



***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

Co-Director: Mario Aldea Rivas

Co-Director: Héctor Pérez Tijero

Julio - 2021

Índice de Contenidos

Agradecimientos.....	4
Resumen.....	5
Abstract	6
1. Introducción	7
1.1 Motivación	7
1.2 Objetivos	7
1.3 Metodología de trabajo	7
2. Tecnologías y herramientas utilizadas	9
2.1 Tecnologías.....	9
2.2 Herramientas.....	10
3. Análisis de requisitos	12
3.1 Requisitos funcionales.....	12
3.2 Requisitos no funcionales.....	13
4. Diseño de la aplicación.....	14
4.1 Clases utilizadas.....	14
4.1.1 RemoteSoft.....	14
4.1.2 RemoteSoft Receives.....	15
4.2 Diseño detallado	16
5. Implementación.....	17
5.1 Detección de caídas	17
5.2 Pulsera Xiaomi	19
5.3 Ubicación.....	20
5.4 ThingSpeak.....	21
5.5 Handlers.....	23
5.6 Alarmas.....	24
6. Pruebas.....	26
6.1 Pruebas de Android.....	26
6.2 Pruebas funcionales.....	26
7. Conclusión y trabajos futuros	27
7.1 Conclusión	27
7.2 Trabajos futuros.....	28
8. Bibliografía	29

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 si mismas y necesitan estar supervisados por alguna persona para asegurar su salud, entonces hemos desarrollado este trabajo para poder ayudar a las personas supervisoras a que las personas dependientes que cuidan esten seguros 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 movil o tambien con la ayuda de otros mecanismos como la pulseras de actividad de Xiaomi 1S. Los sensores que vamos a utilizar en este trabajo son el acelerómetro y la ubicación captada por el teléfono movil y con la pulsera de Xiaomi utilizaremos su pulsómetro para medir las pulsaciones de la persona dependiente. Después de obtener dichos valores lo podrá visualizar la persona dependiente y la persona que lo supervisa, si hay algún valor fuera de lugar se enviarán alarmas a la persona supervisora, las alarmas utilizadas son cuando la persona dependiente tiene 100 o más latidos por minuto (LPM) o cuando mediante un algoritmo se detecte una caída.

Se han desarrollado dos aplicaciones Android para cumplir este propósito, la primera app se llama RemoteSoft y es la que tendra 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 estan en la nube para ser mostrados en esta app, además notificará las alarmas que he hablado anteriormente cuando haya algún valor fuera de lugar, el protocolo de mensajes usado es el MQTT y se ha utilizado el sitio web llamado ThingSpeak que utiliza dicho protocolo.

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 someone to ensure their health, so we have developed this work to be able to help supervisors 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 mechanisms such as the Xiaomi 1S activity wristbands. 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. After obtaining these values, they can be viewed by the dependent person and the person who supervises it, if there is any misplaced value, alarms will be sent to the supervisor, the alarms used are when the dependent person has 100 or more beats per minute (BPM) or when an algorithm detects a fall.

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 is the one that the supervisor will have installed, you can choose the person to supervise and collect the data that are in the cloud to be displayed in this app, it will also notify the alarms that I have mentioned before when there is some value out of place, the message protocol used is the MQTT and the website called ThingSpeak that uses this protocol has been used.

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

1. Introducción

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

1.1 Motivación

Para contextualizar este trabajo hay que hablar sobre un problema que afecta a muchas familias, como a todo el mundo cada vez que pasan los años nos vamos haciendo cada vez más mayores, algunas personas hasta un punto en el que no se pueden cuidar por si mismas y otras personas que debido a enfermedades o problemas que hayan podido ocurrir, es decir, son personas dependientes.

Desgraciadamente estas personas dependientes necesitan apoyo y siempre hay que estar pendientes de cual 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. Además en la actualidad la tecnología cobra bastante importancia y prácticamente todo el mundo posee de un teléfono móvil con conexión a internet, hemos juntado la problemática de la constante necesidad de supervisar a las personas dependientes y el gran uso e importancia de los teléfonos móviles en la actualidad para desarrollar nuestras 2 apps, que tratan de la supervisión de las personas dependientes gracias a los teléfonos móviles, ambas aplicaciones desarrolladas en Android, ya que es el sistema operativo de móvil más usado mundialmente.

1.2 Objetivos

Como se ha mencionado en el apartado anterior, 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 mecanismos como por ejemplo las pulseras de actividad, que disponen de diferentes funciones para analizar a la persona como por ejemplo el pulsómetro.

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

1.3 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 usabamos para comunicarnos por voz al igual que mediante mensajes de texto, en los canales de texto también lo usabamos 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.

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.
















