

***Facultad de Ciencias***

**Prototipo de sistema de teleasistencia para personas dependientes**

(Prototype of a telecare system for dependent people)

Trabajo de Fin de Grado para acceder al

**GRADO EN INGENIERÍA INFORMÁTICA**

Autor: Roberto González Jiménez Codirector: Mario Aldea Rivas Codirector: Héctor Pérez Tijero

Julio - 2021

Índice de Contenidos

[Agradecimientos 6](#_Toc75915717)

[Resumen 7](#_Toc75915718)

[Abstract 8](#_Toc75915719)

[1. Introducción 9](#_Toc75915720)

[1.1 Motivación 9](#_Toc75915721)

[1.2 Objetivos 9](#_Toc75915722)

[1.3 Metodología de trabajo 9](#_Toc75915723)

[2. Tecnologías y herramientas utilizadas 11](#_Toc75915724)

[2.1 Tecnologías 11](#_Toc75915725)

[2.2 Herramientas 12](#_Toc75915726)

[3. Análisis de requisitos 14](#_Toc75915727)

[3.1 Requisitos funcionales 14](#_Toc75915728)

[3.2 Requisitos no funcionales 15](#_Toc75915729)

[4. Diseño de la aplicación 16](#_Toc75915730)

[4.1 Clases utilizadas 16](#_Toc75915731)

[4.1.1 RemoteSoft 16](#_Toc75915732)

[4.1.2 RemoteSoft Receives 19](#_Toc75915733)

[4.2 Diseño detallado 22](#_Toc75915734)

[5. Implementación 24](#_Toc75915735)

[5.1 Handlers 24](#_Toc75915736)

[5.2 Detección de caídas 25](#_Toc75915737)

[5.3 Pulsera Xiaomi 27](#_Toc75915738)

[5.4 Ubicación 28](#_Toc75915739)

[5.5 ThingSpeak 29](#_Toc75915740)

[5.6 Alarmas 33](#_Toc75915741)

[6. Pruebas 35](#_Toc75915742)

[6.1 Mecanismos de notificación de Android 35](#_Toc75915743)

[6.2 Pruebas funcionales 36](#_Toc75915744)

[7. Conclusión y trabajos futuros 37](#_Toc75915745)

[7.1 Conclusión 37](#_Toc75915746)

[7.2 Trabajos futuros 37](#_Toc75915747)

[8. Bibliografía 39](#_Toc75915748)

Índice de Figuras

[Figura 1.1 Diagrama de Gantt, lista de tareas 10](#_Toc75903977)

[Figura 1.2 Diagrama de Gantt, cronograma 10](#_Toc75903978)

[Figura 4.1 Diseño de las aplicaciones 16](#_Toc75903979)

[Figura 4.2 Clases utilizadas en la app RemoteSoft 17](file:///C:\Users\GORDO\Documents\MEGA\MEGAsync%20Uploads\UNI\TFG\Docs\Informe\Gonzalez%20Jiménez%20Roberto.docx#_Toc75903980)

[Figura 4.3 Vista principal de la app RemoteSoft 17](#_Toc75903981)

[Figura 4.4 Vista Acerca de, de la app RemoteSoft 18](#_Toc75903982)

[Figura 4.5 Vista de visualización de los datos, de la app RemoteSoft 18](#_Toc75903983)

[Figura 4.6 Vista de vinculación de pulsera, de la app RemoteSoft 19](#_Toc75903984)

[Figura 4.7 Clases utilizadas en la app RemoteSoft Receives 20](file:///C:\Users\GORDO\Documents\MEGA\MEGAsync%20Uploads\UNI\TFG\Docs\Informe\Gonzalez%20Jiménez%20Roberto.docx#_Toc75903985)

[Figura 4.8 Vista principal de la aplicación RemoteSoft Receives, con y sin el switch principal activado 20](#_Toc75903986)

[Figura 4.9 Vista Acerca de, de la app RemoteSoft Receives 21](#_Toc75903987)

[Figura 4.10 Vista de visualización de los datos recogidos por el acelerómetro, de RemoteSoft Receives 21](#_Toc75903988)

[Figura 4.11 Vista de visualizacion de las pulsaciones, de RemoteSoft Receives 22](#_Toc75903989)

[Figura 4.12 Vista de la ubicación, de RemoteSoft Receives 22](#_Toc75903990)

[Figura 4.13 Vista detallada de las aplicaciones 23](#_Toc75903991)

[Figura 5.1 Calculo de los valores de los 3 ejes del acelerómetro 25](#_Toc75903992)

[Figura 5.2 Funcionamiento de la Linked List 26](#_Toc75903993)

[Figura 5.3 Pulsera Xiaomi Mi Band 1s 27](#_Toc75903994)

[Figura 5.4 Permiso de ubicación en Android 28](#_Toc75903995)

[Figura 5.5 Funcionamiento del protocolo MQTT 29](#_Toc75903996)

[Figura 5.6 Visualización de un canal de ThingSpeak 30](#_Toc75903997)

[Figura 5.7 Canales junto con los campos utilizados 32](#_Toc75903998)

[Figura 5.8 Los dos canales necesarios en ThingSpeak 32](#_Toc75903999)

[Figura 5.9 Formulario de configuración del uso de canales de ThingSpeak 33](#_Toc75904000)

[Figura 5.10 Notificaciones de las alarmas implementadas 34](#_Toc75904001)

Índice de Tablas

[Tabla 1 Requisitos funcionales de la aplicación de captura y envio de datos RemoteSoft 14](#_Toc75883967)

[Tabla 2 Requisitos funcionales de la aplicación de monitorización de datos RemoteSoft Receives: 14](#_Toc75883968)

[Tabla 3 Requisitos no funcionales de las aplicaciones 15](#_Toc75883969)

# Agradecimientos

Gracias a todas las personas que me han ayudado a lo largo de la carrera y a las personas que me han ayudado, de manera directa o indirecta en la realización de este trabajo.

# Resumen

Hay muchas personas que no pueden cuidarse por sí mismas y necesitan ser supervisados por otros para asegurar su salud. Con el objeto de facilitar la labor de teleasistencia, en este proyecto hemos desarrollado una aplicación que permite ayudar a las personas supervisoras a que las personas dependientes que cuidan estén seguras hasta estando separados de ellos.

El principal objetivo es monitorizar el estado de las personas dependientes a través de diversos sensores que se encuentran en el teléfono móvil o también con la ayuda de otros dispositivos como la pulsera de actividad de Xiaomi. Los sensores que vamos a utilizar en este trabajo son el acelerómetro y la ubicación captada por el teléfono móvil y con la pulsera de Xiaomi utilizaremos su pulsómetro para medir las pulsaciones de la persona dependiente. Las lecturas de los sensores estarán disponibles tanto para la persona como para la persona que lo supervisa. Además, la aplicación permite notificar situaciones de alarma a la persona supervisora. Dichas alarmas se producen cuando alguno de los parámetros monitorizados excede los límites preestablecidos. En el estado alcanzado por el prototipo desarrollado en este trabajo, las alarmas son debidas a un número de latidos por minuto excesivo o por la detección de la caída del móvil de la persona dependiente.

Se han desarrollado dos aplicaciones Android para cumplir este propósito, la primera app se llama RemoteSoft y es la que tendrá instalada la persona dependiente en su teléfono, esta aplicación captará con los sensores todos los datos necesarios y los podrá subir a la nube para que su supervisor pueda ver sus valores, la otra aplicación Android se llama RemoteSoft Receives y es la que tendrá instalada la persona supervisora, podrá elegir a la persona que supervisar y recolectará los datos que están en la nube para ser mostrados en esta app, además notificará las alarmas que he hablado anteriormente cuando haya algún valor fuera de lugar.

La comunicación entre las aplicaciones en los teléfonos de las personas dependiente y supervisora se realiza utilizando el protocolo MQTT mediante la infraestructura proporcionada por el sitio web ThingSpeak.

**Palabras** **clave**: Persona dependiente, teléfono móvil, Android, MQTT, sensores, teleasistencia

# Abstract

There are many people who cannot take care of themselves and need to be supervised by others to ensure their health. In order to facilitate telecare work, in this project we have developed an application that allows supervisors to be helped so that the dependent people they take care of are safe even when they are separated from them.

The main objective is to monitor the status of dependent people through various sensors found on the mobile phone or also with the help of other devices such as the Xiaomi activity bracelet. The sensors that we are going to use in this work are the accelerometer and the location captured by the mobile phone and with the Xiaomi bracelet we will use its heart rate monitor to measure the heart rate of the dependent person. Sensor readings will be available to both the person and the person supervising you. In addition, the application allows notifying alarm situations to the supervisor. These alarms occur when any of the monitored parameters exceed the pre-established limits. In the state reached by the prototype developed in this work, the alarms are due to an excessive number of beats per minute or by the detection of the dependent person's mobile falling.

Two Android applications have been developed to fulfill this purpose, the first app is called RemoteSoft and it is the one that the dependent person will have installed on their phone, this application will capture all the necessary data with the sensors and can upload them to the cloud so that their supervisor can see their values, the other Android application is called RemoteSoft Receives and it is the one that the supervisor will have installed, you will be able to choose the person to supervise and will collect the data that is in the cloud to be displayed in this app, it will also notify the alarms that I have discussed above when there is some misplaced value.

Communication between the applications on the phones of the clerk and supervisor is done using the MQTT protocol through the infrastructure provided by the ThingSpeak website.

**Keywords**: Dependent person, mobile phone, Android, MQTT, sensors, telecare

# Introducción

Como introducción, en este primer capítulo describiremos brevemente cuál ha sido la motivación para realizar este trabajo, los objetivos del proyecto y la metodología y el plan de trabajo utilizado para su desarrollo.

## Motivación

Para contextualizar este trabajo hay que hablar sobre un problema que afecta a muchas familias, respecto a años anteriores la sanidad ha sufrido una gran mejoría, antiguamente no se tenía tanto conocimiento y la tecnología utilizada no era tan avanzada, esto ha causado un aumento en la esperanza de vida de la población, como consecuencia de esto hay muchas personas con edad avanzada, algunas tan mayores que no se pueden cuidar por sí mismas y otras personas que debido a enfermedades o problemas que hayan podido ocurrir tampoco, es decir, son personas dependientes.

Desgraciadamente estas personas dependientes necesitan apoyo y siempre hay que estar pendientes de cuál es su estado por si algo puede pasar, lo cual fuerza a otras personas a cuidar de ellas y mantener una constante supervisión de estos. Por otra parte, en la actualidad la tecnología cobra una gran importancia y prácticamente todo el mundo posee de un teléfono móvil con conexión a internet.

Para desarrollar nuestra aplicación hemos aprovechado el gran uso e importancia de los teléfonos móviles en la actualidad para resolver la problemática de la constante necesidad de supervisar a las personas dependientes, la aplicación está desarrollada en Android, ya que es el sistema operativo de móvil más usado mundialmente.

## Objetivos

Como se ha mencionado en el apartado anterior, el principal objetivo es desarrollar una aplicación que permita monitorizar de forma remota el estado de las personas dependientes a través de diversos sensores que se encuentran en el teléfono móvil o también con la ayuda de otros dispositivos como por ejemplo las pulseras de actividad, que disponen de diferentes funciones para analizar a la persona como es el caso del pulsómetro.

Como objetivo secundario y no menos importante, el aprender gracias a la realización del trabajo, en cuanto a el desarrollo de aplicaciones Android y las tecnologías y herramientas utilizadas en este trabajo, como resultado de mejorar para proyectos futuros.

