Facultad de Ciencias

SOLUCIÓN AL PROBLEMA "HOMBRE-SOLO" BASADO EN DISPOSITIVO SMARTWATCH

(SOLUTION TO THE PROBLEM OF ISOLATION BASED ON SMARTWATCH DEVICE)

Trabajo de fin de grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Fernando González Casillas
Director: Carlos Blanco Bueno
Co-Director: Miguel Sierra Sánchez

Junio, 2019

Agradecimientos

A mi familia, por ser la base sobre la que me he construido, por haberme enseñado a ser persona antes que cualquier otra cosa, por haberme proporcionado todo aquello que me ha permitido ser quien soy hoy y por apoyarme en esos momentos oscuros que han habitado mi mente, intentando entenderme y ayudándome a continuar. Gracias por todo.

A mis amigos, por ser tan geniales, por ser una fuente de inspiración constante para mí, por permitirme aprender sobre muchos aspectos de la vida, por apoyarme de manera incondicional, por soportarme y por completar mis referencias a los *Simpsons*.

Agradezco también a Carlos Blanco el haber accedido a ser mi tutor en este trabajo de final de grado aún habiéndoselo pedido el primer día de clase sin haber coincidido antes durante toda la carrera.

También me gustaría agradecer a Miguel Sierra y a todos mis compañeros de SOINCON el haberme brindado esta oportunidad que me ha permitido adquirir conocimientos que no se consiguen en un aula.

Y por último, pero no menos importante, este trabajo me lo dedico a mí. Por todas las adversidades superadas, por seguir siempre hacia adelante, pase lo que pase. Porque cuando en mi vida se aparezca una situación difícil, cogeré aire, pensaré en todo aquello que he dejado atrás y lo que queda por delante, me miraré al espejo y soltaré un sincero *"¿Qué no, Lisa? ¿¡QUÉ NO!?"*.

Resumen

Los accidentes laborales, a día de hoy, siguen siendo un gran problema que sigue sin resolverse a pesar de la evolución tecnológica que presenciamos. Para ello, una parte de la denominada *Industria 4.0*, se dedicará a plantear soluciones a mencionado problema, sustituyendo los métodos rudimentarios de prevención por una tecnología desarrollada con este fin específico.

Uno de los problemas causantes de muchos de los accidentes laborales, sobre todo mortales, son los escenarios en los cuales el trabajador se encuentra aislado de contacto alguno debido a la labor que realiza, ya que en el caso de que sufra cualquier tipo de accidente en el que pierda la consciencia, no será capaz de notificar de dicho accidente, reduciendo las posibilidades de supervivencia de dicho operario.

Por lo que en este proyecto se expone el desarrollo de una aplicación *Android* dedicada al personal que desarrolle su trabajo en ciertas zonas aisladas de la fábrica o planta pueda notificar de un problema físico o accidente tal como una caída, bajada de tensión... Para esto, se plantea una solución basada en un dispositivo de tipo *smartwatch* que cumpla con este objetivo.

Palabras Clave: Industria 4.0, smartwatch, accidentes laborales, prevención, Android, Wear OS.

Preface

Accidents at work, to this day, are still a major problem that remains unresolved despite the technological evolution we are witnessing. For this purpose, a part of the so-called *Industry 4.0* will be dedicated to propose solutions to this problem, replacing rudimentary methods of prevention with a technology developed for this specific purpose.

One of the problems causing many accidents at work, especially mortal ones, are the scenarios in which the worker is isolated from any contact due to the work performed, since in the event that he suffers any type of accident in which he loses consciousness, he will not be able to report the accident, reducing the chances of survival of the operator.

So in this project is exposed the development of an *Android* application dedicated to staff who develop their work in certain isolated areas of the factory or plant can report a physical problem or accident such as a fall, postural hypotension ... For this, we propose a solution based on a smartwatch device that meets this objective.

Keywords: Industry 4.0, smartwatch, work-related accidents, prevention, Android, Wear OS.

Índice

Índice de figuras

Índice de cuadros

1 Introducción

En este primer capítulo se expone, de una manera breve, el contexto por el que se propone este proyecto, demostrando aquellas razones que motivan su realización así como los objetivos que se deben cumplir tras su resolución.

1.1. Introducción

Los accidentes laborales siguen siendo, a día de hoy, un gran problema que se plantea a resolver ya que, aunque mediante los avances tecnológicos sufridos en estas últimas décadas se ha reducido de manera considerable este desagradable número, mantiene una presencia importante en la estadística dentro del panorama nacional.

Estos accidentes se reparten entre los escenarios de trabajo en todos los sectores, encabezando la lista de aquellos que son mortales los sectores de la industria manufacturera, el ámbito de la construcción, transporte y almacenamiento y la agricultura, ganadería y pesca.

En lo relativo al desarrollo tecnológico, en el ámbito laboral, se ha centrado toda la atención en mejorar la producción o desarrollo, no siendo así dichos avances significativos en lo que a la seguridad del trabajador respecta, ya que se mantienen medios rudimentarios como *walkie-talkies*, botones del pánico… Un ejemplo es el problema que SOINCON quiere resolver con este proyecto, el problema "Hombre-Solo", llamado así porque hace referencia a las acciones que debe realizar un trabajador que se encuentra en un entorno aislado (como por ejemplo un conductor de tren o un operario de almacén) en el cual, cada cierto tiempo y por su seguridad, mediante medios tan rudimentarios como los mencionados previamente, tendrá que certificar su consciencia. Por lo que este es una cuestión importante a la que se propone resolver parte de la llamada *Industria 4.0*. SOINCON es una compañía de consultoría en conectividad industrial y transformación digital que nace con la vocación de aportar valor en el mundo de esta industria a través de la consultoría en evaluación tecnológica y propuestas de valor en el ámbito del internet industrial (Industrial IoT).

Dicha industria implica la promesa de una nueva revolución que combina técnicas avanzadas de producción y operaciones con tecnologías inteligentes que se integrarán en las organizaciones, las personas y los activos [5].

Esta revolución está marcada por la aparición de nuevas tecnologías como la robótica, la analítica, la inteligencia artificial, las tecnologías cognitivas, la nanotecnología y el denominado *Internet of Things* (IoT), entre otros.

Las empresas tradicionales desarrollaban su actividad mediante la manipulación de datos y comunicaciones lineales, el cambio que supone esta nueva revolución industrial (proporcionando acceso en tiempo real a los datos y la inteligencia de negocio) transforma la forma en que llevan a cabo sus negocios. La integración digital de la información desde diferentes fuentes y localizaciones permite llevar a cabo negocios en un ciclo continuo. A lo largo de este ciclo, el acceso en tiempo real a la información está impulsado por el continuo y cíclico flujo de información y acciones entre los mundos

físicos y digitales. Este flujo tiene lugar a través de una serie de pasos iterativos conocido como PDP (por sus siglas en inglés physical-to-digital-to-physical):

- Del mundo físico al digital: Se captura la información del mundo físico y se crea un registro digital de la misma.

- De digital a digital: En este paso, la información se comparte y se interpreta utilizando analítica avanzada, análisis de escenarios e inteligencia artificial para descubrir información relevante.

- Del mundo digital al físico: Se aplican algoritmos para traducir las decisiones del mundo digital a datos efectivos, estimulando acciones y cambios en el mundo físico.

En este escenario a solucionar, debemos tener en cuenta las ventajas que nos otorgan las últimas tecnologías como en nuestro caso lo serán los *smartwaches*. Básicamente un *smartwatch* es un dispositivo móvil con pantalla táctil diseñado para ser llevado en la muñeca; como su nombre indica: un "reloj inteligente" que al igual que los ya bien conocidos *smartphones*.

Puede ser que, como toda tecnología que pretende hacerse hueco en un ambiente cotidiano, todavía no esté demasiado depurado y estandarizado tanto el diseño como el funcionamiento para dicho fin pero hablando del campo de la digitalización industrial (la ya mencionada Industria 4.0) estos dispositivos nos presentan varias características de las que podremos hacer uso para conseguir desarrollar un gran número de servicios en esta industria ya que un dispositivo de este tipo posee conectividad a internet, posibilidad de ser totalmente independiente de un *smartphone*, una gran cantidad de sensores…

Todo esto que acabamos de mencionar en conjunto con la idea básica de estos dispositivos, ser llevados en la muñeca, nos otorgará un gran abanico de posibilidades al poder gestionar y ejecutar servicios remotamente desde el dispositivo sin perder ningún detalle por el camino, ahorrando muchos costes de gestión y logrando un objetivo aún más importante: el proporcionar mayor seguridad a las personas en el trabajo.

1.2. Motivación y Objetivos

Tras establecer los pilares del contexto en el que nos moveremos durante el desarrollo de este proyecto, a continuación se potenciará su realización mediante una serie de motivos reflejados por datos oficiales así como mediante la exposición de los objetivos a satisfacer en dicho proyecto.

A continuación, presento una serie de datos del instituto nacional de estadística que avalan mis palabras [19]:

En primer lugar, una primera vista general de los accidentes de trabajo con baja del año 2017 frente a su anterior, tanto en jornada de trabajo como *in itinere* (de camino o vuelta del trabajo) así como los accidentes de trabajo en jornada con baja, graves y mortales.

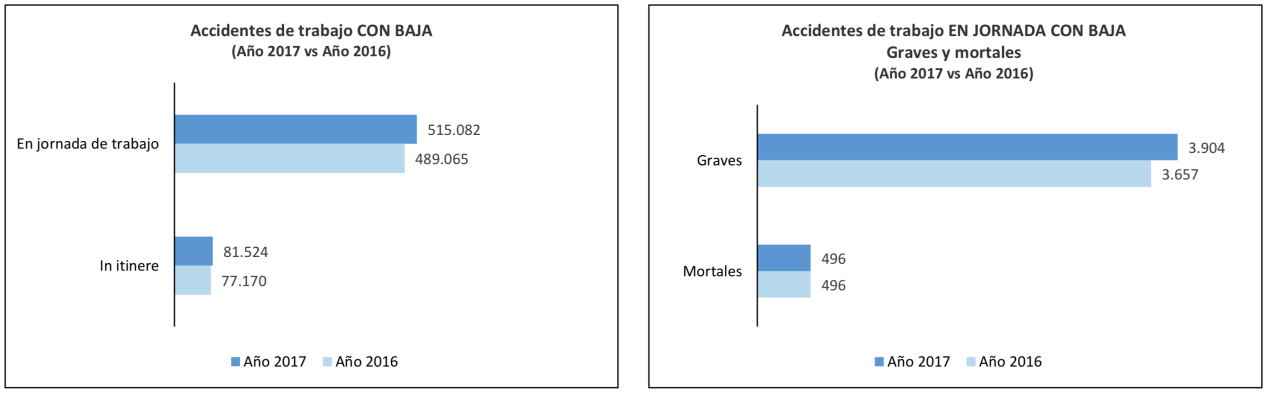

Ilustración 1: Comparativa de los accidentes de trabajo

Apreciamos que el número de accidentes con baja en jornada de trabajo no es nada insignificante, aun así, lo verdaderamente preocupante de un país supuestamente desarrollado es la cifra de los accidentes graves y mortales, no haciendo falta más que una simple cuenta para ver que en nuestro país muere, de media, al menos una persona al día en lo relativo a su jornada laboral.

Ahora, nos centramos en el progreso de los datos anteriores a través de los años, desde 2003 hasta 2017.

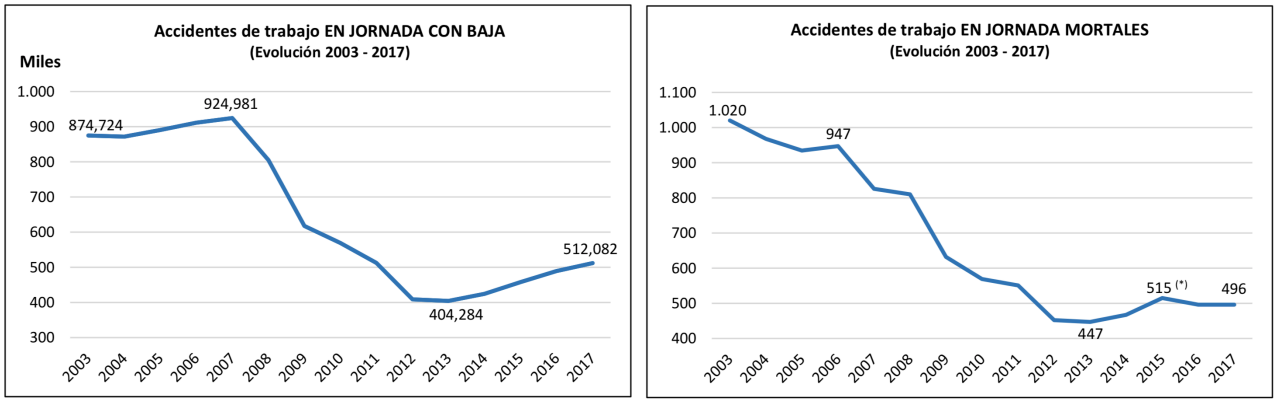

Ilustración 2: Gráfica de los accidentes de trabajo

Podemos contemplar un descenso en los números correspondientes a los accidentes de trabajo en jornada con baja y los accidentes de trabajo en jornada mortales. Aun así, se puede apreciar una tendencia creciente desde el año 2012, pudiéndose deber a cierta dejadez en lo que respecta a la calidad laboral a lo largo de estos últimos años y el aumento de los trabajos temporales.

Y, como se dictaba en los primeros párrafos de esta introducción, en estas tablas podemos observar aquellas secciones de actividad económica más afectadas por los accidentes de trabajo en jornada con baja así como por los accidentes de trabajo en jornadas mortales.

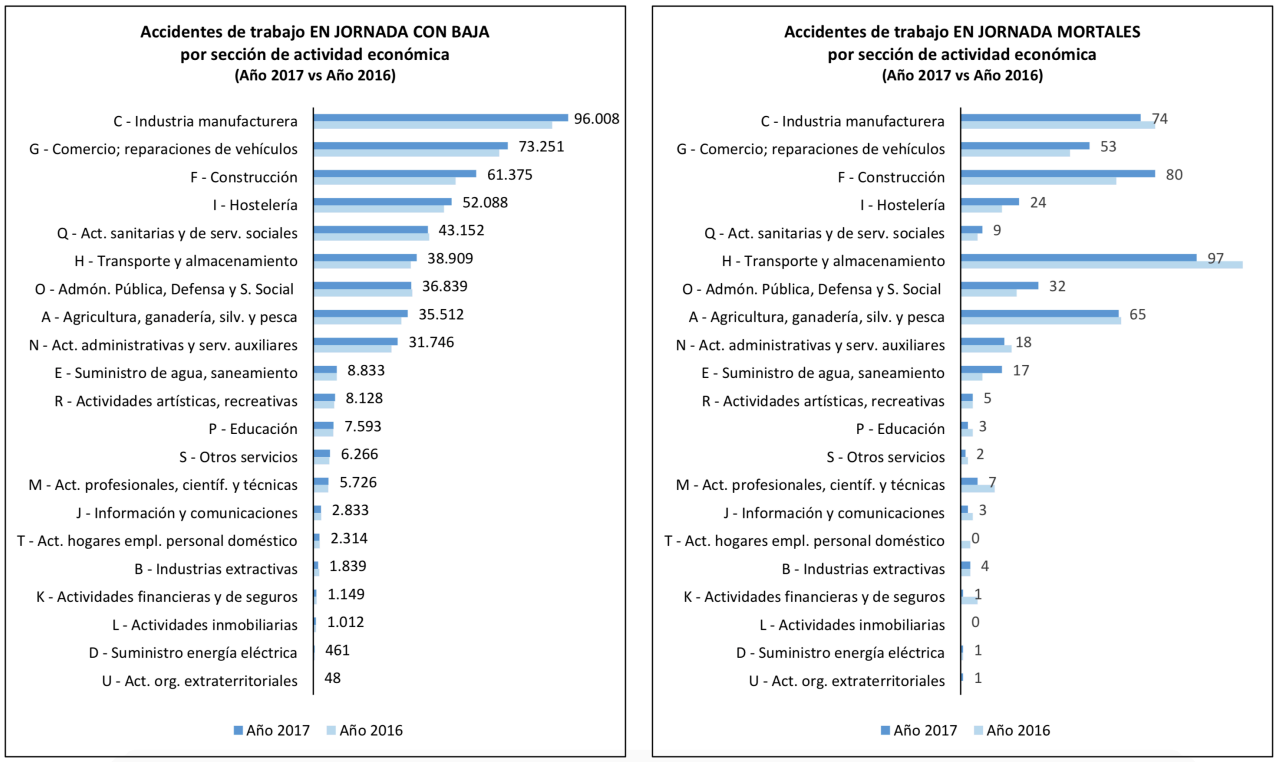

Ilustración 3: Comparativa de los accidentes de trabajo distinguidos por actividad económica

Podemos observar que las industrias más afectadas por los números de accidentes de trabajo en jornada tanto con baja como mortales coinciden en su mayor parte. Esto significa que hace falta una aproximación tecnológica más específica en estos escenarios que encima son los que ejercen más acciones repetitivas.

