Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Detección de caídas usando el ecosistema de sensores integrados en un smartwatch con WEAR OS

Autor:
María del Ser Gutiérrez

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Detección de caídas usando el ecosistema de sensores integrados en un smartwatch con WEAR OS

Autor:
María del Ser Gutiérrez
Tutores:
Mario Corrales Astorgano
Alfonso Bahillo Martínez

El mundo no te da cosas: tú debes tomarlas.
-Evelyn Hugo.

Agradecimientos

En primer lugar, quiero agradecerles el apoyo brindado a los tutores Alfonso y Mario, por guiarme a lo largo de la realización del proyecto para que le diera importancia a las cosas importantes y no perdiera el tiempo yéndome por las ramas.

En segundo lugar, a mis amigos y familia, por aguantarme durante la carrera y más específicamente durante este proyecto, que si no les hubiera tenido para escuchar mis quejas me habría desesperado mucho antes de llegar aquí.

Por último, a mi Maestro, por dejarme el suelo del *dojo* para practicar las caídas con las que he probado el funcionamiento de la app y por enseñarme que la perseverancia es algo que se hace, no algo con lo que se nace.

Resumen

Gracias a los avances tecnológicos realizados en los últimos años, personas en situación de riesgo han conseguido vivir su vida de forma más independiente. Aun así, esto no los deja libres de riesgos. De esta forma, es importante monitorizar los accidentes que puedan suceder, más específicamente, las caídas que puedan sufrir. Hoy en día hay una gran variedad de dispositivos propios que permiten esta monitorización, existiendo la opción de utilizar aplicaciones para relojes inteligentes. Sin embargo, estas aplicaciones son privativas, tanto en el caso de las aplicaciones incluidas por defecto en los smartwatches como en el caso de aplicaciones desarrolladas de manera independiente.

Este proyecto, dado al creciente uso que se les está dando a los relojes inteligentes, propone la creación de una aplicación de monitorización de caídas abierta, haciendo uso de algoritmos documentados en literatura científica accesible. Para ello, se realiza la ampliación de una aplicación ya existente, la cual incluye la monitorización de ubicación, monitorización de retirada del dispositivo y posibilidad de enviar un aviso de emergencia de manera manual.

A lo largo de este proyecto se amplía el proyecto tomado como punto de partida, añadiendo un algoritmo de detección de caídas basado en la doble comprobación de umbrales para la detección de la propia caída: detección de caída libre y detección del impacto. Además, se incluye la determinación de la gravedad de la caída, la cual puede ser grave, leve o falsa. Con esto se tiene una aplicación que detecta entre un 90 % y un 100 % de las caídas verdaderas dadas, y tan solo entre un 20 % y un 30 % de falsas caídas.

Abstract

Thanks to technological advances made in recent years, people at risk have been able to live their lives more independently. Even so, this does not leave them free of risks. Thus, it is important to monitor the accidents that may occur, more specifically, the falls they may suffer. Nowadays there is a wide variety of proprietary devices that allow this monitoring, with the option of using applications for smart watches. Nevertheless, these applications are privative, both for the default applications included in smartwatches and in the case of independently developed ones.

This project, given the growing use of smart watches, suggests the creation of an open fall monitoring application, using algorithms documented in available scientific literature. To accomplish this, the extension of an existing application is undertaken, app which includes location monitoring, device removal monitoring and the possibility of manually sending an emergency alert.

Throughout this project, the starting point of the original project is extended by adding a fall detection algorithm based on dual threshold checks for the detection of the fall itself: free fall recognition and impact detection. In addition, it also includes the determination of the severity of the fall, which can be severe, minor or false. This results in an application that detects between 90-100 % of the given true falls, and just 20-30 % of false falls.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

A lo largo de los últimos años se ha dado un envejecimiento de la población en los países desarrollados, dada la bajada de la tasa de nacimientos y el gran aumento de la esperanza de vida [1]. Este envejecimiento supone una pérdida de las capacidades fisiológicas, lo cual hace a las personas dependientes más propensas a sufrir accidentes en solitario[2].

En especial, la tercera edad supone el grupo de mayor riesgo en cuanto al riesgo de caídas. No solo la población envejecida es más propensa a caerse, si no que, además, en caso de darse una caída, los daños sufridos son mayores, siendo más propensos a necesitar hospitalización. Si una caída no se trata a tiempo, puede resultar en problemas como fracturas de huesos y desgarramiento de músculos o posibles parálisis, las cuales pueden derivar en enfermedades aún más graves o incluso la muerte [3].

Un estudio realizado en Suecia en 2004 [4] revela que, de las caídas reportadas, un 25 % derivaban en lesiones, siendo la parte del cuerpo más afectada la cabeza. Además, es importante destacar que el número de caídas es considerablemente mayor que el número de personas que se cayeron (865 caídas frente a 344 personas), lo cual quiere decir que las personas con riesgo de caídas son propensas a caerse varias veces, pudiendo agravar sus lesiones.

Dado este alto peligro, surge la necesidad de poder monitorizar a aquellas personas que se encuentren en el grupo de riesgo de estos accidentes. Por ello se empezaron a crear dispositivos que permitieran a las personas en riesgo notificar de una emergencia.

Se comenzó creando los conocidos "botones de emergencia manual", como por ejemplo aquellos que comenzaron a utilizar en la fundación Cruz Roja [5]. El funcionamiento es básico, en caso de que el usuario se encuentre en una emergencia pulsará un botón que activará la respuesta de emergencia. A pesar de su utilidad, estos dispositivos tienen un problema en el caso de las caídas, pues el usuario puede quedar demasiado dolido o incluso inconsciente, por lo que no podría activar el botón de emergencia.

Para solventar este problema, se comenzaron a desarrollar dispositivos capaces de detectar automáticamente cuándo un usuario se ha caído y notificarlo a un contacto de emergencia. No obstante, estos pueden no ser la mejor opción debido a su precio relativamente elevado para un dispositivo con una única función (entre 50€ y 300€, además de costes mensuales por el servicio [6]); incapacidad para saber si el usuario lo tiene puesto o se lo ha quitado (debido a que muchos simplemente cuelgan del cuello del usuario con una cuerda) y dificultad para personalizar sus funcionalidades.

Por suerte, con el avance de la tecnología, los relojes inteligentes han podido incluir cada vez más y mejores sensores que permiten convertirlos en dispositivos de monitorización. Además, el porcentaje de la población que hace uso de manera diaria de un reloj inteligente va en aumento [7], por lo que proveerles una aplicación en lugar de un dispositivo especializado sería la opción más sencilla y rápida de llevar a cabo. Tal es esta adecuación de los relojes inteligentes para la monitorización, que las grandes compañías proveedoras de Smartwatches ya incluyen por defecto una aplicación de detección de caídas lista para usar desde el momento que enciendes el dispositivo. Sin embargo, estas aplicaciones son privativas, lo cual limita el uso que se les puede dar y nos lleva a la motivación de este proyecto.

1.2. Motivación

Sabiendo que las grandes compañías proveen sus propias aplicaciones de detección de caídas puede parecer innecesario desarrollar una propia, sin embargo, esto no es así. Debido a que son empresas privadas, las aplicaciones son privativas, por lo que no se puede saber qué algoritmos utilizan estas corporaciones para la detección de caídas y tampoco se pueden extraer datos de las mismas de una manera sencilla (como por ejemplo, número de falsos positivos y falsos negativos para evaluar la eficacia de las mismas; gravedad de las caídas detectadas, etc.).

Dada esta situación, surge la necesidad de realizar una aplicación que permita esta detección de caídas, con posibilidad de mejorar el algoritmo de detección conforme avance la tecnología, pudiendo permitir el envio de datos a un servidor para su posible análisis y manteniendo la notificación a contactos de emergencia.

La idea inicial para realización de este proyecto surge de la propuesta de TFG de los profesores Mario Corrales y Alfonso Bahillo. Junto con ellos se desarrolla la idea de incluir esta funcionalidad en una aplicación de monitorización ya existente, realizada por Sergio Sanz [8]. De esta forma se conseguirá una aplicación de monitorización completa y ajustable a las necesidades de cada usuario.

1.3. Estado de la cuestión

El área de la salud y monitorización de pacientes es una de las mayores áreas de investigación tecnológica. Dada la peligrosidad y frecuencia de las caídas en pacientes vulnerables, se han realizado múltiples estudios para determinar una buena forma para detectar caídas y la gravedad de las mismas. Estos métodos son muy variados y van desde sistemas instalados en los hogares mediante obra, hasta

aplicaciones fácilmente instalables en un reloj inteligente. A lo largo de esta sección se describirán brevemente varios enfoques tomados y cuál de ellos será el utilizado en este proyecto.

1.3.1. Sensores ambientales

Las posibles caídas se detectan dadas las mediciones de una instalación de sensores ambientales, los cuales se han de colocar en las viviendas o lugares comúnmente frecuentados por la persona a la cual se quiere monitorizar.

Un enfoque para este tipo de monitorización puede ser la detección de vibraciones del suelo [9]. Este método se basaba en recoger dichas vibraciones y, dada su intensidad y área que ocupaban, determinar si se podían tratar de una persona cayendo al suelo. Otros ejemplos pueden ser la detección mediante sensores piroeléctricos [10] o sensores de presión en el suelo [11].

La gran desventaja de este enfoque es su coste e inconveniencia en edificios ya existentes. Los sensores han de colocarse mediante obra, lo cual los hace significativamente costosos, más aún si se quieren incorporar a posteriori en un inmueble ya existente.

Otra posible opción son los sensores radar. Estos sensores envían ondas milimétricas, las cuales se ven modificadas al rebotar en los diferentes obstáculos que haya en la estancia. El mismo sensor que envió las ondas recoge estos reflejos y, dada la modificación, puede identificar las diferentes superficies de la estancia. De esta forma, se puede identificar una persona como un cúmulo de ondas de similares características [12].

Los sensores necesarios son lo suficientemente pequeños como para solventar el problema de la instalación [13], pues se pueden colocar y mover de manera sencilla. Sin embargo, al funcionar con ondas, las paredes u otros objetos de gran densidad pueden interrumpir la monitorización. Por ello se requiere al menos un sensor en cada estancia del hogar, haciendo que su coste escale rápidamente con el tamaño y cantidad de habitaciones de la vivienda.

1.3.2. Cámaras

Son una opción que a medida que avanza el tiempo ha comenzado a sustituir a los sensores. Esto se debe a que pueden detectar varios eventos de manera simultánea, además de requerir una instalación muy poco intrusiva en el lugar en el que se quieran utilizar. Además, debido almacenan el vídeo obtenido, se puede observar la propia grabación para comprobar que los datos recogidos son correctos.

Diferentes enfoques para este tipo de monitorización pueden ser el reconocimiento de inactividad [14], el reconocimiento de cambio de formas [15] y la monitorización del movimiento de la cabeza en 3D [16].

Cabe destacar una desventaja con la que no cuenta ninguno otro de los enfoques presentados, la invasión de la privacidad. Debido a que es necesario que la persona en riesgo se encuentre monito-

rizada en todo momento, se han de incluir cámaras en lugares más sensibles como los dormitorios y baños. Esto hace que muchos usuarios que se preocupan en gran medida por su privacidad no escojan este enfoque.

1.3.3. Wereables

Dada su fácil producción, comodidad y precisión igualable a aquella de las soluciones más complejas (incluso en algunos casos, mejor) la popularidad de utilizar dispositivos fácilmente equipables ha aumentado en gran medida en los últimos años. Por ello, últimamente ha surgido una gran cantidad de dispositivos cuyo único propósito es la detección de caídas. Además, grandes compañías incluyen esta funcionalidad por defecto en sus relojes inteligentes.

Sin embargo, a pesar de la gran oferta, no muchas de las opciones están bien valoradas por sus usuarios debido a ser demasiado aparatosas o no cumplir con la efectividad que prometían. Debido a que la detección de caída de este proyecto se basa en un reloj inteligente, vemos aquellos wereables que resultan más relevantes en el mercado actual.

1.3.3.1. Vigi'fall

Recomendado por la Fundación Alzheimer España [17] se trata de un parche triangular que va adherido a la persona que quiera ser monitorizada. Tiene la ventaja de ser muy poco intrusivo, pues es fácil de colocar en los usuarios y difícil que se despegue y se pierda (se puede utilizar incluso en la ducha). No obstante, cuenta con un gran inconveniente, se trata de una solución a medio camino entre wereable y sensor ambiental, pues depende de unos sensores colocados en las paredes de la vivienda del usuario [18], por lo que en caso de salir del lugar de instalación pierde completamente su funcionalidad.

Figura 1.1: Parche triangular Vigifall [19]

1.3.3.2. Medical Guardian

En varias páginas web especializadas en la información sobre el cuidado de ancianos de Estados Unidos [20] [21] se tiene el dispositivo de Medical Guardian como una de las mejores opciones para la

detección de caídas. Este se trata de un pequeño aparato que se utiliza colgado en el cuello, el cual cuenta con altavoces, micrófono y un botón físico, además de resistencia al agua y geolocalización [22]. Sus principales desventajas son la dependencia completa de la empresa de servicios a la que pertenece y la incapacidad para detectar si la persona que lo estaba utilizando se lo ha quitado y lo ha dejado abandonado.

Figura 1.2: Colgante Guardian [22]

1.3.3.3. Apple Watch

Los smartwatches de Apple son una gran parte de su ecosistema, siendo esta la compañía que más relojes inteligentes vende en el mundo [23]. Desde del modelo *Apple Watch Series 4* en adelante, todos sus relojes han incluído un sistema de detección de caídas desde fábrica. Para utilizarlo se ha de configurar el *Medical ID* de Apple añadiendo uno o varios contactos de emergencia y se ha de activar la detección automática de caídas [24]. Su principal desventaja es que la detección de estas caídas tan solo sirve para activar la llamada de emergencia, no se permiten extraer datos de la aplicación para analizar la frecuencia y gravedad de las mismas y Apple se reserva el derecho de utilizar los datos recogidos para la mejora de su sistema.

Figura 1.3: Smartwatch Apple Series 5 [25]

1.3.3.4. Samsung Galaxy Watch

Si bien en sistemas Android se pueden utilizar múltiples dispositivos de diferentes compañías, Samsung es aquella que más ventas tiene en el mercado de los relojes inteligentes [23]. Su funcionalidad es muy similar a la que se encuentra en los smartwatches de Apple, siendo una buena alternativa para

el sistema operativo de Android. Para poder utilizar la detección de caídas hay que activarla desde los ajustes de seguridad y emergencias y desde ahí mismo añadir contactos de emergencia [26]. En este caso no es necesario tener configurada la aplicación de *Samsung Health* para poder utilizar la funcionalidad. A pesar de su simplicidad, además de compartir la desventaja fundamental de Apple, Samsung se ha introducido en el mercado de la de detección de caídas de una forma relativamente reciente, por lo que su falta de experiencia hace que su efectividad no sea tan elevada como la de Apple.

Figura 1.4: Diferentes modelos de Smartwatches Samsung [27]

1.3.3.5. uFallAlert

Se trata de una aplicación privativa instalable para dispositivos que tengan los sensores necesarios para la detección de caídas (como, por ejemplo, acelerómetro) pudiendo utilizarse en aquellos que no cuenten con este servicio por defecto o como alternativa a las opciones previamente mencionadas. Con respecto a las opciones previas, tiene como ventaja el poder personalizar el tipo de alerta que se quiere enviar (email/SMS), un detector de inactividad (si el reloj lleva mucho tiempo sin usarse puede que no se tenga puesto) y un acceso al historial de caídas [28]. Sin embargo, a pesar de parecer una gran opción, aún se encuentra en desarrollo (versión beta), por lo que puede que su funcionamiento no sea el adecuado y generar una cantidad considerable de problemas. Además, parece tener gran dependencia con la parte de la aplicación que se ejecuta en el teléfono móvil, por lo que puede no resultar accesible para aquellas personas con un mayor grado de dependencia o cierto nivel de analfabetismo digital. Asimismo, al tratarse de una aplicación privativa no permiten saber qué algoritmos utilizan para la deteccion de caídas.

Figura 1.5: Logo de la aplicación uFallAlert [28]

1.4. Algoritmos de detección de caídas

Dado que se va a trabajar con relojes inteligentes, el algoritmo para la detección de caídas ha de realizarse considerando que los sensores de los que dependa han de encontrarse en la mayoría de dispositivos objetivo. Hoy en día prácticamente todos los relojes inteligentes del mercado cuentan con giroscopios y acelerómetros (suponemos de 3 ejes para la realización de este proyecto). Teniendo esto en cuenta, se presentan dos enfoques para la detección de caídas: algoritmos basados en umbrales y algoritmos basados en el aprendizaje automático.

1.4.1. Enfoque basado en umbrales

El funcionamiento básico de los algoritmos por umbrales se basa en establecer un valor límite (umbral) para medidas que se puedan obtener mediante cálculos aritméticos relativamente simples efectuados sobre las propias medidas tomadas por los sensores del reloj. En caso de que estas medidas superasen el umbral establecido, podríamos decir que se ha dado una caída. Esta simplicidad hace que los algoritmos basados en umbrales sean muy eficientes computacionalmente y muy rápidos a la hora de realizar medidas en tiempo real. Sin embargo, dada su incapacidad para adaptarse a la variabilidad de los movimientos humanos y condiciones ambientales, hay una gran probabilidad de que se den un gran número de falsos positivos, debido a actividades cotidianas que superan el umbral establecido.

La aproximación más simple a este tipo de algoritmos se basa en suponer que, cuando el dispositivo se encuentra en caída, su aceleración total es cero, pues un cuerpo que se encuentra en caída libre se encuentra en un estado de ingravidez. Para este tipo de algoritmos es suficiente realizar el simple cálculo de la aceleración total y encontrar un umbral apropiado [29], pues al darse una caída nunca se llega a la ingravidez real. Como es comprensible, esta perspectiva es demasiado simplista y detecta una gran cantidad de falsas caídas, además de no tener en cuenta ciertas situaciones que se pueden dar en las medidas tomadas por un acelerómetro colocado en una muñeca (como, por ejemplo, que la persona al caer levante los brazos debido a la inercia).

Una forma de mejorar la eficacia de estos algoritmos es no solo realizar las mediciones para detectar la caída inicial, sino seguir midiendo durante un período más largo de tiempo para realizar diferentes comprobaciones. De esta manera se puede detectar una caída mediante la mencionada posible elevación de brazos (tras detectarse cierta 'ingravidez') [30], la observación del descenso de las aceleraciones mientras se da la caída [31], la detección de un pico con un umbral llamativamente alto, simbolizando el impacto contra el suelo [32] y el periodo tras este impacto, o el análisis continuo de las diferentes fases de la caída (previo a la caída, durante el momento crítico de la misma, dado el impacto y tras la caída) [33].

Otro posible enfoque es, dadas las mediciones del acelerómetro, no basar la detección solamente en el umbral que alcanza la aceleración total, sino calcular otras medidas obtenibles a través de los valores del acelerómetro, como puede ser el ángulo de la aceleración [34]. Este es el enfoque que se utilizará en el proyecto, por lo que se aporta más detalle en la sección 1.4.4: Enfoque seleccionado para el proyecto.

Una mejora que se realiza sobre este enfoque es el Modelo Oculto de Márkov [35], un modelo estadístico en el que el sistema a modelar se trata de un proceso de Márkov [36] en el cual los parámetros son desconocidos. En este modelo, el estado del mismo no es directamente observable, pero sí lo son variables que están influenciadas por el estado oculto. En el caso que nos concierne, no sabemos si el modelo se encuentra en estado de 'caída' o de alguna otra actividad cotidiana, pero podemos observar los valores recogidos por el acelerómetro, que se encuentran directamente influenciados por el hecho de estar cayendo o no. Por lo tanto, dadas las variables observables, se puede obtener la probabilidad para el posible estado oculto [37] [34] [38].

1.4.2. Enfoque basado en aprendizaje automático

El funcionamiento de los algoritmos basados en aprendizaje automático se basa en implementaciones mucho más complejas que los umbrales, como pueden ser redes neuronales o máquinas de vectores de soporte. Estas, aparte de tener que ser desarrolladas, necesitan ser entrenadas con una gran cantidad de datos para permitir a los algoritmos haber obtenido el suficiente aprendizaje como para poder obtener buenos resultados. No obstante, esta gran complejidad hace que este enfoque sea mucho más robusto que la aproximación mediante umbrales, pues el modelo puede aprender qué patrones corresponden a actividades cotidianas y cuáles a caídas verdaderas, reduciendo así el número de falsos positivos.

Al tratarse de mecanismos de aprendizaje automático, se pueden tomar una gran variedad de enfoques diferenciados principalmente por la técnica utilizada (regresión, árboles de decisiones, naive Bayes, K vecinos más cercanos, máquinas de vectores soporte...). Por lo general, cuanto más complejo y completo sea el método utilizado, mejores resultados obtendrá [39]. Dos posibles ejemplos son el uso de XGBoost para un aprendizaje basado en clasificación y regresión [40] y el conjunto de varios algoritmos basados en aprendizaje mediante árboles [41].

Cabe destacar que una gran cantidad de algoritmos basados en aprendizaje automático también usan umbrales. Primero, los datos tomados se filtran mediante umbrales para detectar una posible caída y, después de esta evaluación inicial, se hace uso del modelo de aprendizaje automático para distinguir entre caídas reales y aquellas que han sido falsos positivos.

1.4.3. Uso del giroscopio

Por lo general, las mediciones del giroscopio tan solo se utilizan para complementar las medidas que se toman con el acelerómetro, sea cual sea el enfoque utilizado (umbrales [42] o aprendizaje automático [43]).

En un principio podría parecer que el tomar medidas de diferentes sensores mejoraría los resultados obtenidos. Sin embargo, en el caso del giroscopio, es difícil identificar qué giros corresponden a una posible caída y cuáles a una simple rotación del dispositivo, más aún en relojes inteligentes, los cuales al llevarse en la muñeca se someten a rotaciones constantes. Un estudio realizado en la Universidad

de Málaga [44] corrobora esta idea.

En dicho estudio se obtuvieron resultados de múltiples algoritmos con y sin uso de giroscopio, tanto para detección por umbrales como para detección mediante aprendizaje automático. Comparando los resultados obtenidos, se llega a la conclusión de que la información aportada por el giroscopio solo mejora la efectividad en ciertos casos particulares, pero en la gran mayoría de ellos no solo no realiza un gran aporte, sino que además puede llegar a empeorar los resultados obtenidos. Por ello, los autores recomiendan utilizar tan solo las medidas tomadas por el acelerómetro, pues de esta forma se reduce la complejidad de los algoritmos y el costo energético de los dispositivos.

1.4.4. Enfoque seleccionado para el proyecto

Debido a que el uso del giroscopio no aporta datos significativos e incluso podría empeorar los resultados (como se ha documentado en la sección anterior), para la realización de este proyecto tan solo se hará uso de las medidas tomadas por el acelerómetro de 3 ejes del reloj inteligente.

En cuando al enfoque utilizado, la opción para obtener las mejores detecciones se trata de utilizar un algoritmo que haga uso de un modelo de aprendizaje automático, pues la gran mayoría de estos algoritmos consiguen mejores resultados que aquellos basados en umbrales [39]. Aun así, dadas las limitaciones de este proyecto, no se cuenta con los recursos necesarios para poder implementar un modelo de detección y entrenarlo con la suficiente precisión como para alcanzar buenos resultados. Por ello, se utilizará un algoritmo basado en umbrales. Con el objetivo de contrarrestar la detección de falsas caídas, se añadirá una pequeña confirmación para que el usuario pueda negar la caída, reduciendo el número de avisos falsos que se enviarían al servidor.

En concreto, se utilizará un algoritmo de doble comprobación, donde, para detectar una caída, dos medidas han de superar el umbral establecido, obteniendo de esta forma mejores resultados. Debido a que ambas condiciones se deben cumplir, se reduce la probabilidad de detección de falsas caídas debido al mal ajuste del umbral. Para la realización del proyecto se van a tener en cuenta dos enfoques distintos, pues uno de ellos (el preferido) cuenta con una desventaja que puede ocasionar problemas, por lo que se plantea una alternativa.

1.4.4.1. Enfoque preferido: GSVM y Ángulo.

En este caso, se utilizarán las medidas del ángulo de la aceleración (θ) y la magnitud de la suma de vectores de aceleración ponderada por la gravedad (*GSVM*). Se utilizan estas medidas pues, dado un estudio realizado en Corea [37], son las que mejores resultados obtienen entre una selección de medidas fácilmente calculables mediante las mediciones del acelerómetro.

Dadas las aceleraciones recogidas por el acelerómetro en tres ejes (x, y, z) se puede calcular el ángulo de la aceleración (θ) mediante la siguiente fórmula:

$$\theta = tan^{-1}\left(\frac{\sqrt{y^2+z^2}}{x^2}\right) \times \frac{180}{\pi}$$

Esta fórmula tiene en cuenta que el eje que se toma como referencia como aquel en que actúa la gravedad es el eje X. Ha de comprobarse si ese es el caso para los relojes inteligentes.

Para calcular la magnitud de la suma de vectores de aceleración ponderada por la gravedad (*GSVM*) primero se ha de calcular la magnitud de la suma de vectores de aceleración normal (*SVM*):

$$SVM = \sqrt{x^2+y^2+z^2}$$

Y ponderarla por la gravedad:

$$GSVM = SVM \times \frac{\theta}{90}$$

Una vez obtenidas estas medidas se comprobará si superan el umbral establecido para darse una caída. Primero, la medida de *GSVM* ha de superar su umbral, tras lo cual se han de hacer repetidas medidas de (θ). Una vez se hayan hecho las mismas, dependiendo del número de ellas que hayan superado el umbral se determinará si se ha dado caída o no.

El propio estudio del que se han tomado las medidas [37] llega a la conclusión de que los mejores umbrales para las medidas son $GSVM = 1g$ y $\theta = 60º$, sin embargo, dadas las diferencias en las características del dispositivo utilizado en el estudio y un smartwatch, se tendrá que llevar un análisis para determinar qué umbrales son apropiados en relojes inteligentes. Además, la referencia continúa con un paso más allá para determinar caídas, la implementación de un Modelo Oculto de Markov, el cual mejora considerablemente los resultados. No obstante, por los mismos motivos por los que no se desarrolla un modelo de aprendizaje automático, tampoco se implementará este modelo. De esta forma, manteniendo una comprobación de umbrales directa, se mantiene la simplicidad temporal, de implementación y de ejecución.

Este enfoque tiene una gran dependencia con el ángulo en el cual está colocado el dispositivo, lo cual puede suponer un problema en relojes inteligentes. Debido a que estos se sitúan en la muñeca, se someten a rotaciones constantes y en cualquier momento pueden estar rotados de cualquier forma. Esto supone un problema, pues una rotación de muñeca podría suponer la detección de falsas caídas, lo cual lleva a necesitar un umbral muy ajustado para evitar las mismas. Cuando se da una caída verdadera, la muñeca también puede encontrarse en cualquier posición, lo cual, sumado a la poca flexibilidad necesitada para el umbral, puede llevar a que bastantes caídas no se detecten correctamente. Por ello, se plantea un algoritmo alternativo.

1.4.4.2. Enfoque alternativo: SVM e Impacto.

En este caso, se utilizarán medidas tan solo de la magnitud de la suma de vectores de aceleración (*SVM*), pero haciendo uso de dos umbrales. Estos dos se tratan de uno bajo (próximo a cero), el cual representa la aceleración que se da en caída libre, y otro alto, el cual representa el impacto final de la caída.

Dadas las aceleraciones recogidas por el acelerómetro en tres ejes (x, y, z) se puede calcular la magnitud de la suma de vectores de aceleración (*SVM*) mediante la siguiente fórmula:

$$SVM = \sqrt{x^2 + y^2 + z^2}$$

Una vez obtenidas esta medida se comprobará si supera el umbral establecido para darse una caída. Primero, se ha de superar un umbral bajo, el cual representa que se ha detectado caída libre. Después, se habrá de superar el umbral alto (el cual representa el impacto) en un corto periodo de tiempo tras la detección de la caída libre.

Este enfoque es mucho más propenso a la falsa detección de caídas, pues, en caso de que los umbrales no sean muy estrictos, acciones simples como sentarse o tumbarse (sobre todo cuanto más baja esté la superficie y más brusco sea el movimiento) pueden desencadenar la detección de una caída. Sin embargo, no cuenta con el problema de la variabilidad de los ángulos y puede obtener buenos resultados independientemente de la manera en que se lleve puesto el dispositivo.

1.5. Aplicación que se toma como punto de partida

Para la realización de este proyecto se toma como punto de partida una aplicación de smartwatch ya existente. Esta se trata de un proyecto realizado como un TFG por lo que se encuentra totalmente documentada, desde su análisis hasta su implementación final, en la memoria de dicho TFG [8]. Por ello, en esta sección tan solo se documentarán brevemente sus funcionalidades, pues su implementación se documentará posteriormente si ha de ser modificada y, para mayor detalle, se puede consultar la memoria pertinente.

La aplicación está pensada para que su primera ejecución la realice un administrador debido a que esta necesita una serie de configuraciones para poder comenzar su funcionamiento. Estos pasos iniciales se encuentran detallados en el Apéndice C: Manual de Uso, pues su funcionamiento se mantien igual tras los añadidos en la app. Una vez realizada la conexión, el reloj descarga una serie de datos que se encuentran en el servidor, los cuales sirven para realizar las configuraciones inciales (como, por ejemplo, el tiempo de espera entre comprobaciones de sensores).