## Metodología de trabajo

En cuanto a la metodología de trabajo se ha organizado en tandas de una semana, en la que al principio de la semana se establecían los objetivos a desarrollar en dicha semana, cada semana se realizaba una reunión con los tutores con el fin de compartir el trabajo realizado en cada semana, comentar las posibles mejoras y encontrar los posibles errores y los ajustes que hay que retocar.

En cuanto a las reuniones semanales se han realizado mediante un servidor de Discord, en el que usábamos para comunicarnos por voz al igual que mediante mensajes de texto, en los canales de texto también lo usábamos para recordatorios y anotar detalles, teniendo un canal de recursos donde se encontraban links de páginas interesantes e información referente al trabajo y las tecnologías utilizadas.

Para la organización del proyecto se ha empleado un repositorio en GitHub, en la que también podían acceder los tutores, en este repositorio se encuentra organizado en apartados para encontrar fácilmente donde está la información, conteniendo los proyectos Android de las 2 apps, los archivos APKs de las aplicaciones con el fin de probar si funciona en el teléfono móvil tanto en el propio como en el teléfono de los tutores y todos los documentos, imágenes y recursos que se han ido utilizando en la realización del proyecto.

**El repositorio GitHub del trabajo se encuentra en el siguiente enlace:**

<https://github.com/robgzj/TFG_Teleasistencia_2021>

Entre estos documentos se encuentra un semanario, donde se describe lo que se ha trabajado en cada semana junto con los problemas encontrados, posibles soluciones y anotaciones de importancia para poder avanzar y arreglar en posteriores semanas.

A continuación, se muestra el estado del diagrama de Gantt del proyecto, mostrando las tareas realizadas cada semana a lo largo del proyecto.

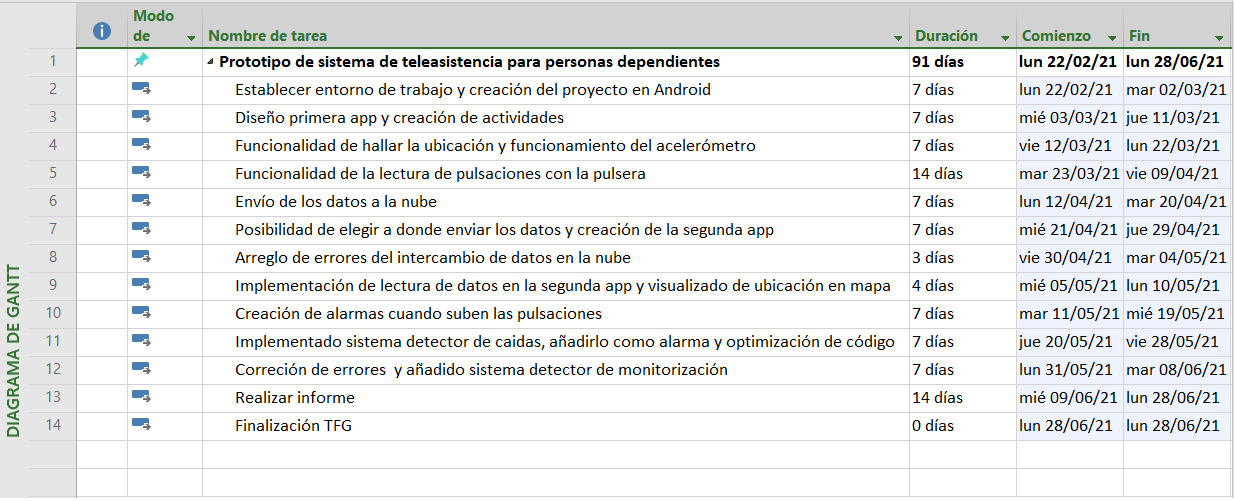


Figura 1.1 Diagrama de Gantt, lista de tareas

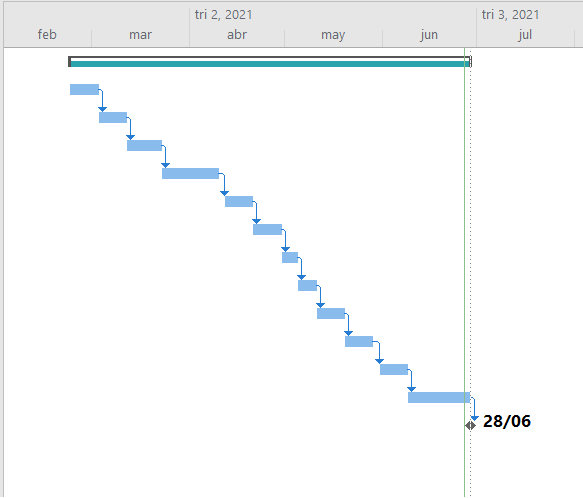


Figura 1.2 Diagrama de Gantt, cronograma

# Tecnologías y herramientas utilizadas

En este apartado describiremos brevemente las tecnologías y herramientas utilizadas en este trabajo, con una simple explicación de que consiste cada una.

## Tecnologías

**Java**

Java[1] es un lenguaje de programación orientado a objetos, independiente de la plataforma hardware donde se desarrolla, y que utiliza una sintaxis similar a la de C++ pero reducida. Fue desarrollado originalmente por James Gosling, de Sun Microsystems, pero actualmente pertenece a Oracle, ya que fue adquirida por esta en 2010. Es un lenguaje muy fácil de aprender y de los lenguajes más populares en uso, particularmente para aplicaciones de cliente-servidor de web.

**Android**

Android[2] es un sistema operativo móvil basado en núcleo Linux y otros softwares de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil como smartphones, tablets, relojes inteligentes, televisores y hasta para algunos automóviles. Inicialmente fue creado por Android Inc., que más tarde fue adquirido por Google, es el sistema operativo móvil más utilizado en el mundo.

**Git**

Git[3] es un software que realiza la función del control de versiones de código de forma distribuida, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Git permite llevar un registro de todos los cambios hechos en cada uno de los ficheros del proyecto y acceder a cualquiera de las versiones si se han guardado los cambios, permite a los desarrolladores disponer de un repositorio local en el cual pueden ir llevando un control de sus cambios sin necesidad de sincronizarlos con el repositorio remoto.



**MQTT**

MQTT[4] son las siglas de Message Queuing Telemetry Transport y es un protocolo de red ligero de publicación-suscripción que transporta mensajes entre dispositivos, en la sección de implementación se explicará en profundidad.

**Eclipse Paho**

Eclipse Paho[5] es una implementación MQTT, disponible para los lenguajes Java, C#, Go, C, Python y JavaScript, es la librería Eclipse que hemos utilizado para poder establecer la comunicación e intercambio de mensajes con ThingSpeak en nuestras aplicaciones Android.

**MiBand SDK**

MiBand SDK consiste en el conjunto de librerías usadas en dar soporte al control de la información obtenida de la pulsera de actividad Xiaomi, en este caso hemos usado un SDK no oficial, desarrollado por un programador llamado **pangliang**[6], con el fin de tener un acceso total a la pulsera en nuestras apps.

## Herramientas

**Android Studio**

Android Studio[7] es el entorno de desarrollo integrado oficial para la plataforma Android. Pertenece a Google y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. , Kotlin es el lenguaje preferido de Google para el desarrollo de aplicaciones Android, aunque admite otros lenguajes de programación, como Java y C++.



**Sourcetree**

Sourcetree[8] es un entorno gráfico para trabajar con repositorios Git sin necesidad de utilizar la consola de comandos. Esta herramienta fue desarrollada por Atlassian y está disponible para Windows y MacOS. Agiliza el proceso de trabajo con Git y es muy útil, ya que te permite visualizar el contenido de los cambios que realizas.

**MQTT.fx**

MQTT.fx[9] es uno de los clientes más populares de MQTT, hecho en Java y basado en Eclipse Paho, utilizado para realizar las primeras pruebas y comprobación del funcionamiento de la comunicación con ThingSpeak.

**ThingSpeak**

ThingSpeak[10] es una plataforma de Internet que entre otras cosas permite recoger y almacenar datos de sensores en la nube utilizando el protocolo MQTT. ThingSpeak es parte de Mathworks que es la empresa de MATLAB y ofrece aplicaciones que permiten analizar y visualizar tus datos en MATLAB y actuar sobre los datos. En la sección de implementación se explicará en profundidad.

**Microsoft Project**

Microsoft Project[11] es un software de administración de proyectos comercializado por Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo. Esta herramienta ha ayudado a la organización semanal del trabajo.

# Análisis de requisitos

En esta sección se recoge el proceso de captura de requisitos, así como el detalle de los requisitos tanto funcionales como los no funcionales.

## Requisitos funcionales

Los requisitos funcionales definen la funcionalidad que debe de cumplir el software, estos requisitos son definidos una tabla para cada aplicación.

|  |  |
| --- | --- |
| **Identificador** | **Descripción** |
| RF-001 | La aplicación calculará la ubicación actual de la persona dependiente, para ser enviada posteriormente. |
| RF-002 | La aplicación medirá con el acelerómetro las aceleraciones en las coordenadas X, Y y Z de la persona dependiente, para ser enviadas posteriormente. |
| RF-003 | La aplicación contará con un algoritmo para detectar si la persona dependiente ha sufrido alguna caída. |
| RF-004 | La aplicación permitirá la vinculación a determinados dispositivos Bluetooth para acceder a más sensores, pudiendo ver los dispositivos y cambiar de un dispositivo a otro. |
| RF-005 | La aplicación medirá las pulsaciones de la persona dependiente, para ser enviadas posteriormente. |
| RF-006 | La aplicación permitirá visualizar los datos de todos los sensores que se han utilizado. |
| RF-007 | El usuario podrá elegir si quiere enviar los datos o no mediante un switch. |
| RF-008 | La aplicación permitirá configurar donde para guardar los datos de los sensores de las personas dependientes. |
| RF-009 | Si no hay nadie monitorizando los datos de los sensores, se guardarán los datos que tengan indicios de peligro hasta que monitorice alguien, para no pasar por alto ninguna alarma y evitar riesgos a la persona dependiente. |

Tabla 1 Requisitos funcionales de la aplicación de captura y envío de datos RemoteSoft

|  |  |
| --- | --- |
| **Identificador** | **Descripción** |
| RF-010 | La aplicación mostrará cada uno de los datos de los sensores captados a la persona dependiente. |
| RF-011 | El usuario podrá elegir si quiere monitorizar los datos o no mediante un switch. |
| RF-012 | Se podrá vigilar a más de una persona dependiente, solamente cambiando el canal de ThingSpeak que pertenezca a la persona que se desea monitorizar. |
| RF-013 | Se mostrará una alarma en la aplicación si las pulsaciones de la persona dependiente son iguales o superiores a 100 latidos por minuto (LPM). |
| RF-014 | Se mostrará una alarma en la aplicación si se ha detectado que la persona dependiente ha sufrido una caída. |
| RF-015 | La aplicación debe informar a la aplicación de envío de datos que hay alguien monitorizando, para que sepa que la alarma ha sido notificada. |

Tabla 2 Requisitos funcionales de la aplicación de monitorización de datos RemoteSoft Receives

## Requisitos no funcionales

Los requisitos no funcionales definen las características generales y restricciones de las aplicaciones, no tienen que ver directamente con el uso que van a hacer los usuarios, sino que son características que debe cumplir el sistema en conjunto para permitir a los usuarios aprovecharse de las funcionalidades que se les ofrece, como ejemplo de estas características pueden ser el rendimiento, disponibilidad y seguridad entre otros. Estos requisitos son los mismos en las dos aplicaciones y están definidos en la siguiente tabla.