La tecnología que nos permite reducir estas cifras está presente, pero todavía no se ha materializado ningún producto a nivel específico para solucionar este grave problema, la inseguridad laboral. Así que el objetivo es que, esa parte de la mencionada *Industria 4.0* consiga que estos resultados mejoren a lo largo de los años.

Y como se ha enunciado más arriba, aprovecharemos esos flujos de acciones entre el mundo físico y digital mediante el desarrollo de un prototipo que, mediante una serie de sensores de los que dispone el dispositivo donde se implementará la aplicación, nos mantenga al tanto de los movimientos del trabajador durante su jornada laboral, para poder, a partir de esta lectura y mediante la lógica de negocio de la aplicación, poder detectar aquellas situaciones que conformen alguna peligrosidad para el trabajador y poder monitorizar dichas acciones para poder responder ante aquellas situaciones que sean consideradas como peligrosas para el trabajador, como por ejemplo una caída y una ausencia de valores en las constantes vitales del operario.

A un nivel general, el principal objetivo de SOINCON es el de desarrollar un prototipo que posibilite maximizar la seguridad laboral en la realización de trabajos en la que los accidentes sean frecuentes reduciendo estos con la detección y aviso de los mismos para poder actuar de manera inmediata así como monitorizar a los operarios que trabajan de forma aislada, los cuales sufren un mayor riesgo de que, al padecer un accidente, no puedan ser asistidos en un periodo de tiempo vital.

De este objetivo global podemos fragmentar un conjunto de objetivos mínimos que han de cumplirse al finalizar el desarrollo de este proyecto. Estos son:

- **Desarrollar un prototipo que posibilite maximizar la seguridad laboral en la realización de trabajos en la que los accidentes sean frecuentes reduciendo estos con la detección y aviso de los mismos**.
- **Estudio y selección de las tecnologías disponibles actualmente en el mercado más adecuadas para la resolución del problema presentado**.
- **Realización de una serie de simulaciones y pruebas en las tecnologías escogidas para ajustar las características del producto a las necesidades reales así como a una utilización normal del prototipo final**.
- **Definir una arquitectura que posibilite alojar la lógica de proceso que seguirá el prototipo**.
- **Diseño e implementación del proceso de negocio que posibilite una simulación en el cliente (smartwatch)**.
- **Diseño e implementación del acceso a BBDD para poder realizar la persistencia de los datos considerados de importancia**.

2 Metodología y Herramientas

En esta parte se va a comentar brevemente la metodología llevada a cabo durante el desarrollo de este proyecto así como las distintas etapas principales que se han planificado durante el desarrollo del mismo. Y por último se comentarán, de manera superficial, las tecnologías y herramientas que se han utilizado durante la realización del proyecto.

2.1. Metodología

Durante el transcurso del presente proyecto se ha empleado la bien conocida y practicada, metodología Iterativa e Incremental [22]. El desarrollo iterativo e incremental de software es un método de desarrollo de software que se basa en un aumento progresivo de las adiciones de funciones y en un patrón cíclico de lanzamiento y actualización.

El desarrollo de software iterativo e incremental comienza con la planificación y continúa a través de ciclos de desarrollo iterativos que incluyen la retroalimentación continua de los usuarios y la adición incremental de características que concluyen con la implementación de software terminado al final de cada ciclo.

En este caso, nuestro proyecto se divide en una serie de iteraciones que aparecerán tras una serie de tareas independientes de la metodología y específicas para este proyecto como la investigación y prueba de las tecnologías, el estudio de mercado... Cada iteración se corresponderá con una fase distinguible dentro del proyecto, con esto me refiero a que cada iteración conformará una fase con una función diferente al resto de iteraciones pero que, en común, permitirán la realización e implementación adecuada del proyecto siendo esto el desarrollo e implementación de cada uno de los módulos o partes del prototipo.

Al ser un prototipo, este proyecto no conforma el desarrollo de un producto software corriente ya que la investigación y las pruebas tomarán un papel muy importante al estar trabajando con algo novedoso y poco corriente.

Un dato a resaltar es que, para informar, de manera continua, acerca de los avances que se han ido realizando sobre el proyecto se han mantenido reuniones, de manera periódica, al menos una vez a la semana con el objetivo de aclarar los objetivos y puntualizar aquellos requisitos o características que el proyecto debía de cumplir con cada iteración.

Otro punto importante es que, como se ha mencionado previamente, estamos ante el desarrollo de un prototipo, por lo que no sólo en la primera fase del proyecto se han introducido todas las herramientas con las que trabajar ya que a medida que se iban descubriendo y comentando en las reuniones nuevos posibles elementos beneficiosos con los que trabajar, se han ido incorporando a estudio con sus respectivos análisis y pruebas pero esto se especificará con mayor detalle en el capítulo correspondiente a la Pllanificación.

2.2. Planificación

El desarrollo de este proyecto se contextualiza en la ejecución de un período de prácticas extracurriculares realizadas en la empresa de Soluciones Industriales de Conectividad (SOINCON). Estas prácticas comenzaron el 1 de marzo y finalizaron con la entrega de este proyecto.

Recordando que se ha seguido una metodología de trabajo iterativa e incremental y como se ha mencionado en la anterior sección, las distintas fases se corresponden con las descritas a continuación, estas son: tras un análisis y la realización de un diseño preliminar, a continuación damos comienzo con una fase de investigación y prueba de tecnologías en la que establecemos los requisitos necesarios del dispositivo sobre el que realizaremos el proyecto así como las tecnologías que utilizaremos para poder desarrollar las funcionalidades requeridas en el prototipo. Dentro de esta fase se incluye el estudio de mercado así como la adquisición del *smartwatch* elegido, además de aprender sobre las funcionalidades de este nuevo dispositivo adquirido con la consiguiente documentación.

Ahora, el resto de fases, al estar trabajando con una metodología incremental e iterativa, se distribuirá en iteraciones en las que iremos desarrollando el prototipo que concierne a este proyecto. La primera iteración se corresponde con el desarrollo del módulo de caídas, la segunda con el módulo de constantes vitales, la tercera con el módulo de movimiento, la cuarta con el módulo de llamadas, la quinta se corresponde con el desarrollo del algoritmo “Hombre-Solo” que más adelante se describe con más detalle, en la sexta iteración encontramos el desarrollo del módulo que actuará como director en la orquestación de todos los módulos desarrollados en las iteraciones anteriores, la séptima iteración se corresponde con el diseño y desarrollo de la base de datos con la que interacciona el resto de la aplicación.

Acorde a la metodología, observamos que en cada iteración se incluye una revisión de lo previamente desarrollado (requisitos y arquitectura), la implementación pertinente en dicha iteración, las pruebas correspondientes al módulo desarrollado así como las de integración con el resto de la aplicación y por último, siempre incluimos una tarea de documentación al final de cada iteración para dejar constancia de todo el trabajo realizado, haciéndonos una idea del que queda por desarrollar.

Y tras esas iteraciones realizamos una serie de tareas sobre todo lo practicado anteriormente como el análisis del sistema completo, las pruebas tanto en escenario real como las de aceptación y por último, una formalización de toda la documentación del proyecto.

Esto se explica mejor mediante un diagrama de Gantt [20] que conforma un conjunto de barras, basadas en el calendario, las cuales señalan al responsable de cada actividad, el tiempo transcurrido previsto y la fecha en que se programó el inicio y el fin de la actividad.

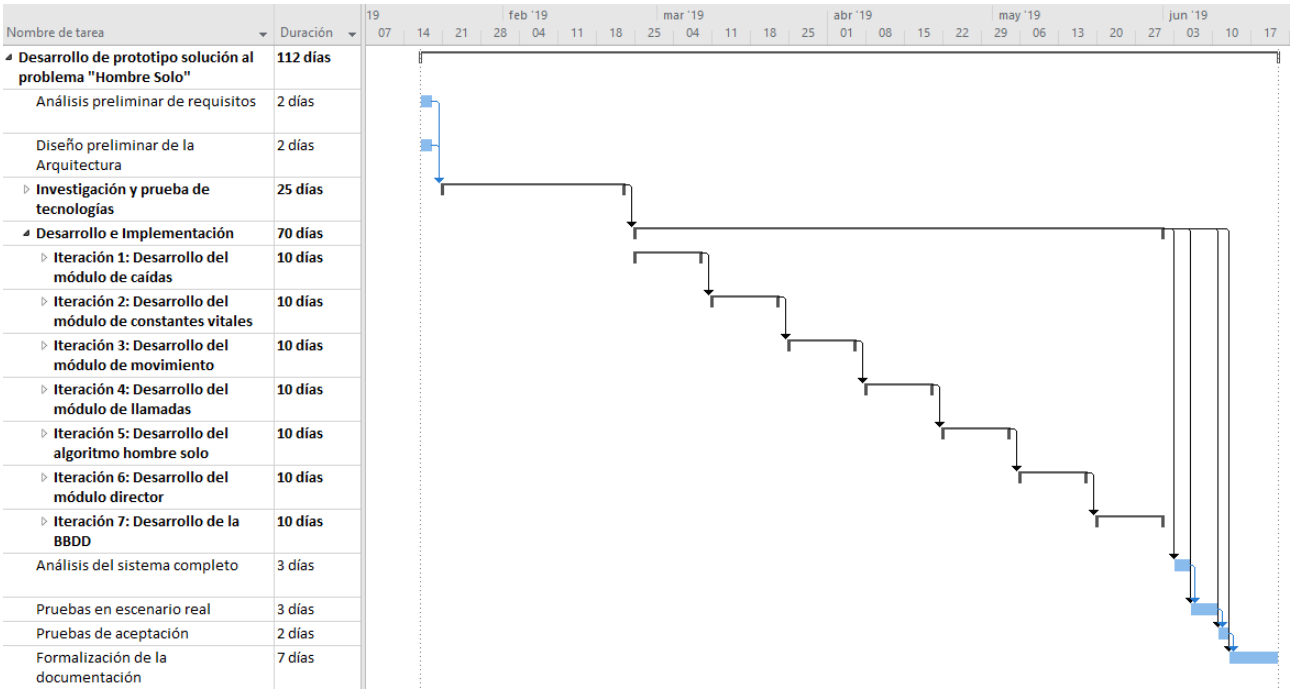

Ilustración 4: Diagrama Gantt del Plan de Trabajo.

2.3. Material Utilizado

En el presente apartado se muestran las tecnologías y herramientas que se han utilizado durante la realización del proyecto.

Tecnologías

Android

Android [11] es un sistema operativo basado en Linux que está diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tabletas. El sistema operativo se ha desarrollado mucho en los últimos 15 años, desde teléfonos en blanco y negro hasta teléfonos inteligentes recientes o miniordenadores. Uno de los sistemas operativos móviles más utilizados en la actualidad es Android. Android es un software que fue fundado en Palo Alto de California en 2003. El proyecto se ha basado en esta tecnología haciendo uso del sistema operativo *Wear OS* que es el sistema que, al igual que *Android*, pertenece a *Google* y se centra en desarrollar y potenciar funcionalidades de los dispositivos corporales (*wearables*).

Java

Java [4] es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes. Este lenguaje de programación ha sido el que se ha utilizado durante todo el desarrollo de la aplicación exceptuando las vistas que se han desarrollado mediante el lenguaje *XML*.

SQLite

SQLite [21] es una biblioteca en proceso que implementa un motor de base de datos SQL transaccional autónomo, sin servidor y sin configuración. El código para SQLite es de dominio público y, por lo tanto, puede utilizarse libremente para cualquier fin, ya sea comercial o privado. SQLite es la base de datos más utilizada en el mundo con más aplicaciones de las que podemos contar, incluyendo varios proyectos de alto calibre. Esta tecnología nos ha permitido desarrollar una base de datos sencilla sobre nuestra aplicación aunque se ha usado mediante *Room*, el *framework* que se describe a continuación.

Room

La biblioteca de persistencia de *Room* [10] proporciona una capa de abstracción sobre SQLite para permitir un acceso más robusto a la base de datos mientras se aprovecha toda la potencia de SQLite. La biblioteca le ayuda a crear una caché de los datos de su aplicación en un dispositivo que esté ejecutando su aplicación. Esta caché, que sirve como la única fuente de verdad de su aplicación, permite a los usuarios ver una copia consistente de información clave dentro de su aplicación, independientemente de si los usuarios tienen una conexión a Internet. Este *framework* nos ha servido para subir el nivel de abstracción al trabajar en el desarrollo y comunicación con la base de datos de la aplicación ya que el núcleo del desarrollo de esta es la tecnología previamente definida: *SQLite*.

Git

Git [12] es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para manejar todo tipo de proyectos, desde pequeños hasta muy grandes, con rapidez y eficiencia. Esta tecnología nos ha permitido mantener un versionado estructurado, coherente y organizado de nuestro código, pudiendo desarrollar diferentes módulos al mismo tiempo de manera independiente así como mantener nuestro proyecto almacenado en la nube.

Herramientas

Android Studio

Android Studio [7] es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA . Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan tu productividad durante la compilación de apps para Android, como las siguientes:

- Un sistema de compilación basado en Gradle flexible.
- Un emulador rápido con varias funciones.
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android.
- Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK.
- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código.
- Gran cantidad de herramientas y frameworks de prueba.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etc.
- Compatibilidad con C++ y NDK (*Native Development Kit*).
- Soporte incorporado para *Google Cloud Platform*, lo que facilita la integración de *Google Cloud Messaging* y *App Engine*.

En este entorno de desarrollo hemos realizado el proyecto en toda su totalidad dado que da soporte al desarrollo de todas las partes de la misma así como la realización de sus pruebas y simulaciones.

SourceTree

SourceTree [13] es un cliente gratuito de Mercurial y Git para plataformas Windows y Mac. Proporciona una interfaz gráfica para repositorios, entre usuarios y Git, en la que se simplifica su uso para principiantes, que no han dominado Git, y expertos, que pueden ser más productivos centrándose únicamente en el código. Esta herramienta nos ha permitido disponer de una manera de mantener el versionado de nuestro código de una manera más visual así como una mayor facilidad en su utilización en vez de haber recurrido al uso comandos.

Lucidchart

Es una herramienta web que permite la creación de todo tipo de diagramas (de flujo, organigramas, esquemas, UML...). Esta herramienta permite un uso colaborativo de la misma pudiendo trabajar en tiempo real con otras personas y además el aprendizaje de su uso ha sido muy sencillo así como la adecuación a su interfaz y me ha permitido realizar diagramas de varios tipos y los cuales se mostrarán más adelante.