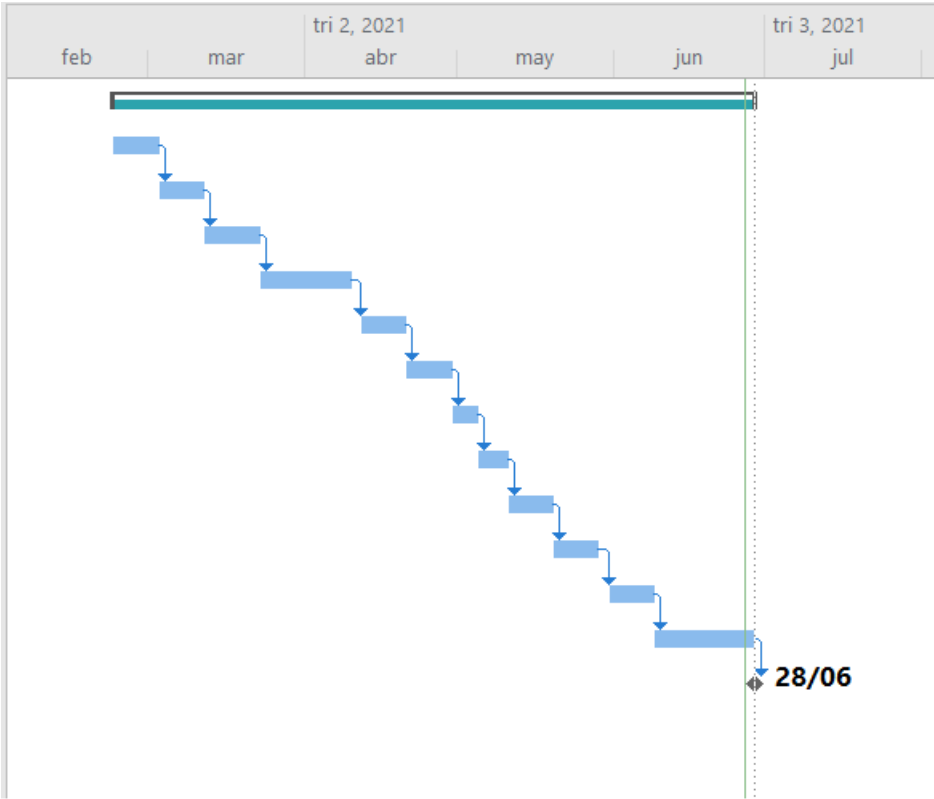
		Modo de	Nombre de tarea	Duración	Comienzo	Fin
1			Prototipo de sistema de teleasistencia para personas dependientes	91 días	lun 22/02/21	lun 28/06/21
2			Establecer entorno de trabajo y creación del proyecto en Android	7 días	lun 22/02/21	mar 02/03/21
3			Diseño primera app y creación de actividades	7 días	mié 03/03/21	jue 11/03/21
4			Funcionalidad de hallar la ubicación y funcionamiento del acelerómetro	7 días	vie 12/03/21	lun 22/03/21
5			Funcionalidad de la lectura de pulsaciones con la pulsera	14 días	mar 23/03/21	vie 09/04/21
6			Envío de los datos a la nube	7 días	lun 12/04/21	mar 20/04/21
7			Posibilidad de elegir a donde enviar los datos y creación de la segunda app	7 días	mié 21/04/21	jue 29/04/21
8			Arreglo de errores del intercambio de datos en la nube	3 días	vie 30/04/21	mar 04/05/21
9			Implementación de lectura de datos en la segunda app y visualizado de ubicación en mapa	4 días	mié 05/05/21	lun 10/05/21
10			Creación de alarmas cuando suben las pulsaciones	7 días	mar 11/05/21	mié 19/05/21
11			Implementado sistema detector de caidas, añadirlo como alarma y optimización de código	7 días	jue 20/05/21	vie 28/05/21
12			Corrección de errores y añadido sistema detector de monitorización	7 días	lun 31/05/21	mar 08/06/21
13			Realizar informe	14 días	mié 09/06/21	lun 28/06/21
14			Finalización TFG	0 días	lun 28/06/21	lun 28/06/21

DIAGRAMA DE GANTT



2. 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.

2.1 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 software de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil como smartphones, tablets, relojes inteligentes, televisores y hasta para algunos automoviles. Inicialmente fue creado por Android Inc., que más tarde fue adquirido por Google, es el sistema operativo móvil mas 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, más adelante 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.

2.2 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. Más tarde se tratará en más 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.



3. Análisis de requisitos

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

3.1 Requisitos funcionales

Los requisitos funcionales definen la funcionalidad que debe de cumplir el software, estos requisitos son definidos una tabla para cada aplicación, una para la aplicación de captura y envío de datos (RemoteSoft) y otra tabla para la aplicación de monitorización de datos (RemoteSoft Receives).