Justo después de finalizarse la descarga de datos necesarios, se mostrará una notificación persistente en el reloj, la cual al ser desplegada expone una pantalla similar a la que aparece en la Figura 1.6 (su apariencia puede variar dependiendo del dispositivo). El ver esta pantalla significa que la app

ha comenzado su ejecución constante en segundo plano y que las funcionalidades de monitorización se están ejecutando.

Figura 1.6: Captura de la pantalla que se muestra tras presionar la notificación

Las funcionalidades principales de la app son un servicio para notificar emergencias de manera manual, así como monitorizaciones sobre la ubicación del mismo y posible retirada del dispositivo. A continuación se detalla más a fondo estas funcionalidades.

1.5.1. Botón de emergencia

La notificación permanente (Figura 1.6) contiene un botón de emergencia. Al pulsar este botón se envía un aviso de emergencia al centro de monitorización, junto con el cual se envía la ubicación GPS para poder localizar al usuario que ha sufrido la emergencia. El uso de este botón se encuentra detallado en el Apéndice C: Manual de Uso.

1.5.2. Monitorización de la ubicación

Cada cierto intervalo de tiempo (configurable por el administrador desde el servidor) se recoge la ubicación GPS del reloj y se envía al servidor. En caso de haber errores de red y no ser posible la conexión para el envío de datos, estos se almacenan en una BD local en el reloj para enviarse una vez recuperada la conexión.

1.5.3. Monitorización de retirada del dispositivo

Cada cierto intervalo de tiempo (también configurable por el administrador desde el servidor) se comprueban los valores recogidos por los diferentes sensores y se ejecuta un algoritmo para detectar si el usuario se ha retirado el reloj. Con el objetivo de garantizar la robustez de este algoritmo, hay dos formas de detectar esta retirada: mediante sensores de luz y mediante el acelerómetro (a efectos de este proyecto el método utilizado es indiferente).

En caso de haberse detectado una retirada del reloj, se pausa el envío de ubicación al servidor, pues la aplicación entra en un bucle de comprobación de retirada hasta que se confirme que el usuario se ha colocado el reloj de nuevo. Tras asegurar un período de tiempo en el cual el reloj no está equipado, se envía una notificación de retirada al servidor.

1.6. Tecnologías empleadas

A lo largo de esta sección se describirán las principales tecnologías utilizadas para la realización de este proyecto. Al tratarse de una extensión a un proyecto previo, no se ha tenido que llevar a cabo una investigación sobre qué tecnologías escoger, si no una adaptación a aquellas ya utilizadas.

En resumen, se ha ha desarrollado una aplicación para WearOS mediante Java. Para implementar la base de datos local se ha hecho uso de la librería Room de Google.

1.6.1. WearOS

Es el sistema operativo sobre el cual se tendrá que realizar la aplicación. Está basado en Android, pero hoy en día también cuenta con compatibilidad para iOS.

Figura 1.7: Logo de WearOS [45]

Se anunció en Marzo de 2014 con el nombre de «*Android Wear*» [46] y en 2018 tomó el nombre de «*WearOS*» [47]. A día de hoy la versión estable de WearOS es la 3, basada en Android 11, pero la versión 4 (basada en Android 13) ya está disponible en modo desarrollador[48].

Al tratarse de una adaptación de Android para enfocarse en relojes inteligentes y otros dispositivos portables, mantiene una arquitectura similar. Sin embargo, no es exactamente igual debido a que estos dispositivos tienen un funcionamiento intrínsecamente diferente al de los teléfonos inteligentes. Estas diferencias vienen dadas por un muy reducido tamaño de pantalla, carencia de puertos de entrada y, en algunos dispositivos, conectividades reducidas.

Para solventar estas desventajas, WearOS está pensado para funcionar gracias a un emparejamiento con un teléfono inteligente. De esta forma se pueden mantener funcionalidades clave a pesar de no contar con componentes con los que sí cuentan los smartphones.

1.6.2. Android

Se trata de un sistema operativo basado en el núcleo de Linux, adaptandose especialmente a dispositivos móviles con pantalla táctil. Es el sistema operativo móvil más utilizado en el mundo [49].

Figura 1.8: Logo de Android [50]

Fue desarrollado por *Android Inc.* (empresa que fue adquirida por Google en 2005 [51]) y se presentó en 2007, a la vez que la fundación de la *Open Handset Alliance* [52] para el progreso y estandarización en el desarrollo de dispositivos móviles.

El sistema operativo se puede describir como una pila de software, es decir, está dividido en capas con distintos niveles de abtracción [53]. Estas capas son las siguientes:

Aplicaciones Se trata de la capa de más alto nivel. Son aquellas aplicaciones que utiliza el usuario final, como SMS, navegadores, etc. Por lo general se desarrollan en Java u otros lenguajes de programación basados en el mismo.

Framework de Aplicaciones Se trata de un marco de trabajo basado en APIs que Android provee para facilitar el desarrollo de aplicaciones finales (por ejemplo, el sistema de notificaciones).

Bibliotecas Escritas en C y C++, sirven para dar soporte al framework de aplicaciones y son utilizadas por varios componentes del sistema (por ejemplo, bibliotecas de gráficos o SQL).

Runtime de Android Se trata del entorno de ejecucuón sobre el cual se ejecutan diferentes máquinas virtuales para cada aplicación. Originalmente se utilizada *Dalvik* como máquina virtual, posteriormente se actualizó a *ART* (Andorid RunTime), pero se sigue manteniendo el formato ejecutable *.dex*.

Núcleo Linux Es la base de Android. Se trata de un núcleo de código abierto que aporta soporte para los sitemas básicos como gestión de memoria, subprocesos o seguridad. Actua como capa puente entre el hardware y el software.

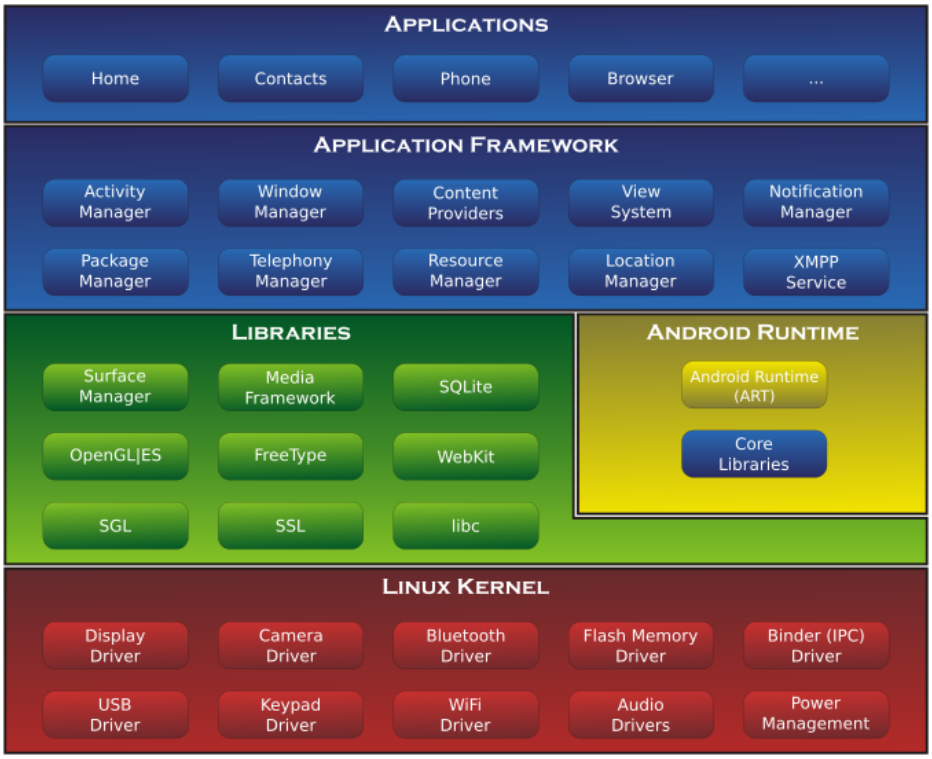

Figura 1.9: Arquitectura de Android [53]

1.6.3. Java

Se trata de un lenguaje de programación multiplataforma orientado a objetos con el cual están desarrolladas la gran mayoría de aplicaciones en sistemas Android.

Figura 1.10: Logo de Java

Fue desarrollado originalmente por *Sun Microsystems* y posteriormente adquirido por *Oracle* [54]. Actualmente se encuentra en la versión Java SE 20. [55].

Una de las características más importantes de Java es su portabilidad, basada en el principio *"Write Once, Run Anywhere"*. Se proporciona un lenguaje que resulta independiente a la plataforma y un entorno de ejecución, JVM (máquina virtual de java). JVM interpreta los binarios (bytecode) de las aplicaciones Java y los ejecuta en cualquier SO, sin necesidad de recompilar el código [56].

Otras características destacables de Java son su orientación a objetos, manejo automático de la memoria (con un recolector de basura), características de seguridad integradas en el lenguaje y permitir la ejecución de varios hilos en paralelo, entre otras. [57]

1.6.4. Room

Se trata de una biblioteca de persistencias que proporciona una abtracción sobre SQLite para Android. Es la forma recomendada de tratar con bases de datos locales según la propia página de referencia de Android Developers [58].

Su utilización se basa en el uso de clases con anotaciones, las cuales permiten comunicarse con la base de datos mediante objetos completos ya formados, tanto para la actualización de la misma como para la descarga de datos.

Su arquitectura está fromada por los siguientes componentes principales:

Entities son clases que definen las tablas de la base de datos.

DAOs son interfaces que proporcionan los métodos necesarios para la interacción con la BD.

Room Database es la clase que representa la BD y sirve como punto de conexión con la misma.

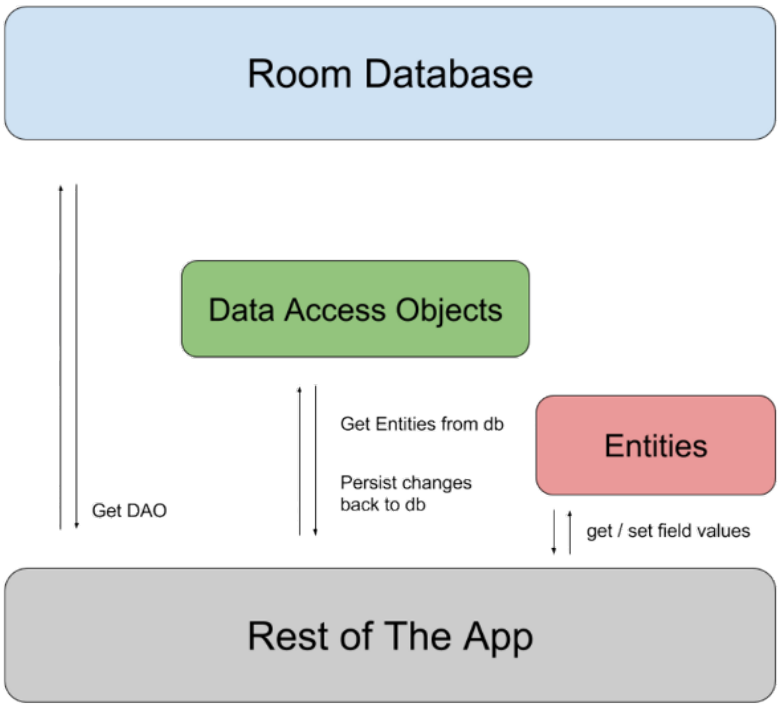

Figura 1.11: Arquitectura de componentes de Room [58]

1.7. Objetivos

El objetivo general de este proyecto es, dada la aplicación que se toma como punto de partida, añadir una nueva funcionalidad, la cual se encargue de detectar caídas. De esta forma se contará con un sistema de monitorización de personas dependientes completo. Para completar este objetivo se pueden tener en cuenta las siguientes metas:

- Investigar e implementar el algoritmo de detección de caídas que mejor se ajuste al proyecto.
- Notificar de la caída a un servidor.
- Pausar de la detección de caídas en caso de que el dispositivo se haya retirado.
- Diferenciar entre caídas leves (el usuario sigue consciente) y caídas graves (el usuario se encuentra inconsciente).

Además, debido al tamaño que toma la aplicación al introducir esta detección, se introduce el objetivo de permitir elegir qué monitorizaciones se quieren llevar a cabo y cuáles no, para un ahorro de la batería del dispositivo y mayor fluidez de ejecución.

1.8. Metodología

El proyecto se trata de una sucesión de pequeños objetivos que se construyen sobre los anteriores para llegar a una meta final, teneindo en cuenta riesgos que pueden afectar directamente al cumplimiento de estos objetivos en el tiempo estimado. Dada esta naturaleza se podrían utilizar metodologías ágiles o iterativas e incrementales. En este caso se escogerá la segunda opción pues las metodologías ágiles están pensadas para grupos de varias personas y este proyecto cuenta con tan solo una desarrolladora.

En la metodología de trabajo iterativa e incremental el proyecto se devide en varios periodos de tiempo de duraciones variables (iteraciones). En cada iteración se trabaja para alcanzar un objetivo concreto, el cual proporcione una nueva funcionalidad que esté lista para ser probada. De esta forma se evita la posibilidad de caer en malas prácticas como el *gold-plating* (implementar funcionalidades que no son verdaderamente útiles con el mero objetivo de tener mayor contenido en la aplicación) o caer en un bucle de correcciones sin progresar a siguientes funcionalidades, pues cada funcionalidad tiene su tiempo límite. Además, en caso de que se diesen cambios en los requisitos iniciales, se pueden tener en cuenta en un pequeño análisis entre iteraciones, de tal forma que se les pueda dedicar una iteración personalizada en caso de que fuera necesario.

Para la realización de este proyecto se utilizarán varias iteraciones de duración entre una y cinco semanas, realizando en cada una de ellas los pasos mostrados en la Figura 1.12 con la meta de cumplir el objetivo concreto de cada iteración. Los objetivos de las diferentes iteraciones y el trabajo

realizado para completarlos se detallarán posteriormente en las secciones de Planificación (Capítulo 2) y Descripción de las iteraciones (Capítulo 3), respectivamente.

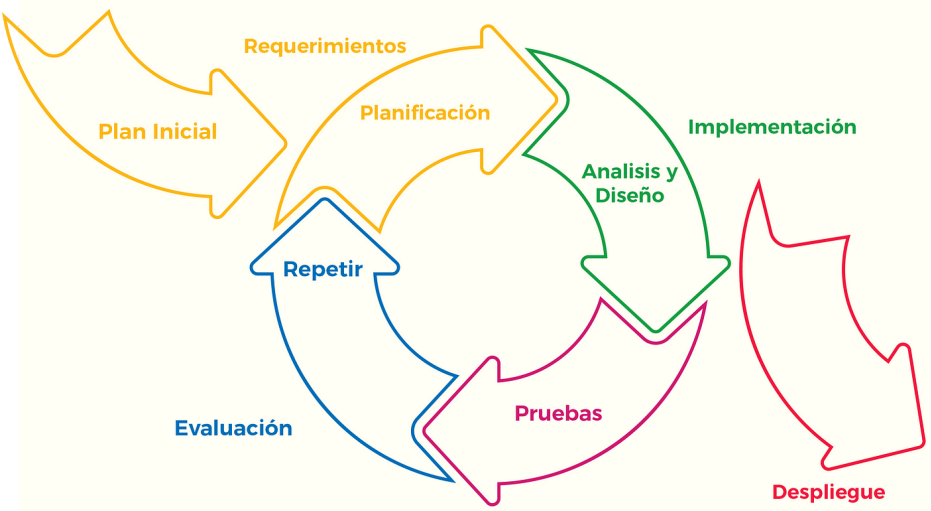

Figura 1.12: Pasos de la Metodología Iterativa e Incremental [59]

1.9. Estructura de la memoria

Este documento se divide en los siguientes capítulos:

- **Introducción:** se presenta el proyecto de una forma general y de alto nivel. Incluye una descripción del contexto y motivación para realizar el proyecto, un breve estudio del estado del arte, tecnologías y metodología utilizadas para llevar a cabo del proyecto y, por último, esta misma sección describiendo brevemente la estructura de la memoria.
- **Planificación:** se detalla una planificación inicial del proyecto tanto temporalmente, con una breve descripción del trabajo a realizar en cada iteración, como en cuanto a costes. Para ello se tienen en cuenta los riesgos del mismo y herramientas necesarias para realizarlo, tanto software como hardware. Por último, se aporta un resumen de cómo se realizó el proyecto finalmente y se compara con esta planificación inicial.
- **Descripción de las iteraciones:** descripciones más en detalle del trabajo realizado en cada una de las iteraciones. Se documenta cómo se realizó cada iteración, junto con los obstáculos encontrados y cómo se solventaron.
- **Estado final de la aplicación:** descripción completa del estado final de la aplicación, incluyendo análisis y decisiones de diseño tomadas para realizar la misma y diagramas para una mejor comprensibilidad.
- **Pruebas:** se describen las pruebas realizadas (y resultados obtenidos) para poder llegar a un nivel de aceptación de la aplicación.

- **Conclusiones:** resumen final del trabajo realizado, aprendizaje dado a lo largo del desarrollo del proyecto y posible trabajo futuro para pulir resultados.
- **Apéndices:** documentación adicional como aclaración de acrónimos, manuales de despliegue y uso, etc.
- **Bibliografía:** lista de referencias bibliográficas utilizadas durante el desarrollo del proyecto.

Capítulo 2

Planificación

2.1. Planificación inicial

2.1.1. Organización temporal

Según lo estipulado en la guía docente, al Trabajo de Fin de Grado se le atribuyen 300 horas. Con el fin de ajustarse a esto, se ha estimado un trabajo de 5 días a la semana donde se dediquen 5 horas de trabajo diario (25 horas semanales). Además, si fuera necesario se podría ampliar el trabajo a los fines de semana, añadiendo 4 horas extra semanales.

Siguiendo esta planificación temporal, se tratará de 12 semanas de trabajo, las cuales estarán comprendidas entre el Lunes 9 de Octubre de 2023 y el Viernes 29 de Diciembre de 2023. Esto resulta en un total de 348 horas, teniendo en cuenta las posibles horas extra de los fines de semana. Considerando que durante estas fechas hay bastantes días festivos, se estima que se alargue el proyecto a la primera quincena de Enero de 2024.

Se tendrá en cuenta un período previo al comienzo del trabajo, el cual tendrá lugar entre el Lunes 11 de Septiembre de 2023 y el Lunes 9 de Octubre de 2023. Durante este tiempo se abordará la planificación inicial, investigación sobre proyectos ya existentes con la misma funcionalidad, dispositivos donde se utilizará el proyecto y familiarización y adaptación al entorno de trabajo para el desarrollo de aplicaciones de Wear OS.

Además, durante este periodo de tiempo se realizará la descarga y familiarización con el proyecto que se toma como punto de partida, así como un pequeño análisis de la facilidad para incorporar nuevas funcionalidades al mismo.

Para llevar a cabo el proyecto se utilizará una metodología iterativa incremental. Esta constará de las siguientes iteraciones:

1. Punto de inicio del proyecto, tendrá una duración estimada de 1 semana, comenzando el 9 de Octubre. Tratará de recopilar toda la información obtenida en el período previo y documentarla y

estructurarla adecuadamente.

2. Comenzará el 16 de Octubre y contará con una duración estimada de 5 semanas. Se llevará a cabo la investigación e implementación del algoritmo de detección de caídas.

3. Comenzará el 20 de Noviembre y tendrá una extensión estimada de 2 semanas. Se desarrollarán funcionalidades adicionales a la detección de una caída (pausar la detección de caídas si el dispositivo no estaba siendo utilizado y detección de consciencia tras la caída).

4. Comenzará el 4 de Diciembre y durará aproximadamente 2 semanas. Durante esta iteración se implementará la posibilidad de activar/desactivar características de la aplicación de manera que solo se monitorice lo seleccionado. Además, se llevará a cabo un pequeño estudio de la eficacia del algoritmo de detección de caídas con datos teóricos y en supuestos reales.

5. Última iteración del proyecto, comenzará el 18 de Diciembre y durará aproximadamente 2 semanas. Se realizarán las correcciones finales, completando el trabajo restante y revisando el proyecto con el fin de prepararlo para la entrega.

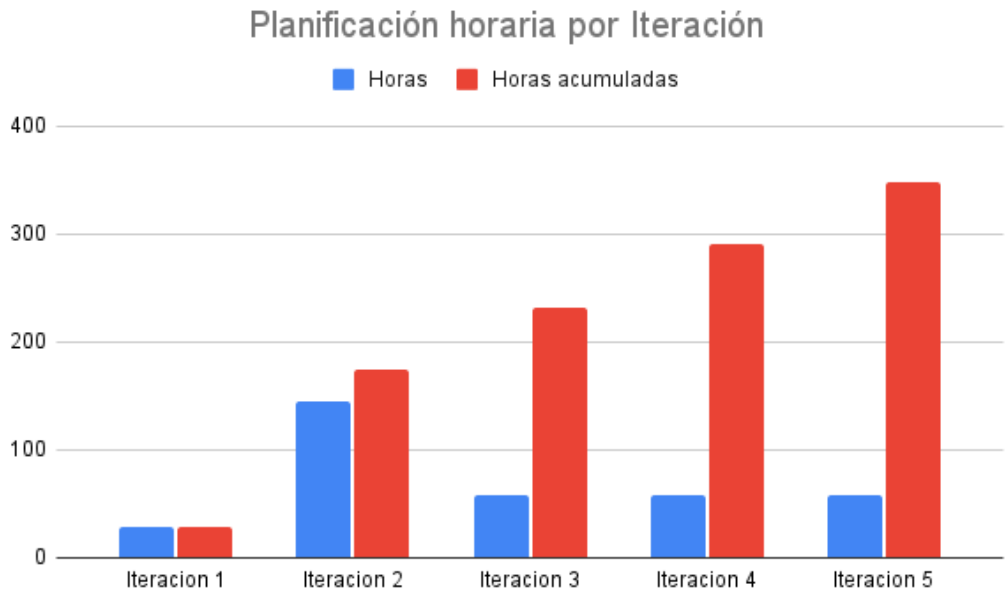

Figura 2.1: Número de horas por iteración

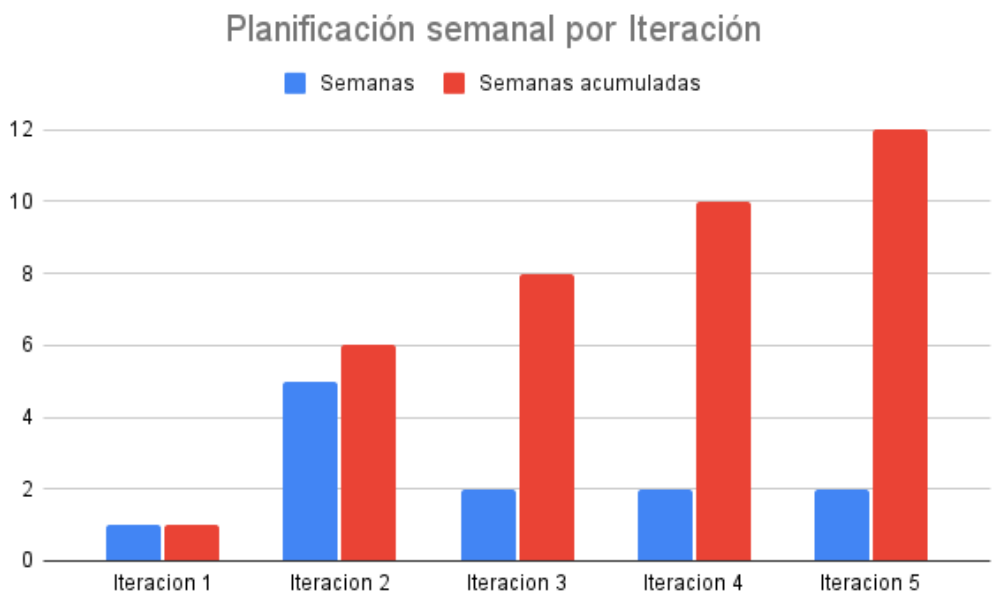

Figura 2.2: Número de semanas por iteración

2.1.2. Análisis de Riesgos

El éxito o fracaso de un proyecto siempre está inherentemente vinculado a cierto grado de incertidumbre. Una forma de mejorar las probabilidades de éxito es identificar qué riesgos pueden suponer retrasos en la entrega del proyecto. En esta sección se tratarán de evaluar los riesgos más prevalentes haciendo uso de dos tablas, las cuales constan de los siguientes apartados:

ID Número que identifica al riesgo.

Riesgo Nombre con el cual referirse al riesgo.

Descripción Descripción corta del riesgo.

Probabilidad Probabilidad de que el riesgo se manifieste.

Retraso Estimado Retraso (en días) que el riesgo provocaría en caso de manifestarse.

Exposición Gravedad del riesgo. Se calcula con el múltiplo de la probabilidad por el retraso.

Reducción Acciones a tomar para reducir la probabilidad de que el riesgo se manifieste.

Mitigación Acciones a tomar para reducir el impacto del riesgo una vez se ha manifestado.

ID	Riesgo	Descripción	Probabilidad	Retraso Estimado (días)
01	Indisposición alumna	La alumna sufre una enfermedad que le impide trabajar.	0.1	5
02	Indisposición tutor/es	El/los tutor/es sufre/n una enfermedad que les supone cogerse una baja temporal.	0.1	7
03	Falta de experiencia	Al trabajar con tecnologías en la que no se tiene experiencia se necesitará más tiempo para resolver problemas.	0.25	7
04	Cambio o poca claridad en los requisitos	Si los requisitos del proyecto cambian o no son lo suficientemente precisos se perderá tiempo en implementar cosas que después serán modificadas.	0.2	15
05	Inutilización de los equipos de trabajo	Si algún dispositivo tuviera fallos o se rompiera no se podrá trabajar hasta que se reponga.	0.1	7
06	Incompatibilidad con el proyecto existente	Al partir de un proyecto ya existente puede que haya funcionalidades que colisionen, por lo que habrá que modificar lo existente para compatibilizarlo con lo nuevo o modificar el diseño de las nuevas funcionalidades.	0.25	15
07	Pérdida de datos	Si no se guarda apropiadamente el trabajo realizado habrá que recuperar trabajo ya realizado.	0.1	5

Tabla 2.1: Riesgos del proyecto. Probabilidad y retraso estimado.

ID	Riesgo	Exposición	Reducción	Mitigación
01	Indisposición alumna	0.5	Tener precaución para evitar contagios y mantener una vida saludable.	Retrasar la fecha de entrega del proyecto; emplear horas extra para suplir las horas perdidas.
02	Indisposición tutor/es	0.7	Establecer formas de comunicación telemática para mantener el contacto.	Planificar el resto del TFG con otro tutor académico.
03	Falta de experiencia	1.75	Establecer un periodo previo al desarrollo del TFG para aprender las tecnologías necesarias.	Aumentar la cantidad de horas dedicadas al proyecto para suplir la falta de experiencia.
04	Cambio o poca claridad en los requisitos	3	Reunirse habitualmente con el/los tutor/es para asegurar el entendimieneto de los requisitos y que la implementación se adapte a los mismos.	Retrasar la fecha de entrega del proyecto; emplear horas extra para ajustarse a los nuevos requisitos.
05	Inutilización de los equipos de trabajo	0.7	Procurar tener varios dispositivos disponibles y tratar adecuadamente los que se estén utilizando.	Intentar avanzar trabajo en algo que no requiera el dispositivo dañado; retrasar el proyecto hasta que se cuente con nuevos dispositivos.
06	Incompatibilidad con el proyecto existente	3.75	Establecer un periodo previo al desarrollo del TFG para familiarizarse con el proyecto existente; diseñar la nueva funcionalidad de forma que resulte compatible.	Aumentar la cantidad de horas dedicadas al proyecto para realizar las modificaciones necesarias.
07	Pérdida de datos	0.5	Asegurarse de que el progreso es guardado con regularidad y mantener copias de seguridad del mismo.	Retrasar la fecha de entrega del proyecto; emplear horas extra para rehacer el trabajo perdido.

Tabla 2.2: Riesgos del proyecto. Exposición y medidas.

2.1.3. Entorno de trabajo

En esta sección se procede a describir las herramientas y dispositivos utilizados para el desarrollo del proyecto.

2.1.3.1. Herramientas utilizadas

Android Studio IDE oficial para el desarrollo de aplicaciones Android, creado por Google. [60]

Gitlab Repositorio remoto para el almacenamiento de código y control de versiones del proyecto. Se utilizará la instancia de la Escuela de Ingeniería Informática de la UVa. [61]

Astah Professional Herramienta para realizar el modelado y diseño del proyecto mediante diagramas UML. Se utilizará bajo la licencia de la UVa. [62]

Overleaf Editor web de texto en línea para realizar la memoria del proyecto en formato LaTeX. [63]

Teams Aplicación de comunicación desarrollada por Microsoft. Permite el almacenamiento de archivos y proporciona canales de comunicación tanto por texto como por voz. Utilizado para la comunicación con los tutores. [64]

Chrome Navegador web gratuito desarrollado por Google. [65]

2.1.3.2. Dispositivos utilizados

Equipo	**ROG Zephyrus G14**
Sistema Operativo	Windows 11 Home
Arquitectura	x64
Procesador	AMD Ryzen 7 4800HS
Disco Duro	1TB
Memoria RAM	16GB

Tabla 2.3: Características Ordenador Portátil

Equipo	Samsung Galaxy Watch4 Classic 4G
Sistema Operativo	WearOS
Pantalla	AMOLED 1,4"
Procesador	Samsung Exynos W920
Disco Duro	16GB
Conectividad	Wi-Fi, Bluetooth, NFC, LTE
Sensores	Acelerómetro de 3 ejes, Barómetro, Giroscopio, Sensor geomagnético, Sensor de luz, Sensor óptico de frecuencia cardíaca, Sensor cardíaco eléctrico, Sensor de análisis de impedancia bioeléctrica, Sensor Hall
Memoria RAM	1.5GB

Tabla 2.4: Características Smartwatch Samsung

Equipo	Oppo Watch 41mm (Wi-Fi)
Sistema Operativo	WearOs
Pantalla	AMOLED 1,91"
Procesador	Snapdragon Wear 3100
Disco Duro	8GB
Conectividad	Wi-Fi, Bluetooth, NFC
Sensores	Acelerómetro de 3 ejes, Barómetro, Giroscopio, Sensor geomagnético, Sensor óptico de frecuencia cardíaca, Sensor de capacitancia, Sensor de luz ambiental
Memoria RAM	1GB

Tabla 2.5: Características Smartwatch Oppo

2.1.4. Estimación del Coste

A lo largo de esta sección se realizará una estimación del coste para la realización del proyecto. Se tendrán en cuenta costes tanto humanos, como informáticos y relativos al espacio de trabajo. No se tendrá en cuenta el coste de las herramientas utilizadas, pues la gran mayoría son gratis y aquellas que son de pago se pueden utilizar bajo la licencia de la UVa.

