Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA DE CONTROL, AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MÁSTER

DISEÑO Y DESARROLLO DE UNA APLICACIÓN EN ANDROID CON REQUISITOS DE TIEMPO REAL FLEXIBLE PARA LA CAPTURA DE DATOS INALÁMBRICA DE UNA CONTERA INTELIGENTE

Estudiante *Lapuente de la Peña, Carlos*
Director/Directora *Portillo Pérez, Eva*
Departamento *Ingeniería de Sistemas y Automática*
Curso académico *2021-2022*

Bilbao, septiembre 2022

Curso académico
2021/2022

Resumen: Se ha llevado a cabo el desarrollo de una aplicación para dispositivos Android cuya principal función es la adquisición con requisitos temporales enviados por una Contera Inteligente (CI). La conexión entre el dispositivo Android en el que se instale la aplicación y la CI se realiza mediante Bluetooth Low Energy. La CI está diseñada para ser colocada en la muleta empleada por personas con Esclerosis Múltiple (EM). Los datos adquiridos son indicativos del estado de marcha de la persona y su objetivo es proporcionar información sobre la persona usuaria de la CI. Este trabajo se engloba dentro del proyecto SMARTIP del grupo de investigación ViSens.

Laburpena: Android-gailuetarako aplikazio bat garatu da. Aplikazio horren funtzio nagusia da datuak Contera Adimentsu batek bidalitako denbora-baldintzak betetzea. Aplikazioa instalatzen den Android-en gailuaren eta Contera Adimentsuaren arteko konexioa Bluetooth Low Energy bidez egiten da. Esklerosi anizkoitza duten pertsonek erabiltzen duten makuluan jartzeko diseinatuta dago Contera Adimentsua. Eskuratutako datuek pertsonaren egoera adierazten dute, eta haren helburua da Contera Adimentsua erabiltzen duen pertsonari buruzko informazioa ematea. ViSens ikerketa-taldearen SMARTIP proiektuaren barruan sartzen da lan hau.

Summary: The development of an application for Android devices has been carried out whose main function is the acquisition of data with temporary requirements sent by a Smart Tip. The connection between the Android device on which the application is installed and the Smart Tip is made via Bluetooth Low Energy. The Smart Tip is designed to be placed on the crutch used by people with Multiple Sclerosis. The data acquired is indicative of the person's walking status and its objective is to provide information about the user of the Smart Tip. This work is included within the SMARTIP project of the ViSens research group.

Palabras clave: aplicación, Android, Contera Inteligente, Bluetooth Low Energy, tiempo real, Esclerosis Múltiple

Hitz gakoak: aplikazioa, Android, Contera Adimentsu, Bluetooth Low Energy, denbora erreala, Esklerosi Anizkoitza

Keywords: application, Android, Smart Tip, Bluetooth Low Energy, real time, Multiple Sclerosis

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

ABREVIATURAS

EM Esclerosis múltiple

CI Contera inteligente

BLE Bluetooth Low Energy

SO Sistema operativo

TR Tiempo real

1 INTRODUCCIÓN

La esclerosis múltiple (EM) es una enfermedad neurológica inflamatoria y autoinmunitaria que afecta a más de 50.000 personas en España y a alrededor 2.8 millones en el mundo [1]. Se caracteriza por la destrucción de la mielina (estructura que envuelve a los axones para facilitar la transmisión de señales nerviosas), lo que provoca daños neurológicos que varían en sus síntomas. La causa de la enfermedad todavía se desconoce, aunque se han propuesto diversas hipótesis como la posibilidad de ser originada por la infección del virus Epstein-Barr [2]. Hasta la fecha no se conoce ninguna cura a la enfermedad, siendo todos los tratamientos paliativos.

El 80% de los pacientes de EM son del tipo conocido como remitente-recurrente, el cual se caracteriza por ataques imprevisibles o brotes. Los brotes se producen a causa de la formación de una nueva lesión desmielinizante. Como consecuencia de un brote, pueden quedar secuelas en forma de dificultad a la hora de realizar ciertas tareas. De hecho, un elevado porcentaje de las personas afectadas requieren de ayuda técnica, como son las muletas y los bastones [3].

Por este motivo, es importante trabajar para mitigar estas secuelas y mejorar su calidad de vida. En este sentido, la predicción de brotes permitiría definir tratamientos más adecuados y personalizados de manera precoz [4]. La dificultad de la previsión de dichos brotes radica en que la evolución de la enfermedad no sigue patrones similares, ni siquiera entre pacientes con síntomas parecidos.

En este aspecto, el nivel de actividad física diaria que cada paciente es capaz de realizar constituye una importante fuente de información para el seguimiento de la enfermedad y adaptación individualizada de las terapias [5]. Es por ello que, dentro de las soluciones propuestas para el diagnóstico del estado del paciente EM, destacan aquellas basadas en dispositivos pasivos para el análisis de la marcha [5]. De hecho, existen test específicos, como los test temporizados de marcha, que evalúan el tiempo y rendimiento de la marcha en un espacio acotado [6]. Sin embargo, estos test solo permiten la valoración del paciente en los momentos puntuales en los que se realizan y, al no realizarse una monitorización continua del paciente, la capacidad para anticipar brotes se ve limitada de manera significativa.

Por tanto, sería deseable poder disponer de soluciones asequibles que permitan monitorizar el estado del paciente de manera continua y no invasiva a partir de variables fácilmente medibles.

En este contexto surge el grupo de investigación Sensorización Virtual para Bioingeniería (ViSens), en el que se inició el proyecto titulado Contera inteligente para el diagnóstico funcional de la marcha en pacientes con Esclerosis Múltiple (SMARTIP). En este proyecto se ha llevado a cabo el diseño y desarrollo de una Contera Inteligente (CI), un dispositivo adaptable a bastones y muletas que permite capturar las variables básicas necesarias para la

caracterización de la marcha y del equilibrio de pacientes EM que usan dispositivos de ayuda en el equilibrio. Para ello, la CI cuenta con un sensor de fuerza, una IMU y un barómetro. Los datos son transmitidos por la CI a través de la tecnología Bluetooth Low Energy (BLE).

Actualmente, la adquisición de los datos de la CI se realiza con un programa escrito en C que se ejecuta en un ordenador. Realizarla con una aplicación Android ejecutada en un móvil o *tablet* resultaría ventajoso a la hora de realizar las tomas de datos de las pruebas, ya que son dispositivos más reducidos y ligeros; y Android es el Sistema Operativo (SO) de dispositivos móviles con más personas usuarias a nivel mundial. De esta forma, se abrirían las puertas a la monitorización continua del estado de salud de las personas con EM.

Por tanto, el objetivo principal del presente trabajo fin de máster es diseñar y desarrollar una aplicación Android que permita la comunicación con la CI para adquirir los datos enviados por la misma con una tasa de errores de recepción mínima.

En cuanto a la estructura del documento, se divide en 5 apartados.

En el primer apartado se detallan los objetivos y el alcance del proyecto.

A continuación, se realiza un análisis del estado del arte. Para ello, se estudian los dispositivos disponibles para la ayuda a personas con EM. También se contextualiza el estado actual del Sistema Operativo Android, haciendo un estudio de las posibilidades de tiempo real (TR) disponibles para Android.

Posteriormente, se describe el diseño de la solución. Por un lado, se explica la parte hardware (HW), compuesta por la CI, el móvil y el BLE. Por otro lado, se describe el diseño y desarrollo de la parte software (SW) compuesta fundamentalmente por la aplicación desarrollada en este trabajo.

Después, se describen los resultados de las pruebas realizadas para la validación de la aplicación.

Finalmente, se exponen las conclusiones y acciones futuras.

Adicionalmente, se incluye como anexo el código original completo de la aplicación desarrollado íntegramente en el marco de este trabajo de fin de máster.

2 OBJETIVOS Y ALCANCE

La adquisición de datos mediante la CI disponible actualmente requiere de un portátil, impidiendo la monitorización continua del paciente en su vida diaria fuera de un entorno clínico. El desarrollo de una aplicación móvil simplificaría esta tarea, abriendo posibilidades a una monitorización continua. Este desarrollo debe realizarse para una plataforma concreta debido a las diferencias entre los SO, como son iOS y Android. La elección de Android como SO sobre el que desarrollar la aplicación permite que sea accesible a un mayor número de usuarios y usuarias debido a su mayor cuota de mercado dentro de los SO para móviles [7].Esta aplicación debe contar con requisitos temporales para garantizar una recepción de datos con un porcentaje de errores aceptable. Esto se debe realizar sin interferir en la configuración del SO ni realizar cambios en su estructura fundamental para garantizar y favorecer el uso de esta nueva aplicación por parte de las y los usuarios potenciales.

2.1 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación para dispositivos Android que sea capaz de comunicar el móvil en el que se esté ejecutando con la CI mediante BLE. Mediante esta comunicación, la aplicación debe ser capaz de recibir los datos enviados por la CI con una tasa de errores que sea lo más cercana al 0% como sea posible. De esta forma, se abriría la puerta a una monitorización continua del estado de las personas usuarias de muletas.

Además, debe poder ser usada en móviles personales sin que impida el correcto funcionamiento del mismo y sin que los errores en la adquisición de datos aumenten. Esto es, el resto de aplicaciones en ejecución no deben ralentizarse cuando la aplicación esté en funcionamiento, y la aplicación no debe ralentizarse cuando haya otras aplicaciones en ejecución. El uso de la aplicación también tiene que ser posible sin modificar el SO del móvil para que sea accesible al público general.

Entre los objetivos secundarios, destaca que los datos se deben almacenar de forma local en el móvil, deben estar accesibles para la persona usuaria y deben poder exportarse para realizar los análisis de datos correspondientes. Todo esto debe ir acompañado por una interfaz amigable e intuitiva.

2.2 Alcance

En lo que respecta a los aspectos de TR, dadas las características de las plataformas móviles, a priori no se pretende que la aplicación sea de TR estricto (*hard real time*), sino de TR flexible (*soft real time*). Esto significa que la pérdida de plazo no conlleva consecuencias graves o, en

otras palabras y en el contexto específico del presente trabajo fin de master, que la pérdida de datos es asumible siempre que sea cercana al 0%.

En lo que respecta al SO móvil, tal y como se ha justificado en el apartado anterior, la aplicación solo estará diseñada para dispositivos Android. Por tanto, no es objetivo de este trabajo desarrollar la aplicación para iOS si bien el diseño y la metodología resultado de este trabajo serían aplicables a los dispositivos iOS.

Por último, la versión mínima de Android para poder ejecutar la aplicación es 8.0. Esto se debe al uso de la librería POI de la Fundación Apache que es utilizada para guardar los datos obtenidos en un archivo Excel. Esta librería requiere de una versión Android 8.0 o superior para funcionar. Esta versión de Android fue lanzada a mediados de 2017 y la versión más reciente de Android es la 13.0, lanzada en agosto de 2022. Según las estimaciones de Android accesibles a través de su software oficial para la creación de aplicaciones (Android Studio) el 88.2% de los dispositivos Android tienen una versión igual o superior a la 8.0, como se puede observar en la Figura 1.

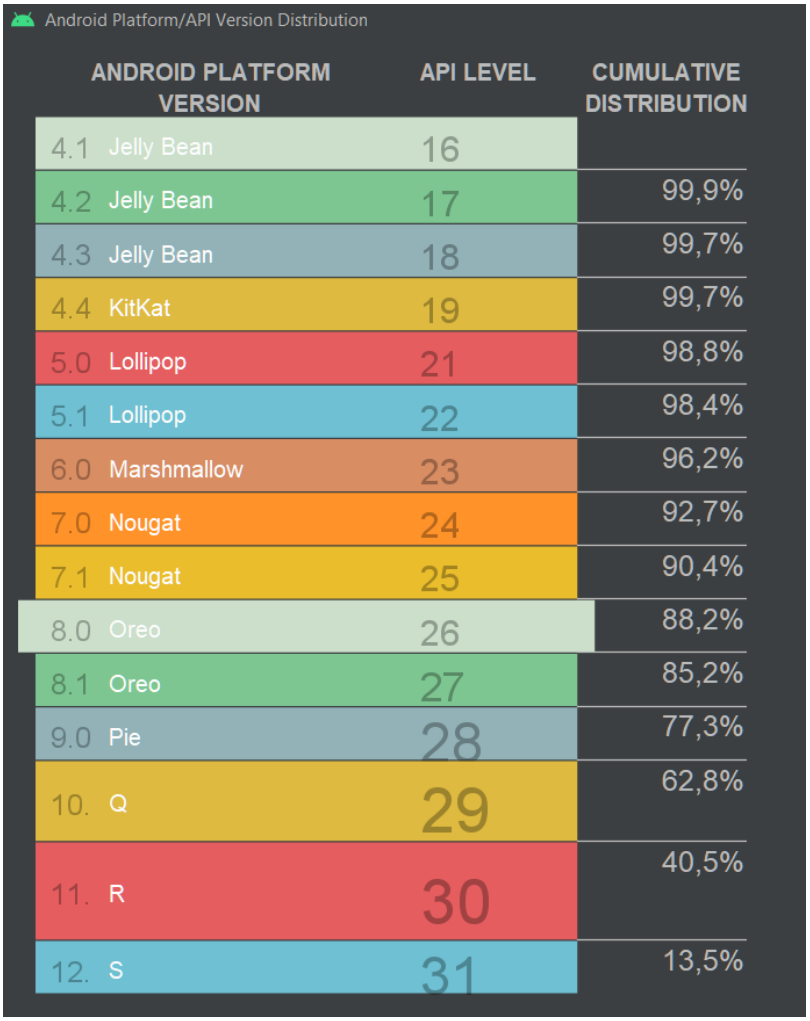

Figura 1. Porcentajes acumulativos de las versiones de Android

3 ESTADO DEL ARTE

La reducción del tamaño y coste de dispositivos electrónicos ha posibilitado la paulatina integración de los mismos en cada vez más aspectos del día a día de las personas. Esta tendencia no solo ha afectado al ocio sino también a sectores profesionales como la medicina. Los *smartphones* son un claro ejemplo de esta tendencia, apareciendo cada año en el mercado dispositivos con mejores prestaciones a precios más asequibles o similares a los de dispositivos anteriores, que eran más limitados en cuanto a funcionalidades. Dentro de este ámbito, Android se ha consolidado como el SO con mayor cuota del mercado, siendo utilizado por alrededor del 70% de los dispositivos del mundo [7] (Figura 2).

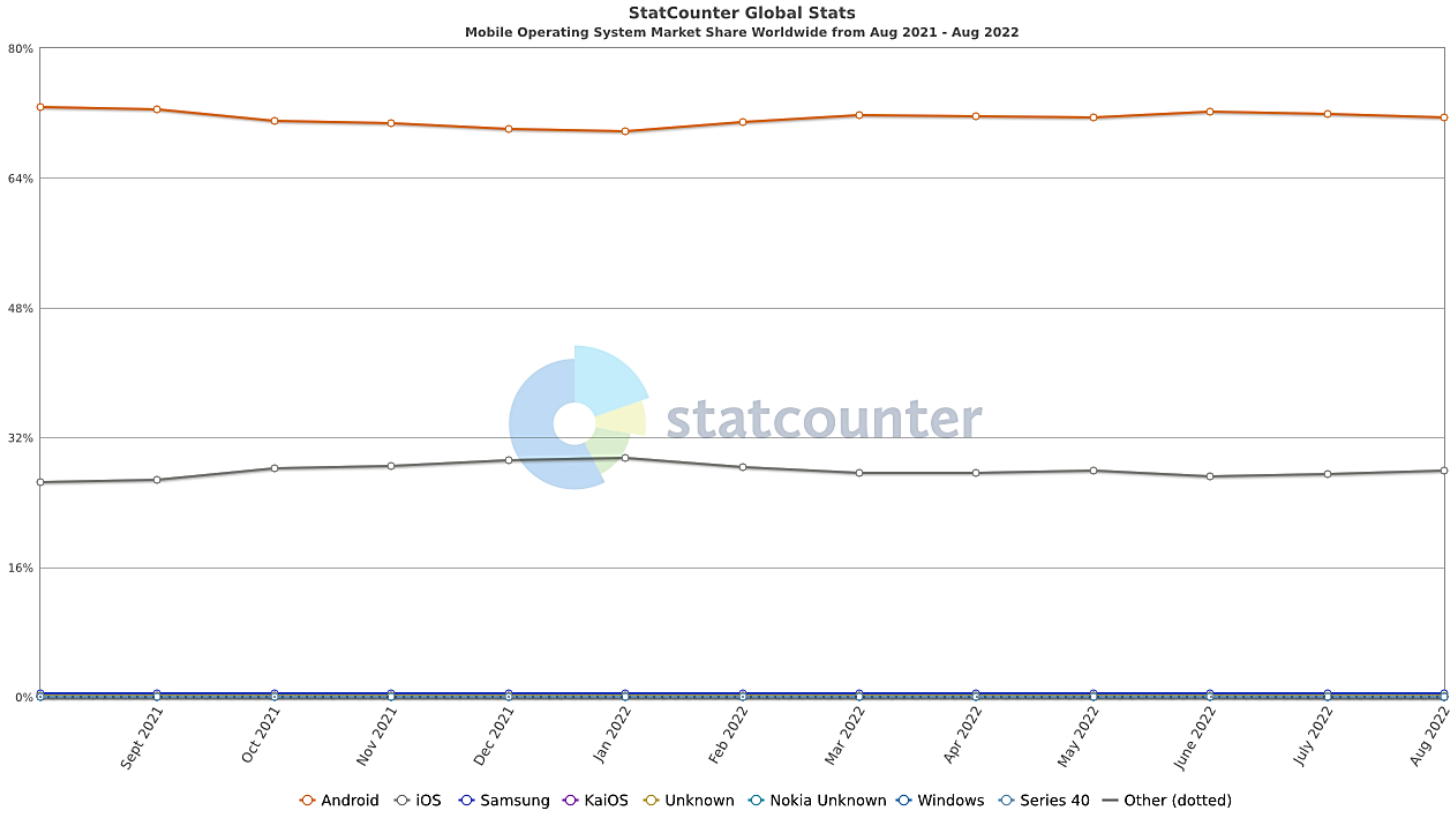

Figura 2. Gráfica del uso mundial de distintos SO móviles [7]

3.1 Soluciones tecnológicas para el diagnóstico y seguimiento de la EM

La incorporación de soluciones tecnológicas para el diagnóstico y el seguimiento de la EM es un área abierta y poco explorada en el campo de la investigación. Sin embargo, a lo largo de los últimos años se han propuesto diferentes alternativas para poder conseguir herramientas de monitorización que permitan seguir una evolución del estado del paciente. Actualmente, estas propuestas se pueden clasificar en dos grandes grupos en función de su capacidad de interacción con el paciente: los dispositivos activos y los pasivos. Por un lado, se entiende por activos aquellos dispositivos que realizan movimientos junto al paciente, ayudando a la

rehabilitación o al movimiento de miembros de forma directa y pudiendo contener también elementos que recojan información. Por otro lado, los dispositivos pasivos son aquellos que únicamente se encargan de recoger información.

3.1.1 Dispositivos activos

En este primer grupo se sitúan los robots de rehabilitación, los cuales están diseñados para asistir en la ejecución de las terapias [8]. En estos dispositivos robóticos se engloban tanto exoesqueletos completos como robots destinados a la rehabilitación de extremidades concretas del cuerpo humano.

Los robots van variando según su tamaño y complejidad en el diseño. Disponen de diferentes sensores integrados que permiten evaluar tanto el movimiento como la fuerza que tiene que ejercer el paciente con el miembro que se esté recuperando. Es muy común utilizar esta clase de robots para la recuperación de la movilidad de extremidades, superiores o inferiores, tras haber sufrido enfermedades cerebrales como un ictus.

Con el fin de cuantificar el estado del paciente, los datos recogidos por los sensores del robot se deben procesar correctamente. Para ello, normalmente se suelen transformar en una serie de indicadores (rango de movimiento, velocidad, ratios de fuerza) o escalas y se muestran en pantallas de monitorización, ofreciendo una cuantificación de un determinado parámetro fisiológico. En la Figura 3 se puede observar un ejemplo de estos dispositivos.

Figura 3. Robot de rehabilitación desarrollado por ViSens en colaboración con Tecnalia

Sin embargo, esta clase de dispositivos activos presentan una serie de inconvenientes. El primero es la falta de estandarización y la gran variedad de indicadores posibles que se pueden

sacar a partir de los datos disponibles. El segundo es que pocas de las propuestas presentadas hasta la fecha han sido correlacionadas con escalas clínicas normalizadas. Y, por último, la mayoría de los robots propuestos están destinados a su uso en clínicas, donde las condiciones están controladas y, por tanto, son inviables para monitorizar al paciente en su vida diaria [9].

3.1.2 Dispositivos pasivos

La variedad de las soluciones tecnológicas existentes en este segundo grupo es más amplia que en los dispositivos activos. Sin embargo, la gran diferencia con el grupo anterior es que la mayoría de las propuestas de dispositivos pasivos están enfocadas al estudio del patrón de marcha del paciente, debido a que diferentes estudios [5] han demostrado que existe una gran correlación entre el estado o avance de la enfermedad de un paciente de EM y la pérdida de autonomía motora. Con este fin se pueden encontrar los siguientes tipos de dispositivos pasivos.

3.1.2.1 Wearable sensors

Es una de las soluciones más adoptadas dentro de los dispositivos pasivos [10]. Los dispositivos más habituales dentro de este grupo son los sensores inerciales, como pueden ser los acelerómetros, giróscopos y magnetómetros. La idea de implementar este tipo de estrategia fue concebida una vez que la tecnología permitió obtener sensores inerciales ligeros y de pequeño tamaño, de tal forma que fueran una opción práctica e interesante para poder medir el movimiento de una persona. Entre los dispositivos más habituales destacan las IMUs (*Inertial Measurement Unit*), las cuales llevan encapsulados los diferentes sensores inerciales en un mismo elemento.

Los sensores inerciales son capaces de aportar información de las aceleraciones y las velocidades del punto en el que se encuentran situados, así como de las rotaciones y la orientación. De esta forma, se consigue conocer el movimiento que realiza la persona usuaria en todo momento. Actualmente, la ubicación de estos sensores es objeto de estudio, pudiéndose colocar en diferentes partes del cuerpo como pueden ser la rodilla o la cadera.

Dos de las grandes ventajas del uso de estos dispositivos son que permiten una monitorización a lo largo de todo el día, es decir, no es una monitorización que ocurre exclusivamente en la clínica como con las resonancias o los robots de rehabilitación, sino que al llevar el paciente los sensores en su cuerpo se puede hacer un seguimiento diario y continuo de su patrón de marcha; y que la monitorización es de bajo coste. Sin embargo, también presentan una serie de inconvenientes. El primero es que el hecho de llevar los sensores en el cuerpo puede dificultar el movimiento de la persona que los use y pueden llegar a ser considerados invasivos. El segundo inconveniente es que estudios recientes [11] han demostrado que para enfermedades como la EM esta clase de sensores sólo ofrecen datos para cuantificar la movilidad del paciente, pero no la actividad física de los mismos y, como ya se ha comentado

anteriormente, tener conocimiento sobre la actividad física es vital para poder optimizar la terapia y reducir la fatiga.

3.1.2.2 Smartphones

En los últimos años otros autores [12] han propuesto como alternativa a los *wearable sensors* utilizar los móviles como elemento sensor, ya que éstos llevan incorporados en la mayoría de los casos sensores inerciales. El uso de los móviles para cuantificar el nivel de actividad de una persona no es algo innovador actualmente, ya que desde hace años existen aplicaciones que permiten dicha cuantificación mediante los sensores de movimiento y localización de los propios móviles.

Esta alternativa presenta dos grandes ventajas: al igual que los *wearable sensors*, permite hacer una cuantificación diaria del movimiento, ya que en la actualidad casi todas las personas llevan consigo el móvil a lo largo de todo el día. Además, al no usar sensores en el propio cuerpo del paciente, es una alternativa considerada menos invasiva. Por el contrario, el procesado de los datos obtenidos de los sensores internos es más complejo que con los *wearable sensors*, ya que el móvil no está fijo en una posición, sino que se lleva en lugares como el pantalón y, por tanto, pueden existir movimientos parásitos, que si no se filtran adecuadamente puedan falsear los datos obtenidos.

3.1.2.3 Dispositivos de asistencia inteligentes

La naturaleza degenerativa de la EM hace que la autonomía de los pacientes, sobre todo en temas de movilidad, se vea afectada en gran medida. Se ha estimado que, tras 15 años desde el primer brote de la enfermedad, más del 80% de los pacientes requerirá de una ayuda técnica (bastón, muletas, etc.) para mantener su autonomía. Por esta razón, el desarrollo de dispositivos de asistencia inteligente ha constituido otra área de gran interés para la cuantificación del estado del paciente.

En este campo se han presentado diferentes propuestas de muletas inteligentes con diferentes sensores como IMUs, inclinómetros, sensores de presión o fuerza, etc. Algunas de ellas se han validado, siendo muy prometedores los primeros resultados obtenidos y demostrando que es posible estimar parámetros con una precisión elevada, ya que son equiparables a los que se obtienen con plataformas de fuerza fija de alto coste. Sin embargo, el principal problema es que la mayoría de los trabajos presentados se encuentran muy enfocados en la caracterización de la carga que el paciente aplica a la muleta, existiendo aún un amplio campo por investigar en áreas de cuantificación del movimiento, monitorización del nivel de actividad física y caracterización funcional del paciente.

Tal y como se ha comentado anteriormente, el presente Trabajo Fin de Máster se centra en este tipo de dispositivos, de manera que se pretende incrementar su usabilidad para pacientes de EM mediante el diseño y desarrollo de una aplicación móvil que capture en tiempo real las

distintas variables de los sensores y así poder caracterizar y monitorizar su nivel de actividad física.

3.2 Arquitectura Android

La aplicación, como se ha comentado, se va a desarrollar en Android ya que es el sistema operativo con mayor cuota del mercado, siendo utilizado en alrededor del 70% de los móviles en uso en el mundo [7]. Así, de cara a poder realizar un correcto diseño y desarrollo de la aplicación objetivo, en este apartado se analizan las características principales de la arquitectura Android, así como los recursos y mecanismos que proporciona.

3.2.1 Arquitectura

Android es una plataforma de código abierto basada en Linux creada para una amplia gama de dispositivos que incluye un sistema operativo, middleware y aplicaciones. Se desarrolla principalmente bajo licencia Apache 2.0, aunque hay determinadas partes que son distribuidas con otros tipos de licencias, por ejemplo, algunos parches del kernel tienen licencia GPLv2. Este tipo de licencias de software libre permite a las y los desarrolladores y a la industria aprovechar las características ofrecidas por Android, aplicar parches desarrollados por terceros y realizar modificaciones [13].

La Figura 4 muestra los componentes de la plataforma Android, que se detallan a continuación.

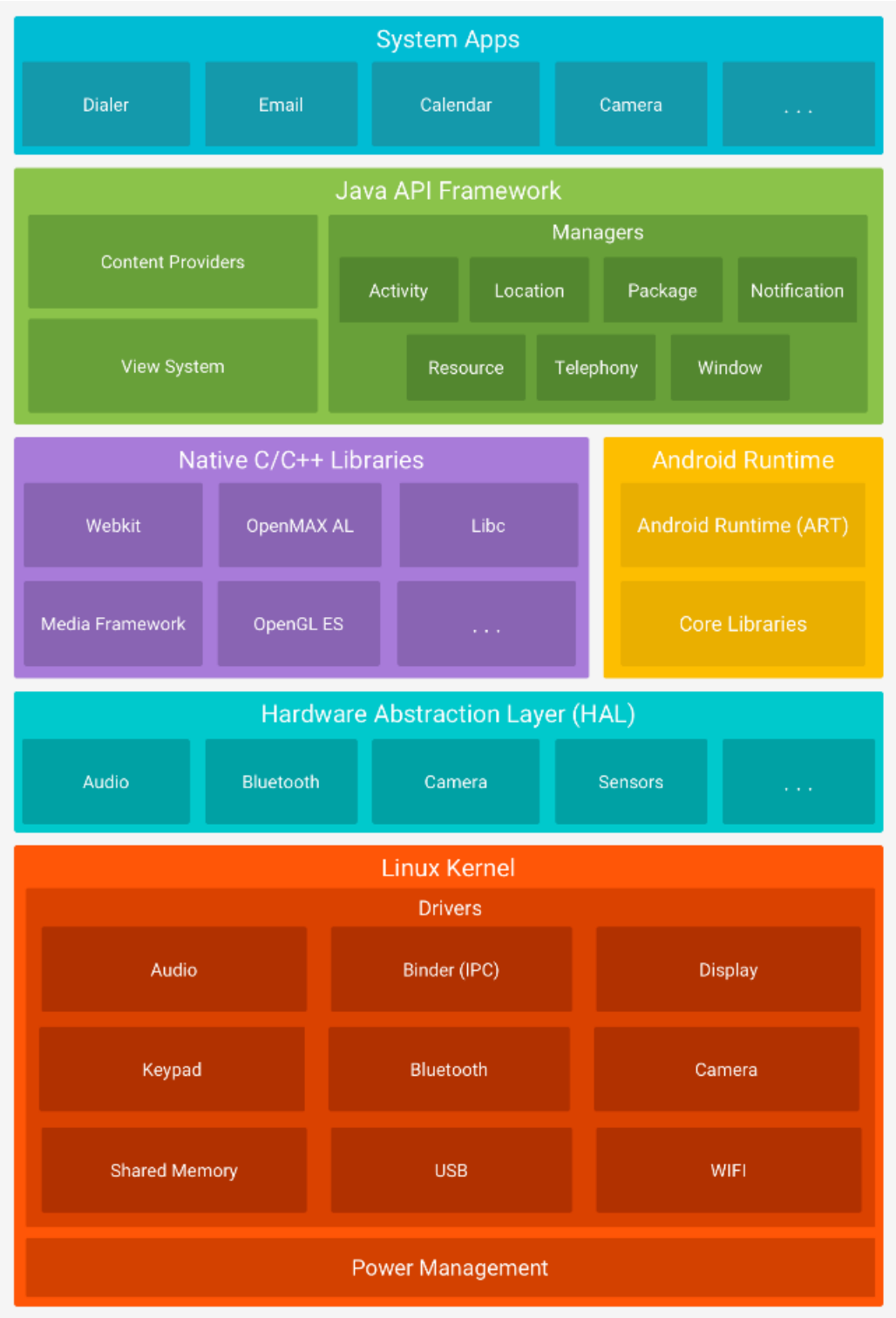

Figura 4. Capas de la arquitectura Android [14]

En la capa más baja se encuentra el kernel de Linux, que proporciona las funcionalidades básicas del sistema, como la gestión de procesos y memoria. El uso del kernel de Linux permite a Android aprovechar sus principales características de seguridad y permite a las y los desarrolladores aprovechar las características ofrecidas por un kernel conocido.

La capa de abstracción de hardware (HAL, por sus siglas en inglés) proporciona interfaces estándar que exponen las capacidades del hardware del dispositivo a la API del *framework* de Java, de nivel superior. La HAL consta de varios módulos de librería, cada uno de los cuales implementa una interfaz para un tipo específico de componente del hardware, como la cámara o el módulo Bluetooth. Cuando una API del *framework* realiza una llamada para acceder al hardware del dispositivo, el sistema Android carga el módulo de librería para ese componente de hardware.

Encima de la HAL, hay un conjunto de librerías nativas escritas en C y C++ que son compiladas para una arquitectura hardware específica. Estas incluyen la versión de libc de Google llamada Bionic, junto con librerías de medios y gráficos (OpenGL ES), compatibilidad con navegadores (Webkit) y una base de datos liviana, SQLite. Muchos componentes y servicios básicos del sistema Android, como ART y HAL, se crean a partir de código nativo que requiere bibliotecas nativas escritas en C y C++. La plataforma Android proporciona APIs del *framework* de Java para exponer la funcionalidad de algunas de estas librerías nativas a las aplicaciones. Por ejemplo, permiten acceder a OpenGL ES a través de la API Java OpenGL.

Junto a las librerías, además del sistema operativo, se encuentra el *runtime*. A diferencia de otras plataformas móviles como iOS o Tizen, que ejecutan software compilado de forma nativa para su arquitectura de hardware específica, la mayoría del software de Android se basa en un lenguaje de código genérico que se transforma de "código de bytes" (*byte-code*) en instrucciones nativas para el hardware en el propio dispositivo. A lo largo de los años y desde las primeras versiones de Android, Dalvik comenzó como una simple máquina virtual con poca complejidad. Sin embargo, con el tiempo, Google sintió la necesidad de añadir mejoras para poder mantenerse al día con los avances del hardware: agregó un compilador JIT y capacidades de subprocesos múltiples. Sin embargo, los avances hardware habían estado superando el desarrollo de Dalvik, por lo que Google decidió crear una nueva máquina virtual, Android Runtime (ART), que sirviera como una base sólida para el futuro, que pudiera escalar con el rendimiento de los dispositivos de 8 núcleos, con grandes capacidades de almacenamiento y memoria [15]. Ambas fueron diseñadas específicamente para Android.

Por encima de las librerías y la capa de tiempo de ejecución de Android, se encuentra la capa del *framework* que proporciona muchos servicios de nivel superior a las aplicaciones en forma de clases escritas en Java. Todo el conjunto de funciones del sistema operativo Android está disponible a través de las API escritas en Java. Estas API forman los componentes básicos necesarios para crear aplicaciones de Android al simplificar la reutilización de los componentes y servicios básicos del sistema modular.

La capa superior es la capa de aplicaciones, la cual contiene una serie de aplicaciones que se distribuyen habitualmente con Android, que pueden incluir correo electrónico, SMS, calendario, contactos y navegador web, entre otros. Las aplicaciones incluidas con la plataforma no tienen un estatus especial entre las aplicaciones que la persona usuaria elige

instalar. Por lo tanto, una aplicación de terceros puede convertirse en el navegador web predeterminado de la persona usuaria, el servicio de mensajería SMS o incluso el teclado predeterminado (se aplican algunas excepciones, como la aplicación Configuración del sistema). Las aplicaciones del sistema funcionan como aplicaciones para las y los usuarios y para proporcionar capacidades clave a las que los desarrolladores pueden acceder desde su propia aplicación. Por ejemplo, si una aplicación desea enviar un mensaje SMS, no necesita integrar esa funcionalidad; en su lugar, puede invocar cualquier aplicación de SMS que ya esté instalada para enviar el mensaje.

3.2.2 Lenguajes

Android soporta 2 lenguajes de programación de forma nativa: Java y Kotlin [16]. Originalmente, Java era el lenguaje oficial para el desarrollo de aplicaciones escogido por Google hasta que, en 2019, Kotlin lo reemplazó. Esto no implica que ya no se pueda usar Java. De hecho, ambos lenguajes se pueden usar simultáneamente en una misma aplicación.

Además de esto, se pueden incluir fragmentos de código escritos en C++, pero no se puede desarrollar una aplicación completa en este lenguaje.

3.2.2.1 Java

Como se ha mencionado, Java era el lenguaje de programación oficial para el desarrollo de aplicaciones de Android hasta hace unos pocos años y, en consecuencia, todavía es el lenguaje más utilizado. Esto implica que tiene una gran comunidad en línea para soporte en caso de problemas en el desarrollo.

Sin embargo, Java es un lenguaje complicado de usar para un o una principiante, ya que contiene mecanismos y recursos complejos como son los constructores, excepciones de puntero nulo, concurrencia, excepciones comprobadas, etc.

3.2.2.2 Kotlin

Kotlin es un lenguaje de programación multiplataforma que se puede usar como una alternativa a Java para el desarrollo de aplicaciones de Android. Puede interoperar con Java y se ejecuta en la máquina virtual de Java.

La única diferencia considerable es que Kotlin elimina las características superfluas de Java, como las excepciones de puntero nulo. También elimina la necesidad de terminar cada línea con un punto y coma. Por todo ello, Kotlin es mucho más fácil de usar para principiantes en comparación con Java.

3.2.2.3 C++

C ++ se puede usar para el desarrollo de aplicaciones de Android utilizando el Kit de desarrollo nativo de Android (NDK). Sin embargo, una aplicación no se puede crear completamente con C++ y el NDK se usa para implementar partes de la aplicación en código nativo de C++.

Si bien C ++ es útil para el desarrollo de aplicaciones de Android en algunos casos, es mucho más difícil de configurar y es mucho menos flexible.