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Tipo** | **Descripción** |
| RNF-001 | Disponibilidad | Se necesitará conexión a internet, la ubicación y el bluetooth con la pulsera vinculada para obtener todos los datos. |
| RNF-002 | Compatibilidad | Las apps funcionarán sobre Android, siendo la API mínima la 30. |
| RNF-003 | Seguridad | Se necesitarán identificadores y claves específicas para acceder o modificar los datos almacenados. |
| RNF-004 | Rendimiento | Los datos de los sensores se enviarán al servidor donde se encuentran los datos cada 20 segundos. |
| RNF-005 | Interfaz | El idioma que se utilizará para las dos aplicaciones será el español. |
| RNF-006 | Usabilidad | Uso del protocolo MQTT para adquirir experiencia en su uso. |

Tabla 3 Requisitos no funcionales de las aplicaciones

# Diseño de la aplicación

Con los requisitos definidos en el apartado anterior se debe diseñar una aplicación que cumpla todos estos, como se ha mencionado previamente se han necesitado 2 aplicaciones Android.

1. **RemoteSoft**: Esta aplicación es la que tendrá instalada la persona dependiente en su teléfono móvil, esta app se encargará de recoger la información de los sensores y de la pulsera de actividad para enviárselo a la persona supervisora.
2. **RemoteSoft Receives**: Esta aplicación es la que tendrá instalada la persona supervisora en su teléfono móvil, esta app recibirá la información de los sensores captados por la app RemoteSoft y la mostrará al usuario con el fin de que la persona supervisora pueda monitorizar a la persona dependiente y estar informado de su estado actual.

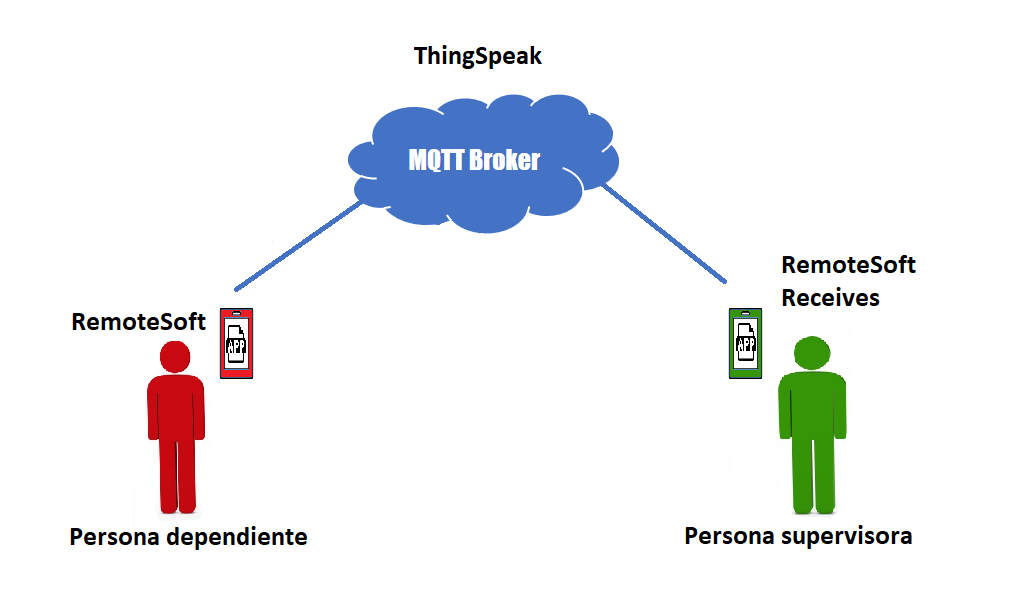


Figura 4.1 Diseño de las aplicaciones

Como puede verse en la Figura 4.1, la comunicación entre la persona dependiente y la supervisora se realiza mediante ThingSpeak, que es un bróker MQTT ya que utiliza este protocolo para el transporte de mensajes entre los dispositivos.

En cuanto a la estructura de las aplicaciones, las dos disponen de una vista principal de inicio, desde esta vista se puede acceder a todas las demás vistas. Se utilizan Activities para cada vista, que contienen el código Java para hacer funcionar la aplicación, utilizan muchos elementos Android para el funcionamiento.

De los elementos más significativos y vitales para la aplicación RemoteSoft son los *listeners*, estos elementos se encargan de controlar los eventos, esperan a que se produzcan y realiza una serie de acciones. Para cada sensor existe un *listener* que va actualizando los valores de los datos que va recolectando.

## Clases utilizadas

### RemoteSoft

Como podemos ver en la imagen hay un total de 6 clases, las 4 primeras clases están en un paquete llamado Activities, en este se encuentran las 4 vistas que tendrá nuestra aplicación, vamos a comentar lo que contiene cada clase.

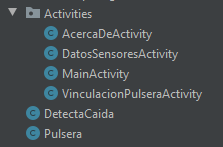


Figura 4.2 Clases utilizadas en la app RemoteSoft

* La clase **MainActivity** es la clase que se mostrará al iniciar la aplicación y contiene 3 botones para acceder a las otras vistas, también otro botón que sirve para configurar los canales de ThingSpeak que se van a utilizar para el envío/recepción de datos en la aplicación. Y finalmente tendremos un switch que nos permitirá encender/apagar la app y realizar el envío de los datos de los sensores.

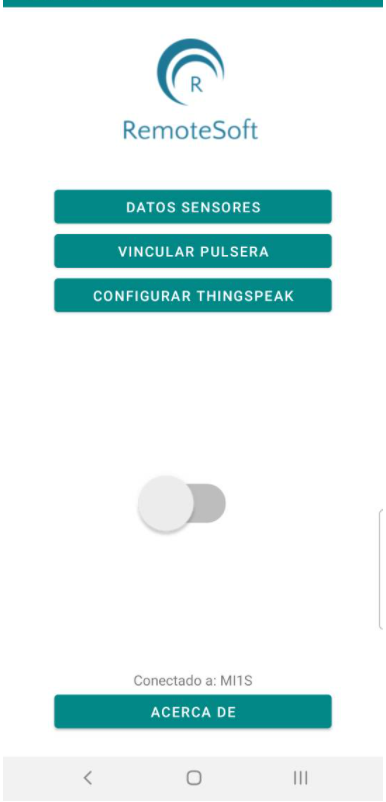


Figura 4.3 Vista principal de la app RemoteSoft

* La clase **AcercaDeActivity** es simplemente una clase informativa, que muestra información básica sobre la app, el autor de quien lo ha desarrollado y requisitos para que funcione la aplicación en su totalidad.

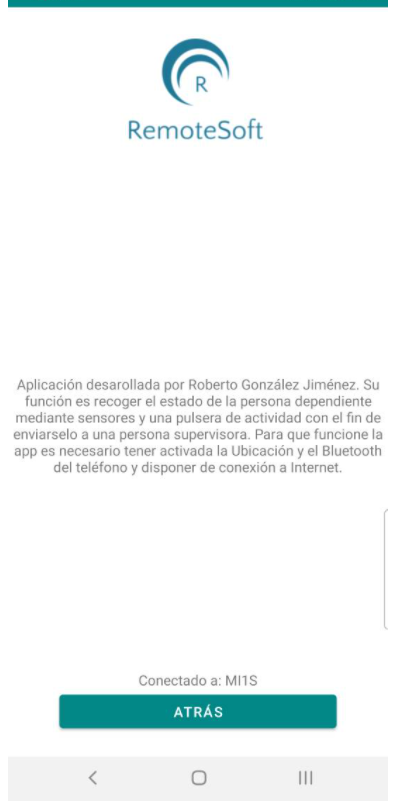


Figura 4.4 Vista Acerca de, de la app RemoteSoft

* La clase **DatosSensoresActivity** esta clase muestra todos los datos que se recolectan de los sensores, permite a la persona dependiente o usuario de la aplicación que pueda observar los propios datos que se están capturando en cada momento.

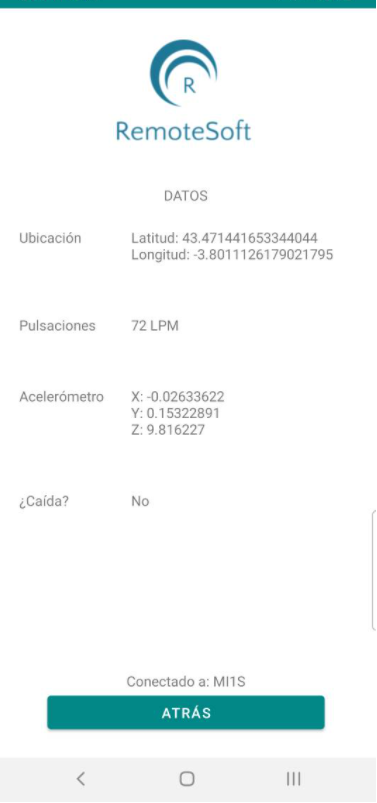


Figura 4.5 Vista de visualización de los datos, de la app RemoteSoft

* La clase **VinculacionPulseraActivity** se encargará de la vinculación de las pulseras, al activarse se empezarán a detectar los dispositivos Bluetooth cercanos, al detectar la pulsera pulsamos y nos conectaría llevándonos a la vista de inicio, cabe destacar que al acceder a esta vista se desvinculará si había una pulsera anteriormente.

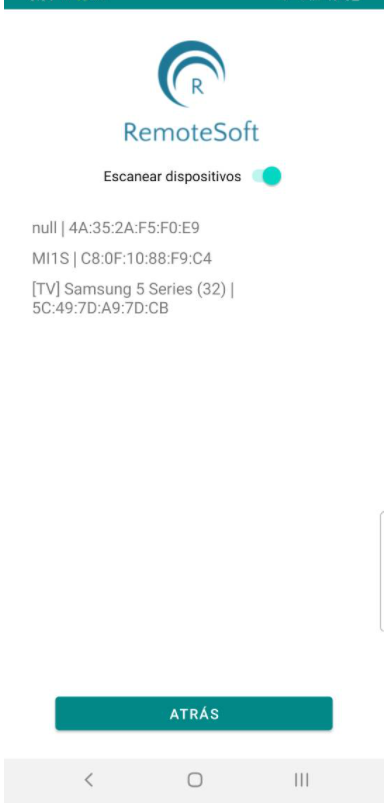


Figura 4.6 Vista de vinculación de pulsera, de la app RemoteSoft

Las siguientes dos clases no son vistas, si no que contienen funciones que se usarán recurrentemente en la aplicación.

* La clase **Pulsera** representa la pulsera utilizada por la persona dependiente, conteniendo todo lo necesario para manejar la vinculación de la pulsera y el cálculo de pulsaciones.
* La clase **DetectaCaida** es la encargada detectar una caída mediante el algoritmo de caída, basado en los datos obtenidos del acelerómetro, en el apartado de implementación se explicará en profundidad el algoritmo.

### RemoteSoft Receives

Como podemos ver en la imagen hay un total de 5 clases, que son 5 vistas que contiene la aplicación, vamos a comentar lo que contiene cada clase.

