Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

DESARROLLO DE APP PARA SMARTWATCH: MONITORIZACIÓN DE PERSONAS VULNERABLES

Autor:
Sergio Sanz Sanz

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

DESARROLLO DE APP PARA SMARTWATCH: MONITORIZACIÓN DE PERSONAS VULNERABLES

Autor:
Sergio Sanz Sanz
Tutores:
Mario Corrales Astorgano
Alfonso Bahillo Martínez

A mi familia y amigos por mostrarme su apoyo durante el transcurso de esta bonita etapa.

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores de este proyecto, Mario Corrales Astorgano y Alfonso Bahillo Martínez, por facilitarme el desarrollo de este proyecto y solucionarme cualquier inconveniente que se ha producido.

En segundo lugar, agradecer a mi familia y amigos su confianza y apoyo mostrado durante estos 4 años y que sin ello hubiera sido muy complicado llegar hasta el final.

Resumen

En los últimos años, la monitorización de la ubicación de personas se ha vuelto cada vez más importante en nuestra sociedad. Desde que comenzó esta práctica en los centros penitenciarios, se ha ido ampliando a otros campos, en muchos de ellos debido a la reciente pandemia.

Son muchos los dispositivos que se han fabricado con el fin de controlar estas situaciones, todos ellos con una gran robustez para evitar su manipulación surgiendo así un gran problema. El estigma que produce a las personas portar este tipo de dispositivos.

Este proyecto, debido al auge que están teniendo en la actualidad los relojes inteligentes y la normalidad de su uso en esta sociedad, propone el desarrollo de una aplicación para este tipo de dispositivos que solucione este problema.

Esta aplicación permite monitorizar la ubicación de la persona que porta el reloj, así como detectar su retirada del cuerpo y poder realizar un aviso al centro de control en el caso de que se produzca una situación de emergencia.

Abstract

In recent years, monitoring the location of people has become more important in our society. Since this practice began in prisions, it has been included in other fields, in many of them due to the recent pandemic.

There are many devices that have been manufactured in order to control these situations, all of them with great robustness to avoid their manipulation, thus arising a great problem. The stigma that people carry this type of devices produces.

This project, due to the boom that smartwatches are currently having and the normality of their use in this society, proposes the development of an application for this type of device that solves this problem.

This application allows monitor the location of the person wearing the watch, as well as detect its removal from the body and be able to notify to the control center in the event of an emergency situation.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

La monitorización continua de la ubicación de personas dependientes se ha convertido en algo fundamental en una gran cantidad de áreas y campos en nuestra sociedad. Debido al avance de la tecnología y la necesidad de garantizar la seguridad y bienestar de los individuos es necesaria esta supervisión continua en distintos ámbitos. Algunos de los más importantes son los siguientes.

- Instituciones correccionales: Los centros penitenciarios son unos de los principales interesados en la monitorización de sus reclusos, principalmente en sus permisos y salidas del centro.
- Centros de salud mental: Cada vez son más los centros que permiten a sus pacientes realizar salidas durante el día y poder realizar actividades. La monitorización en estos casos se vuelve crucial con el fin de evitar situaciones de riesgo.
- Centros de residencia de personas mayores: Interesados en permitir a las personas con capacidades para ello realizar una vida fuera de sus centros, también se vuelve un factor importante la monitorización de la ubicación del individuo durante estas salidas.

Estos son solo algunos de los principales ámbitos donde la monitorización, especialmente de la ubicación del individuo, se vuelve un factor muy importante. La capacidad de conocer la ubicación en tiempo real de los individuos permite evitar situaciones de riesgo y responder de forma inmediata ante situaciones de emergencia.

En la actualidad, marcada por la reciente pandemia de COVID-19 que se inició en 2019 y se extendió a lo largo de 3 años aproximadamente, el control en la localización se ha vuelto aún más relevante en este contexto. Muchos centros de estos campos se han visto obligados a reubicar pacientes en sus correspondientes viviendas familiares con el fin de evitar brotes de contagio en su interior. Surgiendo así la necesidad de tener un control y seguimiento de sus movimientos para evitar situaciones adversas.

Desde hace ya décadas existen en el mercado varias marcas que trabajan con pulseras telemáticas, entre las que se encuentran BI Inc [1] o Track Group [2] a la cabeza. Estas pulseras se encargan de

mantener al individuo localizado de manera constante, basándose en varios componentes electrónicos y sensores como GPS. Además detectan cualquier retirada o manipulación del dispositivo que se desencadena en un aviso a la central de control y activación de una alarma en el propio dispositivo.

La forma de este dispositivo hace que el individuo que la porta no pase desapercibido en nuestra sociedad, ya que cualquier persona puede conocer este dispositivo con el estigma que ello conlleva para el portador.

Por otro lado, los relojes inteligentes han entrado con fuerza en el mercado tecnológico actual. Según varios estudios realizados a la población en Estados Unidos entre 12 y 65 años [3], 68 de cada 100 personas piensan en comprar un reloj inteligente en 2023. Estos dispositivos, entre otras muchas funciones, son capaces de registrar la ubicación del dispositivo y por tanto, monitorizarla. Además, también incluyen una serie de sensores como el acelerómetro o el sensor de proximidad, entre otros, que pueden ayudar a obtener una funcionalidad similar a los dispositivos comentados anteriormente.

1.2. Motivación

En el contexto comentado anteriormente, surge la necesidad de realizar un proyecto que sustituya estos dispositivos, especialmente en algunos campos. Son muchas las empresas dedicadas a los servicios de monitorización con dispositivos precisos y robustos, pero a su vez son pocas las que abordan el problema de la estigmatización social que produce llevar estos dispositivos.

Por eso este proyecto tiene como objetivo principal, a parte de la funcionalidad que debe desempeñar, evitar este tipo de etiquetado del individuo, cuya consecuencia más inmediata es la estigmatización social.

La idea de realizar este proyecto surge de la propuesta de Trabajo de Fin de Grado ofertada por Mario Corrales y Alfonso Bahillo, en la que se propone realizar una app para smartwatch para monitorizar a personas vulnerables, en colaboración con el equipo de psiquiatría del complejo asistencial de Zamora.

Como ya se ha comentado, estos relojes inteligentes nos permiten tanto realizar la monitorización del individuo como detectar la retirada del dispositivo gracias a la gran cantidad de sensores que incluyen. Además, permite evitar ese estigma comentado debido a que es un complemento de uso cotidiano en la vida de las personas.

1.3. Estado de la cuestión

Como se ha comentado anteriormente, en la actualidad cada vez es más importante la monitorización de un individuo en distintos campos. Este sistema lleva funcionando varias décadas especialmente en los centros penitenciarios para vigilar las salidas en permiso de sus reclusos.

Estos dispositivos, además de ser capaces de realizar un seguimiento de la ubicación en tiempo real,

detectar distancias o ratios donde el individuo no puede acceder acompañado de señales acústicas y avisos a los centros de control si esto se incumple, también es importante que detecten cualquier manipulación o retirada del mismo.

Es por esta última causa por la cual la apariencia de estos dispositivos es muy singular, ya que necesitan una fuerte robustez en este aspecto. Son muchas las empresas que se han dedicado a este mercado, las cuales la gran mayoría no han mostrado un gran interés en mejorar esta apariencia pensando en la persona que lo portará.

Esta claro que se trata de un problema real en este campo. Cuando un dispositivo de esta índole se asigna a una persona, esta queda señalada al resto de la sociedad por una condición, por ejemplo de ex-presidiario en el ámbito de instituciones correccionales. El hecho de ser reconocido por el resto de la población y etiquetado de alguna forma produce un estigma en la persona que puede llegar a causar una exclusión social.

Tras la entrada masiva de estos dispositivos en el campo de la salud, especialmente tras la pandemia de COVID-19, es muy importante tratar de resolver este tema. Es por ello que, a continuación, se procede a analizar los dispositivos que hay actualmente en el mercado mostrando sus ventajas y desventajas especialmente en este último punto comentado. Cabe destacar que el análisis realizado a continuación se centra únicamente en los productos hardware en el mercado.

1.3.1. Track Group

Track Group [2] es una empresa líder en este sector fundada en 1995 en Estados Unidos. Esta empresa ofrece soluciones, tanto software como hardware, a la monitorización de la ubicación de personas y además se basa en el comportamiento predictivo, es decir, analiza los datos recogidos de manera estadística. Esta empresa ofrece dos soluciones hardware a este tema: ReliAlert XC4 y SecureCuff que se combinan para formar un único dispositivo. (Ver figuras 1.1 y 1.2)

Figura 1.1: Dispositivo ReliAlert-XC4 de Track Group [2].

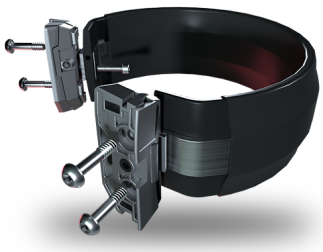

Figura 1.2: Dispositivo SecureCuff de Track Group [2].

ReliAlert-XC4 es un dispositivo GPS de una pieza que permite la comunicación con 4G. Entre las muchas funcionalidades de este dispositivo, destacan las siguientes:

- Actualización de ubicación cada 2 segundos.
- GPS de gran precisión.
- Tecnología de localización secundaria cuando el GPS no esta disponible.
- Duración de batería de 3 días con un intervalo de monitorización de 1 minuto y hasta 2 semanas en modo de suspensión.
- Almacenamiento en memoria interna durante un intervalo de tiempo de 18 días.

Por otro lado, SecureCuff se trata del complemento al dispositivo anterior. Es una correa de acero revestido que ofrece solución al corte de la misma a través de fibra óptica.

Como se observa la combinación de ambos proporciona una gran solución a este problema, pero encontramos la gran desventaja ya comentada. Como se observa en las imágenes anteriores, la peculiar forma de esta pulsera produce un estigma en todo aquel que la porta.

1.3.2. BI Incorporated

BI Incorporated [1] es otra de las grandes empresas líderes en este sector. Fundada en 1978 con oficinas centrales en los Estados Unidos, esta empresa se encarga de dar servicio de monitorización especialmente a las instituciones correccionales.

BI ofrece otras tres soluciones hardware para este campo. Un dispositivo basado en radiofrecuencia, BI HomeGuard 20|20 (Figura 1.3) y dos dispositivos GPS: BI LOC8 XT (Figura 1.4) y BI VeriWatch (Figura 1.5).

Figura 1.3: Dispositivo BI Homeguard 20|20 de BI Inc [1].

Figura 1.4: Dispositivo BI Loc8 XT de BI Inc [1].

Figura 1.5: Dispositivo BI VeriWatch de BI Inc [1].

Analizando los dispositivos, el BI Homeguard tiene grandes ventajas si el objetivo es monitorizar la presencia del individuo en un lugar específico gracias también a un apoyo GPS, permite la comunicación con el centro de control y tiene un robusto sistema de detección de manipulación del dispositivo. Como gran desventaja para nuestro sistema de estudio es su forma y la estigmatización ya comentada anteriormente.

Respecto a los dispositivos que monitorizan la ubicación, el BI Loc8 XT nos encontramos con una descripción similar al anterior. Ventajas en su funcionalidad como gran cobertura GPS y de red celular, un sistema robusto de detección de la retirada y una gran duración de la batería de hasta 60 horas. Pero sigue suponiendo el mismo problema debido a su forma.

Por último, se analiza el dispositivo BI VeriWatch. En este caso, la empresa ha tenido en cuenta este problema e intenta camuflar este dispositivo con el aspecto de un reloj inteligente. Las ventajas de este dispositivo respecto a la monitorización de la ubicación son similares, ya que se incorporan GPS con gran precisión y detectan la retirada del dispositivo a través de sensores. Pero por otra parte como principal desventaja tendríamos que la duración de la batería cae hasta las 16 horas, comparando con los otros dispositivos de la marca la diferencia es más que notable. Además de que se basa en el envío de notificaciones con el reconocimiento del cliente para aumentar esta robustez.

1.3.3. Unidad 2Track

La unidad 2Track [4] es un dispositivo de geolocalización por GPS con funcionalidades de comunicación de voz y datos. Se trata de un dispositivo incorporado por el gobierno de España para controlar las penas de alejamiento en el ámbito de la violencia de género. Este sistema de seguimiento consta de 3 dispositivos (Figura 1.6): 2 dispositivos 2Track, uno para la persona condenada (Figura 1.6 abajo) y otro para la víctima (Figura 1.6 arriba) y un transmisor de radiofrecuencia que portara solamente la persona condenada (Figura 1.6 centro).

El transmisor de radiofrecuencia emite una señal de radiofrecuencia que es recibida por la unidad 2Track y que contiene sensores que permiten controlar su manipulación. Además el dispositivo 2Track de la persona condenada también recibirá esta señal de radiofrecuencia con el fin de detectar que se esta portando el dispositivo.

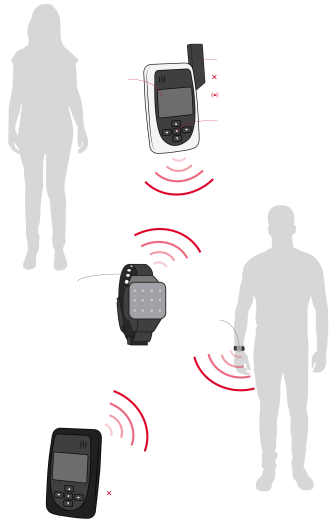

Figura 1.6: Unidad 2Track y dispositivo de radiofrecuencia [5].

Esta unidad supone una gran ventaja en la localización del individuo por la incorporación de la señal GPS además de robustez en la detección de la retirada del mismo. Pero seguimos encontrándonos los mismos estándares en su forma.

Estas son solo algunos de los dispositivos que encontramos actualmente en el mercado. La mayoría de ellos presentan el principal problema que se viene comentando en cuando al etiquetado que produce en la sociedad el portarlos debido a su forma. Es por ello que se debe buscar una solución a este problema, especialmente si se quiere ampliar los campos en los que se utilizan estos dispositivos, como por ejemplo en la sanidad.

1.4. Relojes inteligentes

Estos dispositivos cada vez se encuentran con mayor frecuencia en el mercado tecnológico, como se decide realizar este proyecto con un Smartwatch para sustituir a los dispositivos ya comentados, es necesario realizar un análisis de los distintos dispositivos con Wear OS disponibles en el mercado. Esta comparativa se basa en los sensores, precio y capacidad de batería, al tratarse estos puntos fundamentales para el desarrollo del proyecto se refleja en la siguiente tabla[6][7].

Nombre	Precio	Batería	Sensores
Samsung Galaxy Watch5 Pro (LTE)	519,99€	590mAh	Frecuencia cardiaca, GPS, Temperatura corporal, Acelerómetro, Barómetro, Giroscopio, Sensor Magnético, Luz ambiente.
Mobvoi TicWatch Pro 3 Ultra (Bluetooth)	299,99€	577mAh	Frecuencia cardiaca, GPS, Acelerómetro, Barómetro, Giroscopio, Pulsioxímetro, Brújula.
Samsung Galaxy Watch5 (LTE)	369€	410mAh	Frecuencia cardiaca, GPS, Temperatura corporal, Acelerómetro, Barómetro, Giroscopio, Sensor Magnético, Luz ambiente.
Samsung Galaxy Watch4 Classic 46mm (LTE)	369€	361 mAh	BioActive Sensor , Electrocardiograma, Sensor de análisis de impedancia bioeléctrica, Acelerómetro, Barómetro, Giroscopio, Geomagnético, Luz ambiente.
Oppo Watch 41 mm (Bluetooth)	299€	300 mAh	Frecuencia cardiaca, GPS, Acelerómetro, Barómetro, Giroscopio, Luz ambiente.

Tabla 1.1: Tabla comparativa relojes inteligentes.

Se ha realizado un análisis exhaustivo comparando los mejores relojes inteligentes en el mercado en relación calidad-precio y prestando especial atención a sus características. En la tabla 1.1 se han escogido los 3 mejores relojes con los requisitos mencionados y se incluye a la comparativa tanto el Samsung Galaxy Watch4 Classic 46mm como el Oppo Watch 41mm que se trata de los dispositivos disponibles para realizar las pruebas en este proyecto.

1.5. Entorno de desarrollo y herramientas utilizadas

En esta sección se presentan las tecnologías utilizadas para el desarrollo del proyecto y una descripción de las mismas. Como el proyecto parte desde cero sin ninguna base de otros proyectos, se ha podido elegir las tecnologías que se han considerado oportunas teniendo en cuenta los requisitos y limitaciones del proyecto y la experiencia del programador.

Como resumen general de la sección, se ha utilizado como entorno de desarrollo integrado Android Studio al tratarse del oficial de Android para el desarrollo de aplicaciones, usando como lenguaje Java. Para la implementación de la base de datos local en el dispositivo se ha utilizado la librería de bases de datos de Google, Room. Además se han utilizado otras herramientas de apoyo que se comentarán a continuación.

1.5.1. Android

Android [8][9] es un sistema operativo basado en un núcleo Linux y diseñado especialmente para dispositivos móviles con pantalla táctil como teléfonos inteligentes, tablets o relojes inteligentes entre otros. Fue desarrollado por Android Inc. que pertenece a Google desde el año 2005 y dos años más tarde fue presentado junto con la fundación *Open Handset Alliance*[10], alianza comercial formada por 84 compañías dedicadas al desarrollo de estándares para este tipo de dispositivos.

Figura 1.7: Logo oficial de Android [9].

Android presenta su estructura como una pila de software de código abierto, es decir, divida en capas como podemos ver en la Figura 1.8, tomando como base el kernel Linux[11]. Las capas en las que se divide la arquitectura Linux son las siguientes:

- *Kernel Linux*: Es la base de Android. Se apoya en él para utilizar los servicios base del sistema como la gestión de memoria, creación de subprocesos o temas de seguridad.
- *Capa de Abstracción de Hardware*: También conocida como capa HAL, se encuentra justo encima de la capa Kernel y sirve de enlace para brindar las capacidades hardware del sistema como Bluetooth o sonido al marco de trabajo de la API de Java.

- *Android Runtime*: En esta capa podemos encontrar las diferentes máquinas virtuales donde cada aplicación ejecuta sus procesos.
- *Bibliotecas*: Contiene una serie de bibliotecas nativas escritas en C y C++ que dan funcionalidad a las distintas apps que se ejecutan.
- *Marco de trabajo de la API de Java*: Aquí encontramos todo el conjunto de funcionalidades de Android que los desarrolladores utilizan gracias a APIs en Java.
- *Aplicaciones del sistema*: En esta última capa se encuentran instaladas las aplicaciones por defecto del sistema como puede ser la mensajería SMS, Cámara o Contactos entre otras.

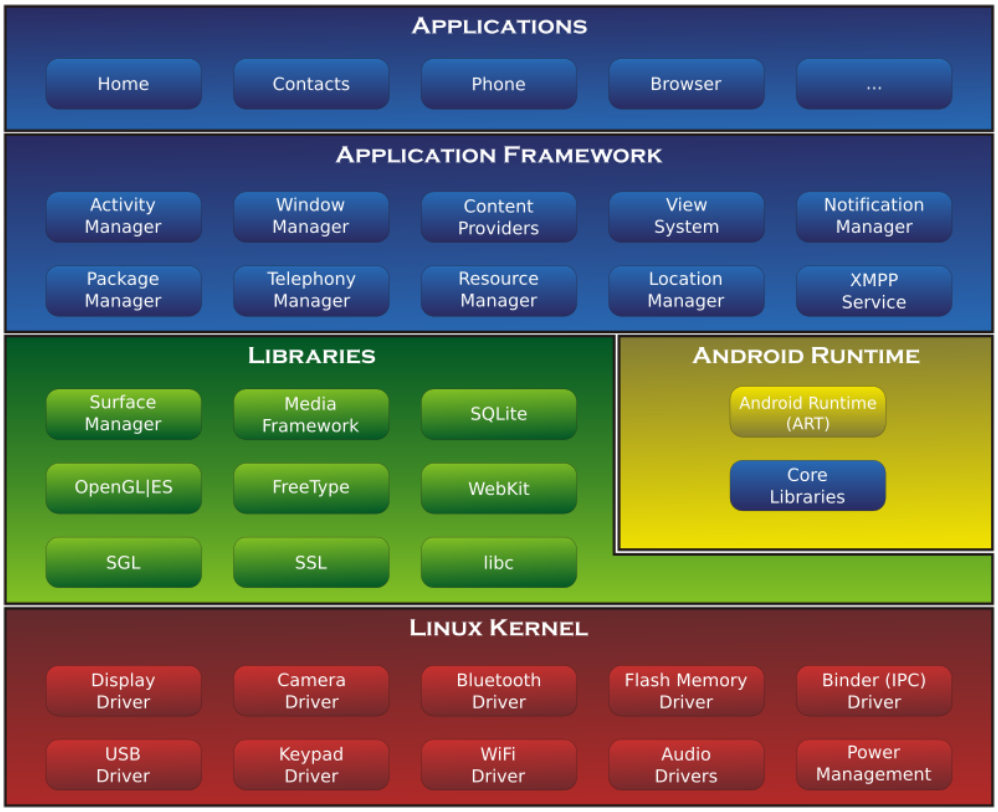

Figura 1.8: Esquema arquitectura Android [9].

1.5.2. Wear OS

Wear OS[12][13] es una versión de Android diseñada para relojes inteligentes y otros portables. Se basa en el emparejamiento con teléfonos móviles (con versiones superiores a Android 6 o la versión 10 de iOS).

Figura 1.9: Logo oficial de Wear OS [13].

La plataforma se anunció en 2014 y a día de hoy es compatible con todo tipo de conectividad (Bluetooth, NFC, WiFi y LTE). Grandes marcas son socios de este en el tema hardware como Asus, Intel, LG o Samsung.

En cuanto a su arquitectura, Wear OS está basado en Android por lo que esta es similar pero con pequeños matices. Esta pensado para desplegarse en dispositivos wearables como smartwatches, los cuales tienen una menor cantidad de recursos hardware que los dispositivos móviles.

Un ejemplo de ello para entender mejor a que nos referimos es que estos dispositivos tienen pantallas menores que los teléfonos móviles o carecen de puertos de entrada, esto hace que el desarrollo software difiera respecto a Android en un teléfono. Sin embargo, como este sistema esta pensado para funcionar gracias a un emparejamiento con un teléfono móvil, esto hace que no pierda sus funcionalidades principal aún careciendo de diversos componentes.

1.5.3. Android Studio

Android Studio [14] es el entorno de desarrollo integrado oficial de Android para el desarrollo de sus aplicaciones. Esta basado en Intellij IDEA [15], un editor de código que contiene una gran cantidad de herramientas para desarrolladores de todos los campos.

Figura 1.10: Logo oficial de Android Studio [14].

Entre las principales características de este IDE se pueden destacar el uso de Gradle como sistema de compilación, es un sistema integrado con multitud de funciones de configuración y que funciona

independientemente de la linea de comandos.

También comentar su emulador que permite adaptarlo a multitud de dispositivos y su integración con Git para el control de versiones entre las más destacadas.

1.5.4. Java

Java [16][17] es un lenguaje de programación con tipado estático, multiplataforma y orientado a objetos.
Las aplicaciones Java se pueden compilar en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora en la que compila.

Figura 1.11: Logo oficial de Java.

Java es uno de los lenguajes de programación más utilizados por la mayoría de empresas. Pertenece a *Oracle* desde el año 2010 cuando efectuó su compra a *Sun Microsystems*. Entre sus principales características que lo hacen tan popular encontramos [18]:

- Lenguaje orientado a objetos.
- Multiplataforma, se puede ejecutar en cualquier sistema.
- Características de seguridad integradas.
- Manejo automático de memoria, libera memoria de objetos que ya no se utilizan.
- Multithreading, permite la ejecución de varios hilos en paralelo.
- Gran cantidad de APIs integradas.

1.5.5. Room

Room [19][20] es una biblioteca que proporciona una capa de abstracción sobre SQLite y facilita la configuración de la base de datos y los accesos a ella, devolviendo objetos ya formados.

La arquitectura de Room se basa en clases con anotaciones. Esta arquitectura esta formada por tres partes: *Entity*, clases que definen las tablas de la base de datos, *DAO*, las interfaces que definen los métodos que se utilizan, y *RoomDatabase*, que es la clase enlace con la base de datos SQLite a través de los DAOs. Podemos observar en la figura 1.12 esta arquitectura.

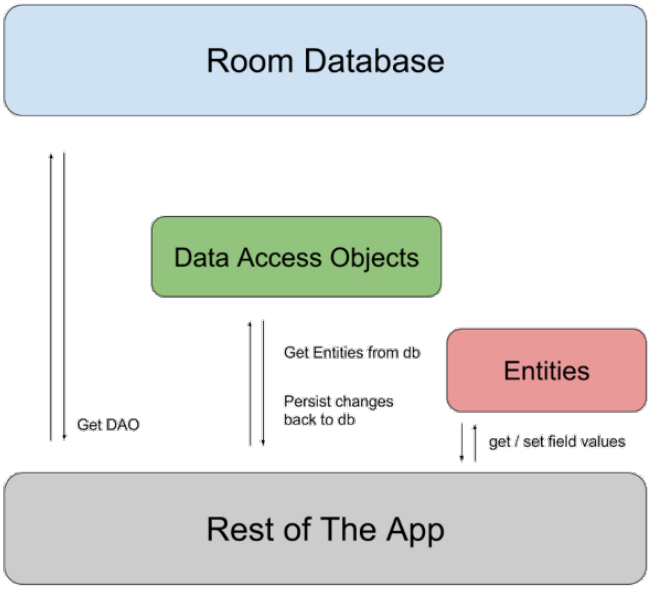

Figura 1.12: Arquitectura de Room en una app [21].

1.5.6. Advanced Rest Client

Advanced Rest Client [22] es una herramienta que actúa como cliente y nos permite realizar peticiones HTTP a APIs de un servidor con el fin de realizar pruebas. En la figura 1.13 podemos observar como es el funcionamiento de esta herramienta.

En la parte superior se elige la petición HTTP que se desea realizar y la dirección URL del servidor. El panel izquierdo almacena las peticiones que hemos realizado anteriormente. Por último, en el panel central obtenemos los campos respuesta de la petición y el cuerpo que podemos completar si es necesario.

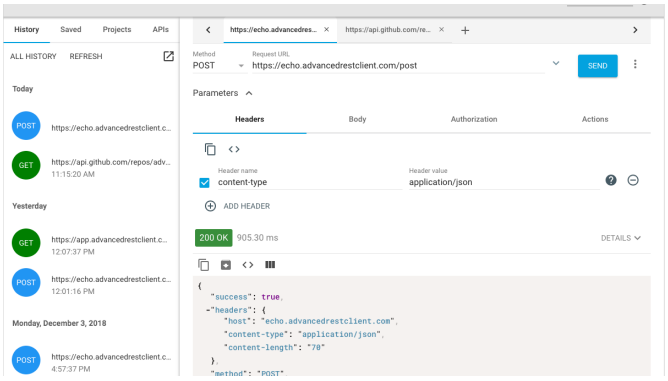

Figura 1.13: Captura de Advanced Rest Client [22].

1.6. Objetivos

El objetivo general de este proyecto es realizar una app android para smartwatches con sistema operativo Wear OS que permita la monitorización de la ubicación del portador, así como una solución en caso de emergencia y el control sobre la retirada del dispositivo de la muñeca del usuario. Este objetivo se puede dividir en varios:

- Monitorización de la ubicación con su posterior envío a un servidor.
- Detección de retirada del dispositivo a través de los distintos sensores.
- Desarrollo de un sistema de accionamiento en caso de emergencia que establezca un canal de comunicación por voz bidireccional con el centro asociado.

1.7. Metodología

Tras realizar un análisis tanto de requisitos como de características deseadas para el proyecto se ha decidido recurrir a una metodología iterativa e incremental. Las principales causas para elegir esta metodología son el desarrollo del proyecto, realizado por un único programador, y la naturaleza del proyecto.

Esta metodología nos permite tener una aplicación con funcionalidad tras cada iteración, lo que nos da la posibilidad de revisar el cumplimiento de las especificaciones con el cliente y de añadir nuevos requerimientos si fueran necesarios. Con todo esto se reducen riesgos que produzcan un retroceso cuando el proyecto esta avanzado.

Se utilizarán iteraciones cortas de 3 a 5 semanas para cumplir con el objetivo correspondiente. En la figura 1.14 se muestra como es un proceso iterativo e incremental aunque se explicará como se ha realizado esta metodología específicamente para este proyecto en el Capitulo 3.

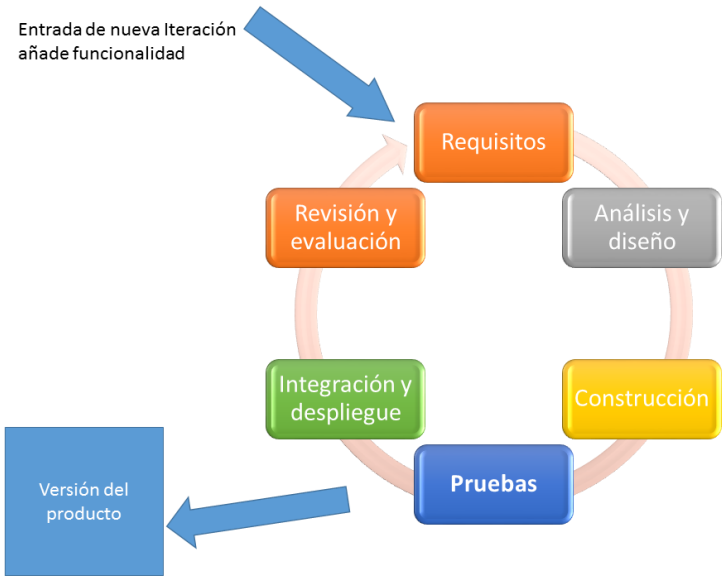

Figura 1.14: Esquema Metodología iterativa e incremental [23].

1.8. Estructura de la memoria

- **Introducción**: Se presenta el proyecto de manera global, con el contexto y motivaciones para llevarlo a cabo, el estado de la cuestión, los objetivos a alcanzar en la realización del proyecto y la metodología usada para el desarrollo del mismo.
- **Planificación**: Se presenta la organización temporal con la estimación del tiempo para cada iteración del proyecto, sus riesgos, costes y el entorno tecnológico en el que se ha desarrollado. Por último, se incluye como ha sido finalmente la planificación del proyecto y una comparación con la planificación estimada.
- **Descripción de las iteraciones**: Se explica el desarrollo del proyecto en cada iteración.
- **Implementación**: Se detalla como han sido implementadas las distintas funcionalidades del sistema en este proyecto.
- **Estado final de la aplicación**: Se describe el análisis y el diseño del proyecto, además de su estructuración. Se incluyen los diagramas correspondientes.
- **Pruebas**: Se incluye la descripción y resultado de las pruebas realizadas a la aplicación.
- **Conclusiones**: Se exponen las conclusiones obtenidas tras la realización del proyecto.
- **Apéndices**: Documentación adicional al proyecto como acrónimos, manuales, etc.
- **Bibliografía**: Referencias bibliográficas utilizadas en el proyecto.

Capítulo 2

Planificación

2.1. Planificación inicial

2.1.1. Organización temporal

Este trabajo se realizará durante un periodo comprendido entre el 30 de enero de 2023 y el 30 de mayo de 2023, correspondiendo con el periodo lectivo del Segundo Cuatrimestre. Corresponderá con un periodo de aproximadamente 5 meses.

Para ajustar con lo expuesto en la guía docente del Trabajo de Fin de Grado donde se atribuyen 300 horas, se ha estimado un trabajo de 5 días semanales donde se dediquen unas 4 horas diarias. Además, se pretende ampliar este trabajo a los fines de semana, lo que permitirá emplearse por si surgiera algún impedimento. Con esta estimación se obtiene un total de 348 horas totales.

Se llevará a cabo un trabajo previo al inicio del proyecto durante el periodo comprendido entre el 22 de diciembre de 2022 y el 7 de febrero de 2023. En esta fase se tratará de abordar la planificación inicial, investigación y análisis sobre proyectos ya existentes con la misma finalidad, comparativa entre los distintos productos del mercado donde se utilizará el proyecto y la adaptación al entorno de desarrollo para aplicaciones en Wear OS.

Para el desarrollo del proyecto se utilizará una metodología iterativa incremental, en el que se realizarán las siguientes iteraciones en función del avance del proyecto:

- 1ª iteración: Trataremos esta fase como punto de inicio del proyecto, que comenzará el 7 de febrero de 2023 y se estima una duración de 5 semanas. Se tratará de recopilar toda la información obtenida en la fase anterior y será estructurada en el formato preciso.

  Durante esta iteración se desarrollará la funcionalidad correspondiente al envío de la ubicación GPS en tiempo real.

- 2ª iteración: Comenzará el 14 de marzo de 2023 y tendrá una duración estimada de 4 semanas.

Se llevará a cabo el análisis y posterior desarrollo de la funcionalidad de detección de retirada del dispositivo.

- 3ª iteración: Comenzará el 11 de abril de 2023 y tendrá una duración estimada de 4 semanas. Desarrollo de la funcionalidad del botón de emergencia y envío de audio en tiempo real tras ser accionado.
- 4ª iteración: En esta última fase se realizarán las pruebas finales y se completará el trabajo restante con el fin de preparar la entrega del proyecto. Comenzará el 9 de mayo de 2023 y tendrá una duración estimada de 3 semanas. Esta fase no llevará a cabo mucho tiempo ya que lo habremos dedicado todo en las fases anteriores y el proyecto debe llegar prácticamente finalizado.

En la figura 2.1 se muestra la gráfica de distribución por semanas en cada iteración, incluyendo además las semanas acumuladas. Como las iteraciones se han estimado con un contenido similar en cuanto a desarrollo, se puede apreciar que la duración es similar.