2.1.4.1. Costes del salario de los participantes

Al tratarse de un Trabajo Final de Grado realizado con el fin de obtener una titulación universitaria, el trabajo de la estudiante no se verá remunerado. Sin embargo, lo tomamos en cuenta para el cálculo del coste total. Tomando como referencia que el salario medio para un desarrollador Android Junior en España es de 2000€ al mes [66], siendo una jornada completa 160 horas mensuales, tenemos un salario de 12'5€/hora. Como el tiempo estimado para realizar este proyecto son 300 horas, tenemos que el salario correspondiente serían 3750€ brutos.

También se tiene en cuenta el salario de los tutores del proyecto. Estimamos un salario promedio de 22€/hora para profesores de la Universidad de Valladolid. Se estima 1 hora dedicada a revisiones y

reuniones por cada iteración, además de 2 horas en la revisión final de la memoria y preparación de la presentación, dando un total de 7 horas. Al tratarse de dos tutores, el coste total sumaría 308€.

2.1.4.2. Coste de los dispositivos

Se tienen en cuenta los siguientes precios de los dispositivos que se utilizarán para la realización del proyecto:

- PC: ROG Zephyrus G14 = 1166'71€
- Smartwatch: OPPO Watch 41mm (Wi-Fi) = 289'99€
- Smartwatch: Samsung Galaxy Watch4 Classic 4G = 369'90€

Sin embargo, estos precios no suponen el coste real de los dispositivos para la realización del proyecto. No se trata de dispositivos de un solo uso, si no que han sido utilizados anteriormente y seguirán siendo utilizados tras la finalización del proyecto. Por ello, calculamos el coste real que los dispositivos suponen al proyecto mediante el cálculo de su coste amortizado.

Para calcular el coste amortizado de los dispositivos utilizamos la siguiente fórmula:

$$\frac{horasDeUsoParaElProyecto}{horasDeVidaUtil} \times precio$$

Considerando que el precio inicial del portátil, aproximando su vida útil a los 8 años y suponiendo que se va a usar a lo largo de todo el proyecto, su coste amortizado es el siguiente:

$$\frac{300}{8 \times 8760} \times 1166'71 = 4'99$$

En el caso de los Smartwatch, aproximamos su vida útil a los 4 años y suponemos que tan solo se van a utilizar aproximadamente un cuarto del tiempo total del proyecto. Su costes amortizados son los siguientes:

Samsung:

$$\frac{300 \div 4}{4 \times 8760} \times 369'90 = 0'79$$

Oppo:

$$\frac{300 \div 4}{4 \times 8760} \times 289'99 = 0'62$$

2.1.4.3. Coste del entorno de trabajo

Se ha de tener en cuenta el coste que supone mantener un espacio en el que poder trabajar, tanto costes fijos (alquiler) como costes variables (agua, gas, internet, luz, etc.). Debido a que el proyecto se desarrollará mayoritariamente en la casa de la estudiante y en lugares públicos como bibliotecas y salas de estudio no tomaremos en cuenta el coste que podría suponer el alquiler de una oficina.

En cuanto a los costes variables, tendremos en cuenta tan solo luz e internet al ser los más relevantes a la hora de desarrollar un proyecto informático.

Suponiendo que un ordenador Asus ROG consume 200Wh a máxima potencia [67], para este proyecto suponemos un consumo del 60 %, por lo tanto 120Wh o 0'12kWh. Dadas 300h para el desarrollo del proyecto, con un precio medio de la luz de 0'163€ por kWh [68], el coste total de la electricidad suma 5'87€.

Dado un precio aproximado de 30€/mes para una tarifa promedio de internet, habiendo 720h en un mes, la hora de internet tiene un coste de 0'042€. Para las 300 horas de proyecto esto supone un coste de 12'6€.

2.1.4.4. Coste total del proyecto

Teniendo en cuenta las estimaciones desarrolladas en las secciones anteriores se da el siguiente coste total:

Componente	**Coste (€)**
Salario estudiante	3750
Salario tutores	308
Portátil	4'99
Smartwatch Samsung	0'79
Smartwatch Oppo	0'62
Luz	5'87
Internet	12'6
Total	**4082'87**

Tabla 2.6: Coste total estimado del proyecto

2.2. Planificación final

En esta sección se realizará una comparación entre la estimación realizada en la planificación inicial y el tiempo final dedicado al proyecto.

En la planificación inicial se estimó una duración de 12 semanas de trabajo. Sin embargo, esta estimación se realizó sin tener en cuenta los múltiples festivos que se han dado a lo largo de la realización del proyecto. Además, debido a que este proyecto se ha realizado el primer cuatrimestre de un curso, no se contaba con una fecha límite demasiado estricta para la entrega del mismo, por lo que, cuando se ocasionaron imprevistos, se recurrió a posponer la fecha de entrega. A su vez, debido a disponibilidad personas, hay semanas en las cuales se dedicó mucho más tiempo al proyecto que otras, por lo que estimación total en semanas acaba careciendo de sentido. Debido a ello, para el resto de esta sección se tendrá en cuenta la planificación inicial en horas.

La planificación inicial suponía un total de 300 horas, siendo ampliables hasta un total de 348 horas. Al final se le han dedicado al proyecto un total de 315 horas, las cuales se distribuyen, por iteraciones, de la forma siguiente:

- Iteración 1. Se estimaron 25h (ampliables hasta 29h). Al final se le dedicaron 25h. Esta iteración no supuso ningún problema, pues se trataba de una iteración de documentación.

- Iteración 2. Se estimaron 125h (ampliables hasta 145h). Al final se le dedicaron 150h. Esta iteración supuso un problema en la planificación, debido a que, tras una reunión con los tutores, se acabaron modificando los requisitos iniciales de la aplicación, lo cual acabó llevando a la división en sub-iteraciones explicada en la sección 3.2. Más concretamente se dedicaron, en orden 75h a la sub-iteración 2.1, 40h a la 2.2, 15h a la 2.3 y, finalmente, 20h a la 2.4.

- Iteración 3. Se estimaron 50h (ampliables hasta 58h). Al final se le dedicaron 30h. Esta iteración tuvo que verse reducida debido a la ampliación de la iteración anterior. Aun con la reducción temporal, se llevaron a cabo todos los objetivos de la misma sin problemas, pues la estimación general era muy generosa.

- Iteración 4. Se estimaron 50h (ampliables hasta 58h). Al final se le dedicaron 70h. Esta iteración también supuso un problema debido a que la reestructura necesaria en el proyecto (detallada en la sección 3.4) fue mayor de lo que originalmente se había planteado y a que el código del proyecto tomado como punto de partida correspondiente a esta sección se encontraba disperso y fue difícil de comprender.

- Iteración 5. Se estimaron 50h (ampliables hasta 58h). Al final se le dedicaron 40h. Al tratarse de una iteración de documentación final, se esperaba poder reducir considerablemente su duración para compensar la extensión de las iteraciones 2 y 4. Aunque se consiguió reducir su duración, los requisitos identificados en una etapa tan avanzada del proyecto (descarga de la configuración mediante el servidor y soporte para diferentes algoritmos de detección de caídas), hicieron que al final se sobrepasara el tiempo estimado de 300h del proyecto.

A pesar de que las iteraciones 2 y 4 excedieron en gran medida el tiempo estimado para las mismas, el hecho de haber dado márgenes más amplios a las iteraciones 3 y 5 hizo que el tiempo dedicado al proyecto final no fuera desproporcionado. A pesar de que se excedieron las estimadas 300h ideales, se consiguió que su duración total estuviera dentro del margen de las 348h que se podrían haber utilizado.

En la figura 2.3 se muestra un gráfico que compara las horas estimadas con las horas reales utilizadas.

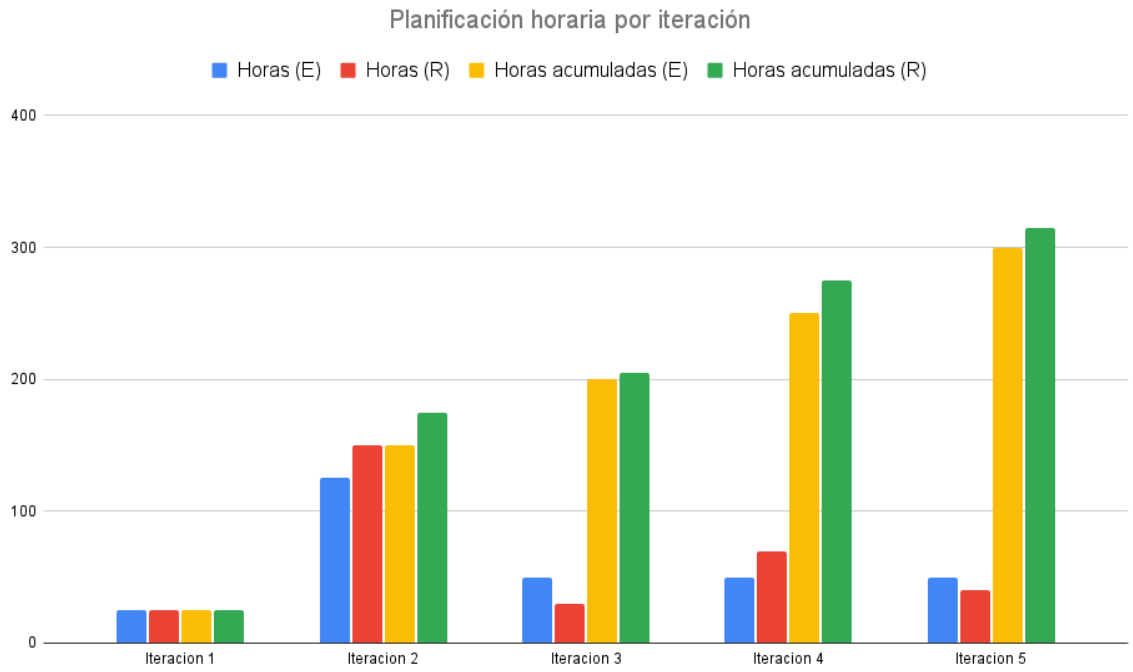

Figura 2.3: Comparación entre horas estimadas y horas reales dedicadas al proyecto

Capítulo 3

Descripción de las iteraciones

A lo largo de este capítulo se detallará el trabajo realizado en cada una de las iteraciones. Para ello se documenta análisis de requisitos de cada iteración, diseño realizado para cumplir los mismos y cómo se implementó finalmente. De esta forma se tendrá una recopilación de los avances y contratiempos encontrados a lo largo del desarrollo.

3.1. Iteración 1

Esta iteración inicial se trata simplemente de una iteración de redacción de la memoria, por lo que no se da ningún contratiempo a lo largo de la misma.

La semana dedicada a esta iteración se emplea en documentar la información recopilada a lo largo del periodo previo al comienzo del inicio del proyecto. Es decir, a lo largo de esta iteración se desarrollan las siguientes secciones de esta misma memoria:

- **Capítulo 1: Introducción.** Se desarrollan todas sus secciones salvo la Sección 1.4: Algoritmos de detección de caídas.
- **Capítulo 2: Planificación.** Se desarrolla la Sección 2.1: Planificación inicial.
- **Capítulo 3: Iteraciones.** Se desarrolla esta propia sección.
- **Capítulo 4: Estado final.** Se desarrolla la Sección 4.1: Análisis.

3.2. Iteración 2

A lo largo de esta iteración se realizó la búsqueda de un algoritmo de detección de caídas que se ajustara a los requisitos del proyecto. Tras unas semanas de trabajo en la misma, se detectó una

necesidad de dedicarle más tiempo, por lo que, tras una reunión con los tutores, se subdividió en las siguientes 'subiteraciones':

- **Iteración 2.1: Investigación.** Se llevó a cabo un trabajo de investigación sobre diferentes algoritmos de detección de caídas, lo cual llevó a la redacción de la sección 1.4: Algoritmos de detección de caídas. Además, se añadió más información a la sección 1.3: Estado de la cuestión.
- **Iteración 2.2: Implementación.** Se llevó a cabo la implementación del algoritmo seleccionado (detallada en la sección 5.5: Detección de la caída) y las pruebas necesarias para obtener los umbrales apropiados, así como definir el tiempo entre la lectura de los sensores.
- **Iteración 2.3: Comparaciones.** Teniendo en cuenta que el Smartwatch utilizado a lo largo del proyecto es un Samsung, se llevó a cabo una comparativa entre la detección de caídas dada una librería proporcionada por ellos y dado el algoritmo utilizado en el proyecto. Dicha comparativa se encuentra documentada en la sección 6.4: Comparación con la API de Samsung.
- **Iteración 2.4: Unificación.** Se incluye el algoritmo implementado en el proyecto que se toma como punto de partida, lo cual lleva al cambio de velocidad de muestreo de los sensores, detallado en la sección 5.2: Sensores.

3.3. Iteración 3

A lo largo de esta iteración se realizaron los pasos posteriores a la detección de una caída, los cuales son los siguientes:

- **Detección de consciencia.** Una vez se detecta una caída, se realizó un la diferenciación entre caídas leves (el usuario mantiene la consciencia), graves (el usuario pierde la consciencia) y falsas (se ha detectado caída, pero no se ha dado de verdad). Esto se detalla en la sección 5.6.
- **Envío del aviso al servidor.** Tras identificarse el tipo de caída, se realiza el envío de una notificación al servidor. Esto se realiza con el servicio ya implementado en el proyecto tomado como punto de partida, el cual está ligeramente detallado en la sección 5.3 y más en profundidad en la memoria del proyecto original [8].
- **Pausa de la detección de caídas.** Se llevó a cabo la implementación que permite pausar la detección de caídas en el momento en que se detecta la retirada del dispositivo. Esto se detalla más en profundidad en la sección 5.7.
- **Modo debug.** Se llevó a cabo la implementación de un modo de ejecución que permite recoger los datos correspondientes a las caídas (medidas tomadas por el acelerómetro y resultado del algoritmo) para poder realizar debug y refinar el mismo en un futuro. Se explica en más detalle en la sección 5.8.

3.4. Iteración 4

El objetivo principal de esta iteración es añadir la funcionalidad extra de poder elegir qué monitorizaciones se quieren realizar en la app. A la hora de realizar este añadido, se detectó la necesidad de refactorizar el proyecto. La documentación proporcionada en la memoria del mismo[8] no coincidía con la implementación final, por lo que, a lo largo de esta iteración, aparte de implementar la nueva funcionalidad (detallada en la sección 5.9), se realizan los siguientes cambios:

- **Cambio en la clase de Configuración.** En el proyecto tomado como punto de partida se utilizaba la clase *'Configuracion'* como un almacén de constantes que afectaban al modo de ejecución de la aplicación. Para este proyecto, esta clase se separa en dos: *'Configuracion'* y *'Constantes'*. *'Configuracion'* se trata de una clase que almacena los valores de configuración que pueden ser modificados durante la ejecución de la aplicación (mediante la descarga de los mismos desde servidor, por ejemplo, qué monitorizaciones se quieren llevar a cabo), mientras que *'Constantes'* almacena valores que han de mantenerse constantes durante toda la ejecución de la aplicación y solo deberían ser modificados antes de la compilación del proyecto (por ejemplo, umbrales para los algoritmos y modos de ejecución). Dado que *'Configuracion'* ha de ser globalmente accesible y modificable, tiene que implementar el patrón *Singleton* (detallado en la sección 4.2.5). En el caso de *'Constantes'*, no es necesario, pues no es modificable.
- **Reestructura de paquetes.** Se realizan cambios en la estructura de paquetes, respetando la estructura principal pero modificando las agrupaciones menores. En concreto, se separan los Sensores de sus Listeners en paquetes diferentes, teniendo una estructura que diferencia más claramente los componentes de la implementación del patrón Observer (detallado en la sección 4.2.4). Además, con esta distinción es más sencillo añadir múltiples listeners a un solo sensor (cosa que en el proyecto tomado como punto de partida no se tenía en cuenta). La estructura final está documentada en la sección 4.2.1. También se cambia la nomenclatura de algunas clases, para hacer mejor referencia a cuál es su objetivo.
- **Uso de strings.xml.** Dado que se trata de una aplicación Android, los textos que en algún momento son visibles para el usuario han de ir almacenados en el archivo *'strings.xml'*, el cual permite la rápida modificación de los mismos y una fácil traducción a otros idiomas.

3.5. Iteración 5

Al tratarse de la iteración final del proyecto, la gran mayoría de trabajo realizado se trata de la redacción de esta memoria y documentación y limpieza de la propia aplicación. Aun así, se llevan a cabo un par de implementaciones para dar un aspecto más acabado a la aplicación.

- **Descarga de la selección de monitorizaciones.** Se decide que la modificación de las monitorizaciones a realizar se ha de realizar en tiempo de ejecución. Esta configuración se ha de descargar desde el servidor, lo cual se detalla en la sección 5.10.

- **Alternativas de algoritmos.** A lo largo del proyecto se ha trabajo con 3 alternativas para el algoritmo de detección de caídas a utilizar. Finalmente, se decide incluir un parámetro de configuración que permita seleccionar qué algoritmo se desea utilizar (explicado en la sección 5.5.3). De esta forma se facilitan pruebas futuras.
- **Actualización de la aplicación.** Para una mejor presentación, se le da un nombre más significativo a la aplicación (TFG-monitorizacion) y se le cambia la versión (a la 2.0).

Por último, cuando se sabe que no se van a realizar más modificaciones al proyecto, se analiza el mismo con el plugin de Android Studio SonarLint [69], el cual realiza un análisis del código y sugiere modificaciones para hacer el mismo más fácil de leer, mantener y modificar. De estas sugerencias, se llevan a cabo aquellas que resultan más importantes y aquellas que son simples de realizar.

Capítulo 4

Estado final de la aplicación

A lo largo de este capítulo se documentará el análisis, diseño e implementación final de la aplicación. Para ello se analizarán las diferentes historias de usuario, haciendo uso de requisitos, casos de uso, patrones utilizados para resolverlas y diagramas pertinentes.

En este capítulo solo se documentarán aquellas áreas relevantes a la realización de este proyecto, excluyendo detalles correspondientes a la aplicación tomada como punto de partida. El análisis, diseño e implementación realizado en el proyecto que se usa como punto de partida se mantendrá en medida de lo posible y solo se documentará en caso de ser modificado.

4.1. Análisis

4.1.1. Análisis de requisitos

4.1.1.1. Requisitos funcionales

En la tabla 4.1 se documentan los requisitos funcionales del proyecto. Estos son aquellos que dictan las funcionalidades que ha de tener la aplicación para satisfacer las necesidades del usuario. En la tabla vemos los requisitos identificados por su ID, acompañados de un nombre significativo y una breve descripción de lo que se tratan.

ID	Nombre	Descripción
RF01	Detección de caída.	El sistema será capaz de detectar si el usuario se ha caído.
RF02	Detección de consciencia.	Tras una caída, el sistema será capaz de detectar si el usuario se encuentra o no consciente.
RF03	Enviar aviso al centro de control.	El sistema será capaz de enviar un aviso de caída al centro de control.
RF04	Configuración monitorizaciones.	El sistema permitirá al usuario activar y desactivar monitorizaciones a demanda.
RF05	Deshabilitar monitorización de caídas.	En caso de que se detecte una retirada del dispositivo, el sistema deberá de pausar la detección de caídas.

Tabla 4.1: Requisitos funcionales

4.1.1.2. Requisitos no funcionales

En la tabla 4.2 se documentan los requisitos no funcionales del proyecto. Estos son aquellos que especifican restricciones, o criterios que el sistema ha de cumplir, sin ser detectados por el usuario. En la tabla vemos los requisitos identificados por su ID, acompañados de un nombre significativo y una breve descripción de lo que se tratan.

ID	Nombre	Descripción
RNF01	Entorno de desarrollo.	El sistema será desarrollado en Android Studio para dispositivos Smartwatch con Android 6.0 en adelante (API 23).
RNF02	Conexión a internet.	El sistema tendrá que estar conectado a internet para poder enviar avisos al centro de control.
RNF03	Almacenamiento datos.	En caso de no haber conexión de red, el sistema almacenará la información de la caída para enviarla cuando vuelva a haber conexión.
RNF04	Distinción de caídas.	El sistema será capaz de diferenciar si se ha producido una caída leve (el usuario está consciente), grave (el usuario está inconsciente) o falsa (el usuario no se ha caído).
RNF05	Precisión de la detección aceptable.	El sistema debe detectar un mínimo de un 85 % de las caídas dadas.
RNF06	Detección de consciencia en tiempo aceptable.	El sistema ha de detectar si el usuario se encuentra consciente tras una caída en un tiempo aceptable (máximo 5 segundos tras la detección de la caída).
RNF07	Envío aviso en tiempo aceptable.	El sistema ha de enviar al centro de control el aviso de caída en un tiempo aceptable (máximo 5 segundos tras la detección de movimiento tras la caída).
RNF08	Informe de errores.	En caso de darse un error, el dispositivo debe informar al sistema y registrar el error en un log.

Tabla 4.2: Requisitos no funcionales

4.1.1.3. Requisitos de información

En la tabla 4.3 se documentan los requisitos de información del proyecto. Estos son aquellos que especifican qué datos ha de almacenar el sistema. En la tabla vemos los requisitos identificados por su ID, acompañados de un nombre significativo y una breve descripción de lo que se tratan.

ID	Nombre	Descripción
RI01	Ubicación usuario.	El sistema ha de almacenar ubicación GPS, fecha, hora y tipo de caída (leve o grave) al momento de detectarse una caída.
RI02	Registro de errores.	El sistema ha de almacenar un log con los errores que se produzcan.
RI03	Configuración del usuario.	El sistema almacenará las preferencias sobre qué funcionalidades han de estar activas para el usuario, así como el tiempo que ha de pasar entre comprobaciones de caídas.
RI04	Datos de la detección.	El sistema almacenará los datos que llevan a una detección de caída, así como si esa detección se trata de una caída falsa, grave o leve.

Tabla 4.3: Requisitos de información

4.1.2. Casos de uso

La figura 4.1 se corresponde al diagrama de Casos de Uso correspondiente a la aplicación del Smartwatch. Se mantienen aquellos correspondientes al proyecto que se toma como punto de partida, indicados con un color verde. De esta forma, se tiene una mejor comprensión de la aplicación final. Estos CCUU tan solo se muestran en el diagrama, no se desarrollarán detalladamente.

En caso de necesitar referenciar alguno de los casos de uso del proyecto punto de partida en el detalle de los casos de uso correspondientes a este proyecto, se hará con el acrónimo CUP (caso de uso previo) y se aportará una breve descripción.

Además, el CU indicado con un color púrpura es un CUP que ha de ser ligeramente modificado (se detallará posteriormente).

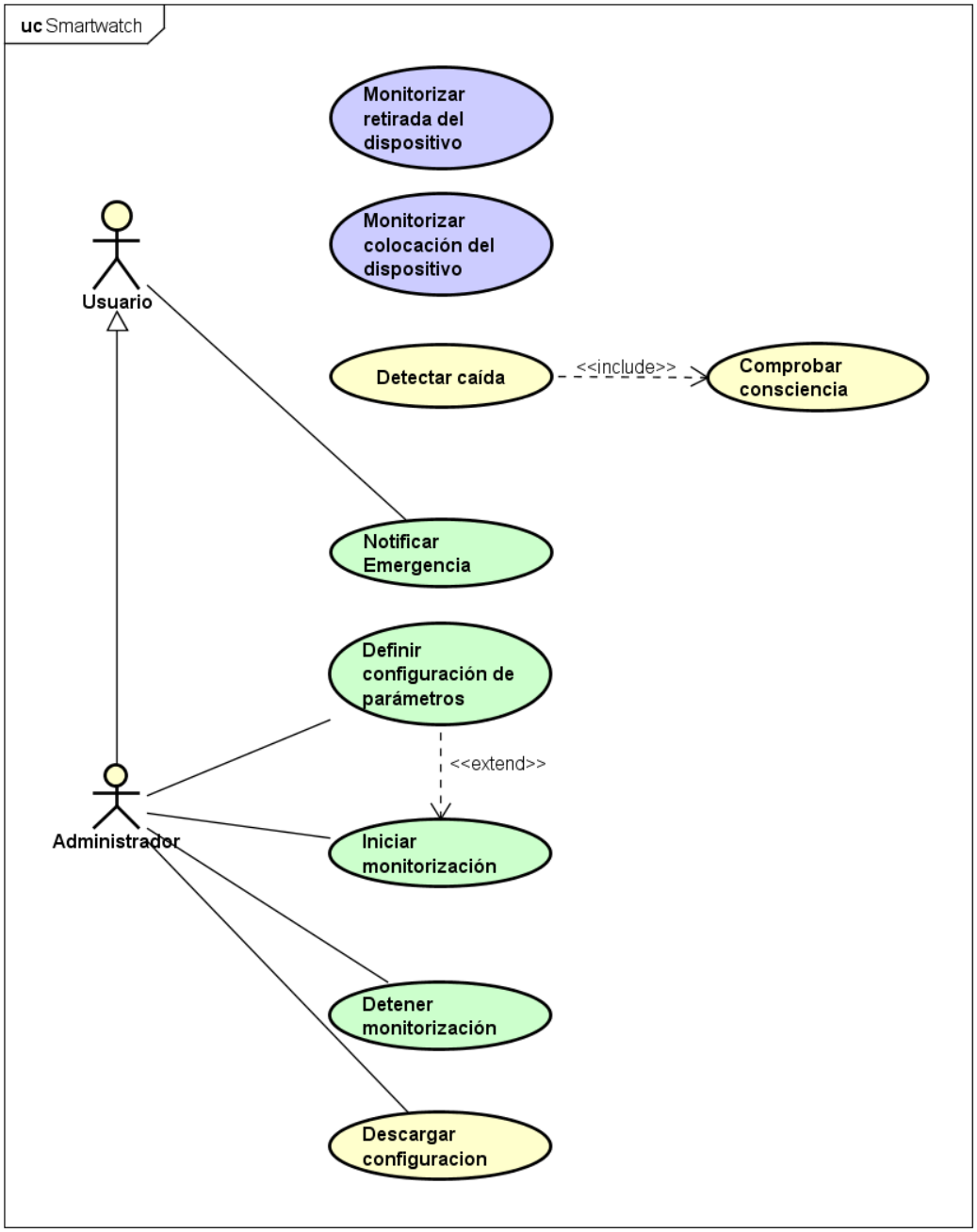

Figura 4.1: Diagrama de CCUU del Smartwatch

A continuación se detallan los casos de uso a desarrollar completamente a lo largo del proyecto.

ID	CU01
Nombre	Descargar configuración.
Descripción	El usuario deberá poder escoger qué opciones de monitorización quiere que se realicen y cuáles no.
Precondición	Ninguna
Secuencia normal	1. El usuario quiere descargar la nueva configuración. 2. El sistema solicita la nueva configuración al servidor. 3. El servidor devuelve la nueva configuración. 4. El sistema almacena dicha configuración y notifica a las clases implicadas.
Postcondición	Solo las opciones escogidas serán monitorizadas.
Excepciones	Ninguna.

Tabla 4.4: Caso de Uso CU01

ID	CU02
Nombre	Detección de caída.
Descripción	El sistema deberá detectar si el usuario que lleva el dispositivo se ha caído.
Precondición	La aplicación debe estar iniciada. La opción de monitorización de caída debe estar activa.
Secuencia normal	1. El sistema obtiene los valores del acelerómetro. 2. El sistema comprueba los valores del sensor y comprueba que se ha dado una caída. 3. El sistema comprueba que el usuario se encuentra consciente, se ejecuta el caso de uso CU03. 4. Si la aplicación sigue iniciada, se vuelve a ejecutar el paso 1 cuando se cumpla un cierto intervalo de tiempo.
Postcondición	Ninguna.
Excepciones	2a. Si no se detecta caída, el sistema ejecuta el paso 1 de nuevo. 3a. Excepciones del CU03.

Tabla 4.5: Caso de Uso CU02

ID	CU03
Nombre	Detección de consciencia.
Descripción	El sistema deberá detectar si el usuario se encuentra consciente tras una caída.
Precondición	Se ha detectado una caída mientras el usuario llevaba el dispositivo.
Secuencia normal	1. El sistema muestra una pantalla de confirmacion de caida al usuario. 2. El usuario pulsa el botón de confirmación. 3. El usuario se encuentra consciente, el sistema manda la notificación de consciencia al servidor, se ejecuta el CUP07.
Postcondición	Se ha enviado la notificación de tipo de caída.
Excepciones	2a. Si el usuario pulsa el botón de cancelación no se ha dado caída, el sistema manda la notificación de falsa caída al servidor, se ejecuta el caso de uso CUP07. 2b. El usuario no pulsa ningún botón tras un corto periodo de tiempo, el sistema manda notificación de inconsciencia al servidor, se ejecuta el caso de uso CUP07.