3 Presentación del Problema y Análisis de Requisitos

A continuación se procede a poner el problema a solucionar en el contexto adecuado para que haya sido desarrollado así como a establecer los requisitos que cumplirá sistema en cuestión, tanto funcionales como no funcionales.

3.1. Presentación del Problema

Con el objetivo bien claro de prevenir un posible accidente laboral en el desarrollo del trabajo, se ha contemplado una solución absolutamente configurable por cada cliente de la aplicación, entendiendo al cliente como a aquella empresa la cual usará el producto software entre sus diferentes operarios/-trabajadores. Esto nos permite disponer de una aplicación que permite varias permutaciones en varios ámbitos de sus características ya que se debe de adaptar a cada escenario del usuario final (siendo aquí el operario/trabajador). Contemplando los escenarios más extremos para que la lógica final de la aplicación sea correcta en todos los posibles casos de ejecución o escenarios.

El escenario el cual este proyecto abarca es el de un operario que está en continuo movimiento realizando operaciones de carga y descarga, por ejemplo, pero que no utiliza ninguna máquina o vehículo para ello, esto es que el trabajador en caso de sufrir una caída o desmayo, su cuerpo caerá bruscamente en ambos casos y la lógica de la aplicación deberá reaccionar ante este accidente laboral. Se detectará la caída debido a una un aumento y una disminución súbitas en el acelerómetro del dispositivo que lleva el operario y a partir de ahí se seguirá con los pasos definidos por el algoritmo que más tarde se desarrollará en la sección de Diseño.

Por lo que tendremos una aplicación que realmente se conformará de varias "mini-aplicaciones" a las que denominaremos módulos. Los cuales son:

- **Módulo de detección de caídas**: El cual se encargará de detectar aquellos patrones que se consideren como caída. En este caso los patrones consistirán en un traspaso del umbral de un valor preestablecido como tal.
- **Módulo de detección de constantes vitales**: Este módulo será el encargado de detectar las constantes vitales del usuario.
- **Módulo de detección de movimiento**: Detectará el movimiento del trabajador para decidir si se da la alarma o no.
- **Módulo de Contacto (Llamadas)**: A cargo de realizar el contacto exterior mediante llamadas en caso que la lógica de proceso lo determine como necesario.

3.2. Requisitos Funcionales

Tras el estudio del sistema que nos concierne en este documento podemos proceder a exponer los requisitos que este nos presente.

Los requisitos funcionales (RF) [15] describen las funciones que el software debe ejecutar. A veces se conocen como capacidades o características. Un requisito funcional también puede describirse como uno para el cual se puede escribir un conjunto finito de pasos de prueba para validar su comportamiento.

Así que, tras conocer el significado de este tipo de requisitos, los identificados en este sistema se centran en la parte del sistema, ya que estamos hablando de un sistema que se ejecutará sin que el usuario directo (trabajador) tenga que interaccionar, de manera continua, con él. Esto es que el sistema conlleva la mayor parte de las decisiones siendo siempre posible abortar ciertas operaciones y es ahí donde aparecerá el usuario como ahora veremos:

ID	**Descripción del Requisito**
RF001	Se leerán los datos de sensores disponibles en el dispositivo para detectar lo que podría ser una posible caída.
RF002	Se considerará caída a cualquier valor del acelerómetro que sobrepase el umbral definido en las variables globales.
RF003	Se leerán los datos vitales (ritmo cardíaco) tras detectar lo que podría ser una posible caída.
RF004	Se detectará el movimiento del usuario mediante el registro del detector de pasos del dispositivo del trabajador.
RF005	Si detectamos caída y posteriormente no hay movimiento se efectuará una llamada.
RF006	En caso de confirmar una caída el sistema deberá de almacenar en la base de datos los datos que sean pertinentes en ese caso (fecha y hora, constantes vitales, datos de los sensores del acelerómetro, trabajador...).
RF007	En caso de la detección de una posible caída el usuario dispondrá de los segundos indicados en las variables globales para cancelar la llamada antes de que se realice.

Cuadro 1: Requisitos Funcionales.

3.3. Requisitos No Funcionales

Tras especificar los requisitos funcionales que debe cumplir el sistema con el que trabajamos, procederemos a contemplar aquellos requisitos no funcionales que el sistema tendrá que satisfacer.

Los requisitos no funcionales (RNF) [15] son aquellos que actúan para limitar la solución. Se conocen a veces como restricciones o requisitos de calidad.

Se pueden clasificar con mayor precisión en función de si se trata de requisitos de rendimiento, de mantenimiento, de seguridad, de usabilidad o de accesibilidad acorde la ISO/IEC 25010 [15].

ID	**Descripción del Requisito**	**Importancia**
	Rendimiento	
RNF001	Los tiempos de respuesta ante una acción que se pueda considerar como una caída deberán ser a lo sumo de 2 segundos.	Alta
RNF002	En caso de caída, se debe reportar el aviso en 1 minuto como máximo.	Alta
RNF003	La batería del sistema debe de soportar la actividad de una jornada laboral completa.	Alta

Mantenibilidad		
RNF004	El sistema adoptará un diseño modular (lo máximo que se pueda) para facilitar tanto su mantenimiento como su posible extensión en un futuro.	Alta
RNF005	El servicio se podrá utilizar en distintos dispositivos smartwatch siempre que dispongan de los sensores utilizados y el sistema operativo sea *Wear OS*.	Media
Usabilidad		
RNF006	La interacción directa del usuario con la aplicación deberá ser limitada a ciertos casos en los que sea absolutamente necesario.	Media

Cuadro 2: Requisitos No Funcionales.

4 Diseño e Implementación

Ahora, tras haber descrito el dominio del problema y ya definidos los requisitos, procederemos a plantear el diseño, a distintos niveles, que se ha seguido en la realización del proyecto así como la implementación de las distintas capas que conforman la arquitectura.

4.1. Diseño Arquitectónico

En esta sección se presenta el diseño arquitectónico seguido en la cual encontraremos distintos niveles de abstracción o de detalle del diseño, ya sea desde una perspectiva de arquitectura de alto nivel, en el cual trataremos con más profundidad los módulos del sistema o desde la perspectiva de bajo nivel en el cual discutiremos el algoritmo que se ha seguido para conseguir la funcionalidad que cumplirá con los requisitos preestablecidos.

Arquitectura del Sistema

En lo que nos respecta a la arquitectura de alto nivel, nos referiremos a los diferentes módulos con los que trabaja el algoritmo de proceso de "Hombre Solo" así como con el resto de procesos que serán manejados por un módulo director. Estos módulos trabajarán de manera independiente, cada uno con su función específica y bien definida, pero de también lo harán de manera conjunta para otorgar la funcionalidad completa requerida a la aplicación.

De esta manera, tenemos las capas de presentación, negocio, soporte y acceso a los datos además de una base de datos en la que persistir los datos que consideren necesarios, por lo que podríamos representar las relaciones entre dichos componentes mediante el siguiente diagrama:

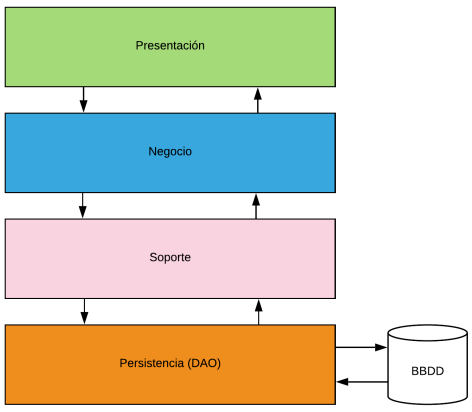

Ilustración 5: Diagrama de Arquitectura.

- Disponemos de una **capa de presentación** en la que estaremos hablando de aquellos módulos con los que interacciona directamente el usuario como es:
  - El módulo director (MainActivity). Desde este módulo se orquesta, a petición del usuario, el proceso/algoritmo que se ejecutará en la aplicación.
  - También formarán parte de esta capa aquellas vistas que irán otorgando los procesos o algoritmos de la capa de negocio que ahora expondremos.
- En este caso, hablamos de la **capa de negocio** y es la encargada de contener lo que denominamos como procesos con los que interacciona directamente el módulo director el cual ejecutará uno de los procesos que se encuentran en esta capa. Estos procesos son aquellos módulos que contienen el algoritmo en cuestión para cada caso requerido, en nuestro caso:
  - Módulo de proceso "Hombre Solo": Este módulo es aquel que contendrá la lógica de proceso central para desarrollar e implementar la funcionalidad de negocio principal de los requerimientos, ya que este es el proceso que, orquestando el resto de módulos, detectará la caída del trabajador y desarrollará la lógica pertinente que se describe más adelante.
  - Módulo de proceso "Detección de CV": Este módulo que representa otro proceso al que puede ser llamado desde el módulo director. Se ha desarrollado para mostrar la escalabilidad y reusabilidad del resto de módulos que describiremos en la capa de soporte.
- Ahora entramos a explicar lo que conformaría la **capa de soporte** de la aplicación. El contenido de esta capa se ha estado mencionando a lo largo de este documento y con este contenido nos referimos a los diferentes módulos que asisten a los diferentes procesos de los cuales dispone la aplicación (como el de "Hombre Solo"). Estos módulos interaccionan con el exterior para recabar información, en diferentes ámbitos, con los que daremos soporte al algoritmo que se esté ejecutando en ese momento, ya que este algoritmo necesitará de datos para poder proseguir con su flujo, así que la interacción con estos módulos será crucial para hacer posible el funcionamiento del algoritmo. Estos módulos son:
  - **Módulo de Detección de Caídas**: Este módulo es el encargado de, una vez activado, detectar una posible caída tras una variación específica en el sensor del acelerómetro.
  - **Módulo de Constantes Vitales**: En este caso, esta pieza auxiliar será la encargada de, mediante el sensor de detección de ritmo cardíaco, proveer de este dato al módulo llamador.
  - **Módulo de Detección de Movimiento**: Es el encargado de que, cuando sea llamado, monitorizar el movimiento del trabajador mediante el sensor de detección de pasos en un rango temporal previamente definido, de esta manera, la lógica de la aplicación determinará si el operario se mueve tras la caída.
  - **Módulo de Contacto (Llamadas)**: Este módulo está a cargo de realizar el contacto exterior mediante llamadas en caso que la lógica de proceso lo determine como necesario.
- Por último, disponemos del **módulo de persistencia o acceso a los datos** el cual interacciona con la BBDD integrada en la aplicación.

4.2. Diseño Detallado e Implementación de la Capa de Presentación

Tenemos que tener en cuenta que para el desarrollo de esta capa, se han tenido en cuenta varias consideraciones siendo la primera, la realización de un estudio de interacción y patrones [16] de comportamiento del usuario ante los diseños de diferentes aplicaciones y situaciones dentro de estas, ya que todos los sistemas operativos proponen diferentes formas de interactuar con los elementos en pantalla. Conocer la diferencia entre ellos y utilizar elementos familiares para el usuario, asegura que se sienta cómodo y seguro usando la aplicación.

Tenemos que tener en cuenta una segunda cosa importante, estamos desarrollando una aplicación Android, sí, pero no para un dispositivo móvil, sino para algo innovador y que pretende hacerse ver dentro del mercado de desarrollo de aplicaciones, y, como ya sabemos, hablamos de que vamos a desarrollar una aplicación para un *SmartWatch* y que, aunque este esté basado en Android, usa su propio sistema operativo (*Wear OS*) y habrá que tratar los aspectos que vamos a definir con un enfoque algo diferente al que haríamos al desarrollar una aplicación para un dispositivo móvil.

En esto aparece lo que se denomina como "Los principios de experiencia de usuario" esto es que cada sistema operativo tiene su propia identidad que es reflejada en la apariencia y comportamiento de cada uno de los elementos que componen su interfaz. En ellos imprime su personalidad, lo que hace que la experiencia sea diferente a las demás.

Sin embargo, todos comparten algunos puntos de vista fundamentales que se manifiestan en el diseño de sus interfaces. Los siguientes conceptos son considerados componentes clave del sistema operativo y de las aplicaciones que en él habitan. Estos componentes son:

Simplicidad

La simplicidad visual está directamente relacionada con la usabilidad. Ser simple implica en cierta medida ser mínimo, contar con pocos elementos, pero sobre todo, que aquellos presentes en la interfaz tengan una función bien definida que contribuya a cumplir el objetivo de la app y ayude al usuario. Por lo que la interfaz gráfica de usuario se ciñe a mostrar únicamente lo esencial durante el proceso de ejecución de la aplicación, considerando cualquier otra información "basura" para la experiencia de usuario.

Consistencia

Una app tiene diferentes pantallas que la componen y al mismo tiempo, está dentro de un sistema operativo que propone un determinado aspecto visual e interacción. El usuario de Android, iOS o Windows Phone ya está habituado a ellos y espera que las aplicaciones se comporten de la misma manera.

La consistencia, entonces, se trata de respetar estos conocimientos y costumbres del usuario, no solo en el interior de la aplicación, sino también en relación con el resto del SO. Esto favorece el uso intuitivo de la app, ya que el usuario puede prever su comportamiento sin demasiado esfuerzo. Esto lo conseguimos mediante la utilización de elementos básicos, intuitivos y bien conocidos por el usuario como son los botones, los textos, las notificaciones, los colores y las vibraciones.

Navegación intuitiva

Un aspecto que merece mucha atención en una aplicación es la forma de navegar entre contenidos, de manera que resulte fácil de comprender para el usuario, evitando la sensación de desorientación que puede ocasionar una navegación confusa.

La navegación intuitiva está también relacionada con la consistencia. Cada sistema operativo propone diferentes elementos para navegar por la app como botones, pestañas y paneles. Hacer uso de ellos hará que el usuario los reconozca a primera vista y, solo con estos componentes, ya sepa cómo ir de una sección a otra.

Notificaciones

Cuestiones como: ¿Qué está haciendo la app? ¿Cómo saber que la acción ha funcionado? ¿Ha terminado o hay que hacer algo más?, seguramente pasan por la cabeza de un usuario cuando no tiene

ninguna confirmación visual de que la acción que acaba de realizar ha ido bien.

Para mitigar esta incertidumbre, se aconseja mostrar explícitamente cómo han ido las cosas o que sucederá en breve con simples mensajes de confirmación. Este tipo de mensajes se presentan en pequeños avisos que desaparecen luego de unos segundos.

A diferencia de los cuadros de diálogo, las notificaciones no requieren la intervención del usuario ni tampoco interrumpen su flujo de trabajo.

En nuestra adaptación de los conceptos al desarrollo *SmartWatch*, limitaremos estos mensajes a aquellos precisos momentos en los que la aplicación ha desencadenado una acción o serie de acciones que posibilita a que el usuario pueda seguir el flujo de la aplicación y perciba lo que ocurre en ese momento así como lo que va a ocurrir de manera inmediata a su lectura.

Esto lo haremos mediante el uso de notificaciones *Toast*, incorporadas en la librería de Android. Esta notificación aparece por un corto período de tiempo, con un texto (generalmente de una sola línea) que le da feedback al usuario, por ejemplo, mientras la app está guardando un cambio. Al tratarse de un tipo de aviso que puede no ser percibido por el usuario, se usa para comunicar mensajes que no tienen una importancia crítica.

Introducción de datos

La introducción de datos en el móvil puede ser tediosa cuando se trata de campos que requieren el uso del teclado, un elemento que ocupa gran parte de la pantalla y que dificulta la navegación entre los campos para introducir información.