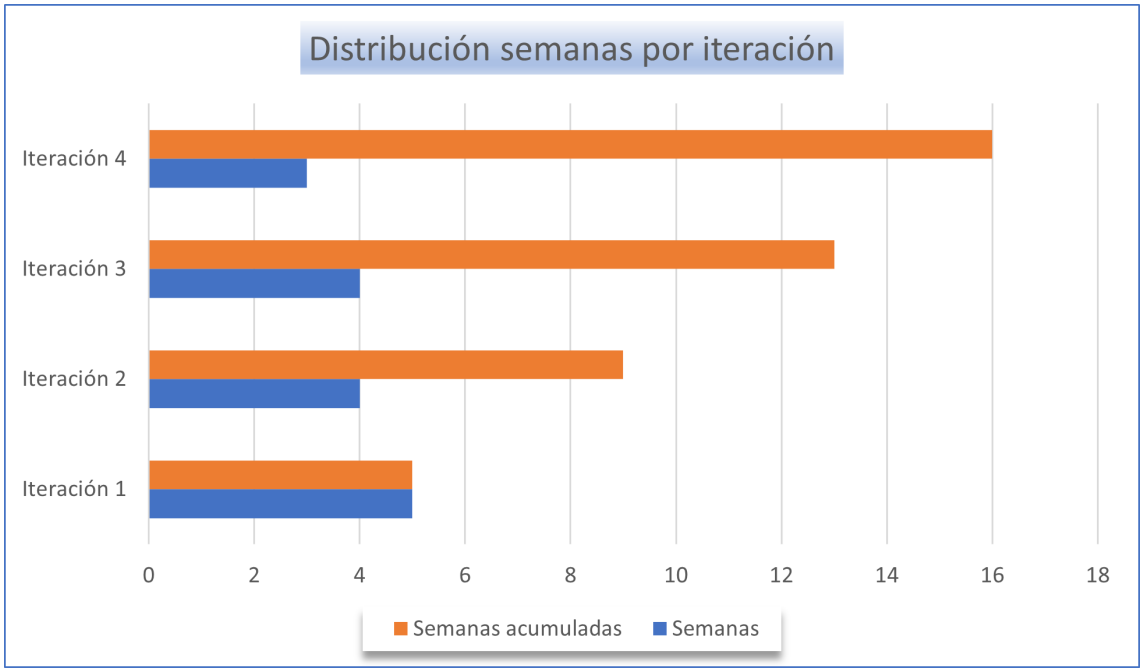

Figura 2.1: Distribución semanas por iteración

En la figura 2.2 se muestra la gráfica con la distribución de horas por cada iteración, incluyendo también las horas acumuladas en cada iteración.

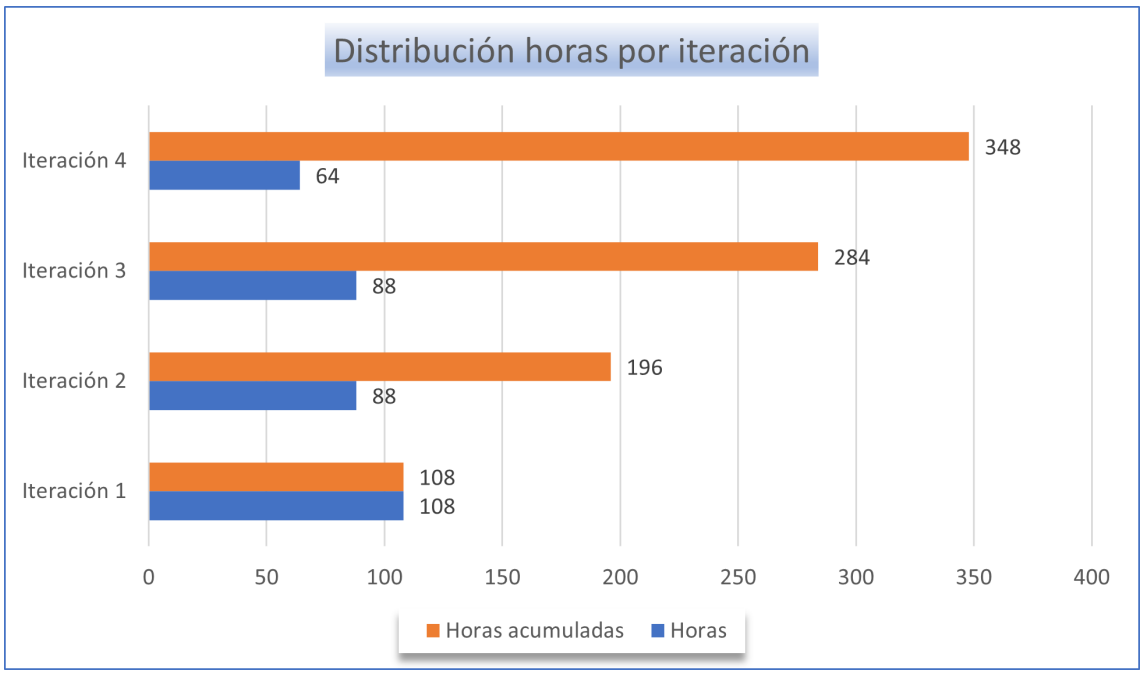

Figura 2.2: Distribución horas por iteración

2.1.2. Gestión de riesgos

En esta sección se tratará de evaluar una serie de riesgos que pueden ocurrir durante el desarrollo del proyecto y que pueden suponer un retraso en la entrega del mismo. Los riesgos se presentaran en una tabla cuyos apartados serán los siguientes:

- ID del riesgo con el que se identifica.
- Nombre del riesgo identificado.
- Descripción del riesgo identificado.
- Probabilidad de que el riesgo se manifieste.
- Retraso estimado que se puede dar, es decir el impacto del riesgo en el proyecto.
- Exposición al riesgo, se calcula con la multiplicación de la probabilidad por el impacto del riesgo.
- Medida de mitigación del riesgo, en caso de que este se manifieste.
- Medida de reducción del riesgo, con el fin de evitar el que el riesgo ocurra.

IDR	Riesgo	Descripción	Probabilidad	Retraso Estimado	Exposición al riesgo
1	Enfermedad.	El alumno sufre una enfermedad grave.	0.1	15	1.5
2	Compatibilidad con asignaturas.	El trabajo se realiza de manera conjunta con otras dos asignaturas, que pueden quitar tiempo para el desarrollo de este.	0.3	6	1.8
3	Baja de tutor/es asignado/s.	El tutor académico obtiene una baja durante el periodo del trabajo.	0.1	7	0.7
4	Cambio en los requisitos.	Se realizan cambios en los requisitos cuando el proyecto ya está avanzado.	0.2	15	3
5	Falta de experiencia con la programación en WearOS.	Al tratarse de un sistema operativo que no se ha estudiado en la carrera, puede no tenerse demasiada experiencia.	0.2	15	3
6	Falta de material necesario.	Falta de reloj smartwatch para pruebas específicas, como realización de llamadas	0.1	10	1
7	Problemas con los equipos de trabajo.	Fallos en el ordenador o smartwatch que se usan para realizar el trabajo.	0.1	6	0.6
8	No finalizar las asignaturas en la convocatoria correspondiente.	Se suspende alguna asignatura en la convocatoria ordinaria o extraordinaria. Lo que supone un retraso en la entrega del TFG.	0.1	150	15

Tabla 2.1: Tabla gestión de riesgos.

En la siguiente tabla se incluyen los riesgos junto con sus correspondientes acciones de mitigación y reducción.

IDR	Riesgo	Descripción	Acción de mitigación	Acción de reducción
1	Enfermedad.	El alumno sufre una enfermedad grave.	Retraso en la fecha final de entrega.	Mantener más precauciones durante este periodo para no enfermar.
2	Compatibilidad con asignaturas.	El trabajo se realiza de manera conjunta con otras dos asignaturas, que pueden quitar tiempo para el desarrollo de este.	Aumentar la cantidad de horas diarias dedicadas al proyecto.	Realizar el trabajo diario de las asignaturas para que no suponga un mayor retraso.
3	Baja de tutor/es asignado/s.	El tutor académico obtiene una baja durante el periodo del trabajo.	Replanificar el TFG con un nuevo tutor académico.	Especificar medios de comunicación telemáticos al inicio del proyecto.
4	Cambio en los requisitos.	Se realizan cambios en los requisitos cuando el proyecto ya está avanzado.	Realizar una reunión con el cliente para adaptar el proyecto a la modificación.	Realizar reuniones periódicas de revisión con el cliente.
5	Falta de experiencia con la programación en WearOS.	Al tratarse de un sistema operativo que no se ha estudiado en la carrera, puede no tenerse demasiada experiencia.	Aumentar la cantidad de horas diarias que se dedican al proyecto y dedicarlas exclusivamente al aprendizaje de WearOS.	Realizar una fase previa al inicio del TFG donde se revise la documentación y se aprenda sobre el nuevo SO.
6	Falta de material necesario.	Falta de reloj smartwatch para pruebas específicas, como realización de llamadas	Realizar las pruebas con un emulador correspondiente.	Planificar correctamente el material necesario al inicio del TFG.
7	Problemas con los equipos de trabajo.	Fallos en el ordenador o smartwatch que se usan para realizar el trabajo.	Uso de otro equipo disponible durante los días que los equipos fallen.	Tener varios equipos disponibles y almacenar toda la información en la nube para evitar pérdidas de datos.
8	No finalizar las asignaturas en la convocatoria correspondiente.	Se suspende alguna asignatura en la convocatoria ordinaria o extraordinaria. Lo que supone un retraso en la entrega del TFG.	Seguir trabajando en el proyecto para que esté listo en cuanto sea posible entregarlo.	Dedicar el tiempo necesario a las asignaturas para poder aprobarse en la convocatoria adecuada.

Tabla 2.2: Tabla gestión de riesgos. Acciones reducción y mitigación.

2.1.3. Entorno tecnológico

2.1.3.1. Herramientas utilizadas

A continuación se procede a describir las herramientas utilizadas durante el desarrollo del proyecto:

- Astah Professional [24]: Herramienta de diseño utilizada para realizar diagramas UML.
- Overleaf [25]: Herramienta de redacción en linea utilizada para la realización de la memoria del proyecto en LaTex.
- Gitlab [26]: Herramienta para el control de versiones y almacenamiento del código desarrollado en la nube.
- Android Studio [14]: Entorno de desarrollo de Android para el desarrollo de la app de smartwatch basado en Wear OS.
- Microsoft Teams [27]: Canal de comunicación utilizado con los tutores del proyecto.

2.1.3.2. Equipos de desarrollo

Equipo	Acer Aspire 5 a515-51G-88W4
Sistema Operativo	Windows 10
Arquitectura	64 bits
Procesador	Intel Core i7-8550U
Disco duro	256 GB SSD
Memoria RAM	8 GB DDR4

Tabla 2.3: Tabla características PC.

Equipo	OPPO Watch 41mm (WI-FI)
Sistema Operativo	WearOS by Google
Pantalla	AMOLED 1,91"
Procesador	Snapdragon Wear 3100 de Qualcomm
Disco duro	8 GB
Conectividad	WiFI, Bluetooth 4.2, NFC
Sensores	Giroscopio, Frecuencia cardíaca, Altímetro, GPS con GLONASS Acelerómetro de 3 ejes, Óptico de latido, de capacitancia de luz ambiental y de control del sueño.
Memoria RAM	1 GB

Tabla 2.4: Tabla características Smartwatch Oppo.

Equipo	Samsung Galaxy Watch4 Classic 4G
Sistema Operativo	WearOS by Google
Pantalla	AMOLED 1,4"
Procesador	Samsung Exynos W920
Disco duro	16 GB
Conectividad	WiFI, Bluetooth 4.2, NFC, LTE
Sensores	Frecuencia cardíaca, GPS, Acelerómetro de 3 ejes, Óptico de latido, impedancia de luz ambiental, barómetro, giroscopio y magnético.
Memoria RAM	1.5 GB

Tabla 2.5: Tabla características Smartwatch Samsung Galaxy Watch4.

2.1.4. Estimación de costes

En este apartado se procede a desarrollar una estimación de costes del proyecto a realizar. Se tratará de estimar tanto costes humanos como costes informáticos y realizar una planificación en función de estos costes.

2.1.4.1. Coste respecto al salario del desarrollador

Teniendo en cuenta que el trabajo no será realizado en convenio con ninguna empresa y se trata de un proyecto de fin de grado para la Universidad de Valladolid, el trabajo no será remunerado al estudiante. Por ello, para realizar una estimación precisa de los costes, se tendrá en cuenta el salario medio de un desarrollador Android Junior que se encuentra en una media anual de 20.500€ [28]. Con esta estimación y teniendo en cuenta una jornada laboral de 40 horas semanales el salario bruto obtenido son 10,68€/hora. Finalmente, como el tiempo establecido para este proyecto son 300 horas, el coste del salario del desarrollador en este proyecto supone 3204€ brutos.

2.1.4.2. Coste respecto al salario de los supervisores

Se incluye una estimación de las horas de asistencia de los supervisores del proyecto, en este caso los tutores. Se estima con un coste de 25 euros por hora. Para el cálculo de horas se estima 1 hora por cada reunión realizada, siendo un total de 5 reuniones, más 2 horas en las revisiones de la memoria. Al tratarse de 2 tutores el coste total sería de 350€.

2.1.4.3. Coste respecto a los equipos utilizados

Los costes estimados de los equipos que se utilizarán durante este proyecto serán los siguientes:

- Acer Aspire 5 a515-51G-88W4: 599€

- Smartwatch OPPO Watch 41mm (WI-FI): 289,99€
- Galaxy Watch4 Classic 4G: 369€

2.1.4.4. Coste respecto al espacio de trabajo

El trabajo será realizado en la vivienda del estudiante, por esto no se tendrá en cuenta un gasto en alquiler pero si los gastos variables (luz, gas...). El gasto total se calculará en función de las 300 horas que se dedicarán al proyecto y el reparto diario del mismo.

Para el gasto de luz se estima que un ordenador consume 0,05kWh [29] cada hora en una jornada laboral de 8 horas, por hora supone un consumo de 0,4kWh, en el trabajo dedicado de 300 horas tendremos un consumo total de 15kwh, suponiendo un precio medio de 0,307€/kWh, el gasto de un ordenador durante el proyecto supondría 4,605€.

En la siguiente tabla se muestra los otros dos gastos variables que se tienen en cuenta, Gas e Internet. El apartado *Gasto Total* se refiere al gasto de dicho servicio en la duración total del proyecto.

Gasto	Cuota Mensual	Precio por hora	Gasto total
Gas	90	0.121	36,29
Internet	45	0.0625	18,75

Tabla 2.6: Gastos

2.1.4.5. Coste total estimado del proyecto

Con todos los costes desglosados anteriormente, se muestra a continuación una tabla que recoge el resumen de todos los gastos para así calcular el coste total del proyecto. Además se incluye una columna del gasto amortizado de los equipos hardware en relación con el uso para este proyecto. Esta columna se ha estimado usando los años de vida medios del dispositivo y el tiempo utilizado en este proyecto.

Servicio	Costo (€)	Coste Amortizado (€)
Salario desarrollador	3204	3204
Salario supervisores	350	350
Equipos utilizados	1338.89	119.64
Espacio de trabajo	59.62	59.62
Coste total	4952.51	3733.26

Tabla 2.7: Coste total proyecto.

2.2. Planificación final

A continuación se realizará una comparación entre la estimación realizada en la planificación inicial con los datos obtenidos tras la realización del proyecto.

En la planificación inicial se estimó una duración del proyecto de aproximadamente 17 semanas, con el fin de ser entregado durante el periodo lectivo del segundo cuatrimestre. Debido a la dedicación de parte del tiempo a otras asignaturas del grado y la demora a la hora de realizar algunas partes del código que se ha tardado más de lo esperado por las dificultades que han ido surgiendo, la entrega prevista se retrasó 6 semanas aproximadamente. En lugar del 30 de mayo se decide entregar, debido a varios retrasos en algunas iteraciones, en fecha de convocatoria extraordinaria el 11 de julio. Respecto a las horas dedicadas al proyecto se estimó un total de 348 horas que finalmente se ha extendido a 514 horas.

En la primera iteración en la cual se realiza la funcionalidad relacionada con la monitorización de la ubicación las dificultades obtenidas se refieren principalmente al desconocimiento de las APIs a utilizar de las cuales se debe estudiar cada funcionalidad.

Además surge la necesidad de la creación de la base de datos, el análisis de las posibilidades para desarrollar la más adecuada también supone una demora respecto a la estimación. Esta iteración se desarrolla con una semana de retraso, que supone unas 24 horas.

En la segunda iteración se debe analizar los distintos sensores del dispositivo y elaborar un algoritmo que permita detectar la retirada de la muñeca del portador con estas mediciones. La idea principal consistía en realizar esta comprobación gracias al sensor de frecuencia cardiaca pero los valores medidos no han sido los esperados, por ello, tras realizar las pruebas correspondientes, no se ha podido utilizar.

Esta iteración se demora más de lo esperado ya que es necesario realizar estas pruebas con cada uno de los sensores para elaborar un algoritmo robusto. En concreto se demora 2 semanas respecto a la estimación, lo que supone 50 horas.

La tercera iteración se dedica a la funcionalidad del botón de emergencia y la correspondiente llamada a realizar, además al encontrase disponible el servidor también se desarrolla la parte correspondiente de envío a este. Estas funcionalidades resultan sencillas de desarrollar y probar, bastante más de lo esperado, y es por ello que se acaba antes de lo estimado. Se decide entonces desarrollar el funcionamiento de la aplicación en segundo plano en vez de avanzar a la siguiente iteración. Se finaliza en las 4 semanas estimadas, aunque la estimación en horas se demora en 10 más.

La última iteración se reserva para pruebas y la redacción de la memoria del trabajo. Aunque la realización de pruebas a un proyecto software necesita de mucho más tiempo, se decide únicamente realizar pruebas de aceptación, que además nos permiten redactar la memoria en paralelo.

La estimación se demora al tiempo estimado en semanas ya que se dedican 6 semanas, este retraso se debe principalmente a la redacción de la memoria y sus respectivas revisiones, se dedicarán varias

horas extra, 82 en concreto.

En la figura 2.3 se muestra la gráfica de distribución por semanas en cada iteración, realizando una comparativa entre la estimación y las semanas reales.

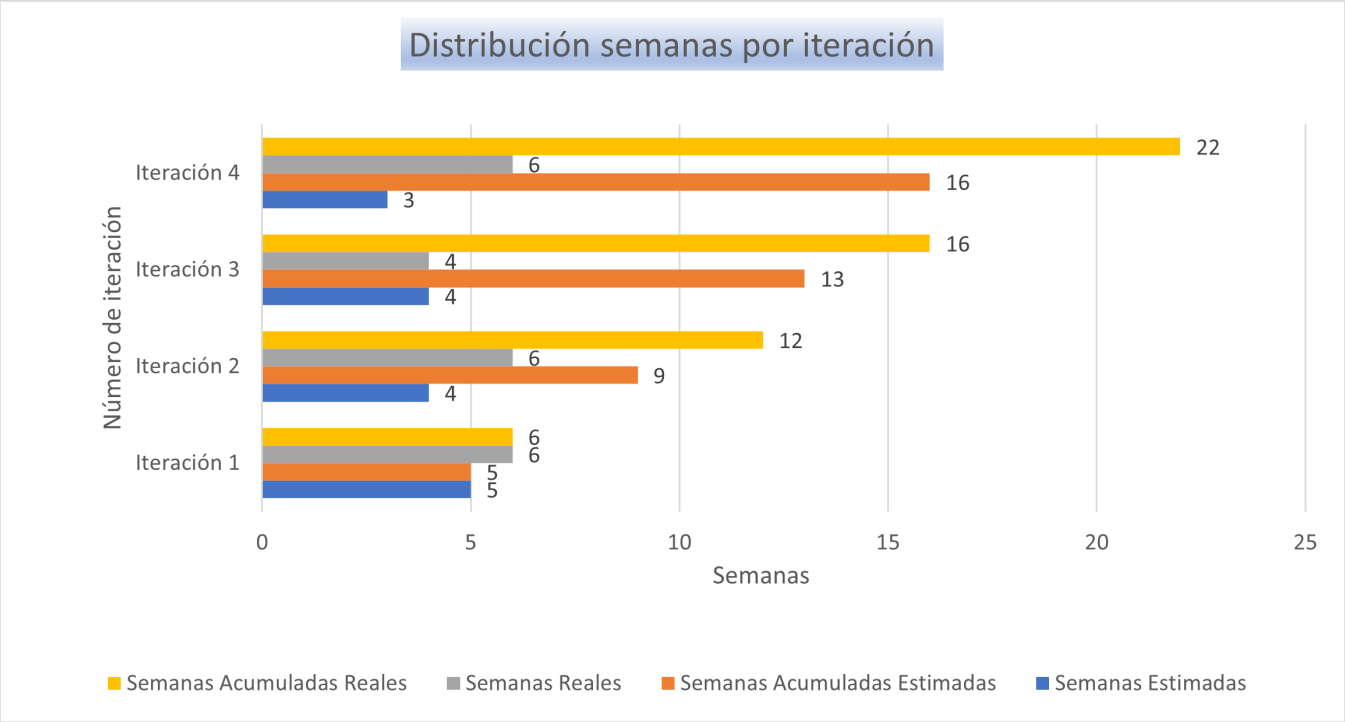

Figura 2.3: Distribución semanas por iteración.

En la figura 2.4 se muestra la gráfica de distribución por horas en cada iteración, realizando una comparativa entre la estimación y las horas reales.

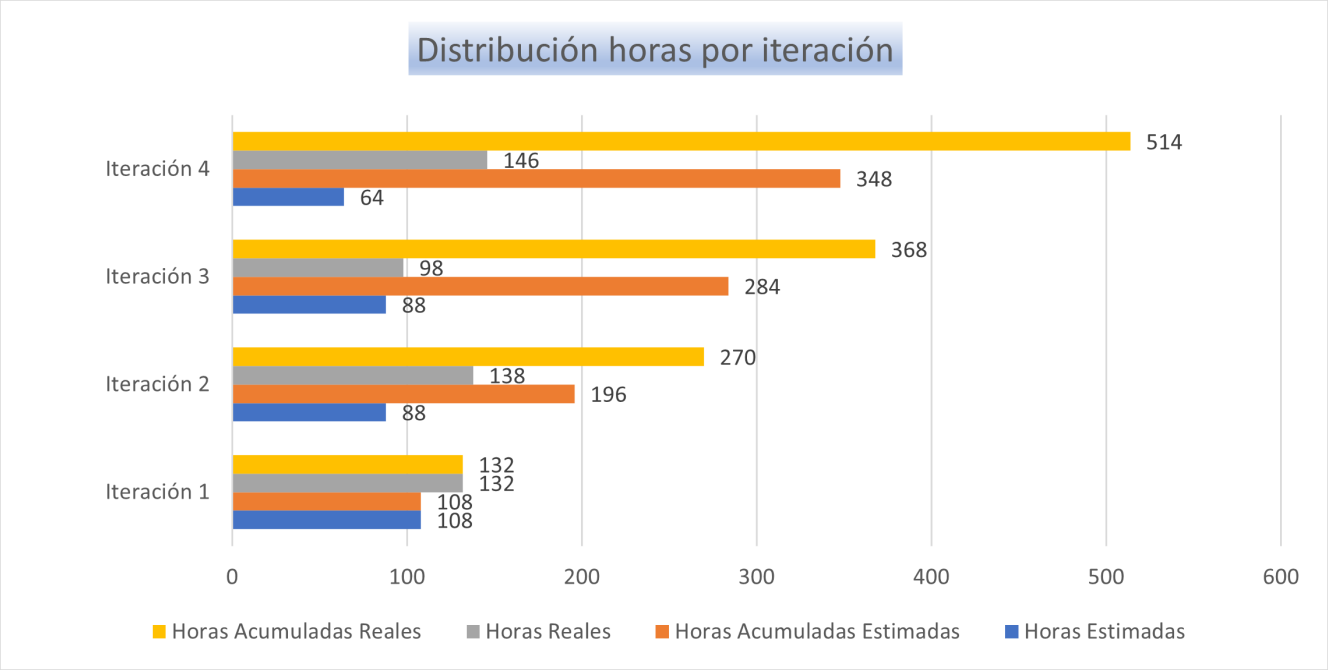

Figura 2.4: Distribución horas por iteración.

Capítulo 3

Descripción de las iteraciones

En este capítulo se desarrollará la explicación de cada una de las iteraciones llevadas a cabo para completar este proyecto. Se comentarán desde el análisis de requisitos hasta la finalización del proyecto, comentando así los avances y contratiempos que han surgido.

3.1. Iteración 1

Esta iteración es la primera en el propio desarrollo del proyecto. Se corresponde con el desarrollo de la funcionalidad de recogida de la ubicación del sensor GPS del dispositivo y posterior envío al servidor.

En primer lugar se realiza un estudio de las distintas herramientas disponibles para el desarrollo. Debido a la experiencia previa se elige como IDE Android Studio, ya que se trata del IDE oficial de Android. En la elección del lenguaje de programación en el cual se desarrollará el proyecto se debate entre Java y Kotlin, al tratarse de lenguajes similares, y Kotlin estar basado en Java pero con pequeños matices que lo diferencian, se decide usar Java debido a la experiencia y con el fin de no retrasar el avance del proyecto en el aprendizaje de nuevas tecnologías.

Se desarrolla la funcionalidad de la monitorización del dispositivo utilizando distintas APIs que nos proporcionan este servicio del sistema, esta ubicación se envía a un servidor. Además, se desarrolla una base de datos local en el dispositivo para almacenar estas ubicaciones en el caso de que no sea posible realizar el envío al servidor.

Al finalizar esta iteración, se tiene una aplicación funcional que monitoriza, envía y almacena la ubicación. Esta iteración se ha realizado en 6 semanas, se demora una semana más respecto a la estimación debido a la necesidad de la creación de la base de datos.

Tras el final de esta fase, se concreta una reunión con los tutores con el fin de revisar los avances realizados en el proyecto y definir el alcance de la funcionalidad que se desarrollará en la siguiente iteración.

3.2. Iteración 2

En esta iteración se propone realizar la funcionalidad correspondiente a la detección de retirada del dispositivo. Esta retirada se va a detectar gracias a los distintos sensores que se encuentran disponibles en estos dispositivos.

Tras realizar un análisis exhaustivo de los distintos sensores comunes en la gran mayoría de relojes inteligentes con Wear OS y que pueden servir en esta funcionalidad se encuentran los siguientes:

- Sensor de frecuencia cardiaca.
- Sensor de proximidad.
- GPS.
- Acelerómetro.
- Sensor de luz ambiente.

Desde un primer momento se busca una fuerte robustez en este aspecto por lo que la idea principal es basarse en comprobaciones de varios sensores en secuencia para asegurarse correctamente de que la retirada es cierta.

El sensor de proximidad es el que más fiabilidad nos aporta, además de que es el usado por la mayoría de las apps incluidas en estos dispositivos que detectan una retirada. Es cierto que este sensor se encuentra disponible en una amplia mayoría de dispositivos que circulan en el mercado, pero por otro lado, esta serie de dispositivos suele ser de una gama mayor y por lo tanto un precio más elevado.

Teniendo esto en cuenta, y que ambos dispositivos que se tienen disponibles para realizar las pruebas de este proyecto no disponen del sensor mencionado, se busca otra solución.

Pensando en un mayor alcance al que puede llegar este proyecto, se implementa dicha parte con una comprobación al iniciar la app en el caso de que el sensor de proximidad este incorporado en el mismo, será la primera comprobación que se estudiará para detectar una retirada.

En segundo lugar se estudia el acelerómetro, tratándose también de un sensor muy preciso y sensible a movimientos mínimos, proporciona una gran fiabilidad en esta función.

El acelerómetro[30] es un sensor capaz de medir la aceleración lineal y angular del dispositivo. En los acelerómetros actuales, esta fuerza de aceleración se mide en tres planos (X, Y y Z) lo que permite medir todo tipo de inclinaciones, vibraciones o cualquier movimiento basándose en las fuerzas gravitacionales. Gracias a este funcionamiento, se vuelve un sensor muy sensible a cualquier movimiento mínimo, incluso el propio pulso de la persona, lo que proporciona una gran robustez a esta funcionalidad del proyecto.

Tras el desarrollo de estos dos algoritmos y basándonos el la robustez mencionada, en el caso de que el dispositivo detecte una retirada con el sensor de proximidad, se activará la medición del

acelerómetro, si este algoritmo confirma la retirada se enviará la notificación correspondiente al servidor (Ver figura 3.1).

Nuevamente surge la duda en el caso de que el dispositivo no incorpore el sensor de proximidad si simplemente el acelerómetro será capaz de detectar la retirada. Se estudia varios sensores que apoyen al anterior y se encuentren disponibles en la mayoría de los smartwatches:

- En primer lugar se realizan varias pruebas obteniendo datos del sensor de frecuencia cardiaca, pensando que este sería el sensor más fiable.
  Al estudiar su funcionamiento, se llega a la conclusión de que esta suposición no es la más correcta ya que, tras la realización de varias pruebas, cuando se realiza una retirada del dispositivo las lecturas de este sensor nos proporcionan valores que podrían estar en el rango de frecuencia de una persona realizando ciertas actividades cotidianas. Por ello, la utilización de los datos que proporciona este sensor queda descartada.
- En segundo lugar se estudia la posibilidad de utilizar la ubicación del dispositivo junto con el algoritmo utilizado en el acelerómetro. Es decir, realizar una serie de lecturas de esta y comprobar que no hay diferencias significativas entre ellas, lo que indica que el dispositivo no se ha desplazado en un intervalo de tiempo configurable.
- Por último, y tras un largo análisis, se estudia la utilización del sensor de luz ambiental. Se trata de un sensor muy sensible a cambios que nos indica el grado de luminosidad que esta recibiendo. Al ser muy sensible, cualquier mínima variación en estos valores indicarían que el reloj se encuentra en la muñeca del individuo. Se aplicaría también el algoritmo mencionado anteriormente, se realizan varias mediciones comparando que estás no son significativamente diferentes que indica que el reloj ha sido retirado.

Tras analizar las cuatro propuestas anteriores y descartar el uso del sensor de frecuencia cardiaca, se decide utilizar los otros tres sensores con el fin de proporcionar una robustez mayor a esta funcionalidad. Se seguirá el modelo de una máquina de estados en el que la primera comprobación la realiza el algoritmo aplicado a los valores que se obtienen del proveedor de ubicación, si este detecta una retirada se pasa a la comprobación de los valores del acelerómetro y si su algoritmo detecta la retirada se comprueba finalmente los valores del sensor de luz ambiental, si este corrobora la retirada, se procede al envío de la notificación correspondiente al servidor. (Ver diagrama en la figura 3.2).

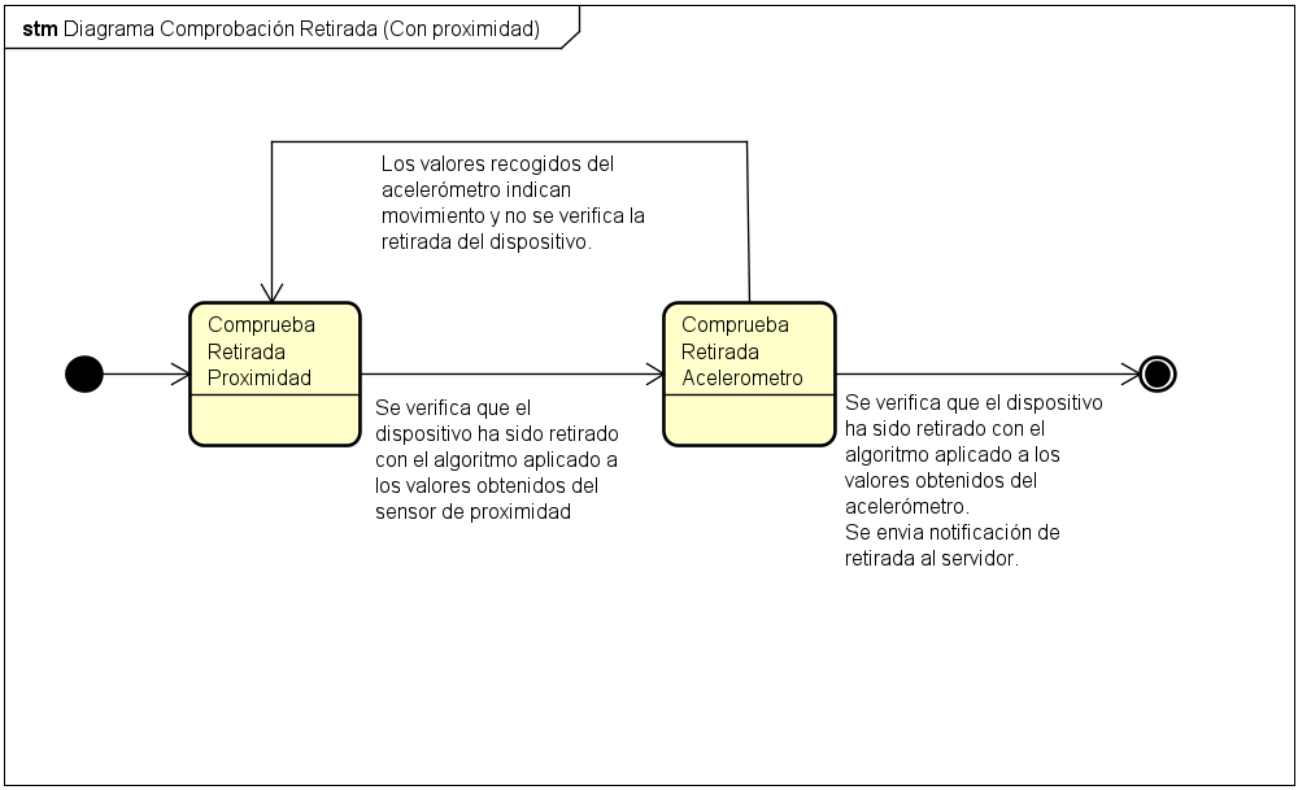

Figura 3.1: Diagrama comprobación retirada (Dispositivo con sensor de proximidad).