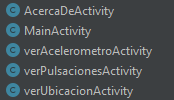


Figura 4.7 Clases utilizadas en la app RemoteSoft Receives

* La clase **MainActivity** es la clase que se mostrará al abrir la aplicación, contiene un botón para acceder a la vista de acerca de, otro botón para configurar el ThingSpeak como en la otra aplicación, para elegir a la persona dependiente que se va a monitorizar, una vez configurado tenemos un switch que al activarlo empezará a recibir datos y monitorizar a la persona dependiente, cuando está activo aparecen 3 botones más, para visualizar los datos obtenidos.

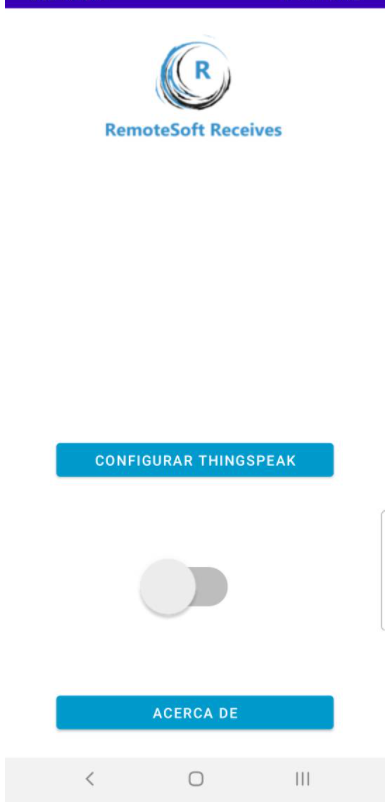
 

Figura 4.8 Vista principal de la aplicación RemoteSoft Receives, con y sin el switch principal activado

* La clase **AcercaDeActivity** es simplemente una clase informativa, que muestra información básica sobre la app, el autor de quien lo ha desarrollado y requisitos para que funcione la aplicación en su totalidad, al igual que en la otra aplicación.

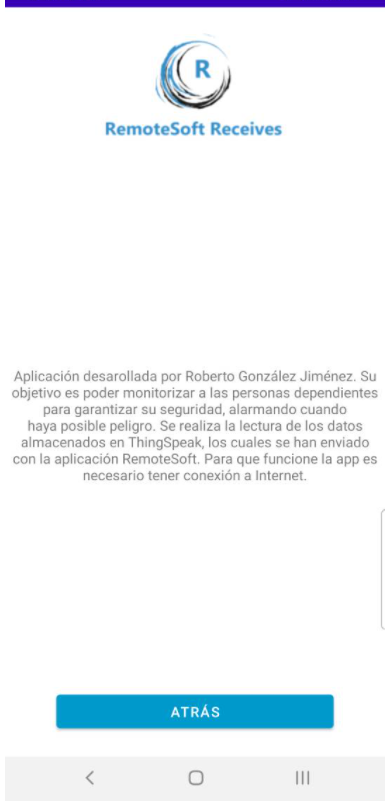


Figura 4.9 Vista Acerca de, de la app RemoteSoft Receives

* La clase **verAcelerómetroActivity** muestra las aceleraciones en las coordenadas X, Y y Z de la persona dependiente y además si ha detectado una caída.

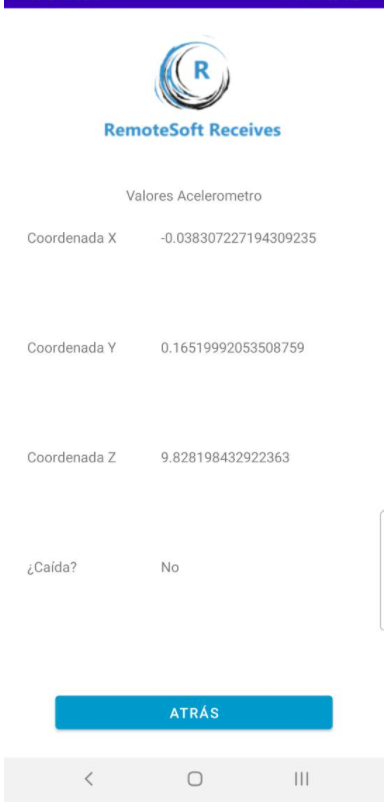


Figura 4.10 Vista de visualización de los datos recogidos por el acelerómetro, de RemoteSoft Receives

* La clase **verPulsacionesActivity** muestra las pulsaciones de la persona dependiente.

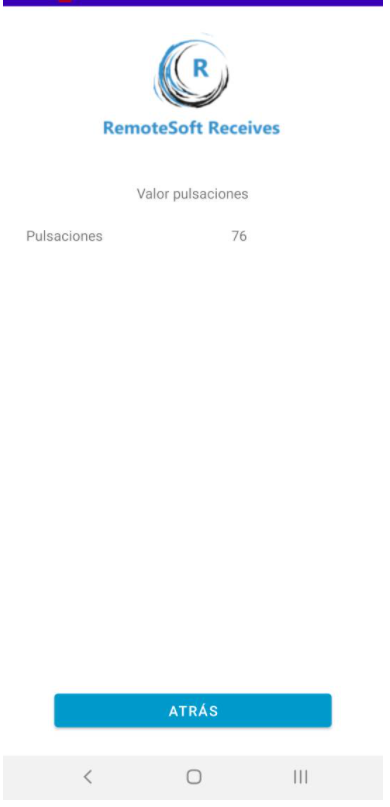


Figura 4.11 Vista de visualización de las pulsaciones, de RemoteSoft Receives

* La clase **verUbicacionActivity** muestra la ubicación exacta en el mapa de la persona dependiente.

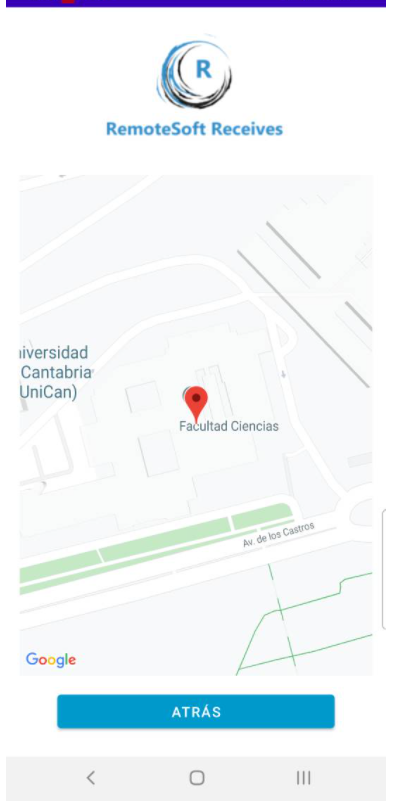


Figura 4.12 Vista de la ubicación, de RemoteSoft Receives

## Diseño detallado

En la Figura 4.13 se puede apreciar cómo funciona el intercambio de mensajes mediante ThingSpeak en entre las 2 aplicaciones, se explicará a continuación cual es el proceso que se lleva a cabo detalladamente.

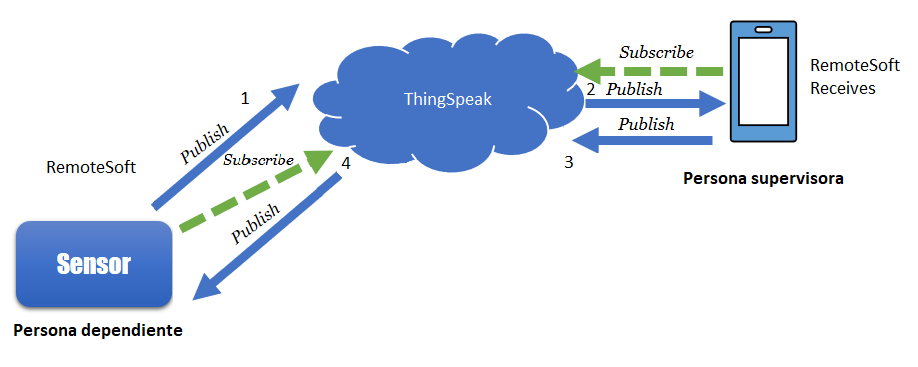


Figura 4.13 Vista detallada de las aplicaciones

Inicialmente ambas aplicaciones deben configurarse a los canales de ThingSpeak de los sensores y de monitorización que pertenecen al usuario, como se explicará más adelante en el apartado de implementación 5.4.

Una vez configurados correctamente los canales, la comunicación entre las aplicaciones se realiza siguiendo los pasos descritos en la Figura 4.13:

1. Se enciende el switch que envía al canal de ThingSpeak toda la información, los sensores, si ha detectado caídas y la ubicación, realizando un publish cada 20 segundos.
2. Desde la aplicación RemoteSoft Receives encendemos de igual manera el switch, una vez configurado los canales de ThingSpeak, entonces se suscribirá a dicho canal, en cuanto la otra aplicación envíe un mensaje, este lo detectará y recibirá dicho mensaje en formato JSON.
3. Como está el switch activo, quiere decir que hay alguien monitorizando a la persona dependiente, esto lo tenemos que informar a la otra aplicación, por lo que realizamos un publish cada 15 segundos informando que hay alguien vigilando.
4. Desde la aplicación RemoteSoft solamente enviando los datos no podemos saber si hay alguien monitorizando, por lo que tenemos que suscribirnos al canal de ThingSpeak que contiene dicha información, ya que sabiendo si hay alguien monitorizando o no impediremos que suenen las alarmas cuando no hay nadie mirando, se guardarán los datos hasta que monitorice alguien.

# Implementación

Una vez establecidos los requisitos y el diseño de las aplicaciones, procedemos a describir la implementación realizada en lenguaje Java de los componentes software para satisfacer las necesidades de los usuarios.

## Handlers

Anteriormente se ha comentado acerca de funciones que se ejecutan de manera periódicas o que se ejecutan después de un periodo de tiempo determinado, en Android esto se realiza mediante Handlers.

Los handlers[12] tienen dos usos principales: uno para programar mensajes y ejecutables para que se ejecuten en algún momento en el futuro y otro uso es poner en cola una acción para que se realice en un hilo diferente al suyo.

Entonces al llamar a los handlers se le asignarán objetos de clase Runnable, todos los objetos de la clase Runnable deben de tener el método llamado run(), que ejecutará el código que lleva dentro de este método en un hilo diferente.

El handler utilizará el método postDelayed, que ejecutará el método run() del Runnable pasado como parámetro después de una cantidad determinada de tiempo en milisegundos pasado como parámetro de tipo long.

*postDelayed(Runnable, long)*

La manera de hacer que un método se repita cada cierto tiempo de manera periódica consiste en que antes de terminar el método run() el handler vuelva a realizar un postDelayed del mismo Runnable[13] (pasándole como parámetro Runnable this) y de segundo parámetro el tiempo en milisegundos que se repetirá la acción una y otra vez.

En nuestras dos aplicaciones hemos utilizado los Handler para un total de 5 Runnables que tratan de 5 funciones distintas, se listan a continuación:

1. Antes de empezar a medir las pulsaciones, cuando se necesitaba esperar 1 segundo tras añadir el perfil de la persona ya que necesitaba unos milisegundos para aplicarse y poder empezar a medir las pulsaciones
2. Medición de las pulsaciones con la pulsera de actividad, se ejecuta periódicamente cada 14 segundos.
3. Enviar los datos de todos los sensores al canal de ThingSpeak, se ejecuta periódicamente cada 20 segundos.
4. Desde la aplicación del supervisor, para informar a la otra app por ThingSpeak que se está monitorizando, se ejecuta periódicamente cada 15 segundos.
5. Cuando se deja de monitorizar se debe informar a la otra app por ThingSpeak, no es periódica y se ejecuta tras 15 segundos para asegurar que no coincida con la señal de que se está monitorizando sin querer y ThingSpeak lo pueda enviar correctamente.