En el caso del desarrollo para dispositivos *SmartWatch* hay que tener clara una cosa que trataremos como un mantra: “Las malas aplicaciones de *SmartWatch* requieren mucho input por parte del usuario”. Por lo que la interfaz de usuario, durante la ejecución completa de la aplicación, se ha ceñido a un diseño en el que la única interacción sea pulsar un botón y nada más, ya que acceder a teclado en estos casos sería un fallo de diseño notorio (aunque se podría recurrir al dictado por voz, aún así no se ha tenido que recurrir a esto).

Tras haber descrito lo que esta capa debe respetar, la capa de presentación será la responsable de posibilitar la interacción con el usuario, dando la opción de seleccionar el proceso a ejecutar mediante una simple pantalla basada en botones:

Ilustración 6: Captura de pantalla *MainActivity*.

Esta pantalla será la directora de los procesos almacenados en el dispositivo y que, dependiendo de la elección del usuario, serán lanzados. Por ejemplo, en este caso, lanzamos el proceso correspondiente a realizar una lectura del ritmo cardíaco así como su posterior muestra por pantalla:

Ilustración 7: Captura de pantalla *DeteccionDeVCActivity*.

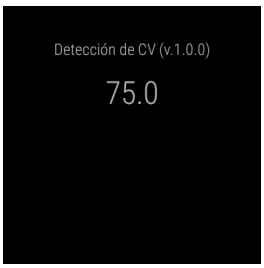

Ilustración 8: Captura de pantalla *DeteccionDeVCActivity* que muestra el ritmo cardíaco medido.

Como podemos observar, el proceso de detección de constantes vitales es muy simple de comprender y utilizar, se observa un botón el cual tendremos que presionar para que, a continuación, se realice la detección y muestra del ritmo cardíaco por la pantalla.

También podremos arrancar el proceso principal en el cual este proyecto se centra, el proceso Hombre-Solo. Simplemente tendremos que seleccionarlo en la pantalla que muestra una de las ilustraciones anteriores (Véase Ilustracion 6) para lanzar el algoritmo deseado:

Ilustración 9: Captura de pantalla *HombreSoloActivity* nada más arrancar el algoritmo.

A partir de esa pantalla, si se detecta una caída se indicará mediante una secuencia de luces y patrones de vibración que para simplificar, de manera visual, su documentación se ha indicado mediante un *Toast* como podemos apreciar:

Ilustración 10: Captura de pantalla en el momento de producirse una caída.

Tras haber registrado una caída, el dispositivo leerá las constantes vitales y se procederá a la pantalla que detecta el movimiento, en el cual podemos observar que se indican 2 cosas, el tiempo restante para que se realice la llamada de aviso y los pasos dados por el usuario. Esto es que el usuario, si da más de un número de pasos, al terminar el tiempo no se realizará la mencionada llamada de aviso debido a que se ha detectado movimiento por parte del trabajador. Esta llamada también se puede cancelar de manera manual mediante un botón. Cuando quedan menos segundos que una cantidad específica predefinida, el dispositivo pondrá la pantalla de un color vistoso además de volver a recurrir a patrones de vibración y sonidos hasta que se termine la cuenta atrás. Todo esto lo podemos apreciar en las siguientes capturas:

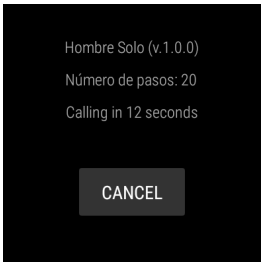

Ilustración 11: Captura de pantalla segundos más tarde de producirse una caída. Se pueden observar en pantalla los elementos mencionados.

Ilustración 12: Captura de pantalla cuando se activa la etapa de alarma debido a que restan pocos segundos para realizar el aviso.

Después de esto podremos presenciar escenarios diferentes dependiendo de lo que haya ocurrido:

1. Se detecta movimiento o se cancela el aviso de manera manual por lo que se vuelve a la pantalla de inicio (Véase Ilustración 9).
2. Que se realice la llamada de aviso.

4.3. Diseño Detallado e Implementación de la Capa de Negocio

En esta capa reside la responsabilidad de determinar las reglas que deben cumplirse acorde a la lógica de proceso especificada en los requerimientos. Esta lógica de proceso hará uso de los diferentes módulos descritos previamente para recabar datos y la labor del algoritmo será hacer que esos datos recogidos por los módulos se conviertan en información valiosa para alcanzar la funcionalidad final requerida.

En lo que respecta a nuestro caso, esta capa se corresponderá con los algoritmos disponibles en la aplicación y en el cual destaca el de “Hombre-Solo” y del cual el proceso que se ha definido es el siguiente:

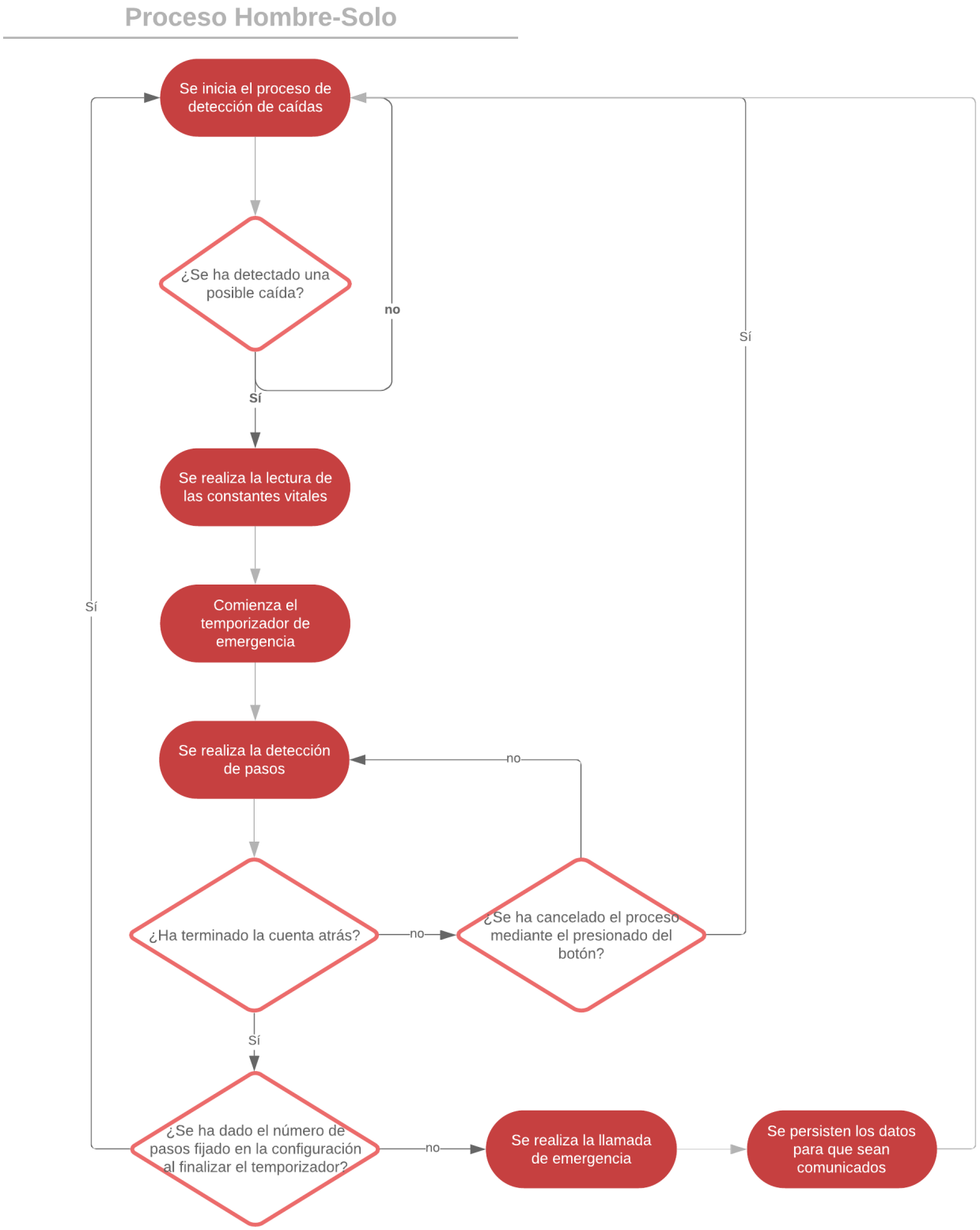

Ilustración 13: Diagrama de flujo del algoritmo ”Hombre-Solo”.

La manera en la que se comunica esta capa con la de soporte (la cual describiremos a continuación) es la siguiente [8]:

El objeto *Intent* empleado para iniciar una actividad que mostrará un resultado no tiene nada de particular, pero transferiremos un argumento de valor entero adicional al método `startActivityForResult()`.

El argumento de entero es un çódigo de solicitud"(*request code*) que identifica la solicitud que enviaremos (esto es necesario ya que dispondremos de varios códigos de solicitud dependiendo de la función que queramos ejecutar en cada momento). Cuando se recibe el *Intent* de resultado, el retorno de la llamada proporciona el mismo código de solicitud para que nuestra app pueda identificar debidamente el resultado y determinar cómo manejarlo. Este es el código de invocación de un nuevo *Intent* que iniciará el módulo de caídas:

```
1 Intent i = new Intent(HombreSoloActivity.this, CaidasActivity.class);
2 startActivityForResult(i, VariablesGlobales.getCaidasActivityRequestCode());
```

Los códigos de solicitud se han agrupado en la clase que contiene las variables globales como podemos observar:

```
1  private static final int CAIDAS_ACTIVITY_REQUEST_CODE = 0;
2  private static final int CONSTANTES_VITALES_ACTIVITY_REQUEST_CODE = 1;
3  private static final int MOVIMIENT0_ACTIVITY_REQUEST_CODE = 2;
4  private static final int LLAMADAS_ACTIVITY_REQUEST_CODE = 3;
5
6  public static int getCaidasActivityRequestCode() {
7     return CAIDAS_ACTIVITY_REQUEST_CODE;
8  }
9
10 public static int getConstantesVitalesActivityRequestCode() {
11    return CONSTANTES_VITALES_ACTIVITY_REQUEST_CODE;
12 }
13
14 public static int getMovimient0ActivityRequestCode() {
15    return MOVIMIENT0_ACTIVITY_REQUEST_CODE;
16 }
17
18 public static int getLlamadasActivityRequestCode() {
19    return LLAMADAS_ACTIVITY_REQUEST_CODE;
20 }
```

Ahora, en el código del módulo lanzado desde el algoritmo (módulo de caídas en este caso), cuando ya tengamos un resultado tras la ejecución de este, prepararemos el dato de la siguiente manera para retornar, como veremos a continuación, a la actividad que controla el algoritmo y poder recibir la información recabada en el módulo:

```
1 Intent returnIntent = new Intent();
2 returnIntent.putExtra("hayCaida", true);
3 returnIntent.putExtra("accelerometerX", gX);
4 returnIntent.putExtra("accelerometerY", gY);
5 returnIntent.putExtra("accelerometerZ", gZ);
6 setResult(Activity.RESULT_OK, returnIntent);
7 finish();
```

Tras haber preparado el dato que deseamos enviar desde el módulo y haber finalizado la ejecución de dicho módulo de manera correcta toca recoger el dato. Por lo que se invoca en el programa que contiene el algoritmo (llamador) al método `onActivityResult()` de la actividad. Este método incluye tres argumentos:

- El código de solicitud con el cual se inició la ejecución del Intent (módulo) correspondiente, llamando a `startActivityForResult()`.
- Un código de resultado especificado por la segunda actividad. Este código puede ser `RESULT_OK` si la operación se realizó correctamente o `RESULT_CANCELED` si se canceló la operación o esta falló por algún motivo.
- Un *Intent* con la información del resultado.

```
1 if (requestCode == VariablesGlobales.getCaidasActivityRequestCode()) {
2
3     hayCaida = data.getBooleanExtra("hayCaida", false);
4     accelerometerX = data.getFloatExtra("accelerometerX", -999);
5     accelerometerY = data.getFloatExtra("accelerometerY", -999);
6     accelerometerZ = data.getFloatExtra("accelerometerZ", -999);
7     returnFromCaidasActivity();
8
9 }
```

Otro punto a considerar en la invocación de los Intents es que, en algunas ocasiones necesitaremos transferir datos importantes para evitar el acoplamiento de los módulos con otras clases. Pasaremos aquellos datos que los módulos requieren para poder desencadenar su lógica correctamente, esto lo conseguimos mediante el uso del método `putExtra()` (de la clase Intent), el cual nos permitirá transferir datos extra y al cual, como veremos, le pasaremos el identificador o key de ese recurso (que más tarde veremos que nos servirá para poder recoger el dato desde la otra actividad que en este caso será el módulo de caídas) y, por supuesto, el dato que queremos transferir entre actividades. Aquí podemos observar cómo se desarrolla este proceso mediante código:

```
1 Intent i = new Intent(HombreSoloActivity.this, MovimientoActivity.class);
2 i.putExtra("numSegundos", VariablesGlobales.getNumSeconds());
3 startActivityForResult(i, VariablesGlobales.getMovimient0ActivityRequestCode());
```

Ahora nos encontramos en el otro lado, es decir, ya en el módulo en cuestión y para poder ejecutar la lógica que se espera de dicho módulo necesitaremos obtener los datos que se han pasado a través del *Intent* como hemos visto previamente. El proceso es parecido al del envío de datos pero en este caso usaremos el método, también de la clase *Intent*, `getExtras()`, al cual tendremos que indicar que es lo que se espera recibir, es decir, el tipo, así como la key, que indicamos en la actividad que enviaba el dato, y que nos permitirá distinguir nuestro dato de los otros posibles que se hayan enviado. Esto lo haremos de la siguiente manera:

```
1 numSegundosGlobal = getIntent().getExtras().getInt("numSegundos");
```

4.4. Diseño Detallado e Implementación de la Capa de Soporte

En este caso hablamos de los módulos que asisten al algoritmo principal en distintas fases de este y de los cuales la lógica de proceso de negocio hace uso para poder interaccionar con el escenario exterior (en mayor parte) en distintos ámbitos que son necesarios para poder desempeñar la funcionalidad de la aplicación de una manera que satisfaga los requerimientos. En esta aplicación haremos uso de 2 tipos de módulos:

- Aquellos que desencadenan un proceso externo pero que no recogen datos del exterior. El único módulo que se clasificará en este grupo será el módulo de llamadas, el cual desencadena un

Intent que Android mantiene como nativo y cuya función es realizar una llamada, como podemos observar:

```
1        /**
2 * Metodo que nos permite realizar una llamada a un numero predefinido como constante
      global.
3 */
4 private void realizaLlamada() {
5
6     PhoneCallListener phoneListener = new PhoneCallListener();
7     TelephonyManager telephonyManager = (TelephonyManager) this
8             .getSystemService(Context.TELEPHONY_SERVICE);
9     telephonyManager.listen(phoneListener,
10            PhoneStateListener.LISTEN_CALL_STATE);
11
12    Intent i = new Intent(Intent.ACTION_CALL);
13    i.setData(Uri.parse("tel:" + numTelefono));
14    startActivity(i);
15
16 }
```

- Aquellos que desencadenan un proceso externo recogiendo datos del exterior de diferente ámbito. Estos módulos serán el de detección de caídas, el de detección de constantes vitales y el de detección de movimiento. Para entender bien la función de este grupo debemos entender cómo se lee un sensor, por lo que, a continuación se describe a qué nos referimos exactamente.