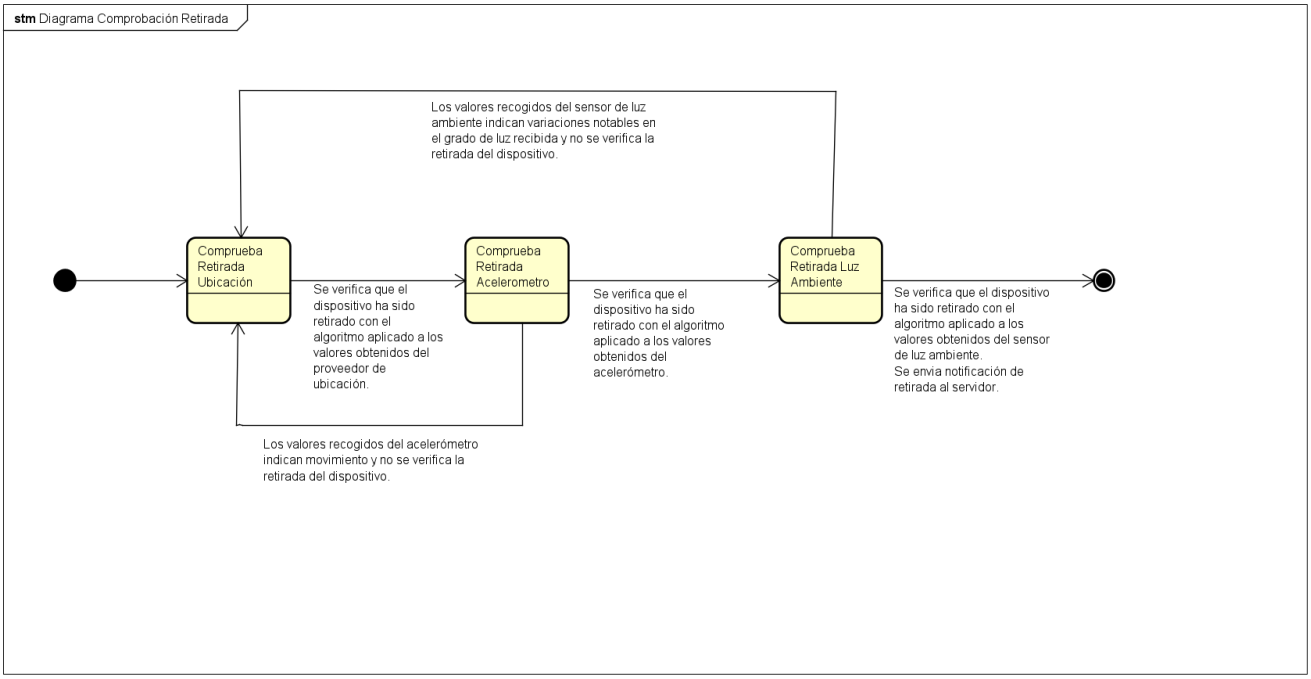

Figura 3.2: Diagrama comprobación retirada (Dispositivo sin sensor de proximidad).

Cabe mencionar que tras detectar una retirada del dispositivo, se deja de enviar la ubicación en forma de monitorización normal, ya que se seguirá realizando la comprobación de retirada continuamente, y será enviada dentro del periodo establecido en esta comprobación junto a la notificación de retirada. Por lo tanto, si el dispositivo se vuelve a colocar en la muñeca, se detectará nuevamente que no está retirado y por ello se volverá a realizar el envío de la ubicación con la monitorización normal. Esto permitirá al administrador saber cuanto tiempo estuvo retirado el dispositivo de la muñeca del portador.

Al finalizar la iteración obtenemos una versión de la aplicación con envío de la ubicación y detección de la retirada. Esta iteración se demora dos semanas respecto a la estimación inicial. Principalmente debido a la realización de los análisis correspondientes para determinar cual es el mejor método para detectar una retirada.

Tras el fin de esta fase, se concreta una reunión con los tutores con el fin de revisar los avances realizados en el proyecto y definir el alcance de la funcionalidad que se desarrollará en la siguiente iteración.

3.3. Iteración 3

Para esta iteración se propone realizar la funcionalidad correspondiente al desarrollo de un botón que el portador pueda accionar si se encuentra en una situación de emergencia. El accionamiento de este botón debe desencadenar en el envío de la notificación correspondiente al servidor monitorizado por el centro de control y recogida y envío de audio en tiempo real.

En primer lugar, cuando se acaba la funcionalidad correspondiente a la iteración anterior, se recibe la comunicación de que el servidor con el que se trabajará en este proyecto está montado. Es por ello que lo primero es la adaptación del código para que se realice el envío de datos al mismo. Tras esto se tiene una versión de la aplicación que monitoriza la ubicación y la envía al servidor y además detecta la retirada del dispositivo.

A continuación se procede a desarrollar la funcionalidad del botón de emergencia con su correspondiente envío del audio. En un principio, el botón de emergencia se configura con la pulsación de uno de los botones físicos del dispositivo, tras ser presionado 4 veces dentro de un periodo de tiempo asignado se muestra la vista mencionada.

Con toda la funcionalidad desarrollada y al no haber llegado a la fecha estimada de fin de iteración se decide realizar también en este periodo la ejecución de la app en segundo plano, cuyo desarrollo se explica en el capítulo 5.

Con la aplicación funcionando en segundo plano en el dispositivo surge un problema respecto al botón de emergencia y se debe a que una app Android solo es capaz de detectar pulsaciones en botones físicos o pantalla si tiene una actividad en primer plano, es decir, se está mostrando una vista al usuario. Como esta opción no se contempla, es necesario modificar el accionamiento del aviso de emergencia. La implementación nueva se basa en la notificación persistente, su implementación se comenta también en el capitulo 5.

Se propone la detención de la app desde los propios ajustes del dispositivo en la opción *FORZAR DETENCIÓN* para que únicamente pueda realizarlo el administrado del dispositivo. Además, al ejecutarse en dispositivos dedicados únicamente para esta aplicación, no es necesario finalizar la aplicación. Simplemente apagar el dispositivo y al encenderse de nuevo se iniciará automáticamente.

Con esto se da por finalizada la iteración en 4 semanas, ajustándose así a la estimación realizada en la planificación inicial, con una app con la funcionalidad completa que se especifica en los requisitos. Tras el fin de esta fase, se concreta una reunión con los tutores con el fin de revisar los avances realizados en el proyecto y definir el alcance de la funcionalidad que se desarrollará en la siguiente iteración.

3.4. Iteración 4

Como se estimó en la planificación inicial, se llega a esta fase con toda la funcionalidad desarrollada y por lo tanto se dedicará a la realización de las pruebas del funcionamiento de la aplicación. Tras la reunión correspondiente, se decide dedicar el tiempo de esta iteración también a la redacción de la memoria del proyecto, en paralelo con la ejecución de pruebas. Además se podrán realizar pequeñas modificaciones en el código de la aplicación en función de los resultados de las pruebas.

Al tratarse de una aplicación que recoge valores de sensores, que pueden llegar a depender de varios factores, se deciden realizar pruebas de funcionamiento de uso real de la aplicación y analizar los resultados. En grandes rasgos podemos dividir las pruebas realizadas en dos tipos que se comentan a continuación.

En primer lugar se realizan una serie de pruebas a las que llamamos pruebas de calibración, se trata de realizar pruebas analizando los valores que se recogen de los distintos sensores con el único fin de ajustar los valores de los umbrales comentados anteriormente, que sirven en este proyecto para detectar la retirada del dispositivo. Aunque si bien es cierto que estos valores se podrán configurar por el administrador correspondiente según el portador del dispositivo, se quiere dejar unos valores por defecto que realicen correctamente esta función.

En segundo lugar, se tiene una serie de pruebas de rendimiento, aunque el alcance de este proyecto se fijó en simplemente la aplicación funcional con las especificaciones requeridas, se cree que es importante estudiar el rendimiento del dispositivo en cuanto a tema de batería, ya que se trata de una aplicación que se estará ejecutando continuamente en segundo plano.

Estas pruebas traen consigo unos cambios mínimos en la aplicación que se comentan brevemente a continuación, debido a que se desarrollará más adelante este tema en la sección 6 dedicada a pruebas.

- Se decide crear una clase con los valores de configuración modificables para el funcionamiento de la aplicación. Estos valores se refieren a los umbrales e intervalos de detección de retirada y el intervalo de la monitorización de la ubicación, es decir, la frecuencia con la que se reciben actualizaciones. Se ha dejado la clase preparada con la intención de que estos datos se descarguen

del servidor cuando este tenga esta funcionalidad, aunque cuando este proyecto finaliza no se da ese caso.

- Tras los resultados obtenidos en las pruebas, se decide dejar sin modificaciones la parte de retirada en los dispositivos con sensor de proximidad. Respecto a los dispositivos que no incluyen sensor de proximidad se decide detectar la retirada únicamente con el sensor del acelerómetro. Se explicará esta decisión más detalladamente en la sección 6.
- Con los resultados obtenidos de las pruebas de rendimiento, se ha decidido realizar dos modos de uso de la aplicación. Se realizan gracias a las distintas prioridades que nos permite ajustar el monitorizador creado con la clase *LocationRequest* de Android. Las dos prioridades que se utilizan en este proyecto son las siguientes:
  - Alta prioridad: se prioriza la exactitud de la ubicación y se cumple estrictamente el intervalo de actualización de la misma. Se observará en los resultados de las pruebas como este modo consume altamente la batería del dispositivo.
  - Ahorro de batería: Se intenta buscar un equilibrio entre la precisión de la ubicación (con un posible error de unos 15 metros), el consumo de batería y el cumplimiento del intervalo que no es estricto.

    Esto último es importante, ya que al no cumplirse el intervalo estrictamente, la actualización dentro de un rango de tiempos. Por ello, realizando pruebas, se establecerán unos rangos por defecto en función de si se quieren recibir actualizaciones cada poco tiempo o si no es necesario esta precisión.

Tras acabar las pruebas y realizar los cambios correspondientes se obtiene la aplicación final con toda la funcionalidad desarrollada. Esta iteración se demora 3 semanas respecto a la estimación inicial.

Capítulo 4

Estado final de la aplicación

En este capítulo se procede a desarrollar el análisis, diseño y estructura final de la aplicación. Desarrollando las distintas historias de usuario, patrones utilizados o incluyendo los correspondientes diagramas.

4.1. Análisis

4.1.1. Análisis de requisitos

En esta sección se desarrolla los distintos requisitos establecidos en la reunión inicial con los tutores para el desarrollo del proyecto. Un requisito es una necesidad sobre el sistema que se documenta y existen 3 tipos: Funcionales, No funcionales y de información.

4.1.1.1. Requisitos funcionales

En la tabla 4.1 se exponen los distintos requisitos funcionales del proyecto. Estos requisitos son aquellos que describen como debe comportarse el sistema que se va a desarrollar. El formato en el que se exponen en la tabla consta del identificador del requisito, el nombre del requisito y una breve descripción del mismo.

ID	Nombre	Descripción
RF01	Enviar ubicación.	El sistema será capaz de enviar la ubicación en tiempo real del usuario.
RF02	Detección de uso.	El sistema será capaz de detectar si el usuario deja de llevar consigo el smartwatch.
RF03	Enviar aviso al centro de control.	El sistema será capaz de enviar un aviso al centro de control.
RF04	Enviar audio.	El sistema será capaz de enviar audio en tiempo real al sistema de control.
RF05	Programar preferencias por usuario.	El sistema permitirá establecer una serie de configuraciones en función del usuario.
RF06	Envío de datos a servidor.	El sistema será capaz de enviar datos a un servidor.
RF07	Almacenamiento en local.	El sistema será capaz de almacenar los datos en el dispositivo.

Tabla 4.1: Tabla requisitos funcionales.

4.1.1.2. Requisitos no funcionales

En la tabla 4.2 se describen los requisitos no funcionales del proyecto. Estos requisitos son aquellos que especifican restricciones o criterios técnicos del sistema que se va a desarrollar. El formato en el que se exponen en la tabla consta del identificador del requisito, el nombre del requisito y una breve descripción del mismo.

ID	Nombre	Descripción
RNF01	Almacenamiento de datos.	El sistema almacenará los datos de posicionamiento en una base de datos relacional SQLite mediante la librería Room.
RNF02	Entorno de desarrollo.	El sistema se desarrollará con Android Studio para Smartwatch con Android 6.0 en adelante (API 23) .
RNF03	Conexión a la red.	El sistema deberá conectarse a la red para poder enviar datos al centro de control.
RNF04	Conexión GPS.	El sistema deberá llevar activado en todo momento el sensor GPS.
RNF05	Gestión de la ubicación mediante Google.	El sistema mostrará las ubicaciones del usuario mediante el uso de la API de Google.
RNF06	Informe de errores.	El sistema informará mediante un mensaje si se produce algún error durante su ejecución y almacenarlos en un log de errores.
RNF07	Recogida de audio en tiempo aceptable.	El sistema deberá comenzar a recoger audio de manera inmediata (máximo 1 segundo de demora) tras la pulsación del botón de emergencia.
RNF08	Envío de la ubicación en tiempo aceptable.	El sistema deberá comenzar a enviar la ubicación en tiempo real de manera inmediata (máximo 5 segundos de demora) tras la pulsación del botón de emergencia.
RNF09	Uso de API para envío de audio en tiempo real.	El sistema realizará la función de envío de audio en tiempo real mediante el uso de la API externa de llamadas de WhatsApp.
RNF10	Uso de botón físico para envío del aviso.	El sistema enviará el aviso de emergencia al centro del control tras el accionamiento de un botón físico en el smartwatch.
RNF11	Detección de uso mediante sensores.	El sistema deberá ser capaz de verificar si el usuario deja de llevar consigo el dispositivo mediante el uso de los sensores de frecuencia cardiaca y acelerómetro.
RNF12	Doble confirmación en botón de emergencia.	El sistema deberá ser capaz de enviar un aviso al centro de control tras realizar una doble confirmación de accionamiento del botón de emergencia.
RNF13	Ejecución en segundo plano.	El sistema será capaz de ejecutarse en segundo plano.

Tabla 4.2: Tabla requisitos no funcionales.

4.1.1.3. Requisitos de información

En la tabla 4.3 se describen los requisitos de información del proyecto. Estos requisitos son aquellos que especifican como y qué datos debe almacenar el sistema a desarrollar. El formato en el que se exponen en la tabla consta del identificador del requisito, el nombre del requisito y una breve descripción del mismo.

ID	Nombre	Descripción
RI01	Ubicación del usuario.	El sistema deberá almacenar la ubicación exacta del usuario junto a la fecha y hora.
RI02	Registro de errores	El sistema almacenará un registro con los errores que se produzcan.
RI03	Configuración del usuario.	El sistema almacenará la configuración(contactos y frecuencia de muestreo del GPS) del usuario con un id único.

Tabla 4.3: Tabla requisitos de información.

4.1.2. Casos de uso

En primer lugar se muestran los diagramas con los casos de uso correspondientes a la propia aplicación del smartwatch (Figura 4.1).

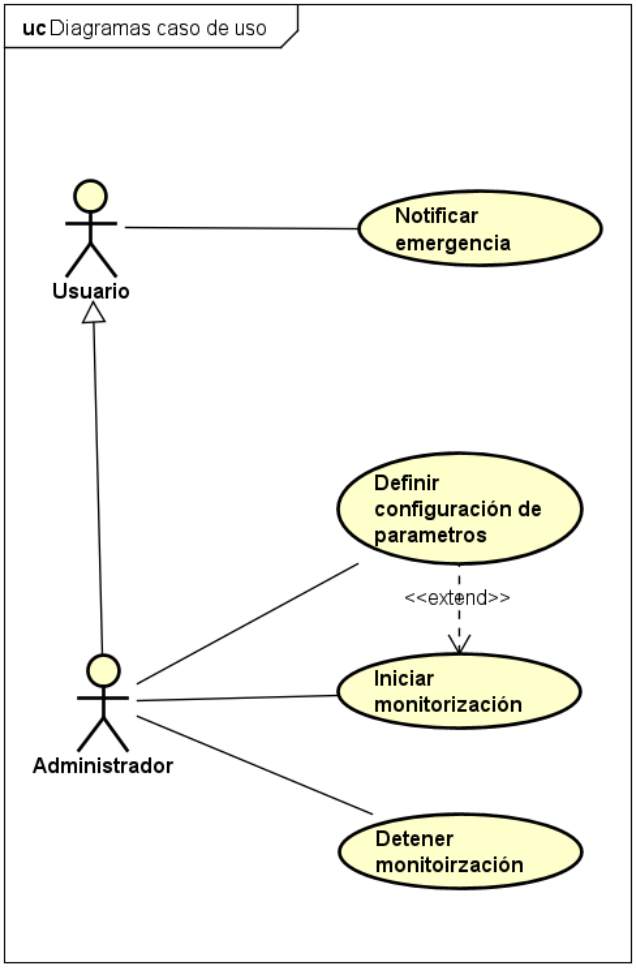

Figura 4.1: Diagrama casos de uso Aplicación Smartwatch.

Aunque esta parte es externa a este proyecto, también se incluye un diagrama de los casos de uso que puede realizar el administrador en el servidor (Figura 4.2) para facilitar la comprensión del proyecto global. Estos casos de uso correspondientes al servidor se muestran únicamente en el diagrama, no se explican detalladamente debido a que se encuentra fuera del propio proyecto.

Cabe mencionar que a partir del caso de uso 6 están relacionados con el sistema, aunque en el diagrama no estén relacionados explícitamente, se entiende que se realizan internamente mientras se ejecuta la aplicación o se desencadenan tras la acción de otro casos de uso.

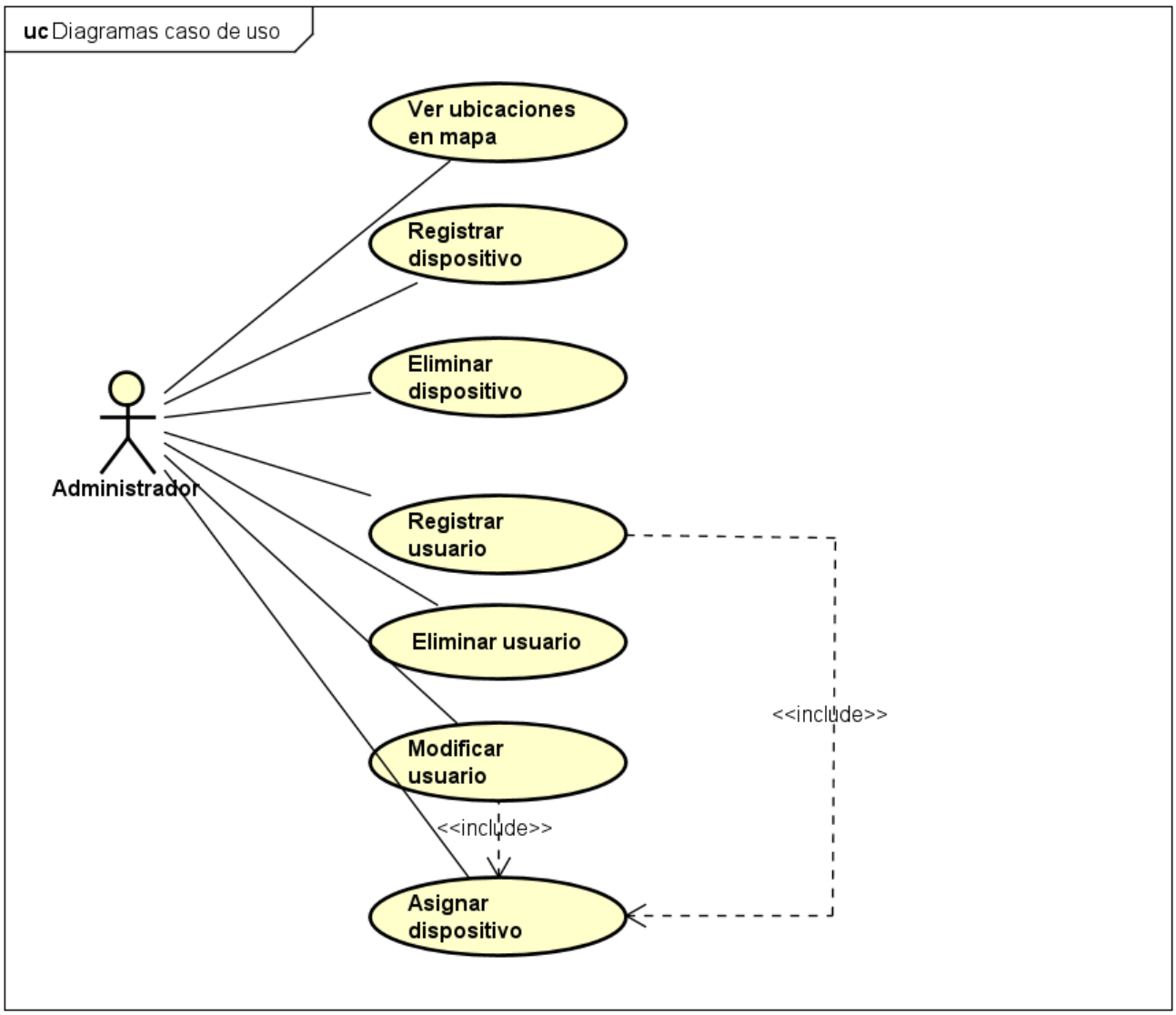

Figura 4.2: Diagrama casos de uso servidor.

ID	CU01
Nombre	Iniciar monitorización.
Descripción	El usuario deberá poder iniciar la monitorización iniciando la aplicación.
Precondición	1. Los parámetros de configuración deben haber sido configurados previamente.
Secuencia normal	1. El actor inicia la aplicación y con ello la monitorización del sistema. 2. El sistema inicia la monitorización de la detección de retirada del dispositivo, se ejecuta el caso de uso CU06. 3. El sistema inicia la monitorización de la ubicación, se ejecuta el caso de uso CU05.
Postcondición	Ninguna.
Excepciones	Ninguna
Frecuencia	Alta

Tabla 4.4: Caso de uso CU01

ID	CU02
Nombre	Detener monitorización.
Descripción	El usuario deberá poder detener la monitorización deteniendo la aplicación.
Precondición	1. La aplicación debe haber sido iniciada previamente.
Secuencia normal	1. El actor detiene la aplicación y con ello la monitorización del sistema. 2. El sistema detiene la monitorización de la detección de retirada del dispositivo. 3. El sistema detiene la monitorización de la ubicación.
Postcondición	Ninguna.
Excepciones	Ninguna
Frecuencia	Alta

Tabla 4.5: Caso de uso CU02

ID	CU03
Nombre	Definir configuración de los parámetros.
Descripción	El usuario deberá poder modificar los parámetros cuando desee.
Precondición	Ninguna
Secuencia normal	1. El actor solicita modificar los parámetros. 2. El sistema solicita al usuario los parámetros a modificar. 3. El actor introduce los parámetros a modificar. 4. El sistema almacena los parámetros.
Postcondición	Ninguna.
Excepciones	Ninguna
Frecuencia	Media

Tabla 4.6: Caso de uso CU03

Estado final de la aplicación

ID	CU04
Nombre	Notificar emergencia.
Descripción	El usuario deberá poder notificar una emergencia cuando se desee.
Precondición	La aplicación debe estar iniciada.
Secuencia normal	1. El actor notifica. 2. El sistema solicita la confirmación de emergencia. 3. El actor confirma la emergencia. 4. El sistema envía la ubicación, se ejecuta el caso de uso 07. 5. El sistema realiza una llamada de voz.
Postcondición	Ninguna.
Excepciones	3a. Si el actor no confirma la emergencia, el caso de uso queda sin efecto.
Frecuencia	Media

Tabla 4.7: Caso de uso CU04

ID	CU05
Nombre	Monitorizar ubicación.
Descripción	El sistema deberá monitorizar la ubicación mientras la aplicación esté iniciada.
Precondición	La aplicación debe estar iniciada.
Secuencia normal	1. El sistema obtiene la ubicación. 2. El sistema envía la ubicación al servidor, se realiza el caso de uso 07. 3. Si la aplicación sigue iniciada, se vuelve a ejecutar le paso 1 cuando el intervalo de tiempo se ha cumplido.
Postcondición	Ninguna.
Excepciones	1a. Si el sistema no puede obtener la ubicación, se notifica un error y el caso de uso queda sin efecto.
Frecuencia	Alta

Tabla 4.8: Caso de uso CU05

ID	CU06
Nombre	Monitorizar retirada del dispositivo.
Descripción	El sistema deberá detectar si el dispositivo es retirado del portador a través de sus sensores.
Precondición	La aplicación debe estar iniciada.
Secuencia normal	1. El sistema obtiene los valores del sensor de proximidad. 2. El sistema comprueba los valores del sensor y detecta que el dispositivo se ha retirado. 3. El sistema obtiene los valores del acelerómetro. 4. El sistema comprueba los valores del acelerómetro y detecta que se ha retirado el dispositivo. 5. El sistema envía una notificación de retirada, se ejecuta el caso de uso 07. 6. Si la aplicación sigue iniciada, se vuelve a ejecutar le paso 1 cuando el intervalo de tiempo se ha cumplido.
Postcondición	Ninguna.
Excepciones	1a. Si el dispositivo no incluye sensor de proximidad, el sistema continua en el paso 3. 2a. Si no se detecta retirada, el sistema ejecuta el paso 1 de nuevo. 4a. Si no se detecta retirada, el sistema ejecuta el paso 3 de nuevo.
Frecuencia	Alta

Tabla 4.9: Caso de uso CU06

ID	CU07
Nombre	Enviar notificación.
Descripción	El sistema deberá enviar la notificación junto con la ubicación al dispositivo.
Precondición	La aplicación debe estar iniciada. Se ha solicitado el envío de la notificación con un tipo concreto.
Secuencia normal	1. El sistema obtiene los distintos valores del dispositivo. 2. El sistema crea la notificación con la ubicación, el tipo y los distintos valores del dispositivo. 3. El sistema recupera todas las notificaciones almacenadas de la base de datos. 4. El sistema envía al servidor las notificaciones recuperadas de la base de datos. 5. El sistema envía al servidor la notificación creada. 6. El sistema vacía la base de datos local.
Postcondición	Ninguna.
Excepciones	3a. Si no hay conexión a internet, el sistema almacena la notificación en la base de datos local y el caso de uso queda sin efecto.
Frecuencia	Media

Tabla 4.10: Caso de uso CU07

4.1.3. Modelo de dominio

Siguiendo los requisitos de información definidos al inicio del proyecto junto con los requisitos funcionales y los casos de usos descritos anteriormente, se elabora el diagrama de clases que se representa en la figura 4.3 y en torno al cual se ha desarrollado la estructura de la aplicación.

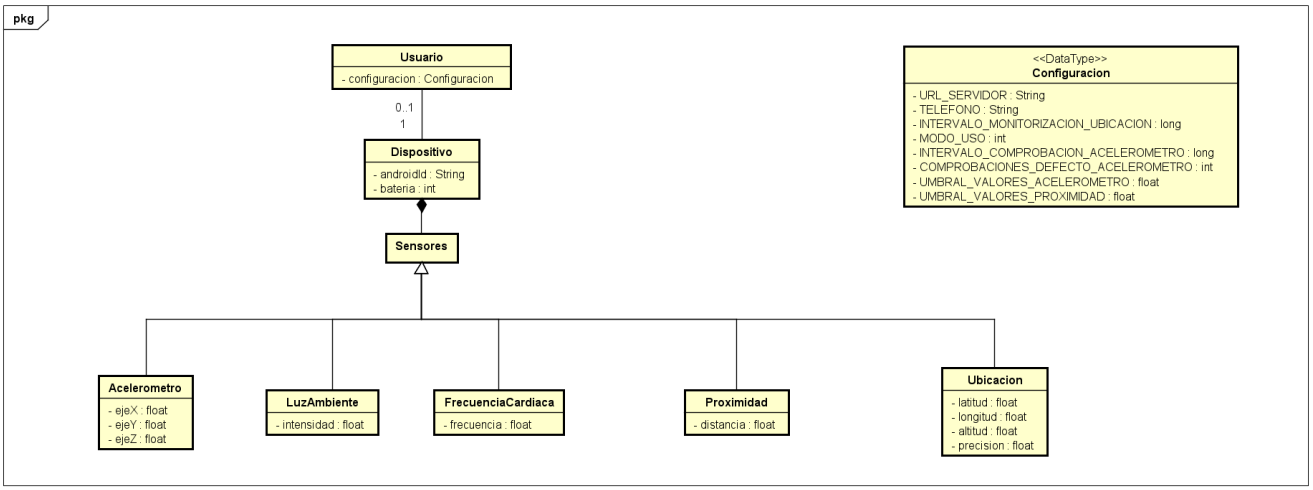

Figura 4.3: Modelo de dominio de la aplicación.

Este modelo de domino podemos interpretarlo de la siguiente manera:

La clase *Usuario* representa al propio usuario de la aplicación. Este usuario se puede relacionar con el correspondiente *Dispositivo* que realiza las monitorizaciones correspondientes a través de los sensores, tanto la clase *Acelerometro* como *Proximidad* se registran en diferentes listeners para poder detectar cambios en sus valores. Además se incluyen los sensores *LuzAmbiente* y *FrecuenciaCardiaca* ya que como se comenta anteriormente, se dejan las clases en el proyecto, a petición de los tutores pensando en un mayor alcance futuro de este, aunque estas clases no estén implicadas en la funcionalidad del sistema en este momento. La clase *Ubicacion* es la encargada de obtener la ubicación del dispositivo, según corresponda, que será enviada al servidor. Por último, se representa el DataType *Configuracion* que permite configurar los distintos parámetros de la aplicación en función del usuario con el que se utilice.

4.1.4. Diagramas de actividad

Esta sección se incluyen los diagramas de actividad correspondientes a la ejecución de la app casos de uso descritos en el apartado anterior.

Es importante mencionar que la aplicación se encuentra en ejecución continua, es decir, aunque en estos diagramas que se han separado por casos de uso la actividad finalice, el hilo principal siempre

se estará ejecutando y las actividades de monitorización se repetirán continuamente cumpliendo los intervalos establecidos. Algunos diagramas han sido divididos en varios para una mejor presentación en el documento.

4.1.4.1. CU01. Iniciar Monitorización

Cuando el usuario desea iniciar una monitorización, es decir, iniciar la aplicación, en primer lugar se comprueban si los permisos necesarios para que esta se pueda desplegar están concedidos, este caso se debe a que la aplicación ya ha sido iniciada anteriormente en ese dispositivo y se otorgaron estos permisos. Entres estos permisos se encuentran de ubicación, llamada, etc.

En caso de que no se dé el caso, se solicitan estos permisos al usuario y una vez concedidos, se ejecuta la aplicación en segundo plano. (Ver figura 4.4).

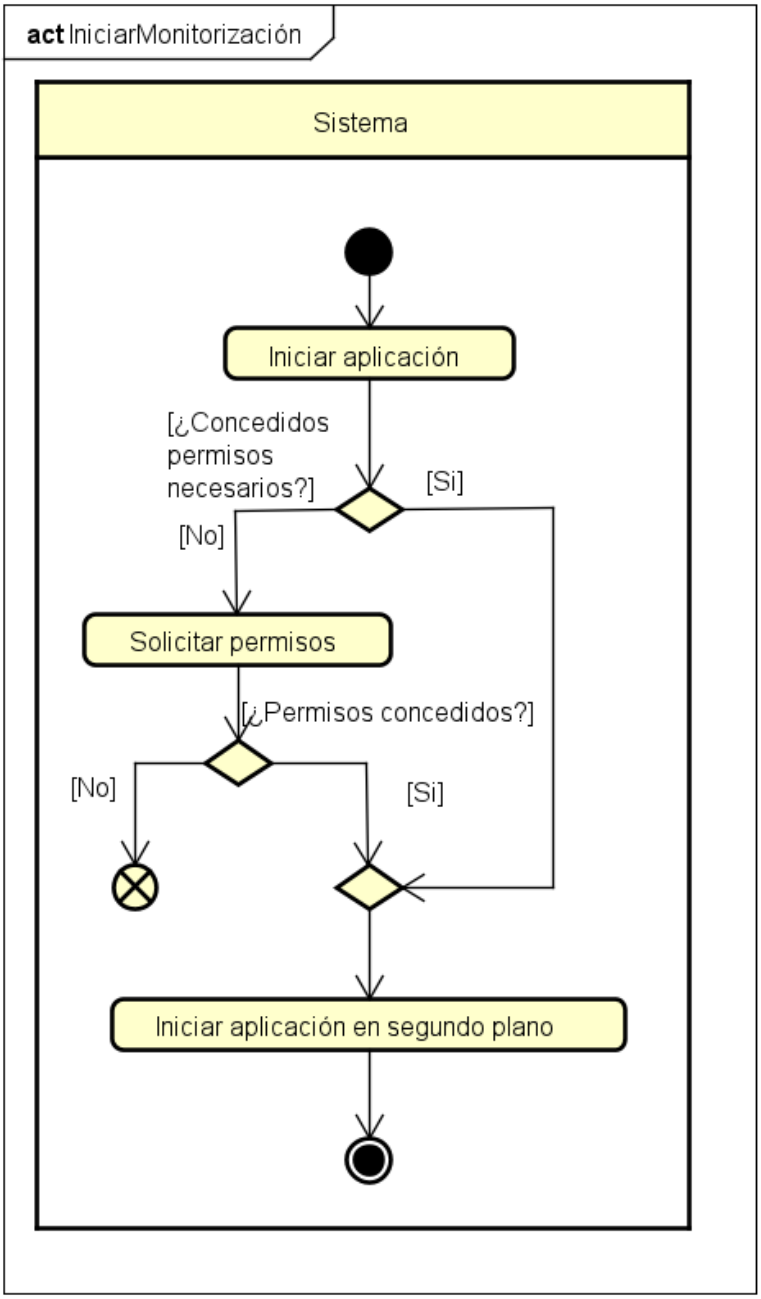

Figura 4.4: Diagrama de actividad correspondiente al CU01.

4.1.4.2. CU02. Parar Monitorización

En este caso no se expone diagrama de actividad, ya que como consecuencia de su contexto, para evitar que un usuario paciente pueda detener la aplicación con facilidad la aplicación debe detenerse desde los ajustes del dispositivo forzando su detención o apagando el dispositivo.

Tras analizar varias ideas para que el administrador pueda detener la aplicación con una serie de

combinaciones de comandos se llega a la conclusión de que con esta solución puede llegar a detenerse en casos que no se desee. Es por ello que se ha decidido detenerla desde ajustes, aunque no es la forma más correcta en la mayoría de nuestras aplicaciones, en nuestro caso si se detiene de esta forma significa que la monitorización en ese instante ya no interesa al administrador y por ello no se va a producir una pérdida de información.

Cabe mencionar que como esta aplicación se propone para usos en ámbitos específicos y por ello se ejecutará en dispositivos dedicados exclusivamente a estas monitorizaciones, no es necesario finalizar nunca esta aplicación. El administrador entregará el dispositivo al paciente que desea monitorizar y en ese momento lo registrará en el servidor, anteriormente el dispositivo simplemente estará apagado.