La manera de parar los handlers al instante es con el método removeCallbacks(Runnable) indicándole lo que se quiere parar, inmediatamente deja el método run() que se está ejecutando.

## Detección de caídas

Para garantizar la seguridad de la persona dependiente se ha realizado un algoritmo para comprobar si la persona dependiente ha sufrido una caída, para este algoritmo se basa en las medidas proporcionadas por el acelerómetro incorporado en el teléfono móvil.

El acelerómetro de los móviles básicamente nos dice en que orientación está colocado el dispositivo, se puede saber si está en vertical, horizontal e incluso si esta boca abajo, el acelerómetro consta de dos placas metálicas, una placa móvil que se mueve dependiendo de la aceleración que le apliques, y de otra placa fija que interpreta el voltaje resultante de este movimiento para determinar la aceleración a la que lo hace y su orientación.

Para medir esto el acelerómetro del teléfono permite medir la aceleración en 3 dimensiones, X, Y y Z, que miden el movimiento en un espacio tridimensional.

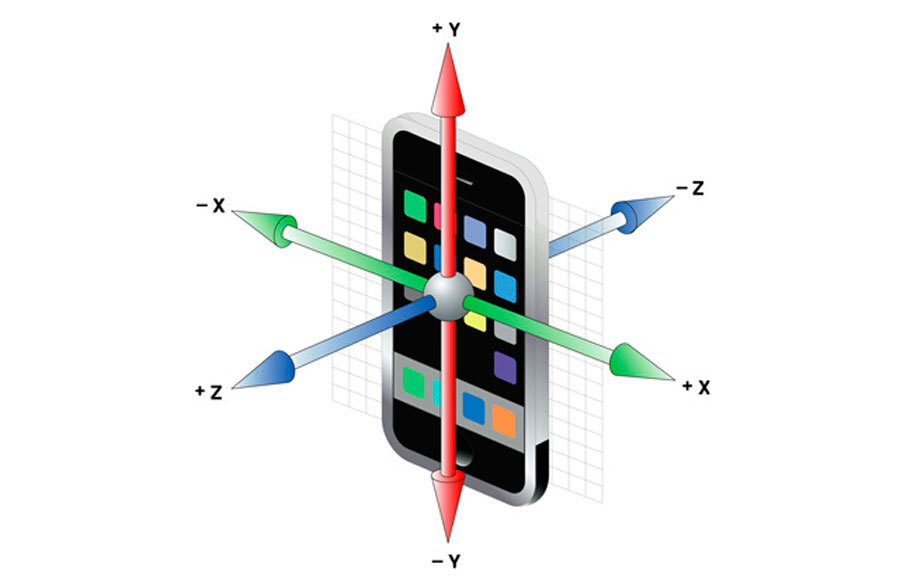


Figura 5.1 Calculo de los valores de los 3 ejes del acelerómetro

Para calcular los valores de aceleración en los 3 ejes es necesario crear un *listener*, este *listener* se puede configurar para controlar los eventos de todos los sensores del teléfono, en nuestro caso elegiremos el acelerómetro. Registramos el listener con un periodo de activación para calcular valores, en nuestra aplicación hemos elegido un tiempo de aproximadamente cada 215-230 ms, entonces el listener cada este tiempo nos devolverá un Array con los valores de la aceleración en los ejes X, Y y Z en cada instante, que posteriormente serán enviados y mostrados, estos valores también los usaremos para el detector de caídas.

Para el algoritmo detector de caídas primero calculamos el módulo de la aceleración que nos da una idea de lo rápido que varía su velocidad. Por ejemplo, si un cuerpo lleva una aceleración de 2 m/s² significa que aumenta su velocidad 2 m/s cada s. El módulo de la aceleración se obtiene mediante la raíz de la suma del cuadrado aceleraciones en los tres ejes de detección:

Entonces nos apoyaremos con la clase que he hablado previamente llamada **DetectaCaida**, la forma de detectar una caída es gracias a una ventana en la que vamos añadiendo las aceleraciones calculadas en cada instante, el tamaño de la ventana se puede configurar, pero por defecto lo creamos a 10 valores. La ventana contiene una Linked List[14] de valores Double (la aceleración es un valor Double), en cada instante que se activa el listener calcula el módulo de aceleración en ese momento y lo añade a la lista, cuando la lista se llena se borra el primer valor de la lista, que es el más antiguo y se añade el nuevo valor al final, por eso hemos usado una Linked List (otra opción habría sido utilizar una cola circular, se eligió la Linked List por su simplicidad de uso).

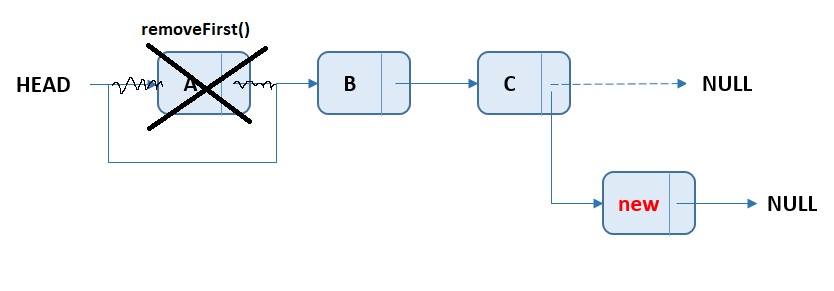


Figura 5.2 Funcionamiento de la Linked List

En cada ejecución del listener del acelerómetro también se hace la comprobación de detección de caída, esta comprobación lo que hace es de todos los valores del acelerómetro en la ventana coge el mayor valor y el menor valor de la aceleración, entonces hace dos comprobaciones, estas comprobaciones se realizan cuando la ventana está llena de valores.

La primera comprobación es que primero se haya detectado el valor de aceleración más pequeño antes que el mayor, ya que para una caída primero se está con un valor de aceleración normal y cuando te caes, es un movimiento brusco, por lo que hace que el módulo de aceleración incremente drásticamente y se tiene que detectar después.

La segunda comprobación es calculando la diferencia de aceleración entre el valor más pequeño y el mayor, esta diferencia tiene que ser superior a un umbral ya que si no es superior al umbral no se podría detectar la diferencia entre una caída y un cambio de aceleración, en nuestro caso la diferencia de aceleración tiene que ser más de 10 m/s² para que sea una caída.

Si se cumplen estas dos comprobaciones satisfactoriamente se ha detectado una caída y se podrá enviar el dato a la persona supervisora.

## Pulsera Xiaomi

La pulsera de actividad de Xiaomi es necesaria para el cálculo de pulsaciones, el modelo utilizado de pulsera es Xiaomi Mi Band 1s[15] , que se comunica mediante Bluetooth con los teléfonos móviles. Se ha utilizado este modelo por la simplicidad, otros modelos tienen muchas más funciones que se van a usar y la función principal de nuestra pulsera es medir pulsaciones que es para lo único que la necesitamos.



Figura 5.3 Pulsera Xiaomi Mi Band 1s

Para acceder a esta pulsera cuando se compra es necesario la app oficial de Xiaomi para poder controlarla, no obstante nosotros tenemos que controlar la pulsera desde nuestra app para poder acceder a los datos y conectarnos correctamente, por lo tanto con la aplicación de Xiaomi no nos sirve y tendremos que utilizar un SDK no oficial e implantarlo en nuestra app, en este proyecto se ha utilizado la SDK del usuario llamado **pangliang**, que deberemos importar la librería de su proyecto al nuestro para poder utilizar sus clases y métodos.

Esta librería permite conectar nuestra pulsera con el teléfono móvil, se crea un objeto de la clase MiBand que pertenece a esta librería en la que se conecta a un objeto de la clase BluetoothDevice, de la librería Bluetooth de Android, una vez conectada a ese dispositivo hay una lista de métodos que podemos usar, el utilizado es lógicamente el de calcular pulsaciones.

Para la conectar la pulsera se realizará mediante el protocolo Bluetooth, por lo que tendremos que darle a nuestra app el permiso de usuario de utilizar Bluetooth.

Para la búsqueda y conexión de dispositivos se utiliza la clase **VinculacionPulseraActivity**, como se ha explicado en el apartado de diseño, esta clase es una vista donde se encuentran los diferentes dispositivos Bluetooth, para ello en la clase existe una lista en la que iremos guardando los dispositivos y se mostrarán en pantalla, por lo que, con el Bluetooth encendido, se llamará a un método de la SDK no oficial de la pulsera llamado startScan() que busca dispositivos Bluetooth, en nuestro caso buscará encendiendo el switch de la vista y de la que va detectando dispositivos los va añadiendo a la lista, importante destacar que al apagar el switch se dejará de buscar y borrara todos los dispositivos de la lista.

Con la lista de dispositivos podremos elegir a cuál nos queremos conectar, solo se podrá conectar a nuestro modelo de pulsera de Xiaomi obviamente, para elegirlo simplemente seleccionaremos el dispositivo deseado tocándolo en la pantalla táctil del teléfono, una vez elegido el dispositivo ya no necesitaremos estar más en la vista de vinculación por lo que se volverá a la vista principal pasándole el dispositivo que hemos seleccionado. Cabe destacar que si queremos desvincular el dispositivo y conectarnos a otro simplemente con volver a esta actividad se desconectará la pulsera conectada y se podrá conectar a otros dispositivos.

La vista principal detectará que se ha recibido un dispositivo de la otra actividad por lo que procedemos a conectarnos, para ello nos ayudaremos con la clase **Pulsera** mencionada en el apartado anterior. Llamaremos al método de connect(), de la librería de la pulsera, mientras se conecta aparecerá una ventana de progreso, que se te retirará una vez conectada correctamente.

Una vez conectada a la pulsera para que funcionen sus métodos hay que añadir un perfil de información sobre la persona con datos como altura, edad, género y peso. En nuestra aplicación solamente utilizaremos la pulsera para calcular las pulsaciones por lo que estos datos son innecesarios para esta función, pero es obligatorio añadirlos para que funcione, por lo que se añaden datos aleatorios. Estos datos tardan en aplicarse unos milisegundos a la pulsera, por lo que hay que esperar un tiempo de aproximadamente 1 segundo para poder empezar a medir las pulsaciones.

Es necesario establecer el *listener* al evento del cálculo de pulsaciones para que cuando se haga una lectura de pulsaciones permita actualizar los valores a los nuevos, este listener pertenece también a la librería de **pangliang** y se llama HeartRateNotifyListener, cada vez que se llame al método de cálculo de pulsaciones el listener actualizará el valor de las pulsaciones al detectado por la pulsera. Este método se ejecutará cada 14 segundos, que es aproximadamente lo que puede tardar de máximo en medir las pulsaciones.

## Ubicación

Otra información importante es saber dónde está la persona dependiente, por si se ha escapado o perdido la persona supervisora necesita saber esta información, por lo que tendremos que hallar la ubicación del teléfono móvil.

Para poder integrarlo en nuestra app, es necesario darle el permiso de usuario de la ubicación a nuestra aplicación, también al ser la ubicación un dato muy personal, por motivos de privacidad se necesita una confirmación en las apps para poder acceder, que es obligatorio implementar en la app, preguntará si le damos permiso a nuestra app RemoteSoft a utilizar la ubicación, en caso de que lo que se deniegue no podrá obtener la ubicación del dispositivo.

