orientativos**. Cada vez que un usuario realice una tarea se le deberá informar con un mensaje de éxito o error, así como avisarlo si se está realizando una tarea crítica.
- **Fluidez**. Se debe proporcionar un cierto grado de fluidez en las operaciones para que los usuarios puedan mantener un cierto ritmo de interacción con la aplicación y que no sea molesto su uso.
- **Seguridad**. Realizar un control y una gestión de usuarios segura gracias a la utilización de la API *Auth* de **Firebase**.

6.2.3 Diagramas de casos de uso

Los diagramas de casos de uso proporcionan una representación gráfica de las posibles interacciones que puede realizar un determinado usuario con el sistema empleando, para ello, los diferentes casos de uso preestablecidos.

La imagen 6.1 muestra el diagrama de casos de uso para un usuario normal, mientras que el usuario administrador tiene el mismo diagrama que el usuario normal salvo que, en este caso, incorpora a mayores los casos de uso expuestos en el diagrama de la figura 6.2.

Todos estos casos de uso vienen determinados directamente por las funcionalidades detalladas en la sección 6.2.1.

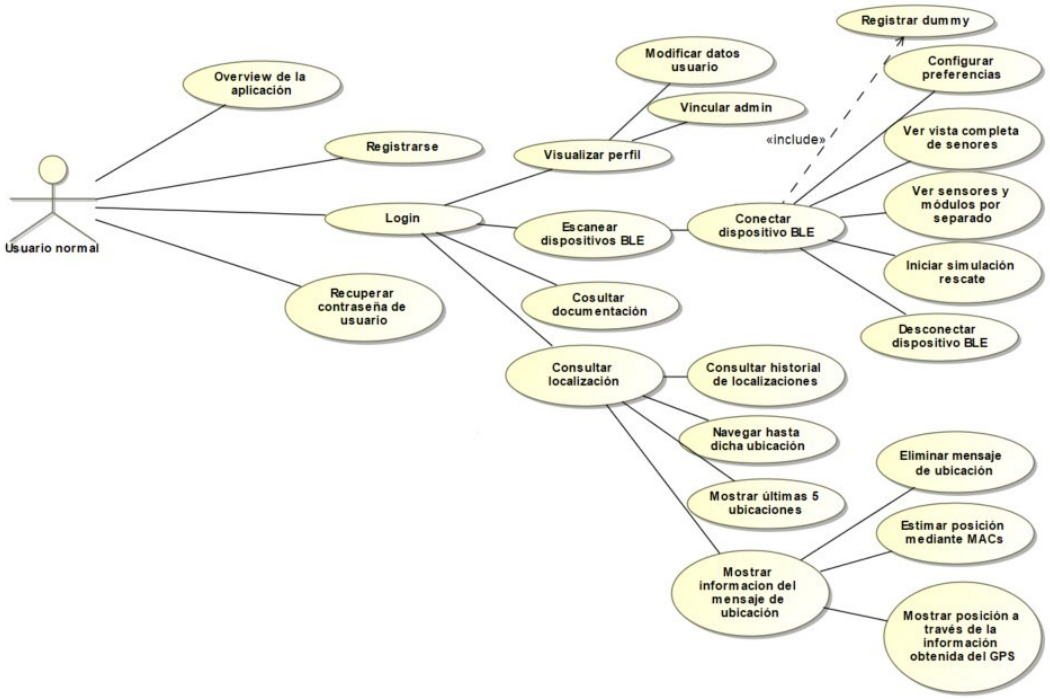


Figura 6.1: Diagrama de casos de uso para un usuario normal.

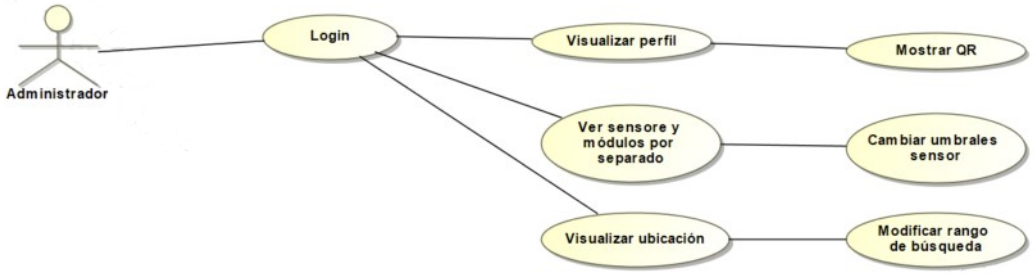


Figura 6.2: Diagrama con los casos de uso específicos para un usuario administrador.

6.3 Diseño

En esta sección, se van a presentar las decisiones de diseño tomadas durante el proceso de desarrollo de la aplicación móvil. Se analizará la arquitectura del sistema y el patrón a utilizar, se presentará el diseño de los principales diagramas de flujo que sigue la aplicación y, finalmente, se detallará la estructura de las bases de datos utilizadas. Además, se explicará brevemente el diseño de los *mockups* para las pantallas de la aplicación.

6.3.1 Patrón utilizado

El principal objetivo de utilizar arquitecturas o patrones de diseño a la hora de desarrollar una aplicación, en este caso en **Android**, es desacoplar diferentes unidades del código de una manera organizada, lo que nos permite cambiar o modificar las diferentes partes del sistema de una manera más fácil sin interferir con el resto.

Una de las arquitecturas más conocidas es la arquitectura **Clean** [67]. En una aproximación básica, esta arquitectura se compone de tres capas principales que tienen una serie de responsabilidades individuales:

- **Data**: Maneja los datos e información de la aplicación. Normalmente obtiene estos datos a partir de una API, internet o bases de datos.
- **Domain**: Contiene la lógica interna de la aplicación. Está formada principalmente por clases de tipo Plain Old Java Object (POJO) [68].
- **Presentation**: Presenta los datos e información en una serie de pantallas y maneja las interacciones con los usuarios.

La principal característica de Clean Architecture (CA) es una separación correcta de las diferentes capas del sistema evitando acoplamientos, de tal forma que la capa *domain* es independiente de cualquiera de las otras dos capas, es decir, no debemos acceder ni interactuar con ninguna de las clases definidas en las otras dos capas. Gracias a esto, tenemos la certeza de que si realizamos cualquier cambio aplicado a los datos o a la presentación no afectará a la lógica de negocio de la aplicación.

En este proyecto, se ha aplicado este patrón de diseño con una serie de modificaciones muy sutiles. La figura 6.3 ilustra la arquitectura y estructura del proyecto siguiendo el patrón CA. En primer lugar, la capa **Presentation** se pasó a denominar **Module** e incluye 3 carpetas: **adapter**, **presenter** y **views**. Con estas carpetas, separamos los diferentes componentes que forman las vistas e interacciones del usuario con la aplicación. En la capa **Domain**, incluimos las clases POJO utilizados para el proyecto, así como las interfaces de los servicios para acceder a las distintas bases de datos y APIs. Por último, en la capa **Data**, incluimos las implementaciones de dichos servicios y los diferentes métodos de acceso a la información.

También se siguió e implementó el concepto de **Clean Code** [69]. Este principio recoge una serie de guías y buenas prácticas a la hora de escribir código para que este sea lo más legible y ordenado posible. Algunas de las características principales de *Clean Code* seguidas en este proyecto son, por ejemplo, el uso de funciones pequeñas y organizadas, siguiendo el principio Don't Repeat Yourself (DRY) [70], y con verbos en su nombre que indiquen su principal propósito. A mayores, en las variables se intentó dar sentido a estas mediante nombres intencionados y descriptivos, evitando las abreviaciones o prefijos.

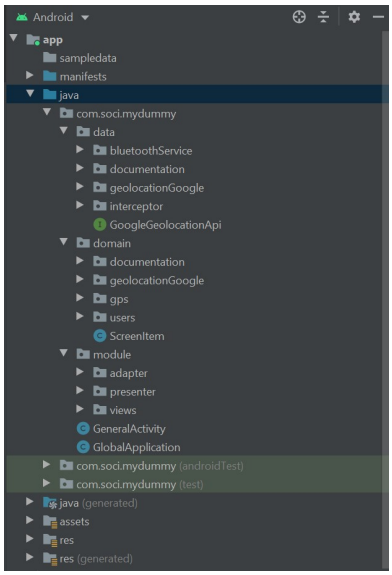


Figura 6.3: Estructura del proyecto siguiendo la arquitectura Clean

6.3.2 Diagramas de flujo

Para comprender mejor algunas de las funcionalidades más complejas que debe proporcionar la aplicación, resulta de gran ayuda realizar diagramas de flujo en la fase de diseño para representar la secuencia de acciones o pasos lógicos necesarios para completar estas tareas.

En primer lugar, vamos a centrarnos en el proceso de registro y de autenticación que un usuario puede realizar en nuestra aplicación. Para ello, la figura 6.4 muestra el diagrama de flujo que siguen estas acciones. A partir de la introducción de los datos de registro y la comprobación de que estos son correctos, estos datos son enviados al servidor de **Firebase** para ser almacenados en la base de datos. El servidor envía una respuesta a la aplicación que es mostrada al usuario y, a mayores, mediante el servicio *Cloud Functions* se envía un correo de confirmación de registro al *email* del usuario. Es entonces cuando el usuario puede iniciar sesión y es redirigido a la pantalla principal de la aplicación.

La conexión BLE desde la aplicación con el *dummy* se realiza siguiendo los pasos indicados en el diagrama de flujo correspondiente a la figura 6.5. A partir de un primer escaneo para encontrar dispositivos *Bluetooth* en el área de alcance, seleccionamos el dispositivo al que nos queremos conectar y, cuando este acepte la conexión, se establecerá la comunicación entre ambos dispositivos. Automáticamente, se leerá una característica BLE para así poder conocer el identificador del *dummy* y se comprobará si ese dispositivo ya se ha registrado por parte de ese usuario. En caso de que no esté registrado, se procederá con su registro en la base de datos de **Firebase**.

En general, si queremos visualizar alguno de los datos devueltos por los sensores del *dummy*, deberemos solicitar la información de la *characteristic* específica y presentar la in-

formación asociada de una manera visual e intuitiva para el usuario.

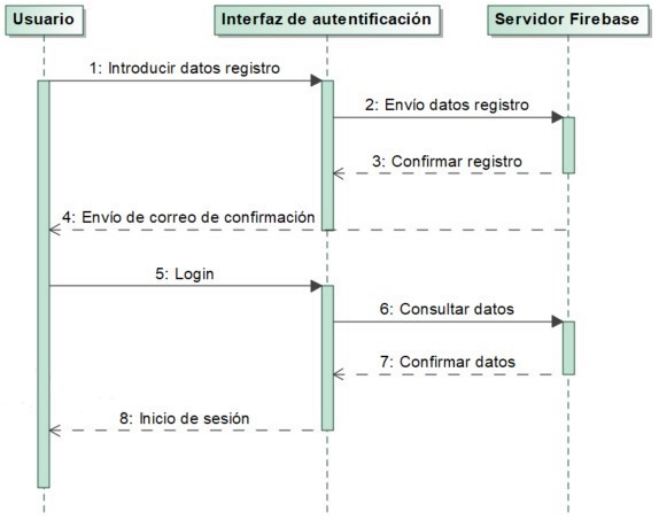


Figura 6.4: Diagrama de flujo del proceso de Registro y Login

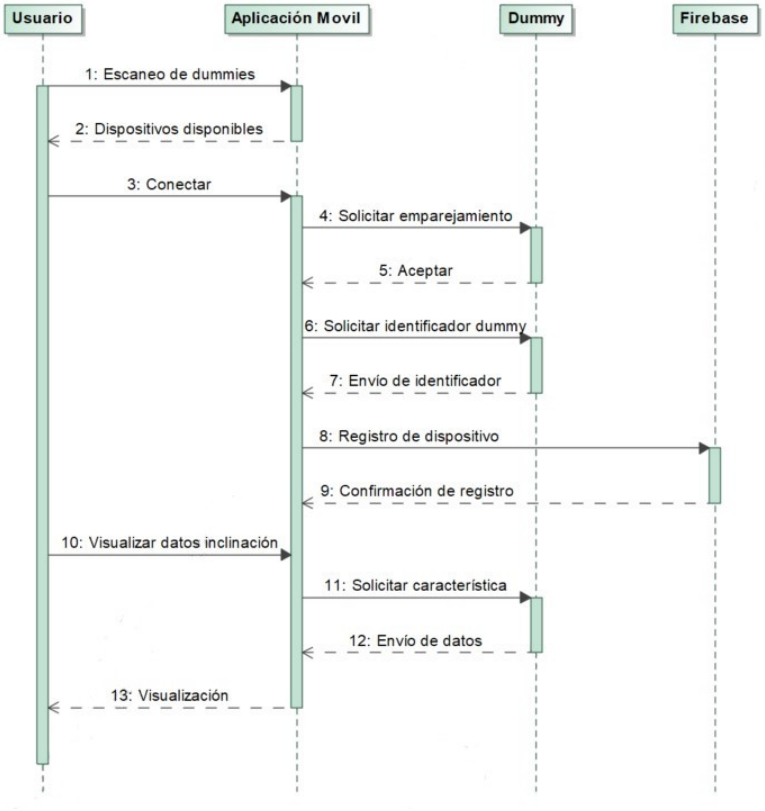


Figura 6.5: Diagrama de flujo del proceso de conexión BLE

Por último, en la figura 6.6, podemos ver el conjunto de pasos que tiene que realizar nuestra aplicación para mostrarle al usuario la última ubicación conocida de un *dummy*. Para ello, se tiene que realizar la comprobación de que dicho usuario tiene registrado algún *dummy* en el servidor de **Firebase** para, luego, solicitarle a este la última entrada de la lista de posiciones para dicho dispositivo. Una vez se recupera ese mensaje, se extraen los valores de las coordenadas y, mediante la API de **Google Maps**, el usuario puede visualizar la última posición conocida del *dummy* sobre un mapa. En el mapa que se genera, el usuario puede seleccionar el *dummy* del que quiere obtener su última posición (de los que tiene registrados).