4.4.1. Implementación de un sensor

Como se ha mencionado, la responsabilidad de esta capa es la de interactuar con el entorno recabando los datos del exterior en tiempo real para poder realizar las operaciones pertinentes para cada estímulo y así poder generar una respuesta adecuada y a tiempo. Estas respuestas podrán ser generadas tras haber operado con los datos que necesitamos que se han obtenido a través de lo que se denominan **sensores**. Estos sensores son capaces de proporcionar datos sin procesar (*raw* o crudo) con alta precisión y exactitud. Para poder realizar dichas acciones haremos uso del framework de sensores otorgado por el *Android SDK* (*Software Development Kit*) que nos permitirá leer esos datos en crudo de la gran mayoría de sensores (hardware o software) del dispositivo de una manera que nos permite desarrollar el proyecto de manera independiente al dispositivo físico donde se instalará (aunque de él dependerá la velocidad de reacción y respuesta). La plataforma Android soporta tres grandes categorías de sensores [9]:

Sensores de movimiento

Estos sensores miden las fuerzas de aceleración y las fuerzas de rotación a lo largo de tres ejes. Esta categoría incluye acelerómetros, sensores de gravedad, giroscopios y sensores de vectores rotativos.

Sensores ambientales

Estos sensores miden varios parámetros ambientales, como la temperatura y la presión del aire ambiente, la iluminación y la humedad. Esta categoría incluye barómetros, fotómetros y termómetros.

Sensores de posición

Estos sensores miden la posición física de un dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.

La estructura del sensor proporciona varias clases e interfaces que le ayudan a realizar una amplia variedad de tareas relacionadas con los sensores. Por ejemplo, se puede utilizar el marco de trabajo de sensores para hacer lo siguiente:

- Determinar qué sensores están disponibles en un dispositivo.
- Determinar las capacidades de un sensor individual, como su alcance máximo, fabricante, requisitos de potencia y resolución.
- Adquirir los datos brutos del sensor y definir la velocidad mínima a la que se adquieren los datos del sensor.
- Registrar y desregistrar a los oyentes o *listeners* de eventos del sensor que monitorean los cambios del sensor.

Antes que nada, debemos de disponer de un dispositivo que disponga de tales sensores. En este caso, realizaremos la lectura de las constantes vitales (ritmo cardíaco). Así que, para realizar la lectura de un sensor de manera correcta deberemos seguir los siguientes pasos [14]:

1. Habilitar los permisos en el `manifest.xml`.

   En este caso, el sensor correspondiente a la medición del ritmo cardíaco es el correspondiente a los sensores corporales: `BODY_SENSORS`.

```
1 <uses-permission android:name="android.permission.BODY_SENSORS" />
```

2. Obtener el acceso al sensor del ritmo cardíaco.

   Para obtener acceso a cualquier sensor de hardware, se necesita un objeto SensorManager. Para crearlo, se utiliza el método `getSystemService()` de nuestra clase *Activity*, y le pasamos la constante `SENSOR_SERVICE`.

```
1 private SensorManager mSensorManager;
2 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

   Ahora, podemos crear un objeto Sensor para el sensor de detección de ritmo cardíaco, invocando el método `getDefaultSensor()` y pasándole la constante `TYPE_HEART_RATE` que especifica el tipo de sensor con el que trabajaremos.

```
1 mSensorHR = mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
```

3. Registrar el *listener*.

   Para poder leer los datos `raw` o crudos generados por un sensor, se debe asociar a un *SensorEventListener* invocando el método `registerListener()` del objeto *SensorManager*. Al hacerlo, también se debe especificar la frecuencia con la cual deberían leerse datos del sensor. En nuestro caso, al estar trabajando en un prototipo del que requerimos los mínimos tiempos de respuesta, pondremos `SENSOR_DELAY_FASTEST` para que, aunque consuma algo más, nos asegure que los datos se obtienen en el menor tiempo posible.

```
1 @Override
2 protected void onResume() {
3     super.onResume();
4     mSensorManager.registerListener(this, mSensorHR, SensorManager.
          SENSOR_DELAY_FASTEST);
5 }
```

Como podemos ver, registramos el listener en el método `onResume()` de la clase *WearableActivity*. También podemos eliminar el *listener* en el método `onPause()`:

```
@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
```

4. Utilizar los datos *raw* o crudos.

   El objeto *SensorEvent*, disponible dentro del método `onSensorChanged()`, posee un array *values* que contiene todos los datos crudos generados por el sensor asociado. En el caso del sensor de ritmo cardíaco, el array contiene un único valor que especifica el valor del ritmo cardíaco en ese momento.

   Al poder estar trabajando con otros sensores, tendremos que comprobar si la variación en el sensor que se ha detectado es el correspondiente al ritmo cardíaco, esto lo hacemos de la siguiente manera:

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_HEART_RATE) {

        Toast.makeText(getApplicationContext(),
                "Leyendo CV", Toast.LENGTH_SHORT).show();

        actualHR = event.values[0];

    }
}
```

   El código está simplificado para que simplemente, al detectar una variación en el ritmo cardíaco, se muestre un *Toast* anunciando tal acción y posteriormente se almacena el valor del ritmo cardíaco en una variable (*actualHR*) que es de tipo `float`.

   Y, también, por mencionar todo lo que respecta a los sensores, aunque no se haya utilizado, hay otro método que implementa la interfaz de *SensorEventListener*, que es:

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}
```

   Lo que nos permite este método es no tener que estar analizando si un valor (detectado por el otro método: `onSensorChanged()`) está por encima de un determinado valor. Este método nos ahorra esa molestia y digamos que se activa únicamente cuando el sensor que se pasa como parámetro obtiene valores que superan el umbral *accuracy* que se pasa también como parámetro.

4.5. Diseño Detallado e Implementación de la Capa de Persistencia

Ahora hablaremos de la capa responsable de definir la estructura de la BBDD a nivel lógico, así como de gestionar los mecanismos de acceso a la misma. Esta capa será también la encargada de encapsular todo el comportamiento necesario para, como su propio nombre indica, persistir los datos. La persistencia [18] de un dato es aquella propiedad de un objeto por la que su existencia trasciende el tiempo es decir, el objeto continúa existiendo después de que su creador deja de existir y/o espacio.

Para poder desarrollar esta parte se ha utilizado *Room*, una nueva librería dentro de *Android Architecture Components* la cual su objetivo es proveer de una capa de abstracción sobre *SQLite*. Teniendo el siguiente diagrama de interacción entre los distintos elementos que participarán en esta capa: base de datos, entidades y *DAO*s (*Data Access Object*):

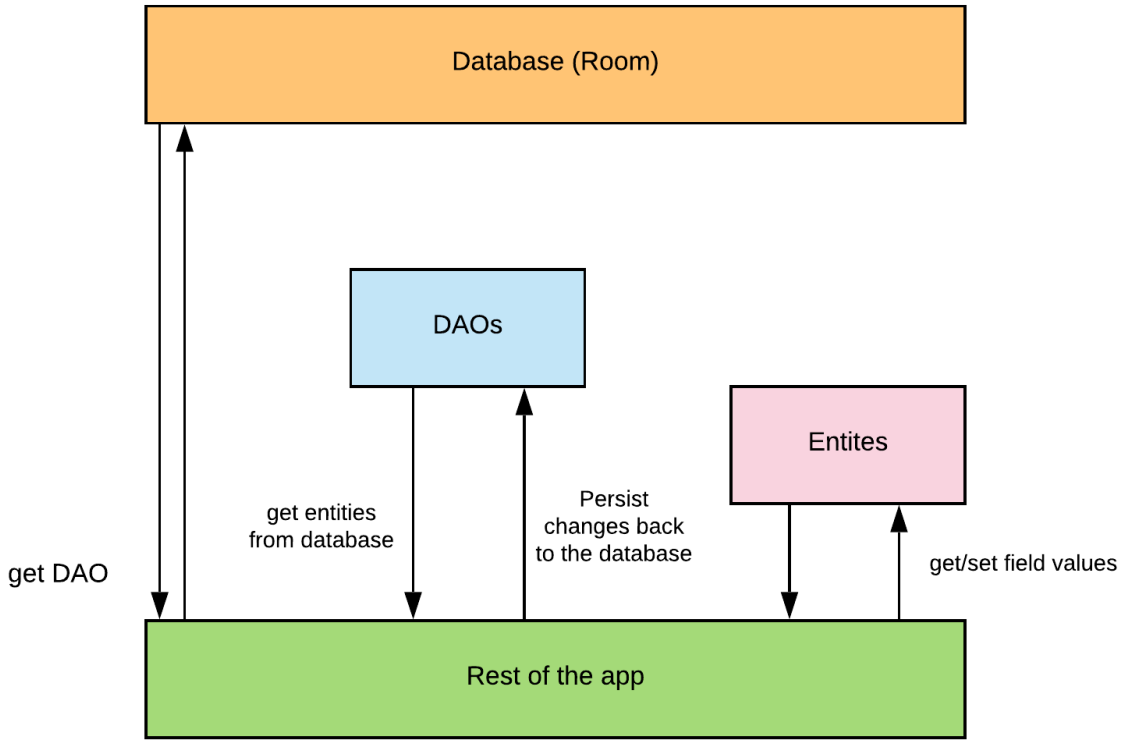

Ilustración 14: Diagrama representativo del funcionamiento del framework *Room* [17].

Como podemos observar a continuación, el conjunto de las entidades de las cuales dispondrá la aplicación no es demasiado complejo ya que, para lo que nos concierne al proyecto, haremos uso de la misma para almacenar los datos de las posibles caídas, en principio. A continuación se presenta el código de la entidad *AvisoCaida*:

```
1  @Entity(tableName = "aviso_caida")
2  public class AvisoCaida implements Serializable {
3
4      public AvisoCaida() {
5      }
6
7      @PrimaryKey(autoGenerate = true)
8      private int id;
9
10     private String idTrabajador;
11
12     private long timeStamp;
13
14     private float accelerometerX;
15
16     private float accelerometerY;
17
18     private float accelerometerZ;
19
20     private float heartRate;
21
22     public int getId() {
23         return id;
24     }
25
26     public void setId(int id) {
27         this.id = id;
28     }
29
30     public long getTimeStamp() {
31         return timeStamp;
32     }
33
34     public void setTimeStamp(long timeStamp) {
35         this.timeStamp = timeStamp;
36     }
37
38     public float getHeartRate() {
39         return heartRate;
40     }
41
42     public void setHeartRate(float heartRate) {
43         this.heartRate = heartRate;
44     }
45
46     public float getAccelerometerX() {
47         return accelerometerX;
48     }
49
50     public void setAccelerometerX(float accelerometerX) {
51         this.accelerometerX = accelerometerX;
52     }
53
54     public float getAccelerometerY() {
55         return accelerometerY;
56     }
57
58     public void setAccelerometerY(float accelerometerY) {
59         this.accelerometerY = accelerometerY;
60     }
61
62     public float getAccelerometerZ() {
63         return accelerometerZ;
```

```
64      }
65
66      public void setAccelerometerZ(float accelerometerZ) {
67          this.accelerometerZ = accelerometerZ;
68      }
69
70      public String getIdTrabajador() {
71          return idTrabajador;
72      }
73
74      public void setIdTrabajador(String idTrabajador) {
75          this.idTrabajador = idTrabajador;
76      }
77
78      @Override
79      public String toString() {
80          return "AvisoCaida{" +
81                  "id=" + id +
82                  ", idTrabajador='" + idTrabajador + '\'' +
83                  ", timeStamp=" + timeStamp +
84                  ", accelerometerX=" + accelerometerX +
85                  ", accelerometerY=" + accelerometerY +
86                  ", accelerometerZ=" + accelerometerZ +
87                  ", heartRate=" + heartRate +
88                  '}';
89      }
90  }
```

En esta capa, de lo que disponemos es de, como hemos visto, entidades pero también de aquellas clases que nos permitirán acceder a los datos que son las denominadas *DAO* y las cuales disponen de operaciones que nos permitirán realizar dichos accesos sobre la base de datos de la aplicación, como por ejemplo, a continuación se presenta la clase *AvisosCaidaDAO*, responsable de ofrecer las operaciones pertinentes de acceso a los datos de tipo *AvisoCaida*:

```
1   @Dao
2   public interface AvisoCaidasDAO {
3
4       @Insert
5       public void addAvisoCaida(AvisoCaida avisoCaida);
6
7       @Query("select * from aviso_caida")
8       public List<AvisoCaida> getAllAvisosCaida();
9
10      @Delete
11      public void removeAvisoCaida(AvisoCaida avisoCaida);
12
13  }
```

Como hemos podido observar, una DAO se trata de una interfaz que reúne los métodos de acceso de datos (inserción, eliminación, actualización, consultas...) para una entidad específica.

Todo lo anterior sería inútil sin la definición de una base de datos en la que realizar todas las operaciones que hayamos definido sobre las entidades que conforman el dominio del problema. Para implementar una base de datos mediante *Room* deberemos crear una clase anotada con *@Database* dentro de la cual describiremos en un array el conjunto de entidades que conformarán el esquema así como la versión de dicho esquema. Además, contaremos con un número de métodos abstractos igual a las DAOs que intervendrán en este proceso. Podemos observar esto en el código de la base de datos creada para dar soporte a la aplicación:

```
1 @Database(entities = {AvisoCaida.class},version = 1)
2 public abstract class AppDataBase extends RoomDatabase {
3
4     public abstract AvisoCaidasDAO avisoCaidasDAO();
5
6 }
```

5 Pruebas

En el presente capítulo se describe el conjunto de pruebas que se han realizado sobre el sistema. Las pruebas [20] intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo. Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa. En este ámbito disponemos de varios tipos de pruebas que aseguran la funcionalidad en diferentes aspectos de la aplicación, como veremos.

5.1. Pruebas Unitarias

La prueba unitarias [20] conforman el proceso de probar los componentes del programa, tales como métodos u objetos de clases. Las funciones o métodos individuales son el tipo más simple de componente. Por lo que las pruebas deben de llamar a estas funciones con una serie de diferentes parámetros de entrada para corroborar el funcionamiento, de manera atómica, de esos métodos.

En la realización de este tipo de pruebas se ha utilizado [2], herramienta que nos permite generar objetos "mock" de manera dinámica. Un mock es un objeto que simula el comportamiento de una clase específica. La idea será centrarse en programar un determinado comportamiento, ejecutar y verificar las llamadas en vez de en los resultados obtenidos por la invocación del método probado, es decir, nos centraremos en la interacción de las clases a probar con las clases de apoyo para las cuales generamos *mocks*.

Como la funcionalidad de este prototipo se realiza mediante llamadas a distintos módulos desde el módulo que llamamos director y que mantiene el algoritmo de acción, vamos a generar un *mock* del contexto de la aplicación para probar que los argumentos que se le pasan a un módulo que se llama desde el módulo que mantiene el algoritmo, anteriormente mencionado como director, los recibe correctamente:

Y, como después de que el módulo llamado desde el módulo director realice las tareas que tiene encomendadas, tendrá la misión de comunicar los resultados al módulo que le llamó (el director) por lo que ahora realizaremos la prueba inversa para ver si se obtienen los resultados correctamente en el módulo director:

```
1
2       @RunWith(AndroidJUnit4.class)
3       public class HombreSoloTest {
4
5           private int numSegundos;
6           private boolean hayCaida;
7           private float gX;
8           private float gY;
9           private float gZ;
10
11          /**
12           * Método que inicializa los atributos que vamos a utilizar en las pruebas
13           *
14           * @throws Exception
15           */
16          @Before
17          public void setUp() throws Exception {
18              numSegundos = 30;
19              hayCaida = true;
20              gX = 15;
21              gY = -10;
22              gZ = 34;
23          }
24
25          /**
26           * Método que prueba el paso de parámetros en la llamada de un módulo secundario
27           * (de la capa de soporte) desde un módulo director (capa de negocio).
28           *
29           * @throws Exception
30           */
31          @Test
32          public void pruebaParametrosDeUnModuloLanzadoDesdeElModuloDirector() throws
                Exception {
33              //  Mock del contexto de la aplicación
34              Context context = mock(Context.class);
35              //  Creamos el intent el cual lanzará un módulo secundario desde el director
                    y añadimos
36              //      los parámetros
37              Intent intent = new Intent(context, MovimientoActivity.class);
38              intent.putExtra("numSegundos", numSegundos);
39              assertNotNull(intent);
40
41              //  Obtenemos los parámetros desde el módulo secundario llamado desde el
                    móodulo director
42              Bundle extras = intent.getExtras();
43              assertNotNull(extras);
44              assertEquals(numSegundos, extras.getInt("numSegundos"));
45          }
46
47          /**
48           * Método que prueba la comunicación de resultados por parte de un módulo
                 secundario
49           * (de la capa de soporte) a un módulo director (capa de negocio).
50           *
51           * @throws Exception
52           */
53          @Test
54          public void pruebaResultadosDeUnModuloLanzadoDesdeElModuloDirector() throws
                Exception {
55              //  Mock del contexto de la aplicación
56              Context context = mock(Context.class);
57              //  Creamos el intent el cual nos retornará al módulo director desde un
                    módulo secundario
```