Aplicación de captura y envío de datos RemoteSoft:

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 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 una pulsera Xiaomi para acceder a más sensores, pudiendo ver los dispositivos y cambiar de una pulsera a otra.
RF-005	Gracias a la pulsera, 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	Se configurará el canal de ThingSpeak 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.

Aplicación de monitorización de datos RemoteSoft Receives:

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	Sonará una alarma en la aplicación si las pulsaciones de la persona dependiente son iguales o superiores a 100 latidos por minuto (LPM).
RF-014	Sonará 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 no se pasen por alto las alarmas.

3.2 Requisitos no funcionales

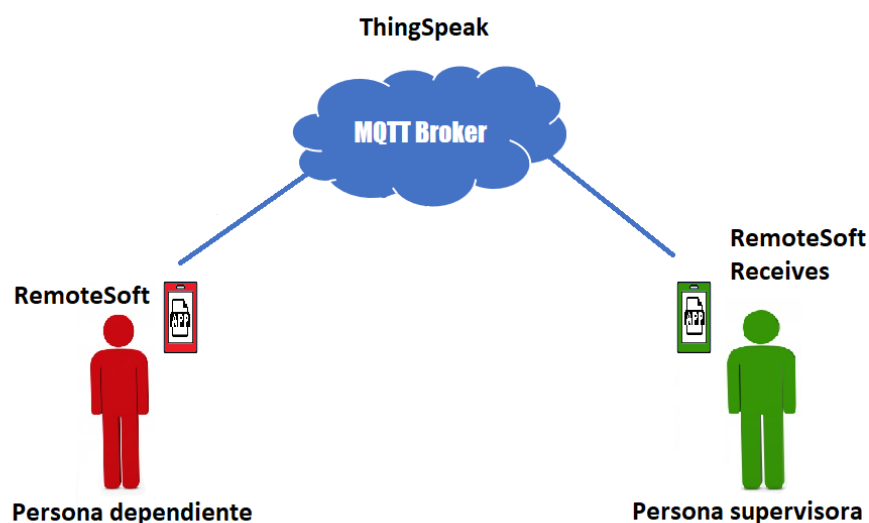
Los requisitos no funcionales definen las características generales y restricciones de las aplicaciones, no tienen que ver directamente con el uso que van a hacer los usuarios, si no que son características que debe cumplir el sistema en conjunto para permitir a los usuarios aprovecharse de las funcionalidades 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 estan 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, siendola API mínima la 30.
RNF-003	Seguridad	No se puede meter todo el mundo a los canales de ThingSpeak, se necesita la ID y las keys de lectura y escritura de cada canal.
RNF-004	Rendimiento	Los datos de los sensores se enviarán a ThingSpeak cada 20 segundos.
RNF-005	Portabilidad	Los datos recibidos se encontrarán en formato JSON y necesitarán ser parseados.
RNF-006	Software	Para el envío y la recepción de datos se usará la librería Paho de Eclipse.
RNF-007	Interfaz	El idioma que se utilizará para las dos aplicaciones será el español.

4. 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.

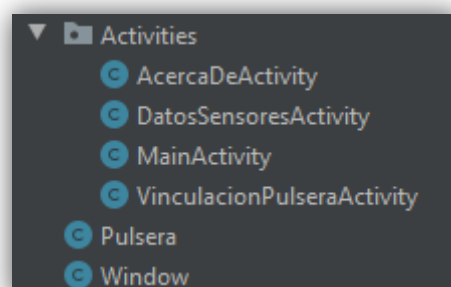


Como se puede ver en la imagen la comunicación entre la persona dependiente y la supervisora se realiza mediante ThingSpeak, que es un broker MQTT ya que utiliza este protocolo para el transporte de mensajes entre los dispositivos.

4.1 Clases utilizadas

4.1.1 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.



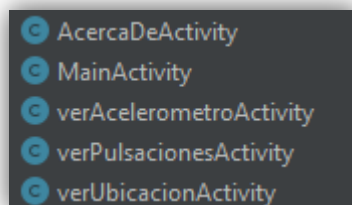
- La clase **AcercaDeActivity** es simplemente una clase informativa, que muestra información básica sobre la app, el autor de quien lo ha desarrollado y requisitos para que funcione la aplicación en su totalidad. (IMAGEN)
- La clase **DatosSensoresActivity** esta clase muestra todos los datos que se recolectan de los sensores, permite a la persona dependiente o usuario de la aplicación que pueda observar los propios datos que se estan capturando en cada momento. (IMAGEN)
- La clase **MainActivity** es la clase que se mostrará al iniciar la aplicación y contiene 3 botones para acceder a las otras vistas, tambien otro botón que sirve para configurar los canales de ThingSpeak que se van a utilizar para el envio/recepcion de datos en la aplicación. Y finalmente tendremos un switch que nos permitirá encender/apagar la app y realizar el envio de los datos de los sensores. (IMAGEN)
- La clase **VinculacionPulseraActivity** se encargará la vinculación de las pulseras, al activarse se empezaran a detectar los dispositivos Bluetooth cercanos, al detectar la pulsera pulsamos y nos conectaría llevandonos a la vista de inicio, cabe destacar que al acceder a esta vista se desvinculará si habia una pulsera anteriormente. (IMAGEN)

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

- La clase **Pulsera** contiene todo lo necesario para manejar la vinculación de la pulsera y el calculo de pulsaciones.
- La clase **Window** es la encargada detectar una caída mediante el algoritmo de caída, basado en los datos obtenidos del acelerómetro.