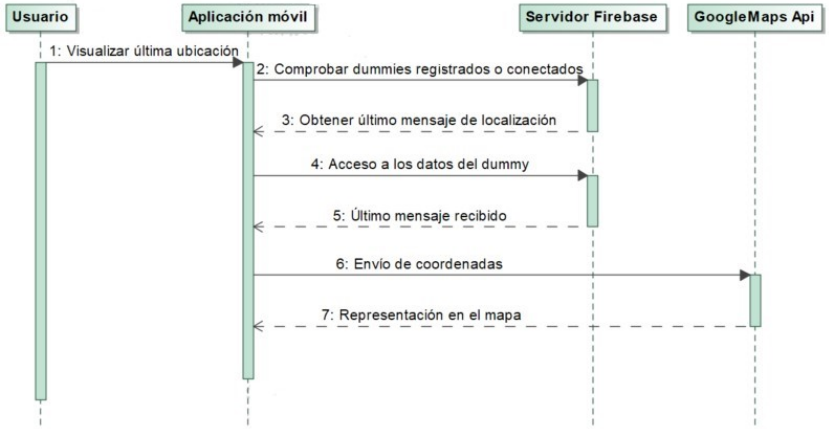


Figura 6.6: Diagrama de flujo para visualización de la última localización de un *dummy*

6.3.3 Bases de datos

En el proyecto se utilizan un total de cinco bases de datos diferentes, para guardar tanto datos de usuarios, mensajes recibidos desde el *dummy*, documentación acerca de las guías de primeros auxilios y preferencias o características de la aplicación. En concreto, las bases de datos utilizadas son:

- **Realtime database**: Base de datos alojada en el servidor de **Firebase**, en la que los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. En ella, crearemos los 3 tipos de tablas que se muestran en la figura 6.7. En primer lugar, tenemos una tabla **Devices** en la que se almacenarán todos los mensajes de posición recibidos de los diferentes dispositivos *dummy* directamente desde el *callback* de **Sigfox**.

  Para cada entrada que se cree en la tabla **Devices**, se efectuará una copia de todos los datos del mensaje mediante el uso de las **Cloud Functions**. Esta información será almacenada en una colección específica en función del identificador del *dummy* que envió el mensaje, para facilitar así su posterior lectura desde la aplicación móvil. Por último, se llevará un registro de todos los *dummies* existentes (registrados para los diferentes

usuarios), guardando la duración de la licencia de Sigfox, la fecha en la cual fue registrado y los usuarios que lo tienen registrado.

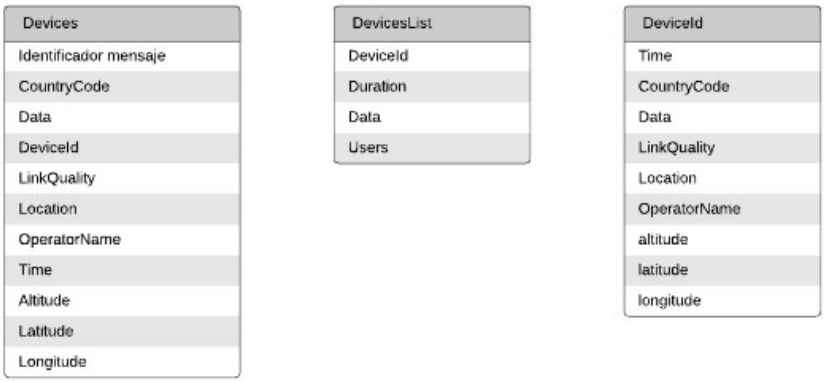


Figura 6.7: Tablas en Realtime Database de Firebase.

- **Firestore Database**: Base de datos no relacional alojada en el servidor de **Firebase** en la que vamos a guardar la información de los distintos usuarios del sistema. Para ello, se crearán dos colecciones, una para los usuarios normales y otra para los administradores. Dentro de cada una de estas colecciones, se guardará un documento u objeto por cada usuario en el que se incluirá la información personal (nombre completo, correo electrónico y administrador vinculado) y los dispositivos que tiene el usuario registrados. En el caso de los usuarios administradores se incluirán, a mayores, los valores de los respectivos umbrales establecidos y el rango de búsqueda fijado por él.

- **Storage**: Permite subir y almacenar archivos de los clientes de una aplicación en los servidores de **Firebase**. Para este proyecto, se utilizará para almacenar las imágenes de perfil de los distintos usuarios, ordenadas por los identificadores de los mismos.

- **SharedPreferences**: Nos permite guardar una pequeña colección de pares clave-valor de los datos de la aplicación, por lo que resulta idóneo para almacenar las preferencias que establezca cada usuario, como silenciar/de-silenciar el *dummy* o activar el modo oscuro. A mayores, podemos guardar si la aplicación ya ha sido iniciada por lo menos una vez, por lo que el *overview* no se reproducirá.

- **Bases de datos interna (Sqlite)**: Para almacenar la información de las guías de primeros auxilios, se creará una base de datos con el gestor **Sqlite** que se importará al proyecto. Cada vez que un usuario quiera visualizar las guías, se accederá mediante un servicio *Room* a la respectiva base de datos interna. La decisión de usar a mayores esta base de datos se debe a que la información que se almacena no varía en ningún momento y se optimiza, de esa forma, el acceso a la información.

6.3.4 Diseño de interfaces

Mediante la herramienta **Adobe XD** se han realizado una serie de pantallas o interfaces que muestran las posibles interacciones de los usuarios siguiendo los requerimientos o casos de uso pre-establecidos. Estos diseños sirven además de guía para ilustrar la navegación entre las diferentes pantallas de la aplicación, además de otorgar a los clientes una vista previa e idea general de la aplicación.

Una de las pantallas más importantes a tener en cuenta, es la que nos permite visualizar toda la información de monitorización del *dummy*. Esta pantalla se puede ver en la figura 6.8 y nos permite visualizar tanto los datos de temperatura y humedad, como inclinación y aceleración del *dummy*, los valores de los distintos sensores de fuerza, reproducción de sonidos y la opción de empezar una simulación de rescate.

Figura 6.8: Pantalla de simulación

Otra pantalla interesante es la que nos proporciona la capacidad de visualizar en un mapa la última posición conocida de un *dummy*. El diseño de esta pantalla se corresponde con la figura 6.9 y, mediante el uso del SDK de **Google Maps** para **Android**, vamos a ser capaces de representar la ubicación del *dummy* a partir de las coordenadas recibidas a través de la red **Sigfox** y almacenadas en la base de datos **Firebase**.

El resto de pantallas o *mockups* que se han realizado para la aplicación se pueden consultar en el apéndice B.

Figura 6.9: Pantalla de localización

6.4 Implementación

En esta sección se comentarán algunos aspectos relevantes en la implementación de las principales funcionalidades o aquellas que suponen un mayor grado de complejidad.

6.4.1 Configuración de Firebase

Cuando iniciamos la opción de simulación de un rescate, el *dummy* va a enviar, cada dos minutos, la posición en la que se encuentra mediante la red de **Sigfox**. Estos mensajes llegan a nuestra base de datos *Realtime* de **Firebase** a través de un *callback* como el que se mostró en la figura 5.3.

Estos mensajes enviados desde **Sigfox** presentan una estructura como la que se puede apreciar en la imagen 6.10 y en la que podemos encontrar varios campos: el campo *Country-Code* nos indica el código del país del operador que recibió el mensaje; el campo *Data* es el *payload* escrito por el *dummy* que, en nuestro caso, podrán ser dos direcciones MAC o las coordenadas GPS en formato hexadecimal; el identificador del *Device*; la calidad del enlace o *LinkQuality*; la localización estimada por **Sigfox**; el nombre del operador que recibió el mensaje; el momento en el cual se recibió el mensaje en formato *epoch* y, por último, los datos de posición obtenidos por el módulo GPS, extraídos del *payload* y formateados, que nos dan una ubicación mucho más precisa que la estimada por **Sigfox**.

Todos los mensajes de ubicación de todos los dispositivos *dummy* llegan a la misma tabla **Devices**. Para poder ordenar y clasificar estos mensajes por su campo *Device* y poder acceder

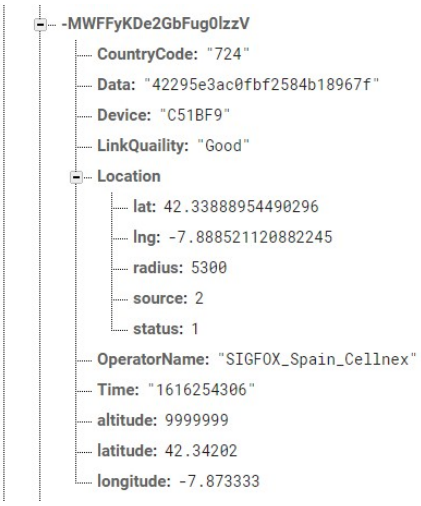


Figura 6.10: Mensaje recibido de Sigfox guardado en Firebase

así de una manera más sencilla y directa a los datos de posición de un *dummy* en concreto desde la aplicación móvil, es necesario la utilización de las *Cloud Functions* de **Firebase**.

Estas funciones se pueden activar directamente con una solicitud HTTP o bien cuando los servidores de **Google** detectan ciertos eventos preestablecidos y ejecutan la función correspondiente. Para nuestro proyecto, fue necesaria la creación de las dos funciones que se muestran en la figura 6.11. La primera de ellas, *moveDeviceMessage*, copia el mensaje de posición que llegó de **Sigfox** y lo almacena en una nueva tabla de *Realtime Database* dependiendo de su *Device ID*, por lo que el *trigger* que activa a esta función es la creación de una nueva entrada en la tabla **Devices**. Esto nos permite copiar los distintos mensajes que llegan a esta tabla **Devices** a tablas especificas para cada *dummy* dependiendo de su identificador.

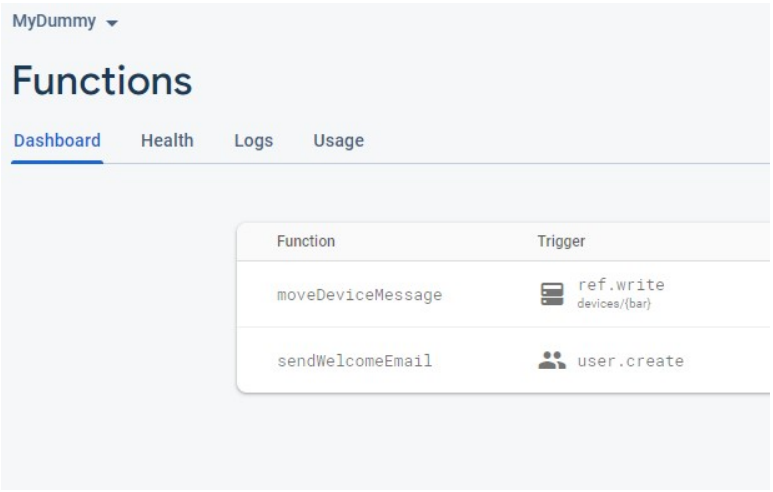


Figura 6.11: Cloud functions en Firebase

En el siguiente cuadro, se muestra el código para la función *moveDeviceMessage*:

```
const functions = require("firebase-functions");
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

exports.moveDeviceMessage =
    functions.database.ref('/devices/{bar}')
    .onWrite((change,context) => {
        const text = change.after.val();
        const data = {
          CountryCode: text.CountryCode,
          Data: text.Data, Device: text.Device,
          LinkQuaility: text.LinkQuality,
            Location:{
             lat : text.Location.lat, lng : text.Location.lng,
             radius : text.Location.radius,
             source :  text.Location.source,
             status : text.Location.status},
          OperatorName: text.OperatorName, Time: text.Time,
          altitude: text.altitude, latitude: text.latitude,
          longitude: text.longitude}
      var ref1 = admin.database()
          .ref("/"+text.Device+"/"+ text.Time)
      return ref1.set(data);
    }
});
```

Además, se ha definido una segunda función en otro contexto diferente, para completar el proceso de registro de un usuario en la aplicación. La función *sendWelcomeEmail* envía un mensaje de bienvenida al correo electrónico de un usuario cuando este se registra, por lo que *trigger* es la creación de un nuevo usuario. Este correo se enviará desde una cuenta **Gmail** propia del proyecto con el mensaje que aparece en la imagen 6.12. Se puede ver la implementación de dicha función en el anexo C.