Tabla 4.6: Caso de Uso CU03

A continuación se aportan unas breves descripciones de los casos de uso correspondientes al proyecto utilizado como punto de partida, que resultan necesarios para la especificación de los casos de uso de este proyecto. Al tratarse de Casos de Uso Previos (desarrollados en el punto de partida) me refiero a ellos con las siglas CUP.

ID	CUP04
Nombre	Notificar emergencia.
Descripción	El usuario deberá poder notificar una emergencia cuando se desee.

Tabla 4.7: Caso de Uso CUP04

ID	CUP07
Nombre	Enviar notificación.
Descripción	El sistema deberá enviar la notificación junto con la ubicación del dispositivo.

Tabla 4.8: Caso de Uso CUP07

Por último, se detalla la modificación al CUP06: Monitorizar retirada del dispositivo. Si se da la retirada del dispositivo es necesario pausar la detección de caídas, de tal forma que no se detecten caídas cuando el usuario no tiene el smartwatch puesto. Para ello, se detalla la funcionalidad ya existente en color negro y el añadido necesario en color azul.

En cuanto al caso de uso "Monitorizar colocación del dispositivo", no se trata de un caso de uso documentado en el proyecto tomado como punto de partida [8], sino que se encuentra contenido en el CUP06. Sin embargo, es importante mencionarlo aquí, pues cuando se detecte la colocación del dispositivo se ha de reanudar la detección de la caída. Como no se encuentra documentado, se proporciona una descripción muy simple del mismo:

ID	CUP06
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 obtiene los valores del sensor de proximidad. 5. El sistema comprueba los valores del acelerómetro y detecta que se ha retirado el dispositivo. 6. El sistema deshabilita temporalmente la detección de caídas. 7. El sistema envía una notificación de retirada, se ejecuta el caso de uso CUP07. 8. Si la aplicación sigue iniciada, se vuelve a ejecutar el paso 1 cuando el intervalo de tiempo se haya cumplido.
Postcondición	No se detectarán caídas hasta la reactivación de la monitorización.
Excepciones	6a. La configuración actual para la monitorización de caídas está desactivada. El CU queda sin efecto.

Tabla 4.9: Caso de Uso CUP06

ID	CUP06.5
Nombre	Monitorizar colocación del dispositivo.
Descripción	El sistema deberá detectar si el dispositivo es colocado de nuevo en el 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 habilita de nuevo la detección de caídas. 3. El sistema continúa con su ejecución normal.
Postcondición	Se vuelve a realizar la monitorización de caídas.
Excepciones	2a. La preferencia actual para la monitorización de caídas está desactivada. El CU queda sin efecto.

Tabla 4.10: Caso de Uso CUP06.5

4.1.3. Modelo de dominio

El modelo de dominio de una aplicación permite observar una representación a grandes rasgos de la estructura de clases y las relaciones entre las mismas que tendrá la aplicación una vez finalizada. De esta forma se pueden tener en cuenta los puntos clave de la aplicación y las interacciones entre los mismos.

Teniendo en cuenta que las funcionalidades a desarrollar son añadidos al proyecto que se toma como punto de partida, sobre los sensores y clases ya presentes en el mismo, el modelo de dominio se corresponde con el que se encontraba en este. Se incluyen en amarillo las clases que se espera que

se mantengan igual y en azul las clases donde se llevarán a cabo las modificaciones.

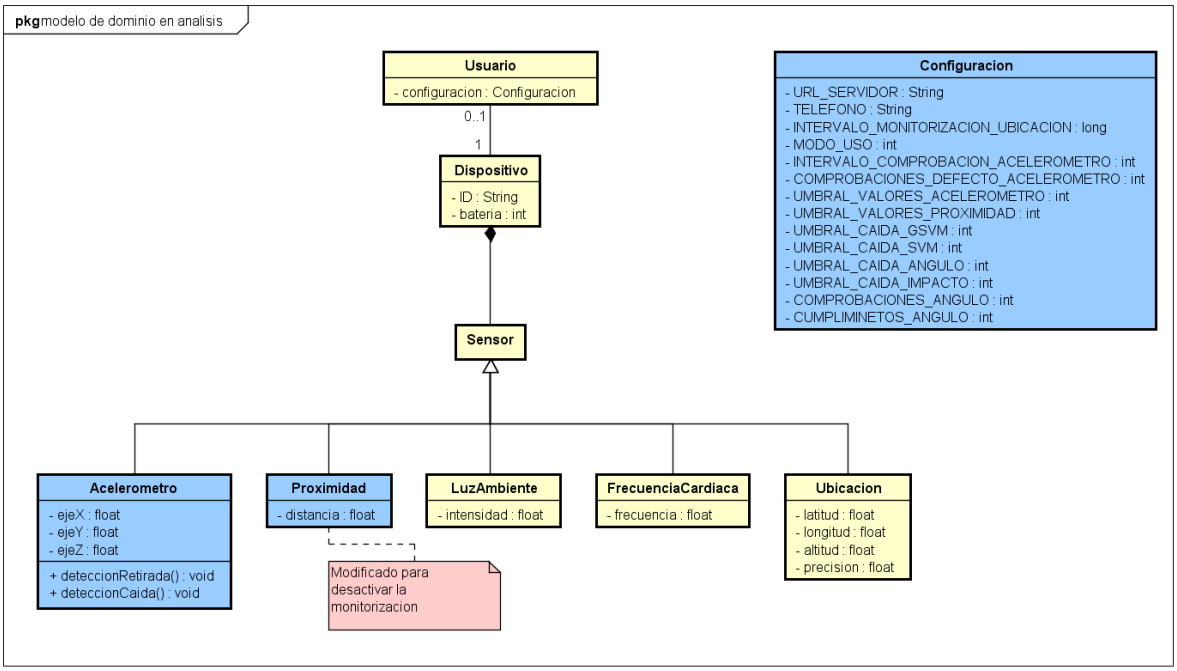

Figura 4.2: Modelo de dominio en análisis de la aplicación

4.1.4. Diagramas de actividad

Mediante los diagramas de actividad representamos el flujo de ejecución de la aplicación, junto con la interacción del usuario con la misma. De esta forma, se pueden visualizar de una forma más clara estos flujos para después poder proceder a su implementación. A continuación se muestran los diagramas de actividad de los casos de uso del proyecto (detallados previamente).

4.1.4.1. CU01 Descargar configuración

En este diagrama de actividad cabe destacar que se necesita que el servidor funcione para su correcto funcionamiento. Se desconoce la implementación concreta del servidor, por lo que se resume con una actividad a realizar.

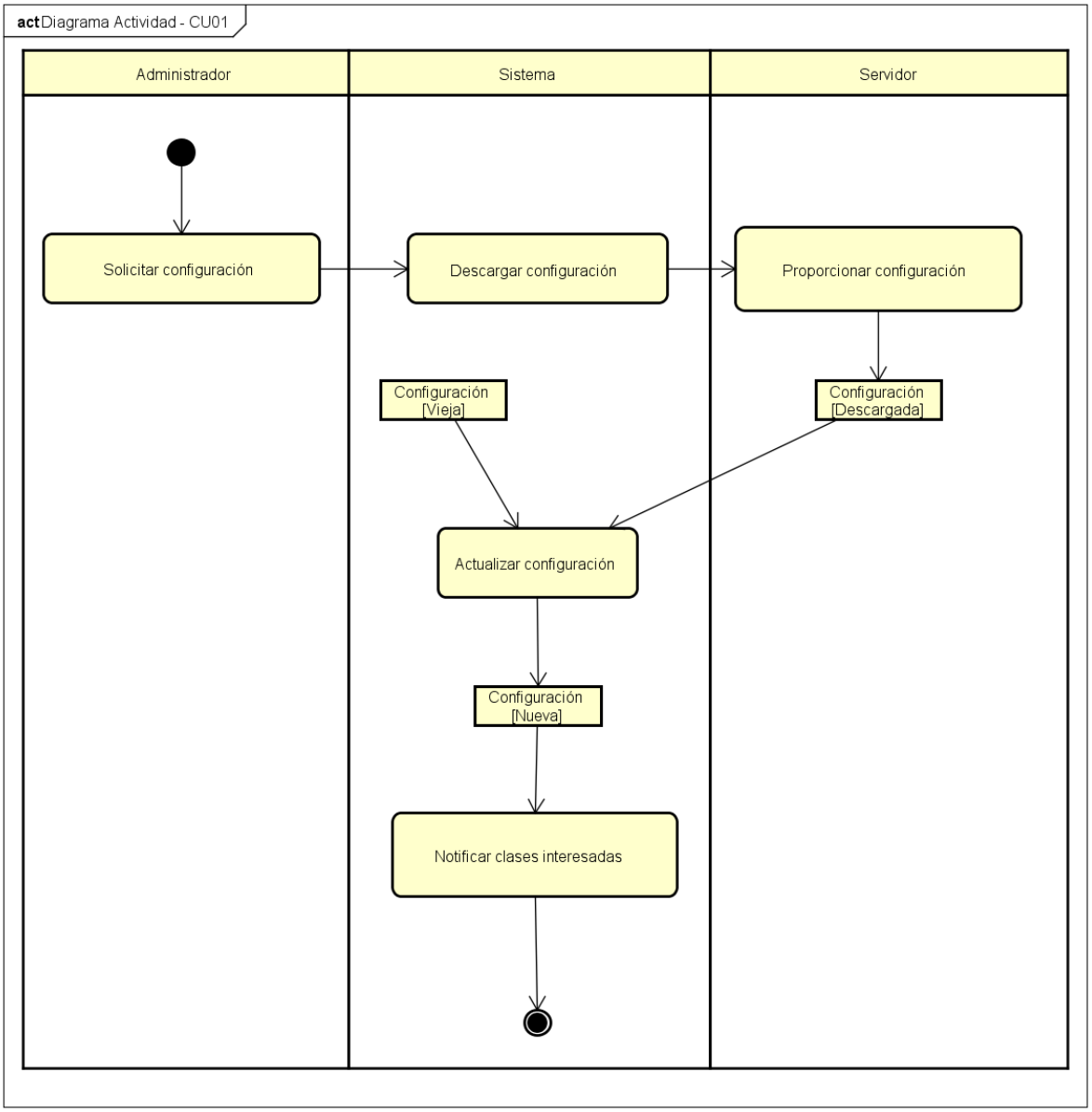

Figura 4.3: Diagrama de actividad del CU01

4.1.4.2. CU02 y CU03 Detección de la caída y consciencia

Debido a que ambos casos de uso actúan juntos, se detallan en conjunto. El flujo de ejecución general es el siguiente:

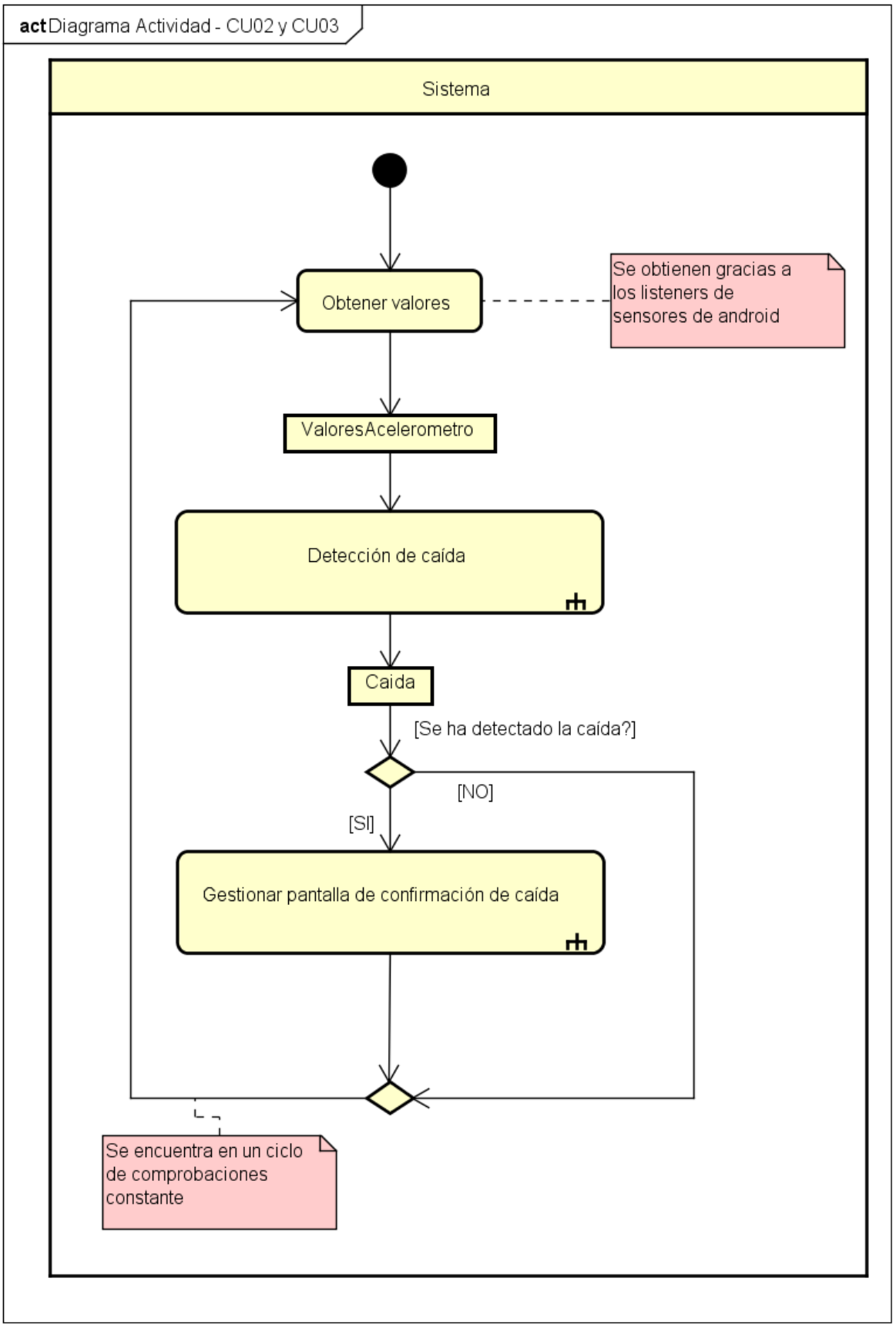

Figura 4.4: Diagrama de actividad de los casos de uso CU02 y CU03

Para dicha ejecución, la detección de caída (parte correspondiente al CU02) se realiza de la siguiente forma:

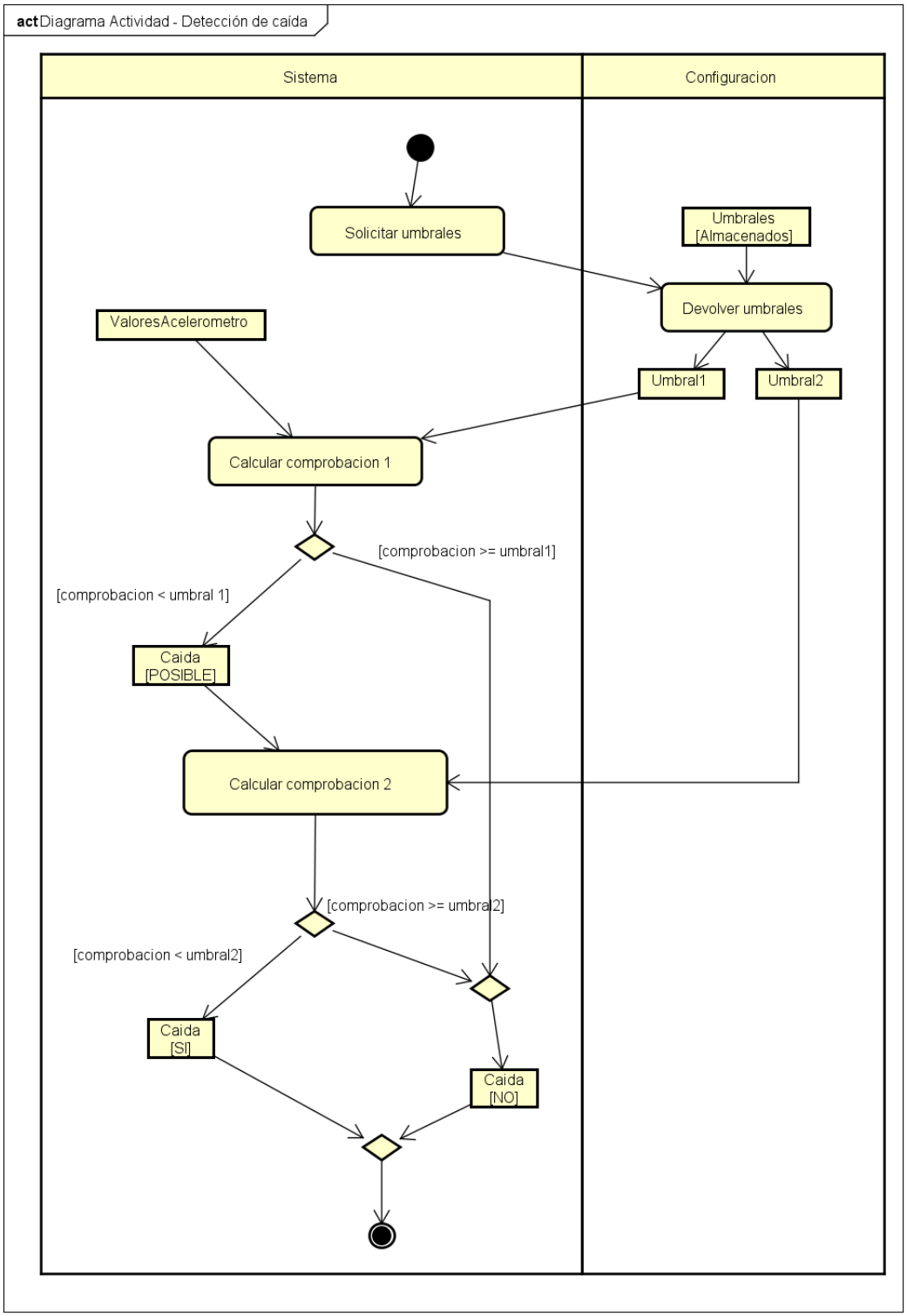

Figura 4.5: Diagrama de actividad de la detección de caída

La parte correspondiente al CU03 se realizará mediante una pantalla de confirmación en la que el usuario pueda indicar si la caída ha sido cierta o no. En caso de que no conteste en un tiempo determinado, se asumirá que la caída ha sido grave, pues el usuario no se encuentra consciente para

contestar. Tras determinar el tipo de caída, se hace uso de la actividad documentada en el proyecto tomado como punto de partida 'Enviar notificación' (resaltada en verde) [8]. El flujo es el siguiente:

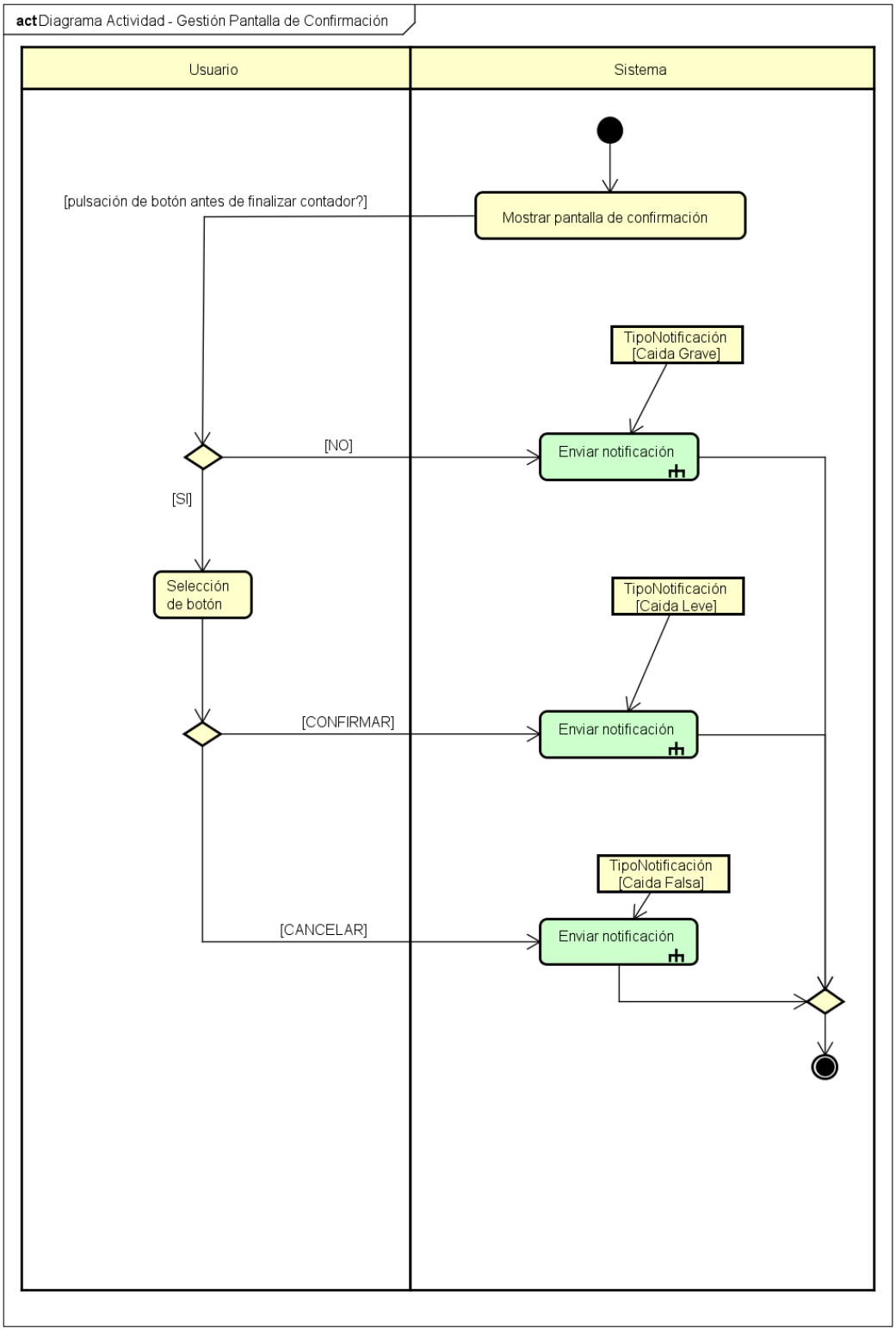

Figura 4.6: Diagrama de actividad de la confirmación de caída

4.1.4.3. CUP06 Monitorizar retirada del dispositivo

Dado que se trata de un caso de uso realizado en el proyecto tomado como punto de partida, se proporciona la documentación del mismo (color verde) con la modificación a realizar durante este proyecto (color amarillo). Para la colocación del dispositivo se sigue el mismo flujo pero comprobando la colocación y no la retirada.

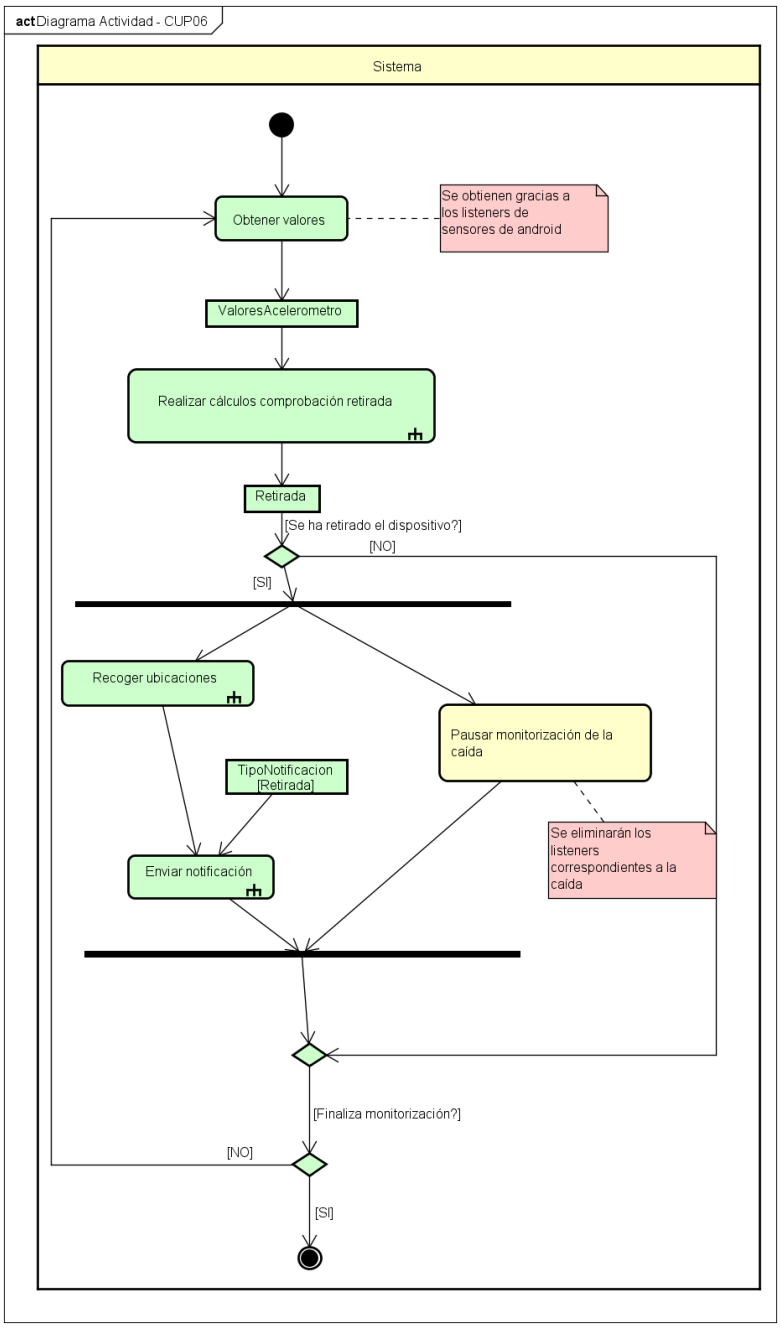

Figura 4.7: Diagrama de actividad del caso de uso CUP06

4.2. Diseño

Una vez realizado el análisis inicial, se realiza el diseño de la aplicación, una versión más detallada y cercana a la implementación que el análisis, cuyo objetivo es facilitar la escritura mediante código. Se detalla el diseño de paquetes de la aplicación, clases de interés, diagrama de despliegue y patrones de diseño utilizados.

4.2.1. Arquitectura del sistema

En este aspecto se lleva a cabo una reestructura de los paquetes originalmente utilizados en el proyecto tomado como punto de partida, pues, aunque en su diseño se planteaba una estructura clara, a la hora de la implementación esta estructura se fue modificando y perdiendo.

Para realizar la distribución de paquetes principal de la aplicación nos basamos en que las aplicaciones de Android están estructuradas en las tres siguientes capas:

1. **Capa de Interfaz de Usuario**. Es la capa encargada de gestionar las pantallas (Activities y Fragments) que se muestran al usuario, así como la interacción del usuario con las mismas. En esta aplicación se trata de las pantallas de confirmación, tanto de emergencia como de caída.
2. **Capa de Dominio**. Es una capa opcional, no todas las aplicaciones de Android la utilizan. Representa toda la lógica de la aplicación. En este proyecto es fundamental, pues es la capa encargada del procesamiento de los datos obtenidos por los sensores y de los algoritmos utilizados para las monitorizaciones.
3. **Capa de Datos**. Contiene la lógica correspondiente a los datos de la aplicación, es decir, el almacenamiento y obtención de los mismos. En el caso de esta aplicación, se corresponde con el almacenamiento de notificaciones en el dispositivo (en caso de no tener internet) y el envío de las mismas al servidor (en caso de sí tenerlo). También sería la capa responsable de la descarga de configuración.

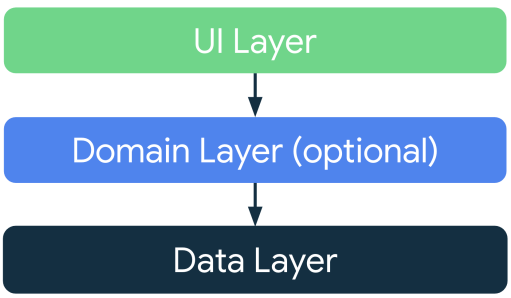

Figura 4.8: Arquitectura en capas de las aplicaciones de Android [70]

Teniendo en cuenta esta estructura, se tiene un paquete correspondiente a cada capa, *'interfaz'* para la capa de interfaz de usuario, *'dominio'* para la capa de dominio y *'persistencia'* para la capa de

datos. Además, se tiene en cuenta un cuarto paquete *'global'* que se trata de clases que han de ser accesibles globalmente (se trataría de una capa relajada en la estructura de capas). Se han detallado las dependencias externas más relevantes, pero al tratarse de una aplicacion Android se tienen muchas más dependencias a subpaquetes del sistema android (android.util, android.content, android.os, android.view...) así como dependencias a paquetes extra de java (java.util, java.io, java.lang...).