```
58              //      y añadimos los resultados obtenidos
59              Intent returnIntent = new Intent(context, HombreSoloActivity.class);
60              returnIntent.putExtra("hayCaida", true);
61              returnIntent.putExtra("accelerometerX", gX);
62              returnIntent.putExtra("accelerometerY", gY);
63              returnIntent.putExtra("accelerometerZ", gZ);
64              assertNotNull(returnIntent);
65
66              //  Obtenemos los resultados desde el módulo director
67              Bundle extras = returnIntent.getExtras();
68              assertNotNull(extras);
69              assertEquals(hayCaida, extras.getBoolean("hayCaida"));
70              assertEquals(gX, extras.getFloat("accelerometerX"), 0.1);
71              assertEquals(gY, extras.getFloat("accelerometerY"), 0.1);
72              assertEquals(gZ, extras.getFloat("accelerometerZ"), 0.1);
73          }
74
75      }
```

5.2. Pruebas de Integración

Los componentes de software son a menudo diferentes componentes compuestos que se forman por la unión de varios objetos que interactúan. Por lo tanto, las pruebas de integración [20] de estos componentes deben centrarse en demostrar que la interfaz del componente se comporta de acuerdo con sus especificaciones. Se asume que las pruebas unitarias en los objetos individuales dentro del componente se han desarrollado correctamente con anterioridad.

Estas pruebas se han ido realizando al finalizar cada iteración planificada del desarrollo. Al final de cada una de estas y, tras asegurarse de que esa "pieza" de software funciona de manera correcta en el ámbito unitario, se adhería al sistema, por lo que se prueba el sistema de igual manera al desarrollo, de forma incremental.

Pero podemos distinguir una serie de pruebas de integración realizadas desde un enfoque diferente, siendo este enfoque el de probar la interacción y funcionamiento entre las diferentes capas de la aplicación, siendo esto de la siguiente manera:

- Capa de Presentación con la capa de Negocio.
- Capa de Negocio con la capa de Soporte.
- Capa de Soporte con la capa de Persistencia.

Así que podemos decir que la metodología utilizada en el desarrollo nos aporta más sencillez a la hora de realizar las pruebas de integración, esto es de manera incremental.

5.3. Pruebas de Sistema

Tras finalizar la fase de pruebas unitarias y la fase de pruebas de integración realizamos las denominadas pruebas de sistema. Estas pruebas tienen como objetivo el de verificar el correcto funcionamiento del sistema en su conjunto total, centrándose en certificar el cumplimiento de los requisitos no funcionales que se hayan descrito. Tendremos varios tipos de pruebas de sistema que iremos documentando debidamente a continuación:

5.3.1. Pruebas de Mantenibilidad

Los esfuerzos de desarrollo de software dan como resultado la entrega de de un producto de software que satisface los requisitos del usuario. En consecuencia, el producto de software debe cambiar o evolucionar. Una vez en funcionamiento, se descubren los defectos, cambian los entornos operativos y aparecen nuevos requisitos de usuario. Por lo que el mantenimiento de software [15] se define como el conjunto de actividades necesarias para proporcionar un soporte rentable al software.

Para verificar esto, deberemos respetar los requisitos no funcionales RNF004 y RNF005 en los cuales el primero hace referencia a la modularidad y escalabilidad de la aplicación, esto se ha probado mediante la realización de aplicaciones independientes a la principal haciendo uso de cada módulo de la aplicación así como añadiendo la opción de seleccionar otro algoritmo en la pantalla principal de la misma permitiendo, además de arrancar el algoritmo "Hombre-Solo" podemos ejecutar un algoritmo que hace uso del módulo de detección de constantes vitales para leer y mostrar las constantes vitales del usuario, queriendo demostrar con esto la modularidad, además de que se han separado todas las operaciones distribuyéndolas en los diferentes módulos de manera coherente, con lo que aseguramos una mejor escalabilidad. Así que podemos concluir que mediante la estructuración modular de las operaciones y la distribución de las responsabilidades entre las distintas capas de la aplicación mencionadas (Véase el apartado de Diseño y Arquitectura) nos permite obtener una gran escalabilidad de la misma.

El segundo requisito no funcional al que hemos hecho referencia menciona la portabilidad de la aplicación, es decir, que la aplicación pueda ser desplegada en otros dispositivos. Esto se ha probado mediante la adquisición de 2 dispositivos totalmente diferentes (Véase en el Estudio de mercado) y se han realizado pruebas que han garantizado una portabilidad adecuada para este prototipo.

5.3.2. Pruebas de Usabilidad

La tarea principal de las pruebas de usabilidad [15] y de interacción con el ordenador humano es evaluar lo fácil que es para los usuarios finales aprender y utilizar el software. En general, puede implicar probar las funciones de software que soportan las tareas del usuario, la documentación que ayuda a los usuarios y la capacidad del sistema para recuperarse de los errores del usuario.

Haciendo referencia a esto, tenemos el requisito RNF006 en el que se menciona que la interacción con el usuario debe ser limitada a los casos indispensables en los que sea necesaria dicha interacción. Esto se ha verificado mediante el cumplimiento de dicho requisito no funcional y restringiendo la interacción del usuario con la aplicación solo en aquellos puntos del algoritmo que sea imprescindible, como la certificación de consciencia. Además, se ha probado por varias personas en el entorno de la empresa desarrolladora y aquellos usuarios no han necesitado de ayuda para entender cómo funciona la interacción con la aplicación más lejos de una breve explicación del proceso que sigue el sistema.

5.3.3. Pruebas de Rendimiento

Las pruebas de rendimiento tiene como objetivo en este caso el de comprobar cómo de rápido se ejecuta y responde el sistema además de cuánta memoria requiere para ello. Estas pruebas se relacionan con los requisitos no funcionales RNF001, RNF002 y RNF003.

Para esto se ha usado una herramienta nativa del entorno de desarrollo *Android Studio*, el *Profiler*. Las herramientas de *Android Profiler* [6] proporcionan datos en tiempo real para ayudarle a comprender cómo utiliza su aplicación los recursos de la CPU, la memoria, la red y la batería. Así que se han realizado pruebas durante la ejecución con esta herramienta y hemos obtenido los siguientes resultados:

En primer lugar, podemos observar que el `MainActivity` no hace consumir ningún tipo de recurso (en estas pruebas focalizamos el término "recursos" al uso de *CPU*, memoria y batería, ya que la aplicación no hace uso de red) al dispositivo hasta que se interacciona y entra en juego el algoritmo

seleccionado (para las pruebas nos hemos centrado en documentar el proceso *HombreSolo*) que como podemos observar, ocupa un espacio muy pequeño en el diagrama, ya que este módulo casi no realiza ningún cómputo porque ese trabajo se lo va delegando a los correspondientes módulos, en este caso, el de detección de caídas, por lo que no lo veremos aparecer demasiado en los siguientes gráficos:

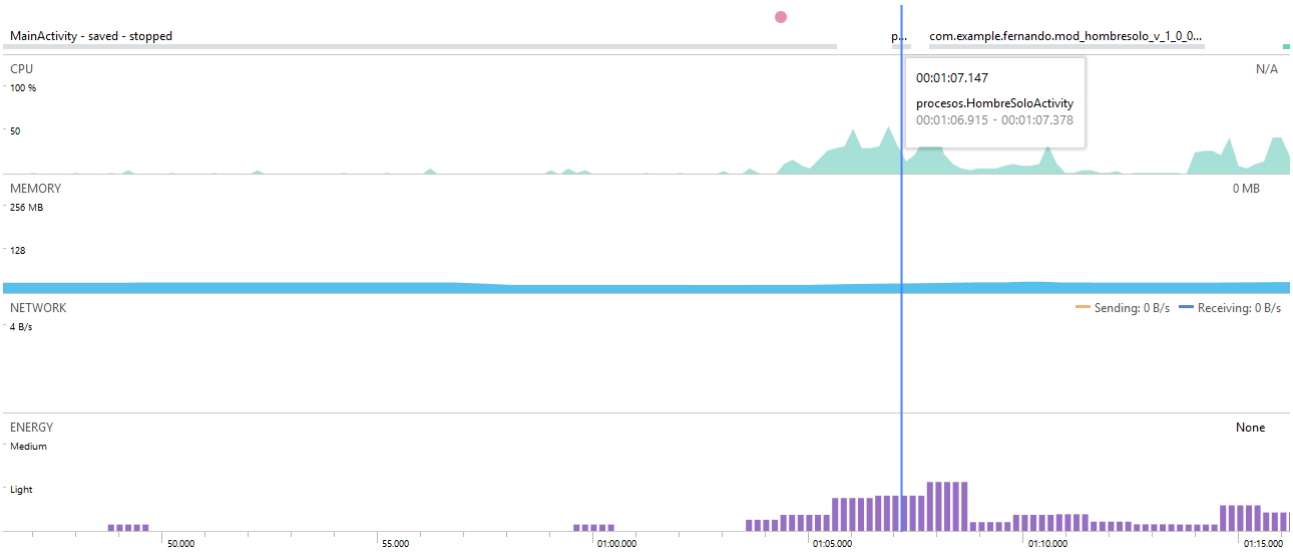

Ilustración 15: Rendimiento de la aplicación en la fase inicial de ejecución de la misma.

Así que, como decíamos, en estas primeras imágenes apreciamos que el consumo de recursos es mínimo hasta que se acciona el mecanismo de detección de caídas:

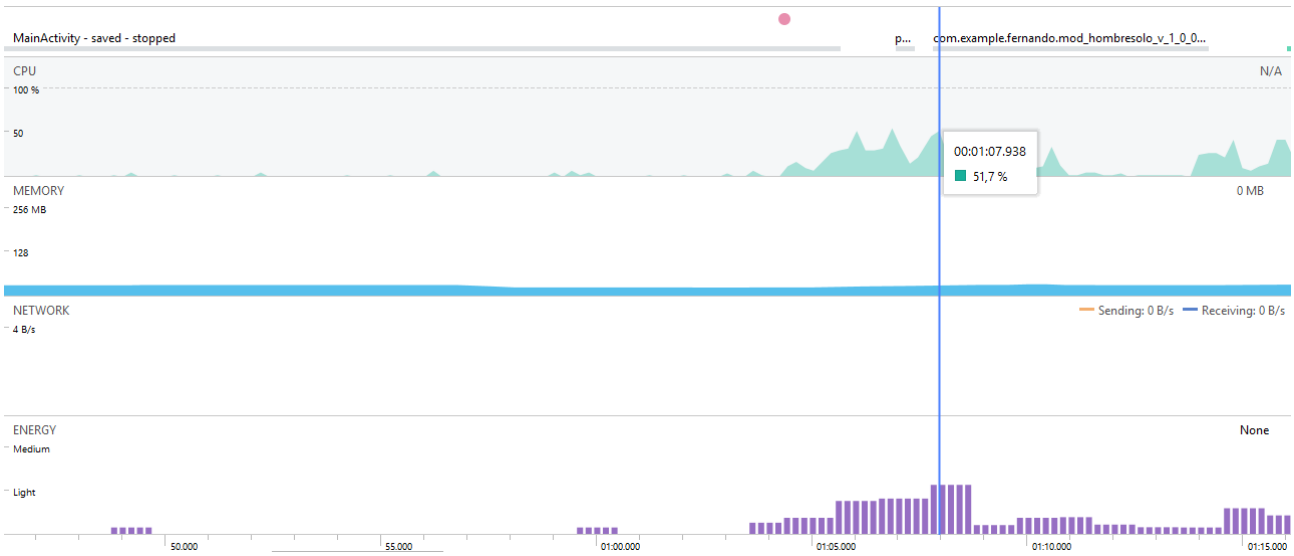

Ilustración 16: Uso de *CPU* en la fase inicial de la ejecución de la aplicación.

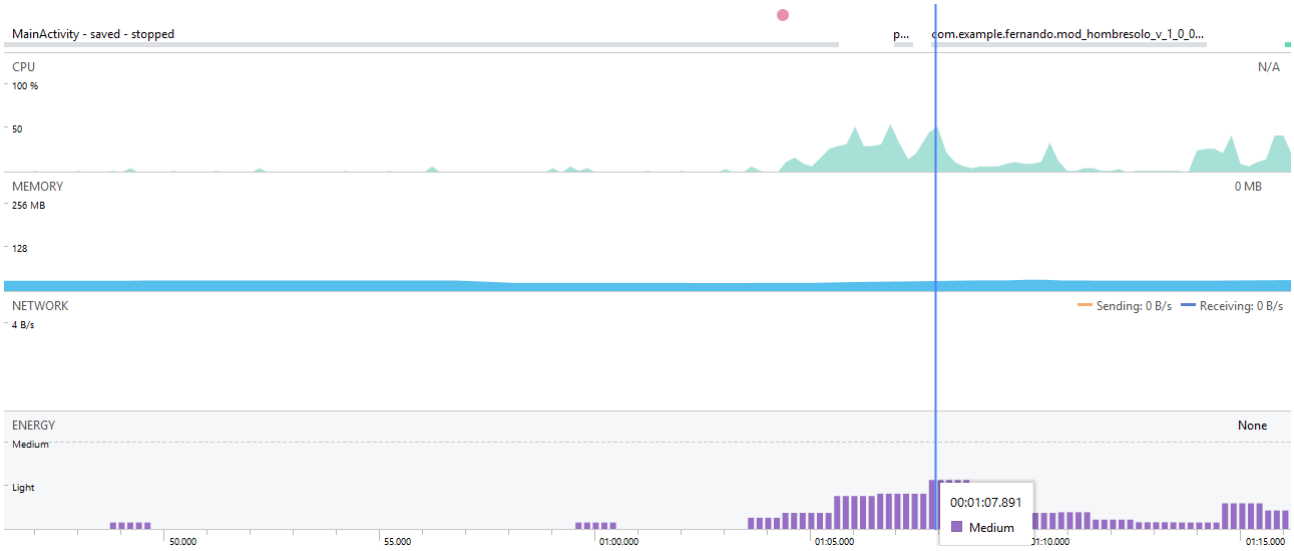

Ilustración 17: Uso de batería en la fase inicial de la ejecución de la aplicación.

Tras el módulo de detección de caídas, observamos que se vuelve a producir un intervalo de mayor consumición de los recursos. Este intervalo se corresponde con el módulo de detección de constantes vitales (que utiliza el sensor de detección del ritmo cardíaco) que ya desde el primer momento tras su ejecución, comienza a detectar el ritmo cardíaco pero luego, la aplicación deja de consumir recursos hasta que vuelve a iniciarse el siguiente módulo que es el de detección de movimientos (el cual utiliza el sensor de detección de pasos). Se incrementa el uso de estos recursos al principio ya que se inicializa el sensor y se comienza a realizar la lectura de datos:

Podemos apreciar un patrón curioso en el uso de recursos, sobre todo notable en el uso de *CPU* de la aplicación, en el cual se usan muy pocos recursos y se hace de manera intermitente durante un período grande de tiempo comparado con los anteriores. Este "patrón" se corresponde con la detección

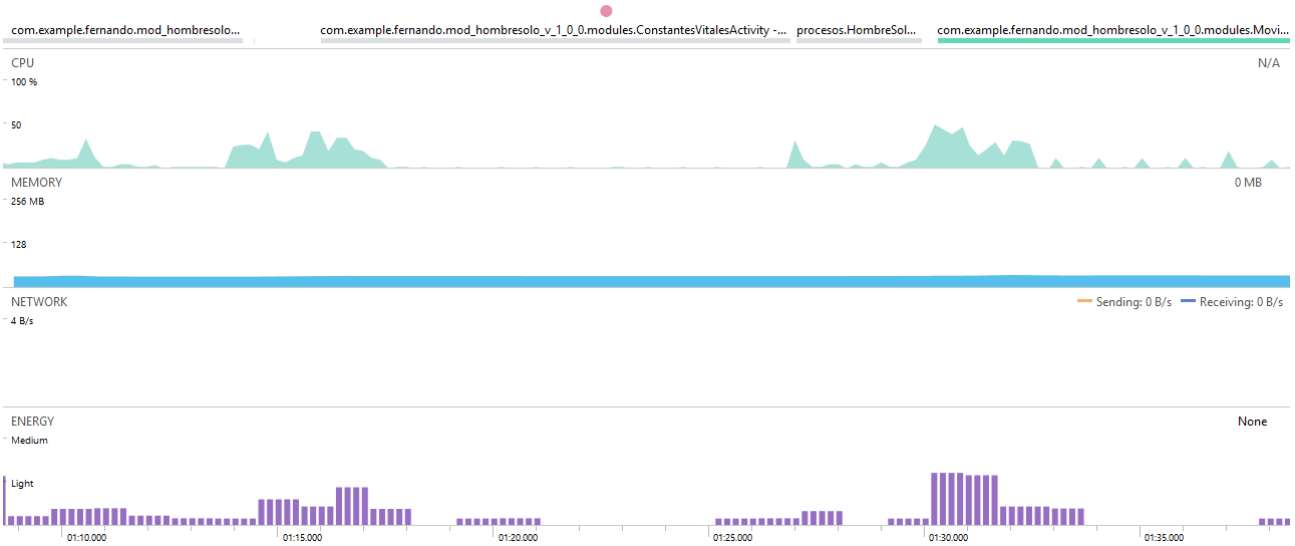

Ilustración 18: Uso recursos correspondiente a los módulos de detección de caídas, constantes vitales y movimiento.

de los pasos realizado en el módulo de movimiento en el que cada pulso de utilización de *CPU* es el muestreo de los datos solicitados:

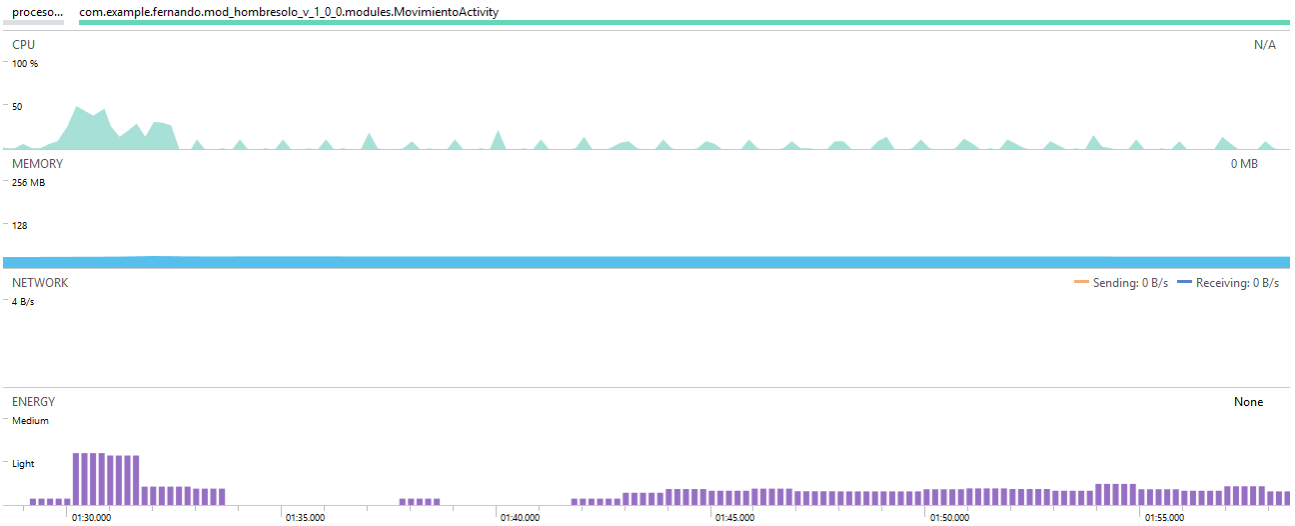

Ilustración 19: Patrón de uso de *CPU* en el módulo de detección de movimiento.

Después de realizar la detección de los pasos, en el caso de que no se haya detectado el suficiente movimiento o no se haya cancelado la la llamada de aviso de manera manual se arrancará el último módulo el cual no consumirá apenas recursos, el módulo de realización de llamadas:

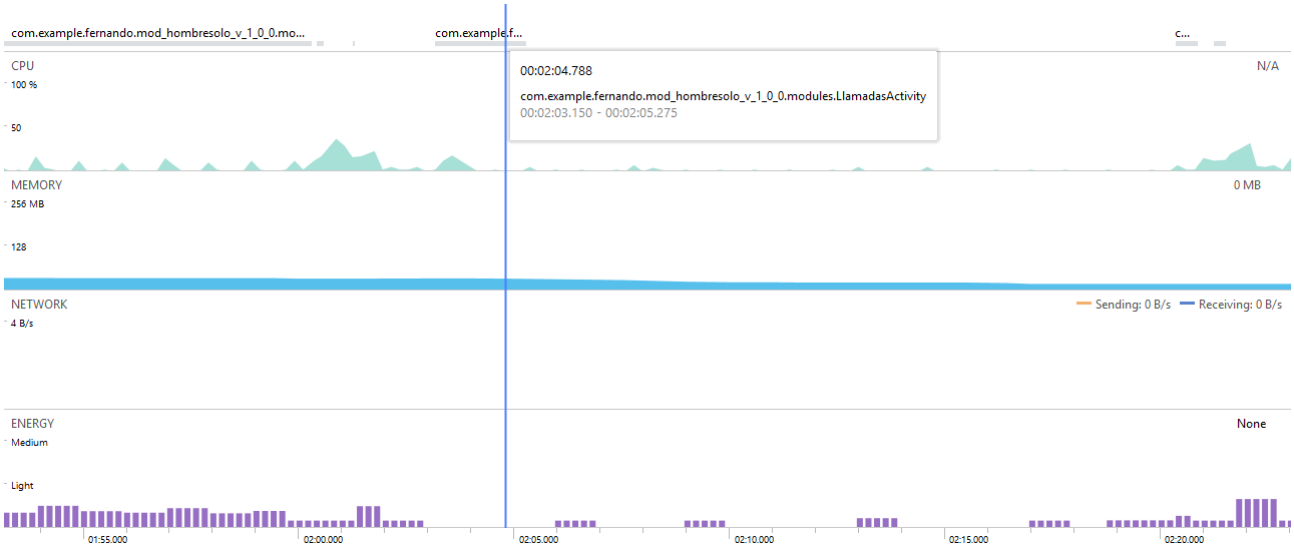

Ilustración 20: Uso de recursos en la ejecución del módulo de realización de llamadas de aviso tras no detectar un movimiento significativo por parte del usuario.

Al lanzarse la llamada de aviso, podemos ver que lo que es el módulo de llamadas en sí no se está ejecutando todo el rato durante ese período de llamada, esto es porque dicho módulo hace uso de la llamada nativa disponible en el dispositivo y como observamos no volveremos a la aplicación en sí hasta que se termina la llamada. El uso de recursos en el transcurso de la llamada es casi nulo:

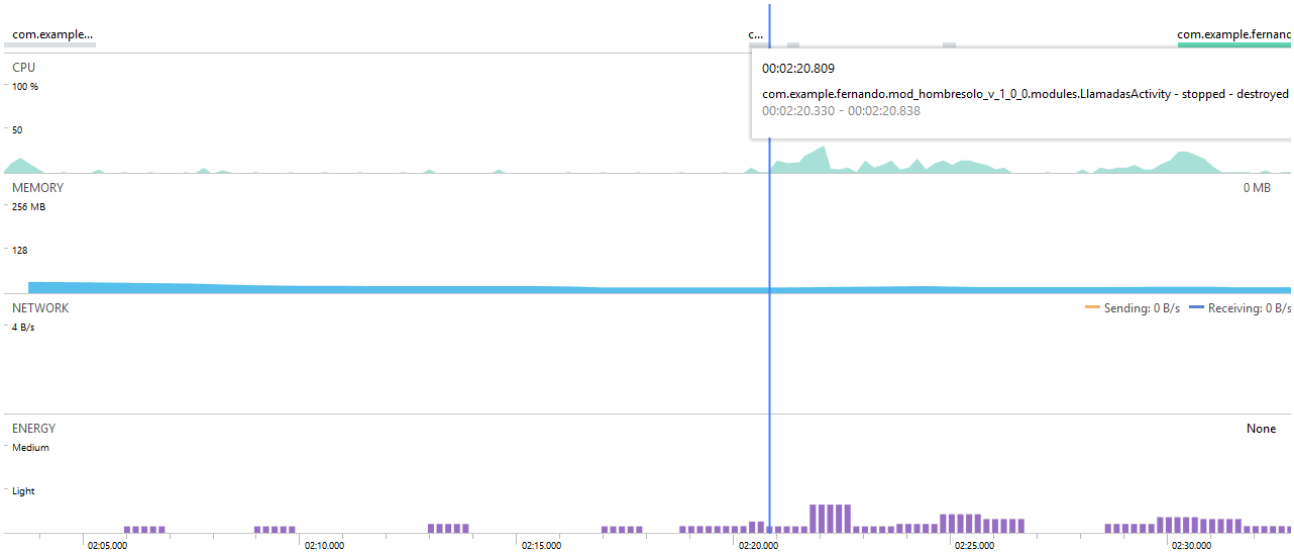

Ilustración 21: Uso de recursos en el inicio, transcurso y retorno de la realización de la llamada de aviso.

En lo respectivo al requisito RNF001, como podemos observar, se satisface correctamente tras una serie de pruebas en las cuales se comprueba lo que se tarda en detectar la caída.

El RNF002 también se complace ya que el desarrollo completo del algoritmo, desde que se detecta una caída hasta que se realiza la llamada de aviso se desarrolla entre los siguientes tiempos (hay que tener en cuenta que 30 segundos se utilizan, como hemos visto, para dar tiempo al usuario a demostrar, mediante su movimiento o cancelando la llamada de manera manual, que está consciente):