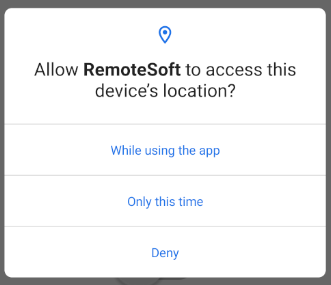


Figura 5.4 Permiso de ubicación en Android

Una vez tenemos el permiso de ubicación podremos empezar a calcularla, muy importante activar el GPS del móvil para que funcione, como es lógico.

Existen diferentes proveedores para obtener la ubicación del móvil en Android, dependiendo de la precisión que se necesite, estos son:

* **Proveedor GPS**: Tiene una precisión de aproximadamente 5 metros, es muy preciso, pero consume mucha batería, además tarda bastante en calcular la ubicación una de las causas de esto es la gran precisión que dispone.
* **Proveedor red**: Tiene una precisión de aproximadamente 50 metros, no es tan preciso como el anterior pero aun así es bastante eficaz, no gasta tanta batería y utiliza el GPS del móvil junto al internet, esto hace que tarde poco en calcular la ubicación.
* **Proveedor pasivo**: La precisión es de aproximadamente 1km y medio, para nada preciso y no utiliza el GPS para calcular la ubicación, lo calcula con los puntos de acceso Wifi cercanos. Apenas consume batería, aunque da el valor rápido.

El pasivo para nuestra aplicación está descartado, ya que necesitamos más precisión y de los otros dos al final se ha elegido el proveedor red, ya que con su precisión nos sirve, además que no consume tanta batería como el proveedor GPS y este además tardaba mucho en mostrar los valores.

De la misma manera que anteriores sensores, para la ubicación también hemos utilizado un *listener* que controla el evento de los cambios en la ubicación del dispositivo para posteriormente enviarlos a la persona supervisora, para enviar esta ubicación se separa en las coordenadas de latitud y longitud.

Desde la aplicación de la persona supervisoras podremos visualizar la ubicación en un mapa para saber dónde esta, se mostrará en el mapa de Google Maps, para ello necesitaremos el permiso de usuario de ubicación. Automáticamente el mapa mostrará haciendo zoom a la ubicación según la latitud y longitud de la persona dependiente, utilizando un marcador a la última ubicación de la persona dependiente, controlando de manera más visual donde está.

## ThingSpeak

Como ya se ha hablado en anteriores puntos, ThingSpeak es la aplicación utilizada en este trabajo para almacenar la información necesaria para supervisar a las personas dependientes. Hemos utilizado esta aplicación por diversas razones, principalmente porque utiliza el protocolo MQTT para la comunicación, existen muchas aplicaciones que utilizan este protocolo y se ha elegido esta por su simplicidad e intuitividad ya que hay que ponérselo fácil las personas que utilicen estas aplicaciones además la interfaz del sitio web es muy cómoda.

El protocolo MQTT funciona como un servicio de mensajería con patrón publicador/suscriptor (*pub-sub*), hay una persona que publica sus datos en el gestor de datos MQTT, en nuestro caso ThingSpeak, los mensajes publicados se organizan en *topics*, un cliente puede publicar un mensaje en un determinado *topic* y otros clientes se pueden suscribir a ese topic y el bróker le hará llegar los mensajes que vayan llegando mientras esté suscrito.

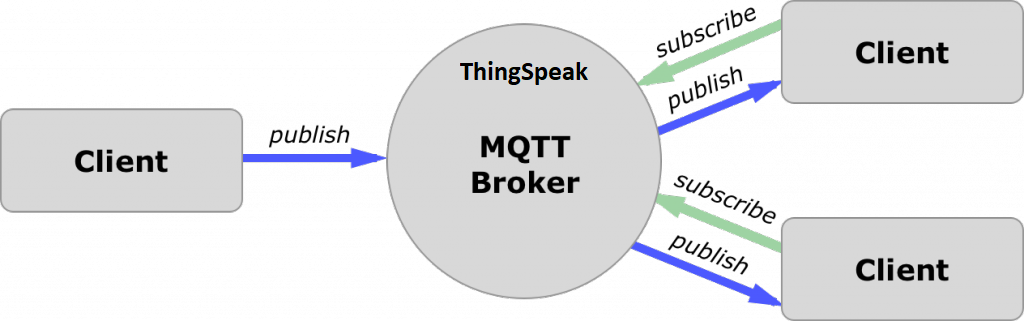


Figura 5.5 Funcionamiento del protocolo MQTT

Para la implementación como se ha mencionado previamente, utilizaremos la librería de Eclipse Paho, que sirve en general para el protocolo MQTT nos permite conectarnos a cualquier servidor que utilice MQTT, en nuestro caso es la de ThingSpeak con la siguiente URL:

*tcp://mqtt.thingspeak.com:1883*

Como base de comunicación utiliza el protocolo TCP/IP, seguido de la dirección del servidor y del puerto 1883, que es el que utiliza el protocolo TCP, para que esto funcione es muy importante darle a la app los permisos de usuario de Internet y a la aplicación indicar que usa el servicio MQTT.

ThingSpeak organiza su información en canales, para cada canal puede haber un total de 8 campos y en la versión gratuita de la aplicación se pueden crear un máximo de 4 canales y se puede enviar mensajes cada 15 segundos.

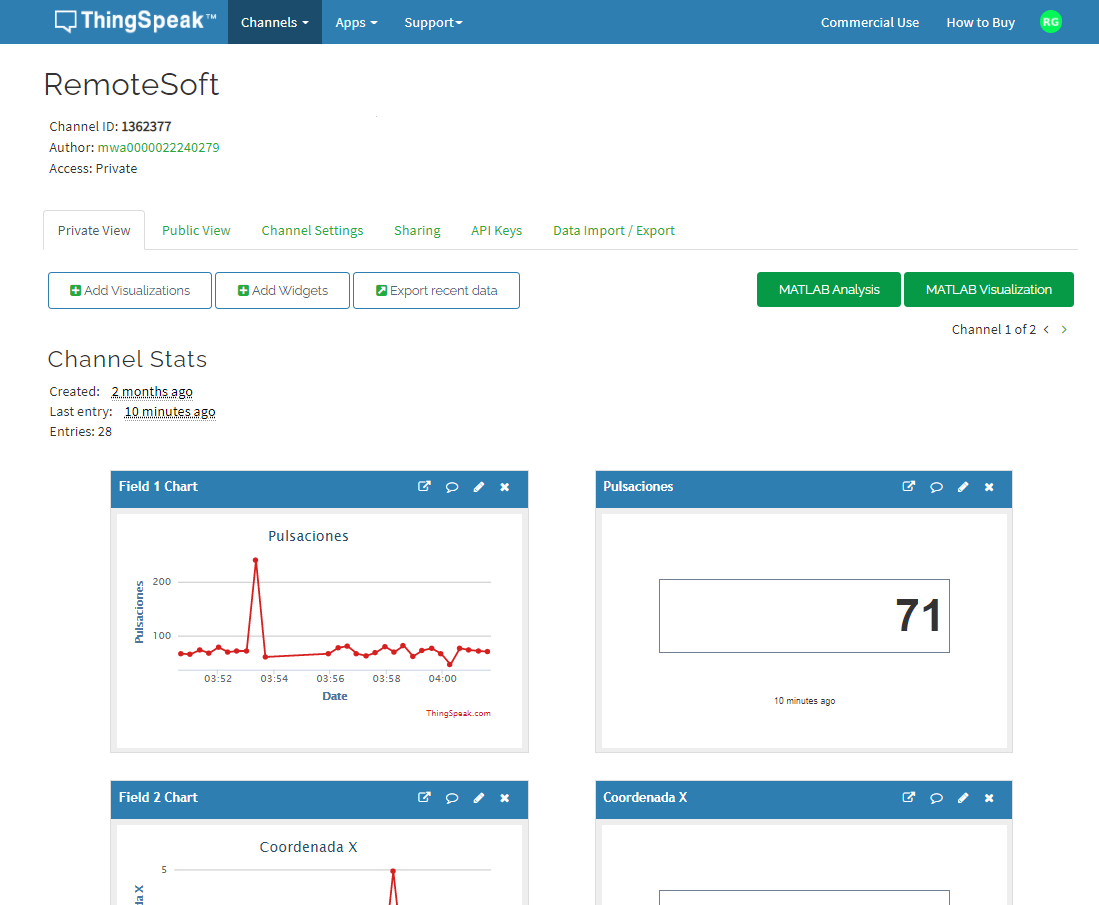


Figura 5.6 Visualización de un canal de ThingSpeak

Para la publicación de mensajes el procedimiento es el siguiente, con la ayuda de las clases de la librería Paho creamos un objeto cliente de la clase MqttAndroidClient[16], pasándole el servidor donde nos queremos conectar, la URL de ThingSpeak mencionada anteriormente y un numero identificador aleatorio para identificar al cliente y llamamos al método connect() para conectarnos al servidor de ThingSpeak.

*client.publish(topic, payload, qos, retained)*

Si se la conexión ha sido exitosa podemos realizar la publicación de mensajes, se publican como se ha mencionado antes en una determinada topic, está compuesta de la id del canal en la que se va a publicar y también esta topic está compuesta de una WRITE API Key que es una clave privada que dispone el canal para permitir publicar mensajes, permitiendo publicar solo a los que dispongan de esta clave.

El contenido del mensaje se envía en el *payload*, en este caso se envían todos los campos, en formato double o String (en teoría se puede en entero también pero no funcionaba correctamente), se indica el número de campo y le sigue el valor que contiene dicho campo, así para todos los campos separados por el símbolo ‘&’. El payload se debe enviar como array de bytes para garantizar mayor seguridad, un ejemplo de payload sería el siguiente:

*("field1=" + valor).getBytes()*

Se dispone de un mecanismo de calidad del servicio o QoS, entendido como la forma de gestionar la robustez del envío de mensajes al cliente ante fallos (por ejemplo, de conectividad), existen tres niveles QoS posibles.

* **QoS 0 unacknowledged (at most one):** El mensaje se envía una única vez. En caso de fallo por lo que puede que alguno no se entregue.
* **QoS 1 acknowledged (at least one):** El mensaje se envía hasta que se garantiza la entrega. En caso de fallo, el suscriptor puede recibir algún mensaje duplicado.
* **QoS 2 assured (exactly one):** Se garantiza que cada mensaje se entrega al suscriptor, y únicamente una vez.

En nuestro trabajo utilizaremos el QoS 0 ya que esperamos que no haya errores de envío en ThingSpeak y no queremos realizar retransmisiones. Y por último un campo retained, que es de tipo boolean y si esta activado se guardará el mensaje después de ser repartido a todos los suscriptores. En nuestra app será siempre false, ya que no está bien implementado en la librería, así sería la publicación de un mensaje.

Para suscribirse en el canal el proceso es bastante parecido al de publicar en cuanto a conectarse al ThingSpeak, solamente que para suscribirse hace falta un usuario y contraseña, el usuario estaría en el perfil de la cuenta de ThingSpeak, disponible en el sitio web y la contraseña sería la MQTT API Key, que es necesaria para poder suscribirte a los canales.