Firebase ofrece a mayores la posibilidad de recuperar la contraseña de los usuarios en caso de que estos la hayan olvidado. Para ello, existen dos maneras de realizarlo: el administrador del sistema puede realizar la recuperación de la contraseña directamente desde la consola de **Firebase** o el usuario es el encargado de realizar esta petición directamente desde la aplicación móvil. Esta última opción se puede implementar mediante el uso de la función *sendPasswordResetEmail* de **FirebaseAuth**. En ese caso, nos llegará un correo similar al que se muestra en la imagen 6.13.

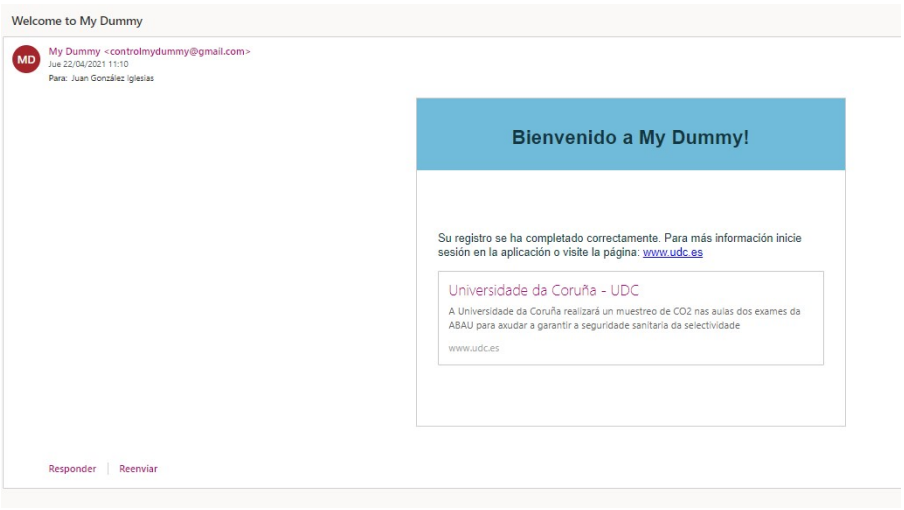


Figura 6.12: Correo electrónico bienvenida

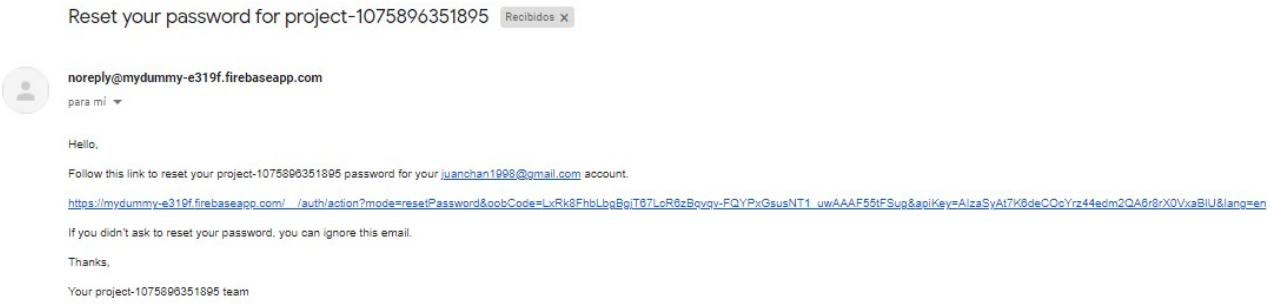


Figura 6.13: Correo electrónico para restablecer la contraseña.

6.4.2 Desarrollo de la aplicación móvil

Todo el desarrollo e implementación de las funcionalidades y casos de uso expuestos en los apartados de análisis (sección 6.2) y diseño (sección 6.3) fue realizado en el entorno de desarrollo de **Android Studio**, en la versión *4.2.1*. La versión mínima utilizada de la SDK para la *app* móvil fue la 21, que se corresponde con la versión de **Android 5.0 Lollipop**.

Un primer aspecto interesante a abordar son los permisos que requiere la aplicación para su completo funcionamiento. Entre ellos, destacan el permiso para activar *Bluetooth*, los permisos de *Camera* (para el lector QR), *Location* e *Internet*. Estos permisos se pueden establecer como se muestra en la figura 6.14 en el documento *Manifest.xml*.

Se hizo uso de la API de **Bluetooth** para **Android** [71] que incorpora la compatibilidad con Bluetooth Low Energy (BLE). Esta API permitió implementar toda la lógica correspondiente al escaneo de dispositivos, establecimiento de la conexión, y envío y recepción de datos mediante BLE. Además, se hizo uso de distintas librerías y servicios, expuestos en la

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
```

Figura 6.14: Permisos necesario para la aplicación móvil

sección 2.3.3, los cuales facilitaron la implementación de algunas de las funcionalidades de la aplicación. El resto de clases y recursos fueron creados siguiendo el patrón de diseño elegido para así conseguir una aplicación estable, funcional y que cumpla con los objetivos previamente definidos. A continuación, se van a comentar los aspectos más destacados del sistema incluyendo, para ello, capturas directamente extraídas de la aplicación final.

Estas pantallas no fueron expuestas en el apartado de *mockups* y vienen a completar el resto de funcionalidades. El resto de capturas sobre otros aspectos y pantallas de la aplicación se encuentran en el anexo D por completitud. Es importante destacar que la preferencia del modo oscuro de la *app* estaba activado a la hora de obtener las capturas.

La imagen 6.15 se corresponde con la visualización del sensor de inclinación del *dummy*. En la parte superior, tenemos los datos numéricos devueltos directamente por el *dummy* mapeados a valores entre -100 y 100 y un *TextView* indicando la posición del muñeco. En la parte central e inferior, se encuentran los umbrales establecidos por el administrador al que el usuario está vinculado y una gráfica 2D que muestra en tiempo real la posición de los ejes del muñeco.

Para la implementación de estas gráficas se hizo uso de la librería **GraphView**. Gracias a esta, podemos dibujar una gráfica 2D que nos permite representar de una manera visual la posición del *dummy* durante la realización de las diferentes técnicas y prácticas. Estos valores son actualizados en tiempo real, a partir de los datos obtenidos del servicio de BLE. Podemos ver el código necesario para la implementación de la librería **GraphView** en el anexo E.

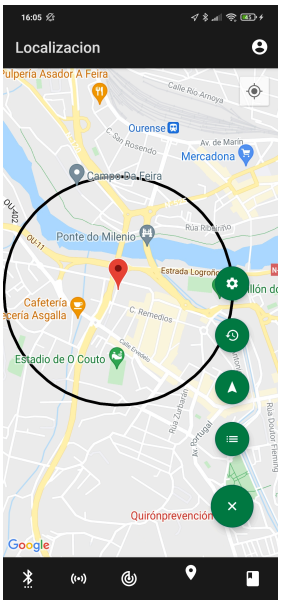
La pantalla de localización 6.16a muestra la última ubicación conocida del *dummy*, además de ofrecer la posibilidad de acceder al historial de localizaciones (permitiendo realizar búsquedas, filtrado y ordenación). A mayores, habilita opciones de empezar la navegación, modificar el rango de búsqueda (solo para el administrador) y mostrar las últimas 5 localizaciones.

Para poder visualizar esta localización en el mapa es necesaria la utilización del SDK de **Google Maps** para **Android**. En primer lugar, se debe configurar una API KEY de **Google Maps** en el *Manifest.xml*, para luego poder representar cualquier posición en el mapa a partir de sus coordenadas. En nuestro proyecto, este ubicación será obtenida mediante la base de datos **Firebase** a partir de los distintos mensajes del *dummy* llegados a través de la red *Sigfox*. En el siguiente código, podemos ver como se representa una ubicación en el mapa con el SDK

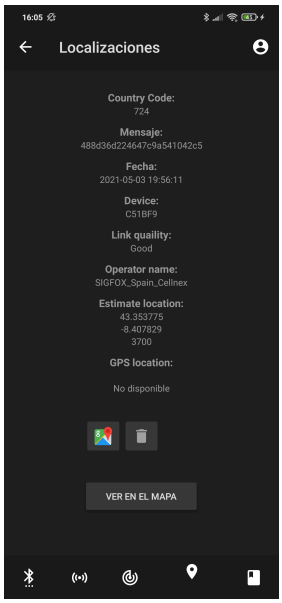
(a) Inclinado hacia la derecha.

(b) En posición horizontal.

Figura 6.15: Pantalla para el sensor de inclinación del *dummy*



(a) Mapa con el posible rango de búsqueda del dummy



(b) Información detallada de la localización.

Figura 6.16: Localización del dummy

de **Google Maps**:

```
private GoogleMap mMap;
SupportMapFragment mapFragment;

mapFragment = ((SupportMapFragment)
    getChildFragmentManager().findFragmentById(R.id.map));

@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    if (ActivityCompat.checkSelfPermission(getContext(),
    Manifest.permission.ACCESS_FINE_LOCATION) !=
    PackageManager.PERMISSION_GRANTED &&
    ActivityCompat.checkSelfPermission(getContext(),
    Manifest.permission.ACCESS_COARSE_LOCATION) !=
    PackageManager.PERMISSION_GRANTED) {
        return;
    }
    mMap.setMyLocationEnabled(true);
    LatLng ourense = new LatLng(42.34, -7.86464);
    mMap.addMarker(newMarkerOptions().position(ourense)
                    .title("Ourense"));
    mMap.moveCamera(CameraUpdateFactory
                    .newLatLngZoom(ourense,15.0f));
    mMap.addCircle(new CircleOptions().center(sydney)
                    .radius(500)
                    .fillColor(Color.TRANSPARENT));
    }
```

En la figura 6.16b, se muestra la pantalla para la visualización de los detalles de un mensaje de posición. En ella, vemos de forma ordenada la información que se almacenó en **Firebase**. En este ejemplo, no disponemos de la información obtenida por el módulo GPS, pero si que podemos hacer uso de la API de geolocalización de **Google** (por lo que solo nos aparece el botón de la geolocalización) a partir de las dos MACs enviadas en el *payload* en formato hexadecimal, y así poder localizar de forma precisa al *dummy*. También ofrece la posibilidad al usuario de eliminar dicho mensaje de la base de datos de **Firebase**.

Para poder convertir las dos direcciones MAC del *payload* del mensaje y que estas sean utilizadas por la API de **Google Maps** se ha implementado el siguiente código:

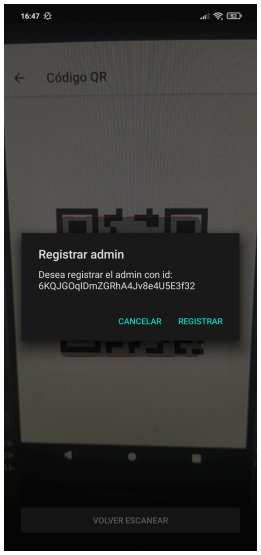
```
String half1 = data.substring(0, data.length() / 2);
String half2 = data.substring(data.length()/2);