Nº Prueba	**Tiempo (segundos)**
PR001	51.95
PR002	50.53
PR003	48.48
PR004	49.27
PR005	50.80
PR006	48.96

Cuadro 3: Tiempos de respuesta de la ejecución del algoritmo *HombreSolo*.

Y, en cuanto al RNF003, con los datos sobre la batería obtenidos con el *Android Profiler* y en cuanto a las especificaciones de batería de los productos en los que se ha probado la aplicación, la aplicación permitiría la ejecución de la aplicación durante una jornada laboral completa, ya que la aplicación al estar basada en sensores, desencadena las acciones pertinentes para cada caso mediante eventos, por lo que no mantiene ninguna operación ejecutándose durante todo el tiempo a parte de la detección de determinados eventos como es un cambio drástico en los valores obtenidos por el acelerómetro.

5.4. Pruebas de Aceptación

Las pruebas de aceptación [20] son en las cuales los clientes prueban un sistema para decidir si está listo o no para ser aceptado por los desarrolladores del sistema e implementado en el entorno del propio cliente.

Al ser este prototipo un proyecto de I+D+I desarrollado de manera interna en la empresa, este tipo de pruebas se han ido realizando conforme a la implementación especificada en las iteraciones descritas en la planificación (Véase Ilustración 4) así como al final de toda la fase de desarrollo, dando por satisfechos todos los objetivos que se plantearon tanto al principio como durante el desarrollo del proyecto.

6 Conclusiones y Trabajos Futuros

En esta sección se recogen las conclusiones finales obtenidas tras la realización del proyecto. Además, se recaban aquellas características que en un futuro se podrían incorporar al prototipo para completar su funcionalidad.

6.1. Conclusiones

El proyecto al que nos hemos referido en este documento tiene como objetivo prevenir los accidentes que se producen en el entorno laboral industrial así como permitir la monitorización del estado de los operarios y una rápida respuesta en caso de que se produzca algún tipo de accidente. Para satisfacer este objetivo se ha establecido una planificación basada en la metodología de desarrollo iterativa e incremental en la que cada iteración que conforma este conjunto desarrolla las fases típicas del ciclo de vida del desarrollo software (definición de requisitos, diseño, implementación, pruebas y documentación).

En el plano personal, la experiencia de haber realizado mi proyecto de final de grado en una empresa externa ha sido completa y satisfactoria en muchos ámbitos, ya que me ha permitido aplicar conocimientos adquiridos en distintas materias del grado en un proyecto real el cual no se encontraba limitado por motivos de aprendizaje, por lo que se ha tenido que lidiar con todas las variables que han ido surgiendo obteniendo nuevos conocimientos que considero que solo se aprenden mediante una experiencia real como esta.

6.2. Trabajos Futuros

Como se ha descrito en el inicio de este documento, este proyecto consiste en la elaboración de un prototipo, por lo que contará con unas características adecuadas y acotadas para poder así hacer un estudio de venta y adecuación a las soluciones reales que necesitará cada cliente. Ya que, partiendo de las características desarrolladas, y dependiendo de la solución personalizada del cliente, estas características adicionales podrán, o no, estar en el software desplegado en el ámbito del cliente. Por lo que, a partir de una base sólida, que es lo que se pretende en el desarrollo de este prototipo, hasta el día de hoy se han encontrado varias características a añadir en un futuro:

- Incorporación de la tecnología de las redes TETRA [3] la cual representa una solución fiable y potente para comunicaciones de datos y voz de radio móvil profesional. Este estándar fue desarrollado por diferentes organizaciones (agencias públicas de seguridad, servicios de emergencias, gobiernos, compañías de transporte, de utilities y el ejército) como un modo de dar respuesta a la demanda de comunicaciones inmediatas, confiables y seguras a través del móvil.

  Por lo que, la empresa la cual desea incorporar el producto, al tener que contratar una red TETRA interna, las comunicaciones serán mucho más rápidas y eficientes, ya que ninguna red de comunicaciones interfiere con esta por lo que será una red dedicada a la seguridad de los trabajadores, incrementando la eficiencia del producto de manera significativa.

- Otra funcionalidad plausible a añadir sería la incorporación de una centralita de mensajería automática.

  Esta funcionalidad podría actuar, o no, en conjunto con la característica a añadir anterior, las redes TETRA, por lo que, en su conjunto, garantizarían un servicio completo, continuo y más eficiente, ya que estas centralitas se pueden configurar y, mediante el uso de redes restringidas TETRA, reportar el accidente de manera inmediata, así como filtrar todos los falsos positivos que puedan producirse, ya que actuaría de intermediario con los servicios de urgencia, así como que agilizaría y especializaría la respuesta ante un accidente real.

- Como se mencionó en la sección de la presentación del problema, al estar definiendo los posibles escenarios en los que la aplicación será de utilidad, tenemos el caso del operario que se encuentra en movimiento, escenario al que se ha dado solución en este proyecto, pero, con miras al futuro, en el otro extremo se encontraría aquel operario que, por ejemplo, se dispone manejando una grúa en una cabina y se desmaya, ya que dentro de la cabina no se podría caer. Ante este escenario, la aplicación no debería responder de la misma manera, ya que aquí no disponemos de un cambio frenético en la aceleración del cuerpo del trabajador, por lo que aquí tendríamos que aplicar una solución diferente así que se han barajado distintas posibles implementaciones que resuelvan este problema:

  - La primera opción es que cada cierto tiempo se disponga de un pequeño puzzle para que el operario lo resuelva en un cierto tiempo y si no, activar el resto de la lógica de la aplicación. El puzzle mencionado podría ser un patrón sencillo que se indique en la pantalla para ser seguido por el operario al cual solo le restaría unos segundos de su trabajo, como si fuese un patrón de desbloqueo de un smartphone, por ejemplo:

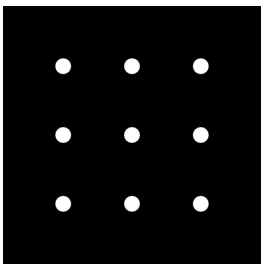

Ilustración 22: Se solicita la realización del dibujo de un patrón de desbloqueo para certificar la consciencia por parte del trabajador.

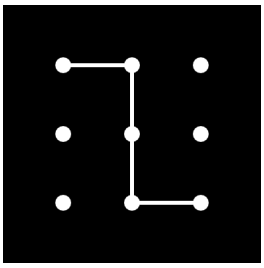

Ilustración 23: Patrón de desbloqueo dibujado para certificar la consciencia por parte del trabajador.

  - Una solución más inmediata y menos tediosa para ser realizada por el usuario (ya que no deberíamos de distraer al operario de su trabajo de manera periódica si no es por la presencia de una situación de emergencia) es aquella que aprovecha una tecnología incipiente y bien desarrollada en nuestros días, esta tecnología es la NFC [23] que permite la comunicación de corto alcance entre dispositivos compatibles. Esto requiere al menos un dispositivo transmisor y otro para recibir la señal. Una serie de dispositivos pueden utilizar el estándar

NFC y se considerarán pasivos o activos. En el caso del operario en una grúa, el dispositivo pasivo o transmisor sería una lámina con el chip incrustado la cual se establecería dentro de la cabina a una distancia accesible por el operario sin que este tenga que realizar un gran esfuerzo o dedicar demasiado tiempo y movimientos en realizar un acercamiento al dispositivo ya que el *smartwatch* actuaría de dispositivo activo o receptor y esta funcionalidad se desarrollaría de la siguiente manera:

  - Cada cierto tiempo, la aplicación notificará al operario de que debe de verificar su estado de plena consciencia, por lo que el usuario tendrá que acercar su *smartwatch* a la lámina, la cual estará en una zona accesible, como se ha dicho previamente, pero que también se situará en una zona en la cual el operario deberá estar consciente para acceder a ella (como un sitio un poco en alto para tener al menos que levantar el brazo) para evitar que el operario, sin estar en plenas facultades, certifique lo contrario al encontrarse el receptor en una demasiado accesible. Así que, manteniendo unos segundos el dispositivo sobre la lámina, se certifica la consciencia del trabajador. En el caso de que no se realice tal certificación, se emitirán avisos perceptibles por el usuario (que podría encontrarse dormido, por ejemplo) mediante vibración y sonido y, tras no reaccionar tras estos estímulos, se procederán a realizar los avisos pertinentes de emergencia que se consideren (llamada a centralita, envío de aviso...).

- Para no tener que recompilar los módulos y, en su defecto, la aplicación entera en cada uno de los *smartwaches* de los usuarios, se ha ideado que la configuración que posee cada dispositivo en lo respectivo al umbral de caída, números para llamadas telefónicas de emergencia, segundos para poder cancelar un aviso... en vez de situarse en el dispositivo, se sitúe en un servicio externo API REST para poder cambiar las configuraciones sin tener que realizar la acción de instalación en todos los dispositivos y así facilitar la labor al cliente. Con esto, tendríamos una parte de gestión web en la que podríamos visualizar todas las configuraciones de los dispositivos de cada empresa, las cuales estarían almacenadas en un servidor. De esta manera podremos configurar los dispositivos sin necesidad de hacerlo de manera atómica con cada uno, ya que podremos desarrollar configuraciones globales, distribuidas por roles, por secciones… así como la posibilidad de cambiar la configuración de un dispositivo en especial debido a algún requerimiento distinto que solicite el cliente, llevando la escalabilidad de la aplicación a un mayor nivel.
- En este prototipo, para focalizar en lo importante que era el algoritmo y el uso de diferentes módulos así como realizar una primera impresión de lo que sería el producto final, para detectar el movimiento tras una posible caída se utiliza el detector de pasos. Se había pensado en otras 2 alternativas que podrían ser complementarias en un futuro para que cada información obtenida avale a la que nos otorgan los otros métodos, estos son:
  - Mediante la triangulación de varias señales WiFi recibidas por el dispositivo, geoposicionar al mismo y así detectar variaciones en la intensidad de las diferentes señales captadas para poder comprobar si el trabajador se mueve tras esa posible caída.