3.2.3 Versiones

Desde su presentación en 2008, Android ha sido actualizado en numerosas ocasiones, tal y como se puede ver en la Tabla 1. Cada nueva versión ha proporcionado nuevas funcionalidades para las y los usuarios y desarrolladores, y se ha adaptado a los cambios en el hardware (desaparición de botones físicos, aparición del *notch* para la cámara, etc.)

La adopción de las nuevas versiones de Android es paulatina, conviviendo diferentes versiones en todo momento. Por ello, las y los desarrolladores deben escoger cuidadosamente la versión mínima necesaria para que su aplicación funcione. Google es consciente de ello y por eso, al escoger una versión mínima para el desarrollo de la aplicación, añade una estimación porcentual del número de dispositivos en uso en el mundo en los que la aplicación funcionará. Se trata de llegar a un compromiso entre establecer una versión mínima lo suficientemente baja para que funcione en un porcentaje de dispositivos aceptable y lo suficientemente alta para poder usar las funcionalidades de las nuevas versiones.

Versión	SDK / nivel de API	Clave de la versión	Nombre clave	Año
Android 13 (DEV)	Level 33	T	Tiramisu 2	2022
Android 12	Level 32 **Android 12L**	S_V2	Snow Cone 2	
	Level 31 **Android 12**	S		2021
Android 11	Level 30	R	Red Velvet Cake 2	2020
Android 10	Level 29	Q	Quince Tart 2	2019
Android 9	Level 28	P	Pie	2018
Android 8	Level 27 **Android 8.1**	O_MR1	Oreo	2017
	Level 26 **Android 8.0**	O		
Android 7	Level 25 **Android 7.1**	N_MR1	Nougat	2016
	Level 24 **Android 7.0**	N		
Android 6	Level 23	M	Marshmallow	2015
Android 5	Level 22 **Android 5.1**	LOLLIPOP_MR1	Lollipop	2015
	Level 21 **Android 5.0**	LOLLIPOP, L		2014
Android 4	Level 20 **Android 4.4W 5**	KITKAT_WATCH	KitKat	

	Level 19 **Android 4.4**	KITKAT		2013
	Level 18 **Android 4.3**	JELLYBEAN_MR2	Jelly Bean	
	Level 17 **Android 4.2**	JELLYBEAN_MR1		2012
	Level 16 **Android 4.1**	JELLYBEAN		
	Level 15 **Android 4.0.3 – 4.0.4**	ICE_CREAM_SANDWICH_MR1	Ice Cream Sandwich	2011
	Level 14 **Android 4.0.1 – 4.0.2**	ICE_CREAM_SANDWICH		
Android 3	Level 13 **Android 3.2**	HONEYCOMB_MR2	Honeycomb	
	Level 12 **Android 3.1**	HONEYCOMB_MR1		
	Level 11 **Android 3.0**	HONEYCOMB		
Android 2	Level 10 **Android 2.3.3 – 2.3.7**	GINGERBREAD_MR1	Gingerbread	
	Level 9 **Android 2.3.0 – 2.3.2**	GINGERBREAD		2010
	Level 8 **Android 2.2**	FROYO	Froyo	
	Level 7 **Android 2.1**	ECLAIR_MR1	Eclair	
	Level 6 **Android 2.0.1**	ECLAIR_0_1		2009
	Level 5 **Android 2.0**	ECLAIR		
Android 1	Level 4 **Android 1.6**	DONUT	Donut	
	Level 3 **Android 1.5**	CUPCAKE	Cupcake	
	Level 2 **Android 1.1**	BASE_1_1	Petit Four	
	Level 1 **Android 1.0**	BASE	*None*	2008

Tabla 1. Versiones de Android [17]

3.3 Soluciones teóricas para la integración de tiempo real en Android

Por defecto, Android no es un SO de tiempo real ni tiene mecanismos de TR [13]. Los principales problemas hallados que impiden la implementación de tareas de TR son:

1) La librería Bionic tiene más restricciones que la libc (librería estándar de C) tradicional y algunas de ellas dificultan su uso en sistemas de tiempo real.
2) La variabilidad en los tiempos de respuesta del kernel de Android y de la máquina virtual Java (ART, Android Runtime) impiden tener tiempos de respuesta acotados.
3) El kernel de Linux utiliza por defecto el planificador *Completely Fair Scheduler* (CFS), "planificador completamente justo". Este planificador proporciona fracciones temporales dinámicas entre las distintas tareas del sistema para intentar ser lo más equitativo posible con todas las tareas que deben ser ejecutadas. Esto es claramente incompatible con los requisitos de determinismo de una aplicación de tiempo real, donde las tareas deben tener tiempos de uso de la CPU acordes con sus prioridades.

Debido a estas limitaciones, en [18] se han propuesto cuatro posibles soluciones para que Android sea adecuado para ejecutar aplicaciones de tiempo real (ver Figura 5).

a) Ejecutar todas las aplicaciones con requisitos temporales directamente sobre un kernel de Linux con características de tiempo real (Figura 5 (a)).
b) Añadir una máquina virtual con características de tiempo real (RT- JVM); de este modo gracias a un kernel de Linux de tiempo real se podrían ejecutar programas Java de tiempo real (Figura 5 (b)).
c) Utilizar los servicios de un kernel de tiempo real junto con una modificación de la máquina virtual de Android (ART VM extendida) (Figura 5 (c)).
d) Utilizar un hipervisor de tiempo real capaz de ejecutar el sistema operativo Android en concurrencia con un sistema operativo de tiempo real (Figura 5 (d)).

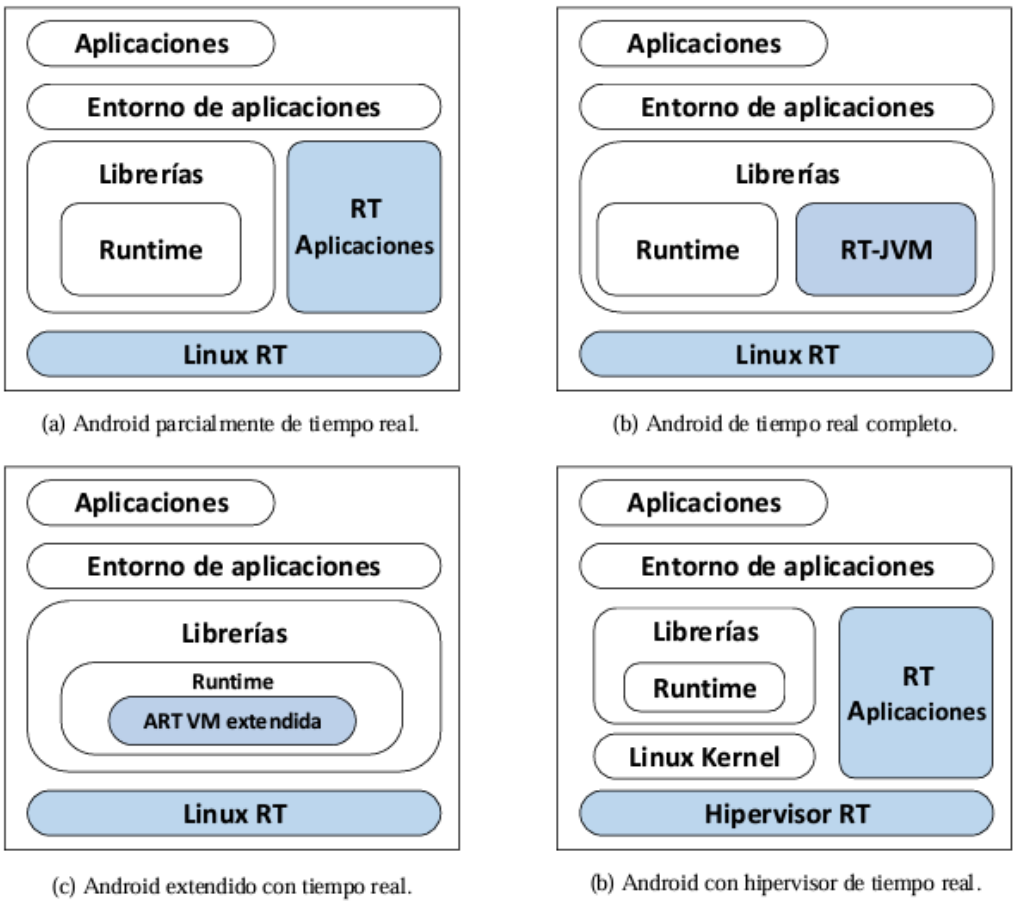

Figura 5. Soluciones propuestas para lograr un comportamiento de TR en Android [13]

A raíz de las propuestas anteriores se han llevado a cabo algunos desarrollos para crear extensiones de tiempo real para Android.

En [19] se ha aplicado el parche PREEMPT_RT sobre el kernel para conseguir características de tiempo real en el sistema. El desarrollo de este parche comenzó en el año 2005 por Ingo

Molnár para tratar de mejorar los tiempos de respuesta de Linux y conseguir otorgarle un mayor grado de predictibilidad temporal. El objetivo de este parche es disminuir las latencias y obtener un sistema más predecible, al mismo tiempo que se mantiene el tipo de kernel monolítico de Linux permitiendo a los desarrolladores escribir aplicaciones de tiempo real en el espacio de usuario. Desde que se inicializó el desarrollo del parche PREEMPT_RT numerosas mejoras aportadas en él se han añadido a la línea principal del kernel de Linux [20].

Además, para permitir una comunicación entre las aplicaciones Java propias de Android con las de tiempo real, han desarrollado un canal de comunicaciones y sincronización entre los dos tipos de aplicaciones.

Otro trabajo [21] [22] ha realizado una adaptación de Android basándose en la solución propuesta en la Figura 5 (c). En este caso también han modificado el kernel con el parche PREEMPT_RT, además de añadir modificaciones en otros componentes de Android como en el recolector de basuras de la máquina virtual Java, en la clase *Service* para permitir el cambio de prioridades a nivel de aplicación Java y en el módulo *Binder* para añadir herencia de prioridades en las llamadas remotas a procedimientos dentro de Android.

Existe otro estudio [23] [24] que ha optado por una solución diferente a las cuatro propuestas en la Figura 5. En este caso se crea un prototipo denominado RTDroid que pretende ser compatible con las aplicaciones escritas para la plataforma Android. Se utiliza una máquina virtual de Java con características de tiempo real (Fiji-VM) sobre un sistema operativo de tiempo real (Linux-RT o RTEMS). La API original de Android ha sido recreada paso a paso para poder ejecutar aplicaciones Android. En cierto modo está solución se basa en la Figura 5 (b), pero en este caso, han construido todo el sistema desde cero.

Las dos primeras implementaciones citadas anteriormente (Figura 5 (a) y Figura 5 (b)) entrañan una alta complejidad relativa a la adaptación del kernel para poder aplicar parches de tiempo real sobre él, ya que las últimas versiones del kernel de Android no están integradas en la rama principal de desarrollo del kernel de Linux. Y la solución propuesta con RTDroid requiere un fuerte proceso de adaptación con cada nueva versión del sistema que aparezca.

Como solución a estos inconvenientes, en [13] se propone una solución más portable y fácil de desarrollar y de mantener. Esta solución consiste en aislar un núcleo de la CPU del dispositivo para que ejecute las aplicaciones con requisitos de TR. Para ello, se deben modificar algunos archivos y características de la configuración del kernel relativas a los núcleos de la CPU antes de compilarlo. Estas modificaciones evitan o mitigan en gran medida los siguientes problemas de una aplicación de TR:

- Interferencias entre aplicaciones de tiempo real de diferentes aplicaciones que se pueden estar ejecutando en el sistema.
- Efecto de los manejadores de interrupciones sobre las tareas de la aplicación.
- Cambios dinámicos de frecuencia y apagado automático de los núcleos del procesador.

Por otro lado, existen limitaciones de la librería Bionic para las aplicaciones de tiempo real. La librería Pthreads [25] que se encuentra en la implementación de la glibc tradicional se basa en el estándar POSIX definido por IEEE para ser compatible con diferentes sistemas operativos. Sin embargo, la librería Bionic ha realizado importantes cambios en este aspecto. Debido a estos cambios, algunas funciones imprescindibles como las relacionadas con los protocolos de los mutexes no se encuentran implementadas.

No obstante, en el mismo trabajo se ha demostrado que es posible utilizar la librería glibc en Android, en lugar de la librería Bionic. De esta forma, sería posible crear aplicaciones de TR en Android, que deben estar escritas en C.

El problema principal que queda por resolver en esta solución es que todavía no se dispone de un mecanismo de comunicación y sincronización entre las aplicaciones de tiempo real ejecutadas en núcleo aislado y el resto de aplicaciones de Android.

3.4 Conclusiones

Las soluciones tecnológicas para el diagnóstico y seguimiento de la EM ofrecen la posibilidad de medir la actividad y evolución de las personas afectadas. Dentro de estas soluciones tecnológicas, destacan los dispositivos pasivos, que permiten una monitorización diaria en contraposición a los diagnósticos puntuales realizados en los entornos clínicos. Mediante los datos recogidos por estos dispositivos se pueden definir diferentes indicadores que permiten cuantificar la cantidad o calidad del movimiento del paciente, permitiendo una posterior clasificación de su estado. Así, para hacer posible esta monitorización continua del estado del o la paciente resulta necesario diseñar y desarrollar soluciones asequibles que claramente pueden beneficiarse de la disponibilidad de dispositivos móviles por parte de la inmensa mayoría de las y los usuarios potenciales.

En ese sentido, Android se ha consolidado como el SO más utilizado en dispositivos móviles a nivel mundial. Distribuido mediante licencias de *software* libre que permiten el uso, modificación y redistribución, ofrece gran flexibilidad tanto para modificarlo como para desarrollar aplicaciones. En este aspecto, son varias las propuestas para agregar a Android mecanismos de tiempo real. No obstante, las modificaciones necesarias no son triviales y requieren una modificación sustancial del SO o no están completamente desarrolladas. Es por ello que la solución propuesta en este trabajo fin de máster se basará en el SO por defecto, de forma que las y los usuarios puedan seguir utilizando sus dispositivos con normalidad. Esto implica que solo se podrá diseñar y desarrollar una aplicación de tiempo real flexible.

4 DISEÑO DE LA SOLUCIÓN

La solución desarrollada se divide en dos partes fundamentales: *hardware* (HW) y *software* (SW). La parte HW la conforman la CI, el móvil y el BLE; y la parte SW se corresponde con la aplicación de Android.

La CI se compone de sensores y una placa de procesamiento basada en el microcontrolador nRF52 con BLE integrado. Esta placa está programada para capturar los datos de los sensores y transmitirlos mediante BLE con un periodo de 20 ms. Esto condiciona la estructura del proceso de adquisición de datos de la aplicación, la cual debe comenzar cada ciclo de adquisición, procesado y almacenado cuando reciba la señal BLE de la CI.

El móvil Android integra la aplicación desarrollada en este trabajo fin de máster (*Crutch app*), que se encarga de recoger los datos transmitidos por la CI y almacenarlos en la memoria interna del móvil. Posteriormente, estos datos se pueden almacenar en un archivo Excel para su posterior procesamiento. También se dispone de la opción de enviar el archivo Excel por email para visualizar y procesar sus datos en un ordenador. En la Figura 6 se puede observar este proceso de forma esquemática.

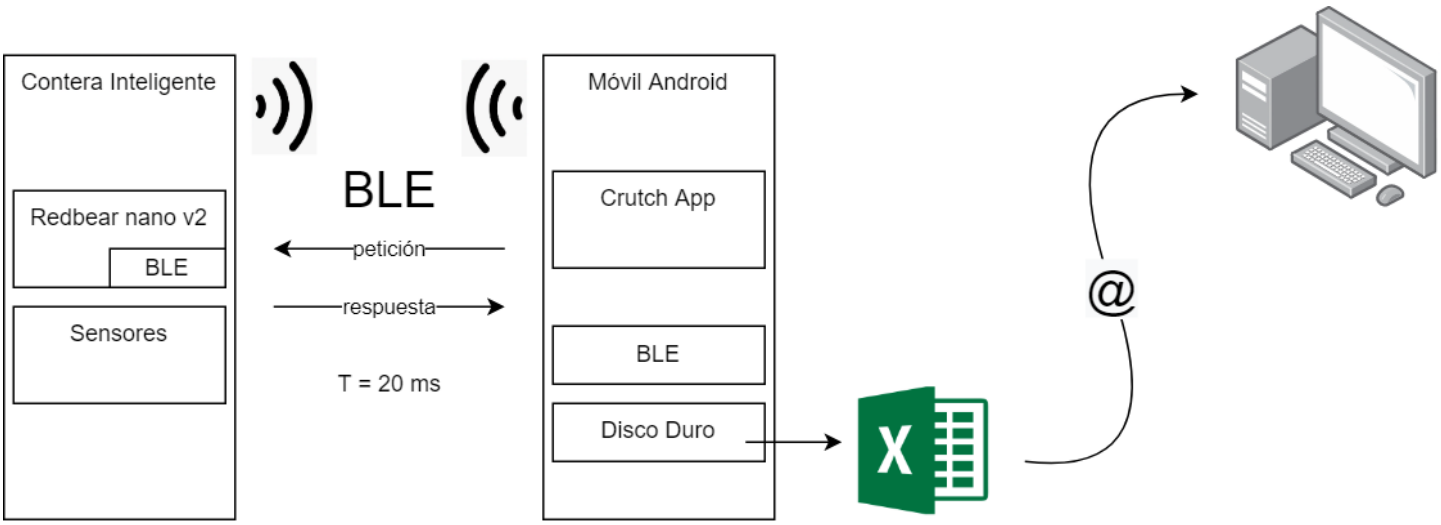

Figura 6. Esquema solución desarrollada

4.1 HW

Los dispositivos hardware que integran el sistema son la CI y el móvil. La primera captura los datos medidos por los sensores y los envía por BLE con un periodo de 20 ms. El móvil se conecta a la CI por BLE y adquiere los datos en tiempo real. En la Figura 7 se muestra el esquema de comunicación entre los componentes hardware.

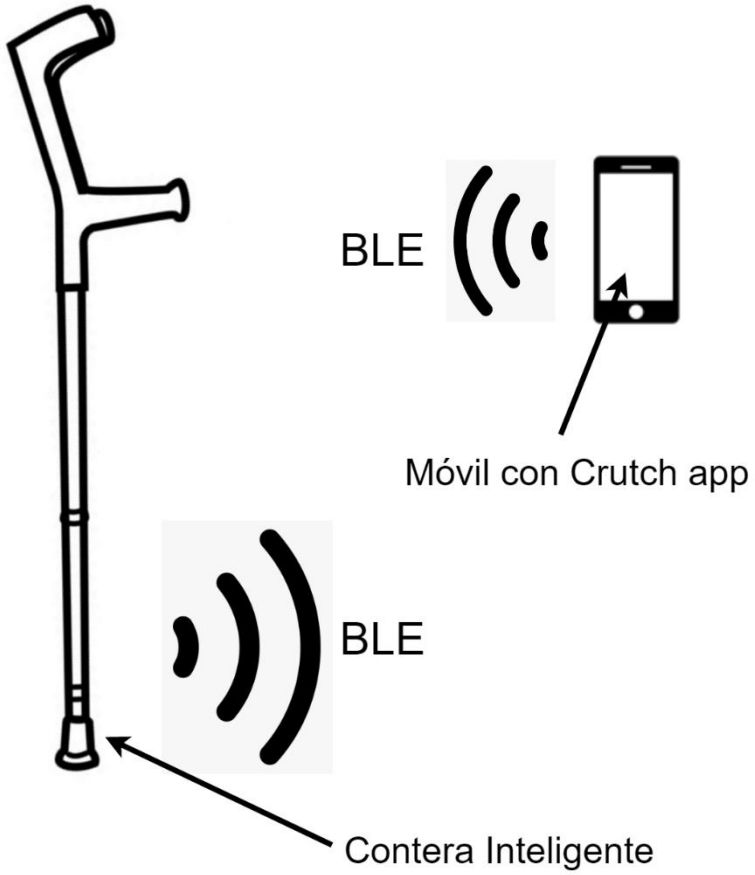

Figura 7. Esquema simple de comunicación entre la CI y el móvil

4.1.1 Contera inteligente

En el estado actual del proyecto SMARTIP, la CI (Figura 8) se encuentra diseñada, fabricada y verificada. Concretamente, se han producido 3 versiones de la CI con las que poder adquirir datos que caractericen el estado de marcha y equilibrio. Cada una de ellas cuenta con un sensor de fuerza, una IMU y un barómetro.

El sensor de fuerza C9C fabricado por HBM permite medir fuerzas de compresión en un rango de 50 N a 50 KN con una precisión del 0.2%. Además, debido a su elevada frecuencia fundamental puede realizar mediciones muy rápidas. Asimismo, es un sensor de dimensiones reducidas.

La IMU MTi-1 de X-Sense permite medir aceleraciones en 3 ejes, orientación e intensidad de campo magnético. Lleva integrados un acelerómetro 3D con rango de ±156.96 m/s^2, un giróscopo con rango de ±2000 °/s y un magnetómetro con rango de ±0.8 G.

El sensor BMP280 permite medir la presión barométrica y la temperatura. El sensor cuenta con un rango de presión de 300 a 1100 hPa y una precisión de 1 hPa. En cuanto a la temperatura, el rango de medición es de 40 °C a 85 °C y la precisión de 1.0 °C.

Una vez que se han captado los datos de los sensores, estos se transmiten mediante una RedBear BLE Nano V2 con un periodo de 20 ms. Esta placa de pequeñas dimensiones y bajo consumo incorpora BLE para poder transmitir los datos.

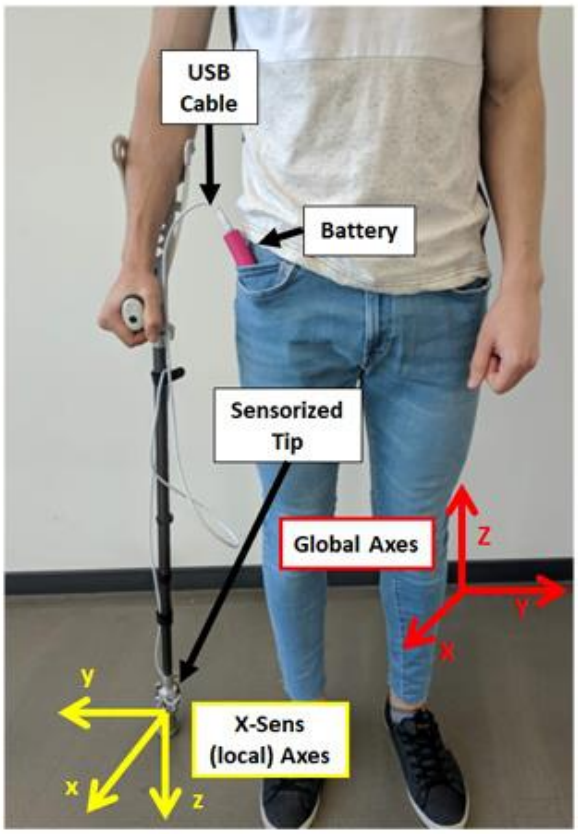

Figura 8. Contera inteligente

4.1.2 Móvil

El móvil empleado es un Motorola One Action (Figura 9). Fue originalmente lanzado al mercado a finales de 2019. En un inicio, incorporaba Android 9.0 de fábrica, pero se ha actualizado a Android 10.0 para la realización del trabajo.

El móvil cuenta con una CPU Samsung Exynos 9609 de ocho núcleos que funciona a una frecuencia base de 2,2 GHz, una GPU Mali G72 MP3 y una memoria RAM de 4 GB [26].

Este modelo tiene una pantalla de 6,3 pulgadas y una resolución de 1080x2520 píxeles, con una relación de aspecto 21:9.

Figura 9. Motorola One Action

4.1.3 BLE

Bluetooth Low Energy (BLE) es un subconjunto del estándar Bluetooth v4.0 de comunicación inalámbrica orientada a operaciones de muy bajo consumo. Se usa a menudo en dispositivos relacionados con la monitorización de la salud y con aplicaciones de bajo consumo debido a las limitaciones de tamaño y, por ende, de capacidad energética de estos dispositivos. Se introdujo en la versión 4.0 de Bluetooth en 2010. Transmite datos a través de 40 canales en la banda de frecuencias Industriales, Científicas y Médicas (ISM, por sus siglas en inglés) de 2.4 GHz, la misma que usa Bluetooth [27].

La pila del protocolo BLE se divide en tres partes básicas, organizadas por niveles: controlador (*controller*), anfitrión (*host*) y aplicación (*app*) (Figura 10) [28].

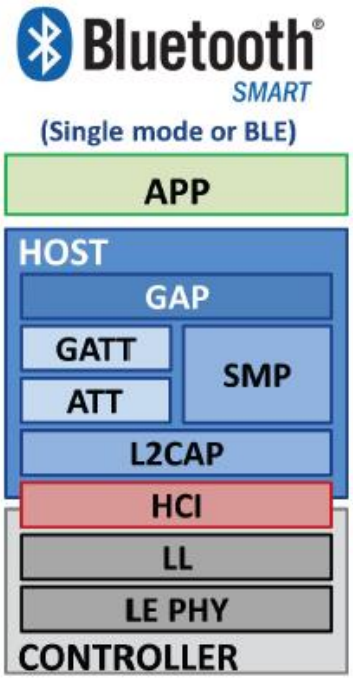

Figura 10. Pila de capas del BLE [28]

La aplicación es el bloque más alto de la pila y representa la interfaz directa con la persona usuaria. Define unos perfiles gracias a los cuales distintas aplicaciones, que reutilizan funcionalidades comunes, son capaces de interoperar.

El anfitrión incluye las siguientes capas:

- Perfil de Acceso Genérico (GAP)
- Perfil de Atributo Genérico (GATT)
- Protocolo de Adaptación y Control de Enlace Lógico (L2CAP)
- Protocolo de Atributos (ATT)
- Protocolo de Administrador de Seguridad (SMP)
- Interfaz de Controlador de Host (HCI), lado del anfitrión

El controlador está estructurado en las siguientes capas:

- Interfaz de Controlador de Host (HCI), lado del controlador
- Capa de Enlace (LL)
- Capa Física (PHY)

De todas ellas, la que es de relevancia para este trabajo es la capa GATT. El GATT encapsula la capa ATT y su función principal es la de establecer cómo intercambiar los datos. Estos datos se organizan en una estructura compuesta por servicios que, a su vez, agrupan los datos en características (Figura 11).

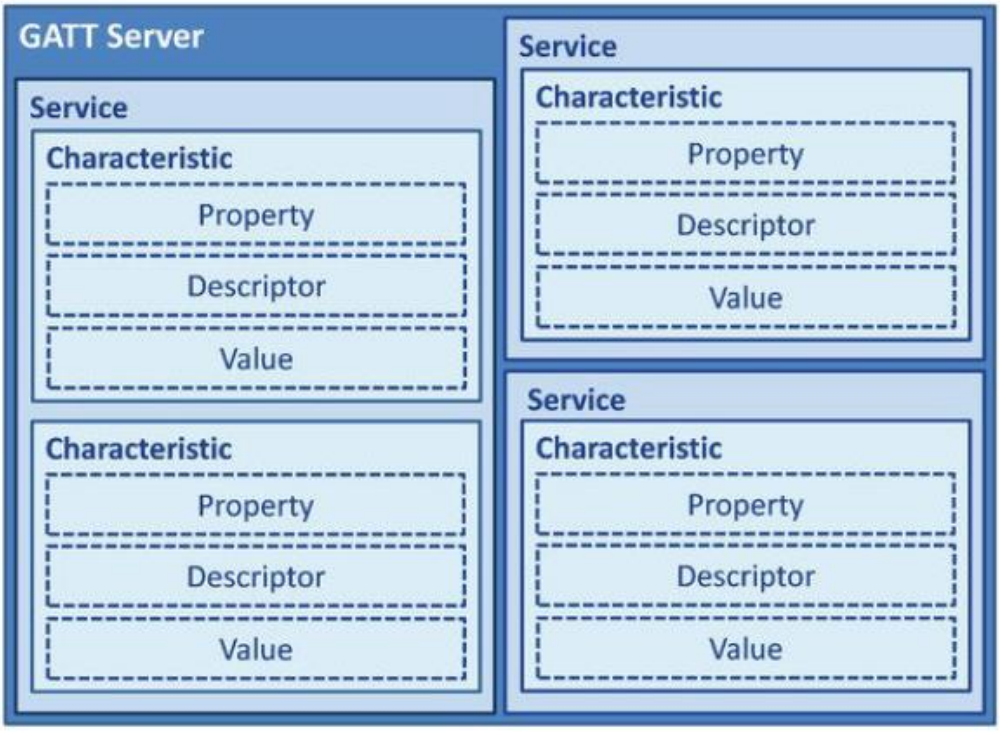

Figura 11. Descripción de la capa GATT: servicios y características [28]

Las características contienen los datos a transmitir y un descriptor. El descriptor contiene las propiedades de la característica asociada. Las propiedades indican las operaciones que se pueden realizar sobre las características, siendo las más frecuentes:

- *Broadcast* (difusión): permite enviar datos a los dispositivos BLE mediante paquetes publicitarios.
- *Readable* (leíble): si se establece, el cliente sólo puede leer el valor de la característica.
- *Writable* (escribible): con esta propiedad, el cliente sólo puede escribir un nuevo valor en la característica.
- *Notifiable* (notificable): si está establecida, el cliente recibe una notificación si el servidor actualiza la característica, para que pueda leer el nuevo valor.

El Bluetooth SIG define algunos servicios y características estándar representados por una dirección de 16 bits, pero el BLE permite a los fabricantes definir sus propios servicios utilizando un UUID de 128 bits para adaptar esta tecnología a sus aplicaciones.

4.1.3.1 BLE en la CI

La contera inteligente está configurada con un único servicio cuyo Identificador Único Universal (UUID, por sus siglas en inglés) toma el valor hexadecimal 1000. Dentro de este servicio, se encuentran integradas dos características para la transmisión de datos. Los UUIDs de las características son 1001 y 1002.

El tamaño máximo de una característica es de 20 bytes (160 bits) y los 14 valores que necesita transmitir la muleta ocupan 317 bits, por lo que se emplean 2 características. De los 317 bits, 309 son los valores recogidos por los sensores y los 8 restantes corresponden a valores para la comprobación de errores de transmisión (ver Tabla 2 para el desglose de los datos de ambas características). Estos valores son números enteros cíclicos del 0 al 9 que toman el mismo valor en cada una de las características al ser enviados y que son empleados para verificar la correcta recepción de los datos. Para verificar que la recepción de datos es correcta, ambos valores deben ser iguales en el momento de la recepción. Además, deben ser recibidos en orden, sin saltos, de forma cíclica (primero “0”, luego “1”, etc.)

Byte	**Característica 1**		**Característica 2**	
	Dato	bits	Dato	bits
0	ErrorCheck 1 Force	4 4	ErrorCheck 2 Accelerometer Y	4 4
1	Force	8	Accelerometer Y	8
2	Force Altitude	4 4	Accelerometer Y	8
3	Altitude	8	Accelerometer Z	8

4	Altitude	8	Accelerometer Z	8
5	Altitude	8	Accelerometer Z	8
6	Altitude Roll	4 4	Accelerometer Z Gyroscope X	1 7
7	Roll	8	Gyroscope X	8
8	Roll	8	Gyroscope X Gyroscope Y	5 3
9	Roll Pitch	6 2	Gyroscope Y	8
10	Pitch	8	Gyroscope Y	8
11	Pitch	8	Gyroscope Y Gyroscope Z	1 7
12	Pitch	8	Gyroscope Z	8
13	Yaw	8	Gyroscope Z Magnetometer X	5 3
14	Yaw	8	Magnetometer X	8
15	Yaw	8	Magnetometer X Magnetometer Y	5 3
16	Yaw Accelerometer X	2 6	Magnetometer Y	8
17	Accelerometer X	8	Magnetometer Y Magnetometer Z	5 3
18	Accelerometer X	8	Magnetometer Z	8
19	Accelerometer X Accelerometer Y	3 5	Magnetometer Z	5

Tabla 2. Descripción bytes características BLE

4.2 SW

La parte SW del proyecto comprende la aplicación desarrollada para dispositivos Android que permite la comunicación con la CI. La aplicación cuenta con 4 estados: configuración, inicialización, adquisición (tiempo real) y visualización, representados en la Figura 12. Estos estados se distribuyen de acuerdo a los 2 modos de uso de la aplicación: *modo de adquisición de datos* y *modo de visualización de datos guardados*.

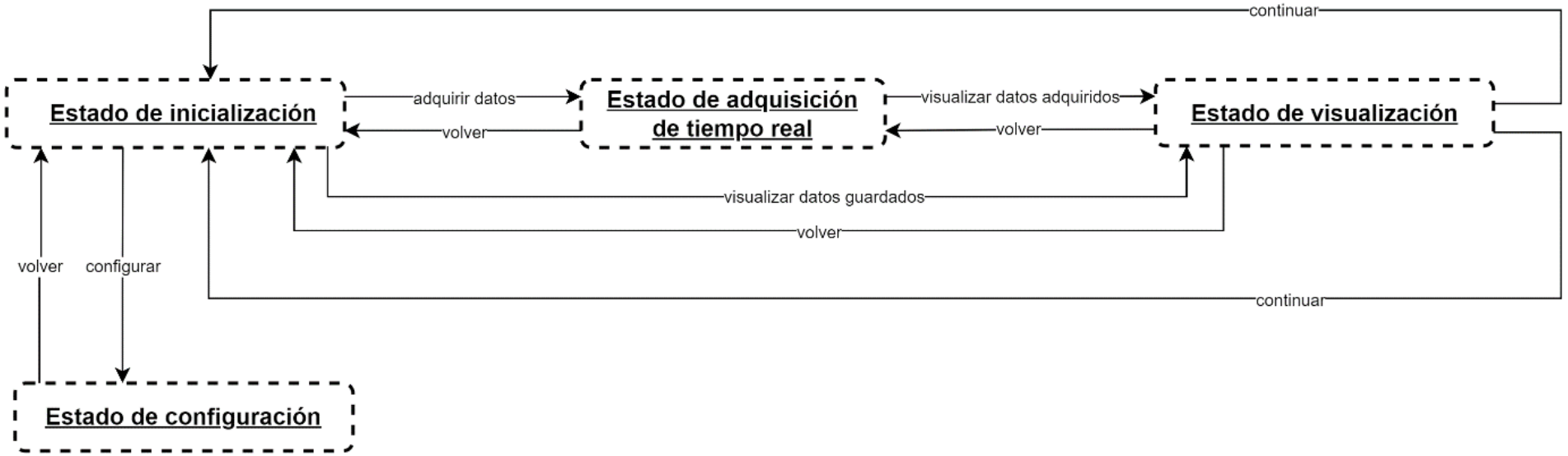

Figura 12. Esquema de los estados de la aplicación

El estado de inicialización es común a los 2 modos de uso (*modo de adquisición de datos* y *modo de visualización de datos guardados*). En él se introducen los parámetros de entrada necesarios para el funcionamiento de la aplicación. Concretamente, si se desea emplear el *modo de adquisición de datos*, se deben introducir la dirección MAC de la CI a la que se desea conectar la aplicación, el ID de usuario para el que se quiere adquirir los datos y la iteración de la prueba, es decir, el número asociado a la prueba, ya que un mismo ID de usuario puede tener asociadas varias pruebas. Una vez introducidos estos valores en el *modo de adquisición de datos*, se pasa al estado de adquisición de datos, que es el único de tiempo real, para realizar la adquisición de datos. Si se desea utilizar la aplicación en el *modo de visualización de datos guardados*, solo se deben seleccionar el ID de usuario y el número de iteración de la prueba asociados a los datos que se desea visualizar. Una vez seleccionados, se pasa al estado de visualización.

El estado de adquisición solo pertenece al *modo de adquisición de datos*. Es el único estado de tiempo real de la aplicación. En este estado se realiza la adquisición de datos enviados por la CI mediante BLE. Los datos recibidos son procesados para convertir su formato de *array* de bytes, que es como se transmiten por BLE, a sus valores *integer* y *float* correspondientes. A continuación, se guardan los datos en la memoria interna del móvil. Una vez finalizada la adquisición de datos, se pasa al estado de visualización.