char divisionChar = ':';
```

```
formattedMAC = half1.replaceAll("(.{2})",
    "$1"+divisionChar).substring(0,17);
formattedMAC2 = half2.replaceAll("(.{2})",
    "$1"+divisionChar).substring(0,17);
```

Una vez realizada esta conversación y obtenidas las dos MAC en el formato correcto, podemos hacer uso de la API de geolocalización de **Google Maps** y obtener la posición geográfica del *dummy*. Para acceder a dicha API, utilizamos las librerías **OkHttp**, **Retrofit** y **Gson**, de las que podemos ver más detalles e implementación en el anexo F.

Un usuario puede agregar o vincular un administrador directamente desde su aplicación. Los usuarios administradores, al acceder a su perfil, tienen la opción de visualizar su código QR personal (figura 6.17a), mientras que los usuarios normales pueden escanear este código para registrar dicho administrador (figura 6.17b), y así importar los umbrales de los sensores y rangos de búsqueda que este tenga establecidos. Para proporcionar esta funcionalidad se hizo uso de la librería **ML kit** de **Google** y de la librería **CameraX**.

(a) QR del administrador.

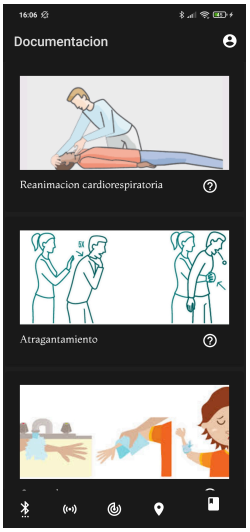


(b) Lector de QR del usuario.

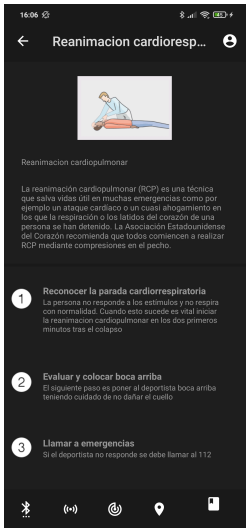
Figura 6.17: Vinculación de un administrador a un usuario

Por otro lado, como ya se ha mencionado, se han incluido una serie de guías o manuales de primeros auxilios que pueden dotar al usuario de una serie de nociones básicas para la realización de las técnicas primeros auxilios más habituales. Estás guías se guardan en una base de datos **Sqlite** y accedemos a ellas a través de la librería de **Room**. Podemos ver una implementación para recuperar esta información con esta librería en el anexo G. Las técnicas o guías incluidas son: *Reanimación cardiorespiratoria*, *Atragantamientos*, *Quemaduras*, *Heridas abiertas*, *Hemorragias*, *Traumatismos*, *Reacción alérgica*, *Picaduras*, *Mordeduras* y *Convulsiones*.

En la figura 6.18, se puede ver un ejemplo de la pantalla en la aplicación para visualizar estas técnicas básicas.



(a) Lista de las distintas guías sanitarias.



(b) Guía reanimación cardiorespiratoria.

Figura 6.18: Apartado de documentación de la aplicación móvil.

Por último, comentar la importancia de proporcionarle al usuario una breve guía sobre la aplicación para mostrarle así su funcionamiento y características básicas. Esta se mostrará la primera vez que el usuario inicie la aplicación, gracias al uso de las *SharedPreferences*. En la figura 6.19, se muestra un par de ejemplos de la información que aparece en esta *overview*.

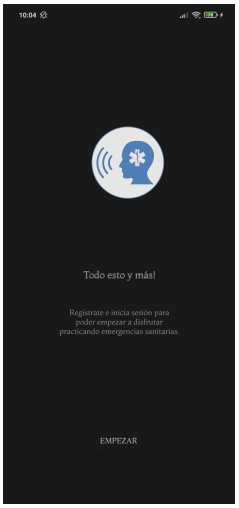


Figura 6.19: *Overview* de las características de la aplicación

Capítulo 7

Pruebas realizadas

PARA comprobar el correcto funcionamiento de todo el proyecto, debemos realizar una serie de pruebas tanto en la aplicación móvil como en toda la parte del hardware. Además, es necesario verificar que el sistema completo funciona según lo esperado cuando se integran ambas partes y, a mayores, sería muy interesante probar el sistema completo en un entorno real de simulación.

7.1 Pruebas aplicación móvil

Las diferentes pruebas realizadas en la aplicación móvil fueron sencillas y básicas, con el objetivo de encontrar posibles fallos o mejorar ciertos aspectos de diseño o determinadas funcionalidades. Estas pruebas normalmente se pueden clasificar en dos tipos:

- **Pruebas de caja negra**: son aquellas pruebas que analizan el comportamiento del sistema en función de las entradas que recibe y las salidas o respuestas que este produce. Normalmente son realizadas directamente desde la propia interfaz gráfica de la aplicación.

  En este proyecto, se escogieron aquellas funciones que podrían producir algún error debido a su complejidad, como pueden ser pruebas de lectura de la base de datos de **Firebase**, conexión/desconexión mediante **ble** con el *dummy*, registro de nuevos usuarios, vinculación de un usuario a un administrador e importación de sus umbrales, ... Estas funciones se han probado paso a paso, intentado comprobar toda la casuística de las mismas para así poder dar solución a los errores encontrados o mejorar dichas funcionalidades.

- **Pruebas de caja blanca**: también conocidas como *clear box testing*, utilizan el conocimiento aprendido a partir del código y la estructura de la aplicación para examinar las partes o módulos aparentemente más problemáticos.

En nuestro caso, este tipo de pruebas fueron realizadas a partir de los módulos o partes marcados como erróneos en las **pruebas de caja negra**, analizando y estudiando el código para entender mejor el funcionamiento y así poder aplicar los cambios necesarios.

7.2 Pruebas parte hardware

En las primeras fases del proyecto, se llevaron a cabo **pruebas de funcionalidad** de cada uno de los módulos hardware de manera individual para así comprobar su correcto funcionamiento. Entre ellas, destacan las pruebas realizadas a los módulos de **Sigfox** (figura 7.1) y GPS para la correcta transmisión de datos y localización del *dummy*, así como las pruebas realizadas a todos los sensores de forma individual para la correcta obtención e interpretación de los datos que proporcionaban (figura 7.2).

Una vez realizadas estas pruebas individuales y después de integrar todos los componentes en el sistema hardware, se realizaron **pruebas de compatibilidad** y pruebas generales del funcionamiento completo de toda la parte **hardware**. Entre estas pruebas, se verificó la obtención de las dos MAC, la posición del *dummy* a partir del módulo GPS o la codificación y envío de los distintos datos a través de la red **Sigfox**.

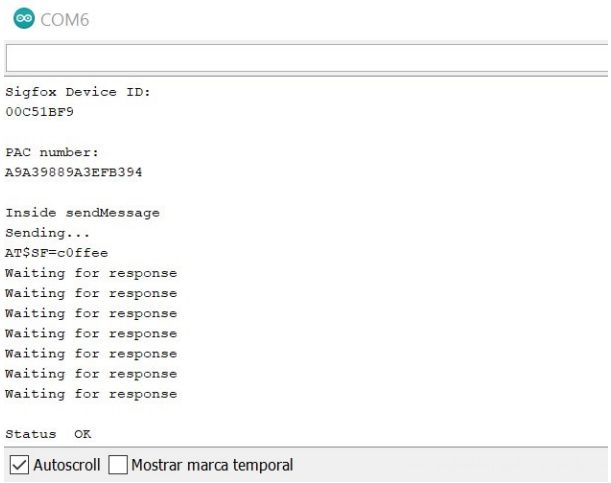


Figura 7.1: Pruebas realizadas con el módulo **Sigfox**.

7.3 Pruebas de integración

Después de haber realizado las diferentes pruebas unitarias en ambas partes del proyecto, es el turno ahora de realizar **pruebas de integración**. Estas pruebas consisten básicamente en valorar si los componentes individuales trabajan en conjunto tal y como se espera de ellos,

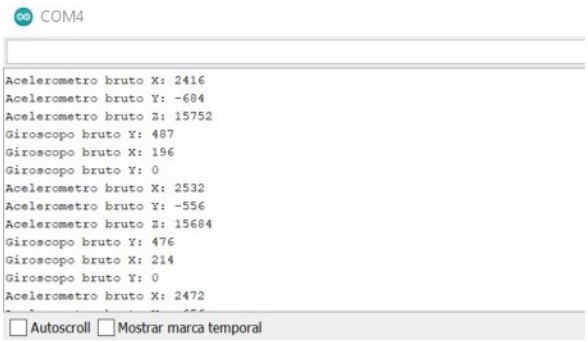


Figura 7.2: Pruebas realizadas de inclinación-aceleración.

centrándose principalmente en probar la comunicación entre los distintos componentes y sus interacciones.

En este proyecto, las pruebas de integración realizadas se centraron sobre todo en la comunicación mediante el protocolo BLE. Este protocolo establece la conexión entre el *dummy* y la aplicación móvil para el envío bidireccional de diferentes datos de monitorización y control, por lo que su correcto funcionamiento es fundamental. Estas pruebas consistieron básicamente en la conexión-desconexión de la aplicación con el *dummy* así como pruebas de alcance y controlar posibles fallos o errores. Además, se comprobó que los datos eran recuperados correctamente desde los sensores y visualizados en las diferentes pantallas de la aplicación móvil. Otro aspecto importante en este conjunto de pruebas fue asegurar la correcta interacción entre **Sigfox**, **Firebase** y nuestra aplicación.

Una vez realizadas estas pruebas y teniendo el sistema completo funcionando correctamente, se llevaron a cabo **pruebas de aceptación** que se centran en determinar si el sistema o proyecto cumple con las necesidades y/o requerimientos del cliente o usuarios. Básicamente, en nuestro proyecto, comprobamos que todos los requerimientos y funcionalidades diseñadas en las primeras etapas del ciclo de vida fueron conseguidas y eran del agrado de un futuro cliente como es la empresa **EFESGAL**.

7.4 Pruebas reales

Para probar el proyecto completo y sus diversas funcionalidades en un entorno real, se contó con la colaboración de la empresa **EFESGAL**. Esta empresa con sede en Ourense está especializada en la formación en emergencias y salud, impartiendo múltiples cursos y actividades de formación en los diferentes ámbitos sanitarios. Gracias a esta colaboración y asesoramiento, fuimos capaces de ir puliendo y mejorando el sistema a medida que discurría el ciclo de vida de este. Se fueron realizando diversas pruebas que incluían ver el funcionamiento del sistema desarrollado para algunas de las técnicas de rescate más conocidas con el material sa-

nitario adecuado. Algunas de estas técnicas de inmovilizado y traslado fueron: *carga bombero*, *arrastre pies*, *arrastre de bombero*, *puente holandés*, *colocación de collarín*, *colocación de sling*, *colocación de férula*, ... A mayores, se realizaron pruebas de rescate y búsqueda en escenarios reales, como la realizada con la Brigada de Refuerzo en Incendios Forestales (BRIF) de **Laza** (figura 7.3).

Figura 7.3: Práctica realizada con la brigada BRIF

Capítulo 8

Conclusiones

Tras la realización del proyecto, procedemos a explicar la situación final del trabajo, las lecciones aprendidas, un breve análisis de las posibles líneas futuras y la relación que este proyecto tiene con las competencias de las menciones de Ingeniería de Computadores y de Ingeniería de Tecnologías de la información.

8.1 Conclusiones

Llegados a este punto del proyecto y comparando los objetivos iniciales que se plantearon con los que se han conseguido al final, podemos decir que estos se cumplieron, consiguiendo desarrollar e implementar todo el conjunto de funcionalidades y características previstas. El objetivo principal consistía en desarrollar una herramienta capaz de mejorar y facilitar el aprendizaje de las técnicas más importantes de inmovilización y rescates sanitarios. Como se ha podido comprobar a lo largo de esta memoria, podemos concluir que este objetivo se ha cumplido y se ha conseguido desarrollar un sistema funcional que integra una parte hardware de monitorización muy completa junto con una parte software para aprovechar todo esa información que nos proporciona el componente hardware.

A mayores, se definieron una serie de funcionalidades o características básicas que eran las siguientes:

- **Sensorización completa del dummy**. Para aportar un correcto nivel de información en la monitorización, fue necesario implementar una serie de sensores y módulos hardware sobre el *dummy*. Esta monitorización consistió principalmente en controlar la inclinación, la aceleración, la fuerza o presión ejercida en diversos puntos del muñeco, la temperatura y la humedad. Esto se consiguió añadiendo los respectivos sensores para medir estos valores, todos ellos conectados a una placa de desarrollo, en nuestro caso la **Arduino Mkr Wifi 1010**.

Además, para completar las posibilidades de nuestro *smart dummy* en el ámbito de las emergencias sanitarias, se le dotó de la capacidad de transmitir información sobre su última posición, para así poder realizar prácticas de rescate con él. Mediante la utilización de la tecnología **Sigfox** y la integración de un módulo GPS y una API de geolocalización de **Google**, fuimos capaces de cumplir este objetivo. Estos mensajes de localización se guardan en una base de datos *online*, en **Firebase**, para que cualquier usuario pueda acceder a sus respectivos datos desde cualquier dispositivo.

Por último, se decidió añadir un par de altavoces para permitir la reproducción de diversos sonidos, además de dotar al maniquí de un sistema de avisos o alertas, informando de si algún movimiento o técnica es realizada de forma incorrecta.

Todo esta sensorización fue incluida en el interior del *dummy*, soldada a mano en una PCB y montada en un armazón proporcionando, de esta forma, un mayor nivel de seguridad y robustez al sistema. Esto permite además que las principales técnicas y movimientos en las actividades se pueda seguir realizando de la misma manera, sin causar ningún tipo de molestia a los usuarios.

- **Diseño y creación de una aplicación móvil**. La principal función de la aplicación móvil es poder controlar y monitorizar todos los datos devueltos por los sensores y módulos integrados en el *dummy*. Para ello, se desarrolló una aplicación **Android** en Java que ofrece la posibilidad de visualizar intuitivamente y de diferentes formas los datos asociados a la monitorización del maniquí.

  Se incorporó también la conexión de esta aplicación con el *dummy* mediante el protocolo BLE, proporcionando así la capacidad de interconexión, control y transmisión de los datos con el *dummy*.

  La utilización de **Firebase** en el proyecto, aparte de permitir acceder a la información de las posiciones devueltas por el *dummy*, nos permite llevar a cabo un correcto control en la gestión de usuarios a mayores de poder guardar diferente información acerca de ellos. La correcta interpretación y visualización de la posición del *dummy* se consigue a través de la API de **Google Maps**, permitiendo obtener esta información de tres posibles maneras: mediante la estimación realizada por **Sigfox**, mediante los datos devueltos por el módulo GPS o mediante las MAC obtenidas por el hardware del *dummy* utilizando, para ello, la API de geolocalización de **Google**.

  Uno de los objetivos del proyecto era familiarizarse con las técnicas y acciones requeridas en un rescate, por lo que para conseguir esto, aparte de hacer un estudio y análisis de las mismas, se integró un apartado de documentación que contiene las técnicas o guías más básicas de primeros auxilios y rescate.

- **Realización de pruebas**. Las pruebas realizadas del sistema fueron en coordinación con la empresa **EFESGAL** que está especializada en formación de emergencias y salud. En este sentido, a aparte de aportar el conocimiento básico en cuanto a las técnicas de inmovilizado y rescate, proporcionó una visión desde un punto empresarial y profesional del sistema.

Teniendo en cuenta todo esto, podemos entonces concluir que los objetivos iniciales de este proyecto se cumplieron con creces y que el resultado final puede ser utilizado para mejorar el aprendizaje y formación en el ámbito de las emergencias sanitarias. Además, dada la creciente demanda de cursos y formación relacionada con este ámbito, y que no existen en el mercado herramientas similares al *smart dummy* creado en este proyecto, hacen de esta una opción muy atractiva para su puesta en producción y comercialización.

Desde el punto de vista personal, me he sentido muy realizado por poder crear algo prácticamente desde cero que busca cubrir una necesidad y que integra las dos partes fundamentales de la informática: hardware y software. Además, tras haber completado este proyecto aprendí el funcionamiento y uso de varias tecnologías que me pueden resultar muy útiles e interesantes de cara al mercado laboral como es, por ejemplo, **Firebase**, el protocolo BLE o diferentes tecnologías para el desarrollo completo de una aplicación móvil. Finalmente, ha sido muy gratificante abordar un proyecto tan complejo y de cierta envergadura como este desde el principio, pasando por todas las fases necesarias para su desarrollo.

8.2 Relación de las competencias de las titulaciones

Debido a la realización de una doble mención, es necesario dividir el proyecto en dos grandes partes. La primera es el desarrollo de un sistema completo de monitorización y comunicación basado en el uso de diferentes sensores y módulos hardware. Esta parte se corresponde con la mención en Ingeniería de Computadores, la principal. El sistema hardware desarrollado guarda una estrecha relación con las principales competencias de esta mención, pues se llevó a cabo un **diseño, implementación y programación de una serie de sistemas hardware y de sensorización**. A mayores, se incluyó una tecnología para redes de comunicación, como es **Sigfox** y la tecnología **BLE** para el intercambio de datos con la aplicación móvil, por lo que hace referencia a la competencia de **diseñar y programar sistemas empotrados basados en microprocesadores y de comunicaciones**.

La otra parte del proyecto consiste en el análisis, diseño e implementación de una aplicación móvil para dispositivos **Android** que recibirá, mostrará y controlará los datos recibidos por parte del *dummy*. Esta parte se corresponde con la mención de Ingeniería de Tecnologías de la información y se trabaja con competencias muy relacionadas con el **diseño, implementación, integración, gestión y mantenimiento de tecnologías *software***.

Resulta fundamental resaltar la importancia de la cooperación y relación entre las dos partes, operando estas de forma conjunta, pues ambos sistemas tienen que comunicarse entre ellos y compartir recursos para lograr el objetivo fundamental de este proyecto, que el usuario final pueda utilizar esta herramienta como medio de formación en el ámbito de las emergencias sanitarias.

8.3 Lineas futuras

Aunque la totalidad de los objetivos iniciales han sido cumplidos, no se han podido desarrollar algunos aspectos a mayores por falta de tiempo y que podían haber sido interesantes incorporar al resultado final. Entre ellos, se puede destacar:

- Desarrollar la aplicación móvil para dispositivos **IOS**. Al elegir como lenguaje de desarrollo **Java**, perdemos la posibilidad de realizar una aplicación *cross-platform*, por lo que resultaría interesante probar e implementar dicha aplicación en otros lenguajes como **Kotlin** o con **Flutter**.
- Permitir a un administrador la visualización y control de los sistemas de monitorización de sus alumnos vinculados. Esto permite que el profesor o monitor encargado de una clase, pueda ver en tiempo real la evolución de sus alumnos a la hora de realizar cualquier práctica o entrenamiento. Esto requiere también la necesidad de disponer de múltiples *dummis* configurados correctamente.
- Posibilidad de añadir algún mecanismo de sensorización al *dummy* que permita la realización de técnicas de compresión de pecho, como es el caso de la RCP.
- Dotar de un mayor nivel de seguridad al sistema. Por ejemplo, haciendo un mejor uso de las reglas de seguridad de **Firebase**.
- En lugar de conectar y soldar manualmente todos los sensores y módulos hardware, diseñar y construir una PCB impresa que ya incorpore todos los elementos y conexiones necesarias, facilitando así también su posible puesta en producción.

Apéndices

Apéndice A

Envío de datos a través de la red Sigfox

En este apéndice se muestra un ejemplo del envío de un mensaje con el contenido en hexadecimal *COFFEE* a través de la red **Sigfox** desde la placa **Arduino Mkr Wifi 1010**. Para ello, se ha utilizado el módulo **Sigfox** *BRKWS01-RC1* conectado a la placa a través de los pines 13 y 14 (TX/RX).