4.1.2 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.



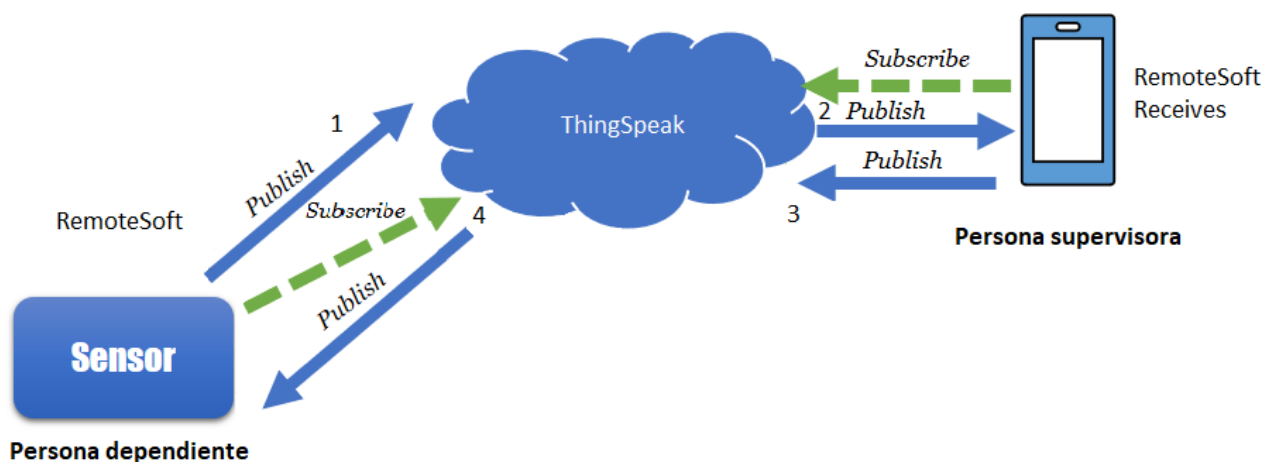
- La clase **AcercaDeActivity** es simplemente una clase informativa, que muestra información básica sobre la app, el autor de quien lo ha desarrollado y requisitos para que funcione la aplicación en su totalidad, al igual que en la otra aplicación.(IMAGEN)
- La clase **MainActivity** es la clase que 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 esta activo aparecen 3 botones más, para visualizar los datos obtenidos.(IMAGEN)
- La clase **verAcelerómetroActivity** muestra las coordenadas X, Y y Z de las persona dependiente y

además si ha detectado una caída.(IMAGEN)

- La clase **verPulsacionesActivity** muestra las pulsaciones de la persona dependiente.(IMAGEN)
- La clase **verUbicacionActivity** muestra la ubicación exacta en el mapa de la persona dependiente.(IMAGEN)

4.2 Diseño detallado

En la siguiente imagen se puede apreciar como funciona el intercambio de mensajes mediante ThingSpeak en entre las 2 aplicaciones, se explicará a continuacion cual es el proceso que se lleva a cabo detalladamente.



1. Inicialmente desde la aplicación RemoteSoft configurada a los canales, se enciende el switch, que envia al canal de ThingSpeak toda la información, los sensores, si ha detectado caidas 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.

5. Implementación

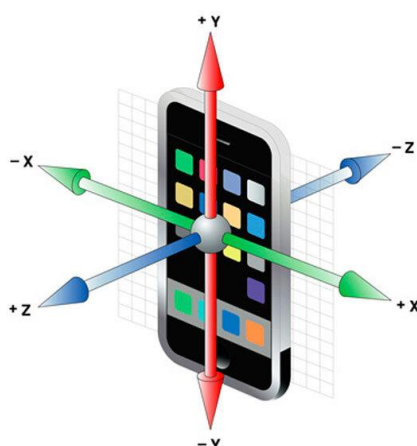
Una vez establecidos los requisitos y el diseño de las aplicaciones, procedemos a implementarlos en lenguaje Java los componentes software para satisfacer las necesidades de los usuarios.

5.1 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 ha necesitado el acelerómetro incorporado en el teléfono móvil.