El estado de visualización es común a los 2 modos de uso (*modo de adquisición de datos* y *modo de visualización de datos guardados*). Se puede acceder a él después de realizar la adquisición de datos o a través del estado de inicialización. En este estado se pueden visualizar de forma gráfica cada uno de los datos adquiridos y también se pueden guardar estos datos en un archivo Excel. Este archivo permite que los datos adquiridos sean visualizados de forma ordenada en cualquier instante y dispositivo al estar guardados en un formato ampliamente utilizado y que es compatible con aplicaciones gratuitas (GoogleDocs, OpenOffice…). Además, se dispone de la posibilidad de enviar este archivo Excel con los datos guardados mediante email para exportar el archivo a otros dispositivos rápidamente. Este archivo Excel se compone de 4 columnas iniciales para guardar la marca de tiempo de adquisición de cada paquete de datos desglosada (horas, minutos, segundos y milisegundos) y otras 16 columnas correspondientes a los datos que captura la CI (Figura 13).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Hour	Min	Sec	Millis	ErrorCheck 1	Force	Altitude	Pitch	Roll	Yaw	AccX	ErrorCheck 2	AccY	Acc Z	Gyroscope X	Gyroscope Y	Gyroscope Z	Magnetic X	Magnetic Y	Magnetic Z
2	23	14	57	72	6	0,354492	109,268082	164,345398	2,4545	-23,665899	-0,4365	6	2,691	-9,4912	-0,0006	0,0014	0,0057	0,0642	-0,5468	0,7726
3	23	14	57	742	7	0,35127	109,268082	164,343903	2,4553	-23,657	-0,4089	7	2,6828	-9,4595	-0,0004	0,001	0,0042	0,0618	-0,5492	0,771
4	23	14	57	759	8	0,35127	109,498268	164,344894	2,4583	-23,641899	-0,4266	8	2,6841	-9,4799	-0,0002	0,0012	0,0052	0,0661	-0,5459	0,7752
5	23	14	57	787	9	0,35127	109,498268	164,347397	2,4559	-23,632999	-0,436	9	2,6699	-9,4697	0,0031	0,0011	0,0073	0,0642	-0,5483	0,7736
6	23	14	57	803	0	0,35127	109,28231	164,346405	2,455	-23,632999	-0,422	0	2,6576	-9,4819	-0,0002	0,002	0,0065	0,0617	-0,5487	0,772
7	23	14	57	82	1	0,35127	109,28231	164,344894	2,4553	-23,6381	-0,4286	1	2,6478	-9,4903	0,0004	0,0001	0,0048	0,0627	-0,5487	0,7721
8	23	14	57	832	2	0,35127	109,432213	164,343094	2,4586	-23,642	-0,4347	2	2,661	-9,4737	0,0009	0,0008	0,005	0,0642	-0,5469	0,7761
9	23	14	57	872	3	0,35127	109,432213	164,341797	2,4595	-23,648899	-0,4261	3	2,6663	-9,4841	-0,0027	-0,0006	0,0066	0,0642	-0,5487	0,7716
10	23	14	57	886	4	0,35127	109,207108	164,343094	2,4564	-23,6553	-0,4362	4	2,6579	-9,4728	0,0029	0,0019	0,0071	0,0657	-0,5487	0,7711
11	23	14	57	895	5	0,35127	109,314827	164,346405	2,4547	-23,6625	-0,4173	5	2,6648	-9,4738	0,0042	0,003	0,0071	0,0632	-0,5507	0,7715
12	23	14	57	922	6	0,35127	109,314827	164,346298	2,4578	-23,6696	-0,4223	6	2,6573	-9,4765	0,0026	0,0005	0,0063	0,0637	-0,5483	0,7746
13	23	14	57	937	7	0,35127	108,837196	164,344696	2,457	-23,6775	-0,4388	7	2,6549	-9,4643	0,0021	0,0003	0,0068	0,0607	-0,5473	0,7721
14	23	14	57	951	8	0,35127	108,837196	164,347107	2,4583	-23,684299	-0,4225	8	2,6652	-9,4985	0,0009	0,0007	0,0072	0,0646	-0,5468	0,7736
15	23	14	57	981	9	0,35127	108,593811	164,344406	2,4547	-23,682501	-0,4327	9	2,6808	-9,4785	-0,0009	0,0035	0,0069	0,0652	-0,5516	0,7706
16	23	14	57	997	0	0,35127	108,593811	164,347702	2,4538	-23,682301	-0,4253	0	2,6648	-9,4825	0,0023	0,0001	0,0069	0,0632	-0,5473	0,7716
17	23	14	58	012	1	0,35127	108,551643	164,347	2,4577	-23,6805	-0,4326	1	2,66	-9,4826	0,0021	-0,0015	0,0069	0,0642	-0,5468	0,7726
18	23	14	58	041	2	0,35127	108,762505	164,346802	2,4601	-23,685101	-0,4281	2	2,6584	-9,4773	0,0002	0,0001	0,0047	0,0618	-0,5493	0,774
19	23	14	58	058	3	0,35127	108,696449	164,348801	2,4598	-23,687901	-0,4279	3	2,6603	-9,4849	0,0025	0,0008	0,0047	0,0632	-0,5478	0,7741
20	23	14	58	071	4	0,35127	108,570442	164,348404	2,4586	-23,694599	-0,4171	4	2,6747	-9,4739	0,0025	0,0002	0,0071	0,0647	-0,5473	0,7712
21	23	14	58	102	5	0,35127	108,570442	164,351105	2,4594	-23,702801	-0,4173	5	2,6713	-9,4709	0,0039	-0,0008	0,0088	0,0618	-0,5502	0,7725
22	23	14	58	117	6	0,35127	108,622269	164,353195	2,4603	-23,711	-0,435	6	2,6536	-9,4908	0,0041	0,0011	0,006	0,0627	-0,5488	0,7726
23	23	14	58	132	7	0,35127	108,622269	164,350296	2,4611	-23,7068	-0,438	7	2,6468	-9,4957	-0,0003	0,0019	0,0062	0,0656	-0,5483	0,7766
24	23	14	58	166	8	0,35127	108,583649	164,349594	2,4615	-23,6961	-0,443	8	2,6624	-9,4936	-0,0008	0,0001	0,0052	0,0617	-0,5478	0,7735
25	23	14	58	18	9	0,35127	108,583649	164,351807	2,4607	-23,7012	-0,4296	9	2,6834	-9,4796	0,004	0,0017	0,0066	0,0608	-0,5497	0,774
26	23	14	58	193	0	0,35127	108,541985	164,350006	2,4597	-23,698099	-0,4363	0	2,6639	-9,4739	-0,0009	0,0033	0,006	0,0622	-0,5478	0,7731
27	23	14	58	223	1	0,35127	108,617691	164,350296	2,4606	-23,6884	-0,4294	1	2,6763	-9,4833	0,0027	0,0009	0,0064	0,0637	-0,5468	0,7711
28	23	14	58	24	2	0,35127	108,617691	164,3517	2,4611	-23,677401	-0,4347	2	2,6667	-9,4735	-0,0002	-0,0007	0,0056	0,0642	-0,5468	0,7731
29	23	14	58	253	3	0,35127	108,37838	164,348999	2,4618	-23,666	-0,4351	3	2,6568	-9,487	-0,0003	0,0015	0,0046	0,0647	-0,5497	0,7741
30	23	14	58	282	4	0,348047	108,37838	164,346802	2,4606	-23,656	-0,4241	4	2,6574	-9,465	0	0,0015	0,0067	0,0617	-0,5458	0,7726
31	23	14	58	298	5	0,35127	108,598389	164,349197	2,4619	-23,6602	-0,4269	5	2,6655	-9,4871	0,0012	0,0007	0,0055	0,0632	-0,5473	0,7751
32	23	14	58	313	6	0,35127	108,598389	164,348206	2,4627	-23,6674	-0,4273	6	2,6579	-9,4835	0,0017	0	0,0062	0,0612	-0,5478	0,7725
33	23	14	58	343	7	0,35127	108,499817	164,348907	2,4611	-23,6635	-0,4292	7	2,6596	-9,46	0,0017	0,0046	0,0059	0,0642	-0,5483	0,7716
34	23	14	58	358	8	0,35127	108,499817	164,349503	2,4596	-23,653799	-0,4273	8	2,669	-9,4868	0,001	0,0023	0,0056	0,0632	-0,5483	0,7755
35	23	14	58	372	9	0,35127	108,635986	164,348602	2,462	-23,6535	-0,4177	9	2,6706	-9,4912	0,0022	0,0009	0,0082	0,0637	-0,5493	0,7735
36	23	14	58	402	0	0,35127	108,561806	164,347504	2,4616	-23,6588	-0,4318	0	2,658	-9,489	0	0,002	0,0058	0,0627	-0,5483	0,775
37	23	14	58	42	1	0,35127	108,561806	164,344604	2,4627	-23,662001	-0,4412	1	2,6563	-9,4841	-0,0018	0,0005	0,0035	0,0628	-0,5507	0,776
38	23	14	58	433	2	0,35127	108,673584	164,341705	2,4644	-23,6675	-0,4394	2	2,6783	-9,4694	-0,0002	0,0003	0,0053	0,0666	-0,5492	0,7721
39	23	14	58	463	3	0,35127	108,673584	164,344406	2,4654	-23,6735	-0,423	3	2,6664	-9,4816	0,0007	0,0007	0,0065	0,0657	-0,5493	0,7741
40	23	14	58	479	4	0,35127	108,499817	164,343796	2,4643	-23,678801	-0,426	4	2,6487	-9,4996	0	0,0009	0,0053	0,0622	-0,5483	0,7745
41	23	14	58	492	5	0,35127	108,499817	164,345001	2,4625	-23,684799	-0,4291	5	2,6585	-9,4876	0,0025	0,0017	0,005	0,0603	-0,5493	0,774
42	23	14	58	523	6	0,35127	108,589241	164,341797	2,4608	-23,6896	-0,4285	6	2,6617	-9,481	-0,002	0,0031	0,0041	0,0617	-0,5473	0,7711
43	23	14	58	539	7	0,354492	108,589241	164,338196	2,4641	-23,6863	-0,4248	7	2,6552	-9,4589	-0,0026	0,0011	0,0048	0,0637	-0,5488	0,7726
44	23	14	58	552	8	0,354492	108,716263	164,337906	2,4613	-23,6745	-0,4284	8	2,6849	-9,4645	0,003	0,0024	0,0051	0,0662	-0,5502	0,7711
45	23	14	58	581	9	0,35127	109,123268	164,340302	2,4623	-23,6653	-0,422	9	2,6577	-9,4709	0,0014	0,0007	0,0074	0,0647	-0,5498	0,776

Figura 13. Estructura del archivo Excel de los datos adquiridos

El estado de configuración también es compartido por los 2 modos de uso (*modo de adquisición de datos* y *modo de visualización de datos guardados*). Su función es poder borrar el contenido de la memoria interna del móvil relativa a los datos guardados y los archivos Excel generados con los datos guardados para impedir que la memoria del móvil se sature.

En otro orden de ideas, cabe hacer unas aclaraciones acerca de los recursos internos que ofrece Android empleados en el desarrollo de los estados mencionados. En Android las pantallas de la aplicación se crean mediante la clase *Activity*, a la que se le asocia un *layout* (interfaz) escrito en XML. La clase *Service* es otro componente que no tiene asociado una interfaz y que permite la ejecución de tareas en primero o segundo plano. A nivel interno, la aplicación consta de 7 *Activities* para cada una de las pantallas, 1 *Service* que implementa el BLE, 1 extensión de la aplicación para el almacenamiento de variables globales y 1 clase que implementa la base de datos.

4.2.1 Estado de configuración

La única función de este estado es evitar que el almacenamiento del dispositivo se sature con los datos guardados, permitiendo borrar los datos antiguos o que ya no sean necesarios.

Concretamente, se pueden borrar la base de datos y los archivos Excel guardados. Un mensaje de confirmación aparece antes de realizar la acción para evitar borrados accidentales.

4.2.2 Estado de inicialización

En el estado de inicialización, el usuario de la aplicación debe introducir los valores necesarios para la adquisición de datos.

Si el modo es el de *adquisición de datos*, los valores necesarios son: la dirección MAC de la CI a la que se va a conectar, el ID del usuario actual y el número de iteración de la prueba (número único para cada prueba ya que un mismo ID de usuario puede tener asociadas varias pruebas). Además, el Bluetooth debe estar conectado en el dispositivo y, si la versión de Android es igual o superior a la 10.0, también debe estar conectada la localización. Esto es un requisito impuesto por el propio Android para poder usar todas las funcionalidades Bluetooth. En este estado se realiza la conexión entre la CI y el dispositivo, pero no se reciben los datos. Este proceso se encuentra representado en el Seudocódigo 1.

```
MAC_introducida = inputField.extraerTexto();
if (Bluetooth_desconectado)
      pedir_activar_bluetooth;
else if (GPS_desconectado && GPS_necesario)
      pedir_activar_gps;
else if (MAC_introducida.estaVacia)
      mensaje("Introduce una dirección MAC");
```

```
else if (MAC_introducida.noEsValida)
      mensaje("La dirección MAC no es válida");
else
      intentarConectar(MAC_introducida);
      mensaje("Intentado conectar a MAC: " + MAC_introducida);
      esperarParaEjecutar({
      if (estadoConexion == CONECTADO)
            mensaje("Conectado a MAC: " + MAC_introducida);
            IdUsuario = inputFieldId.extraerTexto();
            iteracion = inputFieldIteracion.extraerTexto();
            irASiguienteActivity();
      else if (estadoConexion == DESCONECTADO)
            mensaje("No se pudo conectar a MAC: " + MAC_introducida)
      }, 1000 milisegundos)

cuandoCambieEstadoConexion(nuevoEstado) {
      if (nuevoEstado == CONECTADO)
            estadoConexion = CONECTADO;
            mensaje("Conectado a MAC: " + MAC_introducida);
            IdUsuario = inputFieldId.extraerTexto();
            iteracion = inputFieldIteracion.extraerTexto();
            irASiguienteActivity();
      else if (nuevoEstado == DESCONECTADO)
            estadoConexion = DESCONECTADO;
            mensaje("Desconectado de MAC: " + MAC_introducida);
};
```

Seudocódigo 1. Estado de inicialización (modo de adquisición de datos)

Si el modo es el de *visualización de datos guardados*, la dirección MAC no es necesaria y solo se deben escoger el ID del usuario y el número de iteración de la prueba cuyos datos se desea visualizar. En este modo, no es necesario ninguna acción más y se procede directamente al estado de visualización. Este proceso se encuentra representado en el Seudocódigo 2.

```
IdUsuario = inputFieldId.extraerTexto();
iteracion = inputFieldIteracion.extraerTexto();
irASiguienteActivity();
```

Seudocódigo 2. Estado de inicialización (modo de visualización de datos)

4.2.3 Estado de adquisición (tiempo real)

Este estado es el único de TR de la aplicación. Se compone de 4 tareas, 3 de ellas esporádicas: recepción de datos, procesamiento de datos y almacenamiento de datos; y 1 periódica:

actualización de pantalla. El estado requiere de un recurso protegido correspondiente a los datos de las características del BLE descritos en el apartado 4.1.3 BLE. Este estado es representado mediante un diagrama de componentes simplificado HRT-UML tal y como se muestra en la Figura 14, y cuyas tareas se detallan en los siguientes apartados.

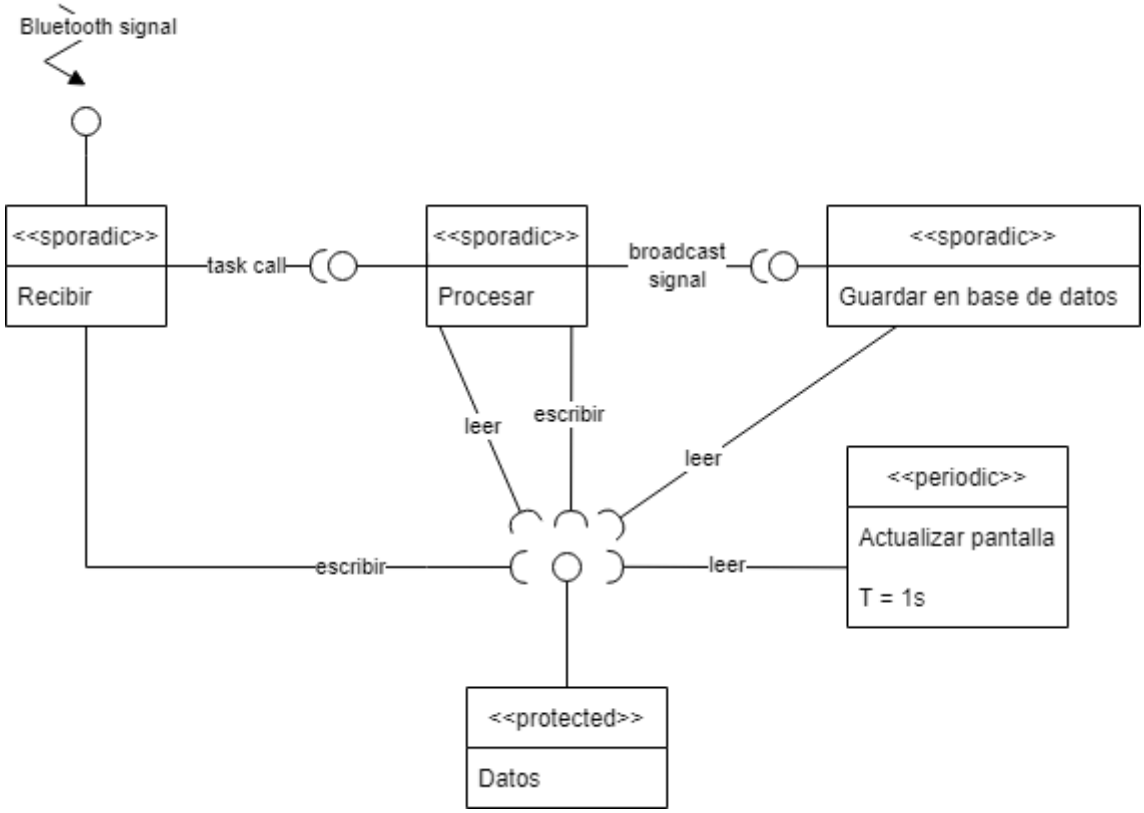

Figura 14. Esquema HRT-UML simplificado del estado de adquisición de datos de tiempo real

Las primeras 2 tareas (recepción de datos y procesamiento de datos) están en el *Service* del BLE (*BluetoothLEService*) y las otras 2 (almacenamiento de datos y actualización de pantalla) se encuentran en la *Activity* que se corresponde con este estado (*DataAcquisitionActivity*).

La tarea de recibir es implementada mediante una función de librería cuya ejecución debe finalizar lo antes posible a fin de estar disponible para la siguiente recepción de datos. Además, los datos procesados son requeridos por la *Activity* para mostrarlos en pantalla, por lo que tiene sentido que el guardado de los mismos se realice en la propia *Activity*. Por estas razones son necesarias 3 tareas esporádicas: una primera que recibe los datos y cuya ejecución debe ser lo más corta posible, otra segunda que procese los datos y los envíe del *Service* a la *Activity* y una tercera que los reciba en la *Activity* y los guarde en la base de datos (ya que la *Activity* necesita los datos pues también se encarga de mostrarlos en pantalla periódicamente).

Para realizar la comunicación entre estos dos componentes (*Service* y *Activity*) se emplea un Broadcast. Este objeto permite enviar señales que actúan como mecanismos de sincronización

y que también pueden actuar como mecanismos de comunicación, pues pueden contener variables cuyo valor se puede enviar a otros componentes de la aplicación. Es este caso, cuando los datos han sido recibidos y procesados en el *Service*, se envía una señal que contiene dichos datos y que recibe la *Activity*. Una vez recibida, se procede al almacenamiento de los datos. Todo este proceso está representado de forma esquemática en el Seudocódigo 3. A continuación, se detalla cada una de las tareas.

```
cambiarNotificacionesCaracteristicas(NOTIFIABLE);

cuandoCambieCaracteristica(caracteristica) {
      if (caracteristica == caracteristica1)
            bytesCaracteristica1 = caracteristica.obtenerValores;
      else if (caracteristica == caracteristica2)
            bytesCaracteristica2 = caracteristica.obtenerValores;
            lock;
            convertirValores();
            unlock;
            enviarDifusion("Guardar_valores");
};

alRecibirDifusion(mensaje) {
      if (mensaje == "Guardar_valores")
            lock;
            guardarValores();
            unlock;
};

repetir (
actualizarPantalla(), cada 1 segundo);
```

Seudocódigo 3. Estado de adquisición

4.2.3.1 Recepción de datos

La tarea de recepción de datos es esporádica con una separación mínima aproximada de 20 ms. La separación mínima es aproximada debido a que no se puede establecer un control sobre ella y depende enteramente de la variabilidad en el envío de los datos por parte de la CI y la recepción por parte del móvil. Esta separación mínima se corresponde con la frecuencia de envío de los datos por parte de la CI. Se activa por evento cuando el estado de alguna de las 2 características ha cambiado. Para implementarla se emplea la función de librería *onCharacteristicChanged* dentro del objeto *BluetoothGattCallback*, que se encuentra en el

Service BluetoothLEService. Hay que poner las características en *notifiable* antes de realizar la adquisición para poder recibir los datos (una de las propiedades de las características comentadas en 4.1.3 BLE).

Si la característica que ha cambiado y, por tanto, la que se recibe, es la primera, simplemente se almacenan sus valores en RAM como un *array* de bytes, el formato en el que se transmiten.

Si la característica recibida es la segunda, quiere decir que ya se han recibido todos los datos de un instante de tiempo, por lo que se señaliza inmediatamente a la tarea esporádica de procesamiento de datos.

4.2.3.2 Procesamiento de datos

Esta tarea se encarga de convertir los valores recibidos del formato *array* de bytes a los formatos correspondientes de los valores: *integer* para los valores correspondientes a la comprobación de errores de transmisión y *float* para el resto de valores (ver Tabla 2. Descripción bytes características BLE).

El acceso a estos datos debe realizarse en exclusión mutua para asegurar que no se produce una condición de carrera. Es decir, se debe evitar que se sobrescriban los valores recibidos mientras se está haciendo la conversión de los mismos. Para garantizar el acceso en exclusión mutua de los datos se utilizan los mecanismos *synchronized* y *lock* que ofrece Java. Estos recursos funcionan como monitores y a nivel interno ambos funcionan de la misma forma. Los monitores son mecanismos de comunicación y sincronización basados en variables compartidas que surgieron como solución a las limitaciones de los semáforos. Concretamente, son módulos que encierran los recursos o variables compartidas como componentes internos privados y ofrece una interfaz de acceso a ellos que garantiza el régimen de exclusión mutua, de manera que tienen asociada una lista en la que se incluyen las tareas que al tratar de acceder al monitor son suspendidas. La diferencia entre synchronized y lock estriba en que *lock* permite más versatilidad en cuanto al bloqueo y liberación de secciones críticas, por lo que se recomienda cuando una aplicación cuenta con varias secciones críticas distribuidas en varios componentes.

Al terminar la conversión de valores se envía un *Broadcast* con los valores convertidos para ser recibido por la *Activity* actualmente en ejecución: *DataAcquisitionActivity*.

4.2.3.3 Almacenamiento de datos

Una vez recibido el *Broadcast*, se ejecuta la tarea de guardado de datos. Esta tarea almacena los datos en el almacenamiento del dispositivo, concretamente la base de datos interna de SQLite. SQLite es un sistema de gestión de bases de datos de SQL que implementa Android a través de sus APIs para el fácil desarrollo de bases de datos. De nuevo, el acceso a los datos

recibidos se realiza en exclusión mutua con los mecanismos comentados en el apartado anterior.

4.2.3.4 Actualización de pantalla

Por último, la tarea de actualización de pantalla es una tarea periódica que se encarga de actualizar los datos mostrados en pantalla. Esto permite visualizar si los datos recibidos son correctos. Esta tarea tiene un periodo de 1 segundo.

4.2.4 Estado de visualización

Una vez finalizada la adquisición de datos, se accede al estado de visualización. En este estado, se ofrecen representaciones gráficas de los valores adquiridos y almacenados. También se muestran mediante texto los detalles principales de los datos: ID de usuario, iteración de la prueba, fecha, tiempo de adquisición, número de muestras y número de errores. El seudocódigo de este estado se representa en el Seudocódigo 4.

```
alSeleccionarDato(dato) {
      mostrarEnGraficas(dato)};
mostrarDetallesDatos();
```

Seudocódigo 4. Estado de visualización

4.2.5 Interfaz de usuario

Como se ha comentado, a nivel interno la aplicación consta de 7 *Activities* para cada una de las pantallas, 1 *Service* para implementar las funcionalidades necesarias del BLE, 1 extensión de la aplicación para el almacenamiento de variables globales y 1 clase que implementa la base de datos.

Las *Activities* están asociadas a las pantallas por las que puede navegar la persona usuaria de la aplicación. Una de ellas (*ConfigurationActivity*) es para acceder a la configuración y las otras 6 conforman los 2 modos de uso de la aplicación: *adquisición de datos* (*MainActivity*, *ConfigureNewUserActivity*, *DataAcquisitionActivity*, *VisualizeDataActivity* y *SaveDataActivity*) y *visualización de datos guardados* (*MainActivity*, *SelectSavedDataActivity*, *VisualizeDataActivity* y *SaveDataActivity*).

La correspondencia entre los estados y las *Activities* se muestra en la Figura 15. La navegación se hace a través de los botones disponibles en cada pantalla. Pulsar el botón físico “atrás” del móvil enviará la aplicación a un segundo plano. Todas las transiciones entre *Activities* corresponden a avanzar a la siguiente pantalla o retroceder a la anterior y no están sujetas a ningún tipo de condición, excepto las transiciones entre *MainActivity* y *ConfigureNewUserActivity* y entre *DataAcquisitionActivity* y *VisualizeDataActivity*. La primera está condicionada a la correcta conexión mediante BLE entre el móvil y la CI, y la segunda a que se haya realizado la adquisición de datos.

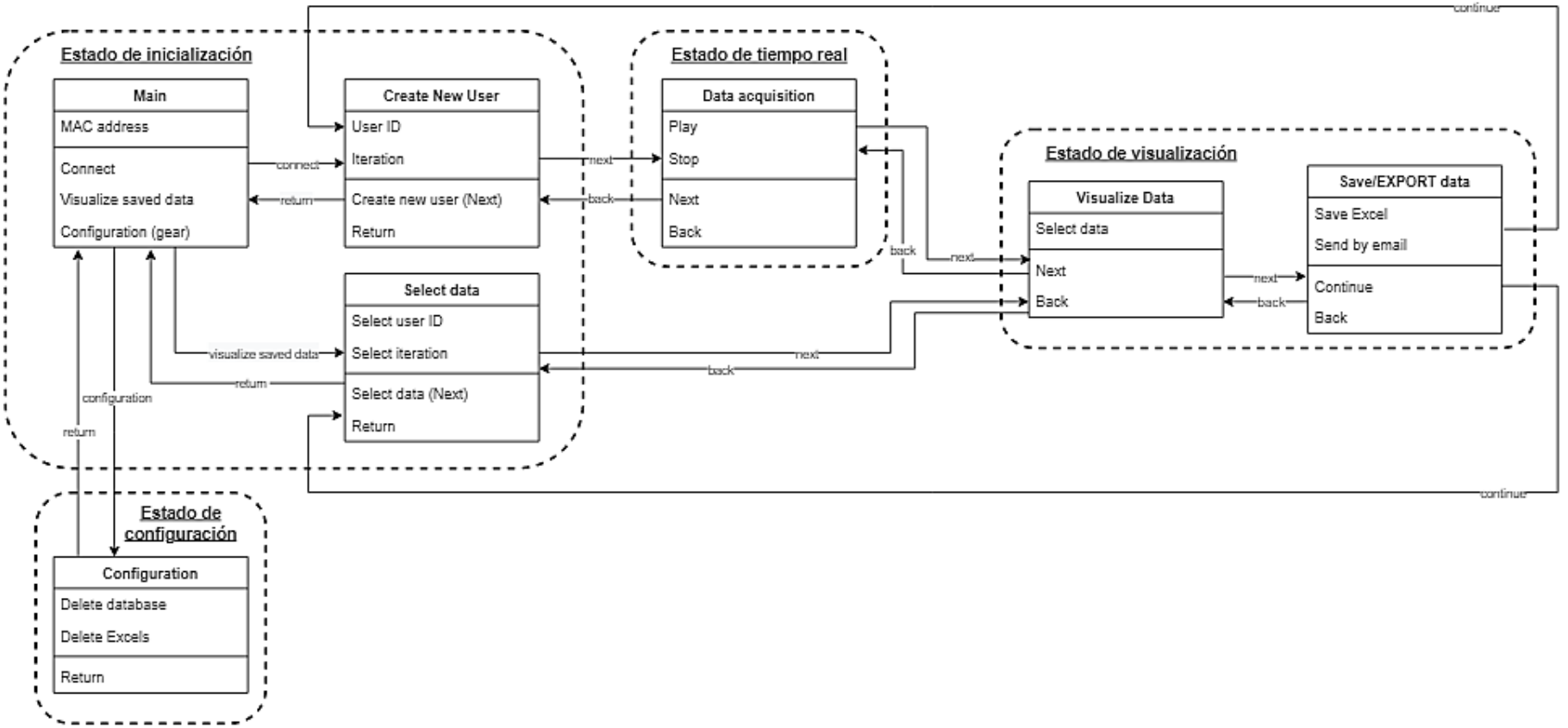

Figura 15. Correspondencia entre las *Activities* (pantallas) y los estados de la aplicación

4.2.5.1 MainActiviy

Esta *Activity* forma parte del estado de inicialización. Es la pantalla principal y se muestra cada vez que se inicia la aplicación. Al iniciar, comprueba que el Bluetooth esté activado en el dispositivo. En caso de que no lo esté, solicita a la persona usuaria que lo active mediante una ventana emergente. También comprueba que se haya concedido el permiso de ubicación a la aplicación. El permiso de ubicación es necesario para poder emplear las funcionalidades del BLE. Además, si el dispositivo tiene Android igual o superior a la versión 10.0, no solo es necesario conceder el permiso de ubicación, sino también activar la propia ubicación. Por lo tanto, se pide a la persona usuaria que active la ubicación si es necesario.

La interfaz de esta A*ctivity* (Figura 16) consta de una entrada de texto para la dirección MAC y 4 botones: *Connect*, *Select Saved Data*, *Configuration* (engranaje) y *Exit App*.

Introduciendo en la entrada de texto la dirección MAC de la CI a la que se desea conectarse y pulsando el botón Conectar, permite avanzar a la siguiente pantalla del *modo de adquisición de datos*, siempre y cuando se haya realizado la conexión a la CI de forma exitosa. También es posible desplegar una lista de las direcciones MAC guardadas pulsando el botón con forma de flecha invertida.

Por otro lado, permite acceder a la pantalla de configuración (*ConfigurationActivity*) con el botón en forma de engranaje y a la de selección de datos (*SelectSavedDataActivity*) con el botón *Select Saved Data*. Esta última es la primera pantalla del *modo de visualización*.

Por último, en la parte inferior se dispone de un botón para salir de la aplicación, que desconecta cualquier CI que estuviera conectada y cierra la aplicación.

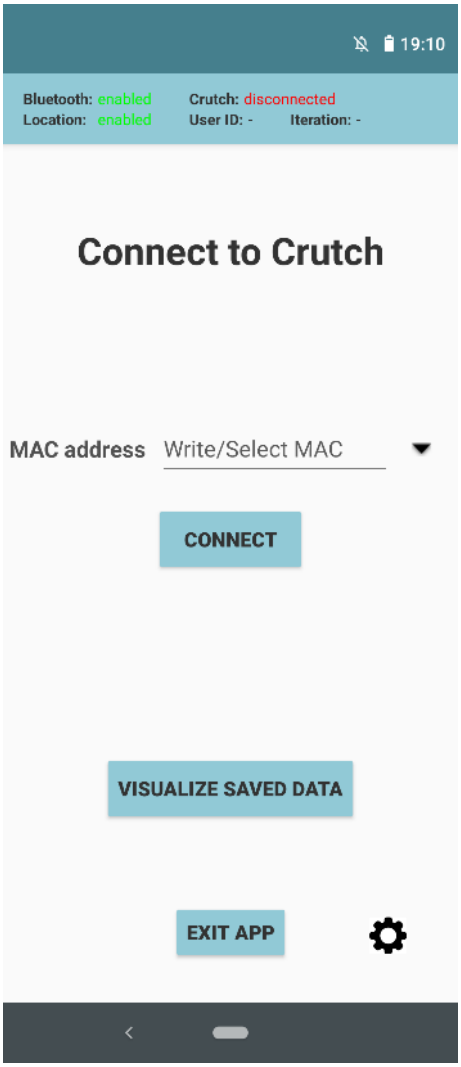

Figura 16. Interfaz de MainActivity

4.2.5.2 ConfigurationActivity

Esta *Activity* se corresponde con el estado de configuración. Se dispone de 2 botones (aparte del de navegación, *Return*, que permite volver a la pantalla principal): *Delete Database* y *Delete Excels* para borrar la base de datos y los archivos Excel, respectivamente (Figura 17).

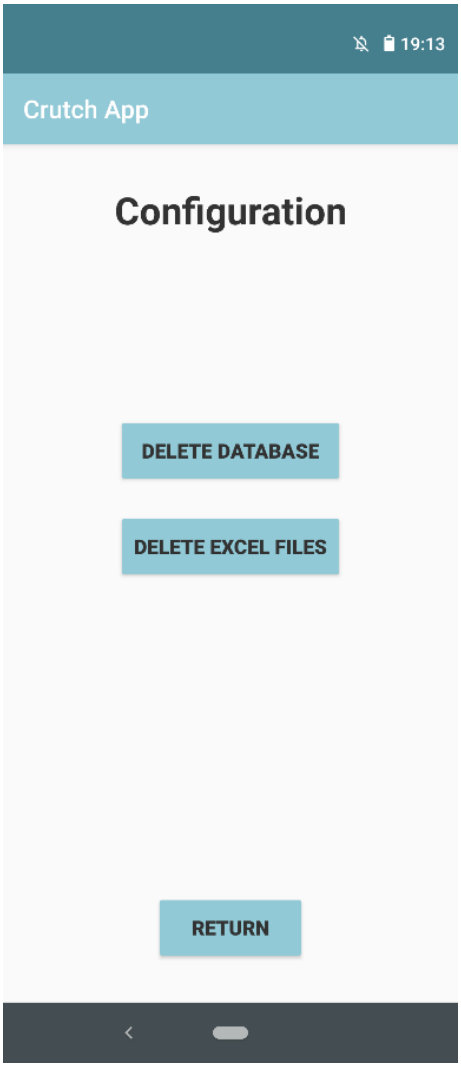

Figura 17. Interfaz de ConfigurationActivity

4.2.5.3 ConfigureNewUserActivity

Esta *Activity* es la segunda y última pantalla correspondiente al estado de inicialización para el *modo de adquisición de datos*. En ella se configuran los nuevos identificadores para la prueba: el ID de usuario y la iteración de la prueba.

La interfaz de esta *Activity* (Figura 18) consta de dos entradas de texto para el ID de usuario y la iteración de la prueba y un botón (además de los de navegación) para crear el nuevo usuario y avanzar a la siguiente pantalla.

No es posible almacenar los datos con el mismo ID de usuario e iteración 2 veces, por lo que se muestra un mensaje de aviso si los mismos valores han sido utilizados previamente, permitiendo sobrescribirlos o no hacer nada para poder modificarlos por otros que no hayan sido usados.

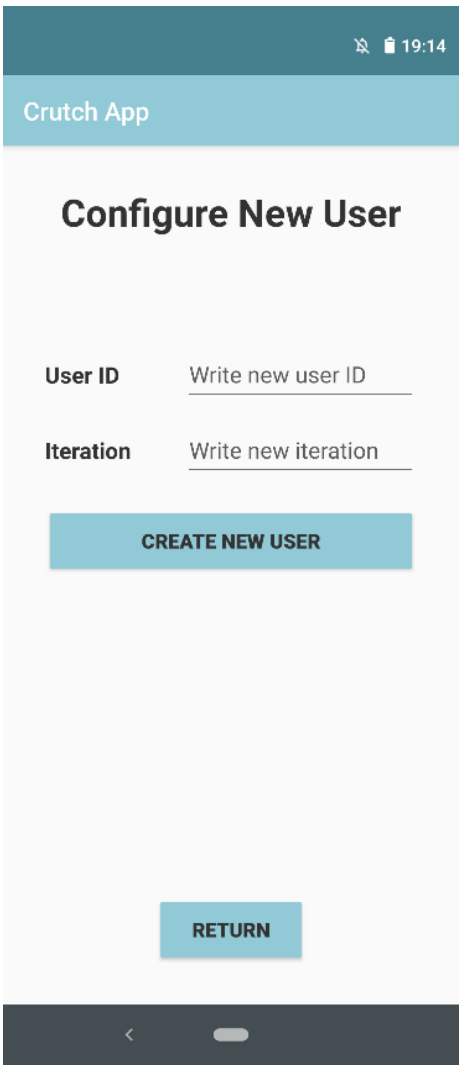

Figura 18. Interfaz de ConfigureNewUserActivity

4.2.5.4 SelectSavedDataActivity

Es la primera y única pantalla de inicialización del *modo de visualización de datos*. En esta *Activity* se pueden seleccionar los datos de un ID de usuario e iteración de prueba previamente guardados en la base de datos interna para su posterior visualización. Esta *Activity* solamente tiene sentido de uso si previamente se han adquirido datos mediante el modo de adquisición de datos.