```
Serial1.begin(9600);

String id = "";
char output;

Serial1.print("AT$I=10\r");

while(Serial1.available()){
    output = Serial1.read();
    id += output;
    delay(10);
}

Serial.println("Sigfox Device ID: ");
Serial.println(id);


String pac = "";
char output;

Serial1.print("AT$I=11\r");

while(Serial1.available()){
    output = Serial1.read();
```

```
    pac += output;
    delay(10);
}

Serial.println("PAC number: ");
Serial.println(pac);

String status = "";
String hexChar = "";
String sigfoxCommand = "";
char output;

sigfoxCommand += "AT$SF=";

uint8_t msg[12];

msg[0]=0xC0;
msg[1]=0xFF;
msg[2]=0xEE;

for (int i = 0; i < size; i++)
  {
    hexChar = String(msg[i], HEX);

    //padding
    if (hexChar.length() == 1)
    {
      hexChar = "0" + hexChar;
    }

    sigfoxCommand += hexChar;
  }

Serial1.println(sigfoxCommand);
```

Apéndice B

Diseño de interfaces (continuación)

En este apéndice, se completan los *mockups* realizados para el diseño de las pantallas de la aplicación móvil. La primera imagen (figura B.1a) se corresponde con la pantalla de carga de la aplicación, denominada normalmente *splashscreen*, la cual desaparece nada más llegar la aplicación principal a memoria. En esta pantalla solo se muestra el logotipo de la aplicación. La imagen B.1b se corresponde con la primera pantalla que ve el usuario después de pasar la *splashscreen* y el *overview* en el caso de que sea la primera vez que se inicia la aplicación. En esta pantalla, el usuario puede encontrar las opciones tanto de iniciar sesión como de registrarse.

(a) *Splashscreen*

(b) Pantalla de bienvenida

Figura B.1: Pantallas de bienvenida y *splashscreen*.

Como se puede apreciar en la pantalla de registro (figura B.3), el usuario debe introducir su nombre completo, su correo electrónico, su provincia e introducir dos veces la contraseña, para así comprobar que se introduce de manera correcta. Por otro lado, en la pantalla de inicio de sesión (figura B.2), es necesario introducir el correo electrónico del usuario y su contraseña. Además, se le ofrece al usuario la posibilidad de recuperar su contraseña, introduciendo su correo electrónico.

Figura B.2: Login

Figura B.3: Registro

Una vez que un usuario inicia sesión, este puede ver su perfil. Si el usuario que inició sesión es un usuario **normal**, verá la pantalla B.4 , mientras que si es un usuario **administrador** verá la pantalla B.5 con la opción de visualizar y mostrar su código QR. En estas pantallas de perfil, los usuarios podrán editar los datos de su cuenta y cerrar sesión.

A mayores, en la barra superior existe un desplegable (figura B.6) que permite a los usuarios visualizar los dispositivos que tienen registrados, poder vincular a un administrador si se trata de un usuario normal, modificar las preferencias de usuario y contactar con el equipo de desarrollo para cualquier duda, cuestión o mejora.

Desde la pantalla principal, el usuario puede además navegar, a través de un menú inferior, por las diferentes pantallas de conexión BLE (figura B.7), visualización de sensores individuales (figura B.8), vista general de monitorización y vista de la posición del *dummy* como se vió en la sección 6.3.4 y, por último, acceso a la documentación (figura B.9). Cada una de estas pantallas ofrece una serie de opciones para acceder a otras funcionalidades que se pueden ver con más detalle en la sección de implementación (sección 6.4).

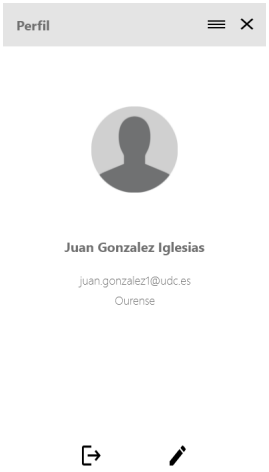


Figura B.4: Pantalla perfil normal.

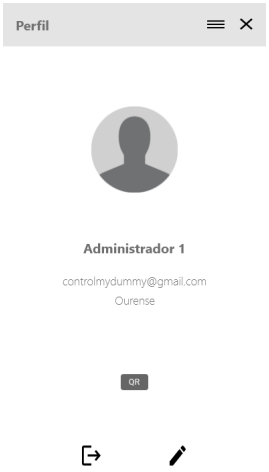


Figura B.5: Pantalla perfil administrador.

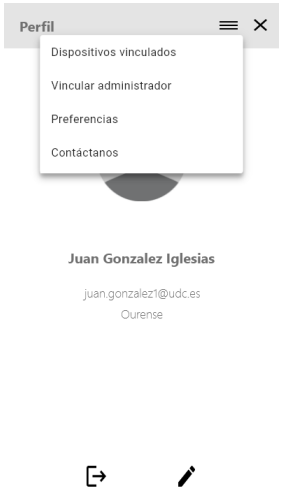


Figura B.6: Desplegable opciones perfil.

Figura B.7: Bluetooth

Figura B.8: Lista Sensores

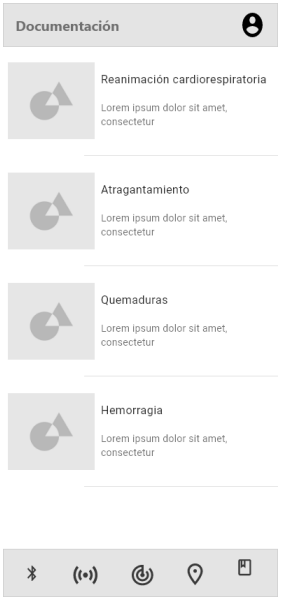


Figura B.9: Documentación

Apéndice C

Programación de las Cloud Functions de Firebase

MEDIANTE el uso de las *Cloud Functions* de **Firebase**, vamos a ser capaces de clasificar los mensajes de ubicación en la base de datos *RealTime* de **Firebase**, procedentes del *dummy*, en función del identificador del dispositivo *Sigfox* que los envió, como ya vimos en la sección 6.4.1.

Por otro lado, las *Cloud Functions* nos dan la oportunidad de enviar un correo de confirmación o bienvenida cada vez que un usuario se registra en nuestra aplicación. Para ello, debemos crear una función en lenguaje JavaScript (JS) para luego desplegarla en nuestro proyecto de **Firebase**. El código correspondiente a esta función se muestra a continuación. Como se puede ver, se utiliza el módulo *nodemailer* para enviar dicho mensaje, definiendo el contenido del mismo en formato HTML:

```
const functions = require("firebase-functions");
const nodemailer = require('nodemailer');

const mailTransport = nodemailer.createTransport("SMTP",{
  host: 'smtp.gmail.com',
  service: 'gmail',
  port: 465,
  secure: true,
  auth: {
        user: 'controlmydummy@gmail.com',
        pass: 'XXXXX'
    }
});



exports.sendWelcomeEmail = functions.auth.user().onCreate((user) =>
```

```
    {
18  const recipent_email = user.email;
19
20  const mailOptions = {
21      from: 'My Dummy <controlmydummy@gmail.com>',
22      to: recipent_email,
23      subject: 'Welcome to My Dummy',
24       html:
25       `<body style="margin: 0; padding: 0;">
26          <table border="0" width="100%"
27                  cellspacing="0" cellpadding="0">
28          <tbody>
29              <tr>
30              <td style="padding: 10px 0 30px 0;">
31                  <table style=
32                  "border: 1px solid #cccccc;
33                  border-collapse: collapse;"
34                  border="0"
35                  width="600"
36                  cellspacing="0"
37                  cellpadding="0" align="center">
38                  <tbody>
39              <tr>
40              <td style=
41                  "padding: 40px 0 30px 0;
42                  color: #153643;
43                  font-size: 28px;
44                  font-weight: bold;
45                  font-family: Arial,
46                  sans-serif;"
47                  align="center"
48                  bgcolor="#70bbd9">
49                  Bienvenido a My Dummy!</td>
50              </tr>
51              <tr>
52                  <td style=
53                  "padding: 40px 30px 40px 30px;"
54                  bgcolor="#ffffff">
55                      <table style=
56                          "height: 124px;"
57                          border="0"
58                          width="552"
59                          cellspacing="0"
60                          cellpadding="0">
61                          <tbody>
62                          <tr>
```