4.1.4.3. CU03. Definir configuración de parámetros

Cuando el administrador desea registrar unos parámetros de configuración de la aplicación debe modificar las variables estáticas de la clase configuración del proyecto. Como se ha comentado anteriormente, la idea de desarrollo de este caso de uso es que estos parámetros se modifiquen en el servidor y la clase Configuración simplemente descargue estos parámetros y actualice sus variables.

Como el servidor es externo a este proyecto y no tiene esta funcionalidad, se ha dejado preparado en forma de clase para una implementación más fácil en el futuro si se diese el caso. (Ver figura 4.5).

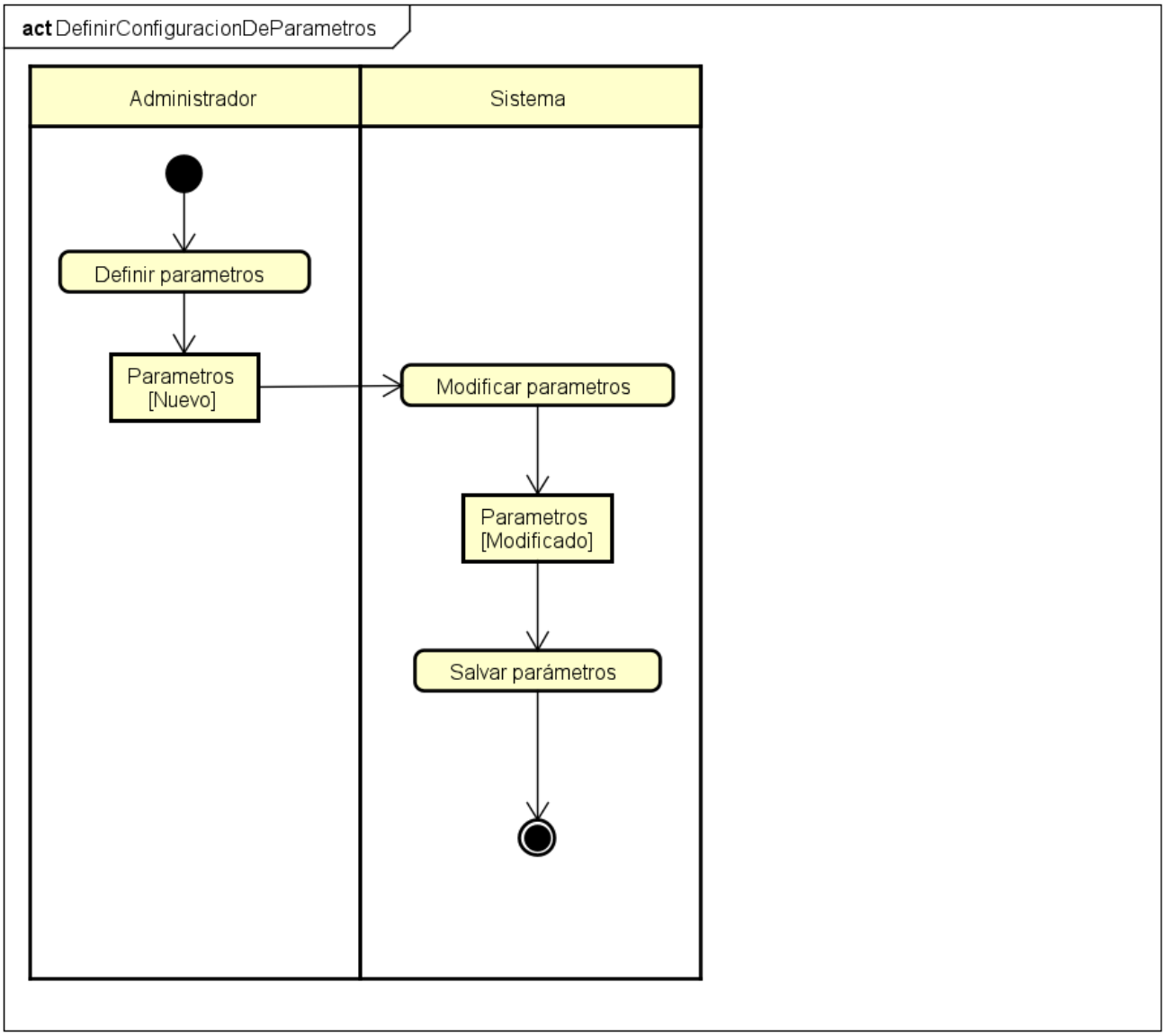

Figura 4.5: Diagrama de actividad correspondiente al CU03.

4.1.4.4. CU04. Notificar emergencia

Cuando el usuario desea reportar una emergencia este acciona el botón que se muestra en la notificación persistente de la app y el sistema muestra la vista correspondiente a la confirmación de esta. (Ver figura 4.6).

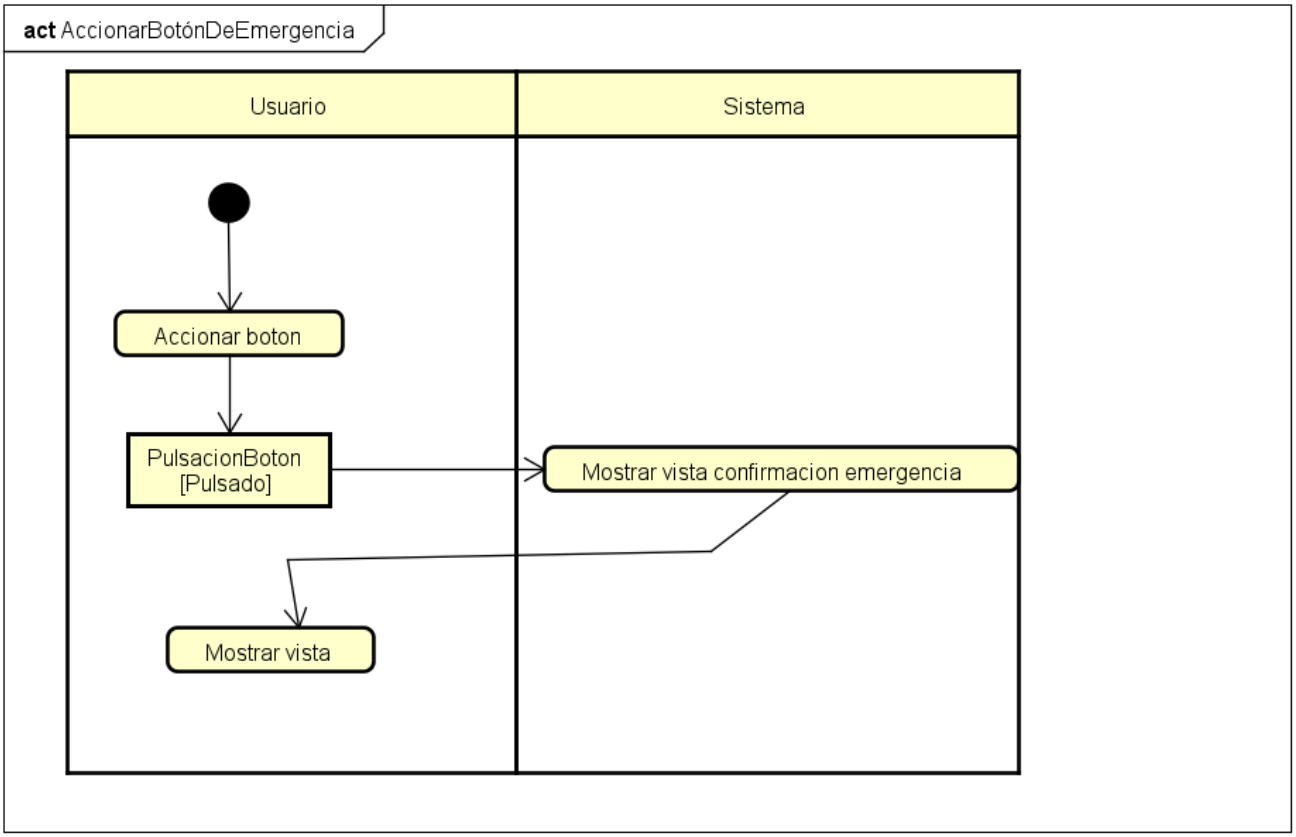

Figura 4.6: Diagrama de actividad correspondiente al CU04.

Cuando la vista de confirmación de emergencia es mostrada al usuario, se compone de dos botones, Cancelar y Aceptar.

En el caso de que se pulse el botón Cancelar, esta actividad finaliza. En el caso de que se confirme la emergencia, el sistema la gestiona realizando la llamada al número configurado en los parámetros de la aplicación (Ver figura 4.8) y, simultáneamente, obtiene la ubicación actual del dispositivo y envía una notificación de emergencia al servidor. (Ver figura 4.7 y 4.16).

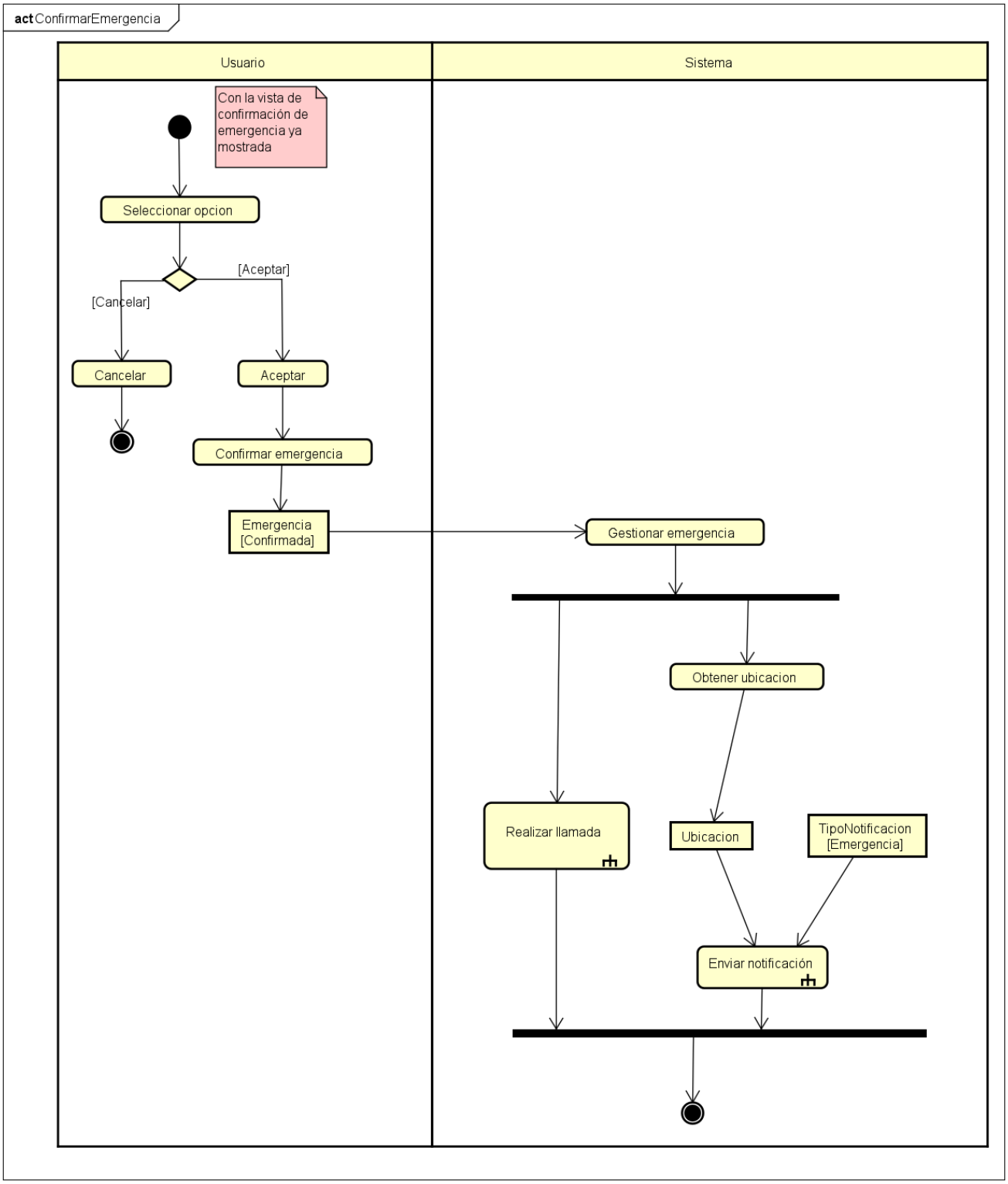

Figura 4.7: Diagrama de actividad correspondiente al CU04-2.

Para realizar la llamada, el sistema utiliza el número de teléfono configurado y realiza la llamada mediante la aplicación por defecto.

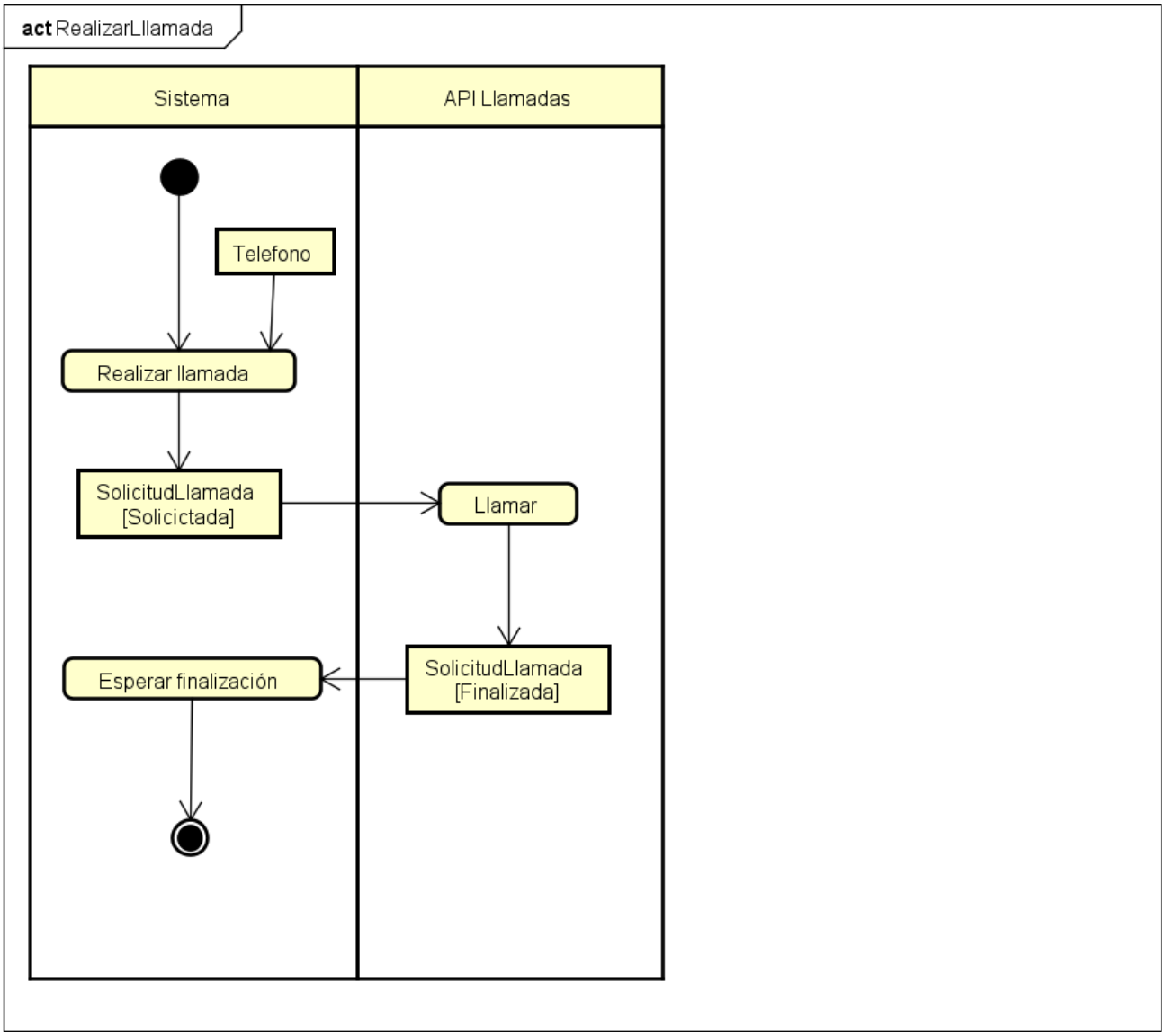

Figura 4.8: Subdiagrama de actividad correspondiente a Realizar Llamada.

4.1.4.5. CU05. Monitorizar ubicación

Cuando se inicia la monitorización de la ubicación del dispositivo, se obtienen de la clase configuración tanto el intervalo de medición como la prioridad que se aplica al monitorizador creado gracias a la clase LocationRequest. (Ver figura 4.9).

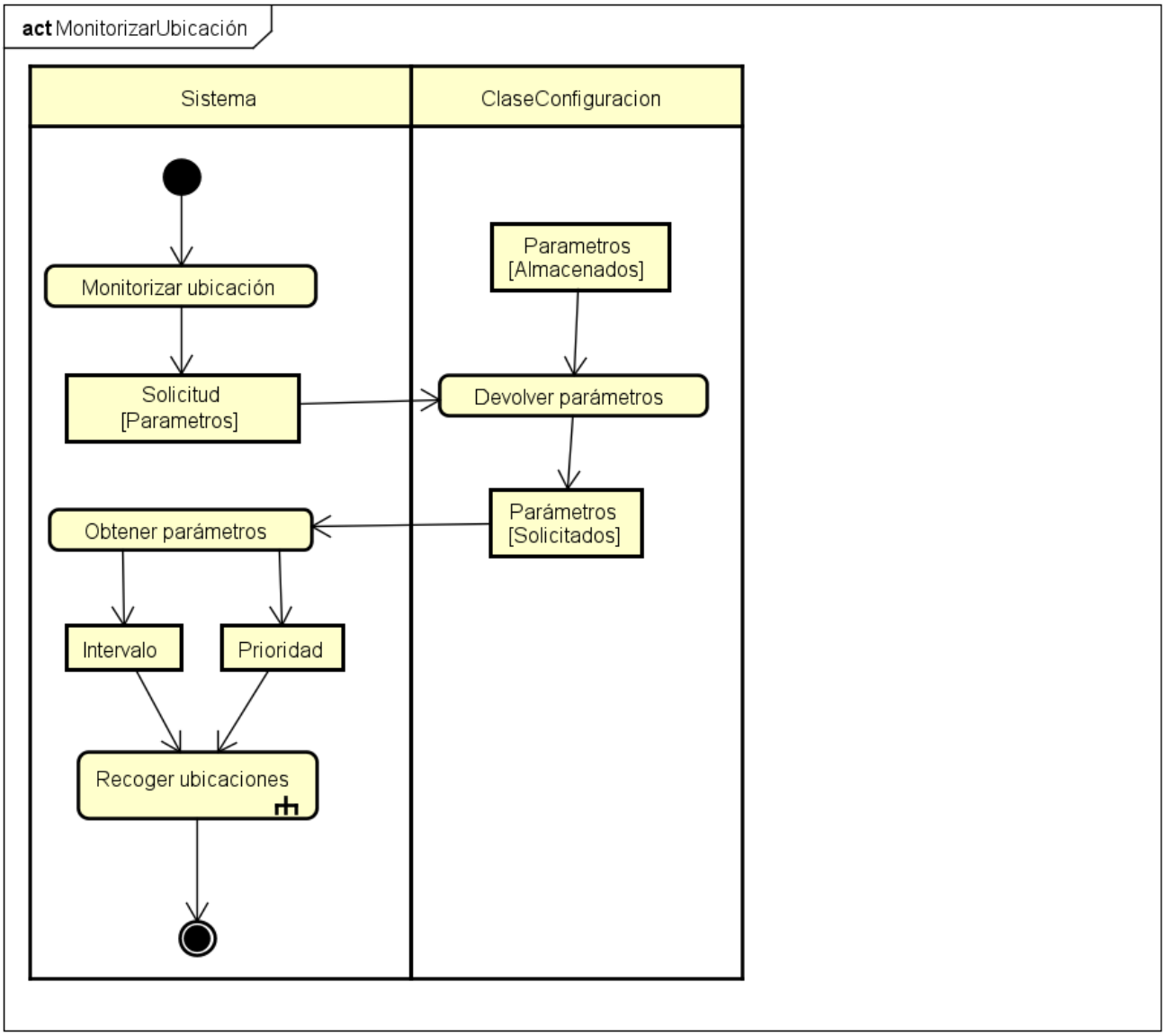

Figura 4.9: Diagrama de actividad correspondiente al CU05.

En este caso se incorpora otro subdiagrama (Ver figura 4.10) para explicar la actividad de recoger ubicaciones. Estas ubicaciones son proporcionadas por el monitorizador (clase LocationRequest), con esta ubicación se crea el JSON (Con los campos necesarios como son latitud, longitud o altitud y otros que también se añaden como lo es la batería del dispositivo en ese momento) que se enviará al servidor.

Antes de este envío se comprueba que haya conexión a internet disponible, si es así, el envío se realiza (Ver figura 4.16). En caso contrario, el JSON se almacena en la base de datos local del dispositivo para su posterior envío cuando sea posible.

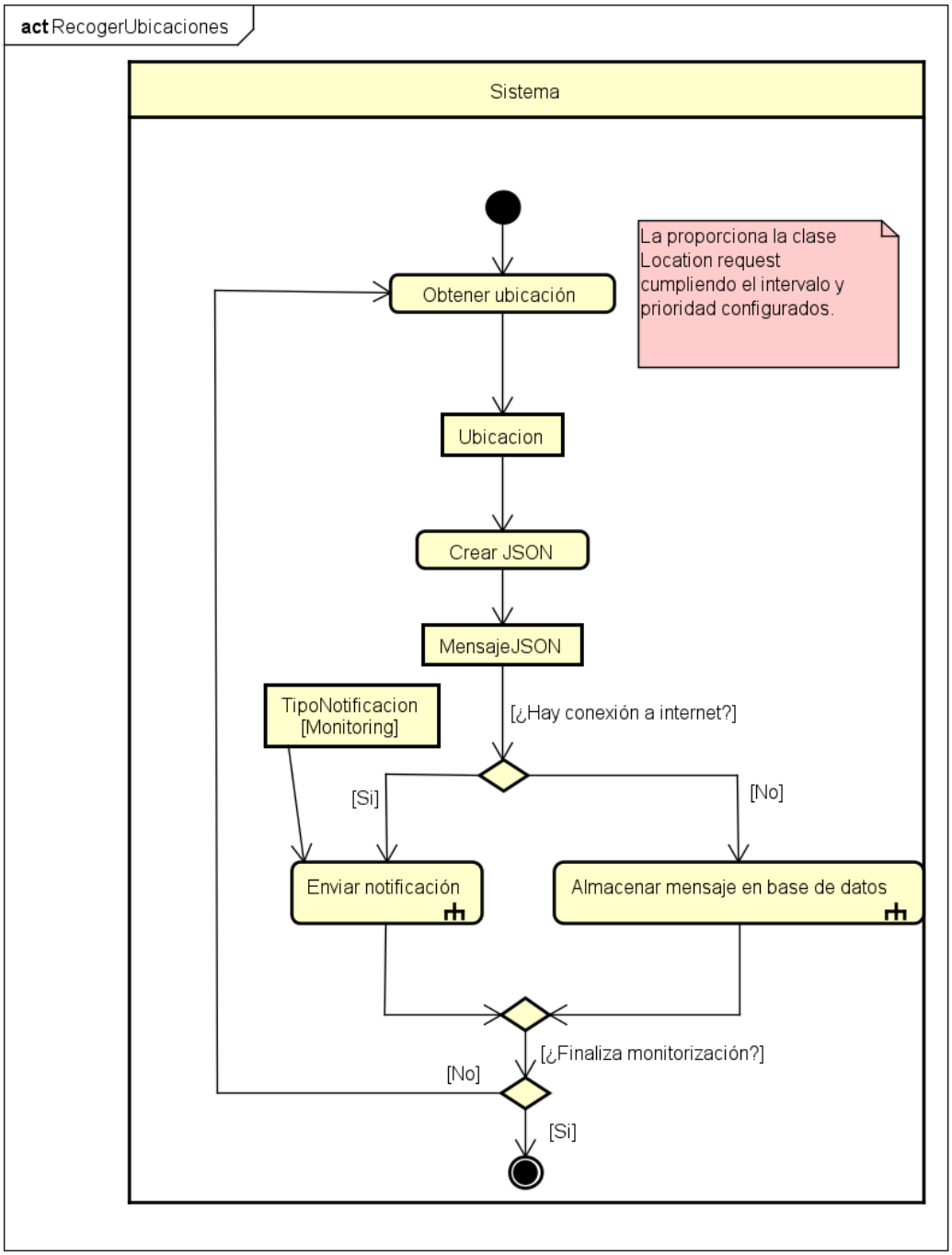

Figura 4.10: Subdiagrama de actividad correspondiente a Recoger Ubicaciones.

Cuando el dispositivo no tiene conexión a internet, el mensaje JSON creado para el envío se almacena en la base de datos local desplegada con Room. (Ver figura 4.11).

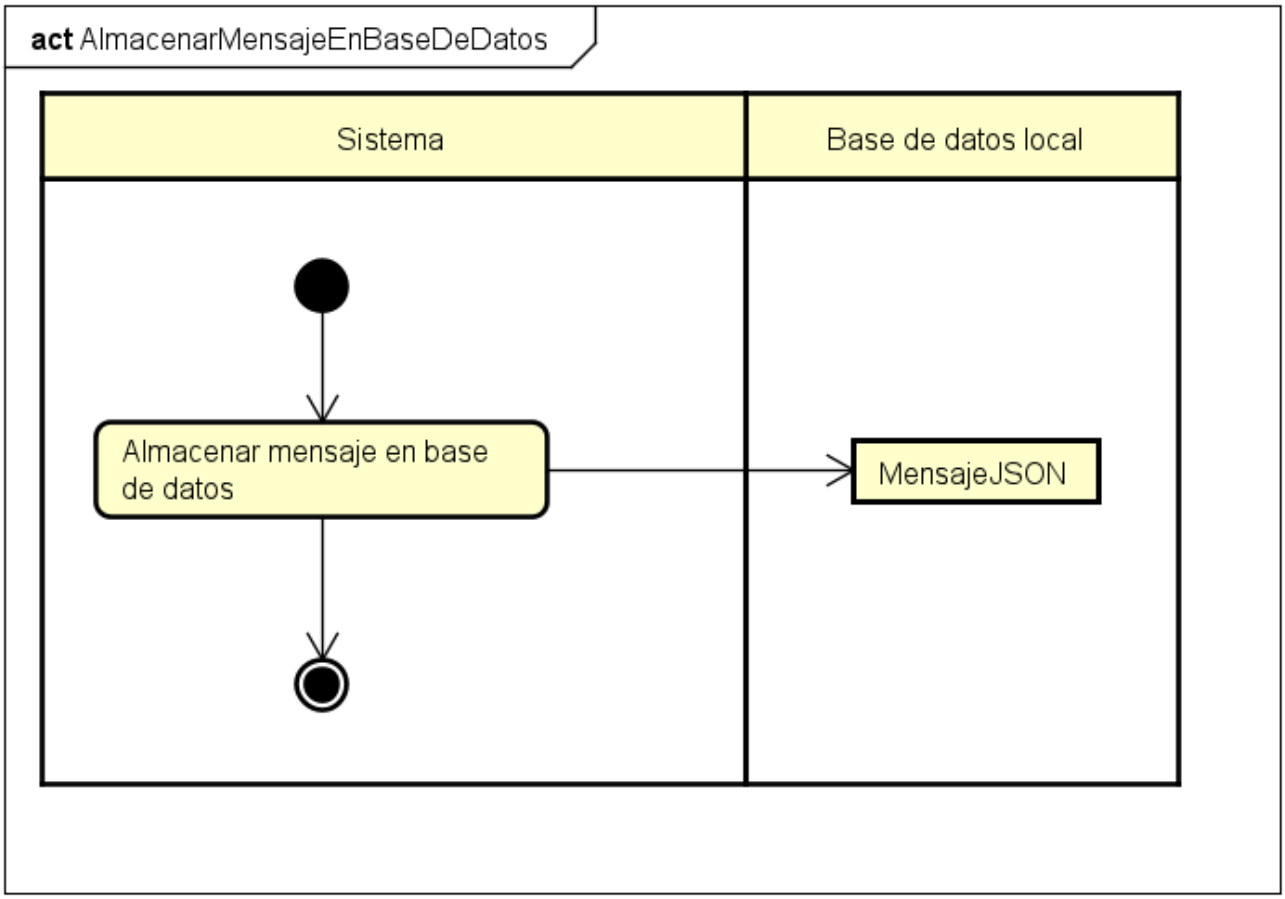

Figura 4.11: Subdiagrama de actividad correspondiente a Almacenar Mensaje en Base de Datos.

4.1.4.6. CU06. Monitorizar retirada dispositivo

Cuando se inicia la monitorización de la retirada del dispositivo, en primer lugar obtenemos de la clase Configuración tanto el intervalo de medida (y el número de comprobaciones a realizar) como los umbrales para detectar esa retirada. (Ver figura 4.12).

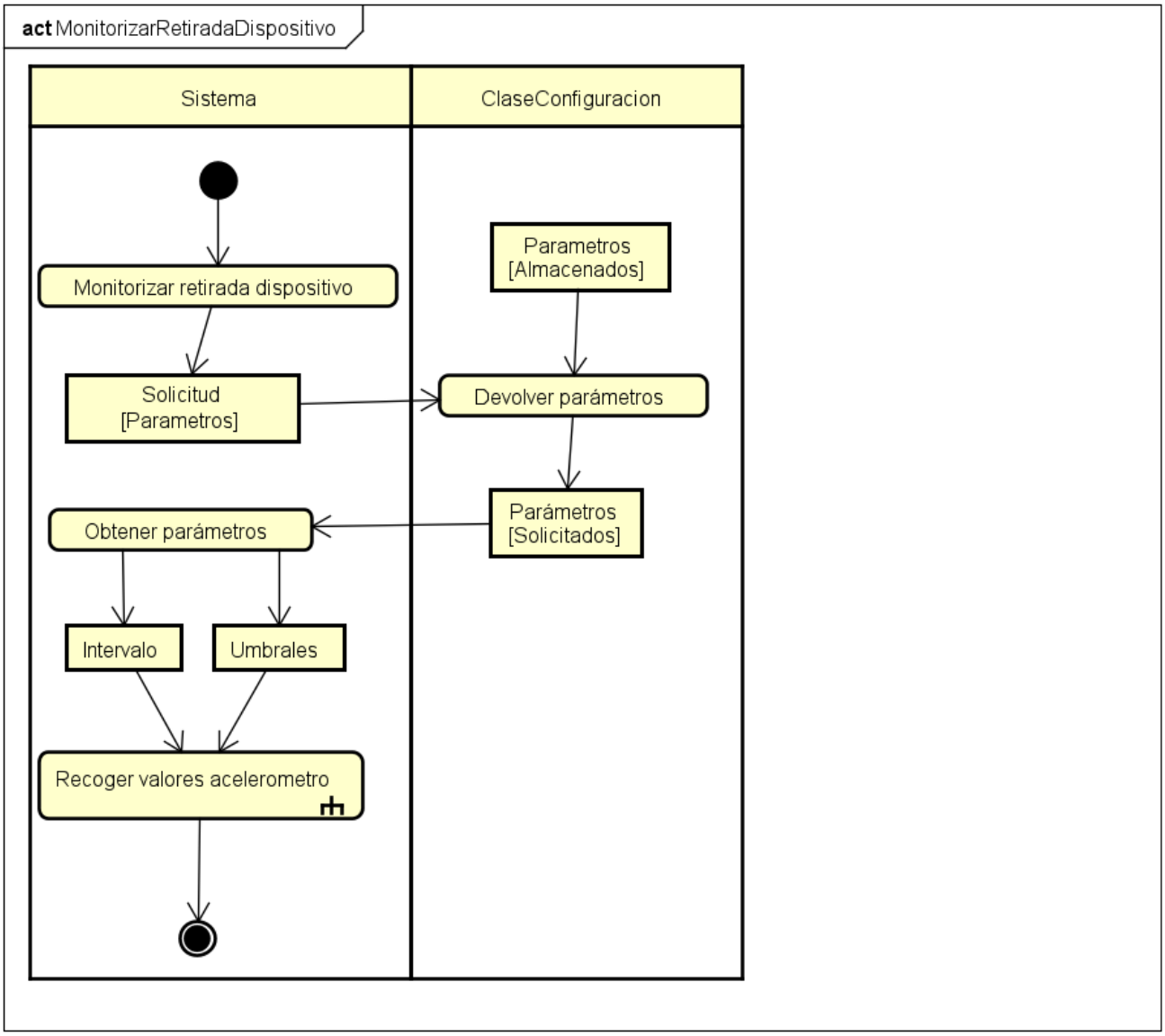

Figura 4.12: Diagrama de actividad correspondiente al CU06.

En este caso se incluye un segundo diagrama que refleja la recogida de datos del acelerómetro. (Ver figura 4.13). En primer lugar se obtienen los valores de los 3 ejes, con estos valores se aplica el algoritmo desarrollado para detectar la retirada, si se determina como retirado, se recoge la ubicación del dispositivo en ese momento y se envía la correspondiente notificación de retirada al servidor (Ver figura 4.16).