La interfaz (Figura 19) es análoga a la del apartado anterior, cambiando las entradas de texto para el ID de usuario y la iteración de la prueba por selectores.

Si no hay datos guardados, los botones de selección se inhabilitan y el de avance a la siguiente pantalla se inhabilita y oculta, permitiendo únicamente volver a la pantalla principal.

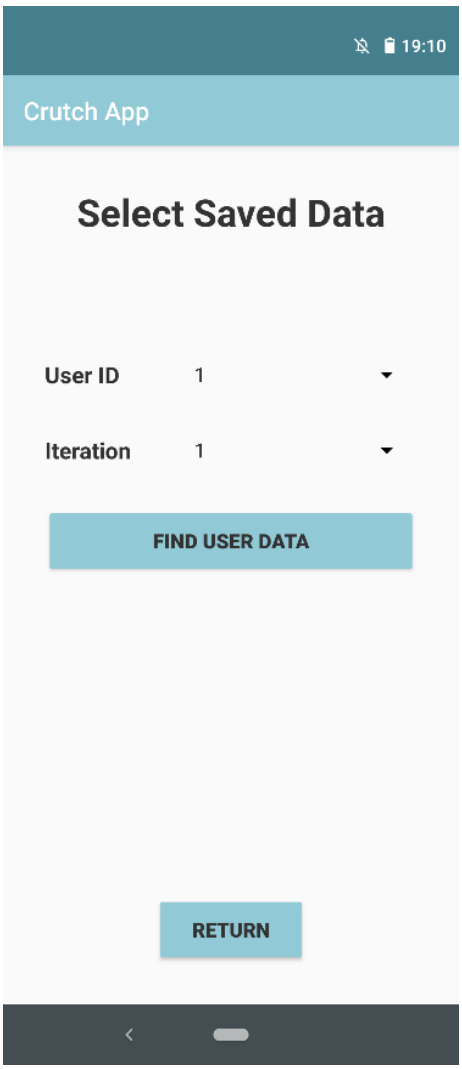

Figura 19. Interfaz de SelectSavedDataActivity

4.2.5.5 DataAcquisitionActivity

Corresponde al estado de tiempo real y es la tercera pantalla dentro del *modo de adquisición de datos*. Esta es la única *Activity* que tiene requisitos de tiempo real.

Al iniciar esta *Activity*, se activan las notificaciones de las características, lo que hace que se comiencen a recibir los datos. De esta forma, los cambios en sus datos disparan la llamada de un método asíncrono, que envía una señal para que la *Activity* guarde los datos.

La interfaz (Figura 20) consta de 14 cuadros de texto para mostrar cada uno de los valores que adquiere la muleta exceptuando los valores de iteración, que solo son para la comprobación de errores y, por tanto, no son de relevancia para la persona usuaria. Además, dispone de 3 botones (además de los de navegación) para el inicio (play), detención (stop) y reseteo (reset) de la adquisición de datos, representados mediante una cabeza de flecha verde, un cuadrado rojo y una flecha curva naranja.

Al pulsar el botón de inicio, se guarda la marca de tiempo del inicio de la prueba y comienzan a guardarse los datos en la base de datos interna. También se oculta este botón y se muestra el de detención en su lugar.

Al pulsar el botón de detención, se guarda la marca de tiempo del final de la prueba y se desactivan las notificaciones de las características, deteniéndose la adquisición de datos. También se oculta este botón y se muestra el de reseteo.

Al pulsar el botón de reseteo, se muestra un mensaje de confirmación mediante una ventana emergente. Si se confirma la acción, los datos guardados durante la adquisición se borran de la base de datos interna, se oculta el botón de reseteo y se vuelve a mostrar el de inicio para poder realizar una nueva adquisición de datos.

Si la adquisición de datos ha terminado y ha sido correcta, se puede avanzar a la siguiente *Activity*.

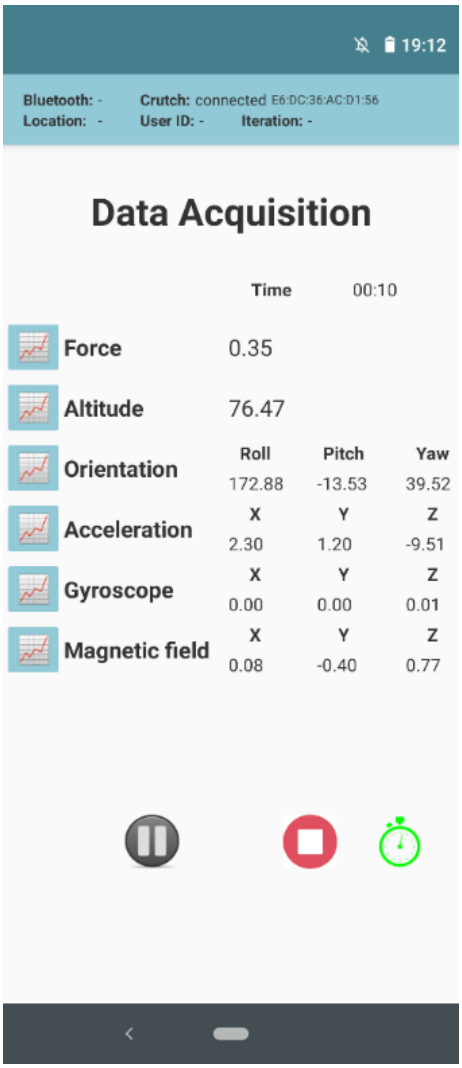

Figura 20. Interfaz de DataAcquisitionActivity

4.2.5.6 VisualizeDataActivity

Esta *Activity* corresponde a la primera pantalla del estado de visualización. Es común a los *modos de adquisición* y *visualización de datos*. Permite visualizar los datos guardados en la base de datos mediante gráficas.

La interfaz de esta *Activity* (Figura 21) consta de un selector para seleccionar el dato que se desea mostrar mediante las gráficas y de los botones de navegación. Los datos que tienen 3 ejes son representados en 3 gráficas diferentes. Solo se grafican las 3000 muestras para evitar problemas de memoria. Los valores de los ejes de las gráficas se muestran con 4 cifras.

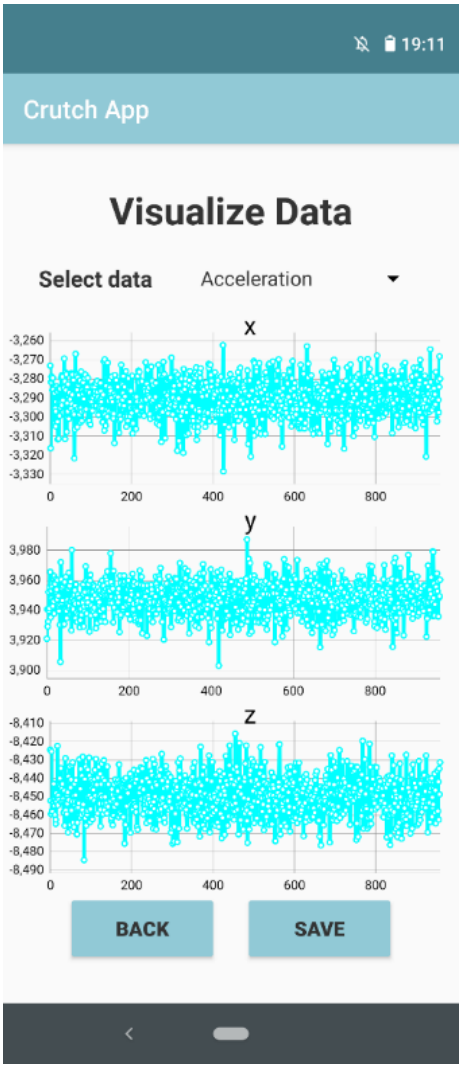

Figura 21. Interfaz de VisualizeDataActivity

4.2.5.7 SaveDataActivity

Esta *Activity* corresponde a la segunda y última pantalla del estado de visualización. Esta pantalla también es común a los *modos de adquisición* y *visualización de datos*. Su función es mostrar el resumen de las características de los datos: ID de usuario e iteración asociados,

fecha de realización de la prueba, tiempo (duración de la prueba), número de muestras y número de errores.

La interfaz de esta *Activity* (Figura 22) consta de 7 cuadros de texto necesarios para mostrar los datos y 2 botones (además de los de navegación): *Save Excel* y *Send by email*.

El botón de guardar Excel permite guardar los datos en formato Excel en la carpeta interna de la aplicación, concretamente dentro de una nueva carpeta, creándola si no existe. Cualquier archivo que existiese previamente con el mismo nombre es sobrescrito.

El botón de enviar por email envía el archivo Excel por correo electrónico, creando el archivo de forma automática si la persona usuaria no ha pulsado previamente el botón para guardarlo.

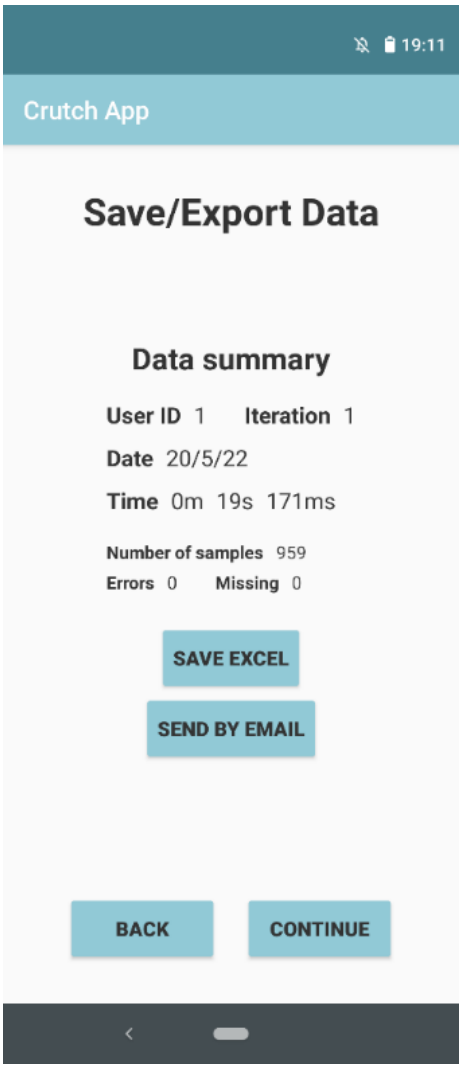

Figura 22. Interfaz de SaveDataActivity

5 RESULTADOS Y ANÁLISIS

Para validar el funcionamiento de la aplicación Android desarrollada se han realizado unas series de pruebas con 2 versiones de la CI de distintas duraciones sobre un mismo circuito. Concretamente, se han realizado adquisiciones de datos de 3, 5 y 15 minutos mientras se caminaba con el dispositivo móvil en el bolsillo y la CI acoplada a una muleta por el circuito representado en la Figura 23. Las pruebas se han realizado en el tercer piso del edificio B de la Escuela de Ingeniería de Bilbao perteneciente a la Universidad del País Vasco/Euskal Herriko Unibertsitatea (UPV-EHU). El circuito consiste en un tramo recto inicial desde el inicio del mismo hasta una columna situada en el extremo opuesto al inicio, una curva para sortear la columna, un segundo tramo recto hasta la pared más alejada del inicio, un cambio de sentido, la repetición en sentido contrario de estos tramos rectos y una última sección que rodea una columna en el lado del inicio. El circuito se repite tantas veces como sea necesario hasta alcanzar los tiempos de cada prueba (3, 5 y 15 minutos).

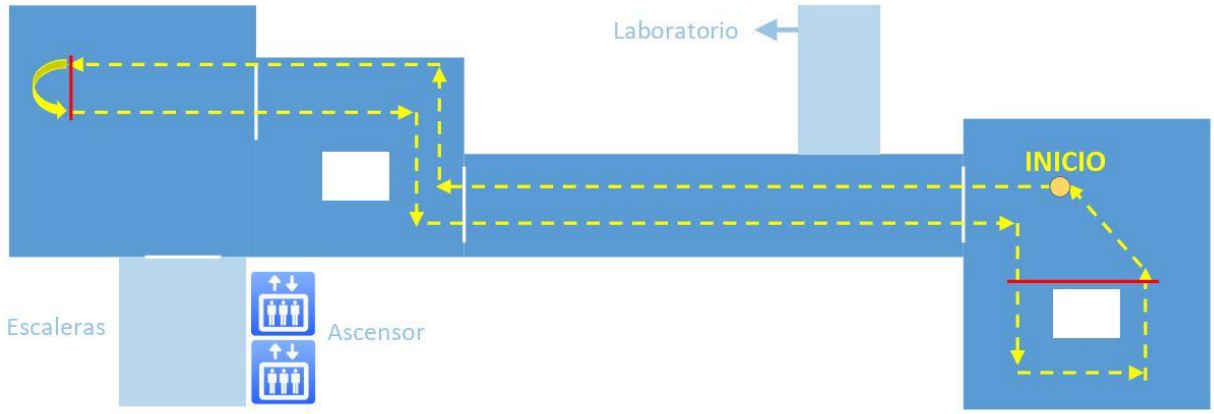

Figura 23. Circuito para las pruebas de validación

Las pruebas se han realizado con dos dispositivos móviles. Uno de ellos es el descrito en el apartado 4.1.2, el modelo One Action de Motorola con Android 10.0, siendo este el dispositivo adquirido por el grupo de investigación para el desarrollo de este trabajo. Por ello no tiene aplicaciones externas instaladas ni archivos guardados. El otro dispositivo empleado en las pruebas es el modelo Galaxy S7 de Samsung con Android 8.0 y es un móvil personal. Las características del Motorola One Action ya han sido descritas en apartado 4.1.2. Las características principales del Samsung Galaxy S7 son: 4 GB de RAM y 1.2 GB libres de almacenamiento.

El dispositivo móvil se llevaba en el bolsillo del pantalón en las pruebas. En la Figura 24 se puede observar un esquema de la ubicación del móvil y la CI al realizar las pruebas.

Las pruebas realizadas con el Motorola One Action se han llevado a cabo con la CI versión 3.2 y las realizadas con el Samsung Galaxy S7, con la CI versión 2.0.

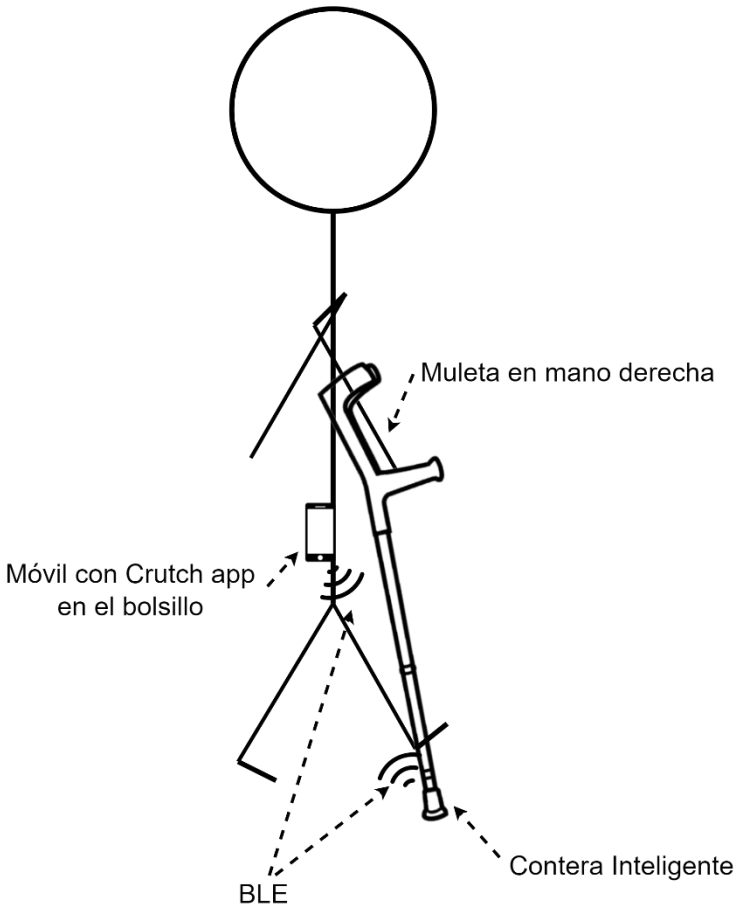

Figura 24. Esquema de la realización de las pruebas

De esta forma se pretende estudiar dos condiciones de uso de la aplicación desarrollada:

1) dispositivo sin aplicaciones externas instaladas ni en uso ni archivos almacenados (Motorola One Action)
2) dispositivo con aplicaciones instaladas y en uso y archivos almacenados (Samsung Galaxy S7).

Es decir, se va a estudiar la capacidad y calidad de los datos adquiridos por un dispositivo exclusivamente dedicado a la adquisición de datos, siendo este el Motorola One Action, y en otro dispositivo, el Samsung Galaxy S7, con el que se está haciendo un "uso normal" del dispositivo mientras se adquieren los datos. El "uso normal" del dispositivo consiste en tener una conexión a internet activa y hacer uso simultáneo de aplicaciones multimedia que requieran de conexión a internet (YouTube, Spotify) con la aplicación desarrollada para la adquisición de datos.

Adicionalmente, dentro de las pruebas realizadas con el Samsung Galaxy S7 se incluyen 2 variaciones: una con el móvil en el bolsillo del mismo lado que la CI y otra en el bolsillo opuesto.

Se han llevado a cabo 3 repeticiones de las pruebas realizadas con el Samsung Galaxy S7 y 1 repetición con el Motorola One Action, por razones de disponibilidad del dispositivo.

El resumen de las pruebas realizadas se recoge en la Tabla 3. La cantidad de datos es el número de muestras obtenidas, es decir, cada pareja de características que conforman un instante de muestreo. Como se puede observar, los errores son mínimos en todos los casos. Estos errores son calculados comparando los valores de las variables de conteo de errores (ErrorCheck 1 y 2, ver Tabla 2) en cada instante de muestreo. Se produce un error cuando estos valores no son iguales en un instante de muestreo y cuando se repite el mismo valor en dos instantes de muestreo consecutivos.

También se puede observar que los errores aumentan cuando el móvil no se encuentra en el bolsillo del mismo lado que la CI. No obstante, el porcentaje de errores respecto a la cantidad de datos se mantiene muy cerca del 0%. Además, tan solo llevando el móvil en el bolsillo del mismo lado de la CI se evitan estos errores.

Versión CI	Dispositivo móvil	Bolsillo	Tiempo	Cantidad datos	Errores
3.2	Motorola One Action	mismo lado	3 min 1 s 406 ms	9070	0
		mismo lado	5 min 0 s 935 ms	15047	0
		mismo lado	8 min 1 s 764 ms	24089	0
2.0	Samsung Galaxy S7	mismo lado	3 min 2 s 794 ms	9139	0
		mismo lado	3 min 3 s 314 ms	9166	2
		mismo lado	3 min 4 s 782 ms	9240	2
		mismo lado	5 min 1 s 645 ms	15082	0
		mismo lado	5 min 1 s 125 ms	15056	1
		mismo lado	5 min 0 s 156 ms	15007	3
		mismo lado	15 min 2 s 651 ms	45132	1
		mismo lado	15 min 0 s 874 ms	45043	0
		mismo lado	15 min 1 s 789 ms	45089	0
		lado opuesto	3 min 1 s 655 ms	9082	9
		lado opuesto	3 min 1 s 378 ms	9068	10
		lado opuesto	3 min 0 s 872 ms	9043	10
		lado opuesto	5 min 2 s 894 ms	15144	14
		lado opuesto	5 min 0 s 915 ms	15045	13
		lado opuesto	5 min 0 s 292 ms	15014	13
		lado opuesto	15 min 1 s 978 ms	45098	38
		lado opuesto	15 min 0 s 455 ms	45022	37
		lado opuesto	15 min 1 s 615 ms	45080	38

Tabla 3. Resumen resultados pruebas validación

No ha sido posible llevar a cabo la prueba de 15 minutos en el Motorola One Action ya que se produce un error de conexión entre la CI y el dispositivo alrededor de los 8 minutos desde el inicio de la adquisición de datos. Este error provoca la desconexión entre la CI y el Motorola One Action y, por lo tanto, se detiene la adquisición de datos. El error se debe a la versión de

la CI y no a la aplicación. Se ha comprobado que con el Motorola One Action y la CI versión 2.0 no se produce esta desconexión a los 8 minutos. Sin embargo, no se ha dispuesto de tiempo suficiente para realizar más pruebas y poder encontrar la causa concreta de la desconexión.

Con el Samsung Galaxy S7 las pruebas se han podido llevar a cabo sin errores de conexión. No obstante, sí que ocurre un error de conexión al finalizar la adquisición de datos y cambiar de *Activity*. Tampoco ha sido posible encontrar la causa de este error. No obstante, este error solo supone una inconveniencia en el uso de la aplicación ya que no se pierden datos (la desconexión ocurre al finalizar la adquisición de datos) y se puede restablecer la conexión volviendo a la pantalla principal y volviendo a conectarse a la CI.

6 CONCLUSIONES

El equipo de investigación ViSens, donde se engloba el proyecto SMARTIP, ha desarrollado y validado una CI que permite adquirir datos sobre el movimiento y posición de una persona que use una muleta. Concretamente, el grupo ViSens se centra especialmente en personas que padezcan EM.

Hasta el presente, la adquisición de datos se ha realizado con un programa desarrollado en C que ha de ser ejecutado en un ordenador. Esto dificulta la adquisición de datos porque, aunque se pueda realizar en un ordenador portátil, no deja de ser un dispositivo más voluminoso y pesado que una *Tablet* o un móvil.

Por ello, se ha desarrollado una aplicación móvil para dispositivos Android. De esta forma, se abre la puerta a una monitorización continua de las personas que padezcan EM. A su vez, se facilita la adquisición de datos en pruebas controladas.

Se ha validado la aplicación con pruebas de adquisición de datos con varios dispositivos en un circuito. Uno de los dispositivos, el modelo One Action de Motorola con Android 10.0, es un móvil cuyo uso es exclusivo para ejecutar la aplicación y adquirir los datos, por lo que no tiene aplicaciones externas instaladas ni archivos multimedia. El otro móvil, el modelo Galaxy S7 de Samsung con Android 8.0, es un móvil de uso personal, por lo que tiene instaladas aplicaciones externas que han sido ejecutadas a la vez que la aplicación desarrollada durante la adquisición de datos.

Los resultados obtenidos han sido prometedores con tasas de errores casi nulas. Sin embargo, se ha encontrado un problema de conexión con el móvil Motorola One Action que impide la adquisición de datos durante más de 8 minutos. Además, con el móvil Samsung Galaxy S7 la aplicación presenta un error de conexión al finalizar la adquisición de datos, aunque no supone una pérdida de los mismos.

6.1 Acciones futuras

Como trabajo futuro, es necesario solucionar los problemas de conexión descritos anteriormente para darle más robustez a la conexión entre el dispositivo móvil donde se instale la aplicación y la CI.

Por otro lado, la interfaz está en inglés, por lo que sería conveniente agregar traducciones de la misma mediante las herramientas que ofrece Android a la hora de desarrollar aplicaciones.

7 REFERENCIAS BIBLIOGRÁFICAS

[1] «EMDATA - Esclerosis Múltiple España (EME),» [En línea]. Available: https://emdata.esclerosismultiple.com/atlas-de-la-em/. [Último acceso: 2022].

[2] K. Bjornevik, . M. Cortese, B. C. Healy, J. Kuhle, M. J. Mina, Y. Leng, S. J. Elledge, D. W. Niebuhr, A. I. Scher, K. L. Munger y A. Ascherio, «Longitudinal analysis reveals high prevalence of Epstein-Barr virus associated with multiple sclerosis,» *SCIENCE,* vol. 375, nº 6578, pp. 296-301, 2022.

[3] A. Souza, A. Kelleher, R. Cooper, R. A. Cooper, L. I. Iezzoni y D. M. Collins, «Multiple sclerosis and mobility-related assistive technology: Systematic review of literature,» *Journal of Rehabilitation Research & Development,* vol. 47, pp. 213-224, 2010.

[4] M. Gurevich, T. Tuller, U. Rubinstein, R. Or-Bach y A. Achiron, «Prediction of acute multiple sclerosis relapses by transcription levels of peripheral blood cells,» *BMC medical genomics,* vol. 2, 2009.

[5] O. Pearson, M. Busse, R. van deursen y C. M. Wiles, «Quantification of walking mobility in neurological disorders,» *QJM,,* vol. 97, pp. 463-475, 2004.

[6] F. Bethoux y S. Bennett, «Evaluating Walking in Patients with Multiple Sclerosis. Which Assessment Tools Are Useful in Clinical Practice?,» *International Journal of Multiple Sclerosis Care,* vol. 13, pp. 4-14, 2011.

[7] «StatCounter - Global Stats,» [En línea]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide. [Último acceso: 2022].

[8] S. H. Scott y S. P. Dukelow, «Potential of robots as next-generation technology for clinical assessment of neurological disorders and upper-limb therapy,» *Journal of Rehabilitation Research & Development,* vol. 48, nº 4, pp. 335-354, 2011.

[9] O. Lambercy, S. Maggioni, L. Lünenburger, R. Gassert y M. Bolliger, «Robotic and Wearable Sensor Technologies for Measurements/Clinical Assessments,» *Neurorehabilitation Technology,* pp. 255-269, 2016.

[10] P. B. Shull, W. Jirattigalachote, M. A. Hunt, M. R. Cutkosky y S. L. Delp, «Quantified self and human movement: A review on the clinical impact of wearable sensing and feedback for gait analysis and intervention,» *Gait & Posture,* vol. 40, pp. 11-19, 2014.

[1] M. Weikert, Y. Suh, A. Lane, B. Sandroff, D. Dlugonski, B. Fernhall y R. W. Motl, «Accelerometry is associated with walking mobility, not physical activity, in persons with multiple sclerosis,» *Medical Engineering & Physics,* vol. 34, pp. 590-597, 2012.

[2] M. Shoaib, S. Bosch, O. Durmaz Incel, H. Scholten y P. J. M. Havinga, «Fusion of Smartphone Motion Sensor for Physical Activity Recognition,» *Sensors (Basel),* vol. 14, nº 6, pp. 10146-10176, 2014.

[3] A. Pérez Ruiz, M. Aldea Rivas y M. González Harbour, *Soporte de aplicaciones de tiempo real en el sistema operativo Android,* 2016.

[4] «Android Developers,» [En línea]. Available: https://developer.android.com/guide/platform#:~:text=Android%20is%20an%20open%20source,components%20of%20the%20Android%20platform. [Último acceso: 2022].

[5] «Anand Tech,» [En línea]. Available: https://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l/. [Último acceso: 2022].

[6] «Geeks for Geeks,» [En línea]. Available: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/. [Último acceso: 2022].

[7] «Android Source,» [En línea]. Available: https://source.android.com/setup/start/build-numbers. [Último acceso: 2022].

[8] C. Maia, L. M. Nogueira y L. M. Pinho, «Evaulating Android OS for Embedded Real-Time Systems,» *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications,* pp. 63- 70, 201.

[9] W. Mauerer, G. Hillier, J. Sawallisch, S. Hönick y S. Oberthür, «Real-time android: deterministic ease of use,» *Proceedings of the Embedded Linux Conference Europe,* 2012.

[20] F. Reghenzani, G. Massari y W. Fornaciari, «The RealTime Linux Kernel: A Survey on PREEMPT_RT,» *ACM Computing Surveys,* 2019.

[21] I. Kalkov, D. Franke, J. F. Schommer y S. Kowalewski, «A real-time extension to the Android platform,» *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems,* pp. 105-114, 2012.

[22] I. Kalkov, A. Gurghian y S. Kowalewski, «Priority Inheritance during Remote Procedure Calls in Real-Time Android using Extended Binder Framework,» *Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems,* 2015.

[23] Y. Yan, S. Cosgrove, . V. Anand y A. Kulkarn, «Real-Time Android with RTDroid,» *Proceedings of the 12th International Conference on Mobile Systems, Applications, and Services,* 2014.

[24] Y. Yan, S. Harsha Konduri, A. Kulkarni, V. Anand, S. Y. Ko y L. Ziarek, «RTDroid: A Design for Real-Time Android,» *Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems,* 2013.

[25] D. Buttlar, J. Farrell y B. Nichols, Pthreads programming: A POSIX standard for better multiprocessing, O'Reilly Media, Inc, 1996.

[26] «Motorola,» [En línea]. Available: https://www.motorola.com/us/smartphones-motorola-one-action/p?skuId=303. [Último acceso: 2022].

[27] «Bluetooth,» [En línea]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. [Último acceso: 2022].

[28] J. Tosi, F. Taffoni, M. Santacatterina , R. Sannino y D. Formica, «Performance Evaluation of Bluetooth Low Energy: A Systematic Review,» *Sensors,* 2017.

ANEXO I: Código original

En este anexo se encuentra el código escrito en Java correspondiente a cada uno de los elementos mencionados y descritos en el apartado 4.2.5 (7 *Activities* para cada una de las pantallas, 1 *Service* para implementar las funcionalidades necesarias del BLE, 1 extensión de la aplicación para el almacenamiento de variables globales y 1 clase que implementa la base de datos) además del archivo Manifest.xml, donde se especifican los permisos necesarios para el funcionamiento de la aplicación.

Solo se ha incluido el código de los archivos principales, es decir, los que implementan la lógica de la aplicación. No se han incluido los archivos de interfaz escritos en XML ni los archivos auxiliares que contienen otros recursos como los colores, dimensiones, etc. empleados en las interfaces.

Los archivos incluidos en este anexo son los siguientes:

- MainActivity.java → Código de la pantalla principal que aparece al iniciar la aplicación
- ConfigureNewUserActivity.java → Código de la pantalla en la que se crea un nuevo usuario mediante su identificador (ID) y número de prueba (iteration)
- SelectSavedDataActivity.java → Código de la pantalla para seleccionar los datos de una prueba guardada en la base de datos
- DataAcquisitionActivity.java → Código de la pantalla
- VisualizeDataActivity.java → Código de la pantalla
- SaveDataActivity.java → Código de la pantalla
- ConfigurationActivity.java → Código de la pantalla de configuración
- AdminSQLiteOpenHelper.java → Código que implementa las funcionalidades de la base de datos interna
- BluetoohLEService.java → Código que implementa las funcionalidades necesarias del BLE
- CrutchApplication.java → Código que extiende la clase *Application* para almacenar variables globales compartidas entre los archivos correspondientes a la implementación de la lógica de las *Activities*
- Manifest.xml → Código necesario en cualquier aplicación Android donde se especifican los permisos necesarios para el funcionamiento de la aplicación y otras características como el nombre, el icono, etc.