  - La otra opción sería la de posicionamiento del trabajador dentro de una nave o planta industrial mediante el uso de los denominados *beacons* [1]. Un *beacon* o baliza es un pequeño transmisor de radio Bluetooth, alimentado por baterías. Estos pequeños dispositivos de hardware transmiten incesantemente señales Bluetooth de baja energía (BLE). Los teléfonos inteligentes (o, en nuestro caso, relojes inteligentes) con Bluetooth son capaces de escanear y mostrar estas señales.
    Así que, a través de esta funcionalidad, si colocamos muchos de estos *beacons* por toda la planta, podremos ir identificando la posición del usuario de una manera más exacta y precisa que mediante triangulación por GPS o WiFi mediante la detección de las señales de varios *beacons* así como la detección de estos mediante su *id* (cada *beacon* posee un *id* único), ya que estas balizas están especializadas en el geoposicionamiento *indoor* (para interiores).

Anexos

A Estudio de Mercado

Para la posibilitar el desarrollo de este proyecto software debíamos contar con un dispositivo el cual nos permitiera cumplir con la lógica de negocio especificada como objetivo. Este dispositivo se corresponde con un *smartwatch*, esto es un dispositivo que, al igual que los bien conocidos y cotidianos *smartphones*, dispone de un procesador así como diversas funcionalidades, dependiendo del fin que se le vaya a dar a dicho dispositivo.

Otra particularidad de estos dispositivos es que no están todavía tan integrados en el mercado como pueden ser otros dispositivos o *gadgets*, así que se tuvo que definir aquellas características que nuestro *smartwatch* iba a disponer. Por lo que, a continuación se exponen aquellas decisiones que se tomaron a la hora de la elección de lo que iba a ser nuestro dispositivo con el cual poder hacer posible el desarrollo de este proyecto:

- En primer lugar, nuestro *smartwatch* debe tener como sistema operativo *Android*, ya que se iba a trabajar en esta plataforma, por lo que ya podríamos descartar, por supuesto, cualquier producto de la marca *Apple* debido a su gran incompatibilidad con nuestras prácticas y tecnologías al operar en *iOS*, su propio sistema operativo. Otro descarte rápido, pero menos esperado, sería el de los dispositivos *Samsung* al trabajar con *Tizen*, un sistema operativo basado en, el gran conocido, Linux. Esto le permite gestionar sus propias interfaces, al igual que *iOS* con *Apple*, pero a nosotros lo único que nos generarían serían problemas de operabilidad. Además de estos 2 principales proveedores de dispositivos de este tipo, muchos otros se descartaron como algunos de *Xiaomi*, *Huawei*, *Garmin*... por motivos parecidos.

- La siguiente característica con la que queríamos contar en nuestro smartwatch sería la posibilidad de poder "independizarlo" de un *smartphone* mediante una nanoSIM, ya sea virtual o física, para no depender de ningún dispositivo externo y que la lógica estuviera en el propio dispositivo para que las conexiones y respuestas fueran más eficientes y rápidas a la vez que exactas al quitarnos al *smartphone* de intermediario. Esta tarjeta nanoSIM nos permitiría poseer una conexión a la red (e incluso 4G). Y aunque es verdad que en el proceso de prueba se ha trabajado con un móvil emparejado en muchas ocasiones, era por reducir el ámbito y rango de la prueba pero en el entorno final, el *smartwatch* será la única pieza "a pie de campo" y la cual representará fielmente al trabajador u operario.

- Además de la conectividad a la red que posibilita el disponer de una SIM propia en el reloj, necesitaremos otros tipos de conectividad para desarrollar otro tipo de tareas, además, cuantas más tecnologías de conectividad dispongamos en el dispositivo más disponibilidad de servicio tendrá nuestro producto con una visión de futuro, ya que, en caso de fallar un tipo, se podrá recurrir a otro, siendo más difícil la anulación del servicio y por consiguiente que aumente la posibilidad de que al ocurrir un accidente, este se obvie por falta de datos al no podernos conectar. Algunas de estas tecnologías son el WiFi, el Bluetooth y NFC. Un factor distinto al de la simple obtención de red, el WiFi, al igual que el Bluetooth nos permiten localizar, dentro de un perímetro, al dispositivo, así como detectar el movimiento, por lo que, junto a routers WiFi o *beacons* Bluetooth, podremos detectar el movimiento del dispositivo para comprobar si, tras un posible accidente, el trabajador se mueve, ya que esto nos indicará o que el trabajador

no ha sufrido ningún accidente o que ha sufrido alguna caída pero se puede mover por su propio pie entre los casos más posibles, ya que nuestro objetivo es la rápida detección de estos percances. Con diferente metodología pero con mismo objetivo tendríamos la tecnología NFC, que nos permitirá medir la consciencia del trabajador u operario habilitando zonas por las que pasar el dispositivo equipado con esta tecnología cada cierto tiempo y realizando alguna acción preestablecida que se indique en la pantalla del reloj (como por ejemplo, mantener el dispositivo sobre la base NFC un número determinado de segundos) para certificar que el trabajador está consciente y no ha sufrido ningún percance. (véase esto en la sección de Trabajos Futuros 6.2).

- Otro factor importante sería el de la batería, ya que, por muchas características que disponga el dispositivo, si no resiste un tiempo considerado como aceptable, no nos servirá de mucho, por lo que tendríamos otro factor bastante a considerar en esto.

- Además, no buscábamos un *smartwatch* que fuera bonito, sino versátil y resistente, capaz de resistir golpes y movimientos propios de un trabajador u operario en una planta o almacén así como resistente a líquidos.

- Algo más secundario que lo anterior pero a lo que podríamos echar mano para determinadas características de la lógica de negocio así como a aplicaciones adicionales mirando al futuro, sería la existencia de un micrófono así como altavoz. Esta funcionalidad permite al trabajador u operario el comunicarse a través del dispositivo sin necesidad de teléfono o *walkie-talkie*, de manera más rápida.

Así que tras tener en cuenta todos estos requerimientos y restricciones y barajar múltiples opciones escogí el **Huawei Watch 2 Pro**. Este dispositivo cumple con todo lo descrito anteriormente.

Este *smartwatch* mencionado es un dispositivo relativamente reciente, resistente a agua y polvo, con una serie de sensores que nos permitirán implementar la lógica de negocio especificada, Android Wear (que es el sistema operativo de Android dedicado exclusivamente a los dispositivos wearables) así como una batería de 420 mAh que cumple debidamente nuestros deseos en este ámbito laboral.

Dispondremos de 4Gb de memoria de la que haremos uso para la persistencia de *logs* en un cierto rango de tiempo para que, en caso de desconexión con el servidor, al reconectar, se dispongan de los datos que advierten un posible peligro de la situación del trabajador.

Este sería el dispositivo el cual utilizaría para realizar la fase inicial y principal del proyecto, pero como queríamos abarcar otro tipo de dispositivos para probar otro tipo de funcionalidades para evaluar si tendrían impacto a la hora de evitar los accidentes laborales, esta vez buscaríamos algo distinto. Esta vez queríamos que el dispositivo tuviera pantalla cuadrada así como cámara en su superficie, ya que esto nos podría abrir un mundo de posibilidades mediante el uso de la cámara. El dispositivo elegido sería el **Xiaomi Lemfo Lem 4 Pro**. Este *smartwatch* dispondría de todo lo mencionado antes menos con la tecnología *NFC*, pero en contraste tendríamos una capacidad de memoria y RAM aumentada considerablemente así como casi el triple de batería, aunque hay que considerar que el dispositivo posee una pantalla más grande por lo que consumirá más.

Referencias

[1] Beaconstac. What is a beacon?, 2018.
https://www.beaconstac.com/what-is-a-bluetooth-beacon

[2] G. J. Centeno. Uso de mock objects en pruebas con mockito, 2009.
https://www.adictosaltrabajo.com/2009/01/29/mockito/

[3] G. Ciuffo. Tetra: frecuencia segura para comunicaciones críticas, 2010.
http://www.redestelecom.es/comunicaciones/reportajes/1047849000303/
tetra-frecuencia-segura-comunicaciones-criticas.1.html

[4] O. Corporation. ¿qué es la tecnología java y para qué la necesito?, 2004.
https://www.java.com/es/download/faq/whatis_java.xml

[5] Deloitte. ¿qué es la industria 4.0?, 2018.
https://www2.deloitte.com/es/es/pages/manufacturing/articles/
que-es-la-industria-4.0.html#

[6] A. Developers. Measure app performance with android profiler, 2018.
https://developer.android.com/studio/profile/android-profiler?hl=en

[7] A. Developers. Conoce android studio, 2016.
https://developer.android.com/studio/intro

[8] A. Developers. Cómo obtener un resultado de una actividad, 2012.
https://developer.android.com/training/basics/intents/result

[9] A. Developers. Sensors overview, 2011.
https://developer.android.com/guide/topics/sensors/sensors_overview

[10] A. Developers. Room persistence library, 2017.
https://developer.android.com/topic/libraries/architecture/room

[11] Elprocus. What everybody ought to know about android : Introduction, features applications, 2013.
https://www.elprocus.com/what-is-android-introduction-features-applications/

[12] Git. About git, 2008.
https://git-scm.com/

[13] GlossaryTech. Sourcetree.
https://glossarytech.com/terms/tools/sourcetree

[14] A. Hathibelagal. Sensores de android en profundidad: Proximidad y giroscopio, 2017.
https://code.tutsplus.com/es/tutorials/android-sensors-in-depth-proximity-and-gyroscope--cms

[15] IEEE. *Swebook v3.0*. IEEE, 2014.

[16] J. V. Javier Cuello. Interacción y patrones, 2013.
http://appdesignbook.com/es/contenidos/patrones-interaccion-moviles/

[17] F. M. L. Jurado. Room, otra forma de crear bases de datos en android androidmeetskotlin, 2018.
https://betabeers.com/blog/room-otra-forma-crear-bases-datos-android-androidmeetskotlin-

[18] D. Lara. Concurrencia y persistencia en programación orientada a objetos, 2015.
https://styde.net/concurrencia-y-persistencia-en-programacion-orientada-a-objetos/

[19] m. y. s. s. Ministerio de Trabajo. Estadística de accidentes de trabajo, 2017.
http://www.mitramiss.gob.es/estadisticas/eat/eat17/Resumen_resultados_ATR_2017.pdf

[20] I. Sommerville. *Ingeniería del software*. Pearson educación, 2011 Novena edición.

[21] SQLite. About sqlite, 2007.
https://www.sqlite.org/about.html

[22] Techopedia. Iterative and incremental development, 2012.
https://www.techopedia.com/definition/25895/iterative-and-incremental-development

[23] R. Triggs. What is nfc how does it work?, 2018.
https://www.androidauthority.com/what-is-nfc-270730/