```
                                <td style=
                                    "color: #153643;
                                    font-family: Arial, sans-serif;
                                    font-size: 24px;
                                    width: 548px;"> 
                                </td>
                                </tr>
                                    <tr>
                                    <td style=
                                    "padding: 20px 0px 30px;
                                    color: #153643;
                                    font-family: Arial, sans-serif;
                                    font-size: 16px;
                                    line-height: 20px;
                                    width: 548px;">
                                    Su registro se ha completado
                                    correctamente. Para m&aacute;s
                                    informaci&oacute;n inicie
                                    sesi&oacute;n en la
                                    aplicaci&oacute;n o visite
                                    la p&aacute;gina:
                                    <a href="http://www.udc.es">
                                        www.udc.es</a>
                                    </td>
                                    </tr>
                                    </tbody>
                            </table>
                        </td>
                        </tr>
                    </tbody>
                </table>
            </body>` // email content
        };

        return mailTransport.sendMail(mailOptions, (error, data) => {
            if (error) {
                console.log(error)
                return
            }
            console.log("Sent!")
          });
        });
```

Apéndice D

Navegación y pantallas de la aplicación

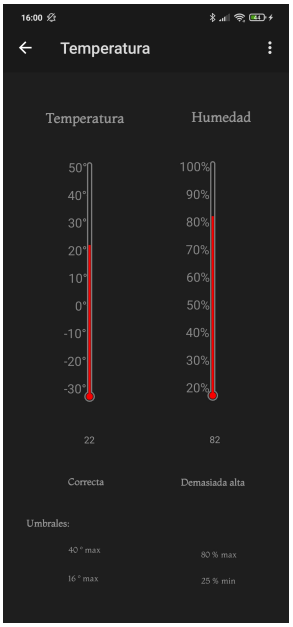
En este apéndice, se muestra una lista de capturas extraídas directamente de la aplicación móvil para completar las presentadas durante la memoria. En la imagen D.1a, podemos ver la representación del sensor de temperatura y humedad. Del mismo modo que para el sensor de inclinación, podemos ver directamente tanto el valor numérico devuelto por el *dummy*, como una representación gráfica y un *TextView* que nos indica si los valores son correctos. En la imagen D.1b, se muestra la pantalla de monitorización completa que presenta, conjuntamente y de una manera muy visual, todos los datos devueltos por el *dummy*. Además, permite la reproducción de una serie de sonidos en el *dummy* y da la posibilidad al usuario de iniciar la simulación de un rescate.

Las pantallas de los sensores de inclinación y aceleración proporcionan al usuario información de si se están realizando los movimientos o técnicas correctamente. La imagen D.2a muestra como se está realizando un movimiento correcto en el *dummy*, mientras que la imagen D.2b muestra que se está produciendo un movimiento demasiado rápido en el *dummy*.

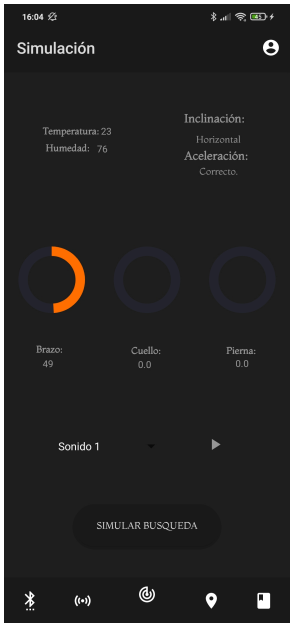
La pantalla de bienvenida (figura D.3) se muestra si el usuario no tiene aún iniciada la sesión en el dispositivo. En caso contrario, se accede directamente a la pantalla de monitorización completa del *dummy*. A mayores, en la figura D.4 se muestran las pantallas de *login* (figura D.4a) y registro (figura D.4b) en la aplicación.

La siguiente imagen D.5a muestra la diferentes opciones o ajustes que se pueden realizar desde la pantalla de perfil de un usuario, mediante el desplegable que se encuentra en la barra de navegación superior. Entre estas opciones se encuentran:

1. Ver los dispositivos registrados para un usuario junto con su información asociada (figura D.5b). Esta información se muestra en un *ListView* con la información extraída de la base de datos personal del usuario de **Firebase**.

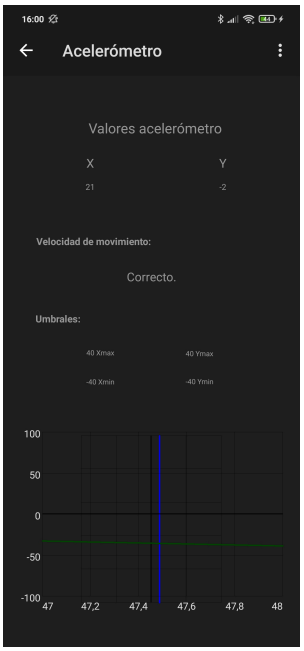


(a) Pantalla de temperatura y humedad



(b) Pantalla monitorización completa.

Figura D.1: Pantallas de monitorización del *dummy*.



(a) Movimiento correcto.

(b) Movimiento rápido.

Figura D.2: Pantalla sensor de aceleración del *dummy*

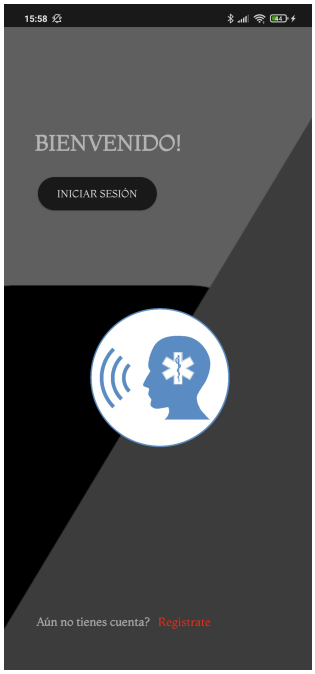
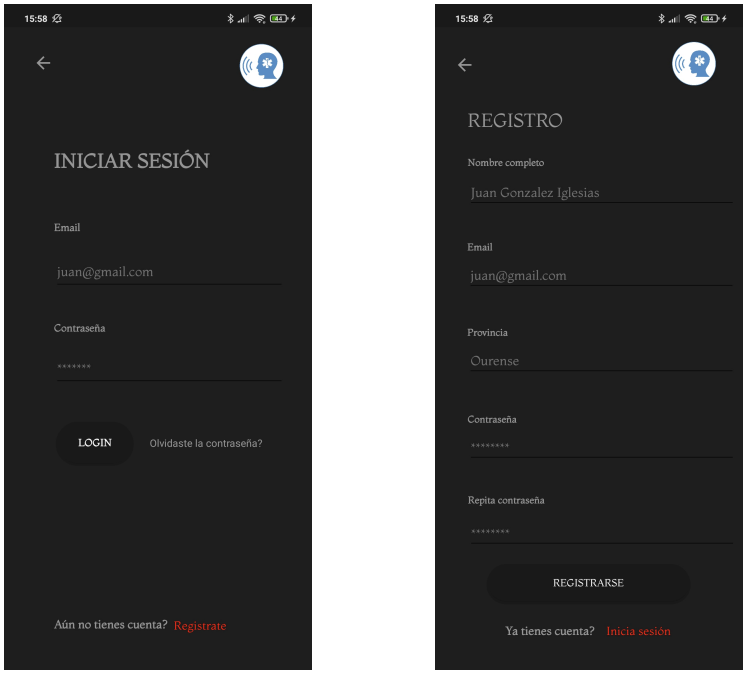


Figura D.3: Pantalla de bienvenida.



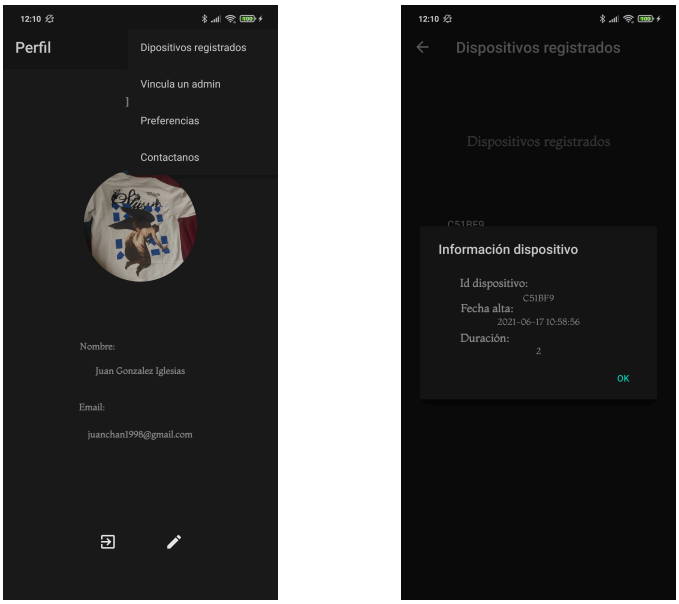
(a) Pantalla de *login* (b) Pantalla de registro

Figura D.4: Pantallas de *login* y registro.

2. Vincular un administrador (en el caso de tratarse de un usuario normal) (figura 6.17b). Esto se puede realizar mediante el escaneo del código QR del administrador.

3. Modificar las preferencias o configuración de la aplicación (figura D.6a). Estas opciones se guardaran en las *SharedPreferences* y puede ser silenciar/desilenciar el *dummy*, y establecer o no el modo oscuro.

4. Contactar con el equipo de desarrollo (figura D.6b). Nos llevará a la aplicación de correo electrónico que el usuario elija para enviar el mensaje.



(a) Opciones perfil de usuario.

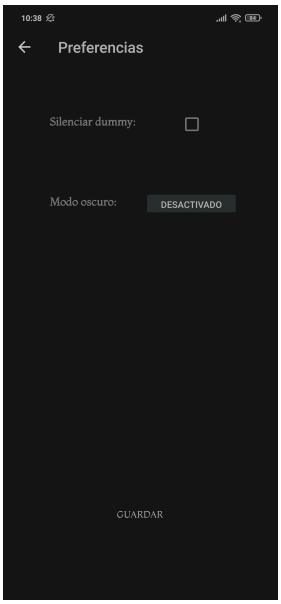
(b) Dispositivos registrados.

Figura D.5: Desplegable pantalla perfil usuario y vincular *dummy*

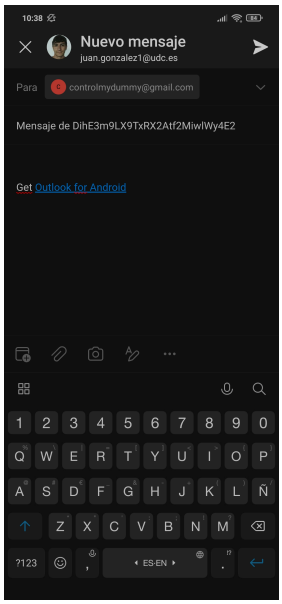
Los administradores tienen la opción de modificar los valores de los umbrales de los diferentes sensores (figura D.7a), y también de modificar el rango de búsqueda para la simulación de un rescate (figura D.7b).

La aplicación móvil ofrece la posibilidad tanto de escanear los dispositivos BLE (figura D.8a) en busca del *dummy* como de establecer la conexión con este (figura D.8b), mostrando las principales características de este enlace.

Por último, podemos ver la lista de localizaciones (figura D.9a) guardadas por las que pasó un *dummy*, en la que se puede tanto consultar los datos de cada posición como realizar una búsqueda por fecha, ordenar dicha lista en función de la fecha del mensaje de ubicación (figura D.9b) o filtrar por *dummy*.

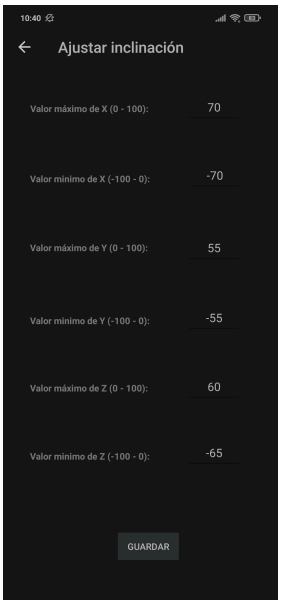


(a) Pantalla de cambio de preferencias.

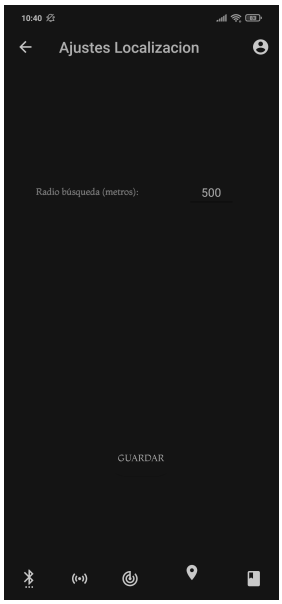


(b) Contactar con el equipo de desarrollo.

Figura D.6: Modificar preferencias y contactar con el equipo de desarrollo.

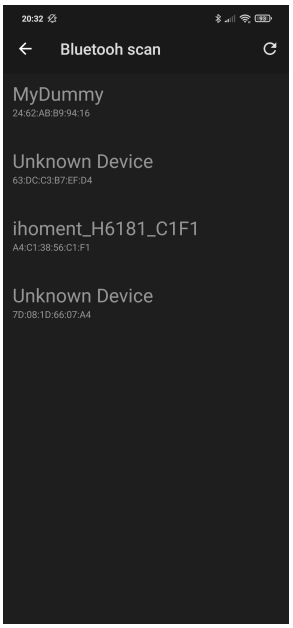


(a) Modificación umbrales inclinación.

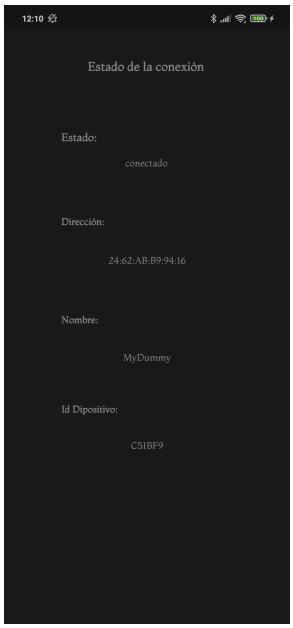


(b) Modificación rango de búsqueda.

Figura D.7: Pantallas de modificación de parámetros por parte del administrador.



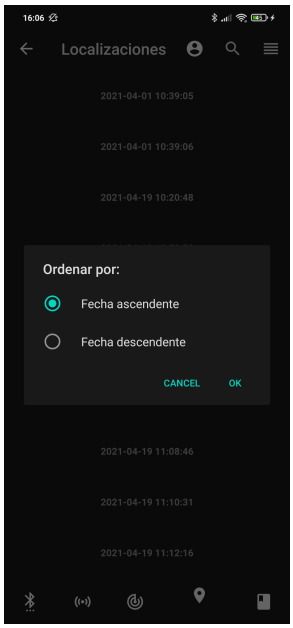
(a) Ejemplo de dispositivos BLE disponibles.



(b) Conexión establecida con el *dummy*

Figura D.8: Pantallas de conexión BLE de la aplicación móvil.

(a) Lista localizaciones por la que pasó un *dummy*.



(b) Ordenar la lista de localizaciones.

Figura D.9: Pantalla del historial de localizaciones de un *dummy*.

Apéndice E

Representación de gráficos 2D en Android

Sencillo ejemplo en el que se va a crear una gráfica 2D con la librería **GraphView**. En esta gráfica, se representa los valores para el eje X actualizándose constantemente para simular los datos en tiempo real aportados por el servicio BLE de la aplicación.