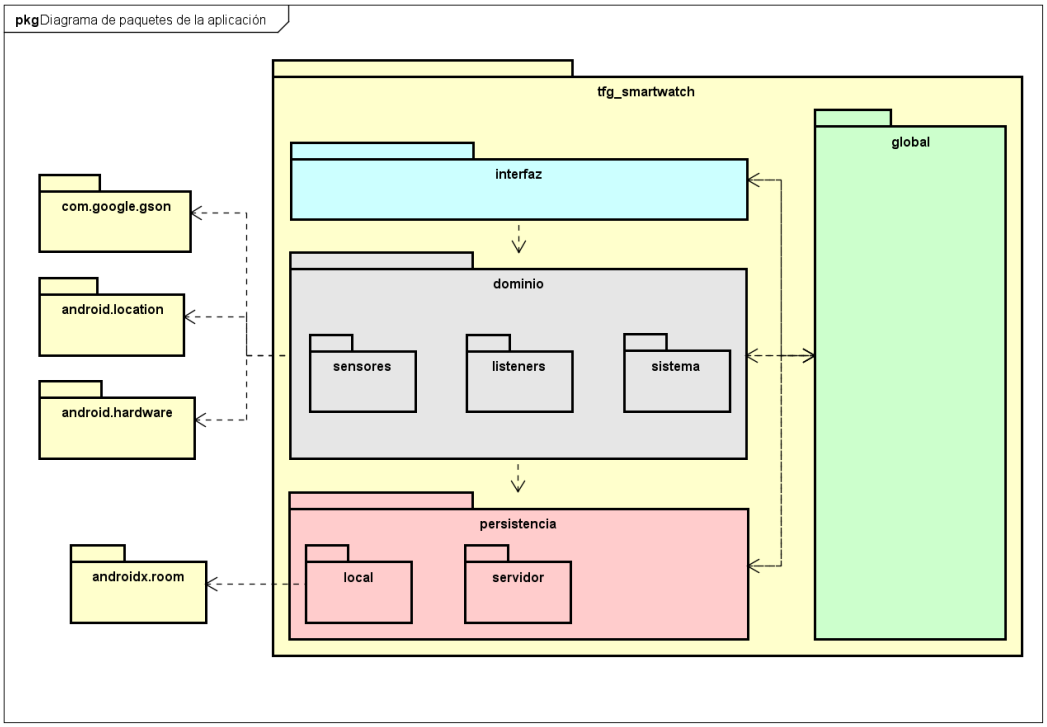

Figura 4.9: Diagrama de paquetes de la aplicación

Dados estos paquetes generales cabe destacar las estructuras de los paquetes más específicos siguientes:

4.2.1.1. Paquete global

Este paquete se incluye en el diagrama general para aportar mayor claridad, pero dada la estructura por defecto de las aplicaciones Android, su contenido se encuentra directamente en el paquete raíz del proyecto (como se puede apreciar en la sección 4.3: Estructura final del proyecto). Este está compuesto por clases cuyo acceso ha de ser global para un correcto funcionamiento de la aplicación. Esto incluye configuración (Configuracion y Constantes), el lanzador de la aplicación (MainActivity) y la clase encargada de la ejecución en segundo plano (Background). Aparte de las dependencias internas, la mayoría de clases de la aplicación dependen de alguna (o todas) las clases de este paquete.

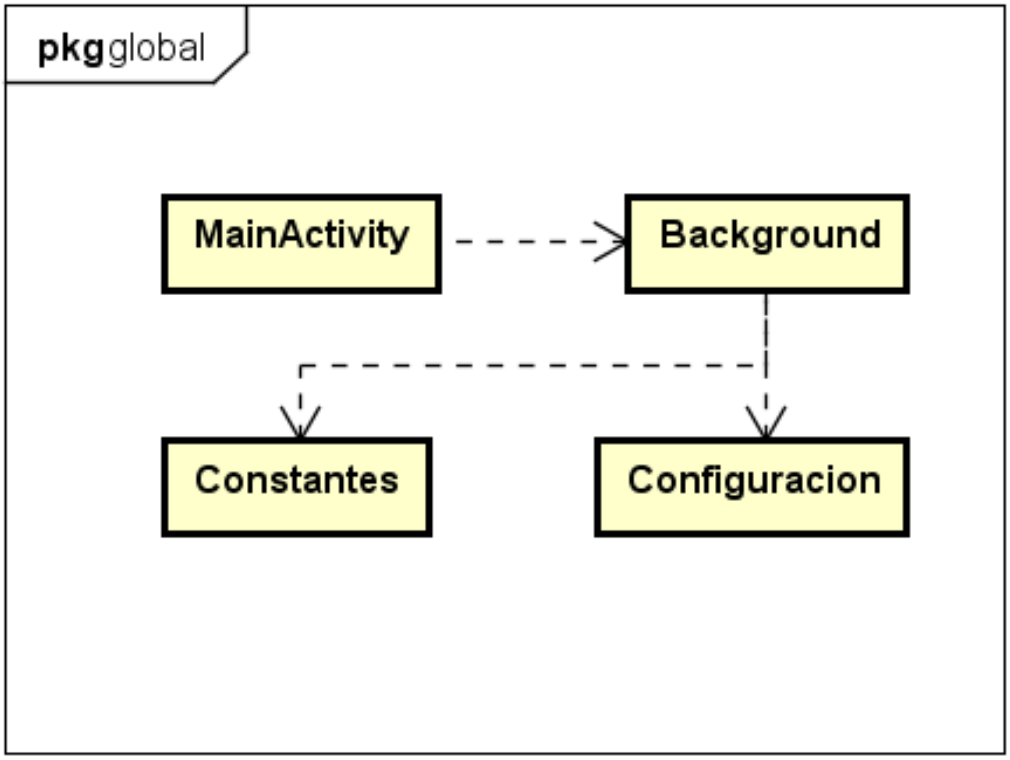

Figura 4.10: Estructura del paquete global

4.2.1.2. Paquete interfaz

Este paquete contiene todas las pantallas que se han de mostrar al usuario. En este caso, estas pantallas se implementarán con Activities, por lo que no es necesario tener más sub-paquetes para mayor diferenciación. Debido a que se necesita una pantalla de confirmación con temporizador en varios lugares de la aplicación, se decide utilizar la siguiente herencia para las mismas:

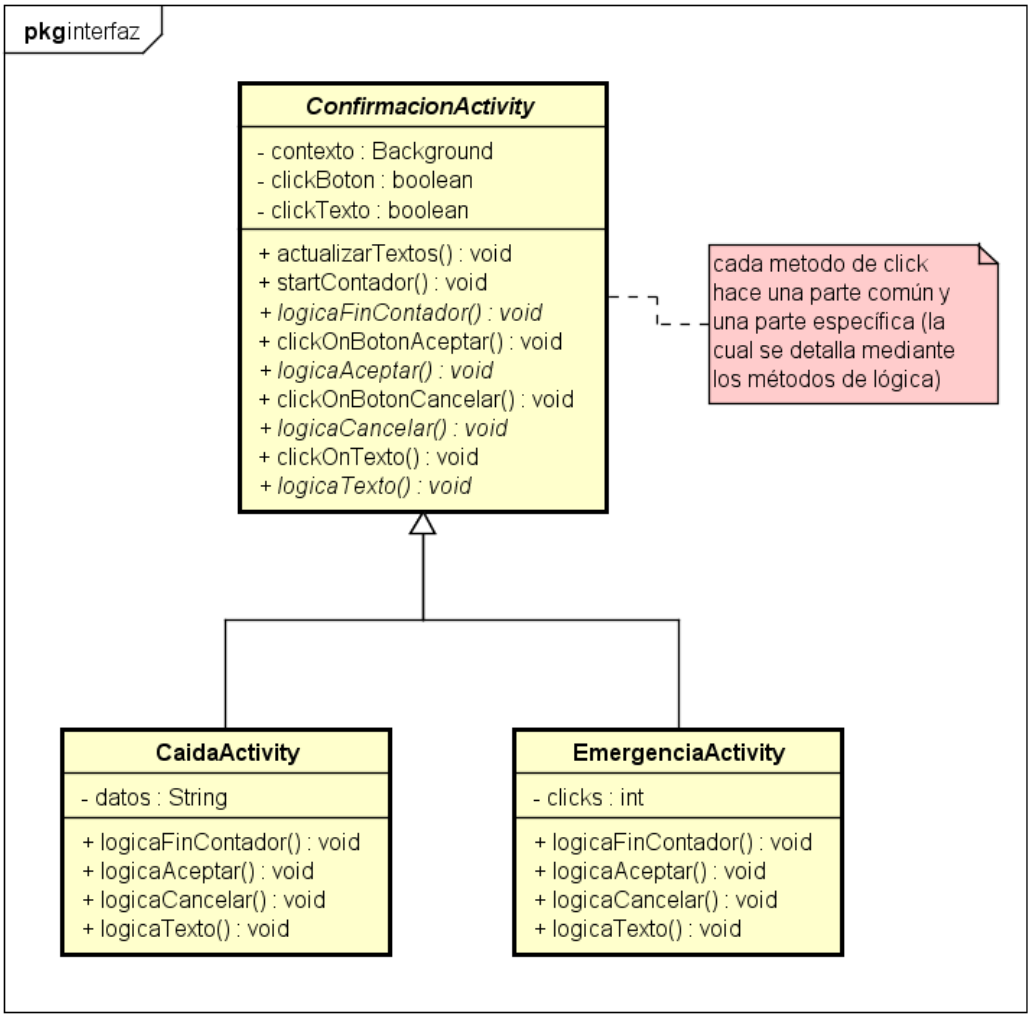

Figura 4.11: Ejemplo de herencia de activities

4.2.1.3. Paquete dominio

Este paquete consta de los tres sub-paquetes siguientes:

1. **Sensores**: Se trata del paquete que contiene las clases que representan los sensores a utilizar en el sistema (acelerómetro, pulsómetro, etc.).
2. **Listeners**: Se trata del paquete que contiene los listeners que se encargan de la toma de datos de los sensores contenidos en el paquete de 'sensores'. A su vez está dividido en sub-paquetes correspondientes a cada sensor, los cuales deberían estar compuestos de una interfaz del listener adaptado al sensor concreto y diferentes implementaciones de la misma. Un ejemplo de esto se puede ver en la figura 4.12.
3. **Sistema**: Se trata del paquete que contiene la lógica relacionada con los servicios del sistema que se han de utilizar. Este paquete se mantiene tal como se encontraba en el proyecto tomado como punto de partida [8].

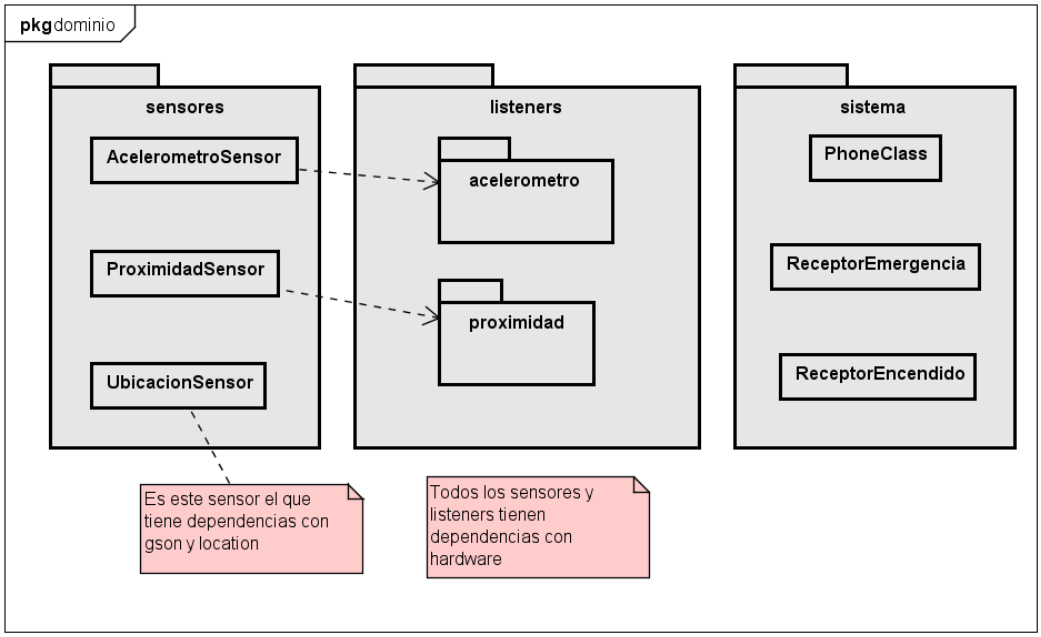

Figura 4.12: Estructura del paquete dominio

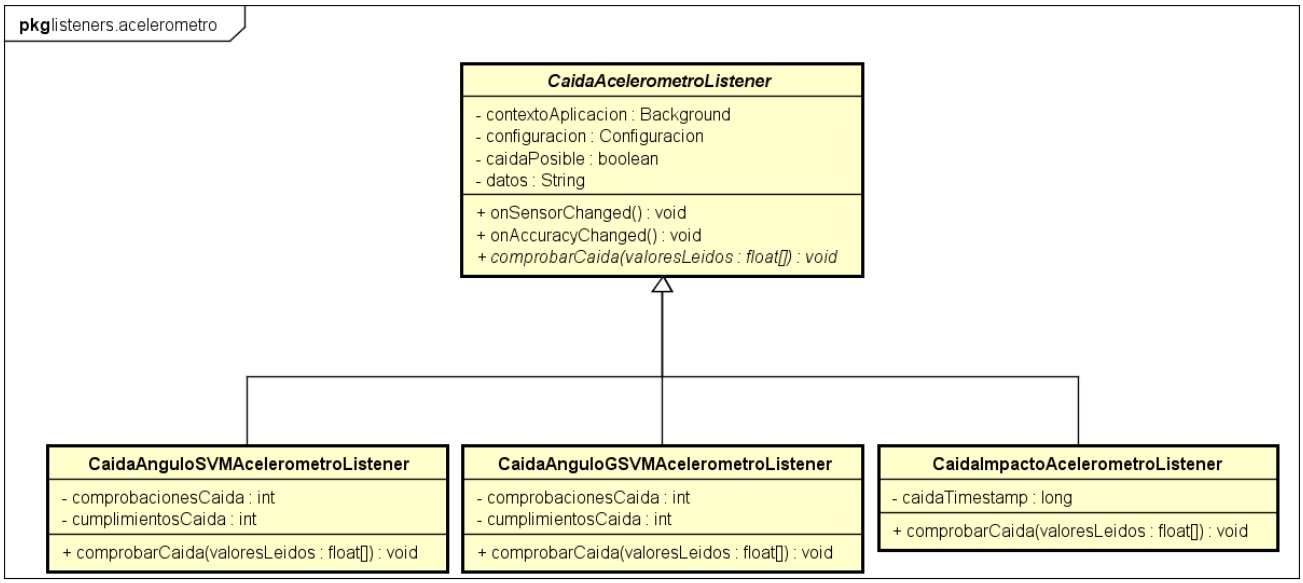

Figura 4.13: Ejemplo de herencia de listeners

4.2.1.4. Paquete persistencia

Este paquete se mantiene como en el proyecto tomado como punto de partida, salvo porque configuración se ha movido a 'global'. Consta de los dos sub-paquetes siguientes:

1. **Local**: Se trata de la interfaz necesaria para la configuración mediante Room de la base de datos local.

2. **Servidor**: Se trata de la clase APIServidor, la cual incluye los métodos necesarios para la subida y descarga de datos del servidor. En un futuro, si se necesitasen muchas configuraciones del servidor, estas deberían ir en este paquete y no en la clase de Configuracion general.

Además, este paquete contiene las clases MensajeJSON y MensajeJsonDAO, lo cuales son la representación de los datos de la aplicación que es necesario almacenar y su DAO correspondiente (patrón que se detalla en la sección 4.2.6).

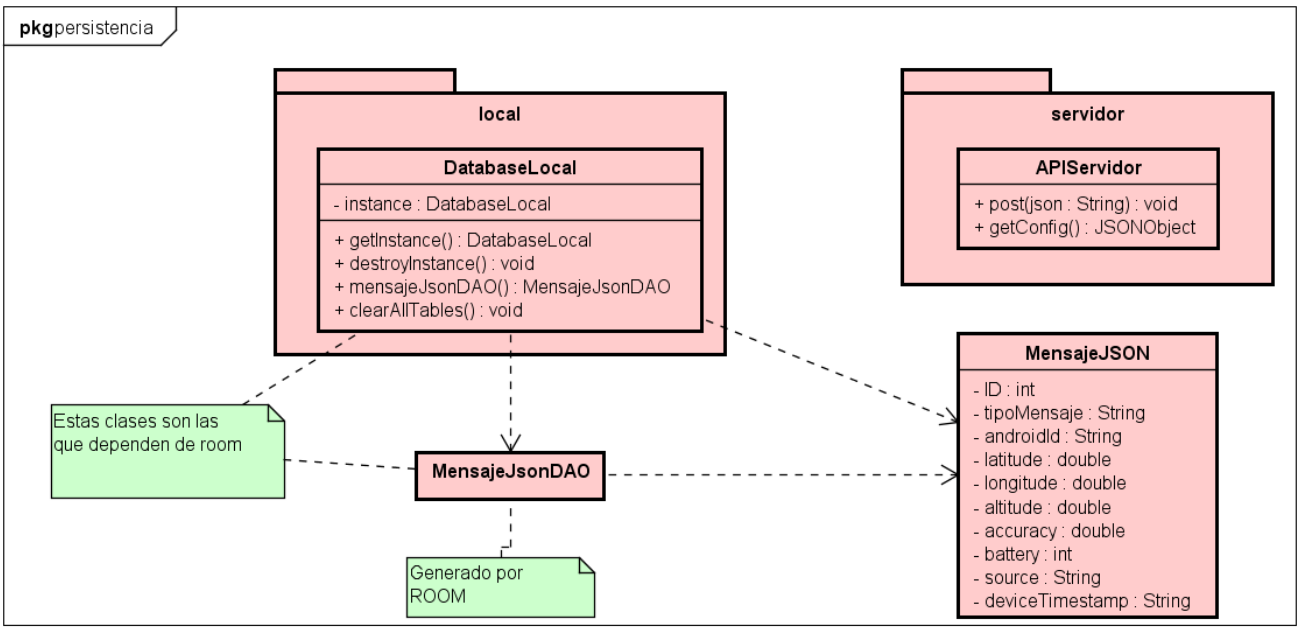

Figura 4.14: Estructura del paquete persistencia

4.2.2. Diagrama de despliegue

El diagrama de despliegue muestra la distribución de los componentes software del sistema en los dispositivos hardware necesarios para la ejecución del mismo. Se incluyen los principales nodos hardware y software necesarios para el uso de la aplicación en un entorno real.

Para el despliegue de este proyecto se necesitan dos dispositivos, el Smartwatch (donde se ejecutará la aplicación) y el servidor con el que la aplicación ha de conectarse. Ambos estarán conectados por el protocolo seguro de HTTP (HTTPS).

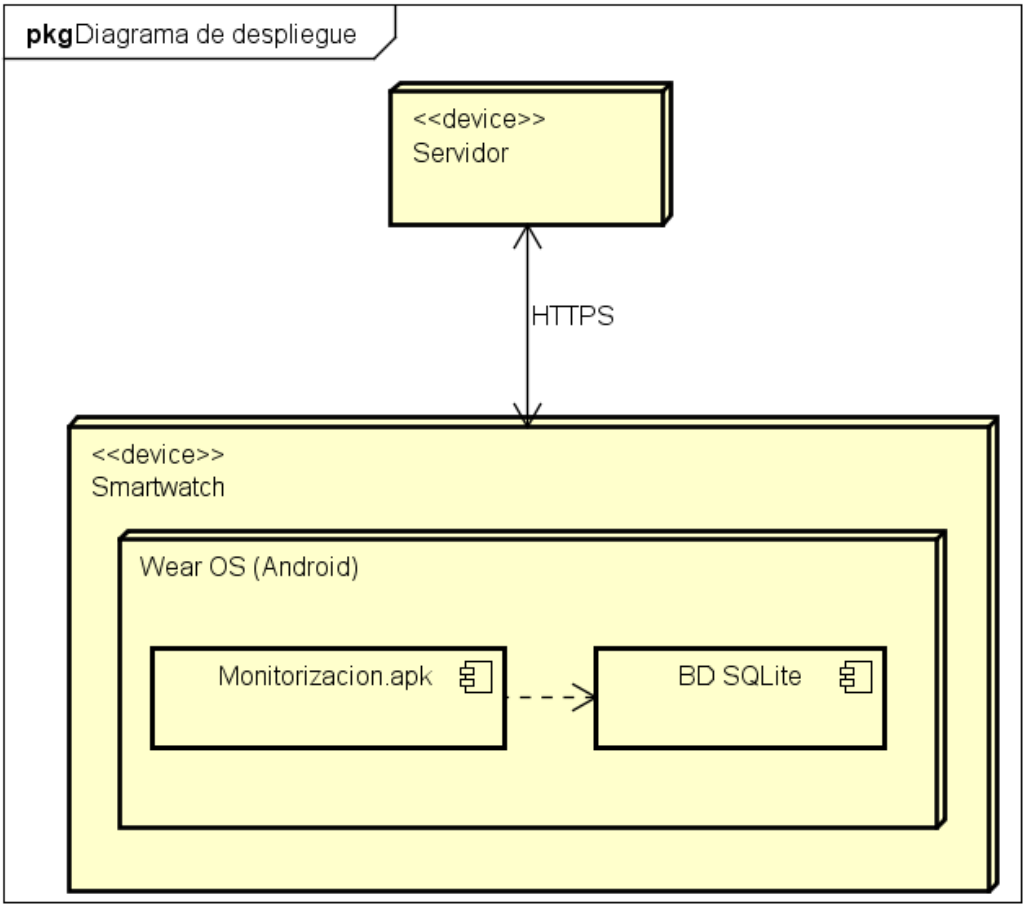

Figura 4.15: Diagrama de despliegue del proyecto

4.2.3. Patrones de diseño

Los patrones de diseño se tratan de respuestas estandarizadas a problemas comunes en la creación de sistemas software. El uso de estos patrones es fundamental a la hora de crear aplicaciones, pues garantiza que estas mantengan una estructura y código fáciles de comprender, modificar, mantener y reutilizar.

Los patrones de diseño utilizados en el proyecto son los siguientes:

4.2.4. Patrón Observer

El patrón de diseño Observer (Observador) [71] permite que una serie de objetos observadores (también denominados subscriptores) reaccionen ante los cambios que se han dado en un sujeto observado (u observable). Este patrón es básico para el desarrollo de aplicaciones basadas en eventos, como son las aplicaciones Android.

Ejemplos de este patrón son cualquier método *'onClick'* de una interfaz o los *'BroadcastReceiver'*, que ejecutan su código tras una señal concreta del sistema. En este proyecto se encuentran *'Brodcas-*

tReceivers' en las clases *'ReceptorEmergencia'* y *'ReceptorEncendido'*.

Además, también se encuentra este patrón entre los Sensores y sus respectivos Listeners. Los sensores crean instancias de la clase *'SensorManager'*, que actúa como sujeto observado; los listeners implementan la clase *'SensorEventListener'*, que actúa como observador.

4.2.5. Patrón Singleton

El patrón de diseño Singleton [72] sirve para asegurarse de que una clase tiene una sola instancia en toda la aplicación y un único punto de acceso a la clase, de tal forma que se convierta en una referencia global fiable. Este patrón es muy común para sesiones, configuraciones, conexiones, etc.

El lugar más importante donde se aplica este patrón es el paquete 'global'. Las clases *'Configuracion'* y *'Background'* lo implementan, de tal forma que estas sean accesible desde cualquier lugar de la aplicación, asegurándonos de que todas las referencias utilizadas trabajan con la misma información.

4.2.6. Patrón DAO y DTO

El patrón de diseño Data Access Object (DAO) [73] sugiere separar la lógica de negocio de la aplicación (dominio) de la lógica de acceso a la base de datos (persistencia). Para ello, propone utilizar una clase que contenga los métodos necesarios para insertar, borrar, modificar y obtener los datos.

El patrón DAO suele utilizarse en compañía del patrón Data Transfer Object(DTO) [74], el cual sugiere utilizar clases concretas cuyo único objetivo es el de la transferencia de datos, en lugar de utilizar las clases utilizadas en el modelo de la aplicación.

Utilizando estos dos patrones en conjunto se obtiene una total separación de la lógica de datos, lo cual proporciona un mayor control sobre los datos que se transfieren entre la aplicación y métodos de persistencia de datos, los cuales pueden no ser seguros.

En este proyecto se aplica el patrón DTO mediante la clase *'MensajeJSON'* y su correspondiente DAO mediante la clase *'MensajeJsonDAO'*. La clase 'MensajeJSON' es la entidad que se almacenará en la base de datos, mientras que MensajeJsonDAO contiene los métodos necesarios para insertar, eliminar, actualizar y realizar dos tipos de consultas (obtención de todos los Mensajes y obtención de un Mensaje concreto dado su ID) en la base de datos.

4.2.7. Patrón Factory

El patrón de diseño Factory (Factoría) [75] proporciona una interfaz para la creación de un objeto de un subtipo determinado a partir de una superclase que engloba a estos diferentes subtipos. El uso de este patrón es común cuando no se conoce el subtipo concreto que se ha de utilizar para realizar una acción, por lo que hay que dar soporte para cualquier tipo que se pueda crear en tiempo de ejecución.

Para esta aplicación, este patrón se utiliza al hacer uso de la clase LocationManager de Android. Esta clase proporciona acceso a diferentes tipos de proveedores de ubicación, los cuales pueden ser de diversos tipos como, por ejemplo, GPS o WiFi. Hasta que no se ejecuta la aplicación y no se comprueba qué proveedor puede utilizar el dispositivo, se desconoce el subtipo concreto del mismo, por lo que se ha de proporcionar un soporte general para todos.

En cuanto a una implementación más propia del mismo, se ha de tener en cuenta para la creación de diferentes listeners para un mismo Sensor. Concretamente, se podría utilizar para dar un mejor soporte a las tres alternativas de listener utilizado para la monitorización de caídas mediante acelerómetro. Sin embargo, por falta de tiempo, no se termina de implementar adecuadamente.

4.3. Estructura final del proyecto

Se mantiene la estructura general utilizada en la aplicación tomada como punto de partida [8], la cual se corresponde con la estructura por defecto de las aplicaciones creadas en Android Studio, utilizando Gradle como gestor de dependencias. Teniendo esto en cuenta y el diseño documentado previamente, se llega a la estructura de archivos mostrada en la figura 4.16. A continuación se describen brevemente los componentes principales de la estructura:

- **AndroidManifest**: Se trata de un archivo XML que contiene la información básica de la aplicación, necesaria para que el sistema operativo sepa cómo ejecutarla. Contiene información sobre los permisos necesarios, arquitectura de la aplicación, actividades (pantallas), servicios y receptores que se van a utilizar y otra serie de metadatos sobre la aplicación.
- **Código fuente**: Se trata del directorio $(/app/src/main/java/com/example/tfg\_smartwatch)$. Ahí se encuentran todas las clases desarrolladas para la aplicación, escritas en su totalidad en Java.
- **Recursos**: Se trata del directorio en el cual se encuentran los recursos visuales de la aplicación. Las carpetas más relevantes son *'drawable'*, dónde se encuentran los iconos de la aplicación (se trata de los iconos por defecto) y, en caso de necesitarse imágenes, también se encontrarían en esta carpeta. Y la carpeta *'values'*, la cual contiene los archivos dónde se guarda la marca de la aplicación (colores y estilos) y el archivo *'strings.xml'* dónde se han de almacenar todos los textos que se vayan a mostrar al usuario, para permitir su fácil traducción sustituyéndolo por el archivo del idioma pertinente.
- **Configuración de Gradle**: El directorio Gradle contiene los ficheros de configuración de Gradle, los cuales contienen la configuración necesaria para gestionar las dependencias, tanto a nivel aplicación (Module: app) como a nivel proyecto (Project TFG-Smartwatch).