MainActivity.java

```
package com.visens.crutch_v2;

// AUTHOR: Carlos Lapuente

// *** NOTE_1 (about ActivityResultLauncher):
// The previous way was to use ActivityCompat.requestPermissions(Activity
```

```
activity,
//  String[] permissions, int REQUEST_CODE) and then override the
onRequestPermissionsResult(
//  int requestCode, String permissions[], int[] grantResults) method in
your activity
// Steps for newest method:
// 1) In your activity or fragment's initialization logic (onCreate), pass
in an implementation
//  of ActivityResultCallback into a call to registerForActivityResult()
and keep a reference
//  to the return value of registerForActivityResult(), which is of type
ActivityResultLauncher.
// 2) To display the system permissions dialog when necessary, call the
launch() method on the
//  instance of ActivityResultLauncher that you saved in the previous step.
// 3) After launch() is called, the system permissions dialog appears. When
the user makes a
//  choice, the system asynchronously invokes your implementation of
ActivityResultCallback,
//  which you defined in the previous step.
// https://developer.android.com/training/permissions/requesting#allow-
system-manage-request-code

// TODO [future] maximum supported Android version is 11 (API level 30).
Maybe still works with
//  Android 12 (API levels 31 and 32) but it's not checked. Try changing it
in Project Structure for
//  tests in Android 12 devices

// TODO [future] use Snackbar instead of Toast for brief info messages.
Toast is always visible
//  for a fixed amount of time (even if the app is minimized or closed) and
appears at the
//  bottom of the screen. Snackbar is more customizable and only appears
when the Activity that
//  created it is visible


/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import static android.bluetooth.BluetoothAdapter.STATE_CONNECTED;
import static android.bluetooth.BluetoothAdapter.STATE_DISCONNECTED;

import android.Manifest;
import android.bluetooth.BluetoothAdapter;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.location.LocationManager;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Looper;
import android.util.Log;
```

```
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.ActionBar;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class MainActivity extends AppCompatActivity {

    /* ////////////////////////////////////   VARIABLES
//////////////////////////////////////// */
    private static boolean initialized = false;

    // Database helper class
    private AdminSQLiteOpenHelper sqliteAdmin;

    // Layout variables
    private AutoCompleteTextView autoCompleteTextViewMACAddress;
    private ImageButton imageButtonSpinnerSelectMAC;
    private Button buttonConnect, buttonSelectSavedData;
    private ImageButton buttonConfiguration;
    private Button buttonExitApp;
    // Action Bar
    private TextView textViewBluetoothState;
    private TextView textViewLocationState;
    private TextView textViewCrutchState;
    private TextView textViewMacAddressValue;
    private TextView textViewUserIDValue;
    private TextView textViewIterationValue;

    // This is the newest way to implement permission(s) request which
allows the system to manage
    //  the request code that's associated with a permissions request and
uses androidx libraries.
    // *** See NOTE_1 at the beginning of the file for more information
    ActivityResultLauncher<String[]> locationPermissionRequest;

    // Bluetooth local variables and constants
    private String tempMacAddress;
    private BluetoothLeService mBluetoothLeService;
    // TODO these variables should go in BluetoothLeService
    private boolean bluetoothActive = false;
    private boolean locationActive = false;
    private static List<String> storedMACAddressesList;
    private static final int CONNECTION_ATTEMPT_TIME = 5000;  // 5s (for
handler.postDelayed)
```

```
    // Handler for bluetooth connection attempt time limit
    private final Handler connectionHandler = new
Handler(Looper.getMainLooper());

    // Runnable for Handler for bluetooth connection attempt time limit
    private final Runnable connectionRunnable = new Runnable() {
        @Override
        public void run() {
            Log.d("ble-debug", "runnable finished");
            if (BluetoothLeService.mConnectionState == STATE_CONNECTED &&

!tempMacAddress.equals(BluetoothLeService.mBluetoothDeviceAddress)) {
                mBluetoothLeService.stopScanning();
                // Message for user
                Toast.makeText(MainActivity.this, "Connection could not be
" +
                        "made. Attempt timeout.",
Toast.LENGTH_SHORT).show();
            } else if (BluetoothLeService.mConnectionState ==
STATE_DISCONNECTED) {
                Log.d("ble-debug", "runnable: state disconnected");
                // TODO
                updateConnectionStateActionBar();
                Log.d("ble-debug", "Scan stop requested");
                mBluetoothLeService.stopScanning();
                // Message for user
                Toast.makeText(MainActivity.this, "Connection could not be
" +
                        "made. Attempt timeout.",
Toast.LENGTH_SHORT).show();
            }
            Log.d("ble-debug", "runnable: connection state = " +
mBluetoothLeService.mConnectionState);
        }
    };

    // ServiceConnection object to manage service lifecycle
    private final ServiceConnection mServiceConnection = new
ServiceConnection() {
        // When Bluetooth service bound, initialize bluetooth objects and
if initialization
        // correct, check if Bluetooth and location enabled
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder
service) {
            mBluetoothLeService = ((BluetoothLeService.LocalBinder)
service).getService();
            if (!mBluetoothLeService.initializeBluetoothObjects()) {
                Log.e("ble-debug", "Unable to initialize Bluetooth");
            } else {
                Log.d("ble-debug", "Bluetooth objects successfully
initialized in Main");
                // Check Bluetooth and location enabled
                bluetoothActive =
mBluetoothLeService.checkBluetoothEnabled(MainActivity.this);
                // location active only needed if Android version >= 10
(API level 29)
                if (Build.VERSION.SDK_INT <= Build.VERSION_CODES.P) {
                    locationActive = true;
                } else {
```

```
                    locationActive =
mBluetoothLeService.checkLocationEnabled(MainActivity.this);
                }
            }
            // Update ActionBar textViews
            updateBluetoothStateActionBar();
            updateLocationStateActionBar();
            updateConnectionStateActionBar();
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            mBluetoothLeService = null;
        }
    };

    // BroadcastReceiver that handles various events fired by the Service
    private final BroadcastReceiver mGattUpdateReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            switch (action) {
                case BluetoothLeService.ACTION_GATT_CONNECTED:
                    // TODO
                    updateConnectionStateActionBar();
                    connectionHandler.removeCallbacks(connectionRunnable);
                    break;
                case BluetoothLeService.ACTION_GATT_DISCONNECTED:
                    // TODO
                    updateConnectionStateActionBar();
                    break;
                case BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED:
                    // Wait until services are discovered to go to next
activity
                    goToNextActivity();
                    break;
                // TODO these actions should go in BluetoothLeService (o en
CrutchApplication a una mala)
                case BluetoothAdapter.ACTION_STATE_CHANGED:
                    int state =
intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, -1);
                    if (state == BluetoothAdapter.STATE_ON) {
                        bluetoothActive = true;
                    } else if (state == BluetoothAdapter.STATE_OFF) {
                        bluetoothActive = false;
                    }
                    // TODO
                    updateBluetoothStateActionBar();
                    break;
                case LocationManager.MODE_CHANGED_ACTION:
                    if (Build.VERSION.SDK_INT <= Build.VERSION_CODES.P) {
                        // If Android version <= 9 (API level 28) location
is not needed
                        locationActive = true;
                    } else if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.R) {
                        // If Android version >= 11 (API level 30)
                        locationActive =
intent.getBooleanExtra(LocationManager.EXTRA_LOCATION_ENABLED,
                                false);
```

```
                    } else {
                        // If Android version >= 9 (API level 28) this
method is available but note
                        // that the only case left is Android version = 10
                        // TODO [future?] not tested
                        LocationManager locationManager = (LocationManager)

getApplicationContext().getSystemService(Context.LOCATION_SERVICE);
                        locationActive =
locationManager.isLocationEnabled();
                    }
                    // TODO
                    updateLocationStateActionBar();
                    break;
            }
        }
    };

    // TODO update ActionBar textViews methods maybe better in
CrutchApplication
    public void updateBluetoothStateActionBar() {
        if (bluetoothActive) {
            textViewBluetoothState.setText(R.string.textTVEnabled);
            textViewBluetoothState.setTextColor(Color.GREEN);
        } else {
            textViewBluetoothState.setText(R.string.textTVDisabled);
            textViewBluetoothState.setTextColor(Color.RED);
        }
    }

    public void updateLocationStateActionBar() {
        if (locationActive) {
            textViewLocationState.setText(R.string.textTVEnabled);
            textViewLocationState.setTextColor(Color.GREEN);
        } else {
            textViewLocationState.setText(R.string.textTVDisabled);
            textViewLocationState.setTextColor(Color.RED);
        }
    }

    public void updateConnectionStateActionBar() {
        if (BluetoothLeService.mConnectionState == STATE_CONNECTED) {
            textViewCrutchState.setText(R.string.textTVConnected);
            textViewCrutchState.setTextColor(Color.GREEN);
            textViewMacAddressValue.setText(String.format("(%s)",
                    BluetoothLeService.mBluetoothDeviceAddress));
        } else if (BluetoothLeService.mConnectionState ==
STATE_DISCONNECTED) {
            textViewCrutchState.setText(R.string.textTVDisconnected);
            textViewCrutchState.setTextColor(Color.RED);
            textViewMacAddressValue.setText("");
        }
    }

    public void updateUserIdIterationActionBar() {
        if (CrutchApplication.userId != null) {
            textViewUserIDValue.setText(CrutchApplication.userId);
        } else {

textViewUserIDValue.setText(R.string.textTVDefaultValueActionBar);
        }
```

```
        if (CrutchApplication.iteration != null) {
            textViewIterationValue.setText(CrutchApplication.iteration);
        } else {

textViewIterationValue.setText(R.string.textTVDefaultValueActionBar);
        }
    }


    /* ////////////////////////////////////   METHODS
//////////////////////////////////////// */

    /******************************  makeGattUpdateIntentFilter
*****************************
     * Register for BroadcastReceiver */
    private static IntentFilter makeGattUpdateIntentFilter() {
        final IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);

intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);

intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
        intentFilter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
        intentFilter.addAction(LocationManager.MODE_CHANGED_ACTION);
        return intentFilter;
    }

    /*************************************  ON CREATE
*************************************
     * Finds layout objects and binds BluetoothLe Service with this
activity.
     * NOTE: to prevent Service from stopping (when an activity finishes
and unbinds) and thus
     * deleting its Bluetooth objects, it's started in CrutchApplication */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity_layout);
        initialized = false;
        // Find layout objects
        autoCompleteTextViewMACAddress = findViewById(R.id.idACTVMAC);
        imageButtonSpinnerSelectMAC = findViewById(R.id.idIBSelectMAC);
        buttonConnect = findViewById(R.id.idBConnect);
        buttonSelectSavedData = findViewById(R.id.idBSelectSavedData);
        buttonConfiguration = findViewById(R.id.idIBConfiguration);
        buttonExitApp = findViewById(R.id.idBExitApp);
        // Set custom ActionBar and get its textViews

Objects.requireNonNull(getSupportActionBar()).setDisplayOptions(ActionBar.D
ISPLAY_SHOW_CUSTOM);
        getSupportActionBar().setDisplayShowCustomEnabled(true);
        getSupportActionBar().setCustomView(R.layout.action_bar_layout);
        View viewActionBar = getSupportActionBar().getCustomView();
        textViewBluetoothState =
viewActionBar.findViewById(R.id.idTVBluetoothValue);
        textViewLocationState =
viewActionBar.findViewById(R.id.idTVLocationValue);
        textViewCrutchState =
viewActionBar.findViewById(R.id.idTVCrutchConnectionValue);
        textViewMacAddressValue =
viewActionBar.findViewById(R.id.idTVMacActionBarValue);
```

```
        textViewUserIDValue =
viewActionBar.findViewById(R.id.idTVUserIDActionBarValue);
        textViewIterationValue =
viewActionBar.findViewById(R.id.idTVIterationActionBarValue);
        // Bind Bluetooth LE service
        Intent gattServiceIntent = new Intent(this,
BluetoothLeService.class);
        bindService(gattServiceIntent, mServiceConnection,
BIND_AUTO_CREATE);
        // Get location permissions ActivityResultLauncher with
registerForActivityResult and set
        // ActivityResultContracts callback
        locationPermissionRequest =
                registerForActivityResult(new ActivityResultContracts
                                .RequestMultiplePermissions(), result -> {
                            Boolean fineLocationGranted =
result.getOrDefault(

Manifest.permission.ACCESS_FINE_LOCATION, false);
                            Boolean coarseLocationGranted =
result.getOrDefault(

Manifest.permission.ACCESS_COARSE_LOCATION, false);
                            if (fineLocationGranted != null &&
fineLocationGranted) {
                                Log.d("ble-debug", "fine location
permission granted");
                            } else if (coarseLocationGranted != null &&
coarseLocationGranted) {
                                Log.d("ble-debug", "coarse location
permission granted");
                            } else {
                                Log.d("ble-debug", "no location permission
granted");
                            }
                        }
                );
        // Ask user for permissions if not granted
        if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_COARSE_LOCATION) ==
                PackageManager.PERMISSION_DENIED) {
            if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                    Manifest.permission.ACCESS_COARSE_LOCATION)) {
                // Provide an additional rationale to the user if the
permission was not granted
                // and the user would benefit from additional context for
the use of the permission.
                // Display a SnackBar with cda button to request the
missing permission.
                /*
                Snackbar.make(mLayout, R.string.camera_access_required,
                        Snackbar.LENGTH_INDEFINITE).setAction(R.string.ok,
new View.OnClickListener() {
                    @Override
                    public void onClick(View view) {
                        // Request the permission

ActivityCompat.requestPermissions(MainActivity.this,
                                new String[]{Manifest.permission.CAMERA},
                                PERMISSION_REQUEST_CAMERA);
                    }
```

```
                }).show();
                */
                // TODO [future] better message for user explaining that
Bluetooth requires location
                //  permission to scan devices and that scan is necessary
for connection
                Toast.makeText(this, "Must grant location permission for
Bluetooth",
                        Toast.LENGTH_SHORT).show();
            } /*else {
                // You can directly ask for the permission.
                // The registered ActivityResultCallback gets the result of
this request.
            }*/
            locationPermissionRequest.launch(new String[]{
                    Manifest.permission.ACCESS_FINE_LOCATION,
                    Manifest.permission.ACCESS_COARSE_LOCATION
            });
        }
    }

    /*****************************************  ON START
*************************************
     * Resets necessary global variables, gets instance of database object,
saves current crutch's
     * MAC addresses in database and calls the implemented methods */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Register Receiver
            registerReceiver(mGattUpdateReceiver,
makeGattUpdateIntentFilter());
            // Reset global app variables
            // "user" table related variables
            CrutchApplication.rowIDUserTable = -1;
            CrutchApplication.userId = null;
            CrutchApplication.iteration = null;
            CrutchApplication.date = null;
            CrutchApplication.time = null;
            CrutchApplication.numSamples = -1;
            CrutchApplication.numErrors = -1;
            CrutchApplication.numMissing = -1;
            // Initialize database variable
            sqliteAdmin = AdminSQLiteOpenHelper.getInstance(this);
            // Manually create an arraylist of the MAC addresses of the
crutches we currently have
            // NOTE: This could be done on the onCreate method of
CrutchApplication
            if (!CrutchApplication.appInitialized) {
                storedMACAddressesList = new ArrayList<>();
                storedMACAddressesList.add("E5:73:AD:CD:D2:2A");  // crutch
v3.2
                storedMACAddressesList.add("FF:C0:B0:D1:2B:D1");  // crutch
v3.1
                storedMACAddressesList.add("E6:DC:36:AC:D1:56");  // crutch
v2
                // Save MAC addresses ArrayList in database if not
duplicated
                for (int i = 0; i < storedMACAddressesList.size(); i++) {
                    boolean duplicatedMAC =
```

```
sqliteAdmin.isMacDuplicated(storedMACAddressesList.get(i));
                    if (!duplicatedMAC) {

sqliteAdmin.saveMacAddress(storedMACAddressesList.get(i));
                    }
                }
                CrutchApplication.appInitialized = true;
            }
            // TODO
            updateUserIdIterationActionBar();
            // Method calls
            SelectMACControls();
            ConnectButton();
            SelectSavedDataButton();
            ConfigurationButton();
            ExitAppButton();
            initialized = true;
        }
    }

    /*************************************  ON DESTROY
**************************************
     * Unbinds the BluetoothLe Service
     * NOTE: to prevent Service from stopping (when an activity finishes
and unbinds) and thus
     * deleting its Bluetooth objects, it's started in CrutchApplication */
    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mGattUpdateReceiver);
        unbindService(mServiceConnection);
        mBluetoothLeService = null;
    }

    /*************************************  ON ACTIVITY RESULT
**********************************
     * Handles the result (user's answer) of activities. It's used to
handle the result of asking
     * the user to enable Bluetooth  */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
intent) {
        // Result of asking to enable Bluetooth
        // NOTE !bluetoothActive check prevents bugs
        if (!bluetoothActive) {
            if (requestCode == BluetoothLeService.BT_REQUEST_CODE) {
                if (resultCode == RESULT_OK) {
                    Toast.makeText(this, "Bluetooth active",
                            Toast.LENGTH_SHORT).show();
                    bluetoothActive = true;
                } else if (resultCode == RESULT_CANCELED) {
                    Toast.makeText(this, "You must activate Bluetooth to
connect",
                            Toast.LENGTH_SHORT).show();
                    bluetoothActive = false;
                    updateBluetoothStateActionBar();
                }
            }
        }
        // Result of asking to enable Location
        // NOTE !locationActive check prevents bugs
```

```
        if (!locationActive) {
            if (requestCode == BluetoothLeService.LOCATION_REQUEST_CODE) {
                if (resultCode == RESULT_OK) {
                    Toast.makeText(this, "Location active",
                            Toast.LENGTH_SHORT).show();
                    // TODO probably need to refresh location
enabled/disabled
                    locationActive = true;
                    updateLocationStateActionBar();
                } else if (resultCode == RESULT_CANCELED) {
                    Toast.makeText(this, "You must activate location to
connect",
                            Toast.LENGTH_SHORT).show();
                    locationActive = false;
                    updateLocationStateActionBar();
                }
            }
        }
        super.onActivityResult(requestCode, resultCode, intent);
    }

    /**********************************  SELECT MAC CONTROLS
**********************************
     * Configures auto complete text view for MAC address and arrow button
to show all stored MAC
     * addresses to select.
     * 1) Creates arraylist of MAC addresses from the database
     * 2) Creates ArrayAdapter for autoCompleteTextViewMACAddress
     * 3) Sets onClickListener of imageButtonSpinnerSelectMAC to show
dropdown of
     * autoCompleteTextViewMACAddress
     * NOTE: This works only partially because when some text is written in
     * autoCompleteTextViewMACAddress, imageButtonSpinnerSelectMAC only
shows the items that match
     * what is written (because is an auto complete text view) */
    private void SelectMACControls() {
        storedMACAddressesList = sqliteAdmin.readMacAddresses();
// create arraylist of MAC addresses from database
        ArrayAdapter<String> myArrayAdapter = new ArrayAdapter<>(this,
// create ArrayAdapter for autoCompleteTextViewMACAddress
                android.R.layout.simple_dropdown_item_1line,
storedMACAddressesList);
        autoCompleteTextViewMACAddress.setThreshold(1);
// one character typed enables dropdown
        autoCompleteTextViewMACAddress.setAdapter(myArrayAdapter);
        imageButtonSpinnerSelectMAC.setOnClickListener(v ->
                autoCompleteTextViewMACAddress.showDropDown());
    }


    /**********************************  CONNECT BUTTON
**********************************
     * Button to connect phone with BLE device (crutch) via the entered MAC
address and switch to
     * first activity in data acquisition cycle (ConfigureNewUserActivity)
if connection successful */
    private void ConnectButton() {
        buttonConnect.setOnClickListener(v -> {
            // Get MAC address from input field and covert to uppercase
            tempMacAddress =
autoCompleteTextViewMACAddress.getText().toString().toUpperCase();
```

```
            // Check Bluetooth active, location active and if already
connected to the same MAC
            // address as the one entered by the user (tempMacAddress)
NOTE: if it's empty is also
            // valid (this makes it easier fo the user to use the app)
            if (!bluetoothActive) {
                bluetoothActive =
mBluetoothLeService.checkBluetoothEnabled(this);
            } else if (!locationActive) {
                locationActive =
mBluetoothLeService.checkLocationEnabled(this);
            } else if (mBluetoothLeService.mConnectionState ==
STATE_CONNECTED &&
                    (tempMacAddress.isEmpty() ||

tempMacAddress.equals(BluetoothLeService.mBluetoothDeviceAddress))) {
                // NOTE: remove callback to prevent crash from
mBluetoothLeService being null
                connectionHandler.removeCallbacks(connectionRunnable);
                goToNextActivity();
            } else {
                // Check MAC address not empty and valid (6 hexadecimal
numbers separated with colons (:))
                if (tempMacAddress.isEmpty()) {
                    Toast.makeText(this, "Introduce a MAC address",
                            Toast.LENGTH_SHORT).show();
                } else if
(!BluetoothAdapter.checkBluetoothAddress(tempMacAddress)) {
                    Toast.makeText(this, "Introduce a valid MAC address",
                            Toast.LENGTH_SHORT).show();
                } else {
                    boolean connectAttempt =
mBluetoothLeService.connectDevice(tempMacAddress);
                    if (connectAttempt) {
                        Toast.makeText(this, "Trying to connect to crutch:
" +
                                tempMacAddress, Toast.LENGTH_SHORT).show();
                        // Connection result (handler because must wait for
connection attempt time)
                        Log.d("ble-debug", "Runnable (countdown) started");
                        connectionHandler.postDelayed(connectionRunnable,
CONNECTION_ATTEMPT_TIME);
                    } else {
                        Toast.makeText(this, "Couldn't connect to crutch: "
+
                                BluetoothLeService.mBluetoothDeviceAddress,
Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });
    }

    /**********************************  CONFIGURATION BUTTON
**********************************
     * Sets setOnClickListener of Configuration Button to go to first
activity in data visualization
     * cycle: SelectSavedDataActivity activity (for revisiting old
experiments) */
    private void SelectSavedDataButton() {
        buttonSelectSavedData.setOnClickListener(v -> {
```

```
            // Settle "from" variable for next activity
            CrutchApplication.from = MainActivity.class.getSimpleName();
            // Go to SelectSavedDataActivity activity
            startActivity(new Intent(MainActivity.this,
                    SelectSavedDataActivity.class));
            // Set exit and enter transitions (forward)
            overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
            finish();
        });
    }


    /**********************************  CONFIGURATION BUTTON
**********************************
     * Sets setOnClickListener of Configuration Button to go to
Configuration */
    private void ConfigurationButton() {
        buttonConfiguration.setOnClickListener(v -> {
            startActivity(new Intent(this, ConfigurationActivity.class));
            // Set exit and enter transitions (forward)
            overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
            finish();
        });
    }

    /**********************************  EXIT APP BUTTON
**********************************
     * Sets the onClickListener of ExitAppButton
     * Creates an alert dialog (small message window) to ask user if they
want to exit the app with
     * two options: YES and NO.
     * YES: exit the app
     * NO: do nothing (don't exit the app)*/
    private void ExitAppButton() {
        buttonExitApp.setOnClickListener(v -> {
            new AlertDialog.Builder(this)
                    .setTitle("Exit App")
                    .setMessage("Are you sure you want to exit the app?
\n\n" +
                            "This will disconnect any connected crutch")
                    .setPositiveButton("YES", (dialog, id) -> {
                        // Close connection before exiting (otherwise the
behaviour would be similar
                        // to pressing the "Home" button in the device)
                        mBluetoothLeService.disconnectGatt();
                        // Just close this Activity to exit the App. There
is only one Activity
                        // active at any given time
                        MainActivity.this.finish();
                        dialog.cancel();
                    })
                    .setNegativeButton("NO", (dialog, id) -> {
                        // do nothing
                        dialog.cancel();
                    }).show();  // NOTE: .show goes at the end of Builder
        });
    }

    /**********************************  goToNextActivity
```

```
************************************/
    // TODO DEBUG THIS method implemented to not write same lines of code 3
times. NOTE: if device
    //  already connected, it should not check MAC address in database
    private void goToNextActivity() {
        Toast.makeText(MainActivity.this, "Connection successful",
                Toast.LENGTH_SHORT).show();
        // Check if MAC address already stored in database
        boolean duplicatedMAC =
sqliteAdmin.isMacDuplicated(BluetoothLeService.mBluetoothDeviceAddress);
        // If new MAC address, save it in "mac" table of database
        if (!duplicatedMAC) {

sqliteAdmin.saveMacAddress(BluetoothLeService.mBluetoothDeviceAddress);
            Toast.makeText(MainActivity.this, "New MAC address added",
                    Toast.LENGTH_SHORT).show();
        }
        // Settle "from" variable for next activity
        CrutchApplication.from = MainActivity.class.getSimpleName();
        // Go to the next activity (ConfigureNewUserActivity)
        startActivity(new Intent(this, ConfigureNewUserActivity.class));
        // Set exit and enter transitions (forward)
        overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
        finish();
    }


}  // </ public class MainActivity extends AppCompatActivity >
```

ConfigureNewUserActivity.java

```
package com.visens.crutch_v2;


/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class ConfigureNewUserActivity extends AppCompatActivity {


    /* ////////////////////////////////////   VARIABLES
////////////////////////////////////// */
    private static boolean initialized = false;

    // Database helper class
    private AdminSQLiteOpenHelper sqliteAdmin;
```

```
    // Layout variables used in various methods
    private EditText editTextUserID, editTextIteration;
    private Button buttonNext, buttonReturn;


    /* ////////////////////////////////////   METHODS
////////////////////////////////////// */

    /**************************************  ON CREATE
**********************************
     * Finds layout objects */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.configure_new_user_activity_layout);
        initialized = false;
        // Find layout objects
        editTextUserID = findViewById(R.id.idETUserID);
        editTextIteration = findViewById(R.id.idETIteration);
        buttonNext = findViewById(R.id.idConfigureNewUserBNext);
        buttonReturn = findViewById(R.id.idConfigureNewUserBReturn);
    }

    /**************************************  ON START
**********************************
     * Resets necessary global variables, gets instance of database object,
sets values if it
     * doesn't come from Main and calls the implemented methods */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Reset global app variables
            //  "user" table related variables
            CrutchApplication.rowIDUserTable = -1;
            CrutchApplication.date = null;
            CrutchApplication.time = null;
            CrutchApplication.numSamples = -1;
            CrutchApplication.numErrors = -1;
            CrutchApplication.numMissing = -1;
            // Initialize database variable
            sqliteAdmin = AdminSQLiteOpenHelper.getInstance(this);
            // Set values automatically if it doesn't come from Main (comes
from
            // DataAcquisitionActivity (Back) or SaveDataActivity
(Continue))
            if
(!CrutchApplication.from.equals(MainActivity.class.getSimpleName())) {
                // Add 1 to Iteration if it comes from SaveDataActivity
(Continue)
                if
(CrutchApplication.from.equals(SaveDataActivity.class.getSimpleName())) {
                    int iIteration =
Integer.parseInt(CrutchApplication.iteration) + 1;
                    CrutchApplication.iteration =
String.valueOf(iIteration);
                }
                editTextUserID.setText(CrutchApplication.userId);
                editTextIteration.setText(CrutchApplication.iteration);
            }
            // Method calls
```

```
            NextButton();
            ReturnButton();
            initialized = true;
        }
    }

    /************************************  NEXT BUTTON
************************************
     * Checks duplicate user ID and iteration. If not, switches to next
activity in data
     * acquisition cycle (DataAcquisitionActivity). If duplicated, create
AlertDialog */
    private void NextButton() {
        buttonNext.setOnClickListener(v -> {
            // Check User ID and User iteration not empty
            if (editTextUserID.getText().toString().isEmpty() ||
                    editTextIteration.getText().toString().isEmpty()) {
                Toast.makeText(this, "Introduce an ID and iteration",
                        Toast.LENGTH_SHORT).show();
            } else {
                // Read user ID and iteration from input fields
                CrutchApplication.userId =
editTextUserID.getText().toString();
                CrutchApplication.iteration =
editTextIteration.getText().toString();
                // Check for User ID and iteration duplicates
                // NOTE: if no real data was stored (probably due to a
crash) findUserRowId erases
                // the garbage data and returns -1 (the same as not
duplicated)
                int duplicatedUserRow =
sqliteAdmin.findUserRowId(CrutchApplication.userId,
                        CrutchApplication.iteration, true);
                // If user ID and iteration are duplicate, ask user if they
want to continue
                if (duplicatedUserRow >= 0) {
                    dialogAlertTwoButtons(duplicatedUserRow);
                } else {
                    Toast.makeText(ConfigureNewUserActivity.this,
                            "New User ID and Iteration",
Toast.LENGTH_SHORT).show();
                    // Go to next activity (DataAcquisitionActivity)
                    startActivity(new Intent(this,
DataAcquisitionActivity.class));
                    // Set exit and enter transitions (forward)
                    overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
                    finish();
                }
            }
        });
    }

    /************************************  RETURN BUTTON
************************************
     * Exits data acquisition cycle switching back to MainActivity */
    private void ReturnButton() {
        buttonReturn.setOnClickListener(v -> {
            // Go to previous activity (Main activity)
            startActivity(new Intent(this, MainActivity.class));
            // Set exit and enter transitions (backward)
```

```
            overridePendingTransition(R.anim.anim_fade_in,
R.anim.anim_fade_out);
            finish();
        });
    }

    /************************************  ALERT DIALOG
*************************************
     * Creates an alert dialog (small message window) to ask user if they
want to erase previous
     * user ID and iteration values with two options: YES and NO.
     * YES: deletes the saved data corresponding to the introduced user ID
and iteration from the
     * database and switches to next activity in data acquisition cycle:
DataAcquisitionActivity
     * NO: do nothing (don't delete previous user ID and iteration data) */
    private void dialogAlertTwoButtons(int duplicatedUserRow) {
        new AlertDialog.Builder(this)
                .setTitle("User ID and iteration are already used")
                .setMessage("Do you want to erase previous user data?")
                .setPositiveButton("YES", (dialog, id) -> {
                    // Delete previous user data from "user" and "data"
tables of database
                    sqliteAdmin.deleteUser(CrutchApplication.userId,
CrutchApplication.iteration);
                    sqliteAdmin.deleteDataValues(duplicatedUserRow);
                    // NOTE: user data is saved at the beginning of next
activity
                    Toast.makeText(ConfigureNewUserActivity.this,
                            "New User ID and Iteration",
Toast.LENGTH_SHORT).show();
                    // Go to next activity (DataAcquisitionActivity)
                    startActivity(new Intent(this,
DataAcquisitionActivity.class));
                    // Set exit and enter transitions (forward)
                    overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
                    finish();
                    dialog.cancel();
                })
                .setNegativeButton("NO", (dialog, id) -> {
                    // do nothing
                    dialog.cancel();
                }).show();  // NOTE: .show goes at the end of Builder
    }


}  // </ public class ConfigureNewUserActivity extends AppCompatActivity >
```

SelectSavedDataActivity.java

```
package com.visens.crutch_v2;


/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import java.util.List;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////////// */
public class SelectSavedDataActivity extends AppCompatActivity {


    /* ////////////////////////////////////   VARIABLES
////////////////////////////////////// */
    private static boolean initialized = false;

    // Database helper class
    private AdminSQLiteOpenHelper sqliteAdmin;

    // Layout variables
    private Spinner spinnerSelectUserId, spinnerSelectIteration;
    private Button buttonNext, buttonReturn;

    // Data search variables
    private List<String> listIds;
    private List<String> listIterations;
    private Boolean dataFoundInDatabase = false;


    /* ////////////////////////////////////////   METHODS
////////////////////////////////////////// */

    /*************************************  ON CREATE
**************************************
     * Finds layout objects and calls the implemented methods */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.select_saved_data_activity_layout);
        initialized = false;
        // Find layout objects
        spinnerSelectUserId = findViewById(R.id.idSUserID);
        spinnerSelectIteration = findViewById(R.id.idSIteration);
        buttonNext = findViewById(R.id.idSelectSavedDataBNext);
        buttonReturn = findViewById(R.id.idSelecSavedDataBReturn);
    }

    /*************************************  ON START
**************************************
     * */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Initialize database variable
            sqliteAdmin = AdminSQLiteOpenHelper.getInstance(this);
            // Read user IDs stored in database
```

```
            listIds = sqliteAdmin.readUserIds();
            // Check if there is data available
            if (listIds.isEmpty()) {
                dataFoundInDatabase = false;
                Toast.makeText(this, "No data available",
Toast.LENGTH_SHORT).show();
            } else {
                dataFoundInDatabase = true;
            }
            // Method calls
            UserIdSpinner();
            IterationSpinner();
            NextButton();
            ReturnButton();
            initialized = true;
        }
    }

    /************************************  USER ID SPINNER
**********************************
     * Configure the spinner that selects users */
    private void UserIdSpinner() {
        if (dataFoundInDatabase) {
            // Configure userId Spinner
            ArrayAdapter<String> arrayAdapterIds = new ArrayAdapter<>(this,
                    android.R.layout.simple_spinner_item, listIds);

arrayAdapterIds.setDropDownViewResource(android.R.layout.simple_spinner_dro
pdown_item);
            spinnerSelectUserId.setEnabled(true);
            spinnerSelectUserId.setAdapter(arrayAdapterIds);
            spinnerSelectUserId.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
                @Override
                public void onItemSelected(AdapterView<?> parent, View
view, int position, long id) {
                    CrutchApplication.userId =
parent.getItemAtPosition(position).toString();
                    listIterations =
sqliteAdmin.readIterationsOfUserId(CrutchApplication.userId);
                    IterationSpinner();
                }

                @Override
                public void onNothingSelected(AdapterView<?> parent) {
                    // do nothing
                    // NOTE: this method must exist
                }
            });
        } else {
            spinnerSelectUserId.setEnabled(false);
        }
    }

    /************************************  IterationSpinner
**********************************
     * Configure the spinner that selects iterations */
    // TODO [future] maybe there is a better way of refreshing this spinner
when a userId is
    //  selected in the other Spinner than calling this method and
rewriting all objects
```

```
    private void IterationSpinner() {
        // !!! NOTE: CrutchApplication.userId != null is needed to prevent
crash
        if (dataFoundInDatabase && CrutchApplication.userId != null) {
            // Configure iteration Spinner
            // NOTE: setting the value of listIterations here prevents
crashes
            listIterations =
sqliteAdmin.readIterationsOfUserId(CrutchApplication.userId);
            ArrayAdapter<String> arrayAdapterIterations = new
ArrayAdapter<>(this,
                    android.R.layout.simple_spinner_item, listIterations);

arrayAdapterIterations.setDropDownViewResource(android.R.layout.simple_spin
ner_dropdown_item);
            spinnerSelectIteration.setEnabled(true);
            spinnerSelectIteration.setAdapter(arrayAdapterIterations);
            spinnerSelectIteration.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
                @Override
                public void onItemSelected(AdapterView<?> parent, View
view, int position, long id) {
                    CrutchApplication.iteration =
parent.getItemAtPosition(position).toString();
                }

                @Override
                public void onNothingSelected(AdapterView<?> parent) {
                    // do nothing
                    // NOTE: this method must exist
                }
            });
        } else {
            spinnerSelectIteration.setEnabled(false);
        }
    }

    /************************************  NEXT BUTTON
**********************************
     *  */
    private void NextButton() {
        // If there is data in database, allow use of button
        if (dataFoundInDatabase) {
            buttonNext.setEnabled(true);
            buttonNext.setOnClickListener(v -> {
                // Check if user ID and iteration not empty
                if (CrutchApplication.userId == null &&
CrutchApplication.iteration == null) {
                    Toast.makeText(this, "Please select a user ID and
iteration",
                            Toast.LENGTH_SHORT).show();
                } else {
                    // Read user values directly to global app variables
                    sqliteAdmin.readUser(CrutchApplication.userId,
CrutchApplication.iteration);
                    // Find rowID of the user in "user" table to associate
it with their
                    // corresponding data in "data" table in the next
activities
                    CrutchApplication.rowIDUserTable =
sqliteAdmin.findUserRowId(
```

```
                            CrutchApplication.userId,
CrutchApplication.iteration, false);
                    // NOTE: only user data is read here as the data values
are read directly in
                    // VisualizeDataActivity
                    CrutchApplication.from =
SelectSavedDataActivity.class.getSimpleName();
                    // Switch to next activity (VisualizeDataActivity)
                    startActivity(new Intent(this,
VisualizeDataActivity.class));
                    // Set exit and enter transitions (forward)
                    overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
                    finish();
                }
            });
        } else {
            buttonNext.setEnabled(false);
            buttonNext.setVisibility(View.INVISIBLE);
        }
    }

    /************************************  RETURN BUTTON
************************************
     * Exits saved data cycle switching back to MainActivity */
    private void ReturnButton() {
        buttonReturn.setOnClickListener(v -> {
            // Go back to MainActivity
            startActivity(new Intent(this, MainActivity.class));
            // Set exit and enter transitions (backward)
            overridePendingTransition(R.anim.anim_fade_in,
R.anim.anim_fade_out);
            finish();
        });
    }


}  // </ public class SelectSavedDataActivity extends AppCompatActivity >
```