El acelerómetro de los móviles básicamente nos dice en qué orientación está colocado el dispositivo, se puede saber si está en vertical, horizontal e incluso si está 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 velocidad a la que lo hace y su orientación.

Para medir esto los acelerómetros están compuestos por 3 ejes, X, Y y Z, que son los que permiten medir el movimiento en un espacio tridimensional.

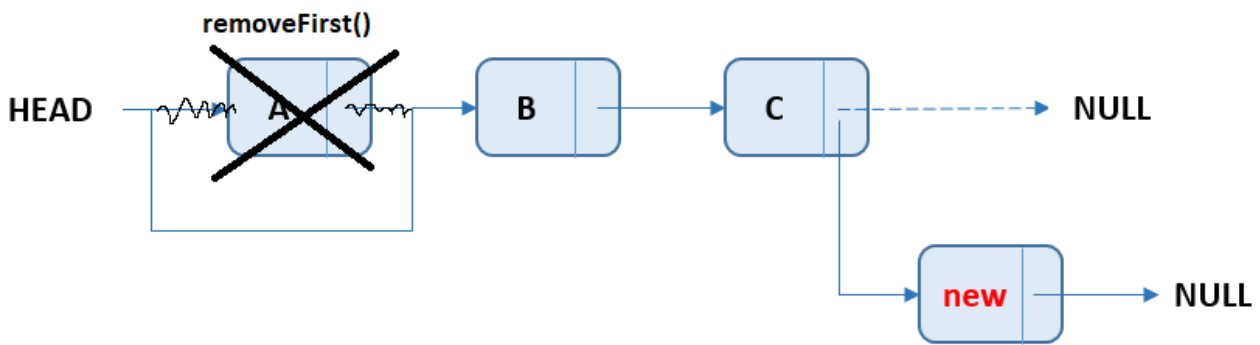


Para calcular los valores de aceleración de los 3 ejes es necesario crear un listener, este listener se puede configurar entre muchas cosas a la lista de sensores del teléfono, en nuestro caso elegiremos el acelerómetro. Entonces registramos el listener con un tiempo de delay para calcular valores, en nuestra aplicación hemos usado aproximadamente cada 215-230 ms y entonces el listener cada este tiempo de delay nos devolverá un array con los valores del acelerómetro X, Y y Z en cada instante, que posteriormente serán enviados y mostrados y también lo 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/s^2 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 de las aceleraciones en los tres ejes de detección:

$$a = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Entonces nos apoyaremos con la clase que he hablado previamente llamada **Window**, 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 `LinkedList`_[12] 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 `LinkedList`.



En cada instante 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/s^2 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.

5.2 Pulsera Xiaomi

La pulsera de actividad de Xiaomi es necesaria para el calculo de pulsaciones, el modelo utilizado de pulsera es XIAOMI MI Band 1S^[13] .



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 y 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.

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 he utilizado la clase **VinculacionPulseraActivity**, como se ha explicado en el apartado anterior, 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 que busca dispositivos Bluetooth, en nuestro caso buscará encendiendo el switch 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 cual 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, 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 también 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 conectar que utiliza la librería de la pulsera, mientras se conecta aparecerá una ventana de progreso, que se te retirará una vez conectada correctamente.

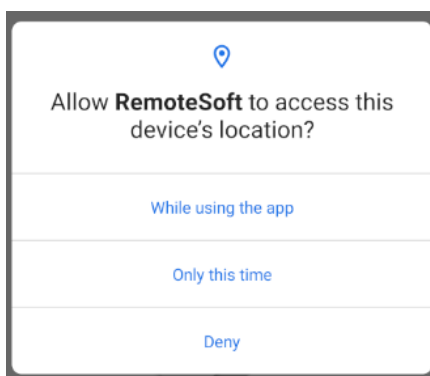
Una vez conectada a la pulsera para poder utilizarla, por defecto hay que añadir un perfil de persona con datos como altura, edad, género y peso. En nuestra aplicación solamente utilizaremos la pulsera para calcular las pulsaciones por lo que estos datos son innecesarios y añadiremos datos aleatorios por defecto. Para calcular empezar pulsaciones es vital dejar aproximadamente un segundo de margen, ya que el perfil de la persona tiene que estar establecido y en aplicarse tarda unos milisegundos.

Es necesario añadir el listener de pulsaciones a la pulsera para que el sensor permita actualizar los valores de las pulsaciones, por lo que creamos un listener al sensor de pulsaciones. Cada vez que se llame al método de calculo de pulsaciones el listener devolverá el nuevo valor de las pulsaciones detectado por la pulsera. Este método se ejecutará cada 14 segundos, que es aproximadamente lo que puede tardar de máximo medir las pulsaciones, la manera de ejecutar periódicamente se explicará en el apartado de Handlers.

5.3 Ubicación

Otra información importante es saber donde esta 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 se deniege no podrá obtener la ubicación del dispositivo.