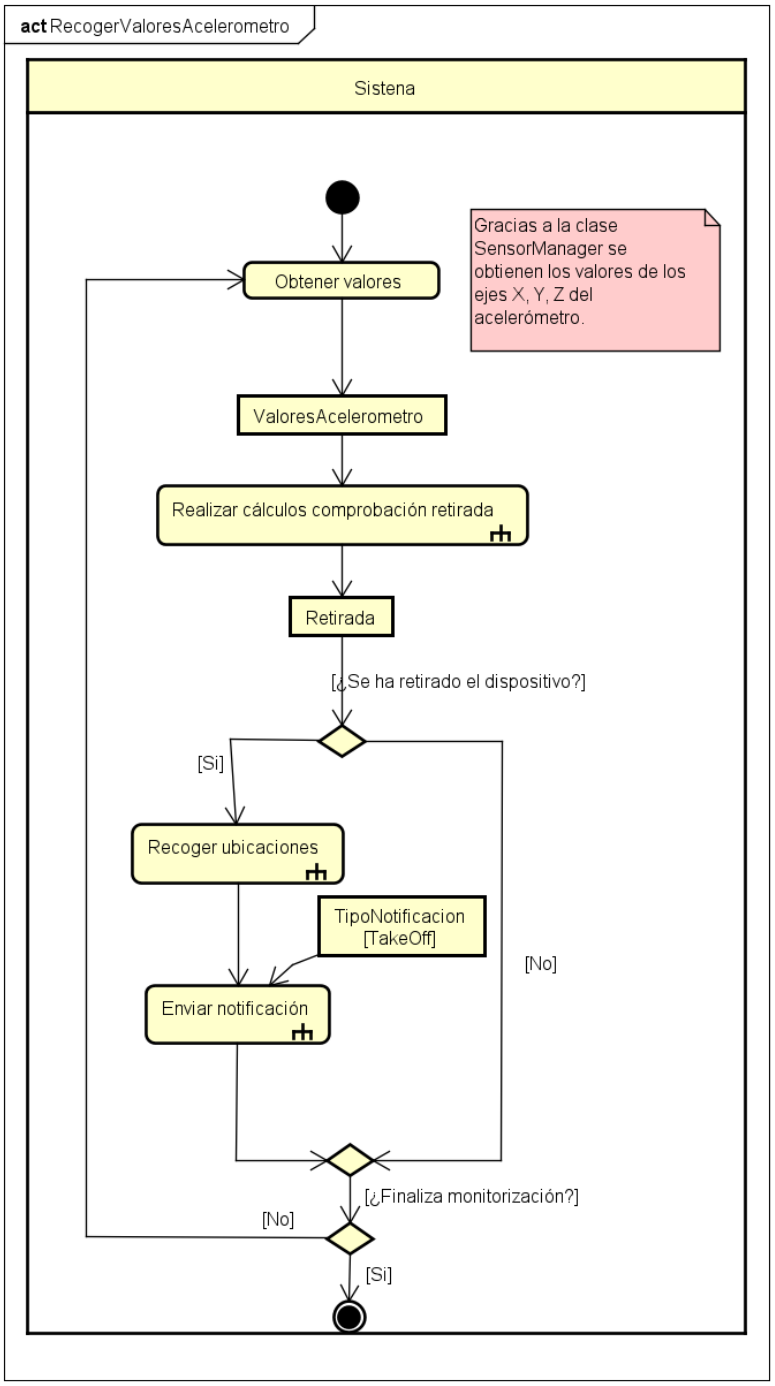

Figura 4.13: Subdiagrama de actividad correspondiente a Recoger Valores Acelerómetro.

El siguiente subdiagrama de actividad trata de explicar el algoritmo desarrollado para realizar el cálculo y comprobar si el reloj ha sido retirado. Gracias a los valores del acelerómetro, se comprueban si existen diferencias significativas entre los valores recientes obtenidos y los valores de la medición anterior. Esta diferencia se compara con el umbral determinado por el administrador en la clase Configuración.

Si este cálculo está por debajo del umbral indica que las dos mediciones son similares y se comprueba si se han realizado el número de comprobaciones determinado con el fin de dar robustez al algoritmo. Si es así, el sistema determina que el dispositivo ha sido retirado del portador. En caso contrario no se corrobora esta retirada, la actividad finaliza y el hilo principal se repetiría para volver a realizar esta comprobación. (Ver figura 4.14).

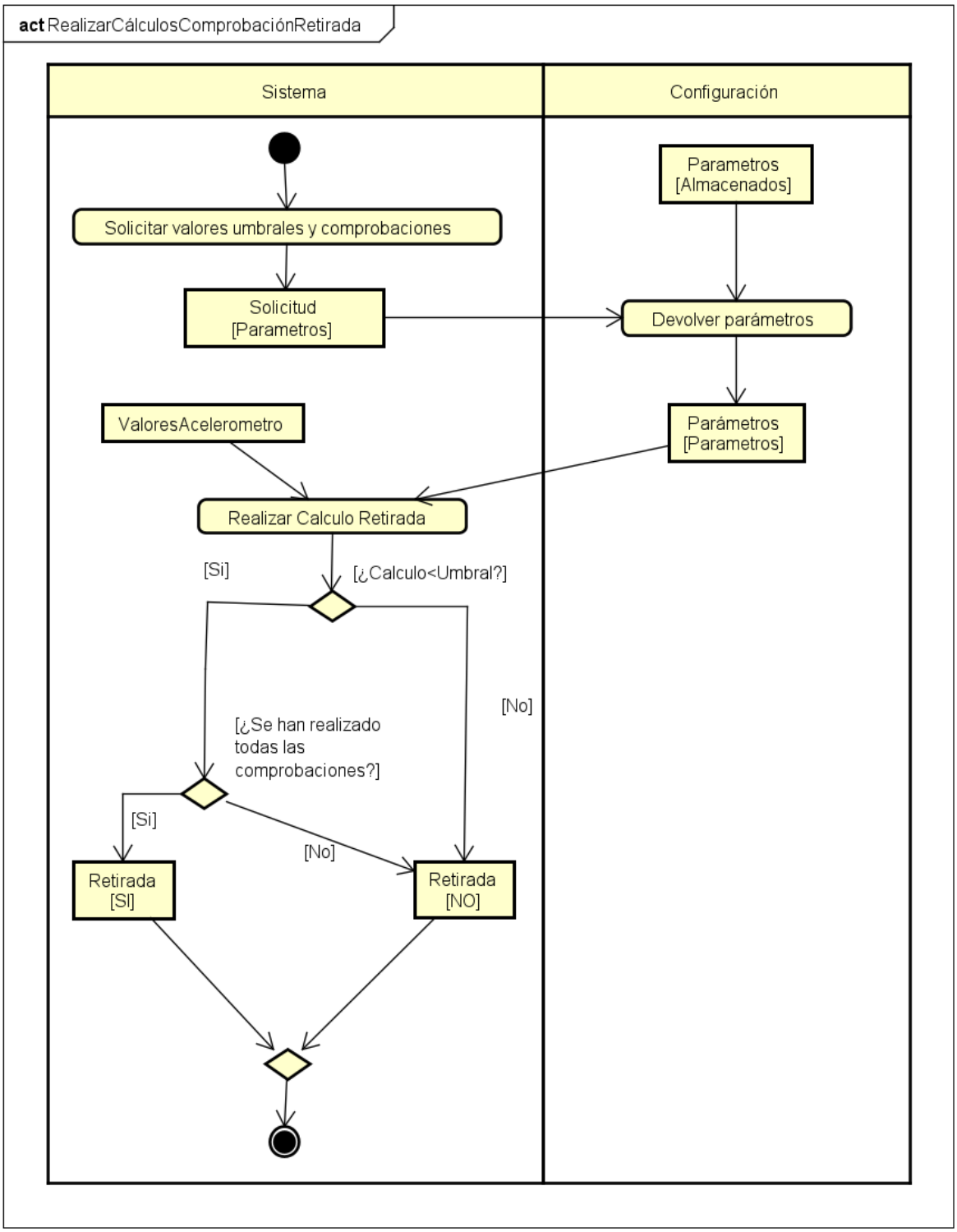

Figura 4.14: Subdiagrama de actividad correspondiente a Realizar Cálculos Comprobación Retirada.

A continuación se muestra el subdiagrama correspondiente a recoger las ubicaciones, este subdiagrama es igual que el anterior ya que se ejecuta de nuevo parte del caso de uso de monitorizar la ubicación para obtener la ubicación actual del dispositivo.

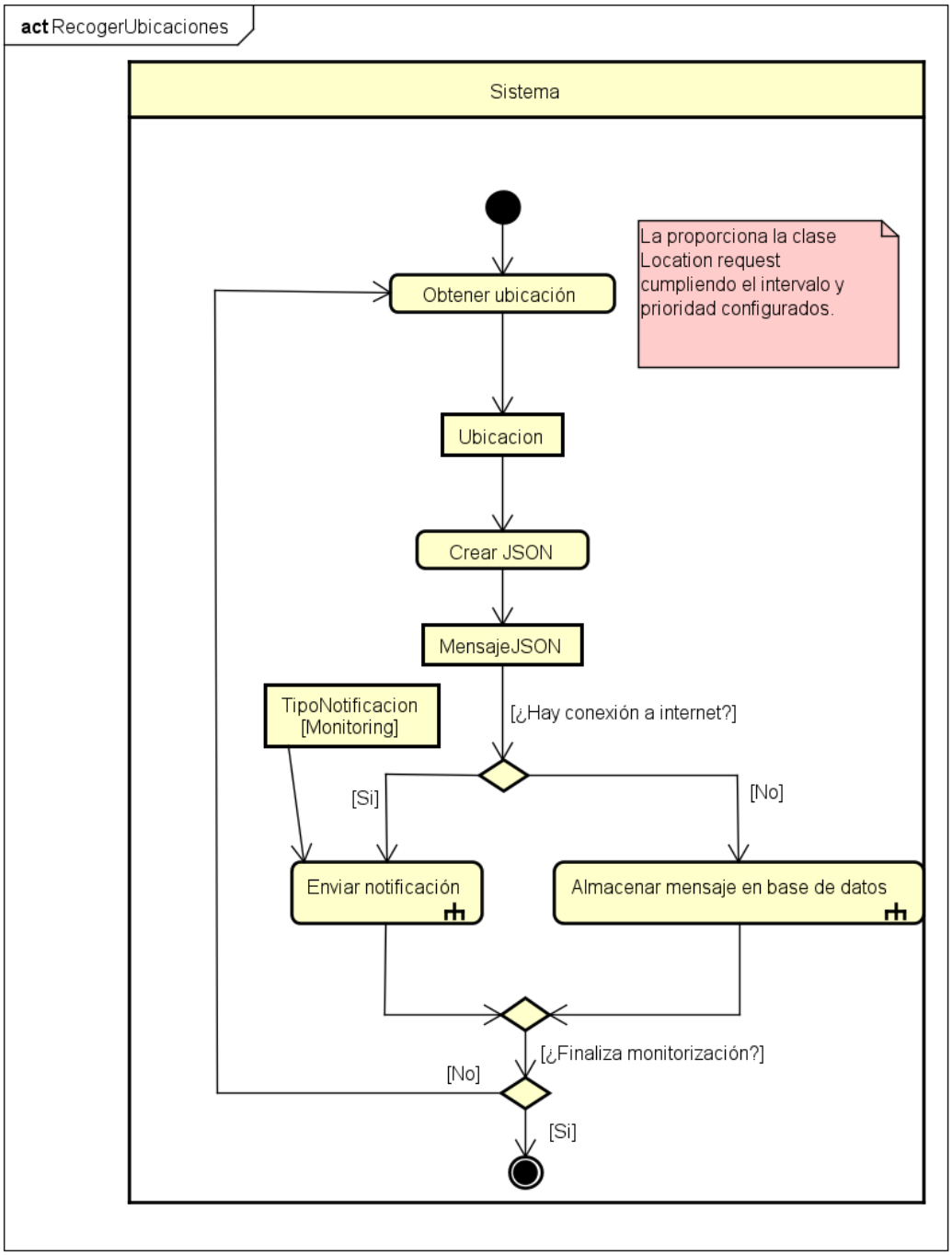

Figura 4.15: Subdiagrama de actividad correspondiente a Recoger Ubicaciones.

4.1.4.7. CU07. Enviar notificación

El siguiente diagrama de actividad muestra como el sistema envía la notificación al servidor, obteniendo en primer lugar todos los JSON almacenados en la base de datos y enviándolos junto con el nuevo mensaje que se crea con la ubicación obtenida y el tipo de notificación correspondiente.

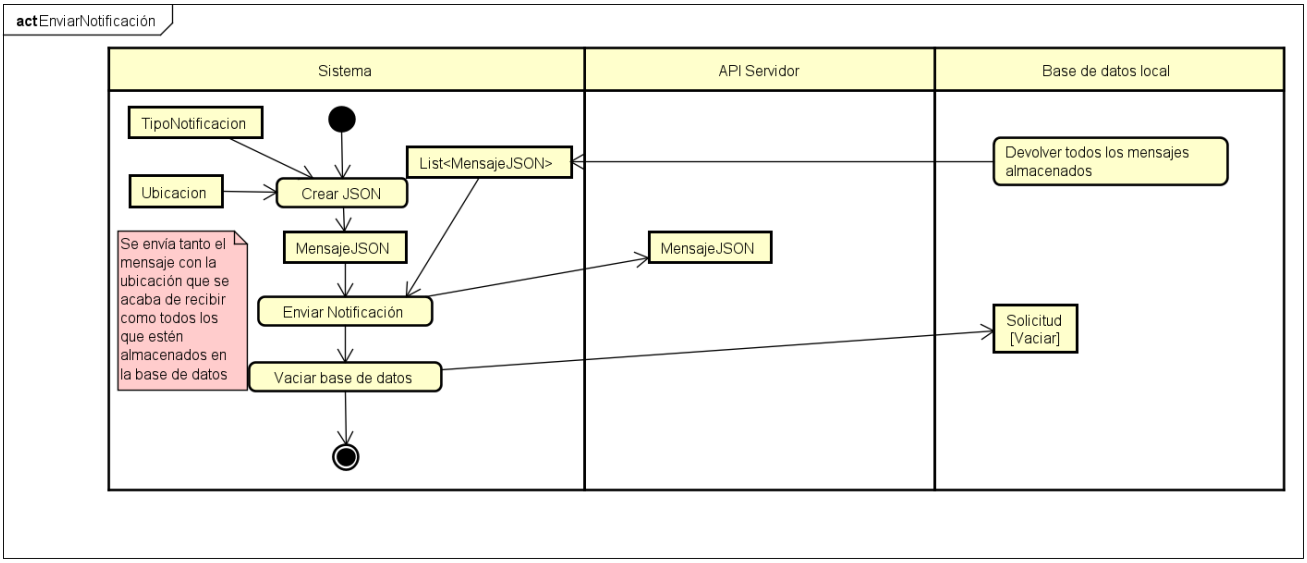

Figura 4.16: Diagrama de actividad correspondiente al CU07.

4.2. Estructura del proyecto

En esta sección se muestra la estructura final del proyecto Android. La figura 4.17 muestra la vista de proyecto Android disponible en Android Studio, se trata de una vista general de todos los componentes del proyecto, a continuación se desarrolla mas detalladamente el contenido de cada uno de ellos.

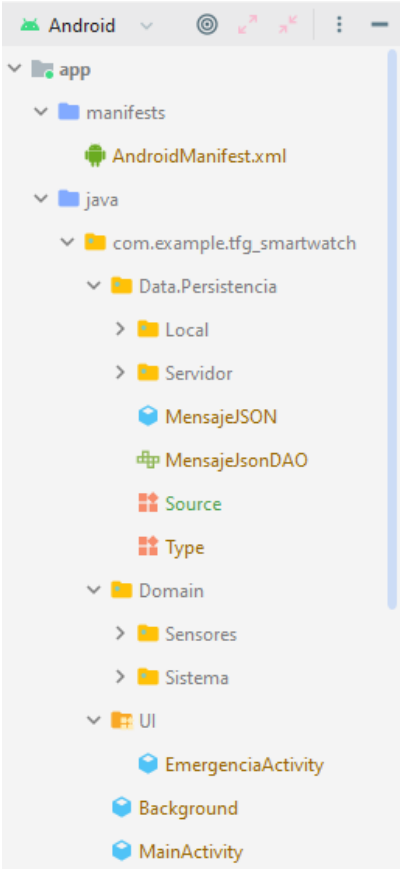

Figura 4.17: Estructura general del proyecto.

En primer lugar observamos externa a la carpeta del código Java una carpeta denominada *Manifests*. En su interior se encuentra el archivo manifiesto de la aplicación. En el se declaran tanto los permisos necesarios de la aplicación como componentes (Actividades, servicios o receptores) que contiene el proyecto.

Dentro de la carpeta del código Java encontramos dos clases dentro del paquete general. Por un lado, la clase *MainActivity* se trata de la actividad principal de cualquier aplicación Android. Esta clase se ejecuta al iniciar la aplicación y su función es la comprobar y solicitar permisos al usuario y con ellos desplegar la clase *Background* y pasar a segundo plano la aplicación.

Por otro lado, como el proyecto es necesario que se ejecute en segundo plano, se encuentra la clase Background. Esta clase extiende las funcionalidades de la clase *Service* de Android, por lo tanto es un servicio. Podemos considerar esta clase como la verdadera clase principal de la aplicación que nos hace de controlador entre la capa de datos y la capa de dominio. Las tres carpetas UI, Data y Domain y su contenido se explica más detalladamente en las siguientes secciones.

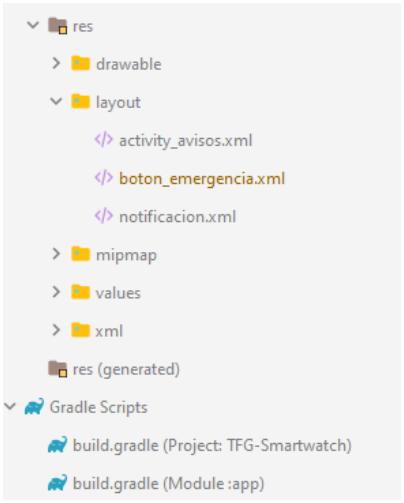

Figura 4.18: Estructura general del proyecto.

Por otro lado como se muestra en la figura 4.18 tenemos una carpeta *res* que contiene varias carpetas:

- **Drawable**: Contiene archivos XML con imágenes y descriptores de imágenes.
- **Layout**: Contiene archivos XML de las interfaces de usuario de la aplicación.
- **Mipmap**: Contiene ficheros XML con descriptores de imágenes.
- **Values**: Contiene ficheros XML con valores o constantes de la aplicación ya sea para determinados Strings, estilos o temas.

Además también se encuentran los archivos Gradle. Son archivos de configuración de la aplicación que contienen las dependencias, opciones de compilación y demás configuraciones. Para estructurar el proyecto se ha seguido la guía de arquitectura de apps de Android [31] y con ella se ha decidido escoger una arquitectura en tres capas.

4.2.1. Arquitectura en tres capas

Se ha seguido la guía de arquitectura de apps de Android [31] ya que incluye prácticas y recomendaciones para desarrollar una app de calidad y sólida.

La arquitectura recomendada en esta guía y la utilizada en este proyecto es la arquitectura de tres capas. Esta arquitectura consiste en organizar el proyecto en tres capas como se muestra en la figura 4.19. Siendo estas capas las siguientes:

- La **capa de usuario** o **capa UI** es la encargada de mostrar los datos de la aplicación por pantalla al usuario.

- La **capa de datos** contiene la lógica de datos de la aplicación, se refiere a almacenamientos, repositorios, etc.
- Una tercera capa, es opcional y no se incluye en todas las aplicaciones Android, se denomina **capa de dominio**. Esta capa se ubica entre la capa de datos y la de usuario encapsulando toda la lógica.

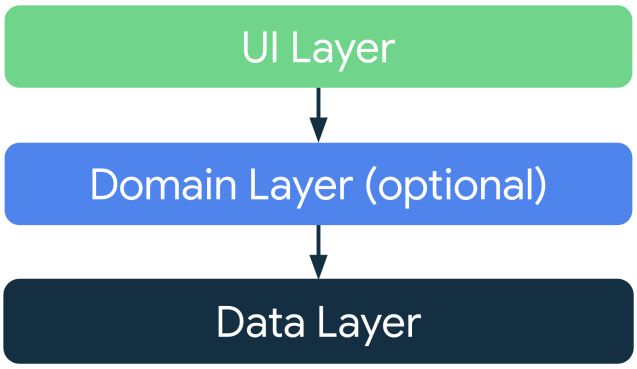

Figura 4.19: Arquitectura en 3 capas de aplicaciones Android [31].

4.2.2. Capa de usuario

La única función de esta capa es la de mostrar los datos de la aplicación al usuario, por ello, cuando estos datos cambian, esta también debe actualizarlos. En esta capa se incluyen tanto elementos de UI, por ejemplo las vistas, como contenedores estados, clases ViewModel, que contienen datos y se encargan tanto de mostrarlos al usuario como de realizar la lógica necesaria.

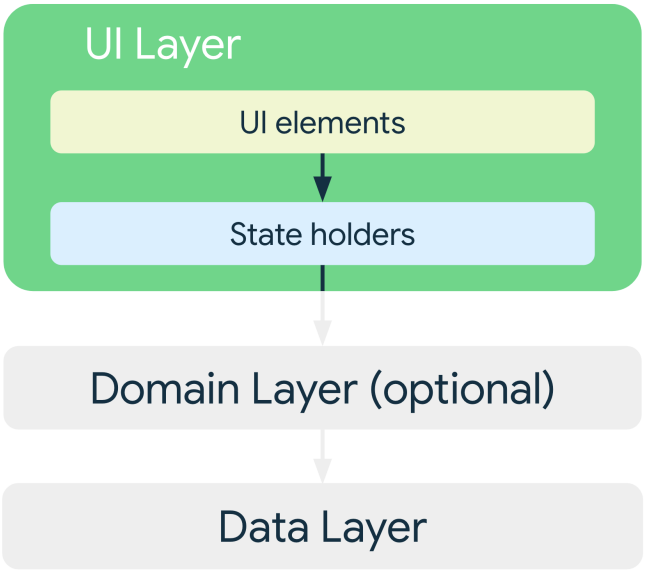

Figura 4.20: Capa de usuario en arquitectura de 3 capas de aplicaciones Android [31].

En cuanto a nuestro proyecto, como se observa en la figura 4.17, esta capa se representa en el paquete UI. Este paquete contiene únicamente la clase *EmergenciaActivity* que se encarga de mostrar la vista relacionada con el caso de uso de confirmar la emergencia.

Esta vista esta compuesta por un texto informativo y dos botones (Aceptar y Cancelar). Con el fin de evitar que esta acción se haya producido de forma accidental, el botón cancelar incluye un contador regresivo que se va mostrando en el propio botón, y cuando este llega a 0, se realiza su pulsación automática y se cancela la confirmación.Esta lógica también es implementada por la propia clase.

4.2.3. Capa de datos

Esta capa contiene la lógica de la aplicación y además las reglas de como la app crea, almacena y modifica los datos. Las clases que contiene esta capa son las denominadas repositorio. Estas clases se encargan de gestionar el almacenamiento de datos, centralizar los cambios de estos datos o exponerlos al resto de las clases.

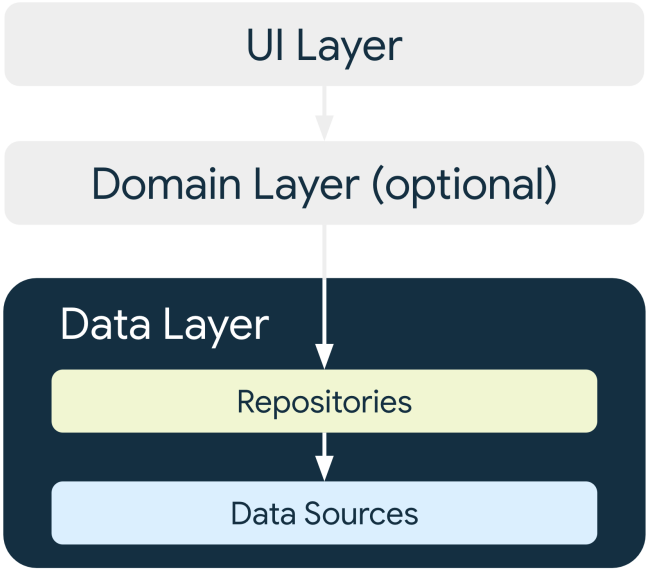

Figura 4.21: Capa de datos en arquitectura de 3 capas de aplicaciones Android [31].

En nuestro proyecto, como se observa en la figura 4.17, esta capa queda representada por el paquete *Data*. En primer lugar encontramos dos paquetes, Local y Servidor. El paquete local contiene la clase *DatabaseLocal* que extiende de la clase *RoomDatabase* y es la encargada de crear y configurar la base de datos local en el dispositivo gracias a esta librería de Android.

El paquete servidor incluye la clase *APIServidor*, encargada de realizar la conexión con el servidor, en concreto, realizar una petición HTTP de tipo POST añadiendo un JSON en el cuerpo de esta. Además, se ha incluido dentro de este paquete la clase *Configuracion*. Como se ha comentado anteriormente en este documento, esta clase incluye una serie de variables estáticas que pueden ser modificadas con el fin de editar los parámetros de uso de la aplicación. Se incluye dentro del paquete servidor debido a que la idea es que el valor de estas variables se descargue desde el servidor al iniciarse la app, si este tiene implementado esta funcionalidad en un futuro.

Las otras dos clases en esta capa son *MensajeJSON* que define la estructura del objeto JSON que se almacena (y envía al servidor) y su correspondiente clase DAO que permite realizar las operaciones con la base de datos local.

4.2.4. Capa de dominio

Esta capa es opcional y no tiene porque ser usada en todas las aplicaciones Android. Se sitúa entre la capa de usuario y la capa de datos y se encarga de encapsular la lógica compleja de la aplicación. Esta capa se usa cuando es imprescindible la reutilización de las clases y, aunque en nuestro proyecto se habría podido desarrollar tanto sin ella como con esta separación, se piensa en un mayor alcance futuro de la aplicación y se implementa con esta arquitectura.

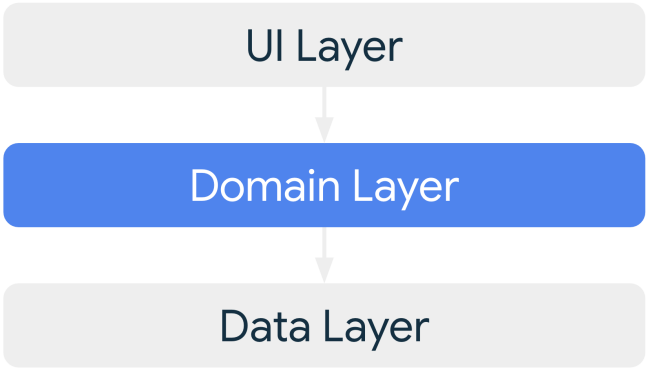

Figura 4.22: Capa de dominio en arquitectura de 3 capas de aplicaciones Android [31].

En nuestro proyecto, como se observa en la figura 4.17, esta capa queda representada por el paquete *Domain*. Este paquete contiene otros dos paquetes principales, *Sensores* y *Sistema*.

El paquete *Sensores* incluye toda la lógica relacionada con la obtención y procesado de los valores obtenidos de los distintos sensores. Cabe mencionar que se han dejado en el proyecto, aunque su funcionalidad no se utiliza, la clases realizadas para el sensor de frecuencia cardiaca y de luz ambiente, que fueron desarrolladas y se decidió no utilizarlas cuando se realizó las pruebas a la aplicación, por petición de los tutores del TFG pensando en que se siga trabajando en este proyecto en futuros trabajos.

Las clases que tienen funcionalidad son las correspondientes a *Acelerómetro*, *Proximidad* y *Ubicación*. Tanto la parte del acelerómetro como proximidad se componen de dos clases cada una:

- La clase principal del correspondiente sensor, cuya función es obtener con la clase *SensorManager* la instancia del sensor, y crear el propio listener y registrarlo para obtener sus valores cuando haya cambios.
- La clase del listener, estas clases extienden de *SensorEventListener*, y se encargan de realizar toda la lógica. Recibe los valores del sensor cuando hay una variación, los compara con el umbral especificado dentro del intervalo seleccionado y el numero de comprobaciones asignado, y determina si el reloj ha sido retirado realizando la llamada correspondiente a la clase *Background*.

La parte de ubicación tiene una única clase, el funcionamiento es similar a los dos anteriores pero a diferencia de los anteriores, para obtener la ubicación no es necesario apoyarse de un listener ya que las propias clases de Android (clase *LocationCallBack*) que se utilizan para esta parte tienen un propio bucle implícito que obtiene ubicaciones según el intervalo que se asigne y no cada vez que esta cambie.

El segundo paquete llamado *Sistema* dispone de tres clases, relacionadas con llamadas propias al sistema.

- La clase *PhoneClass* nos permite realizar llamadas utilizando la aplicación por defecto del dispositivo gracias a uno de los Intents comunes.

- La clase *ReceptorEmergencia* es la encargada de recibir la pulsación en el botón de emergencia de la notificación persistente y realiza la llamada a la clase *Background* para mostrar la vista correspondiente a la confirmación de emergencia.
- La clase *ReceptorEncendido* es un receptor (*BroadcastReceiver*) cuya funcionalidad es volver a iniciar la aplicación automáticamente cuando el reloj se enciende si la aplicación se estaba ejecutando cuando el dispositivo se apagó.

4.3. Diseño

Esta sección está dedicada a las técnicas y los patrones aplicados al proyecto. Se añaden los distintos diagramas de paquetes y despliegue.

4.3.1. Diseño de datos

En este proyecto encontramos una única entidad que se almacena en la base de datos que se corresponde con el mensaje JSON que se envía al servidor. En la figura 4.11 se incorpora el diagrama entidad-relación.

ID	INT
Type	CHAR
AndroidId	CHAR
Latitude	DOUBLE
Longitude	DOUBLE
Altitude	DOUBLE
Accuracy	DOUBLE
Battery	INT
Source	CHAR
DeviceTimeStamp	CHAR

Tabla 4.11: Diagrama entidad relación.

4.3.2. Patrones de diseño

Los patrones de diseño [32][33] son una serie de técnicas problemas comunes en el desarrollo de software, una solución a un problema de diseño. No cualquier solución es considerada un patrón de diseño, para ello debe ser efectiva dicho problema y reutilizable en otros similares. Podemos encontrar 3 tipos de patrones de diseño:

- *Patrones creacionales*, que resuelven los problemas respecto a la creación de instancias, de forma que este proceso de creación se pueda separar de la implementación del resto del sistema. Patrones como Singleton, MVC o Factory se incluyen en esta categoría.

- *Patrones estructurales*, que resuelven problemas de composición de clases y objetos, especifican como se relacionan unas clases con otras. Patrones como Adaptador, Compuesto o Fachada se incluyen en esta categoría.
- *Patrones de comportamiento*, esta categoría incluye a la mayoría de patrones. Resuelven problemas relacionados con la interacción y responsabilidades entre clases y objetos y sus algoritmos. Patrones como Iterador, Observador o Estrategia se incluyen en esta categoría-

A continuación se describen los patrones que se han implementado en este proyecto.

4.3.2.1. Patrón Observador

El patrón de diseño Observador [34] permite observar los cambios producidos por un objeto, cada cambio en su estado enviará una notificación a sus suscriptores. También es conocido esto como *Publisher-Subscriber (Publicador-Suscriptor)*. La figura 4.23 muestra como funciona.

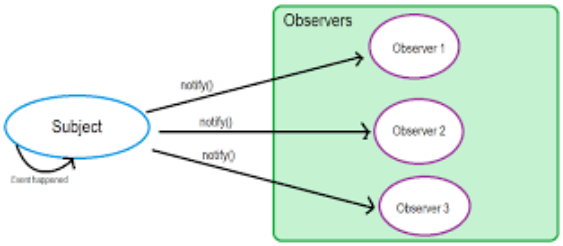

Figura 4.23: Esquema Patrón Observador [35].

En este proyecto, los sensores utilizados en el dispositivo implementan este patrón de manera implícita. Cuando se crea una instancia de la clase *SensorManager*, esta actúa como observable, mientras que las clases de los sensores que implementan la clase *SensorEventListener* actúan como observadores que se registran para recibir los cambios en los sensores. Cuando SensorManager detecta un cambio en un sensor, este notifica a los observadores registrados llamando a su método onSensorChanged.

La solución que proporciona este patrón al proyecto es que las actualizaciones de los valores de los sensores se reciben de manera automática sin necesidad de estar verificando los valores cada cierto tiempo.

4.3.2.2. Patrón DAO y Patrón DTO

El patrón *Data Access Object (DAO)* [36] propone separar la lógica de negocio de la lógica de acceso a la base de datos proporcionando los métodos necesarios para insertar, borrar, actualizar y obtener la información.

Este patrón suele ir acompañado la mayoría de las veces del patrón *Data Transfer Object (DTO)* [37]. Este patrón propone utilizar clases especiales para transmitir los datos en vez de utilizar las clases que se usan en el modelo de la aplicación. Con ello se tiene un control mayor de los datos que se envían.

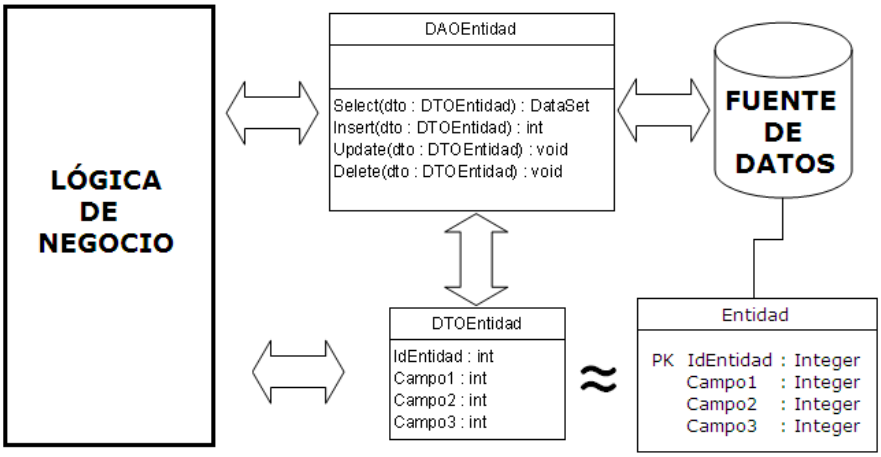

Figura 4.24: Esquema Patrón DAO y DTO [38].

En este proyecto, se implementan estos patrones, con la entidad *MensajeJSON*, que se almacena en la base de datos, se aplica el patrón DTO que permite poder enviar estos objetos, en este caso a una base de datos. La clase *MensajeJSONDAO* aplica el patrón DAO en este proyecto e incluye los métodos necesarios para realizar operaciones con la base de datos. En concreto, consta de 5 métodos: insertar, eliminar, actualizar y dos consultas, una obtiene una lista con todos los objetos MensajeJSON que contiene la base de datos, y otra obtiene un objeto MensajeJSON mediante su ID.