DataAcquisitionActivity.java

```
package com.visens.crutch_v2;


/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import static android.bluetooth.BluetoothAdapter.STATE_CONNECTED;
import static android.bluetooth.BluetoothAdapter.STATE_DISCONNECTED;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothGatt;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Color;
```

```
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.os.SystemClock;
import android.util.Log;
import android.view.View;
import android.widget.Chronometer;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.ActionBar;
import androidx.appcompat.app.AppCompatActivity;

import java.sql.Timestamp;
import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Objects;
import java.util.UUID;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class DataAcquisitionActivity extends AppCompatActivity {


    /* ////////////////////////////////////////   VARIABLES
//////////////////////////////////////// */
    private static boolean initialized = false;

    // Database helper class
    private AdminSQLiteOpenHelper sqliteAdmin;

    // Layout variables
    //  - Data values display
    private TextView textViewForce, textViewAltitude,
            textViewOrientationRoll, textViewOrientationPitch,
textViewOrientationYaw,
            textViewAccelerationX, textViewAccelerationY,
textViewAccelerationZ,
            textViewGyroX, textViewGyroY, textViewGyroZ,
            textViewMagneticX, textViewMagneticY, textViewMagneticZ;
    //  - Time
    private Chronometer myChronometer;
    //  - Controls
    private ImageButton imageButtonPlay, imageButtonPause, imageButtonBack,
imageButtonStop,
            imageButtonChronometerControl;
    // Action Bar
    private TextView textViewBluetoothState;
    private TextView textViewLocationState;
    private TextView textViewCrutchState;
    private TextView textViewMacAddressValue;
    private TextView textViewUserIDValue;
    private TextView textViewIterationValue;

    // Bluetooth related variables
    private BluetoothLeService mBluetoothLeService;
    public static Boolean dataAcquisitionRunning = false;
```

```
    private float fForce, fAltitude, fRoll, fPitch, fYaw, fAccX, fAccY,
// characteristic 1
            fAccZ, fGyroX, fGyroY, fGyroZ, fMagneticX, fMagneticY,
fMagneticZ;  // characteristic 2
    // NOTE: integer values of data values and iteration1 and iteration2
are local variables in
    // broadcastReceiver

    // Chronometer variables
    private long chronometerTime = 0L;
    private ArrayList<String> userTimeStamps = new ArrayList<>();

    // Error counting
    // TODO [future] can't make them private (warning appears to make them
final). Check difference between
    //  private and protected
    protected ArrayList<Integer> ArrayIteration1 = new ArrayList<>();  //
received bit for checking data OK received
    protected ArrayList<Integer> ArrayIteration2 = new ArrayList<>();  //
received bit for checking data OK received
    private int numErrors = 0, numMissing = 0;

    // TODO screen and graphs update put properly maybe with periodic task
or something like that
    // Variable to manage slow down of screen views update
    private int counterUpdateScreenValues = 0;
    // Constant
    private final static int VALUES_AND_GRAPHIC_UPDATE_INTERVAL = 50;  //
100 = 2s*50Hz (desired interval: 1 seconds)
    // NOTE: sampling frequency of 50 Hz is for the 2 characteristics. This
means current
    //  method onCharacteristicChanged is called twice every 0.02s.
Therefore, the value of
    //  the counter C to set S seconds between updates is -> C = 2*S*50 =
100*S

    // Code to manage Service lifecycle
    private final ServiceConnection mServiceConnection = new
ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder
service) {
            mBluetoothLeService = ((BluetoothLeService.LocalBinder)
service).getService();
            // Update connection textView
            if (mBluetoothLeService.mConnectionState == STATE_CONNECTED) {
                textViewCrutchState.setText(R.string.textTVConnected);

textViewMacAddressValue.setText(BluetoothLeService.mBluetoothDeviceAddress)
;
            } else if (mBluetoothLeService.mConnectionState ==
STATE_DISCONNECTED) {
                textViewCrutchState.setText(R.string.textTVDisconnected);

textViewMacAddressValue.setText(R.string.textTVDefaultValueActionBar);
            }
            // Increase Bluetooth priority

mBluetoothLeService.setConnectionPriority(BluetoothGatt.CONNECTION_PRIORITY
_HIGH);
            // Set characteristics notifications to true
```

```
            ArrayList<UUID> arrayUuids = new ArrayList<>();
            arrayUuids.add(BluetoothLeService.characteristic_1_UUID);
            arrayUuids.add(BluetoothLeService.characteristic_2_UUID);
            mBluetoothLeService.setCharacteristicsNotification(arrayUuids,
true);
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            Log.d("ble-debug", "Bluetooth disconnected in
DataAcquisition");
            mBluetoothLeService = null;
        }
    };

    // BroadcastReceiver that handles various events fired by the Service
    private final BroadcastReceiver mGattUpdateReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
                Log.d("ble-debug", "DATA_ACQUISITION: broadcast received:
connected");
                // Set connection state
                textViewCrutchState.setText(R.string.textTVConnected);

textViewMacAddressValue.setText(BluetoothLeService.mBluetoothDeviceAddress)
;
            } else if
(BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
                Log.d("ble-debug", "DATA_ACQUISITION: broadcast received:
disconnected");
                // Set connection state
                textViewCrutchState.setText(R.string.textTVDisconnected);

textViewMacAddressValue.setText(R.string.textTVDefaultValueActionBar);
            } else if
(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
                Log.d("ble-debug", "DATA_ACQUISITION: broadcast received:
services discovered");
            } else if
(BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
                Log.d("ble-debug", "DATA_ACQUISITION: broadcast received:
data available");

CrutchApplication.dataAcquisitionAbsoluteStartTimes.add(SystemClock.elapsed
Realtime());
                Bundle extras = intent.getExtras();
                int iIteration1 =
extras.getInt(BluetoothLeService.EXTRA_ITERATION1);
                int iIteration2 =
extras.getInt(BluetoothLeService.EXTRA_ITERATION2);
                fForce = extras.getFloat(BluetoothLeService.EXTRA_FORCE);
                fAltitude =
extras.getFloat(BluetoothLeService.EXTRA_ALTITUDE);
                fRoll = extras.getFloat(BluetoothLeService.EXTRA_ROLL);
                fPitch = extras.getFloat(BluetoothLeService.EXTRA_PITCH);
                fYaw = extras.getFloat(BluetoothLeService.EXTRA_YAW);
                fAccX = extras.getFloat(BluetoothLeService.EXTRA_ACCX);
                fAccY = extras.getFloat(BluetoothLeService.EXTRA_ACCY);
```

```
                fAccZ = extras.getFloat(BluetoothLeService.EXTRA_ACCZ);
                fGyroX = extras.getFloat(BluetoothLeService.EXTRA_GYROX);
                fGyroY = extras.getFloat(BluetoothLeService.EXTRA_GYROY);
                fGyroZ = extras.getFloat(BluetoothLeService.EXTRA_GYROZ);
                fMagneticX =
extras.getFloat(BluetoothLeService.EXTRA_MAGNX);
                fMagneticY =
extras.getFloat(BluetoothLeService.EXTRA_MAGNY);
                fMagneticZ =
extras.getFloat(BluetoothLeService.EXTRA_MAGNZ);
                // TODO time duration of screen update (start)
                long inicTiempo = SystemClock.elapsedRealtimeNanos();
                // TODO this is another task. Probably should go in another
handler/thread
                // Slow down the actualization of screen views by using a
counter
                if (counterUpdateScreenValues ==
VALUES_AND_GRAPHIC_UPDATE_INTERVAL) {
                    updateDataValues();
                    counterUpdateScreenValues = 0;
                } else {
                    counterUpdateScreenValues++;
                }
                // TODO time duration of screen update (end) and data
saving in database (start)
                long finTiempo = SystemClock.elapsedRealtimeNanos();
                CrutchApplication.screenUpdateDurations.add(finTiempo -
inicTiempo);
                // Get timestamp
                Timestamp mTimestamp = new
Timestamp(System.currentTimeMillis());
                // Save values in data table of database

sqliteAdmin.saveDataValues(CrutchApplication.rowIDUserTable, fForce,
fAltitude,
                        fRoll, fPitch, fYaw, fAccX, fAccY, fAccZ, fGyroX,
fGyroY, fGyroZ, fMagneticX,
                        fMagneticY, fMagneticZ, iIteration1, iIteration2,
mTimestamp.toString());
                // TODO time duration of data saving in database (start)
                long finTiempo2 = SystemClock.elapsedRealtimeNanos();
                CrutchApplication.dataSavingDurations.add(finTiempo2 -
finTiempo);
            } else if
(BluetoothAdapter.ACTION_STATE_CHANGED.equals(action)) {
                if (intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, -1)
                        == BluetoothAdapter.STATE_ON) {
                    // TODO bluetoothActive and locationActive
                    //bluetoothActive = true;
                    textViewBluetoothState.setText(R.string.textTVEnabled);
                } else {
                    //bluetoothActive = false;

textViewBluetoothState.setText(R.string.textTVDisabled);
                }
            } else if (LocationManager.MODE_CHANGED_ACTION.equals(action))
{
                if
(intent.getBooleanExtra(LocationManager.EXTRA_LOCATION_ENABLED,
                        false)) {
                    //locationActive = true;
```

```
                    textViewLocationState.setText(R.string.textTVEnabled);
                } else {
                    //locationActive = false;
                    textViewLocationState.setText(R.string.textTVDisabled);
                }
            }
        }
    };


    /* ////////////////////////////////////   METHODS
//////////////////////////////////////// */

    /*********************************  makeGattUpdateIntentFilter
*******************************
     * Register for BroadcastReceiver */
    private static IntentFilter makeGattUpdateIntentFilter() {
        final IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);

intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);

intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
        intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
        intentFilter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
        intentFilter.addAction(LocationManager.MODE_CHANGED_ACTION);
        return intentFilter;
    }

    /*************************************  ON CREATE
***********************************
     * Finds layout objects */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.data_acquisition_activity_layout);
        initialized = false;
        // Find layout objects
        //  - Data values display
        textViewForce = findViewById(R.id.idTVForceValue);
        textViewAltitude = findViewById(R.id.idTVAltitudeValue);
        textViewOrientationRoll = findViewById(R.id.idTVOrientationValueX);
        textViewOrientationPitch =
findViewById(R.id.idTVOrientationValueY);
        textViewOrientationYaw = findViewById(R.id.idTVOrientationValueZ);
        textViewAccelerationX = findViewById(R.id.idTVAccelerationValueX);
        textViewAccelerationY = findViewById(R.id.idTVAccelerationValueY);
        textViewAccelerationZ = findViewById(R.id.idTVAccelerationValueZ);
        textViewGyroX = findViewById(R.id.idTVGyroValueX);
        textViewGyroY = findViewById(R.id.idTVGyroValueY);
        textViewGyroZ = findViewById(R.id.idTVGyroValueZ);
        textViewMagneticX = findViewById(R.id.idTVMagneticValueX);
        textViewMagneticY = findViewById(R.id.idTVMagneticValueY);
        textViewMagneticZ = findViewById(R.id.idTVMagneticValueZ);
        //  - Chronometer
        myChronometer = findViewById(R.id.idChronometerTimeValue);
        //  - Control buttons
        imageButtonPlay = findViewById(R.id.idCommunicationIBPlay);
        imageButtonBack = findViewById(R.id.idCommunicationIBBack);
        imageButtonPause = findViewById(R.id.idCommunicationIBPause);
        imageButtonStop = findViewById(R.id.idCommunicationIBStop);
```

```
        imageButtonChronometerControl =
findViewById(R.id.idCommunicationIBChronometer);
        // Set custom ActionBar and get its textViews

Objects.requireNonNull(getSupportActionBar()).setDisplayOptions(ActionBar.D
ISPLAY_SHOW_CUSTOM);
        getSupportActionBar().setDisplayShowCustomEnabled(true);
        getSupportActionBar().setCustomView(R.layout.action_bar_layout);
        View viewActionBar = getSupportActionBar().getCustomView();
        textViewBluetoothState =
viewActionBar.findViewById(R.id.idTVBluetoothValue);
        textViewLocationState =
viewActionBar.findViewById(R.id.idTVLocationValue);
        textViewCrutchState =
viewActionBar.findViewById(R.id.idTVCrutchConnectionValue);
        textViewMacAddressValue =
viewActionBar.findViewById(R.id.idTVMacActionBarValue);
        textViewUserIDValue =
viewActionBar.findViewById(R.id.idTVUserIDActionBarValue);
        textViewIterationValue =
viewActionBar.findViewById(R.id.idTVIterationActionBarValue);
        // Set needed contents invisible
        //  - Buttons
        imageButtonPause.setVisibility(View.INVISIBLE);
        imageButtonStop.setVisibility(View.INVISIBLE);
        //  - Graph
        // Bind Bluetooth LE service
        Intent gattServiceIntent = new Intent(this,
BluetoothLeService.class);
        bindService(gattServiceIntent, mServiceConnection,
BIND_AUTO_CREATE);
    }


    /*****************************************  ON START
*****************************************
     */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Register receiver
            registerReceiver(mGattUpdateReceiver,
makeGattUpdateIntentFilter());
            // TODO sustituir por ActionBar
            // Set User ID and Iteration in screen texts
            //textViewUserID.setText(CrutchApplication.userId);
            //textViewIteration.setText(CrutchApplication.iteration);
            // Initialize database variable
            sqliteAdmin = AdminSQLiteOpenHelper.getInstance(this);
            // Find today's date
            Date myDate = Calendar.getInstance().getTime();
            //DateFormat myDateFormat =
DateFormat.getDateInstance(DateFormat.SHORT);
            DateFormat myDateFormat =
DateFormat.getDateInstance(DateFormat.SHORT);
            CrutchApplication.date = myDateFormat.format(myDate);
            // Save user information in "user" table of database to get the
new rowID so the data
            // values can be associated with the user ID and iteration
            // NOTE: time, numErrors and numMissing are not saved here as
```

```
the value is unknown
            // until the data acquisition finishes
            sqliteAdmin.saveUser(CrutchApplication.userId,
CrutchApplication.iteration,
                    CrutchApplication.date, "-1", -1, -1, -1);
            // Find rowID of the new added user in "user" table to
associate it with their
            // corresponding data in "data" table
            CrutchApplication.rowIDUserTable =
sqliteAdmin.findUserRowId(CrutchApplication.userId,
CrutchApplication.iteration, false);
            // Method calls
            ChronometerButton();
            BackButton();                   // go to previous activity
(ConfigureNewUserActivity) ONLY before pressing play button
            PlayButton();                   // start data acquisition
            PauseButton();                  // pause data acquisition ONLY
after pressing play button
            StopButton();                   // stop data acquisition and go
to next activity (VisualizeDataActivity) ONLY after pressing play button

            // TODO debugging durations
            CrutchApplication.dataProcessingDurations = new ArrayList<>();
            CrutchApplication.dataSavingDurations = new ArrayList<>();
            CrutchApplication.screenUpdateDurations = new ArrayList<>();
            CrutchApplication.dataAcquisitionAbsoluteStartTimes = new
ArrayList<>();
            CrutchApplication.characteristicArrivalTimes = new
ArrayList<>();
            CrutchApplication.inicioRecepcionDatos = 0L;

            initialized = true;
        }
    }

    /**************************************  ON DESTROY
**************************************
     */
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Unregister receiver
        unregisterReceiver(mGattUpdateReceiver);
        // Unbind Bluetooth LE service
        unbindService(mServiceConnection);
        mBluetoothLeService = null;
    }

    /************************************  UPDATE VALUES
************************************/
    private void resetDataValues() {
        float defValue = 0;
        // Set data values text views with two decimals
        textViewForce.setText(String.format(java.util.Locale.US, "%.2f",
defValue));
        textViewAltitude.setText(String.format(java.util.Locale.US, "%.2f",
defValue));
        textViewOrientationRoll.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewOrientationPitch.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
```

```
        textViewOrientationYaw.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewAccelerationX.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewAccelerationY.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewAccelerationZ.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewGyroX.setText(String.format(java.util.Locale.US, "%.2f",
defValue));
        textViewGyroY.setText(String.format(java.util.Locale.US, "%.2f",
defValue));
        textViewGyroZ.setText(String.format(java.util.Locale.US, "%.2f",
defValue));
        textViewMagneticX.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewMagneticY.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
        textViewMagneticZ.setText(String.format(java.util.Locale.US,
"%.2f", defValue));
    }

    /*********************************  UPDATE VALUES AND GRAPH
*********************************/
    private void updateDataValues() {
        // Set data values text views with two decimals
        textViewForce.setText(String.format(java.util.Locale.US, "%.2f",
fForce));
        textViewAltitude.setText(String.format(java.util.Locale.US, "%.2f",
fAltitude));
        textViewOrientationRoll.setText(String.format(java.util.Locale.US,
"%.2f", fRoll));
        textViewOrientationPitch.setText(String.format(java.util.Locale.US,
"%.2f", fPitch));
        textViewOrientationYaw.setText(String.format(java.util.Locale.US,
"%.2f", fYaw));
        textViewAccelerationX.setText(String.format(java.util.Locale.US,
"%.2f", fAccX));
        textViewAccelerationY.setText(String.format(java.util.Locale.US,
"%.2f", fAccY));
        textViewAccelerationZ.setText(String.format(java.util.Locale.US,
"%.2f", fAccZ));
        textViewGyroX.setText(String.format(java.util.Locale.US, "%.2f",
fGyroX));
        textViewGyroY.setText(String.format(java.util.Locale.US, "%.2f",
fGyroY));
        textViewGyroZ.setText(String.format(java.util.Locale.US, "%.2f",
fGyroZ));
        textViewMagneticX.setText(String.format(java.util.Locale.US,
"%.2f", fMagneticX));
        textViewMagneticY.setText(String.format(java.util.Locale.US,
"%.2f", fMagneticY));
        textViewMagneticZ.setText(String.format(java.util.Locale.US,
"%.2f", fMagneticZ));
    }

    /*************************************  PLAY BUTTON
*************************************
     */
    private void PlayButton() {
        imageButtonPlay.setOnClickListener(v -> {
```

```
            resetDataValues();
            imageButtonPlay.setVisibility(View.INVISIBLE);
            imageButtonPause.setVisibility(View.VISIBLE);
            imageButtonBack.setVisibility(View.INVISIBLE);
            imageButtonStop.setVisibility(View.VISIBLE);
            // Start chronometer
            myChronometer.setBase(SystemClock.elapsedRealtime());
            myChronometer.start();
            imageButtonChronometerControl.setColorFilter(Color.GREEN);
            // TODO dataAcquisitionRunning
            dataAcquisitionRunning = true;
        });
    }

    /************************************  PAUSE BUTTON
**************************************
     */
    private void PauseButton() {
        imageButtonPause.setOnClickListener(v -> {
            dataAcquisitionRunning = false;
            imageButtonPause.setVisibility(View.INVISIBLE);
            imageButtonPlay.setVisibility(View.VISIBLE);
        });
    }

    /************************************  STOP BUTTON
**************************************
     */
    private void StopButton() {
        imageButtonStop.setOnClickListener(v -> {
            // Stop data acquisition and chronometer
            // TODO dataAcquisitionRunning
            dataAcquisitionRunning = false;
            myChronometer.stop();
            chronometerTime = SystemClock.elapsedRealtime() -
myChronometer.getBase();

            // TODO color changing is not needed anymore. Update this when
chronometer functionality
            //  is updated
            imageButtonChronometerControl.setColorFilter(Color.RED);

            // Decrease Bluetooth priority

mBluetoothLeService.setConnectionPriority(BluetoothGatt.CONNECTION_PRIORITY
_BALANCED);
            // Unsubscribe from characteristics
            ArrayList<UUID> arrayUuids = new ArrayList<>();
            arrayUuids.add(BluetoothLeService.characteristic_1_UUID);
            arrayUuids.add(BluetoothLeService.characteristic_2_UUID);
            mBluetoothLeService.setCharacteristicsNotification(arrayUuids,
false);
            // Count the errors
            countErrors();
            // Get number of samples
            CrutchApplication.numSamples = ArrayIteration1.size();
            // Update global app variables
            CrutchApplication.time = String.valueOf(chronometerTime);
            CrutchApplication.numErrors = numErrors;
            CrutchApplication.numMissing = numMissing;
            // Update time, numSamples, numErrors and numMissing columns in
```

```
user table
            sqliteAdmin.updateUserRow(CrutchApplication.rowIDUserTable,
CrutchApplication.userId,
                    CrutchApplication.iteration, CrutchApplication.date,
CrutchApplication.time,
                    CrutchApplication.numSamples,
CrutchApplication.numErrors, CrutchApplication.numMissing);
            // Update "from" variable
            CrutchApplication.from =
DataAcquisitionActivity.class.getSimpleName();
            // Go to next activity
            startActivity(new Intent(this, VisualizeDataActivity.class));
            // Set exit and enter transitions (forward)
            overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
            finish();
        });
    }

    /************************************  countErrors
************************************/
    private void countErrors() {
        // Read iteration values from database for counting errors
        sqliteAdmin.readIterationValues(CrutchApplication.rowIDUserTable,
ArrayIteration1, ArrayIteration2);
        // Count errors and characteristics missing
        for (int i = 0; i < (ArrayIteration1.size() - 1); i++) {  // .size
- 1 because second error type counting needs i+1 index
            // Iteration1(i) != Iteration2(i) (tangled)
            if (!ArrayIteration1.get(i).equals(ArrayIteration2.get(i))) {
                numErrors++;
            }
            // Iteration1(i) == Iteration1(i+1) (repeated/10
characteristics lost)
            /* NOTE: this error is ambiguous in nature because it can mean
the same
               characteristic repeated or 10 characteristics lost */
            if (ArrayIteration1.get(i).equals(ArrayIteration1.get(i + 1)))
{
                numErrors++;
                //numMissing = numMissing + 10;
            } else
                // Iteration1(i) != ((Iteration1(i+1) - 1) MOD 10)
(characteristics lost)
                // NOTE: modulo operator in java -> %
                if (((ArrayIteration1.get(i) + 1) % 10) !=
ArrayIteration1.get(i + 1)) {
                    numMissing++;
                }
        }
        // Check last pair
        // Iteration1(i) != Iteration2(i) (tangled)
        int lastIndex = ArrayIteration1.size() - 1;
        if
(!ArrayIteration1.get(lastIndex).equals(ArrayIteration2.get(lastIndex))) {
            numErrors++;
        }
    }

    /************************************  BACK BUTTON
************************************
```

```
     * Switch to previous activity (ConfigureNewUserActivity)
     * NOTE: ONLY before pressing play button to start data acquisition */
    private void BackButton() {
        imageButtonBack.setOnClickListener(v -> {
            // Delete user data in "user" table
            sqliteAdmin.deleteUser(CrutchApplication.userId,
CrutchApplication.iteration);
            // Settle "from" variable for next activity
            CrutchApplication.from =
DataAcquisitionActivity.class.getSimpleName();
            // Go to previous activity
            startActivity(new Intent(this,
ConfigureNewUserActivity.class));
            // Set exit and enter transitions (backward)
            overridePendingTransition(R.anim.anim_left_right_in,
R.anim.anim_fade_out);
            finish();
        });
    }

    /*********************************  CHRONOMETER BUTTON
**********************************
     */
    private void ChronometerButton() {
        imageButtonChronometerControl.setOnClickListener(v -> {
            userTimeStamps.add((new
Timestamp(System.currentTimeMillis())).toString());
        });
    }


}  // </ public class DataAcquisitionActivity extends AppCompatActivity >
```

VisualizeDataActivity.java

```
package com.visens.crutch_v2;


/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Description;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
```

```
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.formatter.DefaultValueFormatter;
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet;

import java.util.ArrayList;
import java.util.List;


/* ////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class VisualizeDataActivity extends AppCompatActivity {


    /* ////////////////////////////////////   VARIABLES
//////////////////////////////////////// */
    private static boolean initialized = false;

    // Layout variables
    private Spinner spinnerSelectData;
    private LineChart lineChartTop, lineChartMiddle, lineChartBottom;
    private Button buttonBack;
    private Button buttonNext;

    // Graph variables
    // TODO [future] (same in DataAcquisitionActivity) can't make them
private (warning appears to
    //  make them final). Check difference between private and protected
    protected ArrayList<Float> arrayForce = new ArrayList<>();
    protected ArrayList<Float> arrayAltitude = new ArrayList<>();
    protected ArrayList<Float> arrayAccX = new ArrayList<>();
    protected ArrayList<Float> arrayAccY = new ArrayList<>();
    protected ArrayList<Float> arrayAccZ = new ArrayList<>();
    protected ArrayList<Float> arrayRoll = new ArrayList<>();
    protected ArrayList<Float> arrayPitch = new ArrayList<>();
    protected ArrayList<Float> arrayYaw = new ArrayList<>();
    protected ArrayList<Float> arrayGyroX = new ArrayList<>();
    protected ArrayList<Float> arrayGyroY = new ArrayList<>();
    protected ArrayList<Float> arrayGyroZ = new ArrayList<>();
    protected ArrayList<Float> arrayMagneticX = new ArrayList<>();
    protected ArrayList<Float> arrayMagneticY = new ArrayList<>();
    protected ArrayList<Float> arrayMagneticZ = new ArrayList<>();
    private String titleName = "Force";


    /* ////////////////////////////////////   METHODS
//////////////////////////////////////// */

    /************************************  ON CREATE
************************************
     * Finds layout objects */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.visualize_data_activity_layout);
        initialized = false;
        // Find layout variables
        spinnerSelectData = findViewById(R.id.idSSelectData);
        lineChartTop = findViewById(R.id.idDataAcquisitonLCGraph);
        lineChartMiddle = findViewById(R.id.idLCGraphic2);
```

```
        lineChartBottom = findViewById(R.id.idLCGraphic3);
        buttonBack = findViewById(R.id.idVisualizeDataBBack);
        buttonNext = findViewById(R.id.idVisualizeDataBNext);
    }

    /**************************************  ON START
*************************************
     * Gets instance of database object, reads data values from database
and calls the implemented
     * methods */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Initialize database variable
            AdminSQLiteOpenHelper sqliteAdmin =
AdminSQLiteOpenHelper.getInstance(this);
            // TODO [future] implement buttons to switch between data value
intervals (due to RAM
            //  limitations only a certain amount of values can be
displayed at any given time
            //  without the app crashing)
            // Read data values from "data" table
            // Get user's data values with rowIDUserTable from DATA table
            synchronized (this) {
                SQLiteDatabase database =
sqliteAdmin.getReadableDatabase();
                Cursor cursorDataValues =
sqliteAdmin.readDataValuesCursor(CrutchApplication.rowIDUserTable,
database);
                int columnIndex;
                // TODO [future] a max number of data samples are displayed
until buttons to switch
                //  between data value intervals are implemented
                Toast.makeText(this, "Showing only the first 3000 samples",
                        Toast.LENGTH_SHORT).show();
                int count = 0;
                while (count < 3000 && !cursorDataValues.isAfterLast()) {
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL02);
                    arrayForce.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL03);

arrayAltitude.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL04);
                    arrayRoll.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL05);
                    arrayPitch.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL06);
                    arrayYaw.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL07);
                    arrayAccX.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL08);
                    arrayAccY.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
```

```
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL09);
                    arrayAccZ.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL10);
                    arrayGyroX.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL11);
                    arrayGyroY.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL12);
                    arrayGyroZ.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL13);

arrayMagneticX.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL14);

arrayMagneticY.add(cursorDataValues.getFloat(columnIndex));
                    columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL15);

arrayMagneticZ.add(cursorDataValues.getFloat(columnIndex));
                    // NOTE: values of iteration1 and iteration2 not needed
                    cursorDataValues.moveToNext();
                    count++;
                }
                cursorDataValues.close();
                database.close();
            }
            // Method calls
            SelectDataSpinner();
            NextButton();
            BackButton();
            initialized = true;
        }
    }

    /*********************************  SELECT DATA SPINNER
**********************************
     * Selects the data value (Force, Altitude, etc.) to visualize in
one/three graphs (depends if
     * variable has 3 axis or not) */
    private void SelectDataSpinner() {
        List<String> dataValuesNames = new ArrayList<>();
        dataValuesNames.add("Force");
        dataValuesNames.add("Altitude");
        dataValuesNames.add("Orientation");
        dataValuesNames.add("Acceleration");
        dataValuesNames.add("Gyroscope");
        dataValuesNames.add("Magnetic field");
        ArrayAdapter<String> dataAdapter = new ArrayAdapter<>(this,
                android.R.layout.simple_spinner_item, dataValuesNames);

dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdow
n_item);
        spinnerSelectData.setAdapter(dataAdapter);
        spinnerSelectData.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
```

```
int position, long id) {
                // Get selected variable name from spinnerSelectData
                titleName = parent.getItemAtPosition(position).toString();
                // Refresh corresponding graph(s)
                switch (titleName) {
                    case "Force":
                        lineChartBottom.setVisibility(View.INVISIBLE);
                        lineChartMiddle.setVisibility(View.INVISIBLE);
                        createGraph(lineChartTop, arrayForce, "");
                        break;
                    case "Altitude":
                        lineChartBottom.setVisibility(View.INVISIBLE);
                        lineChartMiddle.setVisibility(View.INVISIBLE);
                        createGraph(lineChartTop, arrayAltitude, "");
                        break;
                    case "Orientation":
                        lineChartBottom.setVisibility(View.VISIBLE);
                        lineChartMiddle.setVisibility(View.VISIBLE);
                        createGraph(lineChartTop, arrayRoll, "roll");
                        createGraph(lineChartMiddle, arrayPitch, "pitch");
                        createGraph(lineChartBottom, arrayYaw, "yaw");
                        break;
                    case "Acceleration":
                        lineChartBottom.setVisibility(View.VISIBLE);
                        lineChartMiddle.setVisibility(View.VISIBLE);
                        createGraph(lineChartTop, arrayAccX, "x");
                        createGraph(lineChartMiddle, arrayAccY, "y");
                        createGraph(lineChartBottom, arrayAccZ, "z");
                        break;
                    case "Gyroscope":
                        lineChartBottom.setVisibility(View.VISIBLE);
                        lineChartMiddle.setVisibility(View.VISIBLE);
                        createGraph(lineChartTop, arrayGyroX, "x");
                        createGraph(lineChartMiddle, arrayGyroY, "y");
                        createGraph(lineChartBottom, arrayGyroZ, "z");
                        break;
                    case "Magnetic field":
                        lineChartBottom.setVisibility(View.VISIBLE);
                        lineChartMiddle.setVisibility(View.VISIBLE);
                        createGraph(lineChartTop, arrayMagneticX, "x");
                        createGraph(lineChartMiddle, arrayMagneticY, "y");
                        createGraph(lineChartBottom, arrayMagneticZ, "z");
                        break;
                }
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
                // do nothing
                // NOTE: this method must exist
            }
        });
    }

    /************************************  CREATE GRAPH
************************************
     * Creates the necessary variables to display the data of
arrayListDataValues in the passed
     * lineChart mLineChart. If the data variable has 3 axis axisName needs
to be specified
     * (axis x, axis y, axis z) and axisName = "" otherwise */
```

```
    private void createGraph(LineChart mLineChart, ArrayList<Float>
arrayListDataValues,
                             String axisName) {
        // INFO (links):
        //   https://weeklycoding.com/mpandroidchart-documentation/
        //   https://www.youtube.com/watch?v=DD1CxoVONFE

        // Set parameters
        mLineChart.setVisibility(View.INVISIBLE);
        //mLineChart.setOnChartGestureListener(VisualizeDataActivity.this);

//mLineChart.setOnChartValueSelectedListener(VisualizeDataActivity.this);
        //mLineChart.setLayoutParams(
        //        new
com.github.mikephil.charting.charts.LineChart.LayoutParams(
        //                LineChart.LayoutParams.WRAP_CONTENT,
        //                LineChart.LayoutParams.WRAP_CONTENT));
        mLineChart.setVisibility(View.VISIBLE);
        mLineChart.setDragEnabled(true);    // allow displacement in x axis
        mLineChart.setScaleEnabled(true);  // allow image magnification

        // Axis configuration
        //  - x axis
        XAxis xAxis = mLineChart.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
        xAxis.setTextSize(10f);
        xAxis.setTextColor(Color.BLACK);
        xAxis.setDrawAxisLine(true);
        //  - y axis
        YAxis yAxis = mLineChart.getAxisLeft();
        yAxis.setTextSize(10f);
        // TODO number of digits, sizes, etc. should be declared in
constants
        yAxis.setValueFormatter(new DefaultValueFormatter(3));
        yAxis.setTextColor(Color.BLACK);
        // TODO yAxis.setDrawAxisLine(true); ESTO NO ESTÁ EN EL EJE Y
        // TODO Esto último qué es? (por defecto se dibujan ejes a la
izquierda y derecha y se
        //  desactiva el de la derecha??)
        mLineChart.getAxisRight().setEnabled(false);

        // Description (subtitle) configuration
        Description description = mLineChart.getDescription();
        if (axisName.equals("")) {
            description.setEnabled(false);
        } else {
            description.setEnabled(true);
            description.setPosition(600f, 50f);
            description.setText(axisName);
            description.setTextSize(20f);
        }

        // Legend configuration
        mLineChart.getLegend().setEnabled(false);

        // Prepare data (y values)
        ArrayList<Entry> yValues = new ArrayList<>();
        for (int i = 0; i < arrayListDataValues.size(); i++) {
            yValues.add(new Entry(i, arrayListDataValues.get(i)));
        }
```

```
        // Line configuration
        LineDataSet lineDatasetGraph = new LineDataSet(yValues, titleName +
" " + axisName);
        lineDatasetGraph.setFillAlpha(110);
        lineDatasetGraph.setColor(Color.CYAN);
        lineDatasetGraph.setLineWidth(3f);
        lineDatasetGraph.setValueTextColor(android.R.color.transparent);
        lineDatasetGraph.setCircleColor(Color.CYAN);

        // Configure
        ArrayList<ILineDataSet> dataSets = new ArrayList<>();
        dataSets.add(lineDatasetGraph);
        LineData data = new LineData(dataSets);
        mLineChart.setData(data);
    }

    /************************************  NEXT BUTTON
************************************
     * Switches to next activity (SaveDataActivity) */
    private void NextButton() {
        buttonNext.setOnClickListener(v -> {
            // Go to next activity (SaveDataActivity)
            startActivity(new Intent(this, SaveDataActivity.class));
            // Set exit and enter transitions (forward)
            overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
            finish();
        });
    }

    /************************************  BACK BUTTON
************************************
     * Switch to previous activity (DataAcquisitionActivity or
SelectSavedDataActivity)
     * If user is in data acquisition cycle, a warning is shown because all
data will be lost if
     * they go back */
    private void BackButton() {
        if
(CrutchApplication.from.equals(SelectSavedDataActivity.class.getSimpleName(
))) {
            // Comes from SelectSavedDataActivity
            buttonBack.setOnClickListener(v -> {
                // Go to previous activity in visualization cycle:
SelectSavedDataActivity
                startActivity(new Intent(this,
SelectSavedDataActivity.class));
                // Set exit and enter transitions (backward)
                overridePendingTransition(R.anim.anim_left_right_in,
R.anim.anim_fade_out);
                finish();
            });
        } else if
(CrutchApplication.from.equals(DataAcquisitionActivity.class.getSimpleName(
))) {
            // Comes from DataAcquisitionActivity
            buttonBack.setOnClickListener(v -> {
                // Ask user if they want to go back to previous activity in
acquisition cycle:
                // DataAcquisitionActivity
                dialogAlertTwoButtons();
```

```
            });
        }
    }

    /************************************  ALERT DIALOG
*************************************
     * Creates an alert dialog (small window) to ask user if they want to
go back to previous
     * activity with two options: YES and NO.
     * All data will be lost if they go back
     * YES: deletes the saved data corresponding to the current user ID and
iteration from the
     * database and switches to previous activity in data acquisition
cycle: DataAcquisitionActivity
     * NO: do nothing (don't go back) */
    private void dialogAlertTwoButtons() {
        new AlertDialog.Builder(VisualizeDataActivity.this)
                .setTitle("Are you sure you want to go back?")
                .setMessage("All data values will be deleted if you go
back!")
                .setPositiveButton("YES", (dialog, id) -> {
                    // Delete user and data values from "user" and "data"
tables
                    AdminSQLiteOpenHelper sqLiteOpenHelper =
AdminSQLiteOpenHelper.getInstance(this);
                    sqLiteOpenHelper.deleteUser(CrutchApplication.userId,
CrutchApplication.iteration);

sqLiteOpenHelper.deleteDataValues(CrutchApplication.rowIDUserTable);
                    // Go to previous activity in data acquisition cycle:
DataAcquisitionActivity
                    startActivity(new Intent(this,
DataAcquisitionActivity.class));
                    // Set exit and enter transitions (backward)
                    overridePendingTransition(R.anim.anim_left_right_in,
R.anim.anim_fade_out);
                    finish();
                    dialog.cancel();
                })
                .setNegativeButton("NO", (dialog, id) -> {
                    // do nothing
                    dialog.cancel();
                }).show();
    }


}  // </ public class VisualizeDataActivity extends AppCompatActivity >
```

SaveDataActivity.java

```
package com.visens.crutch_v2;


/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class SaveDataActivity extends AppCompatActivity {


    /* ////////////////////////////////////////   VARIABLES
//////////////////////////////////////// */
    private static boolean initialized = false;

    // Layout objects
    private TextView textViewUserID, textViewIteration,
textViewNumberOfSamples,
            textViewTimeMinutes, textViewTimeSeconds,
textViewTimeMilliSeconds,
            textViewDate, textViewErrors, textViewMissing;
    private Button buttonSaveExcelFile, buttonSendByEmail, buttonContinue,
buttonBack;

    // File variables
    private static final String EXCEL_FILES_FOLDER_NAME = "Crutch App Data
Excels";
    private String excelFileName;
    private File excelFile;
    private boolean fileWritten = false;

    // Constant for excel file header row names
    private final static ArrayList<String> DATA_NAMES = new
ArrayList<>(Arrays.asList("Hour", "Min",
            "Sec", "Millis", "Iteration 1", "Force", "Altitude", "Pitch",
"Roll", "Yaw", "AccX",
            "Iteration 2", "AccY", "Acc Z", "Gyroscope X", "Gyroscope Y",
"Gyroscope Z",
            "Magnetic X", "Magnetic Y", "Magnetic Z"));
```

```
    /* ////////////////////////////////////    METHODS
//////////////////////////////////// */

    /************************************  ON CREATE
*************************************
     * Finds layout objects */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.save_data_activity_layout);
        initialized = false;
        // Find layout objects
        textViewUserID = findViewById(R.id.idSaveDataTVUserIDValue);
        textViewIteration = findViewById(R.id.idSaveDataTVIterationValue);
        textViewDate = findViewById(R.id.idSaveDataTVDateValue);
        textViewTimeMinutes =
findViewById(R.id.idSaveDataTVTimeValueMinutes);
        textViewTimeSeconds =
findViewById(R.id.idSaveDataTVTimeValueSeconds);
        textViewTimeMilliSeconds =
findViewById(R.id.idSaveDataTVTimeValueMilliSeconds);
        textViewNumberOfSamples =
findViewById(R.id.idSaveDataTVNumberOfSamplesValue);
        textViewErrors = findViewById(R.id.idTVErrorsValue);
        textViewMissing = findViewById(R.id.idTVMissingValue);
        buttonSaveExcelFile = findViewById(R.id.idBSaveExcelFile);
        buttonSendByEmail = findViewById(R.id.idBEmail);
        buttonContinue = findViewById(R.id.idBContinue);
        buttonBack = findViewById(R.id.idSaveDataBBack);
    }

    /************************************  ON START
*************************************
     * Initializes the firebase object and calls the implemented methods */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Method calls
            showDetails();
            SaveExcelFileButton();
            SendDataByEmailButton();
            ContinueNewUserButton();
            BackButton();
            initialized = true;
        }

        Log.d("excel debug", "Save data Activity initialized");
    }

    /************************************  SHOW DETAILS
*************************************
     * Show user and data information on screen */
    private void showDetails() {
        textViewUserID.setText(CrutchApplication.userId);
        textViewIteration.setText(CrutchApplication.iteration);
        textViewDate.setText(CrutchApplication.date);
        // Extract the minutes, seconds and milliseconds
        // NOTE: modulo operator in java -> %
        long timeInMillis = Long.parseLong(CrutchApplication.time);
        long millis = timeInMillis % 1000;
```

```
        long seconds = (timeInMillis / 1000) % 60;
        long minutes = timeInMillis / 60000;
        textViewTimeMinutes.setText(String.valueOf(minutes));
        textViewTimeSeconds.setText(String.valueOf(seconds));
        textViewTimeMilliSeconds.setText(String.valueOf(millis));

textViewNumberOfSamples.setText(String.valueOf(CrutchApplication.numSamples
));

textViewErrors.setText(String.valueOf(CrutchApplication.numErrors));

textViewMissing.setText(String.valueOf(CrutchApplication.numMissing));
    }

    /**********************************  SAVE DATA IN FILE
**********************************
     * Save data in a .xls (Excel) file. This has to be done before
uploading the data to the cloud
     * (Firebase) or sending it by email */
    private void saveDataInExcelFile() {
        // TODO [future] file management has changed a lot in recent
versions of Android. This way,
        //  using
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMEN
TS) was
        //  deprecated in Android 11. The better way of doing this is by
using the Storage Access
        //  Framework. See the comment in AndroidManifest for more info

        /*
        Returns absolute paths to application-specific directories on all
shared/external storage
        devices where the application can place persistent files it owns.
These files are internal
        to the application, and not typically visible to the user as media.

https://developer.android.com/reference/android/content/Context#getExternal
FilesDirs(java.lang.String)

        final File extStorage =
getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS);
        final String extStoragePath = extStorage.getAbsolutePath();
        */

        /*
        Applications should not directly use this top-level directory, in
order to avoid polluting
        the user's root namespace. Any files that are private to the
application should be placed in
        a directory returned by Context.getExternalFilesDir, which the
system will take care of
        deleting if the application is uninstalled. Other shared files
should be placed in one of
        the directories returned by
getExternalStoragePublicDirectory(String)

https://developer.android.com/reference/android/os/Environment#getExternalS
torageDirectory()

        final File extStorage = Environment.getExternalStorageDirectory();
        */
```

```
        // TODO: Organize paths
        // TODO this paths should be static public final and maybe stored
in CrutchApplication
        final File extStorageDocumentsPath =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMEN
TS);
        final File extStorageDownloadPath =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOA
DS);

        final String externalStorageState =
Environment.getExternalStorageState();
        // Method getExternalStoragePublicDirectory was deprecated in
Android version 11 (API level 30)
        final File extStorage;
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.R) {
            extStorage =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMEN
TS);
        } else {
            extStorage =
getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS);
        }
        final String extStoragePath = extStorage.getAbsolutePath();
        final String excelFilesFolderPath = extStoragePath + "/" +
EXCEL_FILES_FOLDER_NAME;
        excelFileName = "User_" + CrutchApplication.userId + "__Iteration_"
+ CrutchApplication.iteration + ".xls";
        FileOutputStream myFileOutputStream = null;

        Log.d("excel debug", "excel folder path: " + excelFilesFolderPath);

        // Check if external storage available and not read only
        boolean isExternalStorageAvailable =
Environment.MEDIA_MOUNTED.equals(externalStorageState);
        boolean isExternalStorageReadOnly =
Environment.MEDIA_MOUNTED_READ_ONLY.equals(externalStorageState);
        if (externalStorageState == null || !isExternalStorageAvailable ||
isExternalStorageReadOnly) {
            Toast.makeText(this, "External storage not available or read
only", Toast.LENGTH_SHORT).show();
            return;
        }

        // Create excel files folder if it doesn't exist
        File excelFilesFolder = new File(excelFilesFolderPath);
        if (!excelFilesFolder.exists()) {
            try {
                boolean folderCreated = excelFilesFolder.mkdirs();
                if (folderCreated) {
                    Toast.makeText(this, "Excel files folder created",
Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(this, "Excel files folder could not be
created", Toast.LENGTH_SHORT).show();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