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 esta 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.

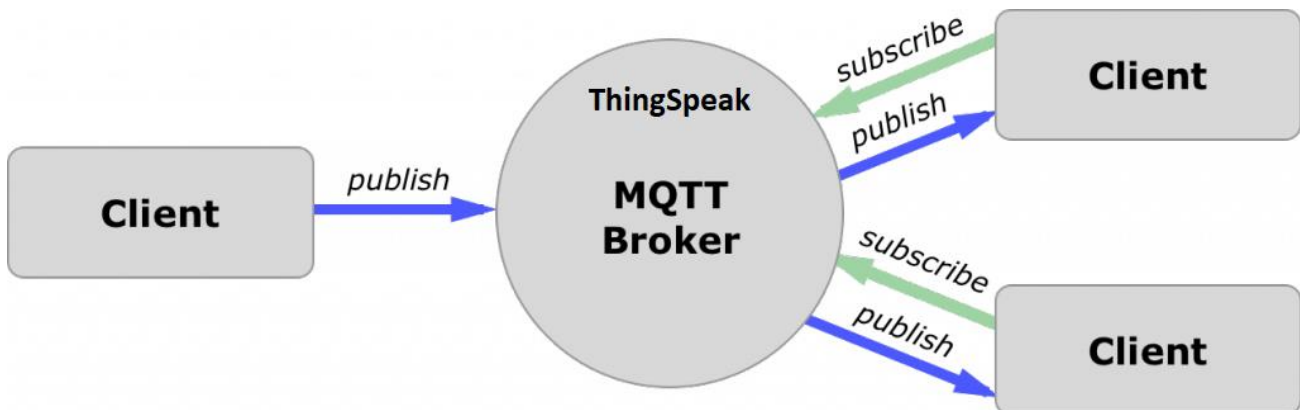
De la misma manera que anteriores sensores, para la ubicación tambien hemos tenido que crear un listener que detecte los valores de la ubicación para posteriormente enviarlos a la persona supervisora, para enviar esta ubicación hay que separarlo en las coordenadas de latitud y longitud.

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

5.4 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.

Esta aplicación utiliza el protocolo MQTT para la comunicación, este protocolo 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 broker le hará llegar los mensajes que vayan llegando mientras esté suscrito.



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 se compone de canales, 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.

(Imagen Página TS)

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`_[14], pasándole el contexto de la actividad actual, el servidor donde nos queremos conectar, la URL de ThingSpeak de antes y un número identificador aleatorio para identificar al cliente y llamamos al método `connect()` para conectarnos al servidor de ThingSpeak.

Si se la conexión ha sido exitosa podemos realizar la publicación de mensajes, se publican como se ha mencionado antes en una determinada topic, esta 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, así se evita que todo el mundo pueda publicar en cada canal.

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

simbolo '&'. El payload se debe enviar como array de bytes para garantizar mayor seguridad.

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 duplicados.
- **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 ultimo 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 esta bien implementado en la librería, así sería la publicación de un mensaje.

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

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.

Una vez conectado habiendo establecidos el usuario y contraseña podremos suscribirnos a los canales, para ello la topic esta compuesta de la id del canal, para identificar a que 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, su función es evitar que todo el mundo se pueda suscribir, el campo qos realiza la misma función que en la publicación pero en la recepción igualmente en este proyecto estará a 0.

client.subscribe(topic, qos)

Los mensajes irán llegando, estos mensajes se podrá 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 multiples llamadas al publish, que se ejecuta de manera periodica.
- 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.