4.3.2.3. Patrón Singleton

El patrón de diseño Singleton [39] recibe este nombre debido a que solo puede tener una única instancia en toda la aplicación y un único punto de acceso a esta clase, con el fin de que exista una referencia global. Restringiendo la libre creación de instancias de esta clase.

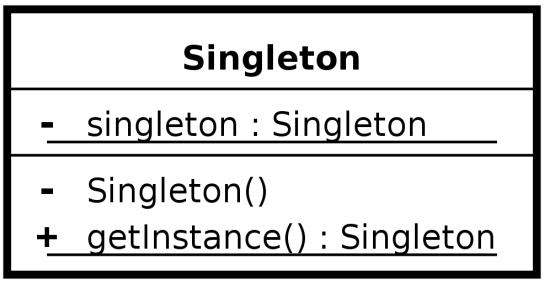

Figura 4.25: Esquema Patrón Singleton [40].

En este proyecto se aplica a la clase *Configuracion*. Al tratarse de una clase con una serie de parámetros declarados con variables estáticas y que se desea tanto que sean únicos para toda la aplicación como que cualquier cambio en ellos pueda ser detectado por todas las clases del proyecto. Al haber una única instancia de esta clase en toda la aplicación nos proporciona la solución necesaria.

4.3.2.4. Patrón Factory

El patrón de diseño Factory [41] permite la creación de objetos de un subtipo determinado a partir de una superclase. Se usa este patrón cuando en tiempo de diseño no se sabe el subtipo que se va a utilizar y por lo tanto estas instancias se crean dinámicamente en tiempo de ejecución.

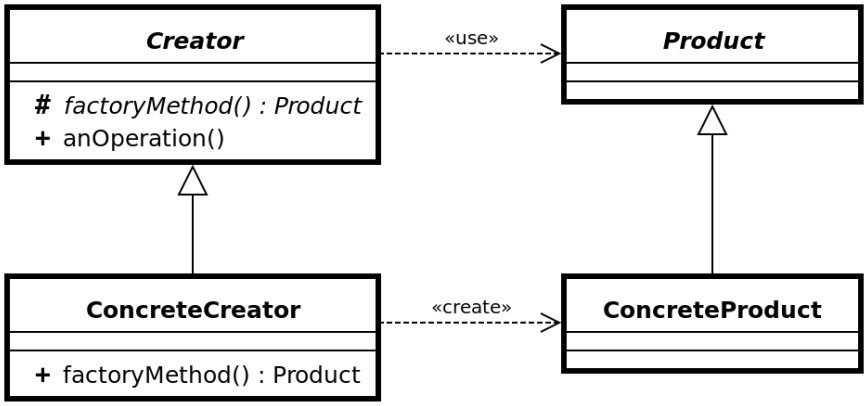

Figura 4.26: Esquema Patrón Factory [42].

En este proyecto, este patrón también está implícito en la clase *LocationManager* de Android. Esta clase proporciona los proveedores de ubicación, que pueden ser de distintos tipos como Wifi o GPS según la disponibilidad del dispositivo. *LocationManager* se encarga de crear instancias de su subclase *FusedLocationProvider*, que presenta una interfaz común para todos los proveedores. La creación de estas subclases con distintos parámetros como la prioridad o la precisión de la ubicación se puede llevar a cabo gracias a la aplicación de este patrón.

4.3.3. Diagrama de paquetes

Un diagrama de paquetes [43] es un tipo de diagrama estructural que se emplea para mostrar la organización de los distintos elementos de un sistema en forma de paquetes. Representa una organización jerárquica de la aplicación. En la figura 4.27 se muestra el diagrama de paquetes de este proyecto.

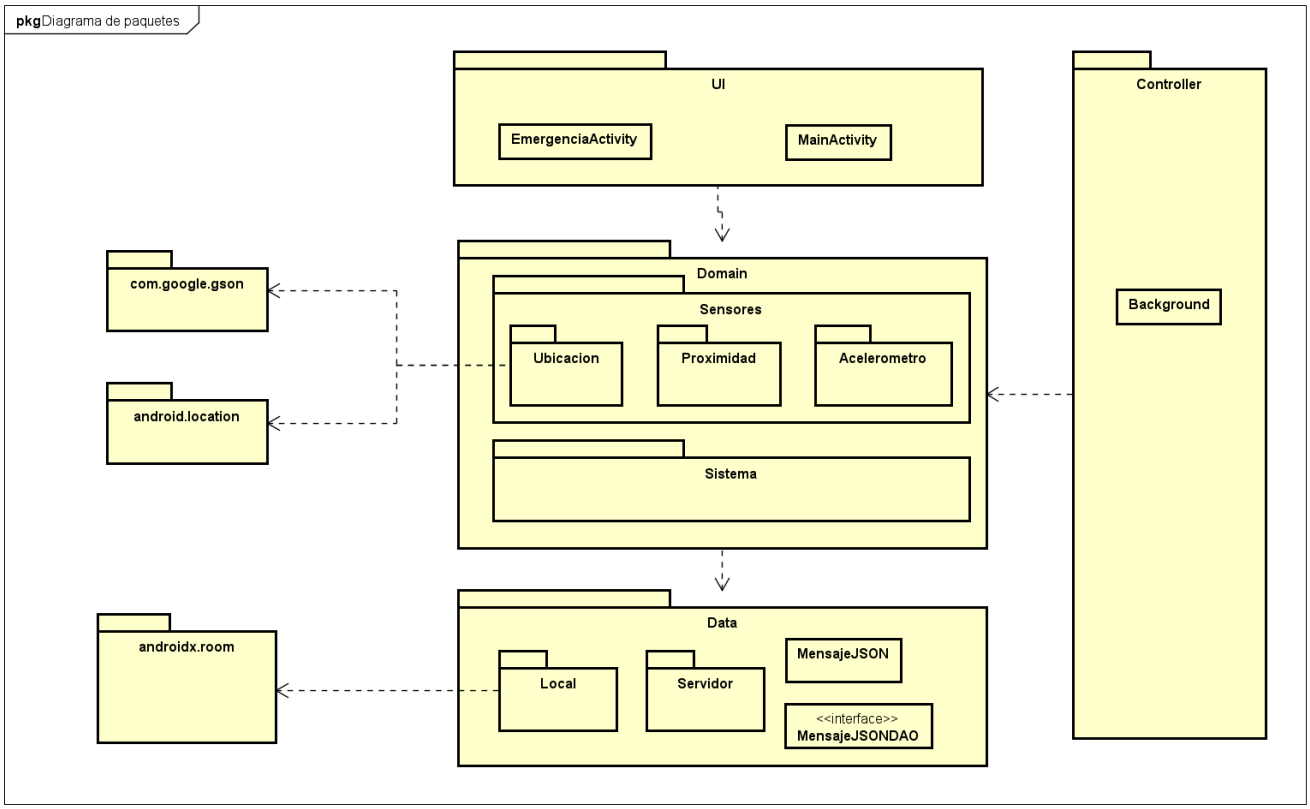

Figura 4.27: Arquitectura de la aplicación.

Se puede observar la clara división en tres capa de la arquitectura de la aplicación. Se representa mediante los paquetes *UI*, *Domain* y *Data*. Además se observa un cuarto paquete que representa las clases que actúan como controlador en el sistema.

El paquete *Controller* incluye tanto la clase *MainActivity* como la clase *Background*, ambas encargadas de actuar como intermediarias entre el usuario y el sistema.

El paquete *UI* incluye las vistas de la aplicación que se muestran al usuario.

El paquete *Domain*, ver figura 4.28, incluye dos paquetes:

- *Sensores*, incluye la lógica relacionada con los sensores del dispositivo.
- *Sistema*, incluye la lógica relacionada con servicios necesarios del sistema. Estos son la realización de un llamada, y dos receptores para detectar el encendido y la pulsación en el botón de emergencia.

El paquete *Data*, ver figura 4.29, incluye la representación de los datos de la aplicación, en este caso, el mensaje JSON que se envía junto a su correspondiente DAO. Además incluye dos paquetes:

- *Local*, incluye la interfaz necesaria para la configuración de la base de datos local con Room.
- *Servidor*, incluye la clase *Configuracion*, que permite establecer los parámetros necesarios de

la aplicación, junto con la clase *APIServidor* que permite realizar peticiones HTTP a la API del servidor.

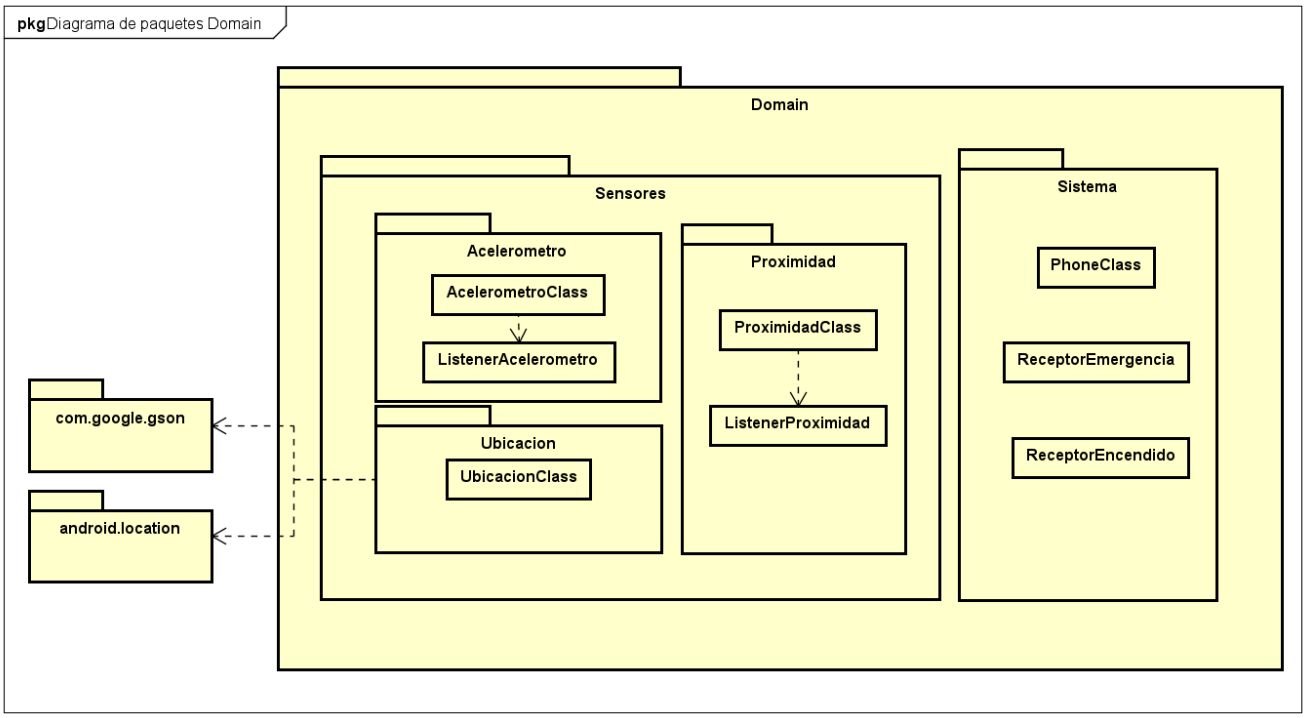

Figura 4.28: Diagrama de paquetes de Domain.

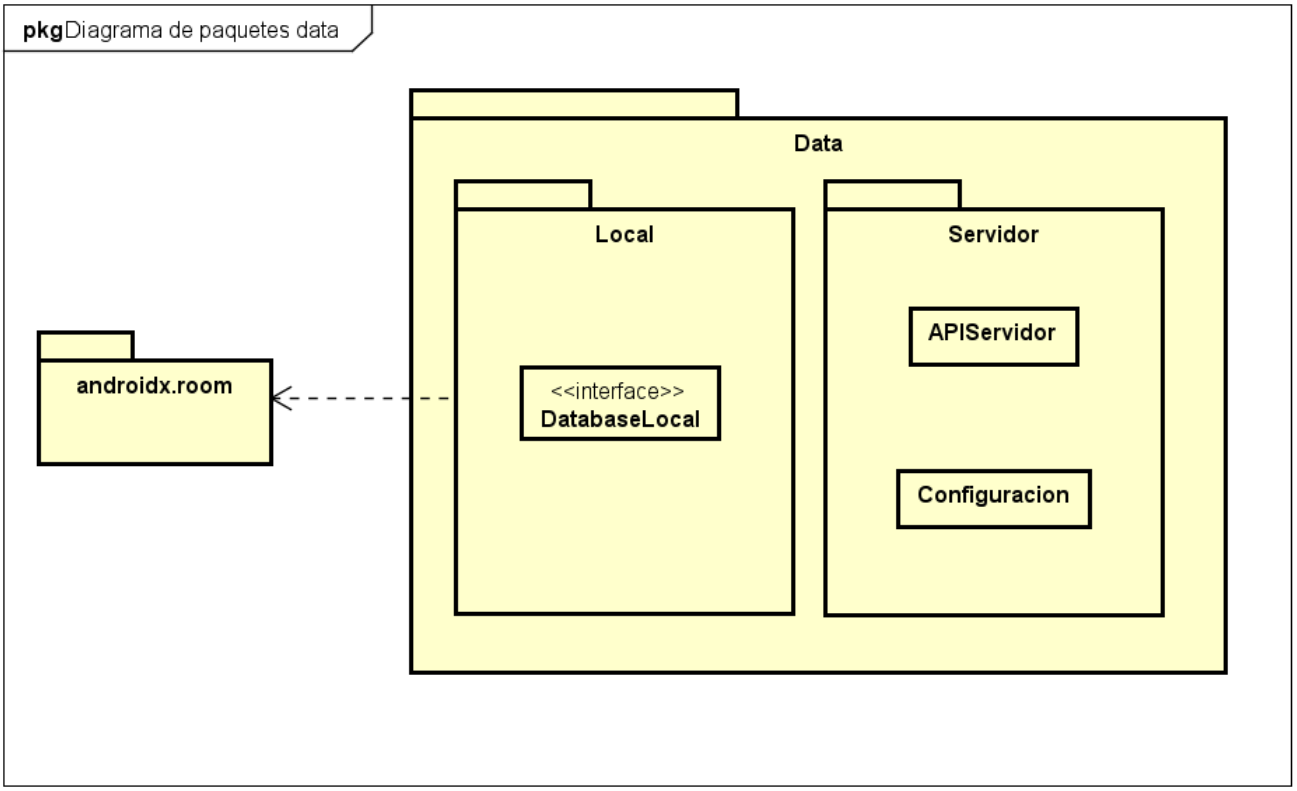

Figura 4.29: Diagrama de paquetes de Data.

4.3.4. Diagrama de despliegue

Un diagrama de despliegue [44] es un tipo de diagrama que muestra la arquitectura de ejecución de un sistema. Incluyendo sus principales nodos tanto hardware como software y como se organizan y conectan entre ellos. Este tipo de diagramas permiten entender como se despliegue el sistema físicamente en el hardware.

En el caso de este proyecto, en la figura 4.30 se puede observar como existe un dispositivo donde se lleva a cabo el despliegue de la aplicación que sera un Smartwatch, contiene un entorno de ejecución Android (Wear OS) donde se ejecuta la app y se conecta con una base de datos local desarrollada con la librería Room de Android. Este realiza una conexión segura mediante el protocolo https con un servidor cuya estructura interna queda fuera de este proyecto.

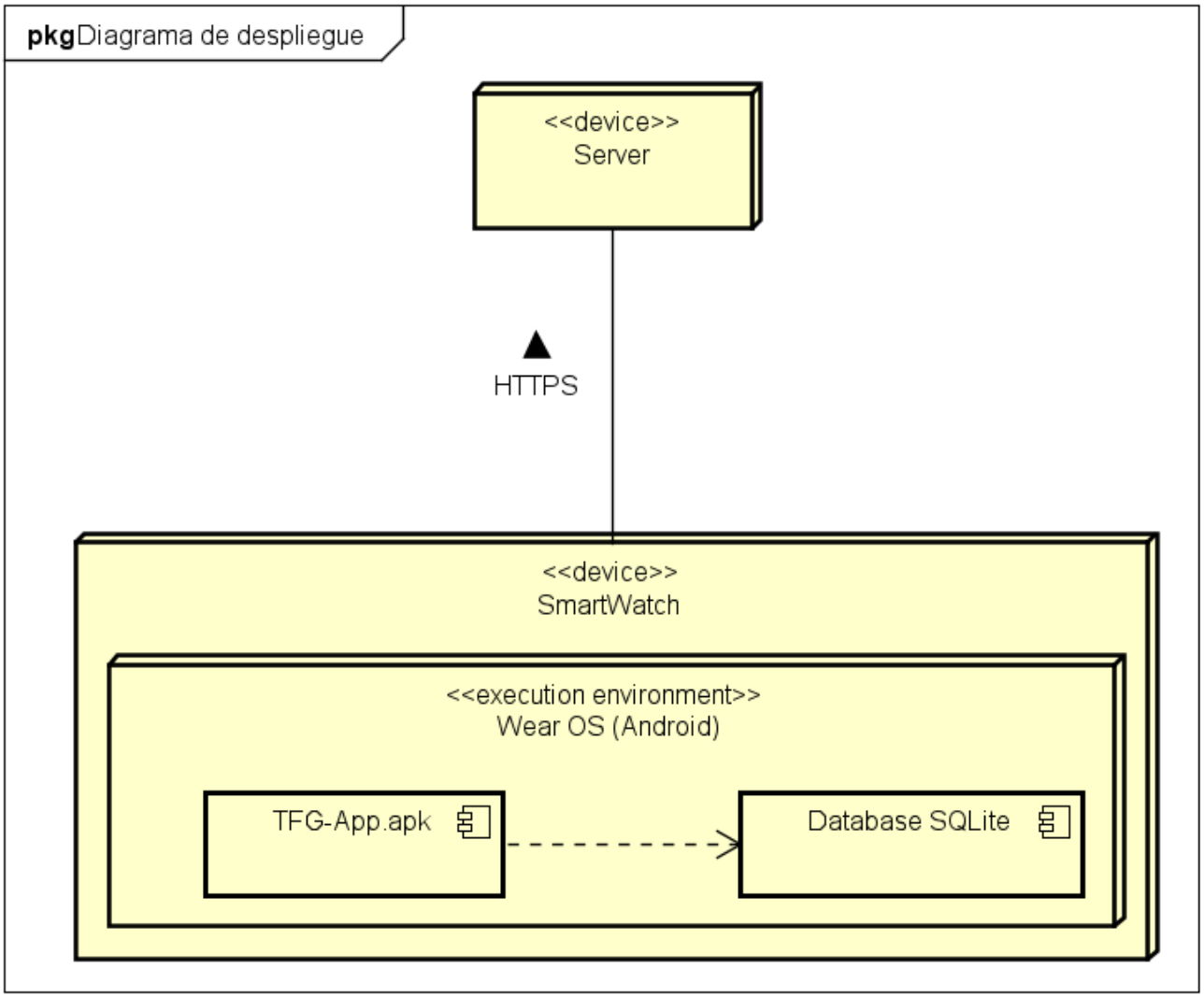

Figura 4.30: Diagrama de despliegue de la aplicación.

Capítulo 5

Implementación

En este capítulo se detalla como han sido implementadas cada una de las funcionalidades del sistema. Se comentará de manera más técnica como se realiza la monitorización de la ubicación y su envío al servidor, la detección de la retirada del dispositivo, el caso de notificación de emergencia y la ejecución en segundo plano de la aplicación. Además, se incluye una sección final en la que se exponen las principales diferencias entre el desarrollo en una aplicación Android general y una aplicación Android para Wear Os.

5.1. Permisos del usuario

Para la utilización de los distintos servicios y sensores del dispositivo es necesario solicitar permisos al usuario e incorporarlos al archivo Manifest[45]. Este archivo, que se encuentra presente en todas las aplicaciones Android, proporciona información esencial de la app a las herramientas de creación de Android o al sistema operativo. Incluye el nombre del paquete, permisos y componentes de la app entre otros.

Para el caso de la ubicación tendremos dos tipos de permisos disponibles: Uno que nos proporciona ubicaciones exactas mientras que otro nos proporciona una ubicación aproximada, en nuestro proyecto se priorizará la exactitud. Además, debido al alcance del proyecto, es necesario el permiso para obtener la ubicación del dispositivo en segundo plano.

5.2. Sensores

En primer lugar, comentar que se procede a usar en el proyecto la clase de Android SensorManager [46]. Esta clase nos permite acceder a todos los sensores del dispositivo, gracias a su método *getDefaultSensor(int type)* siendo type el número del sensor, y a partir de la utilización de listeners que implementan la interfaz SensorEventListener [47] se obtienen los datos de los sensores hardware ne-

cesarios gracias a su método *onSensorChanged()* que se ejecuta cada vez que se obtiene un nuevo valor de ese sensor.

El otro método más importante de la clase *SensorManager* es el método *registerListener()*. Este método nos permite registrar un listener de los comentados anteriormente para comenzar con la monitorización. Recibe 3 parámetros de configuración, el propio listener, el sensor que se va a monitorizar y un periodo de monitorización. Este periodo se define mediante 4 variables predefinidas aunque se puede expresar con un número en microsegundos:

- **SENSOR_DELAY_FASTEST** Para obtener los datos lo más rápido posible.
- **SENSOR_DELAY_GAME** Es la tasa recomendada para juegos.
- **SENSOR_DELAY_NORMAL** Es la tasa predeterminada.
- **SENSOR_DELAY_UI** Es la tasa recomendada para la interfaz de usuario.

5.3. Monitorización de la ubicación

La primera funcionalidad que se desarrolla es la monitorización de la ubicación del dispositivo. Para ello se ha decidido usar la API *FusedLocationProvider* que ha sido desarrollada por Google y es la más común cuando se trata de aplicaciones de estas características. Esta API utilizada tanto el GPS como la conexión a internet para utilizar el mejor proveedor para la ubicación de forma automática.

Tras obtener el proveedor, se utiliza otra API de Google *LocationRequest* que nos permite crear un monitorizador de la ubicación agregando un intervalo en el cual queremos recibir actualizaciones y una prioridad, la combinación de estos valores es importante para el consumo de batería, y se comentará en la sección 6 con las pruebas realizadas. Con todo esto se creará un bucle que devolverá la ubicación dentro del intervalo establecido.

5.4. Procesamiento de la ubicación

Tras obtener la ubicación, es necesario procesarla. Se tiene dos opciones, si hay conexión a internet esta ubicación se enviará a un servidor, sino se almacenará en una base de datos local hasta que se consiga una conexión. Para el envío se ha realizado una clase que realiza una petición HTTP de tipo POST añadiendo un JSON con la ubicación en el cuerpo de esta. El formato del JSON a enviar es el siguiente:

```
{
"type":"string"
"androidId": "string",
"latitude": 0,
```

```
"longitude": 0,
"altitude": 0,
"accuracy": 0,
"battery": 0,
"source": "string",
"deviceTimestamp": "string",
}
```

En el JSON que se envía nos encontramos los siguientes campos:

- **TYPE**: String que define el tipo de notificación que se envía: Debe corresponder a uno de los siguientes valores: *MONITORING*, *TAKEOFF*, *EMERGENCY*. El primero indica que es una notificación de monitorización normal, el segundo que es por retirada y el tercero si se envía al accionar el botón de emergencia.
- **ANDROIDID**: Se corresponde con el id de Android. Es una cadena de 64 bits alfanumérica única por dispositivo, este id puede cambiar si se restablecen los valores de fábrica del dispositivo o si cambia la clave de firma APK, como por ejemplo si se actualiza el sistema operativo. [48]
- **LATITUDE**: Double que se extrae de la medición de la ubicación y se corresponde con la latitud de las coordenadas.
- **LONGITUDE**: Double que se extrae de la medición de la ubicación y se corresponde con la longitud de las coordenadas.
- **ALTITUDE**: Double que se extrae de la medición de la ubicación y se corresponde con la altitud de las coordenadas.
- **ACCURACY**: Float que se corresponde con el radio de precisión horizontal de las coordenadas obtenidas expresado en metros.
- **BATTERY**: Entero que expresa el porcentaje de batería del dispositivo en el momento de la obtención de la ubicación.
- **SOURCE**: String que define el proveedor del cual se ha obtenido la ubicación. Debe corresponder a uno de los siguientes valores: *GNSS*, *CELLULARNETWORK* o *FUSED*, según el proveedor de ubicación sea del propio GPS, de la red móvil o si se trata de una ubicación fusionada. Este último es el más común ya que el proveedor que se intenta utilizar si se puede es de este tipo, que combina varios proveedores de ubicación.
- **DEVICETIMESTAMP**: String en formato fecha UTC que indica la fecha y hora del dispositivo en el momento de recogida de las coordenadas.

La base de datos almacenará este JSON si no es posible realizar el envío. Esta base de datos se ha decidido finalmente, tras explorar varias opciones, escoger la opción de Room. Room es una biblioteca que nos proporciona toda a funcionalidad de SQLite. Cuando se recupera conexión, se recuperan todos los JSON almacenados, se envían al servidor y la base de datos se vacía.

5.5. Detección de la retirada del dispositivo

Para el desarrollo de esta funcionalidad, como ya se ha comentado es necesario el apoyo de distintos sensores del dispositivo, en concreto del acelerómetro y del sensor de proximidad. Para poder obtener los valores obtenidos por este sensor es necesario la utilización de las clases comentadas en el apartado 5.1 de este capítulo.

Cabe mencionar que para detectar una retirada, se basará todo el proyecto en umbrales que podrán ser modificados por el administrador del centro de control en función de las necesidades de ese paciente. Para el sensor de proximidad el cálculo, extraído del código desarrollado para este proyecto según se encuentra durante el desarrollo de esta iteración, será el siguiente:

```
if (distancia > UMBRAL){
        //Se ha detectado retirada, gestionar como corresponda
   }
```

Siendo la distancia el valor obtenido del sensor y el umbral ya comentado configurable según la necesidad. Por defecto se configurará tras la realización de las pruebas necesarias, sección 6.

Para el acelerómetro el cálculo, extraído del código desarrollado para este proyecto según se encuentra durante el desarrollo de esta iteración, será el siguiente:

```
float calculo = Math.abs(valoresActuales[0] -    valoresAnteriores[0]) +
Math.abs(valoresActuales[1] -
valoresAnteriores[1]) +
Math.abs(valoresActuales[2] -
valoresAnteriores[2]);

if(calculo < UMBRAL_VALORES_IGUALES){
        //Se ha detectado retirada, gestionar como corresponda
}
```

En este caso, se ha desarrollado un algoritmo que compara los valores obtenidos con los obtenidos en la medición inmediatamente anterior. Estas dos mediciones se encuentran separadas por un intervalo de tiempo configurable en función de la robustez que se le quiera dar según el portador.

Se realiza una resta de estas dos mediciones por cada eje, y se suma estos resultados con el fin de comprobar si ambas mediciones son significativamente diferentes. Este cálculo se compara con un umbral configurable que proporciona una sensibilidad. Cuando más cercano a 0 sea este umbral, mayor sensibilidad se tendrá en cuanto a mínimos movimientos.

Además se ha procurado potenciar este algoritmo con el fin de evitar falsos positivos añadiendo un número configurable de repeticiones del algoritmo completo antes de notificar la retirada.

5.6. Notificación de emergencia

En primer lugar se realiza una vista que se mostrará por pantalla al activar el botón, todo ello con el fin de realizar una doble comprobación como se especificó en los requisitos. La vista consta de un cuadro texto, en el que se informa al usuario que se va a proceder a notificar la emergencia, junto con dos botones para aceptar o cancelar. Cabe mencionar que para el botón de cancelar se ejecuta un contador de 10 segundos que tras su finalización, si el usuario no ha pulsado ninguno de los botones, se cancela el aviso.

Para acceder a esta vista se decide hacer uso de la propia notificación persistente. Al pulsar sobre esta, se despliega mostrando el Android ID, y así ser visualizado por los responsables para su registro en el servidor (ya que Android no muestra este ID en sus ajustes), junto con un botón con el texto *EMERGENCIA*. Cuando el usuario acciona este botón se desencadena en la vista de confirmación mencionada anteriormente.

Tras la confirmación de la emergencia por parte del usuario, se envía una notificación al servidor gestionado por el centro de control y además se deberá enviar el audio que recoge el dispositivo en tiempo real.

Tras estudiar la opción principal que se especificó en los requisitos siendo esta el uso de aplicaciones externas como Whatsapp para abrir este canal de comunicación, finalmente se decide utilizar uno de los Intents comunes [49]. Un Intent[50] en Android es un objeto de mensajería que se usa para solicitar una acción de otro componente en una app. Entre los denominados Intents comunes se encuentran algunos que nos sirven para gestionar alarmas o temporizadores, acciones de la aplicación cámara y muchos más.

En este proyecto el que realmente nos importa es el que permite realizar acciones con la aplicación predeterminada de teléfono del dispositivo. Este intent nos proporciona dos opciones: la primera abrir la aplicación del teléfono con el teclado de marcaje y un teléfono marcado por defecto y la segunda, que será la utilizada en el proyecto, que permite realizar directamente una llamada sin necesidad de confirmar, aunque es necesario pedir permiso al usuario la primera vez que se inicia la app.

Usando esta opción mucho más sencilla nos permite abrir un canal de comunicación bidireccional con el centro de control, tanto en dispositivos independientes con SIM propia como en aquellos que dependen de estar conectados a un teléfono móvil.

5.7. Ejecución en segundo plano

En primer lugar, la actividad principal que se ejecuta al iniciar la app con dos únicas funciones: Por un lado, solicitar los permisos necesarios al usuario, una vez que se acepten, se almacena esta configuración en el dispositivo y no será necesario en el próximo arranque y por otro lado, dar paso a la ejecución de la aplicación en segundo plano. Esto se ha realizado de la siguiente manera:

- Desde la actividad principal se inicia un servicio que ejecuta toda la lógica de aplicación que pasará a primer plano, esto hace que la aplicación pase a segundo plano.
- Lo primero que realiza el servicio es mostrar una notificación persistente al usuario para indicar que se está ejecutando la aplicación en segundo plano y tras esto se ejecutará con normalidad pudiendo ser utilizado el resto del dispositivo mientras la app sigue ejecutándose.

5.8. Diferencias entre desarrollo para Android para dispositivos móviles y dispositivos Wear OS

En esta última sección del capítulo se considera importante comentar las principales diferencias a la hora de desarrollar aplicaciones Android para dispositivos con Wear OS respecto a aplicaciones Android para el resto de dispositivos móviles.

Como ya se ha comentado en otras secciones de esta memoria, Wear OS está basado en Android, por lo que la mayoría del desarrollo es similar en cuanto a APIs o funcionalidades pero se encuentras algunas diferencias [51][52] en distintos aspectos que se comentan a continuación:

- **Diseño de la UI**: El desarrollo es diferente, principalmente por el tipo de pantallas que utilizan los dispositivos con Wear OS. Se recomienda enfocarse simplemente en desarrollar tareas esenciales que puedan ser realizadas en poco tiempo.
- **Componentes de la UI**: Para el desarrollo en Wear OS se incluyen componentes específicos para las vistas, por ejemplo *WearableRecyclerView* que es una implementación específica de este tipo de vista.
- **Modo oscuro**: En Wear OS solo se permite usar este modo para el ahorro de batería y no a gusto del usuario.
- **Botones físicos**: Los dispositivos con Wear OS incluyen como mínimo un botón físico para su encendido, la mayoría tienen al menos otro botón multifunción pero no siempre se da el caso. Se debe tener en cuenta esto a la hora de implementar una aplicación en la que sea necesario su uso.
- **Navegación**: La navegación es diferente por ello las apps deben ser superficiales y lineales.
- **Interacción con dispositivos vinculados**: Un aspecto importante a tener en cuenta es que estos dispositivos pueden interactuar con aquellos a los que quedan vinculados como puede ser un teléfono. Esto se debe analizar a la hora de desarrollar ya que se pueden obtener datos o realizar alguna funcionalidad con el dispositivo.
- **Servicios de salud**: Estos dispositivos con Wear OS son capaces de obtener datos de salud del portador del reloj a través de sus sensores. Para algunas aplicaciones como la desarrollada en este proyecto esta parte es bastante importante.

Estas son solo algunas de las principales diferencias, podemos encontrar también diferencias específicas en cada API que se utilice durante el propio desarrollo de la aplicación que se deberá consultar en la documentación correspondiente.

Capítulo 6

Pruebas

Las pruebas de software [53] [54] son investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva sobre la calidad del producto a la parte interesada. Existen distintos tipos de pruebas software en función del objetivo y las necesidades del proyecto.

- Pruebas unitarias: Comprueban cada parte del software en el que se está trabajando. Son las primeras que deben realizarse durante todo el desarrollo del proyecto.
- Pruebas de integración: Sirven para comprobar que los distintos componentes del proyecto operan bien juntos. Es un proceso más complejo que debe realizarse después de las pruebas unitarias.
- Pruebas funcionales: También denominadas pruebas de caja negra. Sirven para comprobar las funciones del software y establecer la usabilidad y las características de cara al mercado.
- Pruebas de aceptación: El equipo responsable debe definir cuales son los criterios de aceptación. Sirven para corroborar que el proyecto funciona de manera esperada. Se prueba el software por el equipo o incluso el cliente y se verifica que cumple con las expectativas.
- Pruebas de estrés: Comprueban el punto exacto en el que el sistema se satura.
- Pruebas de rendimiento: Relacionadas directamente con la experiencia del usuario, definen cuestiones como la velocidad o la estabilidad.
- Pruebas de regresión: Comprueban que los cambios en un componente software no provocan fallos en otros elementos.

La realización de pruebas de cualquier proyecto software es un trabajo extenso si se desean realizar unas pruebas de calidad. Por un lado, debido a que el tiempo de dedicación a esta parte se estimó en 3 semanas y junto con la redacción de la memoria del trabajo en paralelo.

Y por otro lado, según las funcionalidad de la aplicación que recoge valores de los distintos sensores, esto dificulta saber a priori cuales son los datos que se desean obtener en esta funcionalidad. Es por

estas razones por las que se decide junto con los tutores realizar únicamente pruebas de aceptación a la aplicación. Se detallan a continuación las pruebas que se han realizado en este proyecto.

6.1. Pruebas unitarias

Aunque este tipo de pruebas, como ya se ha comentado, no han sido realizadas en este proyecto, se detalla a continuación como sería la realización de este tipo de test.