```
        /* TODO
            REPEATED CODE!!!!!!!!!!!!!! THIS FILE IS NOT USED BUT ANOTHER
ONE CREATED IN THE TRY-CATCH
            ASK USER IF THEY WANT TO REWRITE FILE!!!!!!!!!!!
            this is not needed anymore but can be used to erase previous
file when implementing
            asking user */
        /*
        // Create excel file
        excelFile = new File(excelFilesFolderPath, fileName);
        //if (!excelFile.exists()) {
        try {
            excelFile.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
        //}
        */

        // NEEDED LIBRARY:
http://www.apache.org/dyn/closer.cgi/poi/release/bin/poi-bin-3.9-
20121203.tar.gz
        // Official documentation:
        //   You can read and write MS Excel files using Java. In addition,
you can read and write
        //  MS Word and MS PowerPoint files using Java. Apache POI is your
Java Excel solution
        //  (for Excel 97-2008).
        // IMPORTANT: This library needs Apache's commons library to work

        // Create excel workbook and sheet
        HSSFWorkbook myHSSFWorkbook = new HSSFWorkbook();
        HSSFSheet excelSheet = myHSSFWorkbook.createSheet("USER_" +
                CrutchApplication.userId + "_ITER_" +
CrutchApplication.iteration);

        // Write column headers, (set their style) and set column width
        HSSFRow headerRow = excelSheet.createRow(0);        // create
header row
        /* TODO NOTE: can't manage to apply different style to header row
and the rest of rows
        HSSFCell headerCell = headerRow.createCell(0);              // cell
to get default style
        HSSFFont defaultCellFont = myHSSFWorkbook.createFont();     //
create default font
        defaultCellFont.setBold(false);                             // set
font bold
        defaultCellFont.setFontHeight((short) 240);                 // set
font size
        HSSFFont headerCellFont = myHSSFWorkbook.createFont();      //
create header font
        headerCellFont.setBold(true);                               // set
font bold
        headerCellFont.setFontHeight((short) 360);                  // set
font size
        HSSFCellStyle headerCellStyle = headerCell.getCellStyle();  //
extract default style
        headerCellStyle.setFont(headerCellFont);                    //
apply font changes to style
```

```
        */
        for (int ind = 0; ind < DATA_NAMES.size(); ind++) {
            HSSFCell headerCell = headerRow.createCell(ind);
            headerCell.setCellValue(DATA_NAMES.get(ind));
            excelSheet.setColumnWidth(ind, 3600);
        }

        // TODO add one more value to
CrutchApplication.dataAcquisitionStartTimes so that data acquisition
        //  intervals can be calculated using a "for" cycle
        if
(CrutchApplication.from.equals(DataAcquisitionActivity.class.getSimpleName(
))) {
            CrutchApplication.dataAcquisitionAbsoluteStartTimes.add(0L);
        }

        final int HOUR_COLUMN = 0;
        final int MIN_COLUMN = 1;
        final int SEC_COLUMN = 2;
        final int MILLIS_COLUMN = 3;
        final int ITERATION_1_COLUMN = 4;
        final int FORCE_COLUMN = 5;
        final int ALTITUDE_COLUMN = 6;
        final int ROLL_COLUMN = 7;
        final int PITCH_COLUMN = 8;
        final int YAW_COLUMN = 9;
        final int ACC_X_COLUMN = 10;
        final int ITERATION_2_COLUMN = 11;
        final int ACC_Y_COLUMN = 12;
        final int ACC_Z_COLUMN = 13;
        final int GYRO_X_COLUMN = 14;
        final int GYRO_Y_COLUMN = 15;
        final int GYRO_Z_COLUMN = 16;
        final int MAGN_X_COLUMN = 17;
        final int MAGN_Y_COLUMN = 18;
        final int MAGN_Z_COLUMN = 19;

        // TODO execution times headers
        HSSFCell headerCell = headerRow.createCell(MAGN_Z_COLUMN + 1);
        headerCell.setCellValue("Processing duration (ms)");
        excelSheet.setColumnWidth(MAGN_Z_COLUMN + 2, 3600);
        headerCell = headerRow.createCell(MAGN_Z_COLUMN + 2);
        headerCell.setCellValue("Screen update duration (ms)");
        excelSheet.setColumnWidth(MAGN_Z_COLUMN + 2, 3600);
        headerCell = headerRow.createCell(MAGN_Z_COLUMN + 3);
        headerCell.setCellValue("Data acquisition start time (ms)");
        excelSheet.setColumnWidth(MAGN_Z_COLUMN + 3, 3600);
        headerCell = headerRow.createCell(MAGN_Z_COLUMN + 4);
        headerCell.setCellValue("Data acquisition interval (ms)");
        excelSheet.setColumnWidth(MAGN_Z_COLUMN + 4, 3600);
        headerCell = headerRow.createCell(MAGN_Z_COLUMN + 5);
        headerCell.setCellValue("Saving duration (ms)");
        excelSheet.setColumnWidth(MAGN_Z_COLUMN + 5, 3600);
        headerCell = headerRow.createCell(MAGN_Z_COLUMN + 6);
        headerCell.setCellValue("Characteristic arrival (ms)");
        excelSheet.setColumnWidth(MAGN_Z_COLUMN + 6, 4200);

        // TODO cursor used to avoid ram limitations <- NOT CHECKED
        // Initialize database variable
        AdminSQLiteOpenHelper sqliteAdmin =
AdminSQLiteOpenHelper.getInstance(this);
```

```
        SQLiteDatabase database = sqliteAdmin.getReadableDatabase();
        // Get user's data values with rowID from DATA table
        Cursor cursorDataValues =
sqliteAdmin.readDataValuesCursor(CrutchApplication.rowIDUserTable,
database);
        // cursorDataValues.moveToFirst();  // This line is in
sqliteAdmin.readDataValuesCursor
        int ind = 0;
        int columnIndex;
        HSSFCell dataCell;
        HSSFRow dataRow;
        Log.d("excel debug", "Starting workbook data fill");

        //synchronized (this) {
        // Write data in excel sheet of workbook
        while (!cursorDataValues.isAfterLast()) {

            try {
                // Create new row
                dataRow = excelSheet.createRow(ind + 1);  // ind + 1
because row 0 is the headers
                // Create data cells of current row
                //  timestamp values
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL18);
                String timestamp = cursorDataValues.getString(columnIndex);
                String[] timestampSplit = timestamp.split(":");  // e.g.
13:51:27.89 -> [13, 51, 27.89]
                String[] timestampSplitSeconds =
timestampSplit[2].split("\\.");  // e.g. 27.89 -> [27, 89]

                dataCell = dataRow.createCell(HOUR_COLUMN);

dataCell.setCellValue(timestampSplit[0].substring(timestampSplit[0].length(
) - 2));
                dataCell = dataRow.createCell(MIN_COLUMN);
                dataCell.setCellValue(timestampSplit[1]);
                dataCell = dataRow.createCell(SEC_COLUMN);
                dataCell.setCellValue(timestampSplitSeconds[0]);
                dataCell = dataRow.createCell(MILLIS_COLUMN);
                dataCell.setCellValue(timestampSplitSeconds[1]);
                //  iteration_1 values
                dataCell = dataRow.createCell(ITERATION_1_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL16);

dataCell.setCellValue(cursorDataValues.getInt(columnIndex));
                dataCell = dataRow.createCell(FORCE_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL02);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(ALTITUDE_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL03);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(ROLL_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL04);
```

```
dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(PITCH_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL05);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(YAW_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL06);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(ACC_X_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL07);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                //  iteration_2 values
                dataCell = dataRow.createCell(ITERATION_2_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL17);

dataCell.setCellValue(cursorDataValues.getInt(columnIndex));
                dataCell = dataRow.createCell(ACC_Y_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL08);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(ACC_Z_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL09);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(GYRO_X_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL10);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(GYRO_Y_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL11);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(GYRO_Z_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL12);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(MAGN_X_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL13);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(MAGN_Y_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL14);

dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));
                dataCell = dataRow.createCell(MAGN_Z_COLUMN);
                columnIndex =
cursorDataValues.getColumnIndex(AdminSQLiteOpenHelper.DATA_TABLE_COL15);
```

```
dataCell.setCellValue(cursorDataValues.getFloat(columnIndex));

                // TODO manage correctly the debugging times
                // Save durations only if it comes from SaveDataActivity
                if
(CrutchApplication.from.equals(DataAcquisitionActivity.class.getSimpleName(
))) {
                    dataCell = dataRow.createCell(MAGN_Z_COLUMN + 1);
                    dataCell.setCellValue(((double)
CrutchApplication.dataProcessingDurations.get(ind)) * 0.000001);
                    dataCell = dataRow.createCell(MAGN_Z_COLUMN + 2);
                    dataCell.setCellValue(((double)
CrutchApplication.screenUpdateDurations.get(ind)) * 0.000001);
                    dataCell = dataRow.createCell(MAGN_Z_COLUMN + 3);
                    dataCell.setCellValue(((double)
CrutchApplication.dataAcquisitionAbsoluteStartTimes.get(ind)) * 0.000001);
                    dataCell = dataRow.createCell(MAGN_Z_COLUMN + 4);
                    dataCell.setCellValue(((double)
CrutchApplication.dataAcquisitionAbsoluteStartTimes.get(ind + 1) -
CrutchApplication.dataAcquisitionAbsoluteStartTimes.get(ind)) * 0.000001);
                    dataCell = dataRow.createCell(MAGN_Z_COLUMN + 5);
                    dataCell.setCellValue(((double)
CrutchApplication.dataSavingDurations.get(ind)) * 0.000001);
                    dataCell = dataRow.createCell(MAGN_Z_COLUMN + 6);
                    dataCell.setCellValue(((double)
CrutchApplication.characteristicArrivalTimes.get(ind)) * 0.000001);
                }
                ind++;
                cursorDataValues.moveToNext();

            } catch (Exception e) {
                Log.d("excel debug", "Error with workbook data fill: ");
                Log.d("excel debug", e.getMessage() + " -- " +
e.getCause());
                Log.d("excel debug", e.toString());
                e.printStackTrace();
                break;
            }

        }  // </ while (!cursorDataValues.isAfterLast()) >
        cursorDataValues.close();
        database.close();
        //}  // </ synchronized(this) >

        Log.d("excel debug", "Finalized workbook data fill");

        // TODO put this better (last column of the first line instead of
last line)
        dataRow = excelSheet.createRow(ind + 2);  // next empty line
        dataCell = dataRow.createCell(0);
        dataCell.setCellValue("Number of Errors");
        dataCell = dataRow.createCell(1);
        dataCell.setCellValue(CrutchApplication.numErrors);
        dataCell = dataRow.createCell(2);
        dataCell.setCellValue("Characteristics missing");
        dataCell = dataRow.createCell(3);
        dataCell.setCellValue(CrutchApplication.numMissing);

        // Write excel workbook in file
        try {
            Log.d("excel debug", "Trying to create file");
```

```
            excelFile = new File(excelFilesFolderPath, excelFileName);
            myFileOutputStream = new FileOutputStream(excelFile);
            Log.d("excel debug", "Trying to write to file");
            myHSSFWorkbook.write(myFileOutputStream);
            fileWritten = true;
            // Message for user
            Toast.makeText(this, "Data saved in excel file at " +
excelFilesFolderPath +
                    "/" + excelFileName, Toast.LENGTH_SHORT).show();
            Log.d("excel debug", "Excel file written");
        } catch (Exception e) {
            Log.d("excel debug", "Error writing to file " +
e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                if (myFileOutputStream != null) {
                    Log.d("excel debug", "Trying to flush stream");
                    myFileOutputStream.flush();
                    Log.d("excel debug", "Trying to close stream");
                    myFileOutputStream.close();
                }
            } catch (IOException e) {
                Log.d("excel debug", "Error flushing or closing stream");
                e.printStackTrace();
            }
        }

        Log.d("excel debug", "Finalized flushing and closing stream");
        Log.d("excel debug", "Finalized write to file");

    }  // </ public void SaveDataInXLSFile >

    /**********************************  SAVE EXCEL FILE BUTTON
**********************************
     * Sets setOnClickListener of buttonSaveExcelFile to save data in an
Excel File */
    private void SaveExcelFileButton() {
        buttonSaveExcelFile.setOnClickListener(v -> {
            if (ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)
                    == PackageManager.PERMISSION_GRANTED) {
                if (!fileWritten) {
                    saveDataInExcelFile();
                } else {
                    Toast.makeText(this, "Data already saved in Excel",
                            Toast.LENGTH_SHORT).show();
                }
            } else {
                ActivityCompat.requestPermissions(this,
                        new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                        1);
            }
        });
    }

    /**********************************  SEND DATA BY EMAIL BUTTON
**********************************
     * Send the data by email */
    private void SendDataByEmailButton() {
```

```
        buttonSendByEmail.setOnClickListener(v -> {
            // Check for external storage writing permission (needed to
save .xls file)
            if (ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)
                    == PackageManager.PERMISSION_GRANTED) {
                if (!fileWritten) {
                    saveDataInExcelFile();
                }
                String email = "";
                // Send email
                Uri myUri = Uri.fromFile(excelFile);
                Intent emailIntent = new Intent(Intent.ACTION_SENDTO);
                emailIntent.setData(Uri.parse("mailto:"));

                //emailIntent.setType("text/plain");
                emailIntent.putExtra(Intent.EXTRA_EMAIL, new
String[]{email});
                emailIntent.putExtra(Intent.EXTRA_SUBJECT, "File " +
excelFileName);
                emailIntent.putExtra(Intent.EXTRA_TEXT, "Hello, I send you
the file " +
                        excelFileName + " created on " +
CrutchApplication.date);

                //emailIntent.setType("application/xls");
                emailIntent.putExtra(Intent.EXTRA_STREAM, myUri);

                startActivity(Intent.createChooser(emailIntent, "sent"));
                Toast.makeText(getApplicationContext(), "Email sent",
Toast.LENGTH_SHORT).show();
            } else {
                ActivityCompat.requestPermissions(this,
                        new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                        1);
            }
        });
    }

    /*********************************  BUTTON CONTINUE NEW USER
*********************************
     * Switches to next activity (ConfigureNewUserActivity or Main
Activity)
     * This button has different usage if it comes from
SelectSavedDataActivity->VisualizeDataActivity
     * or DataAcquisitionActivity->VisualizeDataActivity */
    private void ContinueNewUserButton() {
        if
(CrutchApplication.from.equals(SelectSavedDataActivity.class.getSimpleName(
))) {
            // Comes from SelectSavedDataActivity->VisualizeDataActivity
            buttonContinue.setOnClickListener(v -> {
                // Go to visualization cycle's first activity:
SelectSavedDataActivity
                startActivity(new Intent(this,
SelectSavedDataActivity.class));
                // Set exit and enter transitions (forward)
                overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
                finish();
```

```
            });
        } else if
(CrutchApplication.from.equals(DataAcquisitionActivity.class.getSimpleName(
))) {
            // Comes from DataAcquisitionActivity->VisualizeDataActivity
            buttonContinue.setOnClickListener(v -> {
                // Go to data acquisition cycle's first activity:
ConfigureNewUserActivity
                CrutchApplication.from =
SaveDataActivity.class.getSimpleName();
                startActivity(new Intent(this,
ConfigureNewUserActivity.class));
                // Set exit and enter transitions (forward)
                overridePendingTransition(R.anim.anim_right_left_in,
R.anim.anim_fade_out);
                finish();
            });
        }
    }

    /************************************  BACK BUTTON
************************************
     * Switches to previous activity: VisualizeDataActivity */
    private void BackButton() {
        buttonBack.setOnClickListener(v -> {
            // Go to VisualizeDataActivity
            startActivity(new Intent(this, VisualizeDataActivity.class));
            // Set exit and enter transitions (backward)
            overridePendingTransition(R.anim.anim_left_right_in,
R.anim.anim_fade_out);
            finish();
        });
    }

}  // </ public class SaveDataActivity extends AppCompatActivity >
```

ConfigurationActivity.java

```
package com.visens.crutch_v2;

/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.widget.Button;
import android.widget.Toast;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import java.io.File;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class ConfigurationActivity extends AppCompatActivity {
```

```
    /* ////////////////////////////////////   VARIABLES
//////////////////////////////////// */
    private static boolean initialized = false;

    // Layout variables
    private Button buttonDeleteDatabase, buttonDeleteExcels, buttonReturn;

    /* ////////////////////////////////////   METHODS
//////////////////////////////////// */

    /************************************  ON CREATE
**********************************
     * Finds layout objects */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.configuration_activity_layout);
        initialized = false;
        // Find layout objects
        buttonDeleteDatabase = findViewById(R.id.idBDeleteDatabase);
        buttonDeleteExcels = findViewById(R.id.idBDeleteExcelFiles);
        buttonReturn = findViewById(R.id.idConfigurationBReturn);
    }

    /************************************  ON START
**********************************
     * Calls the implemented methods */
    @Override
    protected void onStart() {
        super.onStart();
        if (!initialized) {
            // Method calls
            DeleteDatabaseButton();
            DeleteExcelFilesButton();
            ReturnButton();
            initialized = true;
        }
    }

    /*********************************  DeleteDatabaseButton
*********************************
     * Sets the onClickListener of buttonDeleteDatabase
     * Creates an alert dialog (small message window) to ask user if they
want to erase all data
     * from database with two options: YES and NO.
     * YES: delete data from the database
     * NO: do nothing (don't delete data from database) */
    private void DeleteDatabaseButton() {
        buttonDeleteDatabase.setOnClickListener(v -> {
            new AlertDialog.Builder(this)
                    .setTitle("Delete Database Data")
                    .setMessage("Are you sure you want to delete all
database data?")
                    .setPositiveButton("YES", (dialog, id) -> {
                        // Delete database

deleteDatabase(AdminSQLiteOpenHelper.CRUTCH_DATABASE_FILE_NAME);
                        Toast.makeText(this, "All data erased from local
database",
                                Toast.LENGTH_SHORT).show();
                        dialog.cancel();
```

```
                    })
                    .setNegativeButton("NO", (dialog, id) -> {
                        // do nothing
                        dialog.cancel();
                    }).show();  // NOTE: .show goes at the end of Builder
        });
    }

    /********************************  DeleteExcelFilesButton
*********************************
     * Sets the onClickListener of buttonDeleteExcels.
     * Creates an alert dialog (small message window) to ask user if they
want to delete Excel files
     * with two options: YES and NO.
     * YES: delete Excel files
     * NO: do nothing (don't delete Excel files) */
    private void DeleteExcelFilesButton() {
        buttonDeleteExcels.setOnClickListener(v -> {
            new AlertDialog.Builder(this)
                    .setTitle("Delete Excel Files")
                    .setMessage("Are you sure you want to delete all Excel
files?")
                    .setPositiveButton("YES", (dialog, id) -> {
                        // TODO this is badly implemented because I made it
fast and it's just for debugging
                        // Delete excel files
                        final String EXCEL_FILES_FOLDER_NAME = "Crutch App
Data Excels";
                        //final File extStorage =
Environment.getExternalStorageDirectory();
                        /*
                        final File extStorage = getExternalFilesDir(null);
                        final String extStoragePath =
extStorage.getAbsolutePath();*/
                        final File extStorage =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMEN
TS);
                        final String extStoragePath =
extStorage.getAbsolutePath();
                        // TODO another form that may work
                        //final File extStorageDocumentsPath =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMEN
TS);
                        final String excelFilesFolderPath = extStoragePath
+ "/" + EXCEL_FILES_FOLDER_NAME;
                        File excelFilesFolder = new
File(excelFilesFolderPath);
                        if (excelFilesFolder.exists()) {
                            boolean deleteSuccess;
                            File[] files = excelFilesFolder.listFiles();
                            assert files != null;
                            for (File file : files) {
                                file.delete();
                            }
                            if (excelFilesFolder.listFiles() == null) {
                                Toast.makeText(this, "All Excel files
deleted from " +
                                                excelFilesFolderPath,
Toast.LENGTH_SHORT).show();
                            } else {
                                Toast.makeText(this, "Excel files could not
```

```
be deleted",
                                        Toast.LENGTH_SHORT).show();
                            }
                        } else {
                            Toast.makeText(this, "There are no Excel files
saved to be deleted",
                                    Toast.LENGTH_SHORT).show();
                        }
                        dialog.cancel();
                    })
                    .setNegativeButton("NO", (dialog, id) -> {
                        // do nothing
                        dialog.cancel();
                    }).show();  // NOTE: .show goes at the end of Builder
        });
    }

    /************************************  RETURN BUTTON
**********************************
     * Exits Configuration and switches back to MainActivity */
    private void ReturnButton() {
        buttonReturn.setOnClickListener(v -> {
            // Return to Main activity
            startActivity(new Intent(this, MainActivity.class));
            // Set exit and enter transitions (backward)
            overridePendingTransition(R.anim.anim_fade_in,
R.anim.anim_fade_out);
            finish();
        });
    }


}  // </ public class MainActivity extends AppCompatActivity >
```

AdminSQLiteOpenHelper.java

```
package com.visens.crutch_v2;


/* ////////////////////////////////////   IMPORT LIBRARIES
//////////////////////////////////// */

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;


/* ////////////////////////////////////////   MAIN CODE
//////////////////////////////////////// */
public class AdminSQLiteOpenHelper extends SQLiteOpenHelper {

    /* ---------------------------------- Database summary ----------------
------------------ */
    //  ** USER INFORMATION table **
```

```
    //      _ID | userId | iteration | date | time | numSamples | numErrors
| numMissing
    //  ** CRUTCHES MAC ADDRESSES table **
    //      _ID | MacAddress
    //  ** DATA VALUES table **
    //      _ID | _IDofUserTable | force | altitude | roll | pitch | yaw |
accX |
    //      accY | accZ | gyroX | gyroY | gyroZ | magnX | magnY | magnZ |
iteration1 |
    //      iteration2 | timestamp
    /* ------------------------------------------------------------------------
------------------ */


    /* ////////////////////////////////////   VARIABLES
//////////////////////////////////// */
    // NOTE for non-java programmers:
    //  - Public: to be accessed by other classes (MainActivity.class,
etc.)
    //  - Static: for memory efficiency (only one copy of the variable for
all the instances of the
    //      class (objects)). They have to be accessed through the class
name, not through the class
    //      instance (object) name
    //  - Final: constant value

    // Database file name
    public static final String CRUTCH_DATABASE_FILE_NAME =
"crutchDatabase.db";

    // CRUTCHES MAC ADDRESSES table
    public static final String MAC_TABLE_NAME = "MacTable";
    public static final String MAC_TABLE_COL00 = "_ID";         // row ID
    public static final String MAC_TABLE_COL01 = "MacAddress";

    // USER INFORMATION table
    public static final String USER_TABLE_NAME = "userTable";
    public static final String USER_TABLE_COL00 = "_ID";         // row ID
    public static final String USER_TABLE_COL01 = "userId";
    public static final String USER_TABLE_COL02 = "iteration";
    public static final String USER_TABLE_COL03 = "date";
    public static final String USER_TABLE_COL04 = "time";
    public static final String USER_TABLE_COL05 = "numSamples";
    public static final String USER_TABLE_COL06 = "numErrors";
    public static final String USER_TABLE_COL07 = "numMissing";

    // DATA VALUES table
    public static final String DATA_TABLE_NAME = "dataTable";
    public static final String DATA_TABLE_COL00 = "_ID";              // row
ID
    public static final String DATA_TABLE_COL01 = "_IDofUserTable";  //
corresponding row ID in user table (to retrieve corresponding information)
    public static final String DATA_TABLE_COL02 = "force";
    public static final String DATA_TABLE_COL03 = "altitude";
    public static final String DATA_TABLE_COL04 = "roll";
    public static final String DATA_TABLE_COL05 = "pitch";
    public static final String DATA_TABLE_COL06 = "yaw";
    public static final String DATA_TABLE_COL07 = "accX";
    public static final String DATA_TABLE_COL08 = "accY";
    public static final String DATA_TABLE_COL09 = "accZ";
    public static final String DATA_TABLE_COL10 = "gyroX";
```

```
    public static final String DATA_TABLE_COL11 = "gyroY";
    public static final String DATA_TABLE_COL12 = "gyroZ";
    public static final String DATA_TABLE_COL13 = "magnX";
    public static final String DATA_TABLE_COL14 = "magnY";
    public static final String DATA_TABLE_COL15 = "magnZ";
    public static final String DATA_TABLE_COL16 = "iteration1";
    public static final String DATA_TABLE_COL17 = "iteration2";
    public static final String DATA_TABLE_COL18 = "timestamp";

    // Private class object to treat this class as singleton
    private static AdminSQLiteOpenHelper myInstance = null;


    /* ////////////////////////////////////    METHODS
//////////////////////////////////////// */

    /**********************************  CONSTRUCTOR
*********************************
     * Constructor should be private to prevent direct instantiation.
     * Make call to static method "getInstance()" instead */
    public AdminSQLiteOpenHelper(Context context) {
        super(context, CRUTCH_DATABASE_FILE_NAME, null, 1);
    }

    /**********************************  GET INSTANCE
*********************************
     * Used for treating AdminSQLiteOpenHelper class as singleton so the
same instance can be used
     * across activities */
    public static AdminSQLiteOpenHelper getInstance(Context context) {
        /* Use the application context as suggested by CommonsWare.
         * This will ensure that you don't accidentally leak an Activity's
context
         * see this article for more information:
         * http://developer.android.com/resources/articles/avoiding-memory-
leaks.html */
        if (myInstance == null) {
            myInstance = new
AdminSQLiteOpenHelper(context.getApplicationContext());
        }
        return myInstance;
    }

    /**********************************  ON CREATE
*********************************/
    @Override
    public void onCreate(SQLiteDatabase database) {
        // Create table of CRUTCHES MAC ADDRESSES in the database
        database.execSQL("CREATE TABLE " + MAC_TABLE_NAME + " ( " +
// MacTable
                MAC_TABLE_COL00 + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
// _ID
                MAC_TABLE_COL01 + " TEXT )");
// MacAddress
        // Create USER INFORMATION table in the database
        database.execSQL("CREATE TABLE " + USER_TABLE_NAME + " ( " +
// userTable
                USER_TABLE_COL00 + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
// _ID
                USER_TABLE_COL01 + " TEXT, " +
// userId
```

```
                USER_TABLE_COL02 + " TEXT, " +
// iteration
                USER_TABLE_COL03 + " TEXT, " +
// date
                USER_TABLE_COL04 + " TEXT, " +
// time
                USER_TABLE_COL05 + " INTEGER, " +
// numSamples
                USER_TABLE_COL06 + " INTEGER, " +
// numErrors
                USER_TABLE_COL07 + " INTEGER )");
// numMissing
        // Create table of DATA VALUES in the database
        database.execSQL("CREATE TABLE " + DATA_TABLE_NAME + " ( " +
// dataTable
                DATA_TABLE_COL00 + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
// _ID
                DATA_TABLE_COL01 + " TEXT, " +
// _IDofUserTable
                DATA_TABLE_COL02 + " REAL, " +
// force
                DATA_TABLE_COL03 + " REAL, " +
// altitude
                DATA_TABLE_COL04 + " REAL, " +
// roll
                DATA_TABLE_COL05 + " REAL, " +
// pitch
                DATA_TABLE_COL06 + " REAL, " +
// yaw
                DATA_TABLE_COL07 + " REAL, " +
// accX
                DATA_TABLE_COL08 + " REAL, " +
// accY
                DATA_TABLE_COL09 + " REAL, " +
// accZ
                DATA_TABLE_COL10 + " REAL, " +
// gyroX
                DATA_TABLE_COL11 + " REAL, " +
// gyroY
                DATA_TABLE_COL12 + " REAL, " +
// gyroZ
                DATA_TABLE_COL13 + " REAL, " +
// magnX
                DATA_TABLE_COL14 + " REAL, " +
// magnY
                DATA_TABLE_COL15 + " REAL, " +
// magnZ
                DATA_TABLE_COL16 + " INTEGER, " +
// iteration1
                DATA_TABLE_COL17 + " INTEGER, " +
// iteration2
                DATA_TABLE_COL18 + " TEXT )");
// timestamp
    }

    /*************************************  ON UPGRADE
**************************************/
    @Override
    public void onUpgrade(SQLiteDatabase database, int oldVersion, int
newVersion) {
        // EMPTY but method onUpgrade must exist
```

```
    }

    /************************************  isMacDuplicated
**********************************
     * Checks if MAC address already stored in database */
    public boolean isMacDuplicated(String MACAddress) {
        SQLiteDatabase database = this.getReadableDatabase();
        // Filter results
        String selection = MAC_TABLE_COL01 + " = ?";
        String[] selectionArgs = {MACAddress};
        // Sorting order
        String sortOrder = MAC_TABLE_COL00 + " ASC";
        // Read values to cursor
        Cursor cursorDuplicatedMAC = database.query(
                MAC_TABLE_NAME,     // The table to query
                null,       // The array of columns to return (pass null to
get all)
                selection,          // The columns for the WHERE clause
                selectionArgs,      // The values for the WHERE clause
                null,       // don't group the rows
                null,        // don't filter by row groups
                sortOrder);         // The sort order
        // If cursor not empty (MAC duplicated) moveToFirst() returns true
        boolean MacDuplicated = cursorDuplicatedMAC.moveToFirst();
        cursorDuplicatedMAC.close();
        database.close();
        return MacDuplicated;
    }

    /************************************  findUserRowId
**********************************
     * Finds the rowID (_ID) of the given user ID and iteration. If user ID
and iteration not
     * stored in database returns -1.
     * In ConfigureNewUSerActivity would be helpful to delete data that was
stored but is garbage
     * so parameter deleteDataIfGarbage is added for data. In other calls
of this method the data
     * should only be read without deleting to avoid errors in the app,
specially in
     * DataAcquisitionActivity when the new user data is saved in the
database */
    public int findUserRowId(String userID, String iteration, boolean
deleteDataIfGarbage) {
        SQLiteDatabase database = getReadableDatabase();
        int rowID = -1;
        // Specify which columns from the database will actually be used
after the query
        String[] projection = {
                USER_TABLE_COL00,
                USER_TABLE_COL06};
        // Filter results
        String selection = USER_TABLE_COL01 + " = ? AND " +
USER_TABLE_COL02 + " = ?";
        String[] selectionArgs = {userID, iteration};
        // Sorting order
        String sortOrder = USER_TABLE_COL00 + " ASC";
        // Read values to cursor
        Cursor cursorDuplicatedUser = database.query(
                USER_TABLE_NAME,    // The table to query
                projection,         // The array of columns to return (pass
```

```
null to get all)
                selection,          // The columns for the WHERE clause
                selectionArgs,      // The values for the WHERE clause
                null,       // don't group the rows
                null,        // don't filter by row groups
                sortOrder);         // The sort order
        // If cursor not empty (user duplicated) moveToFirst() returns true
        if (cursorDuplicatedUser.moveToFirst()) {
            int columnIndex =
cursorDuplicatedUser.getColumnIndex(USER_TABLE_COL00);
            rowID = cursorDuplicatedUser.getInt(columnIndex);
            if (deleteDataIfGarbage) {
                // Only return rowID if real data was stored. Maybe due to
a crash some elements
                // were filled with garbage and no real data was stored
                columnIndex =
cursorDuplicatedUser.getColumnIndex(USER_TABLE_COL06);
                // If numErrors == -1 data is garbage so delete data and
set rowID to -1 again
                if (cursorDuplicatedUser.getInt(columnIndex) == -1) {
                    deleteUser(userID, iteration);
                    deleteDataValues(rowID);
                    rowID = -1;
                }
            }
        }
        cursorDuplicatedUser.close();
        database.close();
        return rowID;
    }

    /************************************  saveMacAddress
************************************/
    // TODO save method returns the new row ID. Maybe can be used for error
checking
    public void saveMacAddress(String newMAC) {
        SQLiteDatabase database = getWritableDatabase();
        // Insert new row with new MAC address
        ContentValues newRowDatabase = new ContentValues();
        newRowDatabase.put(MAC_TABLE_COL01, newMAC);
        database.insert(
                MAC_TABLE_NAME,
                null,
                newRowDatabase);
        database.close();
    }

    /*************************************  saveUser
*************************************/
    // TODO save method returns the new row ID. Maybe can be used for error
checking
    public void saveUser(String userId, String iteration, String date,
String time,
                         int numSamples, int numErrors, int numMissing) {
        SQLiteDatabase database = getWritableDatabase();
        // Save new user (new row) in "user" table of database
        ContentValues newRowUserTable = new ContentValues();
        newRowUserTable.put(USER_TABLE_COL01, userId);
        newRowUserTable.put(USER_TABLE_COL02, iteration);
        newRowUserTable.put(USER_TABLE_COL03, date);
        newRowUserTable.put(USER_TABLE_COL04, time);
```

```
        newRowUserTable.put(USER_TABLE_COL05, numSamples);
        newRowUserTable.put(USER_TABLE_COL06, numErrors);
        newRowUserTable.put(USER_TABLE_COL07, numMissing);
        database.insert(
                USER_TABLE_NAME,
                null,
                newRowUserTable);
        database.close();
    }