*client.subscribe(topic, qos)*

Una vez conectado habiendo establecidos el usuario y contraseña podremos suscribirnos a los canales, para ello la topic está compuesta de la id del canal, para identificar a qué canal nos conectaremos y de la misma manera que en la publicación existe la READ API Key, que es una clave privada del canal que permite que se suscriban al canal, permitiendo suscribirse solo a los que dispongan de esta clave, el campo qos realiza la misma función que en la publicación pero en la recepción igualmente en este proyecto estará a 0.

Los mensajes irán llegando a medida que se vayan haciendo publish al canal, estos mensajes se podrán elegir en que formato llegan, en nuestro proyecto hemos elegido el formato JSON por comodidad, por lo que cada vez que llegue un mensaje deberá ser parseado al tipo que sea el dato que llega.

Como se ha detallado en el apartado de diseño las dos aplicaciones realizan las funciones de publicar y suscribirse a canales, se indica a continuación lo que se realiza en cada app:

1. RemoteSoft

* Publicación de la información de los datos obtenidos por los sensores y si ha detectado una caída, se realizan múltiples llamadas al publish, que se ejecuta de manera periódica.
* Suscripción a un canal que informa si hay alguien monitorizando, para poder enviar toda la información o guardarla, para que cuando empiece a monitorizar alguien le lleguen las alarmas durante el periodo no vigilado.

1. RemoteSoft Receives

* Suscripción al canal donde se han enviado los datos en la otra aplicación, toda la información necesaria para que la persona supervisora pueda vigilar a la persona dependiente.
* Publicación a un canal si se está monitorizando o no en ese momento, si se está monitorizando se informa de manera constante con publish de manera periódica.

Al final se han necesitado 2 canales ThingSpeak para este proyecto, en el primero es donde va toda la información necesaria para la persona supervisora y en el segundo canal se compone solamente de un campo, que indica si alguien en ese momento está monitorizando, se han tenido que separar en dos canales porque con las limitaciones de 15 segundos de tiempo de espera para enviar mensajes en cada canal se perdían datos.

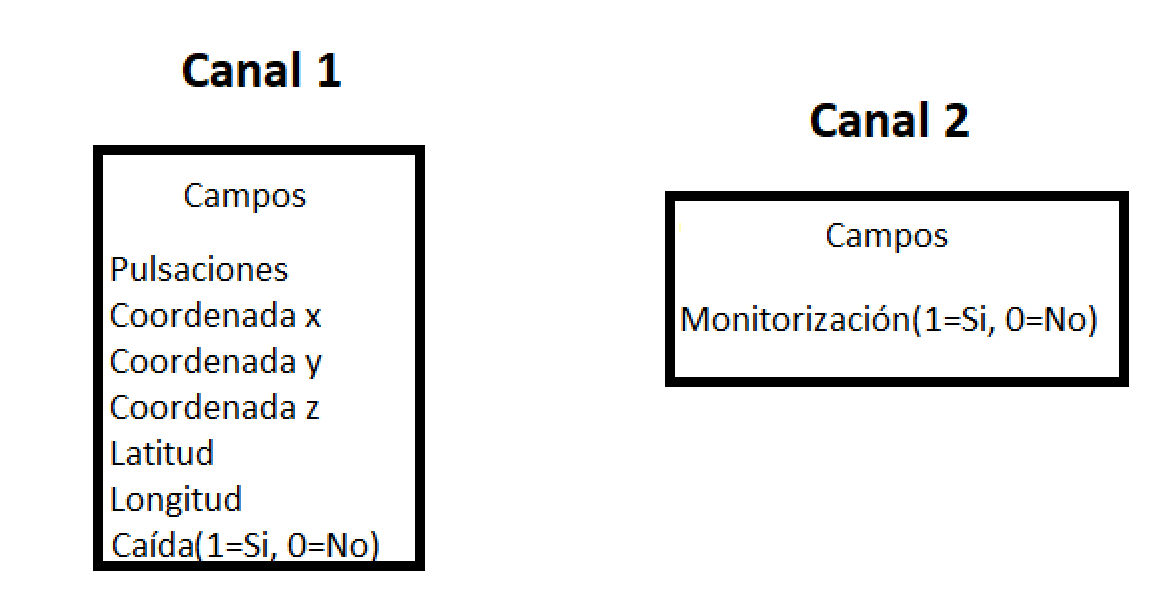


Figura 5.7 Canales junto con los campos utilizados

Como se ha dicho en los requisitos se puede vigilar a más de una persona dependiente, además como cada persona dependiente necesitará sus propios canales se ha implementado una ventana en las dos aplicaciones que permite configurar los canales de ThingSpeak, esto permite que se pueda elegir los canales de ThingSpeak a conectarse , para ello deberá crearse una cuenta en ThingSpeak creando los canales con los campos utilizados y en la aplicación de la persona supervisora simplemente introduce los datos de los canales de la persona dependiente que quiere vigilar.

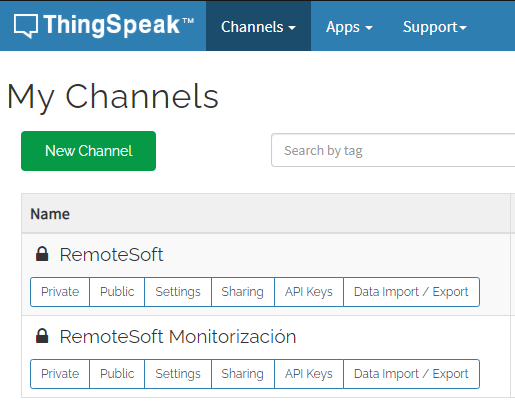


Figura 5.8 Los dos canales necesarios en ThingSpeak

Los datos de los canales se guardarán de manera persistente en las aplicaciones, gracias a la clase SharedPreferences[17] de Android, los datos de los canales se tienen que introducir correctamente, si hay campos en blanco no se aceptarán.

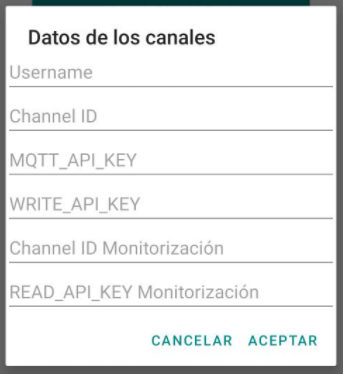


Figura 5.9 Formulario de configuración del uso de canales de ThingSpeak

Para que mientras estemos en la app los sensores y la comunicación no se paren por inactividad debemos añadir un wakelock[18], estos wakelocks permiten que la CPU se mantenga encendida aunque se haya apagado la pantalla, por lo que cada vez que estemos mandando datos a ThingSpeak, es decir con el switch encendido tendremos que adquirir este wakelock para el envío constante de datos y cuando se apague podremos desactivarlo, ya que como no enviamos nada no hace falta y además no se gasta tanta batería.

## Alarmas

La persona supervisora no puede estar todo el tiempo mirando los valores de los sensores esperando a que la persona dependiente muestre algún síntoma de peligro, por lo tanto, se han requerido implementar unas alarmas.

Estas alarmas se activan cuando hay un valor fuera de lo común, enviando una notificación al teléfono, mostrando qué tipo de alarma ha sonado.

Hemos implementado dos tipos de alarmas en nuestra aplicación:

* Cuando se detecta una caída.
* Cuando las pulsaciones igualan o superan los 100 latidos por minuto (LPM).

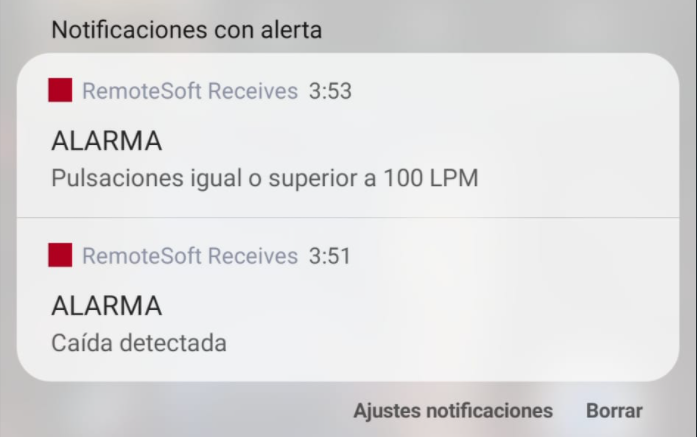


Figura 5.10 Notificaciones de las alarmas implementadas

En los momentos que no haya nadie monitorizando, el supervisor puede perderse algún valor de posible riesgo ya que como su móvil no está recibiendo los mensajes con la información de la persona dependiente las notificaciones no se pueden activar y cuando vuelva a monitorizar ya no están esos valores de riesgo, por lo que se ha creado un atributo que indica si hay alguien monitorizando o no, si hay alguien se enviarían todos los datos de forma normal pero si no hay nadie habría que mantener los datos de posible riesgo hasta que alguien vuelva a monitorizar. El mecanismo utilizado es el descrito a continuación:

* Para la caída, si se ha detectado una caída mientras no haya nadie supervisando el valor permanecerá en Si hasta que monitorice alguien, en el momento en el que haya alguien monitorizando se activará la alarma de detección de caída y podrá cambiarse a no.
* Cuando las pulsaciones igualan o superan a 100 LPM, si hay alguien monitorizando se mostraran los valores en cada momento, pero si no se está monitorizando no se actualizarán los valores de las pulsaciones cuando sean inferiores a 100, por lo que si ha superado los 100 LPM mientras no se está monitorizando a la persona dependiente, cuando empiece a monitorizar siempre verá un valor de igual o superior a 100 LPM, por lo que siempre sonará la alarma.

Para la implementación de las notificaciones hemos creado un método auxiliar llamado showNotification.

*showNotification(String title, String message, int reqCode)*

Los parámetros son el primero el título de la notificación, en nuestro caso “ALARMA”, el mensaje indica que tipo de alarma se ha activado y el reqCode sirve para agrupar los tipos de notificaciones, en nuestro caso hay dos tipos, las de caída y las de pulsaciones, por lo que serán los valores 0 y 1, por ejemplo se ha detectado una caída (tipo 0), pues en el siguiente mensaje se detecta otra vez y vuelve a salir la notificación pero al desplegar el menú inicial saldrá solo 1, ya que es del mismo tipo.

El método utiliza la clase Notification Compat que contiene la clase Builder[19] , para crear notificaciones en Android, se crea un objeto de esta clase y hay que establecerle todos los datos, como el icono de notificación, el título, mensaje descriptivo e incluso se puede establecer un sonido de notificación.

Una vez configurada hay que acceder al servicio de notificaciones del dispositivo, con la clase NotificationManager[20] , se establece la importancia de la notificación, que será siempre importante, establecemos el reqCode que se ha hablado previamente y la mostramos, con esto se notificaría de manera correcta a la persona supervisora.

# Pruebas

En este apartado hablaremos sobre todas las comprobaciones y pruebas realizadas en el trabajo, ya que el trabajo se basa en las aplicaciones Android y en la lectura de sensores y envío/recepción de datos, no hay pruebas unitarias, pero hablaré sobre los métodos que he usado para que funcione y muestre los datos correctamente y las comprobaciones que he realizado sobre el código mientras he estado desarrollando las aplicaciones.

## Mecanismos de notificación de Android

Estas se fueron realizando cuando se querían comprobar datos puntuales, o comprobar si el código ejecutado llegaba a determinadas líneas.

Se comprobaron con dos mecanismos de notificación proporcionados por Android, una fue mediante el Logcat[21] de Android Studio y la otra con la clase Toast[22] de Android.