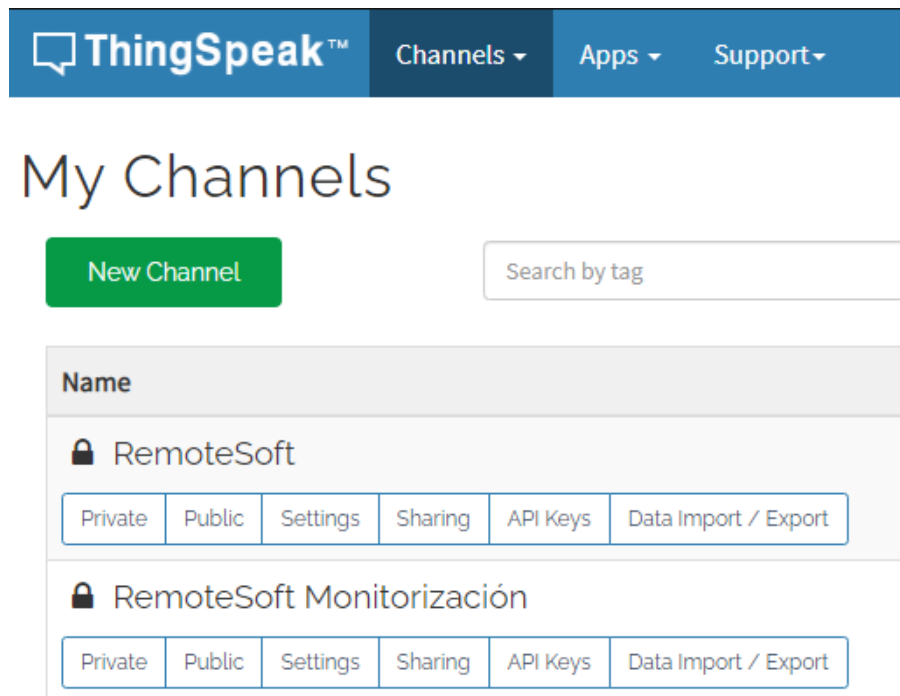
2. RemoteSoft Receives

- 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 esta monitorizando o no en ese momento, si se esta monitorizando se informa de manera constante con publish de manera periodica.

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 por que en el mismo canal los datos chocaban entre sí y con las limitaciones de 15 segundos para enviar mensajes en cada canal se perdían datos.

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



Los datos de los canales se guardarán de manera persistente en las aplicaciones, gracias a la clase SharedPreferences por lo que se guardarán hasta que se cambien los datos, los datos de los canales se tienen que introducir correctamente, si hay campos en blanco no se aceptarán.

(IMAGEN CONF TS)

Para que mientras estemos en la app los sensores y la comunicación no se paren por inactividad debemos añadir un wakelock_[15], 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.

5.5 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^[16] 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 metodo llamado run(), que ejecutará el codigo que lleva dentro de este método en un hilo diferente.

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

postDelayed(Runnable, long)

La manera de hacer que un metodo se repita cada cierto tiempo de manera periódica consiste en que antes de terminar el metodo run() el handler vuelva a realizar un postDelayed del mismo Runnable^[17] (pasandole como parametro Runnable this) y de segundo parametro el tiempo en milisegundos que se repetirá la accion 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 continuacion:

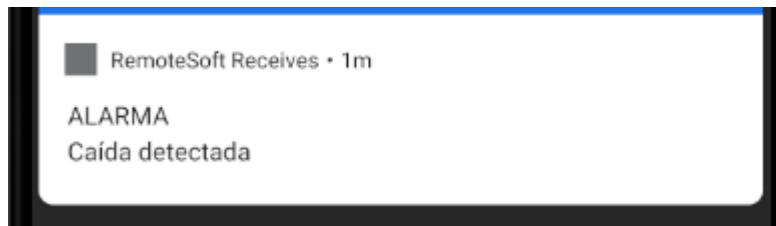
1. Antes de empezar a medir las pulsaciones, cuando se necesitaba esperar 1 segundo tras añadir el perfil de la persona ya que necesitaba unos milisegundos para aplicarse y poder empezar a medir las pulsaciones
2. Medición de las pulsaciones con la pulsera de actividad, se ejecuta periodicamente cada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

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

5.6 Alarmas

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

Estas alarmas funcionan se activan cuando hay un valor fuera de lo común, enviando una notificación al teléfono, mostrando que 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).

Puede ser que no este monitorizando nadie y se falle al notificar al supervisor, ya que por ejemplo puede pasar que no haya nadie y ocurre un valor de posible riesgo, pasan unos 10 minutos y entra la persona y el ultimo valor es un valor normal, esto hace que se haya olvidado una alarma por notificar, por esto se ha creado el campo de si hay alguien monitorizando, para que no haya perdida de alarmas se ha implementado de la siguiente manera en los dos tipos de alarma que tiene la aplicación:

- Para la caída, si se ha detectado una caída mientras no haya nadie supervisando el valor permanecera en Si hasta que monitorice alguien, en el punto en el que haya alguien monitorizando sonará 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 actualizaran 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(Context context, String title, String message, Intent intent, int reqCode)

Los parámetros son el primero el contexto de la aplicación, es decir la actividad en la que se esta lanzando, el titulo de la notificacion, en nuestro caso “ALARMA”, el mensaje indica que tipo de alarma ha sonado, luego el intent se utiliza para ir a otra actividad de la aplicación, en nuestro caso no queremos que lo lleve a ninguna por lo que es un intent vacio 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 seran 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 notificacion pero al desplegar el menu inicial saldra solo 1, ya que es del mismo tipo.

El método utiliza la clase Notification Compat que contiene la clase Builder^[18], 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 titulo, mensaje descriptivo, el intent que llevará cuando se haga click, en este caso a ninguno 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^[19], 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.

6. 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.