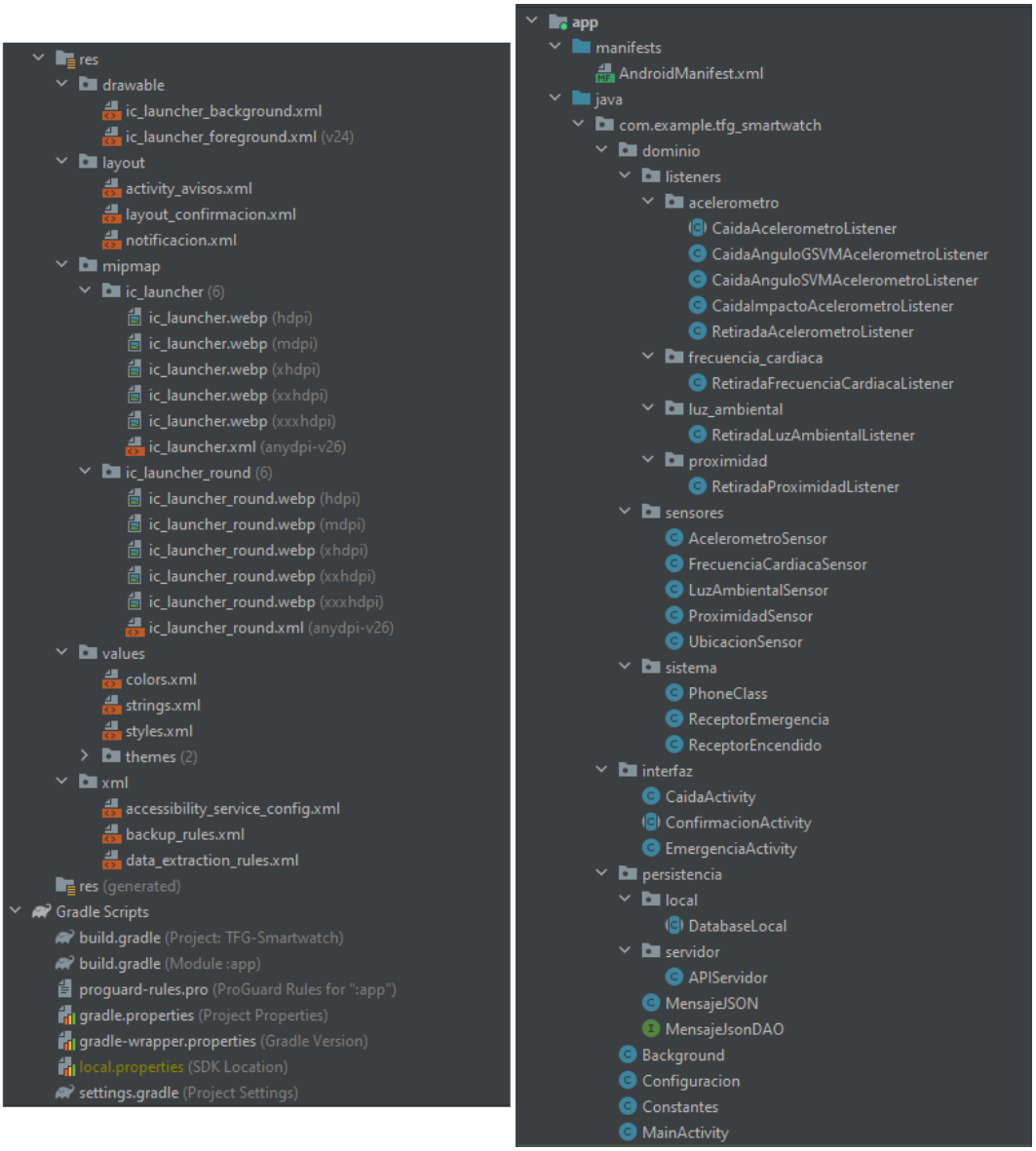

Figura 4.16: Estructura de archivos final del proyecto

Capítulo 5

Implementación

A lo largo de este capítulo se detalla cómo han sido implementadas las diferentes funcionalidades añadidas a lo largo de este proyecto. Las funcionalidades necesarias para el correcto desarrollo del proyecto que ya se desarrollaron en la aplicación tomada como punto de partida se comentan brevemente, pero no se entra en detalle de las mismas. Se desarrolla de manera más precisa cómo se realiza la detección de caídas (con sus 3 modos de detección), la detección de consciencia tras las mismas, la pausa de monitorización de caída tras la retirada del dispositivo, la recolección de datos de las caídas para su posterior análisis, la configuración de preferencias de monitorización y la descarga de configuración desde el servidor.

5.1. Permisos

Para poder hacer uso de los diferentes sensores del smartwatch, así como los servicios que proporciona, es necesario pedir los permisos correspondientes al usuario. Para ello, tan solo es necesario incorporar estos permisos en el archivo *AndroidManifest.xml* del proyecto. Este archivo se puede encontrar en todas las aplicaciones Android, pues incluye información de la aplicación que resulta esencial para las herramientas de creación y el sistema operativo de Android [76].

Para las funcionalidades de este proyecto no se necesitan permisos más allá de los que ya requería la aplicación tomada como punto de partida, por lo que se conservan de la misma forma que se encuentran en la misma [8].

5.2. Sensores

Para la realización de este proyecto se hace uso del acelerómetro del dispositivo, al cual se puede acceder gracias a la clase proporcionada por las librerías de Android *SensorManager* [77]. Mediante esta clase podemos obtener mediciones del acelerómetro haciendo uso del método *getDefaulSensor(int*

tipoSensor) y registrando un *listener* de tipo *SensorEventListener* [78].

Para registrar este *listener*, se ha de hacer uso del método de la clase *SensorManager*: *registerListener()*. Este método toma como parámetros el *listener* a registrar, el sensor a monitorizar y un periodo entre toma de medidas. Una vez registrado, se obtienen valores mediante el método *onSensorChanged()*, que se ejecuta cada vez que se toma una medida del sensor correspondiente.

Para las funcionalidades de este proyecto no es necesario añadir nuevos *listeners* a sensores, pues el único sensor a utilizar, el acelerómetro, ya se utilizaba en el proyecto tomado como punto de partida [8].

Para las funcionalidades de este proyecto se añaden dos *listeners* nuevos al acelerómetro. Dado que es un sensor que ya contaba con *listener* en el proyecto tomado como punto de partida, el añadido de estos nuevos *listeners* se realiza tal y como estaba en ese proyecto [8].

Sin embargo, un cambio a tener en cuenta es el intervalo de toma de mediciones para el *listener*. A pesar de que Android no proporciona valores exactos para los retrasos utilizados en sus configuraciones por defecto [77], las pruebas detalladas en la sección 6.1: Retraso en los sensores, muestran que el originalmente utilizado **(SENSOR_DELAY_NORMAL)** toma medidas aproximadamente cada 200 milisegundos. Esta frecuencia no es suficiente para la detección de caídas, pues se trata de eventos muy rápidos, por lo que, para los nuevos *listeners* del acelerómetro, se ha de hacer uso del intervalo **(SENSOR_DELAY_FASTEST)**, que toma medidas aproximadamente cada 20 milisegundos.

5.3. Monitorización y procesamiento de la ubicación

A la hora de notificar una emergencia al servidor, se añade la ubicación GPS del dispositivo. Para obtener esta ubicación GPS se hace uso de la API de proveedor *FusedLocationProvider* [79], pues al proporcionarla el propio Google, es la que generalmente mejores resultados aporta para dispositivos basados en Android y es por ello la más utilizada. Una vez obtenido el proveedor, se hace uso de la clase *LocationRequest* [80] (también de Google) para realizar la monitorización de la ubicación con el periodo de mediciones que se requiera.

Una vez configurada la monitorización, cada periodo requerido se obtienen las lecturas de la ubicación. Una vez recibidas estas lecturas, se genera un archivo JSON el cual se procesa de una de dos formas: en caso de haber conexión a internet se envía al servidor mediante una petición HTTP de tipo PUSH donde JSON se añade como cuerpo. En caso de no haber conexión, el JSON se almacena en una base de datos local, implementada mediante la biblioteca Room [58] hasta que haya conexión de nuevo. En dicho momento, todos los datos almacenados se envían al servidor y la BD local se vacía.

El JSON formado se compone de los campos: type, androidId, latitude, longitude, altitude, accuracy, battery, source, deviceTimestamp.

El campo **type** se trata de un *String* que indica qué tipo de notificación se está enviando al servidor. Esta puede ser una de las siguientes cinco:

MONITORING Se trata de una notificación de monitorización normal.

TAKEOFF Se trata de una notificación de retirada del dispositivo.

EMERGENCY Se trata de una notificación de emergencia activada al accionar el botón.

SEVEREFALL Se trata de una notificación al detectarse una caída grave.

MINORFALL Se trata de una notificación al detectarse una caída leve.

FALSEFALL Se trata de una notificación al detectarse una caída pero el usuario ha indicado que es falsa.

El resto de parámetros se mantienen tal como se utilizan en el proyecto tomado como punto de partida [8], por lo que no se entrará en detalle sobre ellos.

5.4. Ejecución en segundo plano

El objetivo de la aplicación es que se mantenga continuamente activa para monitorizar al usuario del smartwatch. Para ello es necesario que esta se ejecute de manera constante en segundo plano. Por ello, cuando se inicia la aplicación su actividad principal tan solo tiene dos funciones: solicitar los permisos necesarios para la monitorización y ceder el resto del control al segundo plano. Una vez se ha dado paso a la ejecución en segundo plano, se genera una notificación permanente que indica que la aplicación se está ejecutando.

Para la realización de este proyecto no es necesario modificar la configuración del funcionamiento en segundo plano, por lo que se conserva de la misma forma que se encuentra en la aplicación tomada como punto de partida [8].

5.5. Detección de la caída

Para desarrollar la detección de caída se ha de utilizar el acelerómetro del dispositivo, cuyos valores se obtienen haciendo uso de las clases mencionadas en la sección 5.2: Sensores.

Como se menciona en la sección 1.4.4: Enfoque seleccionado para el proyecto, el algoritmo a utilizar se trata de uno de doble comprobación de umbrales, por lo que su estructura básica es la siguiente:

```
if (medida1 < umbral){
    if (medida2 > umbral){
        se ha detectado caída, notificar
    }
}
```

5.5.1. Enfoque preferido

En este caso, las medidas a utilizar son *GSVM* y θ (explicadas en la sección 1.4.4: Enfoque seleccionado para el proyecto), por lo que primero se ha de calcularlas. Los valores de la medición del acelerómetro se reciben en un vector denominado *valoresActuales*, cuyas componentes son:

- **valoresActuales[0]** Medida del acelerómetro en el eje X.
- **valoresActuales[1]** Medida del acelerómetro en el eje Y.
- **valoresActuales[2]** Medida del acelerómetro en el eje Z.

Para el cálculo del ángulo, es necesario saber qué eje se ha de utilizar como referente de la gravedad, para lo cual se lleva a cabo la primera prueba de la sección 6.2: Medición del acelerómetro en reposo. De esta forma se concluye que el eje de referencia es X.

Con esto en mente, las medidas a utilizar se calculan de la siguiente forma:

```
// Se puede prescindir de estos 3 primeros calculos e incluirlos en las fórmulas,
// pero se mantienen por claridad
double x2 = valoresActuales[0]*valoresActuales[0];
double y2 = valoresActuales[1]*valoresActuales[1];
double z2 = valoresActuales[2]*valoresActuales[2];

double svm = Math.sqrt( x2+y2+z2 );
double angle = Math.atan( Math.sqrt( y2+z2 )/x2 ) * ( 180/Math.PI );
double gsvm = svm * ( angle/90 );
```

Una vez calculados estos valores, se puede proceder a la comprobación de umbrales de los mismos. Sabiendo que el acelerómetro en reposo sí recoge la aceleración de la gravedad (como se muestra en la prueba de la sección 6.2: Medición del acelerómetro en reposo), se puede partir de los umbrales utilizados en el estudio del cual se obtienen las medidas [37]. Finalmente, tras las pruebas correspondientes (sección 6.3) se concluye que los mejores umbrales son 0.8 para *GSVM* y 60 para θ.

Con el objetivo de minimizar la detección de falsas caídas, una vez la primera medida (*GSVM*) supera el umbral, se entra en un estado en el cual se toman 25 medidas consecutivas de θ. Si al menos 17 de estas medidas superan el umbral (un 70 %), se ha dado caída y se procesa la misma.

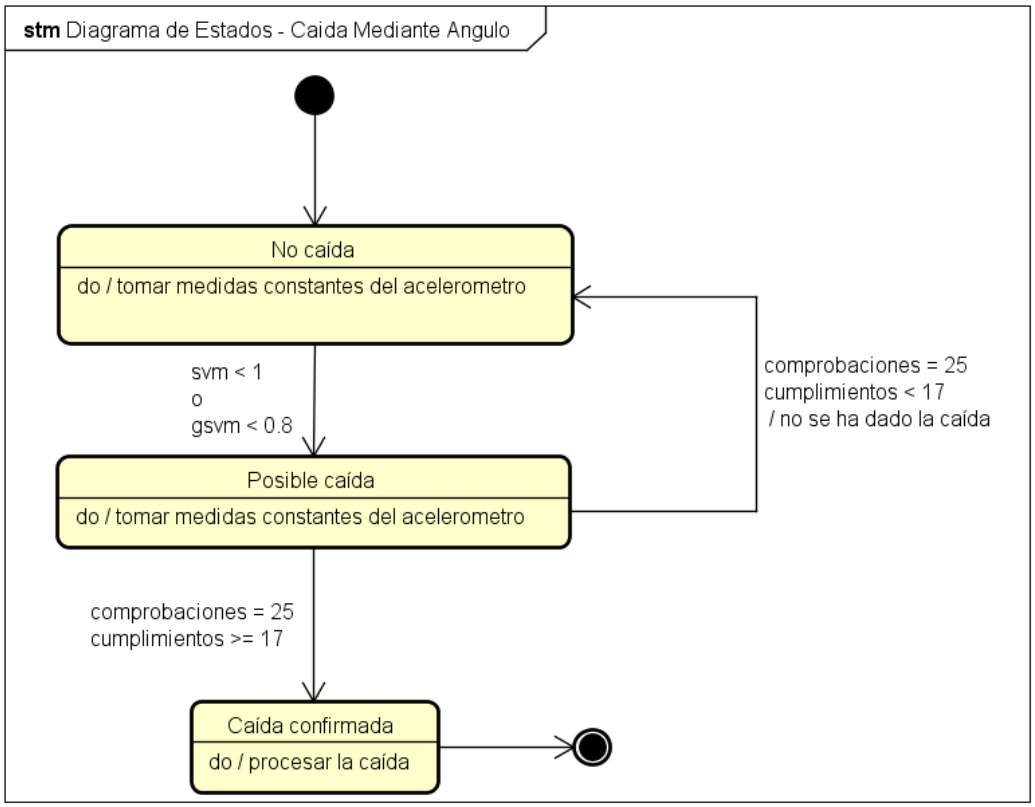

Figura 5.1: Diagrama de estados del algoritmo de detección de caídas basado en ángulos

Este algoritmo funciona de manera óptima para dispositivos que no se someten a grandes rotaciones, pues *GSVM* es muy sensible al ángulo de la aceleración. Sin embargo, debido a que en este proyecto se quiere realizar una aplicación para relojes inteligentes, esta medida no es la más adecuada. Debido a la forma en la que se colocan estos dispositivos, se someten a una gran cantidad de giros de muñeca, lo cual hace que, utilizando *GSVM*, se detecten caídas falsas de manera continua, dependiendo de la posición en la que se encuentre la muñeca. Por ello, se acaba sustituyendo, como primer umbral a superar, por *SVM*, cuyo mejor valor es 1 (obtenido en la sección 6.3). Aun así, los resultados obtenidos para este algoritmo no son los mejores, tal como se detalla en la sección 6.4: Comparación de algoritmos de detección de caídas.

5.5.2. Enfoque alternativo

Dada las debilidades de la detección mediante *GSVM*/*SVM* y θ (angle), se plantea la alternativa mediante detección del impacto. Esta tiene una implementación muy simple, pues tan solo se trata del cumplimiento de dos umbrales. Primero, se empieza calculando SVM para todas las medidas del acelerómetro, tal como se hace en el enfoque anterior:

```
double x2 = valoresActuales[0]*valoresActuales[0];
double y2 = valoresActuales[1]*valoresActuales[1];
double z2 = valoresActuales[2]*valoresActuales[2];
```

```
double svm = Math.sqrt( x2+y2+z2 );
```

Si *SVM* es menor que el umbral seleccionado para la detección de caída (el cual es 1, calculado en la sección 6.3), se procede a comprobar si alguna de las sucesivas medidas es mayor que el umbral seleccionado para la detección del impacto, (109). Este segundo umbral se tiene que detectar en un máximo de 2 segundos (tiempo máximo que podría durar una caída), tiempo tras el cual se asume que la caída no se ha dado. El tiempo se tiene en cuenta de la siguiente forma:

```
if(svm < 1){
    // se ha detectado un primer indicio de caida
    timestampCaida = System.currentTimeMillis(); // el tiempo se obtiene en milisegundos
}

...
(Siguiente monitorizacion del acelerometro)
...

tiempoTranscurrido = System.currentTimeMillis() - timestampCaida;
if(tiempoTranscurrido > 2000){
    // se ha pasado el tiempo, se resetea la monitorizacion
}
```

Si en esa ventana de tiempo se supera el segundo umbral, se ha dado caída y se procesa la misma.

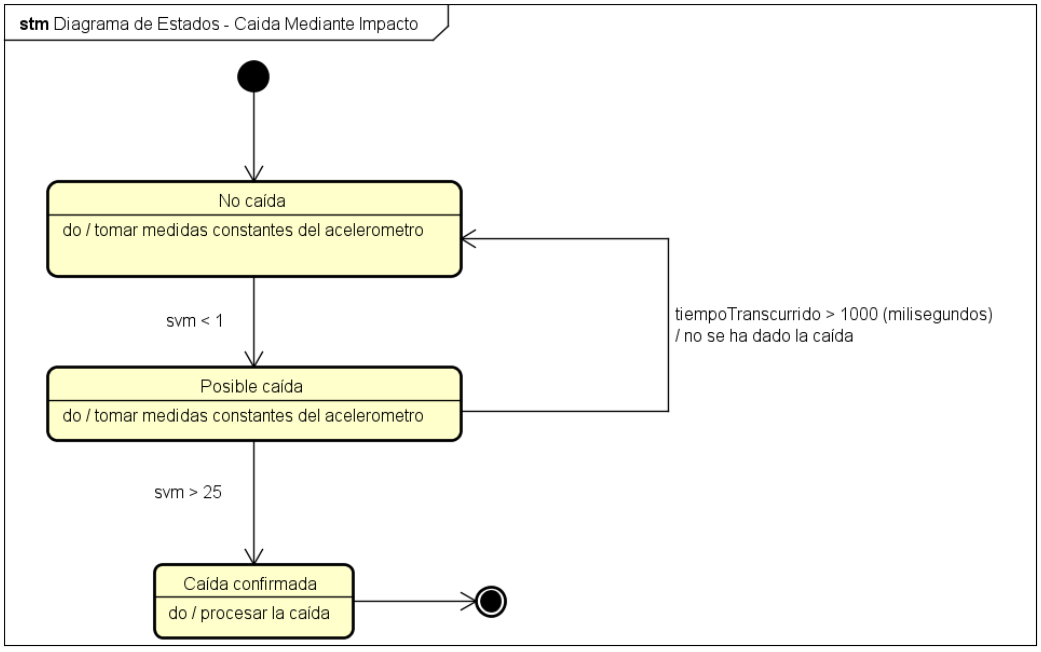

Figura 5.2: Diagrama de estados del algoritmo de detección de caídas basado en impacto

Este enfoque puede obtener mejores resultados en cuanto a la detección de caídas verdaderas, sin embargo, es más propenso a detectar falsas caídas cuando se golpea el dispositivo rápidamente tras sentarse, tumbarse en la cama, etc.

5.5.3. Implementación final

Dada la falta de tiempo para poder comparar ambos algoritmos a una mayor escala, la versión final del proyecto permite elegir entre qué algoritmo utilizar. Para ello, es necesario modificar el archivo *Constantes.java* antes de la compilación de la aplicación. En este, se encuentra el siguiente atributo:

```
public static final int MODO_MONITORIZACION_CAIDA = 1;
```

Dependiendo del valor asignado a este atributo se utilizará uno de los siguientes modos de detección de caída:

- **MODO =1**: Se realiza la comprobación de primer umbral *GSVM* y sucesivas comprobaciones de θ.
- **MODO =2**: Se realiza la comprobación de primer umbral *SVM* y sucesivas comprobaciones de θ.
- **MODO =3. Modo por defecto**: Se realiza la doble comprobación de *SVM*, para caída y para impacto.

De esta forma, al incluir las 3 alternativas, se facilitan las futuras pruebas del algoritmo, para determinar de una forma más precisa cuál de ellos funciona mejor y qué umbrales son necesarios para obtener resultados óptimos en casos de uso menos controlados.

5.6. Comprobación de consciencia tras caída

Cuando el sistema detecta una caída nos encontramos en una de las siguientes tres situaciones:

- El usuario se ha caído y se encuentra consciente. Este caso se trata como una caída leve pues, aunque los daños ocasionados por la caída pueden ir de muy superficiales (como una rozadura) a realmente graves (como una fractura), al conservarse la consciencia el usuario puede decidir si es necesario iniciar el proceso de emergencia o no.
- El usuario se ha caído y ha perdido la consciencia. Este caso se trata como una caída grave, pues no se perdería la consciencia en una caída leve. En este caso es importante notificar al servidor para que desde ahí se iniciase el proceso de emergencia. Estando el usuario inconsciente, no tendría sentido iniciar el proceso de emergencia desde su lado, pues no podría contestar a la llamada.

- El usuario no se ha caído, se ha detectado una caída falsa. Este caso se trata de una caída falsa. Es interesante recogerlas para poder analizar el funcionamiento del algoritmo.

Debido a que una caída se puede dar por muchos motivos, no es posible detectar la consciencia del usuario analizando los sensores tras la caída de una forma tan simple como que al detectar movimiento se encuentre consciente y si no se detecta esté inconsciente (por ejemplo, un usuario puede caer, golpearse la cabeza y tener convulsiones debido al golpe, lo cual detectaría movimiento a pesar de que la persona esté inconsciente). Por ello, se utiliza una pantalla de confirmación de caída (figura 5.3), de forma similar a la que se utiliza para confirmar una llamada de emergencia (puede variar entre dispositivos). Debido a que el funcionamiento de ambas pantallas de confirmación es similar, se agrupan como hijos de una misma *Activity* abstracta (detallado en la sección 4.2.1.2).

Al tratarse de una pantalla que puede mostrarse en cualquier momento en el reloj, sin una acción directa del usuario, es importante que su aparición llame la atención. Para ello, cuando se crea, emite un sonido y hace vibrar el dispositivo, haciendo uso de las herramientas de *Android* enfocadas a ello.

Para el sonido, se utiliza uno de los muchos que ofrece Android por defecto, el cual se puede obtener con el método *'getDefaultUri(...)'* de la clase *'RingtoneManager'*. Este método recibe un atributo para identificar el recurso a utilizar que, en este caso, se trata del sonido por defecto de las notificaciones. Una vez se tiene el recurso a utilizar, se crea el *Ringtone* que se encargará de hacer sonar el recurso seleccionado, haciendo uso del método *'getRingtone(...)'* de la misma clase (*'RingtoneManager'*).

Para la vibración, se crea un efecto de vibración simple (siempre vibra a la misma intensidad) mediante el método *'createOneShot(...)'* de la clase *'VibrationEffect'*. Este recibe como atributos la duración en milisegundos y la intensidad con la que vibrar. Para asegurarse de que el usuario lo nota, se utiliza una duración de un segundo y medio. Una vez creado el efecto, se obtiene el servicio de vibrador mediante el método *'getSystemService(...)'* del contexto de la aplicación y se le hace vibrar con el efecto recién creado.

Una vez se ha llamado la atención del usuario, la pantalla le muestra un texto que avisa de que se ha detectado la caída y espera a recibir una respuesta mediante la pulsación de uno de los dos botones **"SI"** o **"NO"**. Al mismo tiempo, se inicia una cuenta atrás de 5 segundos, mostrando el tiempo restante al lado del botón **"NO"**. Gracias a esta pantalla se puede diferenciar las tres situaciones de la forma siguiente:

- En caso de que se pulse el botón **"SI"** se está confirmando la caída y el usuario se encuentra consciente (pues ha podido pulsar el botón). Se trata de una caída leve.
- En caso de que se pulse el botón **"NO"** se está negando la caída. Se trata de una caída falsa.
- En caso de que el contador llegue a cero sin que el usuario pulse ningún botón, probablemente se encuentre inconsciente. Se trata de una caída grave.

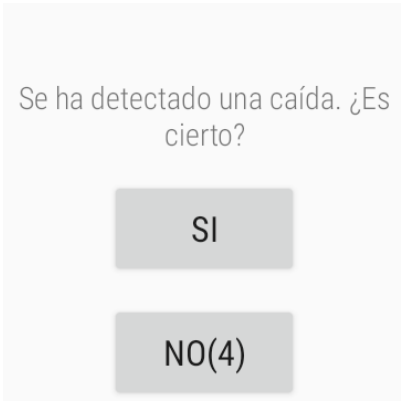

Figura 5.3: Captura de la pantalla que se muestra tras detectar una caída

5.7. Pausa de la detección dada la retirada del dispositivo

Sabiendo que los dispositivos utilizados son smartwatches, es necesario tener en cuenta que estos son fácilmente retirables. Puede darse el caso en que el usuario se lo quite y lo lance o deje caer, por lo que se podría detectar caída mientras el usuario no utiliza el reloj. Por ello, es necesario pausar la detección de caídas cuando se detecta la retirada (así como reanudarla cuando se vuelve a colocar el reloj).

Esta retirada ya se tenía en cuenta en el proyecto tomado como punto de partida [8], pero se almacenaba y utilizaba de forma local mediante el atributo *'retirado'* de la clase *'UbicacionClass'*. Este se sustituye por la variable de mismo nombre en la clase *'Configuracion'*, que es globalmente accesible gracias a implementar el patrón Singleton (documentado en la sección 4.2.5).

Una vez hecho esto, solo se realiza la comprobación de caída cuando el dispositivo no está retirado:

```
// Se obtiene la configuracion
Configuracion config = Configuracion.getInstance();
...

// Se realiza la comprobación
if (!config.isRetirado()){
    comprobarCaida(valores);
}
```

5.8. Modo debug

Para un correcto funcionamiento de los algoritmos de detección de caídas es necesario llevar a cabo muchas pruebas comparando los resultados obtenidos cuando se modifican los umbrales utilizados. Por ello, se tiene que contar con una forma de obtener estos datos cuando se están realizando pruebas.

Para ello, se hace uso de la clase *'Log'* de Android, la cual permite mostrar datos en el Log de la aplicación. Para cualquier medida tomada del acelerómetro se muestran los valores calculados para esa medida. De esta forma, en caso de que no se estén detectando caídas, se puede analizar cuáles son los datos obtenidos. Para ello se utiliza el siguiente método para mostrar los datos en el modo información:

```
Log.i("ETIQUETA_SIGNIFICATIVA", "datos a mostrar");
```

En caso de que se dé la detección de una caída, se guardan todos los datos que han llevado a la misma. Para las detecciones mediante ángulo, se guarda el umbral inicial superado (SVM o GSVM) y las siguientes 25 mediciones del ángulo. Por último, se guardan cuántas de estas han superado el umbral. En el caso de la detección mediante impacto, solo se guardan los dos valores que han superado los umbrales establecidos. Tras ello, en la pantalla de confirmación de caída, se guarda si la caída detectada ha sido falsa, leve o grave.

Estos datos se tratan de almacenar en un fichero de texto de nombre "datos", seguido de una serie de números que representa el instante en el cual se detectó la caída. De esta forma, estos ficheros pueden extraerse del dispositivo posteriormente para poder ser analizados. Como el almacenamiento en los relojes inteligentes es escaso, se trata de escribir este fichero en el almacenamiento externo del mismo, obteniendo la ruta donde guardarlo mediante el método *'getExternalFilesDir()'* de Android. Hay que tener en cuenta que la gran mayoría de relojes inteligentes no tienen soporte para un almacenamiento externo, por lo que, en caso de que el método anterior falle, se utilizará el método *'getFilesDir()'*, una alternativa que proporciona una ruta en el almacenamiento interno del smartwatch. Una vez obtenida la ruta en la que almacenar el fichero, se puede escribir el mismo haciendo uso de la clase *'FileOutputStream'* de java.

Ambos métodos de escritura son bastante propensos a fallos, pues no están completamente estandarizados y hay dispositivos que no permiten su uso, dispositivos que permiten el uso de tan solo uno de ellos y dispositivos que, aunque aparentemente permiten su uso, borran el archivo inmediatamente tras crearlo para mantener el almacenamiento del reloj inteligente relativamente vacío. Es por ello que, a parte de intentar escribir estos datos en almacenamiento persistente, también se guardan y se muestran por el Log de información, haciendo uso del método mencionado previamente.

Por último, debido a que la memoria y el almacenamiento de un reloj inteligente son bastante reducidos, esta forma de recolección de datos solo debe realizarse cuando la aplicación se esté ejecutando en modo desarrollador, que es cuando verdaderamente se analizarán los resultados. Para ello, en la clase Constantes, se incluye el siguiente atributo:

```
public static final boolean MODO_DEBUG = true;
```

Dependiendo del valor asignado a este atributo se utilizará uno de los siguientes modos de ejecución:

- **MODO =true**: Es el modo para desarrolladores. Se muestran por consola los datos de las medidas tomadas por el acelerómetro y, dada una caída, se almacena los mismos en un archivo.

- **MODO =false. Modo por defecto**: Es el modo pasa uso normal de usuarios. No se tratan de mostrar datos por consola, ni se almacenan los mismos dada una caída.

De esta forma, tan solo se gastan recursos en la recogida de estos datos cuando verdaderamente van a ser observados y utilizados.

De esta forma se almacenan los datos junto con el resto de archivos de la aplicación, por lo que la forma de obtenerlo es conectar el dispositivo a un ordenador y navegar hasta la carpeta donde encontramos estos ficheros. La ubicación de esta carpeta varía de un dispositivo a otro, por lo que hay que localizarla. Dado que un smartwatch que utiliza WearOS ha de conectarse a un teléfono móvil, para los dispositivos que lo permitan, se puede localizar esta carpeta de datos consultando las propiedades de la aplicación en: Ajustes >Almacenamiento >Otras aplicaciones >TFG-Monitorizacion.

5.9. Selección de monitorizaciones a llevar a cabo

Con el objetivo de preservar la batería del reloj inteligente, es importante que un reloj solo lleve a cabo las monitorizaciones necesarias para el usuario que va a utilizarlo. Es decir, para cada ejecución de la aplicación se ha de poder elegir si llevar a cabo todas las monitorizaciones (ubicación, retirada y caída), solo la imprescindible (ubicación) o una selección personalizada de las monitorizaciones disponibles. Teniendo en cuenta que la monitorización de la ubicación es imprescindible, tan solo la caída y la retirada son activables/desactivables.