Al haberse desarrollado la aplicación con Java la mejor opción es usar JUnit. JUnit es un conjunto de bibliotecas que permite realizar este tipo de pruebas en Java, además de su ejecución de una manera controlada para cada prueba. Es la mejor opción ya que Android Studio integra este framework y además ya se tiene experiencia con estos test.

El funcionamiento es bastante sencillo, se elabora la prueba de una clase o función de esta con el valor que se espera que sea retornado, si es así, JUnit finalizará exitosamente, en caso contrario, JUnit comunicará un fallo. En Android Studio el proceso de creación de test es el siguiente:

1. En la pestaña Code, al desplegar pulsamos en Generate... y Test...
2. Se abre una nueva pestaña donde se elige qué métodos se desea probar.
3. Tras esto, se crea una clase donde se pueden desarrollar los test que se deseen.

```
1  package com.example.tfg_smartwatch;
2
3  import junit.framework.TestCase;
4
5  public class BackgroundTest extends TestCase {
6
7      public void testGetBattery() {
8          Background bg = new Background();
9          assertEquals(bg.getBattery(bg), actual: 80);
10     }
11 }
```

Figura 6.1: Código ejemplo Test JUnit Android Studio.

En la figura 6.1 se muestra un ejemplo sencillo de como son estos test. En este caso se quiere probar el método *getBattery* de la clase *Background*. Este método se encarga de obtener la batería del dispositivo en ese momento.

En primer lugar, se deben crear los objetos de las clases que necesitamos, tras esto usamos las funciones de JUnit, en este caso *assertEquals* que comprueba que el valor esperado y el retornado por el método son el mismo. Se debe realizar un test JUnit para cada caso de prueba que se desee probar, asi como de cada método.

6.2. Pruebas para calibrar umbrales

Como se comentó anteriormente, la detección de la retirada del dispositivo se comprueba mediante la realización de diferencias significativas entre dos medidas inmediatamente posteriores y esta diferencia se comprueba con un umbral.

Es muy importante ajustar correctamente estos umbrales, ya que, aunque el ideal es que esta diferencia fuera 0 lo cual nos indicaría que las dos mediciones son exactas y por lo tanto el dispositivo no ha detectado movimiento, se debe tener en cuenta que al tratarse de sensores siempre puede haber un margen de error ya que son muy sensibles. Para esto es necesario observar el comportamiento y los valores que devuelve en situaciones normales de uso para ajustar lo máximo posible estos umbrales y evitar así falsos positivos.

Estas pruebas han sido realizadas nada más desarrollar la funcionalidad del sensor correspondiente. Cabe mencionar que no se han podido realizar pruebas en cuanto al sensor de proximidad ya que no se dispone de ningún dispositivo con este sensor disponible en este proyecto. A continuación se exponen algunos de los distintos tipos de pruebas que se han realizado para cada sensor.

6.2.1. Sensor acelerómetro

Algunas de las pruebas realizadas para el sensor del acelerómetro se exponen a continuación.

Nombre	Valores devueltos.
Descripción	Se comprueba que valores se obtienen del sensor del acelerómetro, realizando movimientos del dispositivo para ver como cambian estos y en que rango se encuentran.
Resultado obtenido	- Valores muy variados con pequeños movimientos. - Cuando el dispositivo esta sobre una superficie sin ningún movimiento los valores son muy similares, con pequeñas diferencias de 0.05-0.1
Conclusión tras la prueba	Se corrobora que el acelerómetro es un buen sensor para detectar una retirada cuando el dispositivo se deja sobre una superficie. Al tratarse de un sensor muy sensible, detecta mínimos movimientos como puede llegar a ser el propio pulso de la persona. El umbral se deja ajustado en 0.2. Se deben realizar varias mediciones para evitar falsos positivos si se deja la muñeca quieta completamente.

Tabla 6.1: Prueba 1 acelerómetro.

Nombre	Comprobación algoritmo desarrollado.
Descripción	De la prueba anterior se concluye que es necesario hacer varias mediciones por ello se lleva a cabo el algoritmo explicado en secciones anteriores. Es esta prueba se verifica que realizando varias mediciones en intervalos de tiempo considerables se obtiene una funcionalidad más robusta.
Resultado obtenido	- Aplicando varias mediciones en intervalos de 1-2 minutos no se producen falsos positivos de retirada si la persona no proporciona movimiento al dispositivo. - Aplicando varias mediciones en intervalos mayores no se producen falsos positivos de retirada si la persona no proporciona movimiento al dispositivo.
Conclusión tras la prueba	Con este algoritmo se le proporciona robustez a la detección de la retirada. Se han realizado pruebas de unas 10 comprobaciones en intervalos de 1 minuto y unas 3 comprobaciones en intervalos de 5 minutos y ambos umbrales funcionan adecuadamente ya que es prácticamente imposible que tras esos intervalos no se haya producido ninguna pequeña modificación en el dispositivo. Los parámetros se dejan fijados en 5 comprobaciones en intervalos de 1 minuto.

Tabla 6.2: Prueba 2 acelerómetro.

6.2.2. Sensor frecuencia cardiaca

Algunas pruebas realizadas para el sensor de frecuencia cardiaca se exponen a continuación.

Nombre	Valores devueltos.
Descripción	Se comprueba que valores se obtienen del sensor de frecuencia cardiaca, realizando retiradas del dispositivo y comprobando valores devueltos.
Resultado obtenido	- Valores obtenidos con el dispositivo retirado similares a valores que se obtienen en una persona en determinadas actividades de la vida cotidiana. - Con dispositivo retirado y reflejando el sensor en una superficie se obtienen valores de frecuencia bajos 70-80. - Con dispositivo retirado sin reflejar en una superficie directamente se obtienen valores elevados 150-160.
Conclusión tras la prueba	Al tratarse de un sensor que se basa en el reflejo de un haz de luz obtenemos valores aún cuando el dispositivo no está en la muñeca. Con estos valores no se llegan a encontrar patrones claros para detectar una retirada. Es complicado determinar si estos valores se deben a la realización de actividades como realizar deporte o dormir. Se determina que el sensor de frecuencia cardiaca no sirve de apoyo a esta funcionalidad.

Tabla 6.3: Prueba 1 frecuencia cardiaca.

6.2.3. Sensor luz ambiente

Algunas pruebas realizadas para el sensor de luz ambiente se exponen a continuación.

Nombre	Valores devueltos.
Descripción	Se comprueba que valores se obtienen del sensor de luz ambiente, realizando movimientos del dispositivo, así como con el dispositivo retirado.
Resultado obtenido	- Valores obtenidos muy variados ya que mide el grado de reflejo de la luz en el sensor. - Grandes variaciones a cambios mínimos de luz.
Conclusión tras la prueba	Se observa un sensor muy sensible a cambios. Cuando se esta portando el reloj en la muñeca, los valores que se obtienen son muy variados, con el reloj retirado son valores muy similares, es por ello que se decide incorporarlo en un primer momento como apoyo al sensor del acelerómetro para dar robustez al algoritmo de detección de retirada.

Tabla 6.4: Prueba 1 sensor luz ambiente.

6.2.4. Conclusiones de las pruebas realizadas y estado de la aplicación tras ellas

Tras la realización de estas pruebas comentadas anteriormente, cuyo fin de ajustar los umbrales que permiten la detección de la retirada del dispositivo, se han obtenido unos resultados que dan lugar a cambios en el estado de la aplicación.

- En primer lugar, se decide no utilizar el sensor de frecuencia cardiaca en este algoritmo. Aunque ha sido la idea principal desde el inicio del proyecto se ha observado que se producen demasiados falsos positivos que no nos indican una retirada. Al basarse este sensor en el reflejo de un haz de luz, cuando se retira el dispositivo de la muñeca, este haz sigue reflejando en distintas superficies y devolviendo valores que podrían ser similares a los obtenidos durante la utilización del reloj. Además, con estos valores son muy variados según la superficie en la que refleja y no se ha logrado obtener ningún patrón de comportamiento.
- Se decide en un principio reforzar el algoritmo de detección, en primer lugar se realiza la comprobación con los valores del acelerómetro, si se detecta retirada, se confirma con los valores del sensor de luz ambiente.
- Se realizan las mismas pruebas con este algoritmo desarrollado, comprobando los valores devueltos por ambos sensores. Se observa que el sensor de luz ambiente es muy sensible a cambios mínimos de luz en el entorno. Esto nos puede inducir a errores en las comprobaciones por lo que se llega a la conclusión de detectar la retirada únicamente con las mediciones del acelerómetro, ya que un buen ajuste del umbral hace este algoritmo muy preciso.

6.3. Pruebas de aceptación

Las pruebas de aceptación consisten en realizar pruebas de uso del software y ver que se cumplen las especificaciones del sistema.

Debido a la naturaleza de este proyecto, en el que es necesario comprobar los valores medidos por los sensores y estos son muy variados en función del entorno, y por la falta de tiempo para realizar todas las pruebas que son necesarias en un proyecto software, se ha decidido realizar únicamente pruebas de aceptación. Se piensa que estas pruebas son las más necesarias en este proyecto para comprobar su correcto funcionamiento.

En primer lugar, se han realizado pruebas de monitorización completas comprobando todos las posibles situaciones de uso. Las pruebas realizadas son las siguientes.

- Pruebas para comprobar que la aplicación no se interrumpe en ningún momento.
- Pruebas para comprobar que, con la aplicación ejecutándose y apagando el dispositivo, al iniciarse de nuevo la aplicación se inicia automáticamente.
- Pruebas para comprobar que se detecta la retirada dentro de los intervalos fijados.
- Pruebas para comprobar que no se detectan falsas retiradas cuando el dispositivo se encuentra en la muñeca del portador.
- Pruebas para comprobar que se realiza el envío de la notificación de emergencia y posterior llamada.
- Pruebas para comprobar el correcto envío al servidor y la precisión de la ubicación enviada. Utilizando como apoyo el servidor proporcionado para visualizar los datos.

Tras la realización de estas pruebas, verificando que todos los casos de uso se realizan correctamente sin errores. Se concluye la aplicación funcional de este proyecto.

Como se ha comentado en puntos anteriores, aunque desde el principio se ha priorizado la precisión tanto de la ubicación como de los intervalos en los que esto se envía, es necesario realizar un estudio de la duración de la batería ya que para poder aprovechar su uso en casos reales, es necesario un tiempo de funcionamiento mínimo del dispositivo.

Las pruebas que se muestran a continuación se han realizado con este fin. Con el estado actual de la aplicación, se mide la degradación de la batería con la aplicación ejecutándose de manera continua en segundo plano. Se realizaran cambios en algunos parámetros para observar si se producen mejoras en cuanto al tema de energía sin perjudicar el funcionamiento de la aplicación.

6.3.1. Pruebas estado de la batería

Con la funcionalidad especificada en los requisitos desarrollada por completo, se realizarán distintas pruebas de uso normal del dispositivo para comprobar la duración de la batería. Estas pruebas se realizan en dos dispositivos distintos: Oppo Watch Bluetooth y Samsung Galaxy Watch 4 Classic 4G.

Los parámetros que se modificarán para realizar estas pruebas serán el intervalo de actualización de la ubicación y la prioridad del proveedor de ubicación. Respecto a la prioridad se trabajara con dos tipos:

- *PRIORITY-HIGH-ACCURACY*, se usa para aplicaciones que miden la ubicación en tiempo real, es muy precisa y el intervalo se respeta.
- *PRIORITY-BALANCED-POWER-ACCURACY*, intenta compensar entre la precisión y el uso de energía. Intenta obtener en primer lugar de la ubicación de proveedores con menos consumo (como el Wifi). La precisión se pierde con un error de unos 40 metros. El intervalo no se cumple tal cual, es posible que sufra variaciones.

6.3.1.1. Pruebas en Oppo Watch

Se exponen a continuación algunas de las pruebas realizadas en este dispositivo.

Nombre	Prueba 1 Oppo Watch.
Condiciones de la prueba	-Duración de la prueba: 10 horas. - Intervalo medición ubicación: 3 minutos - Prioridad: *PRIORITY-HIGH-ACCURACY* - Se realiza un uso normal, se realizan dos retiradas del dispositivo y se acciona una vez el botón de emergencia.
Resultados	Duración batería: 10 horas. Ver figura 6.2
Conclusiones de la prueba	Se obtiene una duración batería de 10 horas con la aplicación ejecutándose durante todo el periodo. La degradación de la batería es bastante lineal según avanza el tiempo.

Tabla 6.5: Prueba 1 Oppo Watch.

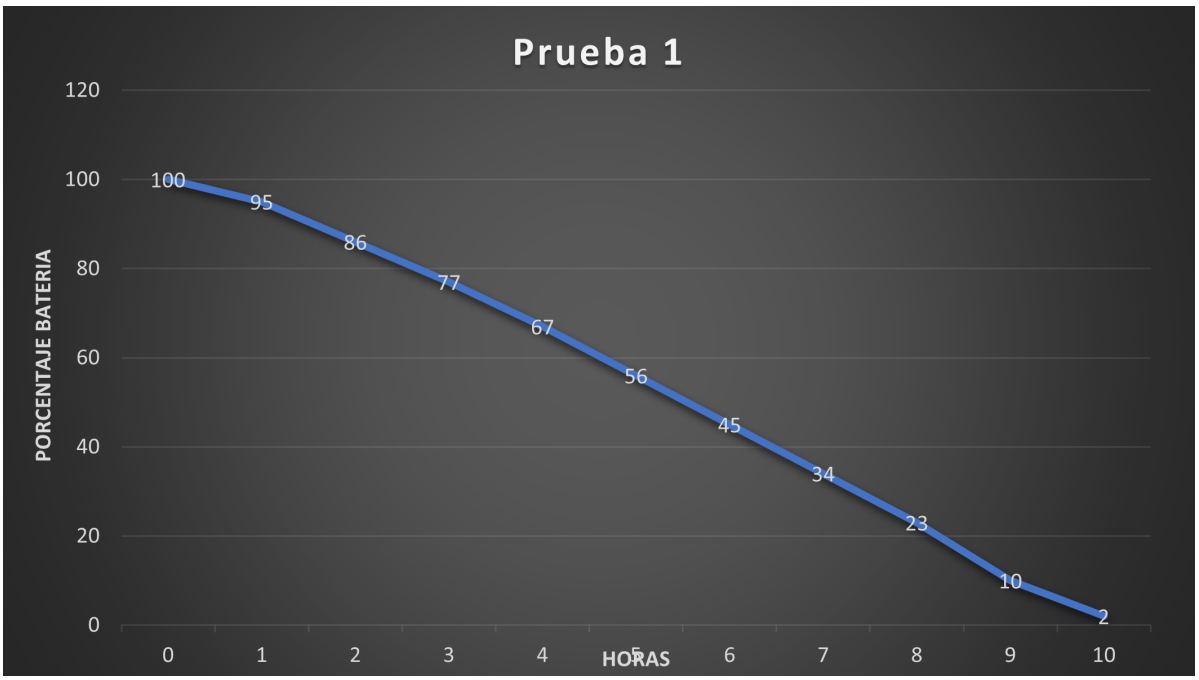

Figura 6.2: Gráfica resultados prueba 1 Oppo Watch.

Nombre	Prueba 2 Oppo Watch.
Condiciones de la prueba	- Duración de la prueba: 9,5 horas. - Intervalo medición ubicación: 8 minutos - Prioridad: *PRIORITY-HIGH-ACCURACY* - Se deja 2 horas al dispositivo sin conexión a internet para que almacene en base de datos y después envíe todas las ubicaciones.
Resultados	Duración batería: 9 horas y media aprox. Ver figura 6.7
Conclusiones de la prueba	Se obtiene una duración batería de 9 horas y media con la aplicación ejecutándose durante todo el periodo. La degradación de la batería es bastante lineal según avanza el tiempo. Se observa que, tras el periodo comprendido entre las 5 y las 7 horas que fueron las dos horas que el reloj estuvo sin conexión, la batería se degrada un poco más rápido por el número de operaciones que tiene que realizar en base de datos y envío al servidor.

Tabla 6.6: Prueba 2 Oppo Watch.

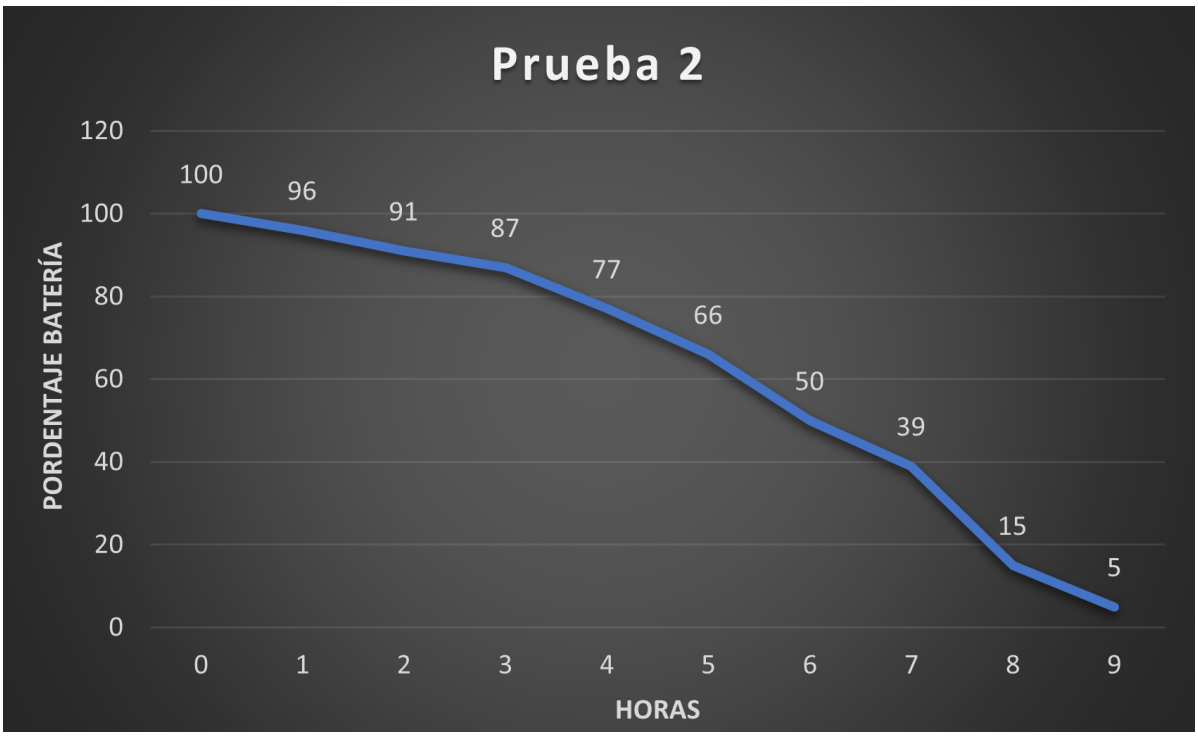

Figura 6.3: Gráfica resultados prueba 2 Oppo Watch.

Nombre	Prueba 3 Oppo Watch.
Condiciones de la prueba	Duración de la prueba: 12 horas. - Intervalo medición ubicación: 15 minutos - Prioridad: *PRIORITY-BALANCED-POWER-ACCURACY* - Se realiza un uso normal, se realizan dos retiradas del dispositivo y se acciona una vez el botón de emergencia.
Resultados	Duración batería: 12 horas aprox. Ver figura 6.4
Conclusiones de la prueba	Se observa un periodo de duración de batería de unas 12 horas, se ha visto una pequeña mejora respecto a las dos pruebas anteriores que se debe al cambio de tipo de frecuencia. La gráfica muestra el porcentaje de batería cada hora ya que las mediciones se toman de la base de datos del servidor y con este intervalo y en este tipo de prioridad el rango de actualización de la ubicación oscila entre los 15 y los 60 minutos.

Tabla 6.7: Prueba 3 Oppo Watch.

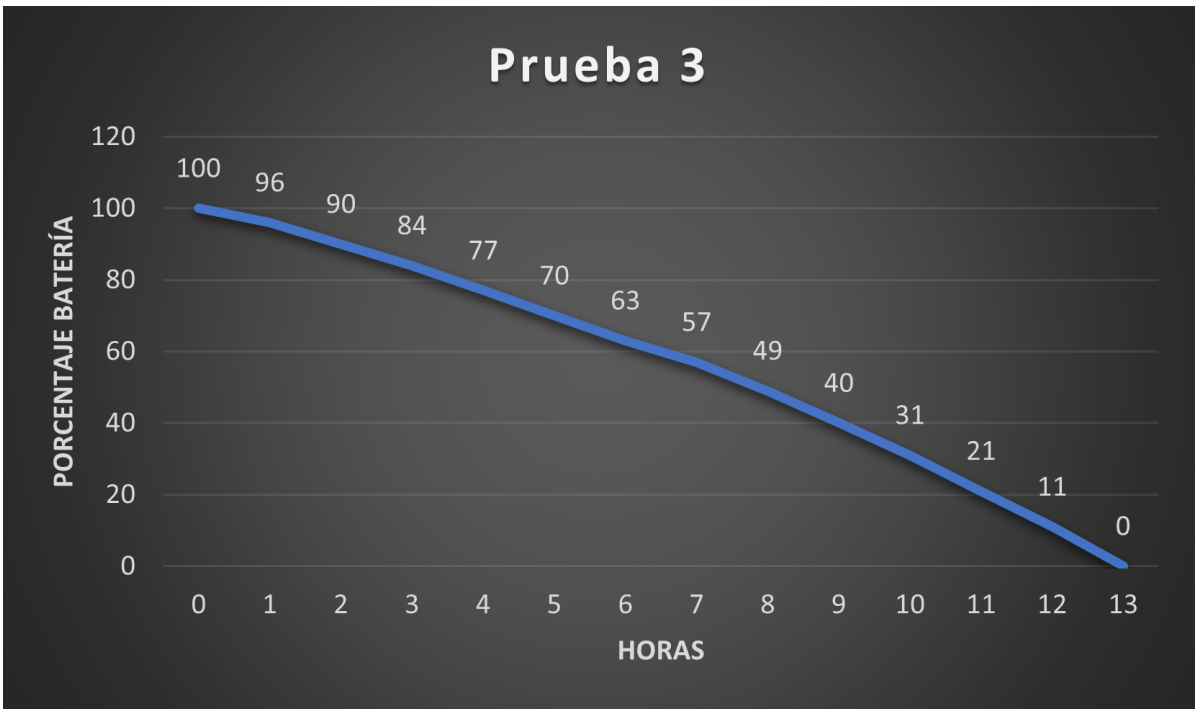

Figura 6.4: Gráfica resultados prueba 3 Oppo Watch.

Nombre	Prueba 4 Oppo Watch.
Condiciones de la prueba	Duración de la prueba: 10 horas. - Intervalo medición ubicación: 5 minutos - Prioridad: *PRIORITY-BALANCED-POWER-ACCURACY* - Se realiza un uso normal, se realizan dos retiradas del dispositivo y se acciona una vez el botón de emergencia.
Resultados	Duración batería: 10 horas aprox. Ver figura 6.5
Conclusiones de la prueba	Se obtiene una duración menor respecto a la prueba anterior con la misma prioridad pero distinto intervalo, en este caso mucho menor. Se refleja en la gráfica un tramo con más valores para que se pueda observar los tiempos de actualización distintos. Se observa que las ubicaciones se actualizan en un rango entre 3 y 7 minutos debido a la prioridad asignada. Se puede concluir con esto que la batería se degrada en un mayor porcentaje cuando el intervalo es menor.

Tabla 6.8: Prueba 4 Oppo Watch.

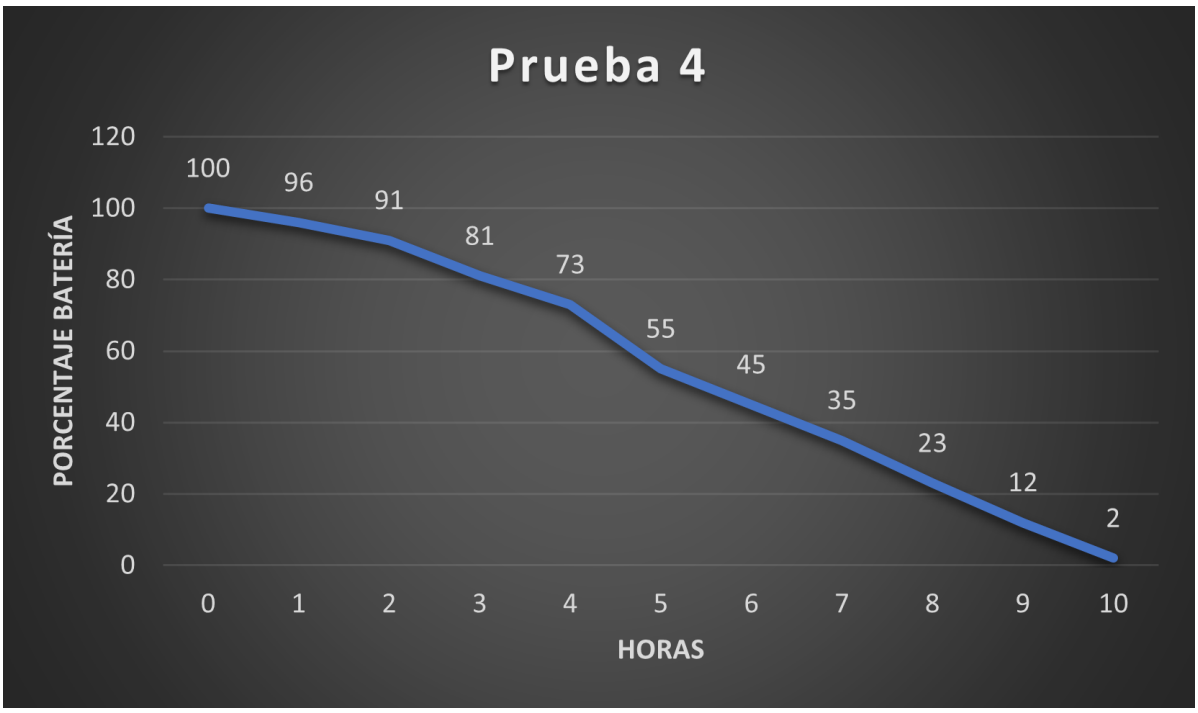

Figura 6.5: Gráfica resultados prueba 4 Oppo Watch.

6.3.1.2. Pruebas en Samsung Galaxy Watch 4

Nombre	Prueba 1 Samsung Galaxy Watch 4.
Condiciones de la prueba	- Duración de la prueba: 16 horas. - Intervalo medición ubicación: 3 minutos - Prioridad: *PRIORITY-HIGH-ACCURACY* - Se realiza un uso normal, se realizan dos retiradas del dispositivo y se acciona una vez el botón de emergencia.
Resultados	Duración batería: 16 horas aprox. Ver figura 6.6
Conclusiones de la prueba	La duración de la batería es bastante considerable. La batería se degrada de forma casi lineal. Una duración de unas 16 horas es óptima ya que no se esperan usos continuos de más de ese tiempo. Si lo comparamos con la prueba 1 realizada en el Oppo (Figura 6.2) se puede observar que la capacidad de la batería de este dispositivo es bastante mayor.

Tabla 6.9: Prueba 1 Samsung Galaxy Watch 4.

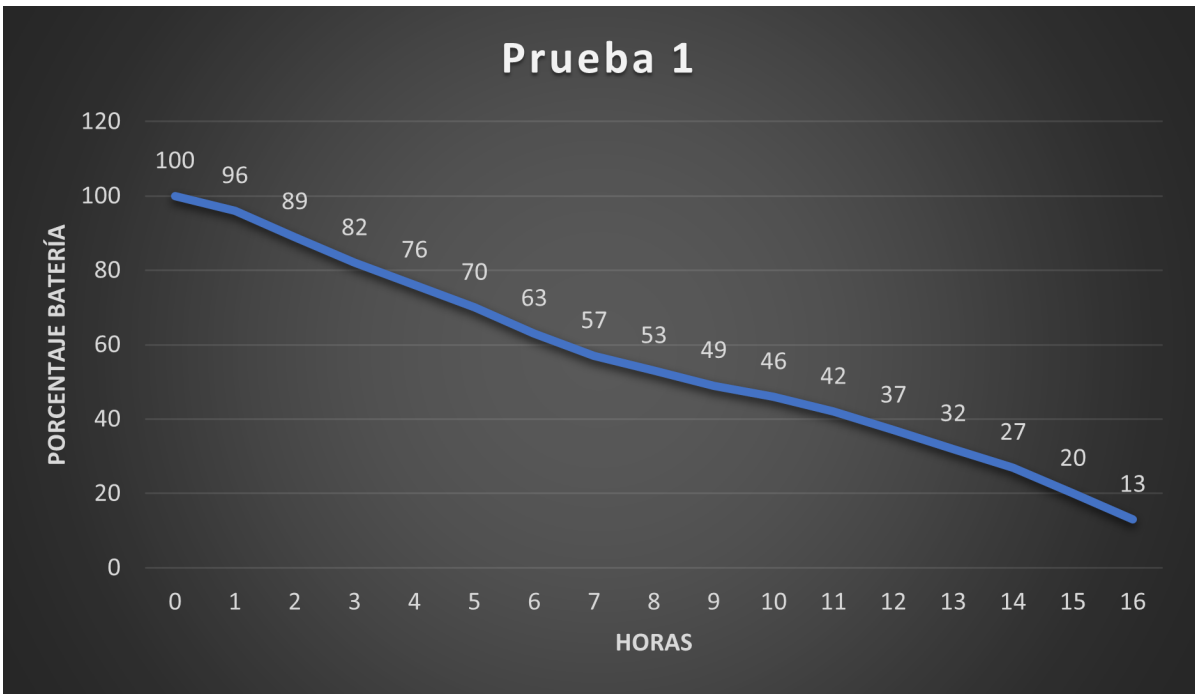

Figura 6.6: Gráfica resultados prueba 1 Galaxy Watch 4.

Nombre	Prueba 2 Samsung Galaxy Watch 4.
Condiciones de la prueba	- Duración de la prueba: 19 horas. - Intervalo medición ubicación: 8 minutos - Prioridad: *PRIORITY-HIGH-ACCURACY* - Se deja 2 horas al dispositivo sin conexión a internet para que almacene en base de datos y después envíe todas las ubicaciones.
Resultados	Duración batería: 19 horas aprox. Ver figura 6.7
Conclusiones de la prueba	Se obtiene una duración batería de unas 19 horas con la aplicación ejecutándose durante todo el periodo. La degradación de la batería es bastante lineal según avanza el tiempo. No se aprecia apenas alteraciones durante el periodo que el dispositivo está sin conexión ni tras este. Se aprecia una pequeña mejora respecto a la prueba anterior.

Tabla 6.10: Prueba 2 Samsung Galaxy Watch 4.

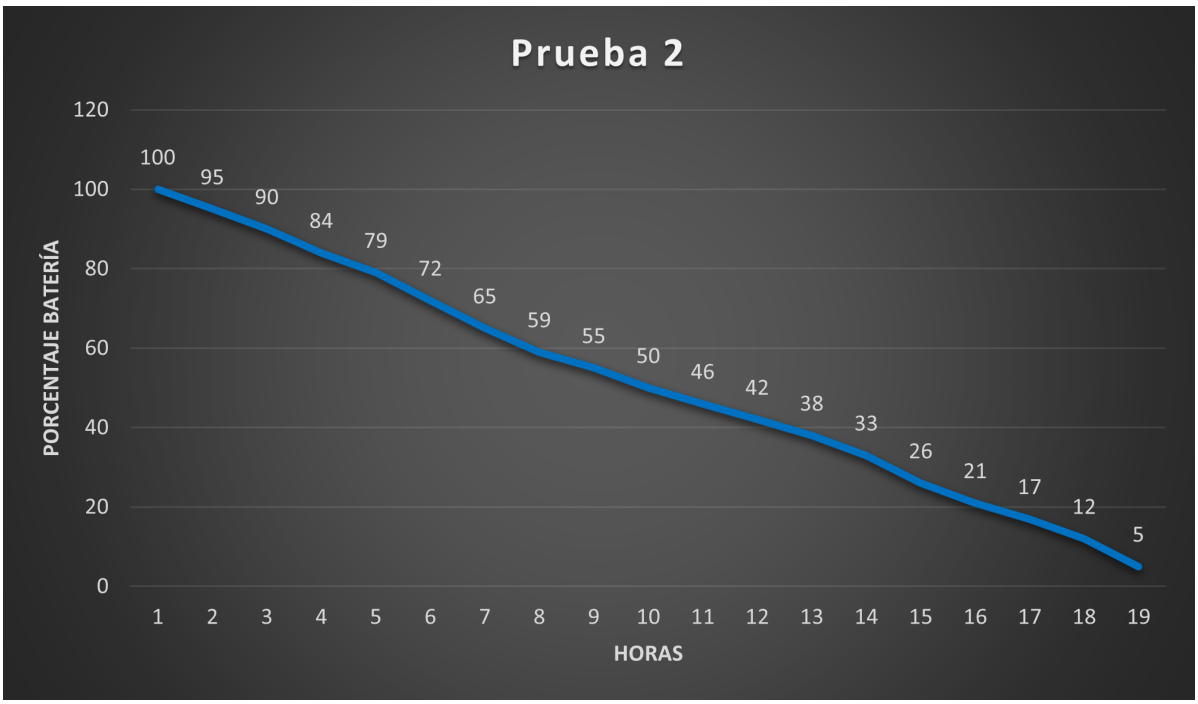

Figura 6.7: Gráfica resultados prueba 2 Samsung Galaxy Watch 4.

Nombre	Prueba 3 Samsung Galaxy Watch 4.
Condiciones de la prueba	- Duración de la prueba: 22 horas. - Intervalo medición ubicación: 15 minutos - Prioridad: *PRIORITY-BALANCED-POWER-ACCURACY* - Se realiza un uso normal, se realizan dos retiradas del dispositivo y se acciona una vez el botón de emergencia.
Resultados	Duración batería: 22 horas aprox. Ver figura 6.8
Conclusiones de la prueba	La duración de la batería mejora notablemente respecto a las dos pruebas anteriores cambiando la prioridad. Además, en este dispositivo, el intervalo se respeta bastante ya que la actualización de la ubicación se realiza en el rango de 13 a 17 minutos. Se considera la mejor prueba realizada hasta el momento, ya que se cree que es un intervalo de monitorización adecuado para todos los casos.