```
GraphView graphView;
private LineGraphSeries<DataPoint> series_acx;
private static double currentX;

graphView = findViewById(R.id.idGraphView);

series_acx = new LineGraphSeries<>();
series_acx.setColor(Color.GREEN);
graphView.addSeries(series_acx);

// activate horizontal zooming and scrolling
graphView.getViewport().setScalable(true);

// activate horizontal scrolling
graphView.getViewport().setScrollable(true);

// activate horizontal and vertical zooming and scrolling
graphView.getViewport().setScalableY(true);

// activate vertical scrolling
graphView.getViewport().setScrollableY(true);

// set manual Y bounds
graphView.getViewport().setYAxisBoundsManual(true);
graphView.getViewport().setMinY(-110);
```

```
graphView.getViewport().setMaxY(110);

for (int i = 0; i < 100; i++) {
 series_acx.appendData(new DataPoint(currentX++,
    valorARepresentar), true, 40);
}
```

Apéndice F

OkHttp, Retrofit y Gson

OkHttp es una librería creada por **Square**, diseñada para actuar como un cliente HTTP, por lo que permite enviar y recibir solicitudes HTTP de una forma sencilla. Normalmente, en **Android**, se utiliza conjuntamente con **Retrofit**, la cual actúa como un cliente de tipo *REST*, facilitando así la comunicación por ejemplo con distintas APIs. En **Retrofit**, se configura además un convertidor para la serialización de los datos. Este convertidor va a ser **Gson**, el cual nos va a permitir convertir tanto objetos **Java** a JSON, como al contrario.

En nuestro proyecto, estas tres herramientas las utilizamos para acceder a la API de geolocalización de **Google Maps**. A esta API le tenemos que pasar dos direcciones MAC para que nos devuelva la posición en la que se encuentra el *dummy*. Esta API requiere de una autenticación mediante una *API KEY* por lo que es necesario declarar un interceptor. En la siguiente figura se muestra un diagrama que ilustra este proceso.

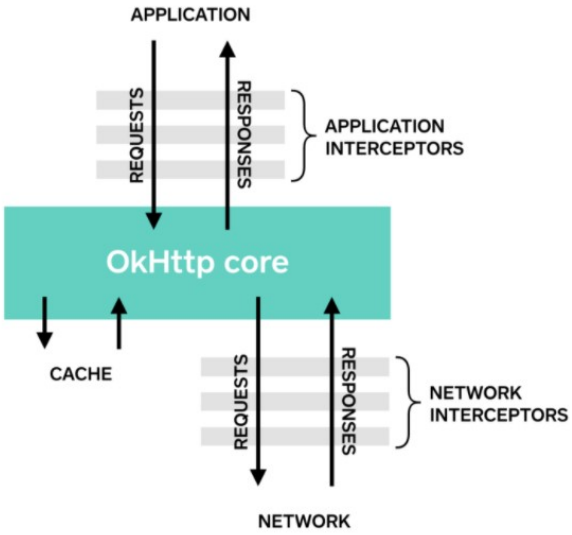


Figura F.1: Diagrama de funcionamiento OkHttp [2].

Estos interceptores nos permiten personalizar las solicitudes que realizamos. En este caso, vamos a interceptar la solicitud original que se realiza y agregarle una serie de encabezados que validarán la llamada con la clave API que nos proporcionó **Google Maps**. El código de este interceptor es el siguiente:

```
public class GoogleApiGeolocationInterceptor implements Interceptor
    {
    @Override
    public Response intercept(Interceptor.Chain chain) throws
    IOException {

        Request original = chain.request();
        HttpUrl originalHttpUrl = original.url();

        HttpUrl.Builder builder = originalHttpUrl.newBuilder();
        builder.addQueryParameter("key",
                "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
        HttpUrl url = builder.build();

        Request.Builder requestBuilder =
                original.newBuilder().url(url);


        Request request = requestBuilder.build();
        return chain.proceed(request);

    }
}
```

Además, debemos exponer una interfaz utilizando **Retrofit** que declare nuestras solicitudes y sus tipos. En nuestro caso, se trata de una operación **POST** accediendo al apartado de *geolocate*.

```
public interface GoogleGeolocationApi {
    @POST("geolocate")
    Call<ObjectGeolocationGoogle> getGeolocation(@Body
    ObjectBodyGeolocation body);
}
```

A continuación, para poder convertir la respuesta JSON que nos va a proporcionar **Retrofit** a un objeto **Java** deberemos utilizar la librería **Gson** y declarar con ella dos clases POJO.

```
public class ObjectGeolocationGoogle implements Serializable {
    @SerializedName("location")
```

```
    ObjectLocationValues location;

    @SerializedName("accuracy")
    int accuracy;

    public ObjectGeolocationGoogle() {
    }

    public ObjectGeolocationGoogle(ObjectLocationValues location,
    int accuracy) {
        this.location = location;
        this.accuracy = accuracy;
    }

    public ObjectLocationValues getLocation() {
        return location;
    }

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

    public int getAccuracy() {
        return accuracy;
    }

    public void setAccuracy(int accuracy) {
        this.accuracy = accuracy;
    }
}
```

```
public class ObjectLocationValues implements Serializable {
    @SerializedName("lat")
    float lat;

    @SerializedName("lng")
    float lng;

    public ObjectLocationValues() {
    }

    public ObjectLocationValues(float lat, float lng) {
        this.lat = lat;
        this.lng = lng;
    }

```

```
    public float getLat() {
        return lat;
    }

    public void setLat(float lat) {
        this.lat = lat;
    }

    public float getLng() {
        return lng;
    }

    public void setLng(float lng) {
        this.lng = lng;
    }
}
```

Por último, tenemos que declarar un servicio y realizar su implementación con una función que reciba las dos MAC, y que realice la petición HTTP a la dirección URL de la API de geolocalización de **Google Maps**.

```
public interface GeolocationDatasource {
        ObjectGeolocationGoogle searchLocation(String mac1, String
    mac2) throws JSONException;
}
```

```
public class GeolocationDatasourceImp implements
    GeolocationDatasource {

    @Override
    public ObjectGeolocationGoogle searchLocation(String mac1,
    String mac2)
        throws JSONException {

        OkHttpClient.Builder okHttpClient =
            new OkHttpClient.Builder();

        HttpLoggingInterceptor loggingInterceptor =
            new HttpLoggingInterceptor()
                .setLevel(BuildConfig.DEBUG ?
                HttpLoggingInterceptor.Level.BODY :
                HttpLoggingInterceptor.Level.NONE);

        List<Interceptor> interceptors =
            okHttpClient.interceptors();

```

```
19          interceptors.add(loggingInterceptor);
20
21          interceptors.add(new GoogleApiGeolocationInterceptor());
22
23          Retrofit retrofit =
24              new Retrofit.Builder()
25              .baseUrl("https://www.googleapis.com/geolocation/v1/")
26              .client(okHttpClient.build())
27              .addConverterFactory(GsonConverterFactory.create())
28              .build();
29
30          GoogleGeolocationApi api =
31              retrofit.create(GoogleGeolocationApi.class);
32
33          List<AccesPoint> accesPoints = new ArrayList<>();
34          accesPoints.add(new AccesPoint(mac1));
35          accesPoints.add(new AccesPoint(mac2));
36
37          Call<ObjectGeolocationGoogle> call =
38              api.getGeolocation(
39                      new ObjectBodyGeolocation(accesPoints));
40
41          try {
42
43              Response<ObjectGeolocationGoogle> response =
44                                              call.execute();
45
46              if (response.isSuccessful()) {
47
48                  return response.body();
49
50              } else {
51
52                  return new ObjectGeolocationGoogle();
53              }
54
55          } catch (IOException e) {
56              e.printStackTrace();
57          }
58
59          return new ObjectGeolocationGoogle();
60      }
61  }
```

Apéndice G

Acceso a Sqlite con Room

Room es una librería creada por **Android Google** que proporciona una capa de abstracción sobre **SQlite** que nos permite acceder a la base de datos de una manera sencilla y cómoda. **Room** presenta tres componentes principales (figura G.1):

- **Base de datos**: Contiene la base de datos en sí y sirve como punto de acceso principal para la conexión. Permite identificar que entidades vamos a utilizar y el Data Access Object (DAO) correspondiente. Se define como una clase abstracta que extiende *RoomDatabase*.
- **Entidad**: Vienen a simular las diferentes tablas dentro de una base de datos.
- DAO: Son los objetos que nos van a permitir acceder a los datos.

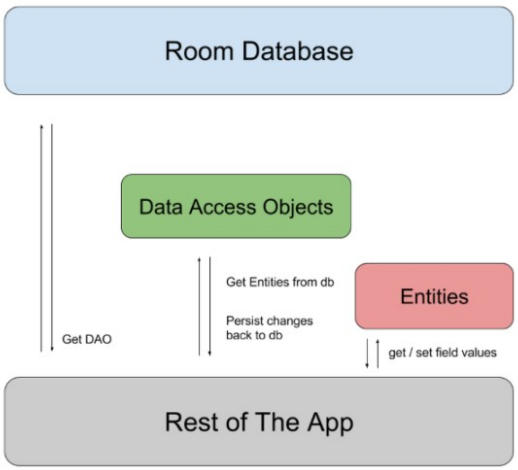


Figura G.1: Diagrama de arquitectura Room [3].

En nuestro caso concreto, para acceder a la base de datos interna en la que se almacenan las

diferentes instrucciones para realizar técnicas de primeros auxilios, se definen las siguientes clases y servicios.

Primero definimos la clase Data Access Object (DAO). Podemos definir diferentes interfaces, cada una de cuales representa una operación de acceso o modificación en la base de datos. En nuestro caso, solo definimos una operación de acceso, para devolver todas las entradas de la tabla *pasosRCP*.

```
@Dao
public interface RcpDao {
    @Query("SELECT * FROM pasosRcp")
    public List<RcpEntity> getSteps();
}
```

Lo siguiente que vamos a hacer es declarar la entidad que viene a representar la tabla de la que queremos obtener los datos, definiendo para ello los atributos que la forman.

```
@Entity(tableName = "pasosRcp")
public class RcpEntity {
    @PrimaryKey(autoGenerate = true)
    public int id;
    @NonNull
    @ColumnInfo(name = "name")
    public String name;
    @NonNull
    @ColumnInfo(name = "description")
    public String description;

    public RcpEntity(int id, String name, String description) {
        this.id = id;
        this.name = name;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

```

```
30      public void setName(String name) {
31          this.name = name;
32      }
33
34      public String getDescription() {
35          return description;
36      }
37
38      public void setDescription(String description) {
39          this.description = description;
40      }
41  }
```

Declaramos la base de datos a la que queremos acceder, en nuestro caso guardada internamente con el nombre de *rcp.db*. Incluimos también las entidades a las que vamos acceder y declaramos un método abstracto que no tenga argumentos y muestre la clase anotada con *@Dao*.