La activación o desactivación de estas monitorizaciones se tiene en cuenta mediante los atributos booleanos correspondientes en la clase *'Configuracion'*. En el caso de añadir nuevas monitorizaciones futuras, se tendrán que añadir nuevos atributos para las mismas. Las correspondientes a la retirada y la caída con las siguientes:

```
private boolean monitoreoCaidaActivo;
private boolean monitoreoRetiradaActivo;
```

La aplicación arranca con ambos atributos activados (valor=true) por defecto para asegurarse de que ambas monitorizaciones son capaces de ejecutarse sin dar lugar a errores. No es hasta que se actualiza la configuración que tan solo se activan aquellos necesarios.

Para esta activación/desactivación se aprovecha el hecho de que las monitorizaciones se realicen mediante *listeners*. Se separa cada monitorización desactivable (retirada y caída) en un *listener* personalizado, dando lugar a la separación del *listener* del acelerómetro en los *listeners 'CaidaAcelerometroListener'* y *'RetiradaAcelerometroListener'*. Además, para la retirada también se tiene en cuenta la monitorización mediante sensor de proximidad (*'RetiradaProximidadListener'*). Estos *listeners* son administrados mediante la clase *'SensorManager'* de Android, la cual permite registrar/eliminar *listeners* de los diferentes sensores haciendo uso de los métodos *'registerListener(...)'* y *'unregisterListener(...)'*.

Gracias a ello, se puede crear un método que actualice el registro de *listeners* (dependiendo de la configuración) de la forma siguiente:

```
// Obtener la configuracion
Configuracion config = Configuracion.getInstance();

if (config.isMonitoreo_FUNCION_Activo()) {
    // Si la monitorizacion está activa, se registra el listener
    // No es necesario comprobar si ya esta registrado pues lo hace android y
    // no se puede tener el mismo listener registrado dos veces
    sensorManager.registerListener(_LISTENER_, _SENSOR_, _DELAY_);
} else {
    // Si la monitorizacion no está activa, se elimina el listener
    sensorManager.unregisterListener(_LISTENER_);
}
```

Donde se han de rellenar los campos siguientes:

- **FUNCION** Monitorización a llevar a cabo (Caida/Retirada).
- **LISTENER** *Listener* que se encarga de la monitorización (caidaAcelerometroListener / retiradaAcelerometroListener / retiradaProximidadListener).
- **SENSOR** Sensor cuyos *listeners* actualizar (acelerometroSensor / proximidadSensor).
- **DELAY** Intervalo en el cual tomar medidas (valores explicados en la sección 5.2).

Dado que los *listeners* de los sensores solo se encuentran activos cuando están registrados a un sensor concreto, en caso de que la monitorización no esté activa, no tomarán medidas, preservando la batería del dispositivo.

5.10. Descarga de la configuración.

La configuración de la aplicación ha de poder ser cambiada en tiempo de ejecución. Dado que la aplicación final está pensada para ser utilizada en segundo plano con la mínima interacción posible del usuario, se decide que esta configuración se descargue desde servidor. Esta descarga no se puede realizar al comienzo de la ejecución, pues, para que la conexión funcione, es necesario que el ID del dispositivo haya sido registrado en el servidor de forma manual. Este ID se muestra en la notificación permanente una vez la aplicación se ejecuta correctamente por primera vez, lo cual hace imposible la conexión antes de la aparición de esta notificación permanente.

Por ello, la descarga de la configuración se 'esconde' en la pantalla de notificación de emergencia (figura 5.4). Al hacer click 5 veces en el texto de esta pantalla, se inicia el proceso de descarga.

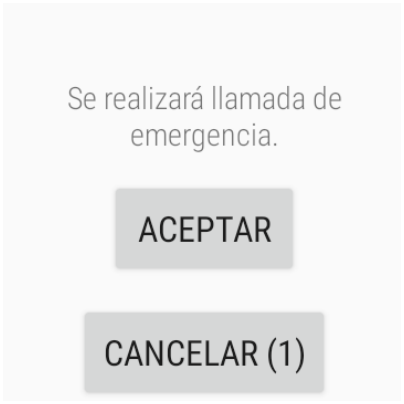

Figura 5.4: Pantalla para aceptar/cancelar la alerta de emergencia.

Para realizar esta descarga se realizan los siguientes pasos:

1. Se prepara una petición *GET* al servidor. Para ello se obtiene la URL necesaria mediante la clase *'URL'* de java, la cual permite crear una nueva instancia indicándole (mediante una cadena de caracteres) la URL a la que conectarse. Esta URL ha de estar compuesta de la dirección de descarga de configuración del servidor y del ID del dispositivo. Una vez indicada la dirección, se puede abrir una conexión (se crea una clase *'HttpURLConnection'*) mediante el método *'openConnection()'* de *'URL'*, tras lo cual se pueden recibir datos gracias a su método *'conexion.getInputStream()'*.
2. Una vez recibida la respuesta del servidor, se procesa la misma como un *'JSONObject'* de nombre *'nuevaConfiguracion'*, el cual está compuesto por pares configuración-valor. En caso de que la conexión con el servidor o el procesamiento de la respuesta falle, la configuración se tratará de un objeto *'null'*.
3. Se actualizan los valores del Singleton *'Configuracion'* con el objeto JSON obtenido en el paso anterior. Para ello se han de utilizar los diferentes métodos de *'JSONObject'* para obtener valores del mismo (por ejemplo, para la monitorización de caída se habría de usar *'nuevaConfiguracion.getBoolean("fall")'*). En caso de que haya habido errores, se mantiene la configuración por defecto.
4. Una vez se ha actualizado la Configuración, es necesario notificar a los Managers de los Sensores, para que activen/desactiven las monitorizaciones pertinentes.

Por último, es importante destacar que esta funcionalidad depende de que el servidor esté funcionando y contenga datos para las configuraciones requeridas por la aplicación. En caso de que esto no sea así, aunque sí se trate de hacer la descarga, se ejecutará en su modo por defecto. En este caso se ejecuta con todas las monitorizaciones activadas a no ser que el fichero *'Configuracion.java'* sea modificado antes de la compilación de la aplicación.

Capítulo 6

Pruebas

A lo largo de este capítulo se describirán y documentarán las pruebas llevadas a cabo para ajustar los umbrales del proyecto y comprobar si el mismo cumple los requisitos de calidad necesarios para su aceptación.

6.1. Retraso en la medición del acelerómetro

A la hora de tomar medidas de los sensores de un dispositivo es importante saber cada cuánto tiempo mínimo se están recibiendo estas medidas. En el caso de la detección de caídas es especialmente crítico, pues una caída es un evento que dura muy poco tiempo y si el acelerómetro no toma medidas lo suficientemente rápido puede que se dé una caída y pase desapercibida. Por ello es importante asegurarse de que la frecuencia utilizada es apropiada para la detección.

En este caso se comparan dos frecuencias por defecto proporcionadas por la utilidad *SensorManager* de Android. Estas son **SENSOR_DELAY_NORMAL** y **SENSOR_DELAY_FASTEST**.

Nombre	Delay del acelerómetro.
Descripción	Cada vez que se toma una medida del acelerómetro se guarda el *timestamp* de la misma. Una vez tomadas medidas durante 5 segundos, se calcula el tiempo promedio que pasa entre las mismas.
Resultado	Para la frecuencia SENSOR_DELAY_NORMAL el tiempo promedio entre medidas es de 200ms (aproximadamente se toman 5 medidas cada segundo). Para la frecuencia SENSOR_DELAY_FASTEST el tiempo promedio entre medidas es de 20ms (aproximadamente se toman 50 medidas cada segundo).
Conclusión	La frecuencia proporcionada por SENSOR_DELAY_NORMAL no es lo suficientemente elevada para la detección de caídas, por lo que se ha de utilizar SENSOR_DELAY_FASTEST.

Tabla 6.1: Prueba delay acelerómetro

6.2. Medición del acelerómetro en reposo

Para poder tomar como referencia las fórmulas y los umbrales obtenidos del estudio mencionado [37], hay que asegurarse de que las condiciones en reposo son las mismas. Es decir, hay que comprobar que las medidas que obtiene el acelerómetro en reposo se correspondan a la gravedad (esta no es eliminada por Android mediante software) y que el eje que recibe mayor fuerza de la gravedad es el eje X.

Para ello, se hace uso de la utilidad proporcionada por Google para la prueba de los sensores de los dispositivos que utilizan WearOS [81], la cual muestra información sobre el sensor seleccionado.

Nombre	Medidas del acelerómetro en reposo colocado sobre una mesa.
Descripción	Se observan las medidas tomadas por los diferentes ejes del acelerómetro mientras se mantiene el reloj colocado plano sobre una mesa, con la pantalla hacia arriba y sin moverlo durante 5 segundos.
Resultado	Se obtienen las medidas aproximadas de -0,048 m/s para el eje x, -0,213 m/s para el eje y, 9,903 m/s para el eje z.
Conclusión	Las medidas devueltas por el sensor se tratan de unos valores que sí incluyen la aceleración de la gravedad, por lo que se pueden tomar como referencia los umbrales supuestos.

Tabla 6.2: Prueba de gravedad del acelerómetro en reposo

Nombre	Medidas del acelerómetro en reposo colocado en la muñeca.
Descripción	Se observan las medidas tomadas por los diferentes ejes del acelerómetro mientras se mantiene el reloj colocado en la muñeca, estando de pie con los bazos a los lados, intentando moverse lo menos posible, durante 5 segundos. Dado que solo se necesita la orientación, no se toman valores precisos.
Resultado	Colocado en la muñeca derecha se obtienen valores aproximados de 9 para el eje x, 0 para el eje y, 0 para el eje z. Colocado en la muñeca izquierda se obtienen valores aproximados de -9 para el eje x, 0 para el eje y, 0 para el eje z.
Conclusión	El eje que soporta la mayor atracción de la gravedad es el eje X, por lo tanto, sí se puede utilizar como referencia para el cálculo del ángulo.

Tabla 6.3: Prueba de ejes del acelerómetro en reposo

6.3. Umbrales apropiados para la detección de caídas

Para asegurarnos de que la detección de caídas es apropiada, es necesario ajustar los umbrales de las medidas de tal forma que el número de detecciones reales sea el mayor posible, mientras que el número de falsos positivos se mantenga como el menor posible.

Primero se comienza observando los datos obtenidos en el caso de que sí se haya dado una caída, para comprobar si los posibles umbrales se asemejan a *GSVM* = 1g y $\theta = 60º$ (los documentados en

el estudio referenciado [37]). En el caso de *SVM* y del Impacto, no se han encontrado datos concretos proporcionados por un estudio. Para *SVM*, la mayoría de estudios sugieren 1g, mientras que para el impacto depende ampliamente del dispositivo utilizado, por lo que se habrá de obtener mediante pruebas.

Primero, se calculan estos valores haciendo uso de un set de datos obtenido tomando medidas para un acelerómetro de tres ejes [82]. Este set está compuesto por diferentes carpetas de datos, cada una correspondiente a la actividad que se estaba realizando al tomar las medidas. En este caso nos centraremos en las carpetas *'runFall'* y *'walkFall'*, que se trata de medidas tomadas cuando se detecta una caída mientras se corre y se anda, respectivamente.

Nombre	Observación umbrales caída para *SVM*, *GSVM* Y θ.
Descripción	Haciendo uso de datos del conjunto de *'runFall'* y de **'walkFall'**, se calcula *SVM* y *GSVM* para las 10 primeras medidas (aproximadamente es en esas medidas cuando se detectaría el comienzo de la caída) y se calcula θ para las siguientes 25 medidas (las mismas que se usa en el algoritmo).
Resultado	Eliminando *outliers*, se obtienen los siguientes valores promedio: -Caídas mientras se corre: 0.995 para *SVM*, 0.646 para *GSVM* y 54.478 para θ, con una desviación estándar de 0.041, 0.025 y 0.926 respectivamente. -Caídas mientras se anda: 0.988 para *SVM*, 0.641 para *GSVM* y 58.399 para θ, con una desviación estándar de 0.054, 0.045 y 1.114 respectivamente.
Conclusión	Los valores obtenidos se ajustan al umbral supuesto en el estudio. Dada la variabilidad, se han de realizar pruebas para ajustar estos umbrales en los rangos siguientes: [0.934, 1.042] para *SVM*, [0.686, 1] para *GSVM* y [59.513, 60] para θ.

Tabla 6.4: Prueba de observación de umbrales para caída mediante set de datos

Desafortunadamente, no he encontrado un set de datos que incluya el momento del golpe final, por lo que este se habrá de calcular mediante pruebas específicas. Para ello, se tendrá en cuenta que la estatura promedio en España es de 169cm [83] y que la muñeca se sitúa aproximadamente a mitad de cuerpo, por lo que el reloj caerá desde una altura aproximada de 84cm. Desde esta altura, se comprueba qué valor máximo toma *SVM* al impacto.

Nombre	Observación umbrales caída para el impacto.
Descripción	Se deja caer el reloj un total de 20 veces, observando en cada caso cuál ha sido el valor máximo alcanzado para *SVM*, el cual corresponde al momento del impacto.
Resultado	Se obtiene que el valor promedio es de 120.961, con una desviación estándar de 11.944.
Conclusión	Se ha de probar en el rango siguiente: [109, 133].

Tabla 6.5: Prueba de observación de umbrales para caída mediante prueba específica

Una vez calculados unos posibles umbrales, se ha de ajustar los mismos para reducir la falsa detección de caídas. Esto se puede conseguir comenzando por el umbral superior e ir reduciéndolo hasta el punto en que un cambio no detecte verdaderas caídas. Debido a que la diferencia obtenida en la

prueba anterior para *SVM* para caída libre y θ es muy pequeña, se utilizarán los umbrales de **1** para *SVM* y **60º** para θ y tan solo habrá que ajustar el umbral de *GSVM*. Dada la prueba anterior, donde se concluye que hay que probar umbrales entre 0.686 y 1, se decide realizar incrementos de 0.1.

Nombre	Ajuste umbrales GSVM.
Descripción	Con el reloj inteligente puesto se realizan 10 caídas para comprobar la precisión del algoritmo (caídas ciertas detectadas). También se realizan 10 acciones de sentarse (pues es la acción más parecida a caerse) para comprobar la exactitud del algoritmo (falsas caídas detectadas). Esto se realiza para cada valor de umbral a evaluar. Para esta detección no se tienen en cuenta las comprobaciones sucesivas del ángulo.
Resultado	Los datos concretos recogidos se incluen en las matrices de confusión descritas más adelante.
Conclusión	Se trata de una medida muy sensible, lo cual implica una gran cantidad de falsas detecciones. Como este número de falsas detecciones se reduce considerablemente al añadir la comprobación del ángulo, el umbral más apropiado es **0.8**, pues es el umbral más bajo (menos probabilidad de detectar falsas caídas) que detecta todas las caídas que han sucedido.

Tabla 6.6: Prueba de ajuste de umbrales para GSVM en caída

Las matrices de confusion nos permiten saber la precisión, exactitud y especifidad del algoritmo, recogiendo los datos de las pruebas en una matriz de la siguiente forma:

Verdaderos positivos	Falsos positivos
Falsos negativos	Verdaderos negativos

Tabla 6.7: Estructura de una matriz de especifidad

Para los diferentes umbrales obtenemos los siguientes datos, los cuales permiten calcular las precisiones y exactitudes mencionadas previamente:

10	6
0	4

Tabla 6.8: Matriz de especifidad para el umbral 1

10	4
0	6

Tabla 6.9: Matriz de especifidad para el umbral 0.9

La misma prueba habría de repetirse en el caso de *SVM* para el impacto. Debido a la falta de tiempo, esta prueba no se puede realizar, por lo que se toma el umbral de **109** para el impacto. Se toma el umbral menor debido a que la prueba realizada para la obtención del mismo se trata de dejar caer tan solo el reloj, por lo que esta caída se aproxima más a la caída libre de lo que haría cuando una persona

10	4
0	6

Tabla 6.10: Matriz de especifidad para el umbral 0.8

9	2
1	8

Tabla 6.11: Matriz de especifidad para el umbral 0.7

llevase el reloj puesto. Sabiendo esto, el impacto que se da contra el suelo es mayor al impacto que se da cuando una persona lleva el reloj puesto, por lo que cogemos el umbral inferior para contrarrestarlo.

6.4. Comparación de algoritmos de detección de caídas.

Dado que la implementación final de la aplicación cuenta con 3 modos de detección de caídas, se lleva a cabo una comparación de estos algoritmos, tanto realizando pruebas en un entorno controlado como haciendo uso de los relojes inteligentes en el día a día, de tal forma que se pueda observar cuál proporciona mejores resultados. Además, como el dispositivo que se utiliza para la realización del proyecto es un Samsung, se puede utilizar la API que proporciona la propia empresa para la detección de caídas. Descargando el proyecto de prueba que aportan [84], se añade en una misma aplicación todos los algoritmos desarrollados, de forma que todas las detecciones se ejecuten de manera simultánea. De esta forma, se puede comparar la efectividad de los mismos bajo un entorno controlado.

Nombre	Comparación en un entorno controlado.
Descripción	Con el reloj inteligente puesto se realizan 20 caídas para observar si se da detección de caída. Estas acciones se llevan a cabo mediante varias personas de tamaños y constituciones diferentes, para poder obtener resultados más precisos.
Resultado	Todos los algoritmos detectan el 100 % de las caídas, salvo la detección que hace uso de SVM como primer umbral y el Ángulo como el segundo, la cual solo detecta un 75 %.
Conclusión	Debido a que todos los algoritmos detectan la gran mayoría de caídas en un entorno controlado, la diferencia entre estos radicará en la detección de caídas y falsas caídas en un caso de uso real, por lo que es necesaria una segunda prueba para comparar las mismas.

Tabla 6.12: Prueba de comparación de algoritmos en un entorno controlado.

En el caso de la prueba en un entorno no controlado, no se pueden utilizar todos los algoritmos a la vez, pues es complicado mostrar en una pantalla tan pequeña como es la de un reloj inteligente qué algoritmos están funcionando apropiadamente y cuáles no. Por ello, se instalan los 4 algoritmos (3 realizados durante este proyecto y el correspondiente a Samsung) en 4 relojes inteligentes diferentes y se dan a 4 personas para que realicen vida normal. De esta forma se tendrá que observar cuándo se

detectan caídas falsas y apuntarlas de manera manual para obtener los resultados.

Nombre	Comparación en un entorno real.
Descripción	Se lleva el reloj puesto y se hace vida normal con él durante 36h. Se ha de tomar nota de aquellas veces que se detecta una falsa caída. Además, a lo largo de la prueba se realizarán caídas verdaderas, para comprobar si se detectan adecuadamente.
Resultado	-Samsung: la aplicación proporcionada por Samsung no es capaz de detectar ninguna caída ni ninguna caída falsa. Se sospecha que pueda ser debido a que en el momento en que la aplicación pierde la conexión con el móvil Samsung al que está vinculado deja de tener acceso a los servicios de Samsung Health, por lo que deja de detectar caídas. Esto lo hace muy poco fiable para casos de uso real, donde el usuario no necesariamente tiene que tener un móvil Samsung conectado al reloj en todo momento. -Algoritmo GSVM y ANGULO: este algoritmo es capaz de detectar todas las caídas producidas. Sin embargo, cuando se gira la muñeca para, por ejemplo, colocarla encima de una mesa, se detectan caídas constantes debido a que el dispositivo está girado con respecto a su posición por defecto. Esto lo hace un algoritmo muy poco fiable, pues no se puede determinar si detecta caídas debido a la propia caída o simplemente detecta cualquier rotación de muñeca. -Algoritmo SVM y ANGULO: este algoritmo consigue no detectar caídas falsas. Sin embargo, tan solo es capaz de detectar entre un 20 % y un 40 % de las caídas verdaderas dadas. Como SVM se trata de una medida muy sensible, es necesario ajustar demasiado los umbrales del ANGULO, lo cual hace que, para evitar la detección de caídas falsas, se dejen de detectar muchas caídas verdaderas. -Algoritmo SVM para caída e impacto: este algoritmo es el que mejores resultados consigue. Se detectan entre un 90 % y un 100 % de las caídas verdaderas, con muy pocas detecciones de caídas falsas (entre un 20 % y un 30 %).
Conclusión	Teniendo en cuenta la desconexión de los servicios de Samsung Health, no se puede tomar esta alternativa como una válida, por lo que la mejor opción radica en un algoritmo de implementación propia. Entre los 3 considerados, los algoritmos basados en ángulo, a pesar de obtener resultados mucho mejores en un entorno controlado, a la hora de un caso de uso real se comportan bastante peor que el algoritmo basado en el impacto, por lo que será este último el que se tenga en cuenta para la realización del proyecto.

Tabla 6.13: Prueba de comparación de algoritmos en un entorno no controlado.

6.5. Comparación de nivel de batería.

Dado que el objetivo de poder activar/desactivar monitorizaciones era el de preservar la batería del dispositivo, se lleva a cabo una comparación entre la duración de la batería cuando se tienen todas las monitorizaciones activas y la duración cuando las monitorizaciones opcionales (retirada y caída) se desactivan.

Nombre	Comparación de nivel de batería dada activación/desactivación de monitorizaciones.
Descripción	Dado el modo de ejecución normal (no el de ahorro de batería), cada hora se toman medidas del nivel de batería del dispositivo desde el máximo de carga (100 %) hasta que se apague (0 %). Se realiza la prueba dos veces, con todas las monitorizaciones activas permanentemente y con las monitorizaciones de caída y retirada desactivadas.
Resultado	-Monitorizaciones activas: duración de la batería = entre 26 y 27 horas. -Monitorizaciones inactivas: duración de la batería = entre 29 y 30 horas.
Conclusión	Si bien se da una diferencia en los niveles de batería, al tratarse de apenas unas 3 horas esta no es verdaderamente significativa. Dado un caso de uso real, al no alcanzarse los dos días de duración en ninguno de los casos, probablemente lo más eficiente sea cargar el dispositivo mientras se duerme, por lo que esta diferencia de 3 horas no supone un cambio significativo.

Tabla 6.14: Prueba de comparación de nivel de batería de Samsung Galaxy Watch 4.

Más en detalle, se puede observar cómo, al principio de la vida del dispositivo, ambas monitorizaciones suponen una reducción bastante similar de la batería. La diferencia se comienza a apreciar tras el umbral aproximado del 50 % de la batería.

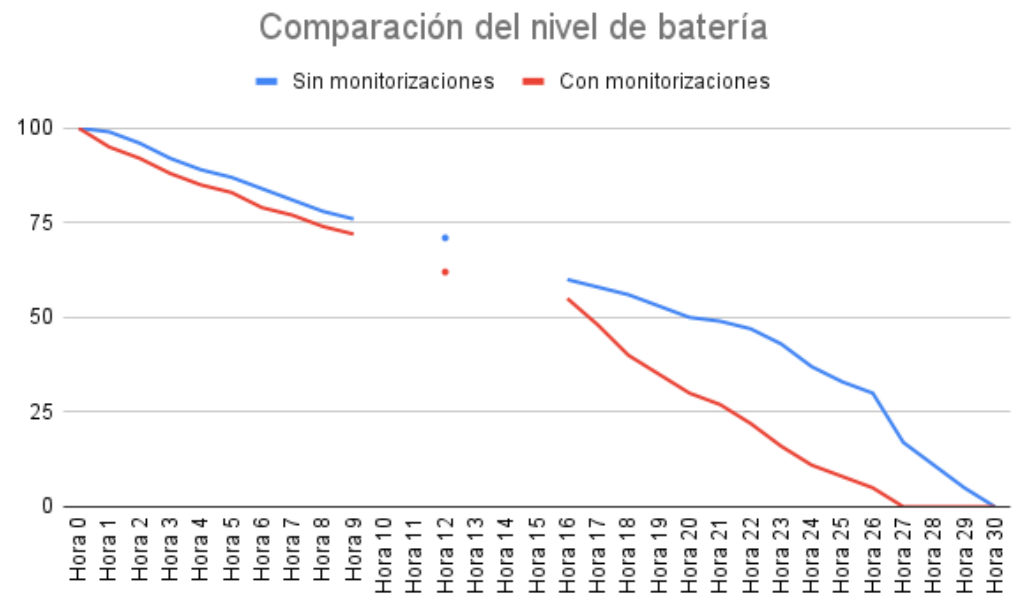

Figura 6.1: Comparación de niveles de batería

Estas medidas se decidieron tomar de manera manual, para prevenir el hecho de que la automatización de la recogida de estos datos pudiera afectar a los niveles de batería. Por ello se presenta una discontinuidad entre las horas 9 y 16, pues se corresponden a un período en el que se estaba durmiendo.

Capítulo 7

Conclusiones

En este último capítulo se resume y evalúa el trabajo realizado y se comenta brevemente el aprendizaje percibido a lo largo de este proyecto.

Este trabajo forma parte de una serie de iteraciones que se han de llevar a cabo para conseguir una app de monitorización completa. No se trata del primer paso realizado en este proyecto global, ni tampoco del último, pues aún hay cosas que mejorar. Esto aporta al trabajo una capa de realismo, pues no se trata de un proyecto académico que nadie más vaya a tocar (como podría ser el caso de otros TFGs), sino una aplicación en la que hay que esforzarse por mantener una legibilidad y un mantenimiento adecuados, para que otras personas puedan trabajar sobre ella. Así, por fin, se le da importancia a esa parte que nos repiten constantemente a lo largo de la carrera de que el programar es un esfuerzo colectivo tuyo y de gente a la que no tienes por qué conocer.

En cuanto a este trabajo colectivo, partir de un proyecto ya creado puede parecer una ventaja, pues tienes una estructura en la que basarte y funcionalidades que poder reutilizar. Sin embargo, esto a veces no son todo beneficios. Como ha sucedido en este proyecto, puede que la estructura planteada por personas anteriores no sea compatible con los cambios que se quieran introducir, haciendo que se tenga que dedicar mucho tiempo a una reestructura que permita estos cambios. Esto me ha hecho ser más consciente de asegurarme de que mi código sea fácilmente generalizable, manteniendo baja su complejidad y preparando estructuras de herencia que se puedan ampliar en un futuro.

Acerca del trabajo realizado en Wear OS, dado que el proyecto se ha realizado con Java no he ampliado demasiado los conocimientos en el área de la programación. Sí he aprendido a trabajar con sensores, más concreto con el acelerómetro, lo cual nunca se había hecho y me ha resultado un trabajo muy interesante, pues abre la puerta a poder desarrollar aplicaciones mucho más interesantes, no solo para smartwatches sino para dispositivos móviles en general. Además, se ha trabajado en un entorno que, según lo observado, no está del todo estandarizado. Hay un sinfín de smartwatches diferentes, lo cual hace que tengas que tener en cuenta que hay funcionalidades que no van a funcionar correctamente en todos los dispositivos (como el reproductor de sonido o la escritura de ficheros) y te fuerza a plantear alternativas que no habrías pensado en un primer lugar.

Con respecto al trabajo sobre algoritmos he aprendido a apreciar todas aquellas calibraciones que

se tienen en cuenta en las aplicaciones del día a día. A pesar de utilizar una gran cantidad de datos y pruebas manuales para ajustar los umbrales de los algoritmos, una vez estos se ponían a prueba en situaciones de uso real, los resultados eran completamente dispares a los obtenidos en un entorno controlado. Por ello, se han tenido que realizar múltiples cambios, no solo en los umbrales del algoritmo, sino en el método utilizado para la detección de caídas, con los umbrales que eso conllevara. Esto ha puesto en perspectiva lo realmente importante que es hacer una gran cantidad de pruebas de tus aplicaciones y el hecho de que estas pruebas sean de calidad.

Por último, y a pesar de las restricciones temporales al tratarse de un proyecto de género académico, puedo decir que se ha llegado a cumplir todos los objetivos iniciales. Algunos de estos han sufrido modificaciones desde su planteamiento inicial, pero se ha llegado a conseguir el objetivo requerido. Además de estos, también se han añadido objetivos secundarios que se consideraron interesantes o necesarios a lo largo del proyecto (como es el caso de la recogida de datos y diferentes modos de ejecución), los cuales suponen mejoras significativas en la aplicación. Dado esto, me atrevo a decir que el proyecto final ha sido un éxito.

7.1. Trabajo futuro

Que se hayan cumplido los objetivos principales no quiere decir que el proyecto general de la app de monitorización esté finalizado (ni mucho menos). En esta sección se plantean futuros objetivos a conseguir:

- **Descarga de configuración desde el servidor.** En la versión actual del proyecto se incluye la descarga de las configuraciones relacionadas con la selección de monitoreo de las caídas y la retirada del dispositivo. En una versión completa de la app se han de descargar todos los valores que se almacenan en la clase *'Configuracion'*, pues todos ellos pueden varias entre dispositivos.

- **Mejora de algoritmos propuestos.** Los algoritmos de detección de caídas propuestos en este proyecto se han sometido a bastantes pruebas para calibrar sus umbrales. Aun así, están lejos de tratarse de algoritmos comparables a, por ejemplo, la detección de caídas en dispositivos de Apple. Es interesante realizar un mayor estudio sobre estos algoritmos, mejorando sus umbrales con una mayor batería de pruebas o incluso sustituyéndolos por un algoritmo de aprendizaje automático lo suficientemente eficiente como para ejecutarse en un reloj inteligente.

- **Recogida de datos de algoritmos.** Actualmente, la recogida de datos en modo desarrollador se realiza imprimiendo datos por consola y guardándolos en ficheros de manera local. Resulta interesante enviar estos datos al servidor, de forma que el almacenamiento del dispositivo se mantenga libre y estos se puedan consultar con mayor facilidad. De esta forma se facilitaría la forma de probar diferentes algoritmos.

- **Implementación de pruebas.** A lo largo del proyecto se han realizado pruebas haciendo uso de la aplicación como tal. Sería interesante realizar una implementación de pruebas propia, extrayendo

los algoritmos de detección de caídas a un método o una clase que permita utilizarlos con una gran batería de datos, recogidos a lo largo del tiempo u obtenidos mediante una base de datos.