Tabla 6.11: Prueba 3 Samsung Galaxy Watch 4.

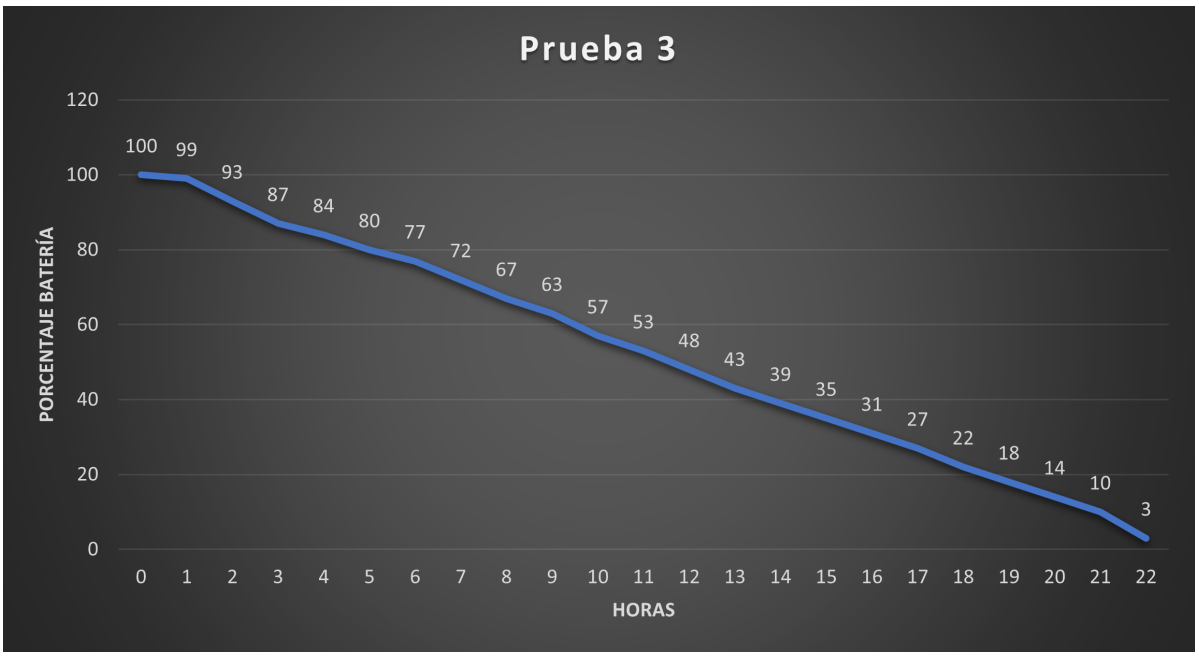

Figura 6.8: Gráfica resultados prueba 3 Galaxy Watch 4.

Nombre	Prueba 4 Samsung Galaxy Watch 4.
Condiciones de la prueba	-Duración de la prueba: 18 horas. - Intervalo medición ubicación: 5 minutos - Prioridad: *PRIORITY-BALANCED-POWER-ACCURACY* - Se realiza un cambio de wifi y bluetooth por datos móviles a las 6 horas.
Resultados	Duración batería: 18 horas aprox. Ver figura 6.9
Conclusiones de la prueba	La duración de la batería es bastante considerable ya que nos proporciona una vida útil de 18 horas. En la gráfica se observa una caída mayor en en el porcentaje de batería en una sola hora tras el cambio de Wifi a datos pero sin importancia. Esta combinación es muy favorable para su uso ya que favorece a un menor consumo de batería y el rango de envío de ubicación observado es de entre 4 y 6 minutos, respetando bastante el intervalo asignado.

Tabla 6.12: Prueba 4 Samsung Galaxy Watch 4.

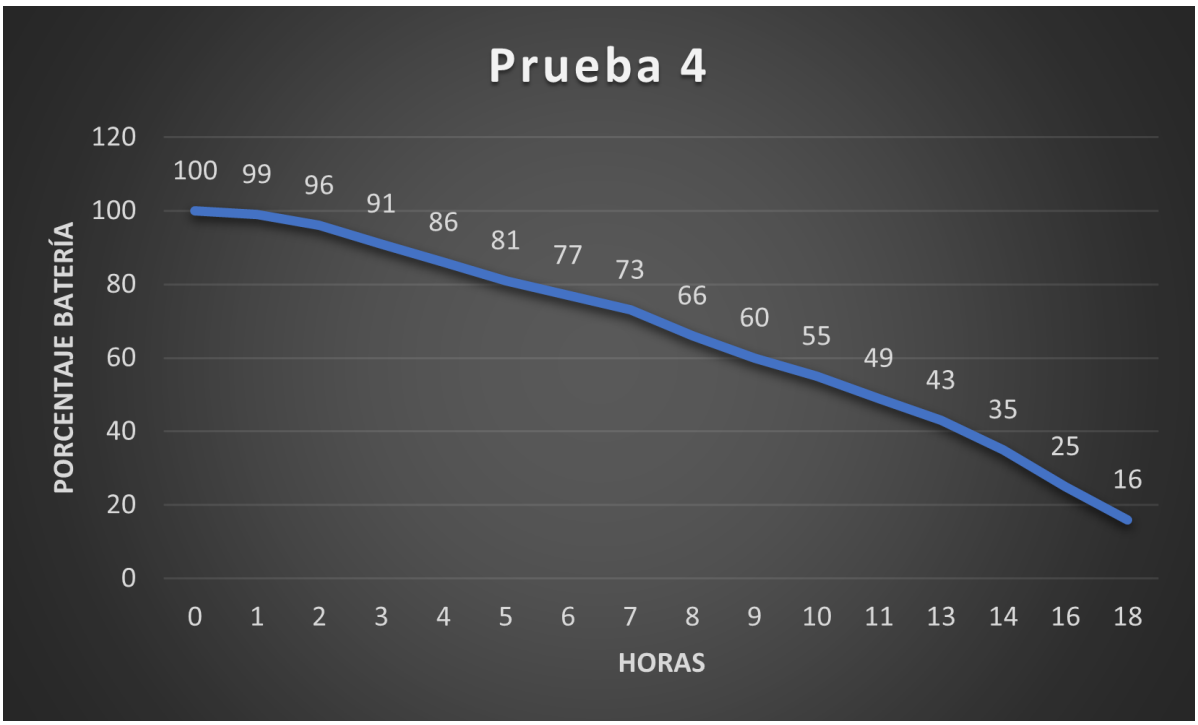

Figura 6.9: Gráfica resultados prueba 4 Galaxy Watch 4.

6.3.2. Conclusiones de las pruebas realizadas y estado de la aplicación tras ellas

Tras la realización de estas pruebas obtenemos varias conclusiones, en primer lugar, y como una de las conclusiones generales, se observa que la duración de la batería es mucho mayor en el dispositivo Samsung, por lo que la capacidad de batería del dispositivo es importante para la utilización de la aplicación. Respecto a las conclusiones por dispositivo, se puede comentar lo siguiente:

- El dispositivo Oppo Watch tiene una batería con poca capacidad para este dispositivo, por ello es necesario priorizar su ahorro para conseguir un tiempo de uso del dispositivo aceptable para estas funciones. La prueba 3 se consigue una duración que puede ser útil pero podría no llegar a compensar con su configuración ya que el intervalo de medida no se ajusta demasiado debido a la prioridad que se ha seleccionado.
- El dispositivo Samsung tiene una batería mucho mayor y todas las pruebas realizadas con distintas configuraciones podrían llegar a ser funcionales para nuestro caso. Además, usando la prioridad que compensa entre la precisión y el consumo de batería, se consigue una duración de 22 horas, el mejor resultado en las pruebas ya que, al contrario que en el otro dispositivo, si que compensa porque se respeta el intervalo en todo momento y la precisión es mucho más alta.

Con todo esto se concluye de manera general que, para la utilización de esta aplicación, se debe elegir correctamente el dispositivo en el que se va a usar. Se debe tener en cuenta tanto su capacidad como la forma en la que consume la batería el propio dispositivo.

Se ha pensado una nueva funcionalidad en la aplicación pensando en dispositivos como el Oppo Watch. Esta funcionalidad permite utilizar 2 modos de uso, uno de ellos para ahorrar batería y el otro para priorizar la precisión, dejando en manos del administrador a que se prefiere dar importancia en cada caso concreto. Para dispositivos como el de Samsung se recomienda el modo con prioridad, ya que la duración es muy aceptable y compensa una mayor precisión de la ubicación.

Capítulo 7

Conclusiones

En este capítulo se van a exponer las distintas conclusiones que se han obtenido una vez finalizado este proyecto de fin de grado. Se comienza con los objetivos que se definieron al inicio del proyecto, analizando su cumplimiento, hasta terminar con posibles trabajos futuros que permitan mejorar el mismo.

En primer lugar, el trabajo ha consistido en realizar una aplicación para Smartwatch con sistema operativo Wear OS. Como no se ha partido de ninguna base de trabajos ni aplicaciones ya desarrolladas, se ha tenido un alto grado de libertad por parte del alumno para la realización de las distintas funcionalidades de esta aplicación así como de las herramientas y tecnologías para su desarrollo.

Al principio esta libertad pudo llegar a parecer un inconveniente, sin embargo, ha supuesto una gran ventaja ya que se han realizado grandes tareas de análisis tanto de tecnologías de desarrollo como dentro del propio proyecto al desarrollar funcionalidades como la detección de retirada del dispositivo. Estas tareas han permitido encontrar distintas ventajas y desventajas en cada uno de los puntos, y se ha dedicado un gran trabajo para poder decantarse por uno u otro.

Tras finalizar un proyecto, es muy importante evaluar el cumplimiento de todos los objetivos que se propusieron en la planificación inicial, y si se diese el caso de que alguno de estos no haya sido posible explicar sus motivos. En este proyecto se han cumplido los 3 objetivos propuestos.

El primero se basó en la monitorización de la ubicación del dispositivo y su posterior envío a un servidor. En el desarrollo de esta funcionalidad no han surgido apenas inconvenientes, además se ha podido aprender sobre las APIs de Android que proporcionan este servicio. El análisis que se realizó durante el periodo de pruebas con el fin de optimizar el uso de la batería se centró principalmente en esta funcionalidad ya que el GPS es de los sensores que más consumen. Se ha analizado este consumo ajustando distintos parámetros lo que ha permitido realizar una aplicación que permite al usuario modificarlos en función de sus necesidades. Sobre esto se comenta un poco más adelante.

El segundo objetivo consistía en la detección de la retirada del dispositivo de la muñeca del portador. Esta funcionalidad ha sido la que más pruebas y análisis ha conllevado durante su desarrollo ya que se han estudiado el funcionamiento de los sensores que incluyen estos dispositivos para poder utilizar uno

de ellos. Se implementó un algoritmo común para todos los sensores y se observó el comportamiento de estos para detectar esa retirada ya que la idea era reforzar con la comprobación de varios sensores. Finalmente se decidió utilizar únicamente el acelerómetro ya que, con el algoritmo implementado, era lo suficientemente robusto y la utilización de otros sensores podía llegar a detectar falsos positivos en esta tarea. Con ello se ha conseguido detectar la retirada de una manera bastante precisa.

El último objetivo se fijó como el desarrollo de un botón de emergencia que desencadenaría en una llamada al centro de control. Cabe mencionar que junto a este objetivo también se decidió desarrollar el despliegue de la aplicación en segundo plano, esta funcionalidad no se fijó como objetivo principal desde un principio ya que quedó a libre elección en función del tiempo que se involucrase en el resto de objetivos. Esta parte mencionada es importante ya que cambio la idea principal del botón de emergencia. Desde un momento se pensó en utilizar los botones físicos, pero la ejecución en segundo plano no lo permitía por lo que desembocó en otro proceso de análisis concluyendo en la incorporación de un botón dentro de la notificación persistente de la aplicación. Por lo tanto, este objetivo también se consiguió junto con la realización de la llamada.

Respecto a las tecnologías y herramientas usadas se decidió utilizar Android Studio con Java como lenguaje de programación. Aunque la principal idea del alumno para realizar este trabajo era utilizar como lenguaje Kotlin, para poder tener una nueva experiencia de lenguaje, ante el tiempo que conlleva aprender una nueva tecnología sumado a que se debía analizar como desarrollar aplicaciones para Wear OS y con la idea de la finalización de este trabajo en el periodo lectivo del segundo cuatrimestre se optó por la opción de utilizar Java debido a la experiencia previa que ya se tenía.

El periodo de pruebas ha resultado un punto muy positivo en este proyecto, ya que ha permitido aprender nuevos conocimientos sobre el funcionamiento de estos dispositivos y sobre todo el impacto que tienen este tipo aplicaciones en el consumo de la batería. Las pruebas a los sensores han permitido realizar mejoras en cuanto a la funcionalidad de retirada del dispositivo. Por otro lado las pruebas respecto a la batería han permitido obtener los mejores ajustes de parámetros en función de cada dispositivo. También dió lugar a una funcionalidad que permite dos modos de uso de la aplicación, uno que prioriza el ahorro de batería, siendo menos preciso en la ubicación y otro que prioriza la ubicación respecto al consumo de batería. Esta elección queda a manos del usuario final.

Aunque el proyecto ha sido finalizado con pequeña demora respecto al tiempo estimado, se ve como un punto negativo una falta de tiempo para el periodo de pruebas que habría permitido mejorar mucho más la aplicación. Aún así, el periodo de pruebas realizado ha sido bastante satisfactorio ya que, aunque es muy difícil en este tipo de proyectos de fin de grado realizar una alta batería de pruebas, se ha conseguido detectar una serie de mejoras considerable.

7.1. Trabajos futuros

En este apartado se presentan una serie de ideas que se han ido teniendo durante el desarrollo del proyecto que pueden llegar a presentar mejoras en la aplicación.

- **Implementación de pruebas.** Como se ha comentado en las conclusiones, sería importante realizar todas las pruebas necesarias en cualquier proyecto software.
- **Botón de emergencia.** En este proyecto no se ha podido implementar la pulsación de un botón físico, que mejoraría la experiencia del usuario, para la notificación de la emergencia, pero se cree que para este apartado podría analizarse APIs externas u otras soluciones que permitan esa funcionalidad y que para este proyecto no ha sido posible debido al tiempo.
- **Mejoras consumo de energía.** Aunque va ligado a la realización de más pruebas, se puede seguir probando la funcionalidad y optimizando la aplicación para que el consumo de energía sea menor y pueda funcionar durante un mayor periodo de tiempo en el dispositivo.
- **Actualizaciones de código.** Las librerías y APIs de Android cambian y se actualizan constantemente por ello, para seguir utilizando la aplicación en un futuro se debe mantener actualizadas. Esto es primordial ya que se tratan de librerías de Google en su mayoría que pueden caer en desuso.
- **Análisis de datos.** Sería muy interesante realizar un análisis de los datos que se envían de cada dispositivo. Con ello se podría llegar a mejorar el código en función del dispositivo con el que se trabaje.
- **Optimización del dispositivo.** Se ha pensado que sería una buena idea optimizar el dispositivo en el que se ejecuta con el tema de mejorar el gasto energético. Esta optimización podría ser eliminando todas las apps innecesarias entre otras cosas.
- **Detección de apagado del dispositivo.** Sería muy interesante poder detectar cuando se apaga el dispositivo y así enviar una notificación el centro de control. Esta funcionalidad no ha podido ser estudiada en este proyecto debido a la falta de tiempo para ello.

Apéndices

Apéndice A

Acrónimos

- **ADB**: Android Debug Bridge.
- **API**: Application Programming Interface.
- **APK**: Android Package Kit.
- **APP**: Aplicación.
- **CU**: Caso de uso.
- **DAO**: Data Access Object.
- **DTO**: Data Transfer Object.
- **GB**: GigaByte.
- **GPS**: Global Positioning System.
- **HAL**: Hardware Abstraction Layer.
- **HTTP**: Hyper Text Transfer Protocol.
- **JSON**: JavaScript Object Notation.
- **JVM**: Java Virtual Machine.
- **LTE**: Long Term Evolution.
- **MVC**: Model View Controller.
- **NFC**: Near-Field Communication.
- **OS**: Operative System.
- **RAM**: Random Access Memory.
- **REST**: Representational State Transfer.

- **RF**: Requisito Funcional.
- **RI**: Requisito de Información.
- **RNF**: Requisito No Funcional.
- **SQL**: Structured Query Language.
- **SSD**: Solid State Drive.
- **TFG**: Trabajo Fin de Grado.
- **UI**: User Interface.
- **UML**: Unified Modeling Language.
- **URL**: Uniform Resource Locator.
- **UTC**: Universal Time Coordinated.
- **UVA**: Universidad de Valladolid.

Apéndice B

Manual de despliegue

La aplicación debe ser instalada en un dispositivo smartwatch con Wear OS como sistema operativo, con una versión de Android 6.0 al menos.

La instalación puede ser llevada a cabo de dos maneras distintas. En ambas se debe instalar Android Studio y clonar el repositorio git y a partir de él se podrá realizar la instalación. Es por esto que es necesario de un ordenador.

B.1. Instalación Android Studio y clonar repositorio.

Lo primero de todo se debe instalar el entorno de trabajo, para ello accedemos a la siguiente URL: `https://developer.android.com/studio`, en la sección Download. Simplemente descargamos los archivos, ya que detecta automáticamente el sistema operativo del dispositivo y su arquitectura.

Una vez instalado, ejecutamos el programa y nos dirigimos a la pestaña **Git** y en el desplegable seleccionar **Clone...** Esto abrirá una ventana donde se deberá insertar en el campo URL la siguiente dirección: y con esto ya se tendría disponible el proyecto. (Ver figuras B.1 y B.2).

B.2. Instalación por Depuración ADB

Primero se debe activar el dispositivo la depuración ADB y la depuración por Wifi. Esto se realiza dentro de los ajustes, en las opciones de desarrollador. Para activarlas se realizan los siguientes pasos:

1. Entrar en los ajustes del dispositivo.

2. Entrar en la opción **Acerca del reloj**.

3. Pulsar varias veces seguidas en la opción **Versión de software** hasta que aparezca en la pantalla del siguiente texto: *Activadas Opciones de desarrollador*.

Tras esto, en ajustes, ir a **Opciones de desarrollador**, activar **Depuración ADB** y **Depurar con Wifi** y seleccionar la opción de **Sincronizar nuevo dispositivo** que nos mostrará un código.
Seguidamente se debe ir a Android Studio, seleccionar la ventana **Device Manager**, la pestaña **Physical** y pulsamos el botón **Pair using Wi-Fi**, se desplegará una ventana y se debe seleccionar **Pair using pairing code** donde introduciremos el código proporcionado anteriormente.
Con esto el dispositivo quedará conectado a Android Studio y el siguiente paso será dar al botón **Run** y la app se instalará en el dispositivo.

B.3. Instalación por APK

Para este caso, simplemente debemos ir a la pestaña **Build** en Android Studio, en el menú desplegable seleccionar **Build bundle(s)/APK(s)** y en el siguiente desplegable **Build APK(s)**. Esto generará un archivo APK que simplemente deberá ser transferido al dispositivo, ya sea conectando el reloj por USB al ordenador o a través del móvil con alguna de sus aplicaciones de transferencia de archivos.

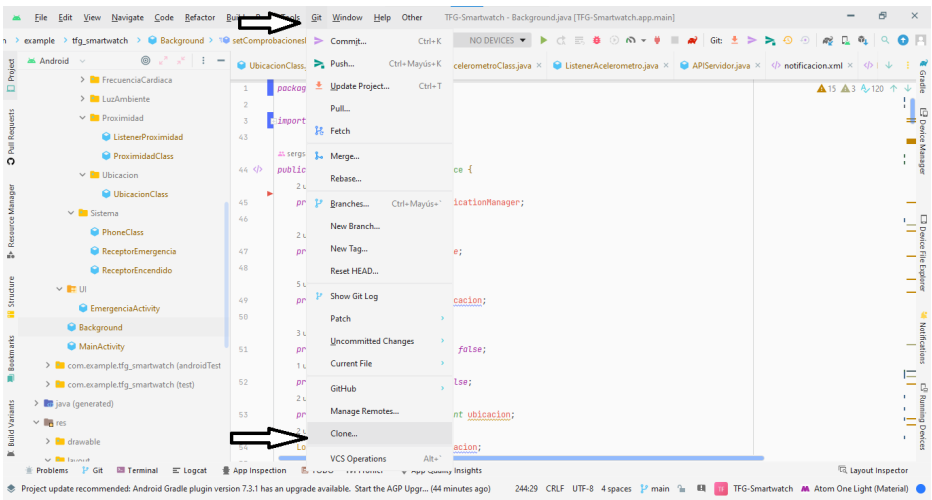

Figura B.1: Captura despliegue en Android Studio.

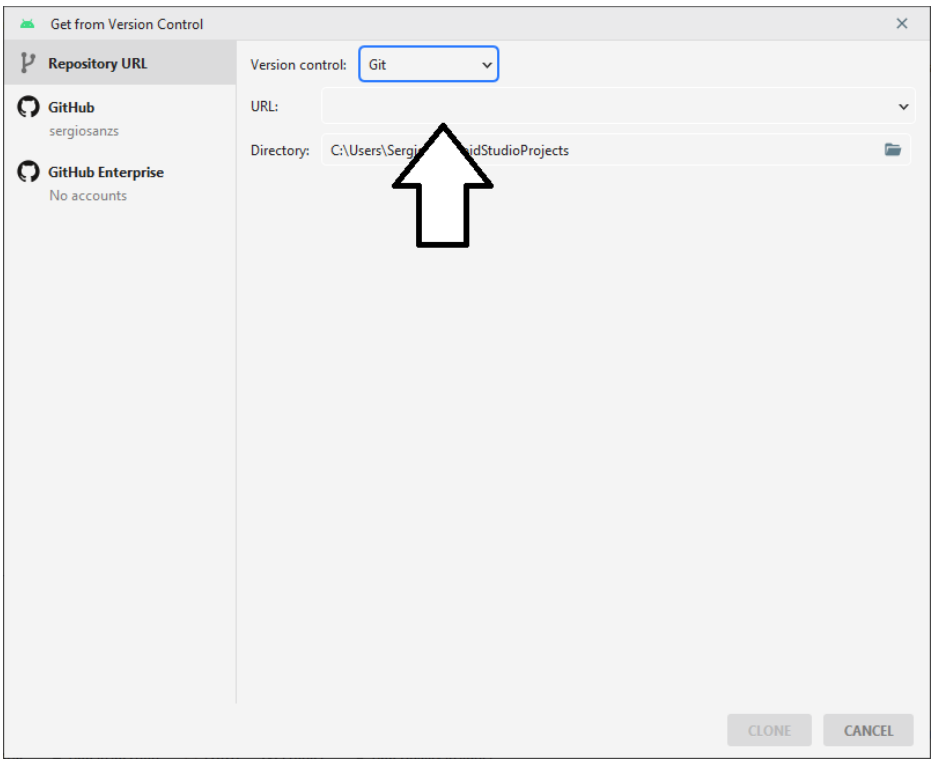

Figura B.2: Captura despliegue en Android Studio.

Apéndice C

Manual de uso

En este apéndice, con la aplicación ya instalada en el servidor, se expone un manual para su utilización.

C.1. Manual de uso Administrador

Al iniciar la aplicación, esta no mostrará ninguna vista y se ejecutará automáticamente en segundo plano. Tras ello, saltará una notificación persistente en el reloj con lo que se observa que la app se ha ejecutado correctamente. El diseño de la notificación es similar al mostrado en la figura C.1 aunque puede variar según el modelo del dispositivo.

Figura C.1: Captura de pantalla de la notificación persistente.

Tras pulsar encima de la notificación se despliega su contenido, se puede ver en la figura C.2, se muestra el Android ID del dispositivo junto con el botón de emergencia.
El administrador debe registrar este Android ID en el servidor, así quedará vinculado este dispositivo, y tras volver a pulsar sobre el texto de la notificación se volverá a mostrar como en la figura C.1 y la aplicación estará lista para ser usada por el paciente.

Figura C.2: Captura de pantalla de la notificación persistente tras pulsación.

El administrador podrá ver en caso de algún fallo de la app el motivo del mismo accediendo al Log de errores. Para ello simplemente conectar el dispositivo a un ordenador con Android Studio iniciado y en la pestaña Logcat aparecerá el error con una descripción. Por ejemplo *.ERROR", "No se encuentra un proveedor de GPS disponible"*.

C.2. Manual de uso paciente

Con la aplicación previamente inicializada por el administrador, el paciente deberá utilizar el reloj con normalidad ya que para un uso normal de la aplicación no es necesario la interacción del usuario.

En el caso de que el paciente quiera notificar una emergencia deberá realizar los siguientes pasos:

1. En primer lugar, desplazarse en la pantalla hasta la notificación persistente, que tiene el aspecto que se muestra en la figura C.1. Esta notificación se ha configurado para que siempre salga la primera de todas y facilitar que sea encontrada.
2. Pulsar sobre la notificación, se mostrará la pantalla como en la figura C.2.
3. Pulsar sobre el botón de la pantalla que incluye el texto EMERGENCIA.
4. Se mostrará una pantalla similar a la figura C.3 donde aparecerán dos botones Aceptar y Cancelar.
5. Si se desea confirmar esa emergencia, se deberá pulsar en el botón Aceptar, antes de que el contador que se muestra en el otro botón llegue a 0, y se realizará una llamada al centro de control automáticamente. Si no se pulsa el botón Aceptar antes de que finalice el contador, se cancelará la notificación de la emergencia, desaparecerá esta pantalla y la aplicación seguirá funcionando con normalidad.
6. Si se desea cancelar ese envío de notificación de emergencia, basta con pulsar el botón Cancelar y desaparecerá esta pantalla y la aplicación seguirá funcionando con normalidad.

Figura C.3: Captura de pantalla de la vista para confirmar la emergencia.

Apéndice D

Contenido del TFG

Se adjunta junto con la memoria del TFG, cuyo nombre es MemoriaTFGSergioSanzSanz.pdf, un repositorio GitLab de la Escuela de Ingeniería Informática de la Universidad de Valladolid que contiene el código de la aplicación desarrollada. Este repositorio se accede mediante la siguiente URL: `https://gitlab.inf.uva.es/sergsan/tfgsergiosanzsanz.git`

El repositorio contiene lo siguiente:

- **app**: Carpeta que contiene la aplicación, sigue la estructura de las aplicaciones Android. Siguiendo la ruta *app >src >main >java >com >example >tfg_smartwatch* se encuentran todas las carpetas del proyecto que se han comentado en esta memoria.
- **build.gradle**: Archivo de configuración comentando en la sección correspondiente.
- **Otros archivos**: Otros archivos genéricos de las aplicaciones Android.
- **README.md**: Fichero que contiene información de la aplicación.

Bibliografía

[1] Bi incorporated. Visitado: 2023-06-04. [Online]. Available: https://bi.com/

[2] Track group. Visitado: 2023-06-04. [Online]. Available: https://trackgrp.com/

[3] Smartwatch: El avance de la tecnología de la mano de la moda. Visitado: 2023-06-04. [Online]. Available: https://www.merca2.es/2023/04/18/smartwatch-avance-tecnologia-mano-moda-1283361/

[4] Dispositivos de control telemático de medidas y penas de alejamiento. Visitado: 2023-06-04. [Online]. Available: https://violenciagenero.igualdad.gob.es/informacionUtil/recursos/dispositivosControlTelematico/home.htm

[5] Las pulseras electronicas de proximidad y el quebrantamiento de condena. Visitado: 2023-06-04. [Online]. Available: https://peritoit.com/2017/07/17/las-pulseras-electronicas-de-proximidad-y-el-quebrantamiento-de-condena/

[6] Xataka. Visitado: 2023-01-10. [Online]. Available: https://www.xataka.com/categoria/analisis

[7] Best wearos smartwatches. Visitado: 2023-01-10. [Online]. Available: https://geekflare.com/es/best-wearos-smartwatches/

[8] What is android? Visitado: 2023-06-04. [Online]. Available: https://www.android.com/intl/es_es/what-is-android/

[9] Android. Visitado: 2023-06-04. [Online]. Available: https://es.wikipedia.org/wiki/Android

[10] Open handset alliance. Visitado: 2023-06-04. [Online]. Available: https://es.wikipedia.org/wiki/Open_Handset_Alliance

[11] Arquitectura de la plataforma android. Visitado: 2023-06-04. [Online]. Available: https://developer.android.com/guide/platform?hl=es-419

[12] Wear os by google. Visitado: 2023-06-04. [Online]. Available: https://wearos.google.com/intl/es_es/

[13] Wear os. Visitado: 2023-06-04. [Online]. Available: https://es.wikipedia.org/wiki/Wear_OS

[14] Android studio. Visitado: 2023-03-29. [Online]. Available: https://developer.android.com/studio

[15] Intellij idea. Visitado: 2023-06-04. [Online]. Available: https://www.jetbrains.com/idea/

[16] What is java?. Visitado: 2023-06-04. [Online]. Available: https://www.java.com/es/download/help/whatis_java.html

[17] Java (lenguaje de programación). Visitado: 2023-06-04. [Online]. Available: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaciÃşn)

[18] ¿qué es java y para que sirve? Visitado: 2023-06-04. [Online]. Available: https://www.cursosaula21.com/que-es-java/

[19] Room. android developers. Visitado: 2023-06-04. [Online]. Available: https://developer.android.com/jetpack/androidx/releases/room?hl=es-419

[20] Persistencia de datos en android con room. Visitado: 2023-06-04. [Online]. Available: https://www.adictosaltrabajo.com/2019/03/04/persistencia-de-datos-en-android-con-room/

[21] Room, la libreria de base de datos de android. Visitado: 2023-06-04. [Online]. Available: https://devexperto.com/room-la-libreria-de-base-de-datos-de-android/

[22] Adavnced rest client. Visitado: 2023-06-04. [Online]. Available: https://install.advancedrestclient.com/

[23] Esquema metodología desarrollo iterativo e incremental. Visitado: 2023-06-03. [Online]. Available: http://alaricoi.esy.es/wp-content/uploads/2016/04/iterativo-incremental-1.png

[24] Astah professional. Visitado: 2023-03-29. [Online]. Available: https://astah.net/products/astah-professional/

[25] Overleaf. Visitado: 2023-03-29. [Online]. Available: https://es.overleaf.com

[26] Gitlab. Visitado: 2023-03-29. [Online]. Available: https://gitlab.inf.uva.es/

[27] Microsoft teams. Visitado: 2023-03-29. [Online]. Available: https://www.microsoft.com/es-es/microsoft-teams/

[28] Sueldo medio desarrollador juniro android en españa. Visitado: 2023-01-10. [Online]. Available: https://www.glassdoor.es/Sueldos/desarrollador-android-junior-sueldo-SRCH_KO0,28.htm

[29] Cuanto gasta un ordenador encendido 24 horas. Visitado: 2023-01-10. [Online]. Available: https://www.profesionalreview.com/2022/09/29/cuanto-gasta-un-ordenador-encendido-24-horas/

[30] ¿como funciona y qué hace el acelerómetro?. Visitado: 2023-02-22. [Online]. Available: https://www.tme.eu/es/news/library-articles/page/22568/Como-funciona-y-que-hace-el-acelerometro/

[31] Guía de arquitectura de apps de android. Visitado: 2023-06-09. [Online]. Available: https://developer.android.com/jetpack/guide?hl=es-419

[32] Patrón de diseño. Visitado: 2023-06-09. [Online]. Available: https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

[33] Patrón de diseño de software. Visitado: 2023-06-09. [Online]. Available: https://devexperto.com/patrones-de-diseno-software/

[34] Patrón de diseño de software. observador. Visitado: 2023-06-09. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/observer

[35] Patrón observador frente a pub-sub. Visitado: 2023-06-09. [Online]. Available: https://hackernoon.com/es/observador-vs-pub-sub-patron-50d3b27f838c

[36] Data access object (dao). Visitado: 2023-06-09. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/dao

[37] Data transfer object (dto). Visitado: 2023-06-09. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/dto

[38] AbstracciÓn de datos (i): El patrÓn dao. Visitado: 2023-06-09. [Online]. Available: https://danielggarcia.wordpress.com/2009/05/13/abstraccion-de-datos-i-el-patron-dao/

[39] Singleton. Visitado: 2023-06-09. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/singleton

[40] Singleton. Visitado: 2023-06-09. [Online]. Available: https://refactoring.guru/es/design-patterns/singleton

[41] Factory method. Visitado: 2023-06-09. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-de-diseno/factory-method

[42] Patron de diseño factory. Visitado: 2023-06-09. [Online]. Available: https://www.oscarblancarteblog.com/2014/07/18/patron-de-diseno-factory/

[43] Todo acerca de los diagramas de paquetes uml. Visitado: 2023-06-11. [Online]. Available: https://www.lucidchart.com/pages/es/que-es-un-diagrama-de-paquetes-uml

[44] La guía fácil de los diagramas de despliegue uml. Visitado: 2023-06-11. [Online]. Available: https://creately.com/blog/es/diagramas/tutorial-de-diagrama-de-despliegue/

[45] Androidmanifest. Visitado: 2023-02-07. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro

[46] Sensormanager. Visitado: 2023-02-16. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorManager

[47] Sensoreventlistener. Visitado: 2023-02-16. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorEventListener

[48] Android id. Visitado: 2023-03-25. [Online]. Available: https://developer.android.com/reference/android/provider/Settings.Secure#ANDROID_ID

[49] Intents comunes. Visitado: 2023-05-10. [Online]. Available: https://developer.android.com/guide/components/intents-common

[50] Intents y filtros de intents. Visitado: 2023-05-10. [Online]. Available: https://developer.android.com/guide/components/intents-filters

[51] Diferencias entre el desarrollo para wear os y dispositivos móviles. Visitado: 2023-07-03. [Online]. Available: https://developer.android.com/training/wearables/wear-v-mobile?hl=es-419

[52] Principios del desarrollo para wear os. Visitado: 2023-07-03. [Online]. Available: https://developer.android.com/training/wearables/principles?hl=es-419

[53] Pruebas de software. Visitado: 2023-06-11. [Online]. Available: https://es.wikipedia.org/wiki/Pruebas_de_software

[54] La importancia de las pruebas de software. Visitado: 2023-06-11. [Online]. Available: https://www.unir.net/ingenieria/revista/pruebas-software/