Logcat es el nombre de la herramienta para acceder al registro de mensajes del sistema de Android. Los desarrolladores de aplicaciones pueden usar este registro para imprimir mensajes útiles para consultar el estado y posibles problemas que sucedan con la aplicación. Para enviar el mensaje al registro se realiza de la siguiente manera:

*Log.d(TAG, String);*

El parámetro TAG se establece como atributo de manera pública con el nombre de la app, para agrupar todo el apartado de depuración en la misma etiqueta y con esto se iban realizando comprobaciones y depurando el código.

Otra manera de depurar es con un sistema de notificaciones que usado en las aplicaciones de Android llamado Toast, es un método de informar mostrando una pequeña ventana emergente.

Se ha usado en el desarrollo para comprobar datos como por ejemplo cuando se conecta la pulsera mirar si el nombre del dispositivo es correcto y también en la aplicación se usan diferentes Toast para indicar que se ha conectado a la pulsera satisfactoriamente y también para cuando se conecta a ThingSpeak.

*Toast.makeText(Actividad.this, String, Toast.LENGTH\_SHORT).show();*

La estructura es la siguiente, el primer parámetro es el contexto de la actividad en la que se está lanzando el Toast, en este caso en la actividad de la vista que estemos, el siguiente parámetro es el String del texto que se va a mostrar y el tercer parámetro es el tiempo que va a estar mostrando la ventana emergente, en el ejemplo es unos pocos segundos y luego con el .show() para mostrarlo en la aplicación.

## Pruebas funcionales

Las pruebas funcionales se realizaron a mano con las aplicaciones Android, comprobando que cumplen todos los requisitos propuestos en el proyecto. A medida que se implementaban las funcionalidades se iban comprobando que las lecturas correspondientes a cada sensor eran correctas y que llegaban sin errores a ThingSpeak.

La manera de realizar estas pruebas es activando todos los sensores, teniendo las dos aplicaciones abiertas, comprobando que los datos son coherentes y que llegan de una aplicación a otra comprobando las vistas que corresponden a la visualización datos de los sensores.

En ThingSpeak también se pueden comprobar los valores de los sensores, hay una gráfica por cada campo en la cual muestra todos los valores que han recibido cada uno.

Se pueden comprobar si son coherentes por ejemplo moviendo el móvil para mirar que los valores del acelerómetro varían, todos los campos se pueden comprobar, la ubicación, pulsaciones incluso podemos simular una caída haciendo movimientos bruscos con el teléfono.

En cuanto a la conexión de la pulsera para saber si se ha conectado a un dispositivo la propia app tiene un cuadro de texto que indica si hay una pulsera vinculada, mostrando el nombre del dispositivo y para saber si está midiendo pulsaciones se puede observar una luz verde por la parte que hace contacto con el brazo.

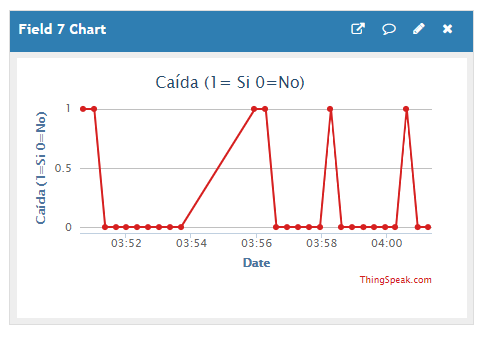


Figura 6.1 Gráfica que muestra las caídas detectadas

# Conclusión y trabajos futuros

En este último apartado se recogen las conclusiones que se han obtenido tras el desarrollo de este proyecto y una colección de posibles cambios, arreglos y mejoras que se podrían aplicar en futuras versiones de las aplicaciones.

## Conclusión

El objetivo de este proyecto ha sido proporcionar un apoyo a las personas que tengan que cuidar a personas dependientes, permitiéndoles tenerles más controlados con el fin de que reaccionar lo más pronto posible a posibles imprevistos que puedan pasar.

Después de mucho trabajo, el proyecto ha logrado cumplir todos los requisitos propuestos para proporcionar esa supervisión a las personas dependientes. En cuanto al desarrollo del proyecto, se puede dividir en tres partes, la primera parte referente a la aplicación RemoteSoft, todo el tema de obtener los datos de los sensores y la pulsera, la segunda parte centrada en MQTT y hacer la conexión mediante ThingSpeak, que se establezca la conexión y se puedan enviar/recibir datos en las aplicaciones correctamente y una tercera parte de hacer funcionar la aplicación de recepción de RemoteSoft Receives y que muestre los datos a la persona supervisora y el tema de notificar con alarmas cuando hay un valor fuera de lo normal.

A nivel personal, la experiencia del TFG ha sido buena, me he tenido que enfrentar a diversos problemas que han ido surgiendo, pero solucionarlos resulta gratificante, te da confianza a ti mismo y poco a poco se va mejorando para alcanzar un alto nivel de conocimientos. Un ejemplo de esto fue que cuando teóricamente había conseguido vincular la pulsera al teléfono no me realizaba ninguna función, fue entonces cuando no me di por vencido y después de muchas horas buscando información y posibles soluciones a mi problema logré solucionarlo, la sensación fue muy positiva hasta tal punto que avancé más rápido en el desarrollo y con mucha motivación. En conclusión, el TFG es una experiencia recomendable y muy positiva tanto para la persona como para aumentar tus conocimientos de cara al futuro.

## Trabajos futuros

Como bien dice el nombre de este proyecto, es un prototipo, por lo que no es una versión final y hay muchas funcionalidades que se pueden añadir y mejorar otras cosas, se puede aprovechar muchos sensores que disponen el teléfono y la pulsera para completarla en un futuro. A continuación, se listan algunas posibles líneas de trabajo futuro:

* **Diferentes tipos de sensores**

Como se ha mencionado anteriormente, el dispositivo móvil dispone de más sensores que los utilizados en esta práctica, como por ejemplo el barómetro, sensor de proximidad o incluso función de termómetro, se podría buscar alguna utilidad para mayor supervisión de la persona dependiente. De igual manera utilizar la pulsera Xiaomi como podómetro o calorías.

Se podrían añadir más dispositivos externos para un mayor control de la persona dependiente, como un dispositivo con sensores médicoscomo tensiómetro, oxímetro y glucómetro, el problema de esto es que habría que buscar si hay alguna librería compatible para controlarlo desde la aplicación.

* **Varios tipos de alarma**

En nuestro prototipo contamos con dos alarmas, una para cuando las pulsaciones son altas y otra cuando detecta una caída. Aunque se podrían avisar de más cosas como por ejemplo estableciendo una zona de seguridad que si la persona dependiente la abandona que notifique a su supervisor o añadir nuevas alarmas con los sensores que he comentado previamente. No sería mala idea añadir varios tipos de alarma según la urgencia de la situación y cada tipo de alarma notifique de manera diferente.

* **Modo de comunicación en la nube**

En este caso hemos usado ThingSpeak que funciona con MQTT, pero también había otras posibilidades como servicios de notificaciones tipo “Google Cloud Messaging” o Firebase, Google Pub/Sub o DDS, si los requerimientos adicionales necesitan otra tecnología para que puede funcionar se podría cambiar de protocolo, incluso existe una versión de pago de ThingSpeak que añade más funciones y permite trabajar con los datos en MATLAB, se podría añadir una aplicación de pago que incluya esto y mejorar las apps para que sea compatible con estas nuevas funcionalidades.

* **Mejorar el funcionamiento en segundo plano**

Las aplicaciones funcionan correctamente, pero si se sale de las apps al cabo del tiempo se apagan los sensores y las apps dejan de enviar a ThingSpeak. Esto se debe al gran consumo de batería que se produce al tener el acelerómetro cambiando de valores repetidamente.

Por lo que en las apps definitivas se podría implantar que al activar el switch de enviar a ThingSpeak las acciones se ejecuten como servicio en segundo plano. En este trabajo como es un prototipo nos vale con que funcione en la aplicación, ya que gracias al wakelock mientras se esté dentro de las apps nunca se apagará la CPU.

* **Automatización de llamada a emergencias**

Como he mencionado anteriormente se podrían añadir varios tipos de alarmas según la urgencia de la situación, por lo que sería buena idea que para las más urgentes directamente llame a los servicios de urgencia, para que acudan donde está la persona dependiente, así evitaríamos posibles desgracias en el caso de que la persona supervisora no esté atenta.

* **Publicación en la Google Play Store**

Si la idea funciona y después de varias mejoras, la versión final este bien optimizada y cómoda para cualquier usuario se podría plantear subirlo a la Play Store para comercializarlo, de manera que todo el mundo pueda usar las funcionalidades y poder supervisar a las personas dependientes que necesiten.

# Bibliografía

[1] Java, lenguaje de programación. [Online]

<https://www.java.com/es/download/help/whatis_java.html>

Cadenhead, R. & Lemay, L. (2008). Programación Java 6. España : Anaya Multimedia

[2] Android, sistema operativo. [Online]

<https://developer.android.com/guide/platform?hl=es-419>

[3] Git, software de control de versiones. [Online]

<https://git-scm.com/about>

[4] MQTT, protocolo de transporte de mensajes. [Online]

<https://mqtt.org>

[5] Eclipse Paho, librería de Eclipse que implementa MQTT. [Online]

<https://www.eclipse.org/paho/>

[6] SDK no oficial para Xiaomi Band 1S de pangliang. [Online]

<https://github.com/pangliang/miband-sdk-android>

[7] Android Studio, entorno de desarrollo integrado para Android. [Online]

<https://developer.android.com/studio/intro>

[8] Sourcetree, entorno gráfico para trabajar con Git. [Online]

<https://www.sourcetreeapp.com>

[9] MQTT.fx, cliente de prueba MQTT, documentación. [Online]

<https://softblade.de/en/mqtt-fx/>

[10] ThingSpeak, plataforma de almacenar datos que utiliza el protocolo MQTT. [Online]

<https://es.mathworks.com/help/thingspeak/>

[11] Microsoft Project, software gestion proyectos. [Online]

<https://www.microsoft.com/es-es/microsoft-365/project/project-management-software>

[12] Documentación Handler Android. [Online]

<https://developer.android.com/reference/android/os/Handler>

[13] Documentación clase Runnable. [Online]

<https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html>

[14] Linked List Documentación. [Online]

<https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

[15] Pulsera de actividad Xiaomi Mi Band 1S. [Online]

<https://www.mi.com/global/miband/#01>

[16] Clase MqttAndroidClient de Eclipse Paho. [Online]

<https://www.eclipse.org/paho/files/android-javadoc/org/eclipse/paho/android/service/MqttAndroidClient.html>

[17] Documentación clase SharedPreferences. [Online]

<https://developer.android.com/training/data-storage/shared-preferences?hl=es>

[18] Documentación WakeLock. [Online]

<https://developer.android.com/reference/android/os/PowerManager.WakeLock>

[19] Clase Builder de NotificationCompat. [Online]

<https://developer.android.com/reference/androidx/core/app/NotificationCompat.Builder>

[20] Documentación Notification Manager de Android. [Online]

<https://developer.android.com/reference/android/app/NotificationManager>

[21] Documentación de uso del Logcat de Android Studio. [Online]

<https://developer.android.com/studio/debug/am-logcat?hl=es>

[21] Documentación del uso del Toast en Android. [Online]

<https://developer.android.com/guide/topics/ui/notifiers/toasts?hl=es-419>