- **Detección de la colocación del dispositivo.** Si bien los métodos correspondientes a la colocación del dispositivo existen en el proyecto, esta comprobación no se realiza nunca, sino que se asume que un tiempo después de la retirada del mismo este se vuelve a colocar. Es importante tener un método de detección de colocación real, ya sea mediante algoritmo de detección o mediante pulsación de botón.
- **Limpieza del código.** A pesar de haberse realizado una reestructura del proyecto y haberse utilizado patrones de diseño, los escaneos realizados mediante la herramienta SonarLint revelan que aún hay mucho código que se puede limpiar y simplificar para conseguir una mayor legibilidad y hacer el mantenimiento más fácil. A pesar de que algunos de los cambios propuestos ya se han realizado, debido a la falta de tiempo, hay muchos otros que no se han podido realizar. Es interesante revisar el código ya escrito cuanto antes, para facilitar su modificación y el añadido de nuevas características.
- **Mantenimiento del código.** El código base de Android es actualizado constantemente, lo cual da lugar a métodos que quedan obsoletos y partes de nuestro código que pueden dejar de funcionar. Es importante que el código de la aplicación se mantenga actualizado para que la misma funcione correctamente.

Apéndices

Apéndice A

Acrónimos

- **API**: Application Programming Interface.
- **BD**: Base de Datos.
- **CU** / **CCUU**: Caso de Uso / Casos de Uso.
- **DAO**: Data Access Object.
- **GSVM**: Gravity-wighted Sum Vector Magnitude / Vector Magnitud de la Suma de aceleraciones ponderado por la Gravedad.
- **IDE**: Integrated Development Environment.
- **JSON**: Java Script Object Notation.
- **OS** / **SO**: Operating System / Sistema Operativo.
- **RAM**: Random Access Memory.
- **SMS**: Short Message Service.
- **SQL**: Structures Query Language.
- **SVM**: Sum Vector Magnitude / Vector Magnitud de la Suma de aceleraciones.
- **TFG**: Trabajo Fin de Grado.
- **UVa**: Universidad de Valladolid.

Apéndice B

Manual de despliegue

Se mantienen los requisitos del proyecto tomado como punto de partida, por lo que la aplicación debe ser instalada en un dispositivo smartwatch con WearOS como sistema operativo, con una versión mínima de Android 6.0.

B.1. Instalación del IDE y obtención del proyecto.

Para poder realizar la instalación de la app es necesario hacer uso del IDE de Android Studio. No necesita configuración especial durante la descarga ni instalación, solo seguir los pasos del Wizard. Se puede obtener en el siguiente link:

```
https://developer.android.com/studio
```

Una vez se tiene el IDE, hay que descargar el proyecto completo. Se recomienda acceder a la versión web de Gitlab y descargar el proyecto en formato zip. Una vez descargado, se ha de descomprimir. Una vez descomprimido, se puede abrir con Android Studio. También se puede clonar, si así se desea.

B.2. Configuraciones antes de la instalación.

Antes de instalar la aplicación se ha de abrir el fichero *'Constantes.java'* y comprobar que la aplicación se va a instalar con los valores que se requieren.

Este fichero se ha de modificar para realizar las siguientes selecciones, cuyos valores se encuentran explicados en el propio fichero:

1. $(MODO\_MONITORIZACION\_UBICACION)$ La app va a tener dos modos de monitorizar la ubicación: modo preciso y modo ahorro de batería.

2. $(MODO\_MONITORIZACION\_CAIDA)$ La app va a tener tres modos de monitorizar la caída: mediante GSVM y Angulo; mediante SVM y Angulo; mediante SVM para caída e impacto.

3. $(MODO\_DEBUG)$ La app va a tener el modo de ejecución para desarrolladores.

Además, en este fichero también se pueden modificar los valores de los umbrales utilizados en los diferentes algoritmos de la aplicación (tanto los diferentes métodos de detección de caída, como de retirada).

B.3. Primer modo de instalación. Depuración ADB.

Para poder instalar la aplicación mediante depuración ADB es necesario activar esta opción en los ajustes del dispositivo. Para ello, hay que dirigirse a **AJUSTES >ACERCA DEL RELOJ >VERSIÓN DE SOFTWARE** y pulsar en esta opción hasta que se muestre por pantalla el texto .ªctivadas Opciones de Desarrollador". Tras ello, hay que dirigirse a **AJUSTES >OPCIONES DE DESARROLLADOR >DEPURACIÓN ADB** y activar esta opción.

En caso de que el dispositivo se pueda conectar mediante USB al ordenador, tan solo hace falta conectarlo. En dicho momento se solicitarán permisos para conectarse con el ordenador, aceptarlos.

En caso de que el dispositivo no se pueda conectar mediante USB, hay que dirigirse a **AJUSTES >OPCIONES DE DESARROLLADOR >DEPURACIÓN ADB >DEPURAR VÍA WIFI** (activar esta opción) **>EMPAREJAR NUEVO DISPOSITIVO** lo cual mostrará un código por pantalla. Una vez se vea el código, dirigirse a **Android Studio** y seleccionar **DEVICE MANAGER >PHYSICAL >PAIR USING WIFI >PAIR USING PAIRING CODE** donde habrá que introducir el código que muestra el reloj. De nuevo, al conectarse pedirá los permisos pertinentes.

Tras cualquiera de las dos opciones, en la parte superior de Android Studio deberá aparecer el nombre del dispositivo conectado, tal como se ve en la figura B.1. Al lado se encuentra el botón 'run' (triángulo verde), hacer click y la aplicación se instalará y ejecutará en el dispositivo.

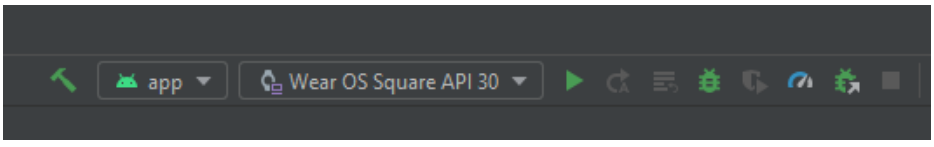

Figura B.1: Interfaz de ejecución de Android Studio

B.4. Segundo modo de instalación. Uso de APK.

Para generar el archi de instalación APK de la aplicación, en Android Studio, hay que dirigirse a **BUILD >BUILD BUNDLE(S)/APK(S) >BUILD APK(S)**. Tras esto se generará un archivo de instalación (se puede localizar mediante el botón 'locate' de Android Studio) que se tiene que transferir al dispositivo mediante el móvil al que esté conectado. Para ello, se tiene que utilizar una aplicación de transferencia de archivos de terceros, pues las proporcionadas por defecto por Android no suelen dejar transferir archivos ejecutables. Una vez transferido solo hace falta ejecutarlo y la aplicación se instalará.

Apéndice C

Manual de uso

A lo largo de este apéndice se detallará cómo utilizar la aplicación una vez instalada en el dispositivo requerido. Para su correcto uso el servidor con el que conecta ha de estar funcionando.

C.1. Uso del Administrador

C.1.1. Lanzamiento de la aplicación

Es necesario que el lanzamiento de la aplicación lo realice un administrador, pues es necesario realizar ciertas configuraciones que el paciente no es capaz de realizar por si mismo.

Al iniciar la aplicación por primera vez lo primero que se muestra es una serie de pantallas similares a la que se puede ver en la Figura C.1 (las vistas pueden variar dependiendo del dispositivo), pidiendo acceso a las diferentes funcionalidades necesarias para la monitorización. En caso de que no se encuentre el botón necesario para aceptarlos, hacer scroll hacia abajo para verlo. Si no se aceptan estos permisos, la aplicación no se ejecutará correctamente y dará error.

Figura C.1: Pantalla de permisos.

Una vez se concedan los permisos, la aplicación comenzará su ejecución en segundo plano, tras lo

cual aparecerá una notificación persistente en el reloj, cuya apariencia varía dependiendo del dispositivo (en la Figura C.2 se muestra una posible apariencia). Esta notificación siempre aparecerá la primera (arriba del todo), de tal forma que resulta fácil de localizar en caso de emergencia.

Figura C.2: Notificación persistente en el dispositivo.

Al desplegar dicha notificación, se podrá ver un contenido similar al de la Figura C.3, mostrando el ID del dispositivo utilizado y el botón de Emergencia. Este ID se ha de resgistrar en el servidor para vincular el dispositivo con el usuario correspondiente y poder descargar las configuraciones establecidas para el mismo. Una vez descargadas se minimizará la notificación (la pantalla volverá a aparecer como en la Figura C.2) y la aplicación estará lista para utilizar.

Figura C.3: Captura de la pantalla que se muestra tras presionar la notificación.

C.1.2. Actualización de la configuración

Una vez se haya ejecutado la aplicación correctamente, es necesario realizar la descarga de la configuración desde el servidor. Para ello se han de seguir los siguientes pasos:

1. Localizar la notificación persistente, que tiene el aspecto de la Figura C.2. Esta siempre se encontrará por encima del resto de notificaciones.

2. Pulsar en la misma, de tal forma que se despliegue como se ve en la Figura C.3.

3. Pulsar sobre el botón ”EMERGENCIA”. En caso de no verse el botón debido a variaciones en la apariencia, realizar scroll hacia abajo hasta verlo.

4. Se podrá ver una pantalla similar a la de la figura C.4, en la cual se encuentran los botones ”ACEPTAR” y ”CANCELAR”.

5. Pulsar repetidas veces sobre el texto informativo. Cuando se llegue a las 5 veces, se iniciará el proceso de descarga y actualización de la aplicación y desaparecerá la pantalla actual. No es necesario realizar nada más.

C.2. Uso del Paciente

Una vez la aplicación ha sido iniciada por un administrador el paciente puede utilizar el reloj inteligente de manera normal, pues las funcionalidades de monitorización se ejecutan de manera constante en segundo plano. Hay dos casos en los cuáles el paciente ha de interactuar con el dispositivo: para notificar una emergencia de forma propia o para confirmar/cancelar una emergencia tras la detección de una caída leve.

C.2.1. Notificar emergencia propia

Se mantiene la funcionalidad que ya existía en la aplicación tomada como punto de partida [8]. Para notificar una emergencia se han de seguir los siguientes pasos:

1. Localizar la notificación persistente, que tiene el aspecto de la Figura C.2. Esta siempre se encontrará por encima del resto de notificaciones.

2. Pulsar en la misma, de tal forma que se despliegue como se ve en la Figura C.3.

3. Pulsar sobre el botón ”EMERGENCIA”. En caso de no verse el botón debido a variaciones en la apariencia, realizar scroll hacia abajo hasta verlo.

4. Se podrá ver una pantalla similar a la de la figura C.4, en la cual se encuentran los botones ”ACEPTAR” y ”CANCELAR”.

5.1 Si se desea aceptar la emergencia, pulsar sobre el botón ”ACEPTAR”. Esto iniciará una llamada al centro de control de manera automática. En caso de que no se pulse el botón antes de que finalice el contador que se muestra en el otro botón, se cancelará la emergencia (de la misma forma que si se hubiera pulsado el boton ”CANCELAR”).

5.2 Si se desea cancelar la emergencia, pulsar sobre el botón ”CANCELAR”. Al cancelarse la emergencia, no se realiza ninguna notificación ni llamada al centro de control. La pantalla actual desaparecerá y se volverá al estado que se tenía en la FiguraC.2.

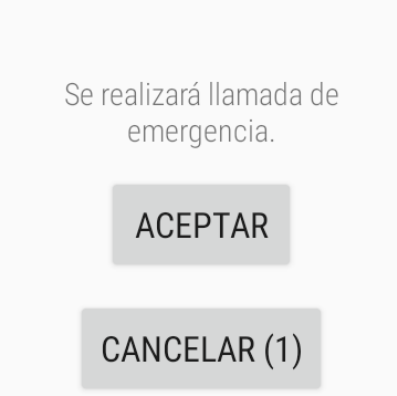

Figura C.4: Pantalla para aceptar/cancelar la alerta de emergencia.

C.2.2. Confirmar emergencia tras detección de caída

Cuando se detecta una caída, se inicia automáticamente la ejecución de confirmación de caída. Esto desencadena los siguientes pasos:

1. Se despliega automáticamente una pantalla similar a la de la figura C.5, en la cual se encuentran los botones ”SI” y ”NO”.

2.1 Si se desea confirmar la detección de caída, pulsar sobre el botón ”SI”. Esto enviará una notificación de caída leve al centro de control de manera automática. La pantalla actual desaparecerá y se volverá al estado que se tenía en la FiguraC.2. En caso de que se quiera iniciar un caso de Emergencia dada cierta gravedad de la caída, se ha de notificar de manera manual (explicado en la sección anterior).

2.2 Si se desea cancelar la detección de caída, pulsar sobre el botón ”NO”. Esto enviará una notificación de caída falsa al centro de control de manera automática, para su posterior análisis. La pantalla actual desaparecerá y se volverá al estado que se tenía en la FiguraC.2.

2.3 Si no se pulsa ninguno de los botones antes de que el temporizador finalice, se supondrá que se ha dado una caída grave. Esto enviará una notificación de caída grave al centro de control de manera automática. Se sume que se toman las medidas pertinentes desde el centro de control, por lo que no es necesaria mayor interacción con la aplicación. La pantalla actual desaparecerá y se volverá al estado que se tenía en la FiguraC.2.

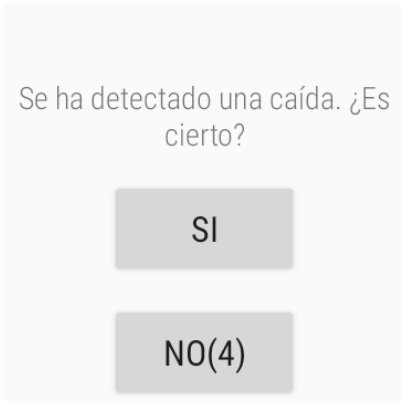

Figura C.5: Pantalla para aceptar/cancelar la alerta de caida.

Apéndice D

Contenido del TFG

El código de la aplicación desarrollado a lo largo del proyecto se encuentra en un repositorio de Gitlab de la Escuela de Ingeniería Informática de la UVa. Se puede acceder mediante el siguiente enlace:

`https://gitlab.inf.uva.es/tfg/tfg_maria-delser-gutierrez-2023-2024`

Además, se ha conservado una referencia a la versión original de la aplicación para poder realizar comparaciones entre el código ya existente y los cambios y añadidos realizados a lo largo del proyecto. Se puede consultar en el siguiente enlace:

`https://gitlab.inf.uva.es/tfg/punto-de-partida`

Bibliografía

[1] España - piramide de población. Visitado: 2024-02-21. [Online]. Available: https://datosmacro.expansion.com/demografia/estructura-poblacion/espana

[2] Cuántas personas mayores viven solas en casa. Visitado: 2024-02-21. [Online]. Available: https://www.madrimasd.org/cuantas-personas-mayores-viven-solas-en-casa#:~:text=Casi%20cinco%20millones%20de%20personas,tienen%2070%20o%20m%C3%A1s%20a%C3%B1os.

[3] D. A. Ganz and N. K. Latham, "Prevention of falls in community-dwelling older adults," *New England Journal of Medicine, 382(8)*, 20-02-2020.

[4] S. Sadigh, A. Reimers, R. Andersson, and L. Laflamme, "Falls and fall-related injuries among the elderly: A survey of residential-care facilities in a swedish municipality," *Journal of Community Health, Vol 29, Num 2*, 04-2004.

[5] Teleasistencia móvil de cruz roja. Visitado: 2023-10-19. [Online]. Available: https://www2.cruzroja.es/web/teleasistencia/teleasistencia-movil

[6] J. Wisniewski. How much do the top medical alert systems cost in 2023?: Researched and tested by experts. Visitado: 2023-10-19. [Online]. Available: https://www.ncoa.org/adviser/medical-alert-systems/medical-alert-systems-cost/

[7] Number of users of smartwatches worldwide from 2018 to 2027. Visitado: 2023-10-19. [Online]. Available: https://www.statista.com/forecasts/1314339/worldwide-users-of-smartwatches

[8] S. S. Sanz, *DESARROLLO DE APP PARA SMARTWATCH: MONITORIZACIÓN DE PERSONAS VULNERABLES*. Universidad de Valladolid, 2017.

[9] M. Alwan, P. Rajendran, S. Kell, D. Mack, S. Dalal, M. Wolfe, and R. Felder, "A smart and passive floor-vibration based fall detector for elderly," in *A Smart and Passive Floor-Vibration Based Fall Detector for Elderly*, vol. 1, 01 2006.

[10] A. Sixsmith and N. Johnson, "A smart sensor to detect the falls of the elderly," *Pervasive Computing, IEEE*, vol. 3, 05 2004.

[11] Technical solutions autralia. Visitado: 2023-11-02. [Online]. Available: https://tecsol.com.au/

[12] Fall detection using ti mmwave radar sensors. Visitado: 2024-02-29. [Online]. Available: https://www.youtube.com/watch?v=lsIo_HIk4GY&ab_channel=TexasInstruments

[13] Iwr6843aop evaluation module for single-chip 60ghz antenna-on-package (aop) mmwave sensor. Visitado: 2024-02-29. [Online]. Available: https://www.ti.com/tool/IWR6843AOPEVM#tech-docs

[14] B. Jansen and R. Deklerck, “Context aware inactivity recognition for visual fall detection,” in *Context aware inactivity recognition for visual fall detection*, 01 2007.

[15] D. Anderson, J. M. Keller, M. Skubic, X. Chen, and Z. He, “Recognizing falls from silhouettes,” *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2006.

[16] C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, “Monocular 3d head tracking to detect falls of elderly people,” *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 1, 02 2006.

[17] M. D. Sola. ¿cómo elegir un detector de caídas? Visitado: 2023-10-19. [Online]. Available: http://www.alzfae.org/fundacion/461/como-elegir-un-detector-de-caidas

[18] El detector de caídas "vigi’fall". Visitado: 2023-10-19. [Online]. Available: https://www.valida.es/blog/post/el-detector-de-caidas-vigifall/

[19] Parche triangular vigifall. Visitado: 2024-02-21. [Online]. Available: https://www.dependenciasocialmedia.com/2013/06/vigifall-un-diminuto-parche-anticaidas-para-personas-mayores/

[20] L. Schlenker and T. Shuman. Best medical alert systems with fall detection for seniors. Visitado: 2023-10-19. [Online]. Available: https://www.seniorliving.org/medical-alert-systems/best/fall-detection/

[21] S. Olson and S. Lindberg. 8 best fall detection devices of 2023: Our experts tested and reviewed. Visitado: 2023-10-19. [Online]. Available: https://www.ncoa.org/adviser/medical-alert-systems/best-medical-alert-systems-fall-detection/

[22] Mini guardian medical alert system. Visitado: 2023-10-19. [Online]. Available: https://www.medicalguardian.com/medical-alert-systems/gps-button-fall-detection-technology

[23] F. Laricchia. Smartwatch market share worldwide from 2020 to 2022, by vendor. Visitado: 2023-10-19. [Online]. Available: https://www.statista.com/statistics/1296818/smartwatch-market-share/

[24] Use fall detection with apple watch. Visitado: 2023-10-19. [Online]. Available: https://support.apple.com/en-us/HT208944

[25] Smartwatch apple series 5. Visitado: 2024-02-21. [Online]. Available: https://www.apple.com/es/newsroom/2019/09/apple-unveils-apple-watch-series-5/

[26] Use the detect fall feature on your samsung smart watch. Visitado: 2023-10-19. [Online]. Available: https://www.samsung.com/us/support/answer/ANS00087244/

[27] Diferentes modelos de smartwatches samsung. Visitado: 2024-02-21. [Online]. Available: https://news.samsung.com/latin/actualizan-funciones-de-personalizacion-y-salud-para-los-galaxy-watch-galaxy-watch-active-galaxy-watch-active2-y-galaxy-watch3

[28] ufallalert app. Visitado: 2023-10-19. [Online]. Available: https://unfoldlabs.com/ufallalert/

[29] Free fall detection using 3-axis accelerometer. Visitado: 2023-11-27. [Online]. Available: https://www.hackster.io/RVLAD/free-fall-detection-using-3-axis-accelerometer-06383e

[30] O. Hammarstedt, *Fall detection bracelet with an accelerometer and cellular connectivity*. Uppsala Universitet, 06 2019.

[31] A. Bourke, J. O'Brien, and G. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait & Posture*, vol. 26, no. 2, pp. 194–199, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0966636206001895

[32] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, and T. Jämsä, "Comparison of low-complexity fall detection algorithms for body attached accelerometers," *Gait & Posture*, vol. 28, no. 2, pp. 285–291, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096663620800026X

[33] A. Bourke, P. van de Ven, M. Gamble, R. O'Connor, K. Murphy, E. Bogan, E. McQuade, P. Finucane, G. ÓLaighin, and J. Nelson, "Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities," *Journal of Biomechanics*, vol. 43, no. 15, pp. 3051–3057, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021929010003866

[34] S. S. Jeong, N. H. Kim, and Y. S. Yu, "Fall detection system based on simple threshold method and long short-term memory: Comparison with hidden markov model and extraction of optimal parameters," *Applied Sciences*, vol. 12, no. 21, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/21/11031

[35] Modelo oculto de markov. Visitado: 2023-11-27. [Online]. Available: https://es.wikipedia.org/wiki/Modelo_oculto_de_M%C3%A1rkov

[36] Proceso de markov. Visitado: 2023-11-27. [Online]. Available: https://es.wikipedia.org/wiki/Proceso_de_M%C3%A1rkov

[37] D. Lim, C. Park, N. H. Kim, S.-H. Kim, and Y. S. Yu, "Fall-detection algorithm using 3-axis acceleration: Combination with simple threshold and hidden markov model," *Journal of Applied Mathematics*, 2014. [Online]. Available: https://doi.org/10.1155/2014/896030

[38] S. Yu, H. Chen, and R. Brown, "Hidden markov model-based fall detection with motion sensor orientation calibration: A case for real-life home monitoring," *IEEE Journal of Biomedical and Health Informatics*, vol. PP, pp. 1–1, 12 2017.

[39] O. Aziz, M. Musngi, E. J. Park, G. Mori, and S. N. Robinovitch, "A comparison of accuracy of fall detection algorithms (threshold-based vs. machine learning) using waist-mounted tri-axial accelerometer signals from a comprehensive set of falls and non-fall trials," *Medical & Biological Engineering & Computing*, 2017. [Online]. Available: https://doi.org/10.1007/s11517-016-1504-y

[40] L. Zheng, J. Zhao, F. Dong, Z. Huang, and D. Zhong, "Fall detection algorithm based on inertial sensor and hierarchical decision." *Sensors (Basel, Switzerland)*, vol. 23,1 107, 2022. [Online]. Available: https://doi.org/10.3390/s23010107

[41] N. Nahar, M. S. Hossain, and K. Andersson, “A machine learning based fall detection for elderly people with neurodegenerative disorders,” in *Brain Informatics*, M. Mahmud, S. Vassanelli, M. S. Kaiser, and N. Zhong, Eds. Cham: Springer International Publishing, 2020, pp. 194–203.

[42] Q. Li, J. Stankovic, M. Hanson, A. Barth, J. Lach, and G. Zhou, “Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information,” in *Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information*, 06 2009, pp. 138–143.

[43] B. Andò, S. Baglio, C. O. Lombardo, and V. Marletta, “A multisensor data-fusion approach for adl and fall classification,” *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 9, pp. 1960–1967, 2016.

[44] E. Casilari, M. Álvarez Marco, and F. García-Lagos, “A study of the use of gyroscope measurements in wearable fall detection systems,” *Symmetry*, vol. 12, no. 4, 2020. [Online]. Available: https://www.mdpi.com/2073-8994/12/4/649

[45] Logo de wearos. Visitado: 2024-02-21. [Online]. Available: https://logowik.com/google-android-wear-os-vector-logo-4836.html

[46] D. Bohn. Motorola, lg announce upcoming android wear smartwatches. Visitado: 2023-10-19. [Online]. Available: https://www.theverge.com/2014/3/18/5522340/motorola-lg-announce-upcoming-android-wear-smartwatches

[47] C. Welch. Google just changed the name of android wear to wear os. Visitado: 2023-10-19. [Online]. Available: https://www.theverge.com/2018/3/15/17124448/google-wear-os-announced-android-wear-rebranding-smartwatch

[48] Cómo crear y ejecutar una app en wear os. Visitado: 2023-10-19. [Online]. Available: https://developer.android.com/training/wearables/get-started/creating?hl=es-419

[49] A. Turner. Android vs. apple market share: Leading mobile operating systems. Visitado: 2023-10-19. [Online]. Available: https://www.bankmycell.com/blog/android-vs-apple-market-share/

[50] Logo de android. Visitado: 2024-02-21. [Online]. Available: https://logowik.com/android-logo-vector-4-40311.html

[51] B. Elgin. Google buys android for its mobile arsenal. Visitado: 2023-10-19. [Online]. Available: https://webcitation.org/5wk7sIvVb

[52] Industry leaders announce open platform for mobile devices. Visitado: 2023-10-03. [Online]. Available: http://www.openhandsetalliance.com/press_110507.html

[53] Arquitectura del sistema operativo android. Visitado: 2023-10-19. [Online]. Available: https://es.wikipedia.org/wiki/Android#Arquitectura

[54] Acquisition of sun microsystems by oracle corporation. Visitado: 2023-10-19. [Online]. Available: https://en.wikipedia.org/wiki/Acquisition_of_Sun_Microsystems_by_Oracle_Corporation

[55] Java development kit 20. Visitado: 2023-10-19. [Online]. Available: https://openjdk.org/projects/jdk/20/

[56] Independencia de la plataforma de java. Visitado: 2023-10-19. [Online]. Available: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#Independencia_de_la_plataforma

[57] Filosofia de java. Visitado: 2023-10-03. [Online]. Available: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#Filosof%C3%ADa

[58] Cómo guardar contenido en una base de datos local con room. Visitado: 2023-10-03. [Online]. Available: https://developer.android.com/training/data-storage/room?hl=es-419

[59] Pasos de la metodología iterativa e incremental. Visitado: 2024-02-21. [Online]. Available: https://medium.com/sue%C3%B1os-graficos/dise%C3%B1o-iterativo-la-metodolog%C3%ADa-que-perfeccionar%C3%A1-tus-proyectos-21034b0d277e

[60] Ide android studio. Visitado: 2023-10-19. [Online]. Available: https://developer.android.com/studio

[61] Gitlab uva. Visitado: 2023-10-19. [Online]. Available: https://gitlab.inf.uva.es/

[62] Astah professional. Visitado: 2023-10-19. [Online]. Available: https://astah.net/

[63] Overleaf. Visitado: 2023-10-19. [Online]. Available: https://es.overleaf.com

[64] Microsoft teams. Visitado: 2023-10-19. [Online]. Available: https://www.microsoft.com/es-es/microsoft-teams/

[65] Google chrome. Visitado: 2023-10-19. [Online]. Available: https://www.google.com/intl/es_es/chrome/browser-tools/

[66] Sueldos para el puesto de desarrollador android junior en españa. Visitado: 2023-10-19. [Online]. Available: https://www.glassdoor.es/Sueldos/desarrollador-android-junior-sueldo-SRCH_KO0,28.htm

[67] E. Nesbo. How many watts does a laptop use and how much does it cost to run? Visitado: 2023-10-19. [Online]. Available: https://www.makeuseof.com/how-many-watts-laptop-use-how-much-cost-to-run/

[68] El precio de la luz en septiembre. Visitado: 2023-10-19. [Online]. Available: https://www.ocu.org/vivienda-y-energia/gas-luz/informe/precio-luz

[69] Sonarlint plugin. Visitado: 2024-01-25. [Online]. Available: https://plugins.jetbrains.com/plugin/7973-sonarlint

[70] Arquitectura por capas de las aplicaciones android. Visitado: 2024-01-29. [Online]. Available: https://developer.android.com/topic/architecture?hl=es-419

[71] Patrón de diseño observador. Visitado: 2024-01-29. [Online]. Available: https://refactoring.guru/design-patterns/observer

[72] Patrón de diseño singleton. Visitado: 2024-01-29. [Online]. Available: https://refactoring.guru/design-patterns/singleton

[73] Patrón de diseño dao. Visitado: 2024-01-29. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/dao

[74] Patrón de diseño dto. Visitado: 2024-01-29. [Online]. Available: https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/dto

[75] Patrón de diseño factory. Visitado: 2024-01-29. [Online]. Available: https://refactoring.guru/design-patterns/factory-method

[76] Archivo androidmanifest. Visitado: 2023-12-19. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

[77] Clase sensormanager. Visitado: 2023-12-19. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorManager

[78] Clase sensoreventlistener. Visitado: 2023-12-19. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorEventListener

[79] Fused location provider de google. Visitado: 2023-12-19. [Online]. Available: https://developers.google.com/location-context/fused-location-provider?hl=es-419

[80] Location request de google. Visitado: 2023-12-19. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest

[81] Location request de google. Visitado: 2023-12-19. [Online]. Available: https://github.com/google/wear-sensors

[82] Fall detection accelerometer dataset. Visitado: 2023-12-19. [Online]. Available: https://www.kaggle.com/datasets/harnoor343/fall-detection-accelerometer-data/data

[83] Estatura media de hombres y mujeres en todo el mundo. Visitado: 2023-12-19. [Online]. Available: https://www.datosmundial.com/estatura-promedio.php

[84] Fall detection accelerometer dataset. Visitado: 2023-12-19. [Online]. Available: https://developer.samsung.com/health/blog/en/2022/12/14/events-monitoring#Resources