6.1 Pruebas de Android

Estas se fueron realizando cuando se querían comprobar datos puntuales, o si el código se ejecutaba diferentes partes del código, se comprobaron de dos maneras, una fue mediante el Logcat^[20] de Android Studio, la mayoría de estas comprobaciones fue depuración por lo que usábamos la llamada a:

```
Log.d(TAG, String);
```

El TAG estando definido de manera pública con el nombre de la app, para agrupar todo el apartado de depuración en toda la aplicación, y con esto se iban realizando comprobaciones y depurando nuestro código.

Otra manera de depurar, es con un sistema de notificaciones que se muestran en las aplicaciones de Android el 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 correctamente a la pulsera satisfactoriamente y si se conecta correctamente 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.

6.2 Pruebas funcionales

Las pruebas de aceptación se realizan a mano con las aplicaciones Android, comprobando que cumplen todos los requisitos propuestos en el proyecto. La manera de realizar estas pruebas es activando todos los sensores, teniendo las dos aplicaciones abiertas y comprobar que los datos son coherentes y que llegan de una aplicación a otra.

Nos podemos ayudar visitando el sitio web de ThingSpeak.com, accediendo con nuestra cuenta a los canales usados para las aplicaciones, ahí se irán mostrando los valores de cada campo que vamos enviando, mirando si los valores son los que corresponden, por ejemplo moviendo el móvil para mirar que los valores del acelerómetro varían, la ubicación correcta, pulsaciones, provocando movimientos bruscos para simular una caída y verificar que si se está monitorizando desde la otra aplicación.
(IMAGEN GRAFICAS TS Caída por ejemplo).

7. 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.

7.1 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.

Con este proyecto he aprendido mucho sobre desarrollar aplicaciones Android, lo cual es muy positivo por si en el futuro tengo que desarrollar aplicaciones, aunque sea otro lenguaje distinto la mecánica de trabajo es parecida con lo que me permitirá avanzar rápidamente en los momentos más problemáticos.

Gracias a trabajar con los sensores del teléfono he aprendido que existen muchas funcionalidades que desconocía en un móvil, aparte de saber como funciona cada una de las que he implementado y como es el código de estas. De igual manera con las pulsaciones de la pulsera, que he aprendido como detectar dispositivos Bluetooth y vincularse con ellos, muy positivo que sea un software externo al teléfono la pulsera Xiaomi, ya que al trabajar con una librería que no sea la oficial de Xiaomi te abre un mundo de posibilidades y mejora la creatividad a la hora de programar.

Respecto al MQTT, es un protocolo de comunicación que esta muy bien para aprender a como funciona el intercambio de mensajes, te da una idea de como es mandar información a la nube para que sea accesible a otros usuarios de manera remota, además que hay muchas compañías que utilizan este protocolo, puede ser que más adelante lo vuelva a usar.

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 habia 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 rapido 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.

7.2 Trabajos futuros

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

- **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 la pulsera Xiaomi como podómetro o calorías. Se podría comprar también para mayor control un dispositivo a parte que tenga sensores médicos como 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 version 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**

La aplicación funciona correctamente pero si se sale de la app al cabo del tiempo se apagan los sensores y las apps deja de enviar a ThingSpeak por el gran consumo de batería 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 y todo funcione hasta cuando se entran en otras aplicaciones hasta que se desactive, pero en este prototipo nos vale con que funcione en la aplicación, ya que con el wakelock mientras se este dentro no se apagará la CPU.

- **Automatización de llamada a emergencias**

Como he mencionado anteriormente se podría 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 esta la persona dependiente, así evitaríamos posibles desgracias en el caso de que la persona supervisora no este atenta.

- **Publicación en la Google Play Store**

Si la idea funciona y después de varias mejoras, la version final este bien optimizada y comoda 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.

8. Bibliografía

- [1] Java, lenguaje de programación. [Online]
https://www.java.com/es/download/help/whatis_java.html
- [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] Linked List Documentacion. [Online]
<https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>
- [13] Pulsera de actividad Xiaomi Mi Band 1S. [Online]
<https://www.mi.com/global/miband/#01>
- [14] Clase MqttAndroidClient de Eclipse Paho. [Online]
<https://www.eclipse.org/paho/files/android-javadoc/org/eclipse/paho/android/service/MqttAndroidClient.html>
- [15] Documentación WakeLock. [Online]
<https://developer.android.com/reference/android/os/PowerManager.WakeLock>
- [16] Documentación Handler Android. [Online]
<https://developer.android.com/reference/android/os/Handler>
- [17] Documentación clase Runnable. [Online]
<https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html>

- [18] Clase Builder de NotificationCompat. [Online]
<https://developer.android.com/reference/androidx/core/app/NotificationCompat.Builder>
- [19] Documentación Notification Manager de Android. [Online]
<https://developer.android.com/reference/android/app/NotificationManager>
- [20] Documentación uso de Logcat Android. [Online]
<https://developer.android.com/studio/debug/am-logcat?hl=es>