```
1  @androidx.room.Database(version = 1, entities = RcpEntity.class)
2  public abstract class RcpDatabase extends RoomDatabase {
3
4      public abstract RcpDao rcpDao();
5
6      private static RcpDatabase INSTANCE;
7      private static final int NUMBER_OF_THREADS = 4;
8      static final ExecutorService databaseWriteExecutor =
9              Executors.newFixedThreadPool(NUMBER_OF_THREADS);
10
11     public static RcpDatabase getInstance(Context context){
12         if(INSTANCE == null){
13             INSTANCE = getDatabase(context);
14         }
15         return INSTANCE;
16     }
17
18     static RcpDatabase getDatabase(final Context context) {
19         if (INSTANCE == null) {
20             synchronized (RcpDatabase.class) {
21                 if (INSTANCE == null) {
22                     INSTANCE =
23
       Room.databaseBuilder(context.getApplicationContext(),
24                             RcpDatabase.class, "rcp_database")
25                             .createFromAsset("rcp.db")
26                             .allowMainThreadQueries()
27                             .build();
```

```
                }
            }
        }
        return INSTANCE;
    }
}
```

Por último, solo tenemos que declarar una interfaz de servicio y una implementación para dicho servicio, para hacer uso de la arquitectura **Room**.

```
public interface RcpDatasource {
    List<ObjectModelRcp> searchStepsRcp();
}
```

```
public class RcpDatasourceImp implements RcpDatasource {

    List<RcpEntity> rcpEntity = new ArrayList<>();
    List<ObjectModelRcp> listaPasosRcp = new ArrayList<>();

    @Override
    public List<ObjectModelRcp> searchStepsRcp() {
        RcpDatabase db =
    RcpDatabase.getInstance(GlobalApplication.getAppContext());
        rcpEntity = db.rcpDao().getSteps();

        for (RcpEntity juego:rcpEntity){
            listaPasosRcp.add(new
    ObjectModelRcp(juego.getId(),juego.getName(),
    juego.getDescription()));
        }

        return listaPasosRcp;
    }
}
```

Lista de acrónimos

API Application Programming Interface. 8, 18, 24, 25, 28, 29, 37, 53–56, 58, 61, 67, 68, 70, 71, 78, 101, 102, 104

ATT Attribute Protocol. 11, 35, 39

AWG American Wire Gauge. 42

BLE Bluetooth Low Energy. vi, 10–12, 23, 28–30, 38, 41, 43, 50–53, 56, 59, 67, 68, 75, 78, 79, 86, 96, 98, 99

BRIF Brigada de Refuerzo en Incendios Forestales. vi, 76

CA Clean Architecture. 58

DAO Data Access Object. 107, 108

DMP Digital Motion Processor. 32

DRY Don't Repeat Yourself. 58

DSL Digital Subscriber Line. 8

ETSI European Telecommunications Standards Institute. 39

FSR Force Sensitive Resistor. v, vii, 14, 33, 34

GAP Generic Access Profile. 11

GATT Generic Attribute Profile. 11

GB GigaByte. 25

GHz Gigahertz. 25

GNSS Global Navigation Satellite System. 12

GPS Global Positioning Sytem. v, vii, 9, 10, 15, 27–29, 36, 43, 52–56, 64, 70, 74, 78

GSM Global System for Mobile communications. 15

HTML Hypertext Markup Language. 89

HTTP Hypertext Transfer Protocol. 18, 65, 101, 104

I2C Inter-Integrated Circuit. 13, 32

IDE Integrated Development Environment. 17

IOS iPhone Operating System. 16

IoT Internet of Things. 7, 10, 12, 38, 41

ISM Industrial, Scientific and Medical band. 10

JS JavaScript. 89

JSON JavaScript Object Notation. 17, 18, 61, 101, 102

Kb Kilobyte. 38

LiPo Lithium Polymer Battery. 38

LNA low-noise amplifier. 8

LoRa Long Range. 7, 8

LPWAN Lower Power Wide Area Network. 7, 8

M2M Machine to machine. 7

MAC Media Access Control. 28, 29, 37, 52, 54, 55, 64, 70, 71, 74, 78, 101, 104

MEMS micro-electromechanical system. 13, 32

NMEA National Marine Electronics Association. 10, 36

NTC Negative Temperature Coefficient. 14, 31

PAC Porting Authorisation Code. 36

PCB Printed Circuit Board. 13, 15, 42, 78, 80

POJO Plain Old Java Object. 58, 102

PTC Positive Temperature Coefficient. 14

PWM pulse-width modulation. 13, 38

QR Quick Response code. 18, 47, 50, 51, 67, 71, 86, 96

RAM Random Access Memory. 38

RCP Reanimación cardiopulmonar. 80

RF Radio Frequency. 10

RTD Resistance Temperature Detector. 13

SCL Serial Clock. 32

SDA Serial Data. 32

SDK Software Development Kit. 30, 63, 67, 68

SIM Subscriber Identity Module. 15

SPDY Speedy. 18

SPI Serial Peripheral Interface. 13

UART Universal asynchronous receiver-transmitter. v, 10, 36, 43

UNB Ultra-Narrow Band. 8

URL Uniform Resource Locator. 39, 104

USB Universal Serial Bus. 38

UTC Coordinated Universal Time. 10

UUID Universally unique identifier. 11

Bibliografía

[1] "Diagrama de funcionamiento del módulo GPS," accedido Mayo 2020. [En línea]. Disponible en: https://www.techplayon.com/gps-gps-module-used-base-station-applications-choose-gps-module/

[2] "Diagrama funcionamiento OkHttp interceptors," accedido Mayo 2020. [En línea]. Disponible en: https://square.github.io/okhttp/interceptors/

[3] "Diagrama arquitectura Room," accedido Mayo 2020. [En línea]. Disponible en: https://developer.android.com/training/data-storage/room

[4] "Maxpreven," accedido Mayo 2020. [En línea]. Disponible en: https://www.maxpreven.com/realidad-virtual/

[5] "Lifetec," accedido Mayo 2020. [En línea]. Disponible en: https://www.lifetec.com.au/

[6] "Maniobra RCP," accedido Mayo 2020. [En línea]. Disponible en: https://www.mayoclinic.org/es-es/first-aid/first-aid-cpr/basics/art-20056600

[7] "Lora Alliance," accedido Mayo 2020. [En línea]. Disponible en: https://lora-alliance.org/about-lora-alliance/

[8] "Tecnología Sigfox," accedido Mayo 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Sigfox

[9] "Sigfox," accedido Mayo 2020. [En línea]. Disponible en: https://build.sigfox.com/sigfox

[10] "Ultra-Narrow Band," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Ultra_Narrowband

[11] "Low-Noise Amplifier," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Low-noise_amplifier

[12] "Backhaul," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Backhaul_(telecommunications)

[13] "Digital Subscriber Line," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Digital_subscriber_line

[14] "Cobertura nacional Sigfox," accedido Mayo 2020. [En línea]. Disponible en: https://www.sigfox.com/en/coverage

[15] "Global Positioning System," accedido Mayo 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/GPS

[16] "Partes del mensaje GPS," accedido Mayo 2020. [En línea]. Disponible en: https://www.puntoflotante.net/TUTORIAL-SISTEMA-DE-POSICIONAMIENTO-GLOBAL-GPS.htm

[17] "Radio Frequency Connector," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/RF_connector

[18] "Universal Asynchronous Receiver-Transmitter," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

[19] "National MarineElectronics Association," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/NMEA_0183

[20] "Coordinated Universal Time," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Coordinated_Universal_Time

[21] "Bluetooth Low Energy," accedido Mayo 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Bluetooth_de_baja_energ%C3%ADa

[22] "Zigbee," accedido Mayo 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Zigbee

[23] "Z-wave," accedido Mayo 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Z-Wave

[24] "Protocolo ATT," accedido Mayo 2020. [En línea]. Disponible en: https://epxx.co/artigos/bluetooth_gatt.html

[25] "Arduino IDE," accedido Mayo 2020. [En línea]. Disponible en: https://www.arduino.cc/en/software

[26] "Global Navigation Satellite System," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Satellite_navigation

[27] "Sensor Tilt," accedido Mayo 2020. [En línea]. Disponible en: https://learn.adafruit.com/tilt-sensor

[28] "Mems," accedido Mayo 2020. [En línea]. Disponible en: https://gl.wikipedia.org/wiki/MEMS

[29] "Serial Peripheral Interface," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

[30] "Inter-Integrated Circuit," accedido Mayo 2020. [En línea]. Disponible en: https://learn.sparkfun.com/tutorials/i2c/all

[31] "U-blox," accedido Mayo 2020. [En línea]. Disponible en: https://www.u-blox.com/en/iot-location-service

[32] "Groupe Spécial Mobile," accedido Mayo 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Sistema_global_para_las_comunicaciones_m%C3%B3viles

[33] "Java," accedido Mayo 2020. [En línea]. Disponible en: https://docs.oracle.com/en/java/

[34] "Kotlin," accedido Mayo 2020. [En línea]. Disponible en: https://kotlinlang.org/docs/home.html

[35] "Swift," accedido Mayo 2020. [En línea]. Disponible en: https://swift.org/documentation/

[36] "Objective-C," accedido Mayo 2020. [En línea]. Disponible en: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[37] "Javascript," accedido Mayo 2020. [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[38] "Flutter," accedido Mayo 2020. [En línea]. Disponible en: https://flutter.dev/docs

[39] "Parse," accedido Mayo 2020. [En línea]. Disponible en: https://parseplatform.org/

[40] "Back4app," accedido Mayo 2020. [En línea]. Disponible en: https://www.back4app.com/

[41] "Firebase," accedido Mayo 2020. [En línea]. Disponible en: https://firebase.google.com/

[42] "Android Studio," accedido Mayo 2020. [En línea]. Disponible en: https://developer.android.com/studio

[43] "Gradle," accedido Mayo 2020. [En línea]. Disponible en: https://docs.gradle.org/current/userguide/userguide.html

[44] "Adobe XD," accedido Mayo 2020. [En línea]. Disponible en: https://www.adobe.com/products/xd.html

[45] "Librería Firebase para Android," accedido Mayo 2020. [En línea]. Disponible en: https://firebase.google.com/docs/libraries?hl=es

[46] "Librería Camera X," accedido Mayo 2020. [En línea]. Disponible en: https://developer.android.com/training/camerax

[47] "Librería ML Kit," accedido Mayo 2020. [En línea]. Disponible en: https://developers.google.com/ml-kit

[48] "Librería Retrofit," accedido Mayo 2020. [En línea]. Disponible en: https://square.github.io/retrofit/

[49] "Protocolo OkHttp," accedido Mayo 2020. [En línea]. Disponible en: https://square.github.io/okhttp/

[50] "Protocolo SPDY," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/SPDY

[51] "Librería Gson," accedido Mayo 2020. [En línea]. Disponible en: https://github.com/google/gson

[52] "Librería Room," accedido Mayo 2020. [En línea]. Disponible en: https://developer.android.com/jetpack/androidx/releases/room

[53] "Librería Picasso," accedido Mayo 2020. [En línea]. Disponible en: https://square.github.io/picasso/

[54] "Librería GraphView," accedido Mayo 2020. [En línea]. Disponible en: https://github.com/jjoe64/GraphView

[55] "Software de control de versiones GIT," accedido Mayo 2020. [En línea]. Disponible en: https://git-scm.com/

[56] "Git-Flow," accedido Mayo 2020. [En línea]. Disponible en: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[57] "Convenio analista-programador," accedido Mayo 2020. [En línea]. Disponible en: https://cgtsoprasteria.org/blog/rangos-salariales-anuales-tic/

[58] "Sigfox Atlas," accedido Mayo 2020. [En línea]. Disponible en: https://build.sigfox.com/geolocation-sigfox-atlas

[59] "Librería DHT.h," accedido Mayo 2020. [En línea]. Disponible en: https://www.arduino.cc/reference/en/libraries/dht-sensor-library/

[60] "Librería Wire.h," accedido Mayo 2020. [En línea]. Disponible en: https://www.arduino.cc/en/reference/wire

[61] "Cqrobot," accedido Mayo 2020. [En línea]. Disponible en: https://www.cqrobot.com/

[62] "Librería Tiny.gps," accedido Mayo 2020. [En línea]. Disponible en: https://www.arduino.cc/reference/en/libraries/tinygps/

[63] "U-Blox," accedido Mayo 2020. [En línea]. Disponible en: https://www.u-blox.com/en/we-build-last

[64] "Regulación ETSI," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/ETSI

[65] "Arduino BLE," accedido Mayo 2020. [En línea]. Disponible en: https://www.arduino.cc/en/Reference/ArduinoBLE

[66] "Maniquí RCP-AED," accedido Mayo 2020. [En línea]. Disponible en: https://iberomed.es/emergencias/formacion-y-simuladores/maniqui-rcp-aed-adulto-prestan.html?gclid=CjwKCAjw2ZaGBhBoEiwA8pfP_u1OzpVfZwZ08Ckb0EyLwToraItHnAFpRHeDWq95Cx6cVN32MNKxmRoCsqYQAvD_BwE

[67] "Clean Architecture," accedido Mayo 2020. [En línea]. Disponible en: https://medium.com/android-dev-hacks/detailed-guide-on-android-clean-architecture-9eab262a9011

[68] "Plain Old Java Object," accedido Mayo 2020. [En línea]. Disponible en: https://www.ramoncarrasco.es/es/content/es/kb/119/que-es-pojo

[69] "Clean Code," accedido Mayo 2020. [En línea]. Disponible en: https://samuelcasanova.com/2016/09/resumen-clean-code/

[70] "Patrón DRY," accedido Mayo 2020. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

[71] "Api Bluetooth de Google," accedido Mayo 2020. [En línea]. Disponible en: https://developer.android.com/guide/topics/connectivity/bluetooth