    /************************************  saveDataValues
************************************/
    // TODO save method returns the new row ID. Maybe can be used for error
checking
    public void saveDataValues(int rowIDUserTable, float fForce, float
fAltitude,
                               float fRoll, float fPitch, float fYaw,
                               float fAccX, float fAccY, float fAccZ,
                               float fGyroX, float fGyroY, float fGyroZ,
                               float fMagneticX, float fMagneticY, float
fMagneticZ,
                               int iIteration1, int iIteration2, String
timestamp) {
        SQLiteDatabase database = getWritableDatabase();
        ContentValues newRowDatabase = new ContentValues();
        newRowDatabase.put(DATA_TABLE_COL01,
String.valueOf(rowIDUserTable));
        newRowDatabase.put(DATA_TABLE_COL02, fForce);
        newRowDatabase.put(DATA_TABLE_COL03, fAltitude);
        newRowDatabase.put(DATA_TABLE_COL04, fRoll);
        newRowDatabase.put(DATA_TABLE_COL05, fPitch);
        newRowDatabase.put(DATA_TABLE_COL06, fYaw);
        newRowDatabase.put(DATA_TABLE_COL07, fAccX);
        newRowDatabase.put(DATA_TABLE_COL08, fAccY);
        newRowDatabase.put(DATA_TABLE_COL09, fAccZ);
        newRowDatabase.put(DATA_TABLE_COL10, fGyroX);
        newRowDatabase.put(DATA_TABLE_COL11, fGyroY);
        newRowDatabase.put(DATA_TABLE_COL12, fGyroZ);
        newRowDatabase.put(DATA_TABLE_COL13, fMagneticX);
        newRowDatabase.put(DATA_TABLE_COL14, fMagneticY);
        newRowDatabase.put(DATA_TABLE_COL15, fMagneticZ);
        newRowDatabase.put(DATA_TABLE_COL16, iIteration1);
        newRowDatabase.put(DATA_TABLE_COL17, iIteration2);
        newRowDatabase.put(DATA_TABLE_COL18, timestamp);
        database.insert(
                DATA_TABLE_NAME,
                null,
                newRowDatabase);
        // TODO database close removed to improve performance
        // database.close();
    }

    /************************************  updateUserRow
************************************/
    // TODO update returns the number of rows updated. This method should
return it too
    public void updateUserRow(int rowIDUserTable, String userID, String
iteration, String date,
                              String time, int numSamples, int numErrors,
int numMissing) {
        SQLiteDatabase database = getWritableDatabase();
```

```
        ContentValues updatedRowUserTable = new ContentValues();
        updatedRowUserTable.put(USER_TABLE_COL01, userID);
        updatedRowUserTable.put(USER_TABLE_COL02, iteration);
        updatedRowUserTable.put(USER_TABLE_COL03, date);
        updatedRowUserTable.put(USER_TABLE_COL04, time);
        updatedRowUserTable.put(USER_TABLE_COL05, numSamples);
        updatedRowUserTable.put(USER_TABLE_COL06, numErrors);
        updatedRowUserTable.put(USER_TABLE_COL07, numMissing);
        // Selection (which row to update)
        String selection = USER_TABLE_COL00 + " = ?";
        String[] selectionArgs = {String.valueOf(rowIDUserTable)};
        // Update row
        database.update(
                USER_TABLE_NAME,
                updatedRowUserTable,
                selection,
                selectionArgs);
        database.close();
    }

    /************************************  readMacAddresses
************************************/
    public ArrayList<String> readMacAddresses() {
        SQLiteDatabase database = getReadableDatabase();
        ArrayList<String> MACList = new ArrayList<>();
        // Sorting order
        String sortOrder = MAC_TABLE_COL00 + " ASC";
        // Read values to cursor
        Cursor cursorAllMACAddresses = database.query(
                MAC_TABLE_NAME,     // The table to query
                null,       // The array of columns to return (pass null to
get all)
                null,       // The columns for the WHERE clause
                null,    // The values for the WHERE clause
                null,       // don't group the rows
                null,        // don't filter by row groups
                sortOrder);         // The sort order
        cursorAllMACAddresses.moveToFirst();
        int columnIndex =
cursorAllMACAddresses.getColumnIndex(MAC_TABLE_COL01);
        // Save MAC addresses to ArrayList "MACList"
        while (!cursorAllMACAddresses.isAfterLast()) {
            MACList.add(cursorAllMACAddresses.getString(columnIndex));
            cursorAllMACAddresses.moveToNext();
        }
        cursorAllMACAddresses.close();
        database.close();
        return MACList;
    }

    /************************************  readUserIds
************************************/
    public ArrayList<String> readUserIds() {
        SQLiteDatabase database = getReadableDatabase();
        ArrayList<String> listUserIDs = new ArrayList<>();
        // Specify which columns from the database will actually be used
after the query
        String[] projection = {USER_TABLE_COL01};
        // Sorting order
        String sortOrder = USER_TABLE_COL01 + " ASC";
        // Read values to cursor
```

```
        Cursor cursorAllUserIDs = database.query(
                USER_TABLE_NAME,     // The table to query
                projection,          // The array of columns to return (pass
null to get all)
                null,        // The columns for the WHERE clause
                null,     // The values for the WHERE clause
                null,        // don't group the rows
                null,         // don't filter by row groups
                sortOrder);          // The sort order
        // If data available extract user IDs
        if (cursorAllUserIDs.moveToFirst()) {
            while (!cursorAllUserIDs.isAfterLast()) {
                int columnIndex =
cursorAllUserIDs.getColumnIndex(USER_TABLE_COL01);
                listUserIDs.add(cursorAllUserIDs.getString(columnIndex));
                cursorAllUserIDs.moveToNext();
            }
            // Erase duplicates
            HashSet<String> hashSetIDs = new HashSet<>(listUserIDs);
            listUserIDs.clear();
            listUserIDs.addAll(hashSetIDs);
            // Put values in descending order
            Collections.sort(listUserIDs);
        }
        cursorAllUserIDs.close();
        database.close();
        return listUserIDs;
    }

    /*********************************  readIterationsOfUserId
*********************************/
    public ArrayList<String> readIterationsOfUserId(String userId) {
        SQLiteDatabase database = getReadableDatabase();
        ArrayList<String> listIterations = new ArrayList<>();
        // Specify which columns from the database will actually be used
after the query
        String[] projection = {USER_TABLE_COL02};
        // Filter results
        String selection = USER_TABLE_COL01 + " = ?";
        String[] selectionArgs = {userId};
        // Sorting order
        String sortOrder = USER_TABLE_COL02 + " ASC";
        // Read values to cursor
        Cursor cursorIterationsOfUser = database.query(
                USER_TABLE_NAME,     // The table to query
                projection,          // The array of columns to return (pass
null to get all)
                selection,           // The columns for the WHERE clause
                selectionArgs,       // The values for the WHERE clause
                null,        // don't group the rows
                null,         // don't filter by row groups
                sortOrder);          // The sort order
        // If data available extract user IDs
        if (cursorIterationsOfUser.moveToFirst()) {
            while (!cursorIterationsOfUser.isAfterLast()) {
                int columnIndex =
cursorIterationsOfUser.getColumnIndex(USER_TABLE_COL02);

listIterations.add(cursorIterationsOfUser.getString(columnIndex));
                cursorIterationsOfUser.moveToNext();
            }
```

```
            // Put values in descending order
            Collections.sort(listIterations);
            // NOTE: No need to erase iterations duplicates (they can't
exist because there can't be
            //   the same user id and iteration in the database)
        }
        cursorIterationsOfUser.close();
        database.close();
        return listIterations;
    }

    /************************************  readUser
*************************************/
    public void readUser(String userID, String iteration) {
        SQLiteDatabase database = getReadableDatabase();
        // Filter results
        String selection = USER_TABLE_COL01 + " = ? AND " +
USER_TABLE_COL02 + " = ?";
        String[] selectionArgs = {userID, iteration};
        // Sorting order
        String sortOrder = USER_TABLE_COL00 + " ASC";
        // Read values to cursor
        Cursor cursorUserValues = database.query(
                USER_TABLE_NAME,     // The table to query
                null,      // The array of columns to return (pass null to
get all)
                selection,          // The columns for the WHERE clause
                selectionArgs,      // The values for the WHERE clause
                null,      // don't group the rows
                null,       // don't filter by row groups
                sortOrder);         // The sort order
        cursorUserValues.moveToFirst();
        // Extract rowID of user for searching corresponding user's data in
DATA table
        int columnIndex =
cursorUserValues.getColumnIndex(USER_TABLE_COL00);
        CrutchApplication.rowIDUserTable =
cursorUserValues.getInt(columnIndex);
        // Extract time, date and errors
        columnIndex = cursorUserValues.getColumnIndex(USER_TABLE_COL03);
        CrutchApplication.date = cursorUserValues.getString(columnIndex);
        columnIndex = cursorUserValues.getColumnIndex(USER_TABLE_COL04);
        CrutchApplication.time = cursorUserValues.getString(columnIndex);
        columnIndex = cursorUserValues.getColumnIndex(USER_TABLE_COL05);
        CrutchApplication.numSamples =
cursorUserValues.getInt(columnIndex);
        columnIndex = cursorUserValues.getColumnIndex(USER_TABLE_COL06);
        CrutchApplication.numErrors = cursorUserValues.getInt(columnIndex);
        columnIndex = cursorUserValues.getColumnIndex(USER_TABLE_COL07);
        CrutchApplication.numMissing =
cursorUserValues.getInt(columnIndex);
        cursorUserValues.close();
        database.close();
    }

    /*********************************  readIterationValues
**********************************/
    public void readIterationValues(int rowIDUserTable, ArrayList<Integer>
ArrayIteration1,
                                    ArrayList<Integer> ArrayIteration2) {
        SQLiteDatabase database = getReadableDatabase();
```

```
        // Specify which columns from the database will actually be used
after the query
        String[] projection = {
                DATA_TABLE_COL16,
                DATA_TABLE_COL17
        };
        // Filter results
        String selection = DATA_TABLE_COL01 + " = ?";
        String[] selectionArgs = {String.valueOf(rowIDUserTable)};
        // Sorting order
        String sortOrder = DATA_TABLE_COL00 + " ASC";
        // Read values to cursor
        Cursor cursorDataValues = database.query(
                DATA_TABLE_NAME,    // The table to query
                projection,         // The array of columns to return (pass
null to get all)
                selection,          // The columns for the WHERE clause
                selectionArgs,      // The values for the WHERE clause
                null,       // don't group the rows
                null,        // don't filter by row groups
                sortOrder);         // The sort order
        cursorDataValues.moveToFirst();
        int columnIndex;
        while (!cursorDataValues.isAfterLast()) {
            columnIndex =
cursorDataValues.getColumnIndex(DATA_TABLE_COL16);
            ArrayIteration1.add(cursorDataValues.getInt(columnIndex));
            columnIndex =
cursorDataValues.getColumnIndex(DATA_TABLE_COL17);
            ArrayIteration2.add(cursorDataValues.getInt(columnIndex));
            cursorDataValues.moveToNext();
        }
        cursorDataValues.close();
        database.close();
    }

    /*********************************  readDataValuesCursor
*********************************/
    public Cursor readDataValuesCursor(int rowIDUserTable, SQLiteDatabase
database) {
        // !!! NOTE: database is passed as a parameter because it must
remain opened until the
        // operations with the cursor have finished (to avoid crash)
        // Filter results
        String selection = DATA_TABLE_COL01 + " = ?";
        String[] selectionArgs = {String.valueOf(rowIDUserTable)};
        // Sorting order
        String sortOrder = DATA_TABLE_COL00 + " ASC";
        // Read values to cursor
        Cursor cursorDataValues = database.query(
                DATA_TABLE_NAME,    // The table to query
                null,       // The array of columns to return (pass null to
get all)
                selection,          // The columns for the WHERE clause
                selectionArgs,      // The values for the WHERE clause
                null,       // don't group the rows
                null,        // don't filter by row groups
                sortOrder);         // The sort order
        cursorDataValues.moveToFirst();

        return cursorDataValues;
```

```
    }

    /************************************  deleteUser
************************************/
    // TODO delete returns the number of rows deleted. This method should
return it too
    public void deleteUser(String userID, String iteration) {
        SQLiteDatabase database = getWritableDatabase();
        // Selection (which row to update)
        String selection = USER_TABLE_COL01 + " = ? AND " +
USER_TABLE_COL02 + " = ?";
        String[] selectionArgs = {userID, iteration};
        // Delete user data from "user" table of database
        database.delete(
                USER_TABLE_NAME,
                selection,
                selectionArgs);
        database.close();
    }

    /**********************************  deleteDataValues
**********************************/
    // TODO delete returns the number of rows deleted. This method should
return it too
    public void deleteDataValues(int duplicatedUserRow) {
        SQLiteDatabase database = getWritableDatabase();
        // Selection (which row to update)
        String selection = DATA_TABLE_COL01 + " = ?";
        String[] selectionArgs = {String.valueOf(duplicatedUserRow)};
        // Delete user data from "user" table of database
        database.delete(
                DATA_TABLE_NAME,
                selection,
                selectionArgs);
        database.close();
    }


}  // </ public class AdminSQLiteOpenHelper extends SQLiteOpenHelper >
```

BluetoothLeService.java

```
package com.visens.crutch_v2;

// NOTE: If Android version >= 12, runtime permissions are needed
// Use this code:
// if (Build.VERSION.SDK_INT > Build.VERSION_CODES.S) {
//      if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.BLUETOOTH_SCAN)
//                    != PackageManager.PERMISSION_GRANTED) {
//          ActivityCompat.requestPermissions(this,
//                        new String[]{Manifest.permission.BLUETOOTH_SCAN},
//                        5);
//       }
//       if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.BLUETOOTH_CONNECT)
//          != PackageManager.PERMISSION_GRANTED) {
//                ActivityCompat.requestPermissions(this,
//                        new
String[]{Manifest.permission.BLUETOOTH_CONNECT},
```

```
//                          5);
//        }
// }

/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.app.Activity;
import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.bluetooth.le.BluetoothLeScanner;
import android.bluetooth.le.ScanCallback;
import android.bluetooth.le.ScanFilter;
import android.bluetooth.le.ScanResult;
import android.bluetooth.le.ScanSettings;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.location.LocationManager;
import android.os.Binder;
import android.os.IBinder;
import android.os.SystemClock;
import android.provider.Settings;
import android.util.Log;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.concurrent.locks.ReentrantLock;


/**
 * Service for managing connection and data communication with a GATT
server hosted on a
 * given Bluetooth LE device.
 */
public class BluetoothLeService extends Service {

    /* ////////////////////////////////////   VARIABLES
////////////////////////////////////// */
    private final static String TAG = "ble-debug";

    // Bluetooth variables
    private BluetoothManager mBluetoothManager;
    private BluetoothAdapter mBluetoothAdapter;
    public static String mBluetoothDeviceAddress;
    private BluetoothGatt mBluetoothGatt;
    private BluetoothLeScanner mBleScanner;
    public static int mConnectionState;  // starting value assigned in
onCreate because otherwise "Illegal forward reference" error pops out
    private boolean deviceScanned = false;  // before connecting to ble
device must scan for it
```

```
    private byte[] bytesCharacteristic_1 = new byte[20];
    private byte[] bytesCharacteristic_2 = new byte[20];
    private int numNotifications = 0;
    private int numUuids = 0;
    private boolean notificationEnabled;
    private ArrayList<UUID> arrayCharacteristicsUuids = new ArrayList<>();

    // Internal variable for not sending broadcasts if Service is stopping
    boolean stoppingService = false;

    // Constants
    private static final int STATE_DISCONNECTED = 0;
    private static final int STATE_CONNECTING = 1;
    private static final int STATE_CONNECTED = 2;

    public static final int BT_REQUEST_CODE = 0;
    public static final int LOCATION_REQUEST_CODE = 1;

    public final static String ACTION_GATT_CONNECTED =
            "com.example.bluetooth.le.ACTION_GATT_CONNECTED";
    public final static String ACTION_GATT_DISCONNECTED =
            "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED";
    public final static String ACTION_GATT_SERVICES_DISCOVERED =
            "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED";
    public final static String ACTION_DATA_AVAILABLE =
            "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";

    //  UUIDs (Universally Unique Identifier: 128-bit number used to
identify services, characteristics and descriptors)
    public final static UUID serviceUUID = UUID.fromString("00001000-0000-
1000-8000-00805f9b34fb");
    public final static UUID characteristic_1_UUID =
UUID.fromString("00001001-0000-1000-8000-00805f9b34fb");
    public final static UUID characteristic_2_UUID =
UUID.fromString("00001002-0000-1000-8000-00805f9b34fb");
    public final static UUID CCCD_UUID = UUID.fromString("00002902-0000-
1000-8000-00805f9b34fb");
    // NOTE: CCCD = Client Characteristic Communication Descriptor

    public static final String EXTRA_ITERATION1 = "extra_iteration1";
    public static final String EXTRA_ITERATION2 = "extra_iteration2";
    public static final String EXTRA_FORCE = "extra_force";
    public static final String EXTRA_ALTITUDE = "extra_altitude";
    public static final String EXTRA_ROLL = "extra_roll";
    public static final String EXTRA_PITCH = "extra_pitch";
    public static final String EXTRA_YAW = "extra_yaw";
    public static final String EXTRA_GYROX = "extra_gyrox";
    public static final String EXTRA_GYROY = "extra_gyroy";
    public static final String EXTRA_GYROZ = "extra_gyroz";
    public static final String EXTRA_ACCX = "extra_accx";
    public static final String EXTRA_ACCY = "extra_accy";
    public static final String EXTRA_ACCZ = "extra_accz";
    public static final String EXTRA_MAGNX = "extra_magnx";
    public static final String EXTRA_MAGNY = "extra_magny";
    public static final String EXTRA_MAGNZ = "extra_magnz";


    /*****************************  INSTANCE OF BluetoothGattCallback
*****************************
     * Implements callback methods for GATT events that the app cares
about. For example,
```

```
     * connection change and services discovered */
    private final BluetoothGattCallback mGattCallback = new
BluetoothGattCallback() {

        /**  onConnectionStateChange  */
        @Override
        public void onConnectionStateChange(BluetoothGatt gatt, int status,
int newState) {
            Log.d("ble-debug", "New state " + newState +
                    " (0-Disconnected 1-Connecting 2-Connected 3-
Disconnecting)");
            String intentAction;
            if (newState == STATE_CONNECTED) {
                mBluetoothDeviceAddress = gatt.getDevice().getAddress();
                Log.d("ble-debug", "sending broadcast: connected");
                mConnectionState = STATE_CONNECTED;
                intentAction = ACTION_GATT_CONNECTED;
                broadcastUpdate(intentAction);
                // Attempts to discover services after successful
connection
                mBluetoothGatt.discoverServices();
            } else if (newState == STATE_DISCONNECTED) {
                mConnectionState = STATE_DISCONNECTED;
                intentAction = ACTION_GATT_DISCONNECTED;
                if (!stoppingService) {
                    Log.d("ble-debug", "sending broadcast: disconnected");
                    broadcastUpdate(intentAction);
                }
                closeGatt();
            }
        }

        /**  onServicesDiscovered  */
        @Override
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
                Log.d("ble-debug", "sending broadcast: services
discovered");
                broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
            } else {
                Log.w(TAG, "onServicesDiscovered received: " + status);
            }
        }

        /**  onCharacteristicChanged  */
        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt,
                                            BluetoothGattCharacteristic
characteristic) {
            Log.d("ble-debug", "characteristic changed!");
            if (CrutchApplication.inicioRecepcionDatos == 0L) {
                CrutchApplication.inicioRecepcionDatos =
SystemClock.elapsedRealtimeNanos();
                CrutchApplication.characteristicArrivalTimes.add(0L);
            }
            // TODO dataAcquisitionRunning
            if (DataAcquisitionActivity.dataAcquisitionRunning) {
                if (characteristic.getUuid().equals(characteristic_1_UUID))
{
                    bytesCharacteristic_1 = characteristic.getValue();
                } else if
```

```
(characteristic.getUuid().equals(characteristic_2_UUID)) {
                    // TODO characteristic arrival times
                    CrutchApplication.characteristicArrivalTimes.add(
                            SystemClock.elapsedRealtimeNanos() -
CrutchApplication.inicioRecepcionDatos);
                    bytesCharacteristic_2 = characteristic.getValue();
                    Log.d("ble-debug", "sending broadcast: characteristic 2
changed " +
                            "(data available)");
                    broadcastUpdateWithProcessing(ACTION_DATA_AVAILABLE);
                }
            }
        }

        @Override
        public void onDescriptorWrite(BluetoothGatt gatt,
BluetoothGattDescriptor descriptor,
                                      int status) {
            numNotifications--;
            if (numNotifications > 0) {
                setCharacteristicsNotification(arrayCharacteristicsUuids,
notificationEnabled);
            }
            Log.d(TAG, "descriptor written!");
        }

    };  // </ private final BluetoothGattCallback mGattCallback >

    /*****************************  INSTANCE OF mLeScanCallback
*****************************
     * Implements callback methods for LeScanner
     * NOTE: it's very specific for this app */
    private final ScanCallback mLeScanCallback = new ScanCallback() {
        @Override
        public void onScanResult(int callbackType, ScanResult result) {
            super.onScanResult(callbackType, result);
            BluetoothDevice device = result.getDevice();
            // Get device MAC address
            String sBLEDeviceAddress = device.getAddress();
            Log.d("ble-debug", "Device found. MAC: " + sBLEDeviceAddress);
            // NOTE: scan filter is set for the MAC address of the desired
device on startScan
            if (!deviceScanned) {
                Log.d(TAG, "attempting to connect");
                // Stop scan and connect
                mBleScanner.stopScan(mLeScanCallback);
                mBluetoothGatt =
device.connectGatt(BluetoothLeService.this,
                        false, mGattCallback);
                Log.d(TAG, "Trying to create a new connection.");
                mConnectionState = STATE_CONNECTING;
                deviceScanned = true;
            }
        }  // </ end public void onScanResult(int callbackType, ScanResult
result) >

        @Override
        public void onScanFailed(int errorCode) {
            // for debugging
            Log.d("ble-debug", "Scan failed. Error: " + errorCode);
        }
```

```
    };  // </ private final ScanCallback myLeScanCallback = new
ScanCallback() >

    // Custom Binder Class
    public class LocalBinder extends Binder {
        BluetoothLeService getService() {
            return BluetoothLeService.this;
        }
    }

    // Instance of Binder
    private final IBinder mBinder = new LocalBinder();

    // BroadcastReceiver that handles Bluetooth disabled action
    private final BroadcastReceiver mGattUpdateReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (BluetoothAdapter.ACTION_STATE_CHANGED.equals(action)) {
                if (intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, -1)
                        == BluetoothAdapter.STATE_OFF) {
                    mConnectionState = STATE_DISCONNECTED;
                    broadcastUpdate(ACTION_GATT_DISCONNECTED);
                    closeGatt();
                }
            }
        }
    };

    /* ////////////////////////////////////   METHODS
//////////////////////////////////// */

    /*********************************  makeGattUpdateIntentFilter
*******************************
     * Register for BroadcastReceiver */
    private static IntentFilter makeGattUpdateIntentFilter() {
        final IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
        intentFilter.addAction(LocationManager.MODE_CHANGED_ACTION);
        return intentFilter;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "BLE onStartCommand !!!");
        mConnectionState = STATE_DISCONNECTED;
        stoppingService = false;
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d(TAG, "BLE service created!!!!!");
        registerReceiver(mGattUpdateReceiver,
makeGattUpdateIntentFilter());
    }

    // TODO [future?] check if this works. probably not so try binding
```

```
every Activity to the service
    /** Important: sometimes the service is stopped when the app is still
running. The documentation
     * says that Android may stop the service to free memory in some cases.
Maybe one of those cases
     * is when there is no activity bound to the service even if the
service is started. In this
     * case the service is started again and tries to reconnect to previous
MAC address */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "BLE service destroyed!!!!!!!!!!!!!!!!");
        // TODO [future?] should still unregister because after starting
again onCreate is called
        //  and there registerReceiver is called?
        if (stoppingService) {
            unregisterReceiver(mGattUpdateReceiver);
        } else {
            Log.d(TAG, "BLE service should not be destroyed because app is
still running!!! " +
                    "Staring service again and reconnecting");
            // disconnect to reset variables
            disconnectGatt();
            startService(new Intent(getApplicationContext(),
BluetoothLeService.class));
        }
    }

    /************************************  onTaskRemoved
************************************
     * Stops the service.
     * When the service is still running and the user removes the task that
comes from the service's
     *  application this callback is triggered.
     * ALTERNATIVE: Set ServiceInfo.FLAG_STOP_WITH_TASK in the <service>
declaration of the manifest
     *  will not trigger this callback; instead, the service will simply be
stopped */
    @Override
    public void onTaskRemoved(Intent rootIntent) {
        super.onTaskRemoved(rootIntent);
        Log.d(TAG, "BLE service stopping itself!!");
        // NOTE: this line is moved to disconnectGatt because that method
is only called when the
        // service is stopping and that can happen when the app is being
closed through the ExitApp
        // button in MainActivity
        // stoppingService = true;
        disconnectGatt();
        stopSelf();
        Log.d(TAG, "BLE service stopped!!");
    }

    /************************************  onBind
************************************/
    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }
```

```
    /*
    /*************************************  onUnBind
*************************************/
    /*@Override
    public boolean onUnbind(Intent intent) {
        return super.onUnbind(intent);
    }*/


    /***********************************  broadcastUpdate
***********************************/
    private void broadcastUpdate(final String action) {
        final Intent intent = new Intent(action);
        sendBroadcast(intent);
    }

    /*********************************  EXTRACT BITS FROM BYTE
*********************************
     * Extract k bits of a byte dataByte starting from position p and
return the value as integer
     * NOTE: integer types are used because most math operations are
performed over integer values;
     * if the variable is another type (e.g. short) the conversion is done
internally
     * IMPORTANT: k+p <= 8 always (because 1 Byte = 8 bits)
     * NOTE: positions start at 0 and are counted from left to right (less
significant to more
     * significant digit)
     * NOTE for non java programmers: << and >> are bit displacement
operators
     * ------------------------------------------------------------------------
------------------------
     * E.g.  dataByte = 0110_1111;  k = 6;  p = 2  =>  return 0001_1011
     * 1) Number 1 in binary shifted k positions to the left (e.g.
0000_0001 << 6 = 0100_0000)
     * 2) Subtract 1 from the result of (1<<k) to obtain the mask (0s and
1s at desired positions)
     * (e.g. 0100_0000 - 0000_00001 = 0011_1111)
     * 3) dataByte shifted p positions to the right (e.g. 0110_1111 >> 2 =
1101_1011)
     * 4) AND between previous values (mask and dataByte shifted)
     * (e.g. 0011_1111 & 1101_1011 = 0001_1011) */
    private static int extractBitsFromByte(byte dataByte, int k, int p) {
        return (((1 << k) - 1) & (dataByte >> p));
    }

    /***********************************  broadcastUpdate
***********************************
     * Gets the value of the characteristic for passing it as an extra in
the intent
     */
    private void broadcastUpdateWithProcessing(final String action) {
        final Intent intent = new Intent(action);

        ReentrantLock myLock = new ReentrantLock();
        myLock.lock();
        //synchronized (this) {

        //long inicioTiempo = System.nanoTime();
        long inicioTiempo = SystemClock.elapsedRealtimeNanos();
```

```
        // Extract bits corresponding to each variable
        // NOTE: | symbol -> OR function to combine all the bits from
different bytes (e.g. 001 | 101 = 101)
        // NOTE 2: Byte positions go from 0 (left, less significant bit) to
7 (right, more significant bit)

        //  - Characteristic 1
        // Iteration 1: 4 bits (1/2 Byte)
        int iIteration1 = extractBitsFromByte(bytesCharacteristic_1[0], 4,
4);
        // Force: 16 bits (2 Bytes)
            /* TODO la máscara (& 255) es necesaria porque lasvariables son
Bytes (8 bits) dentro de
                un array de Bytes y los valores finales son mayores que 1
Byte (la fuerza, por
                ejemplo, 2 Bytes) */
        int iForce = (extractBitsFromByte(bytesCharacteristic_1[0], 4, 0)
<< 12 |
                (bytesCharacteristic_1[1] & 255) << 4 |
                extractBitsFromByte(bytesCharacteristic_1[2], 4, 4));
        // Altitude: 32 bits (4 Bytes)
        int iAltitude = extractBitsFromByte(bytesCharacteristic_1[2], 4, 0)
<< 28 |
                (bytesCharacteristic_1[3] & 255) << 20 |
                (bytesCharacteristic_1[4] & 255) << 12 |
                (bytesCharacteristic_1[5] & 255) << 4 |
                extractBitsFromByte(bytesCharacteristic_1[6], 4, 4);
        // Roll: 26 bits (3 Bytes + 2 bits)
        int iRoll = extractBitsFromByte(bytesCharacteristic_1[6], 4, 0) <<
22 |
                (bytesCharacteristic_1[7] & 255) << 14 |
                (bytesCharacteristic_1[8] & 255) << 6 |
                extractBitsFromByte(bytesCharacteristic_1[9], 6, 2);
        // Pitch: 26 bits (3 Bytes + 2 bits)
        int iPitch = extractBitsFromByte(bytesCharacteristic_1[9], 2, 0) <<
24 |
                (bytesCharacteristic_1[10] & 255) << 16 |
                (bytesCharacteristic_1[11] & 255) << 8 |
                (bytesCharacteristic_1[12] & 255);
        // Yaw: 26 bits (3 Bytes + 2 bits)
        int iYaw = (bytesCharacteristic_1[13] & 255) << 18 |
                (bytesCharacteristic_1[14] & 255) << 10 |
                (bytesCharacteristic_1[15] & 255) << 2 |
                extractBitsFromByte(bytesCharacteristic_1[16], 2, 6);
        // Acceleration X: 25 bits (3 Bytes + 1 bit)
        int iAccX = extractBitsFromByte(bytesCharacteristic_1[16], 6, 0) <<
19 |
                (bytesCharacteristic_1[17] & 255) << 11 |
                (bytesCharacteristic_1[18] & 255) << 3 |
                extractBitsFromByte(bytesCharacteristic_1[19], 3, 5);
        //  - Characteristic 2
        // Iteration 2: 4 bits (1/2 Byte)
        int iIteration2 = extractBitsFromByte(bytesCharacteristic_2[0], 4,
4);
        // Acceleration Y: 25 bits (3 Bytes + 1 bit)
        // NOTE: part of this value was in bytesCharacteristic_1
        int iAccY = extractBitsFromByte(bytesCharacteristic_1[19], 5, 0) <<
20 |
                extractBitsFromByte(bytesCharacteristic_2[0], 4, 0) << 16 |
                (bytesCharacteristic_2[1] & 255) << 8 |
                (bytesCharacteristic_2[2] & 255);
```

```
        // Acceleration Z: 25 bits (3 Bytes + 1 bit)
        int iAccZ = (bytesCharacteristic_2[3] & 255) << 17 |
                (bytesCharacteristic_2[4] & 255) << 9 |
                (bytesCharacteristic_2[5] & 255) << 1 |
                extractBitsFromByte(bytesCharacteristic_2[6], 1, 7);
        // Gyroscope X: 20 bits (2 Bytes + 4 bits)
        int iGyroX = extractBitsFromByte(bytesCharacteristic_2[6], 7, 0) <<
13 |
                (bytesCharacteristic_2[7] & 255) << 5 |
                extractBitsFromByte(bytesCharacteristic_2[8], 5, 3);
        // Gyroscope Y: 20 bits (2 Bytes + 4 bits)
        int iGyroY = extractBitsFromByte(bytesCharacteristic_2[8], 3, 0) <<
17 |
                (bytesCharacteristic_2[9] & 255) << 9 |
                (bytesCharacteristic_2[10] & 255) << 1 |
                extractBitsFromByte(bytesCharacteristic_2[11], 1, 7);
        // Gyroscope Z: 20 bits (2 Bytes + 4 bits)
        int iGyroZ = extractBitsFromByte(bytesCharacteristic_2[11], 7, 0)
<< 13 |
                (bytesCharacteristic_2[12] & 255) << 5 |
                extractBitsFromByte(bytesCharacteristic_2[13], 5, 3);
        // Magnetic field X: 16 bits (2 Bytes)
        int iMagneticX = extractBitsFromByte(bytesCharacteristic_2[13], 3,
0) << 13 |
                (bytesCharacteristic_2[14] & 255) << 5 |
                extractBitsFromByte(bytesCharacteristic_2[15], 5, 3);
        // Magnetic field Y: 16 bits (2 Bytes)
        int iMagneticY = extractBitsFromByte(bytesCharacteristic_2[15], 3,
0) << 13 |
                (bytesCharacteristic_2[16] & 255) << 5 |
                extractBitsFromByte(bytesCharacteristic_2[17], 5, 3);
        // Magnetic field Z: 16 bits (2 Bytes)
        int iMagneticZ = extractBitsFromByte(bytesCharacteristic_2[17], 3,
0) << 13 |
                (bytesCharacteristic_2[18] & 255) << 5 |
                extractBitsFromByte(bytesCharacteristic_2[19], 5, 3);

        // Convert all values from integer to float to make operations
(except iteration
        // which is a value to check if data is received correctly)
        // NOTE: force and altitude conversion different from others
        // NOTE: first apply mask to extract desired bits only

            /* TODO las máscaras (& numHexadecimal) son porque las
variables son integers,
                por lo que tienen más capacidad que la que tienen que
guardar (MEMORIA INEFICIENTE)
                y se llenan de basura */
        // - Characteristic 1
        // Force
        float fForce = (float) (iForce / (1024 / 3.3));
        // Altitude
        float fAltitude = Float.intBitsToFloat(iAltitude);
        // Euler's angle: Roll (26 bits)
        iRoll = (iRoll & 0x03FFFFFF);
        if ((iRoll & 0x02000000) == 0x02000000) {
            iRoll = iRoll | 0xFC000000;
        }
        float fRoll = (float) iRoll / 10000;
        // Euler's angle: Pitch (26 bits)
        iPitch = (iPitch & 0x03FFFFFF);
```

```
        if ((iPitch & 0x02000000) == 0x02000000) {
            iPitch = (iPitch | 0xFC000000);
        }
        float fPitch = (float) iPitch / 10000;
        // Euler's angle: Yaw (26 bits)
        iYaw = (iYaw & 0x03FFFFFF);
        if ((iYaw & 0x02000000) == 0x02000000) {
            iYaw = (iYaw | 0xFC000000);
        }
        float fYaw = (float) iYaw / 10000;
        // Acceleration X (25 bits)
        iAccX = (iAccX & 0x01FFFFFF);
        if ((iAccX & 0x01000000) == 0x01000000) {
            iAccX = (iAccX | 0xFE000000);
        }
        float fAccX = (float) iAccX / 10000;
        // - Characteristic 2
        // Acceleration Y (25 bits)
        iAccY = (iAccY & 0x01FFFFFF);
        if ((iAccY & 0x01000000) == 0x01000000) {
            iAccY = (iAccY | 0xFE000000);
        }
        float fAccY = (float) iAccY / 10000;
        // Acceleration Z (25 bits)
        iAccZ = (iAccZ & 0x01FFFFFF);
        if ((iAccZ & 0x01000000) == 0x01000000) {
            iAccZ = (iAccZ | 0xFE000000);
        }
        float fAccZ = (float) iAccZ / 10000;
        // Gyroscope X (20 bits)
        iGyroX = (iGyroX & 0x000FFFFF);
        if ((iGyroX & 0x00080000) == 0x00080000) {
            iGyroX = (iGyroX | 0xFFF00000);
        }
        float fGyroX = (float) iGyroX / 10000;
        // Gyroscope Y (20 bits)
        iGyroY = (iGyroY & 0x000FFFFF);
        if ((iGyroY & 0x00080000) == 0x00080000) {
            iGyroY = (iGyroY | 0xFFF00000);
        }
        float fGyroY = (float) iGyroY / 10000;
        // Gyroscope Z (20 bits)
        iGyroZ = (iGyroZ & 0x000FFFFF);
        if ((iGyroZ & 0x00080000) == 0x00080000) {
            iGyroZ = (iGyroZ | 0xFFF00000);
        }
        float fGyroZ = (float) iGyroZ / 10000;
        // Magnetic field X (16 bits)
        iMagneticX = (iMagneticX & 0x0000FFFF);
        if ((iMagneticX & 0x00008000) == 0x00008000) {
            iMagneticX = (iMagneticX | 0xFFFF0000);
        }
        float fMagneticX = (float) iMagneticX / 10000;
        // Magnetic field Y (16 bits)
        iMagneticY = (iMagneticY & 0x0000FFFF);
        if ((iMagneticY & 0x00008000) == 0x00008000) {
            iMagneticY = (iMagneticY | 0xFFFF0000);
        }
        float fMagneticY = (float) iMagneticY / 10000;
        // Magnetic field Z (16 bits)
        iMagneticZ = (iMagneticZ & 0x0000FFFF);
```

```
        if ((iMagneticZ & 0x00008000) == 0x00008000) {
            iMagneticZ = (iMagneticZ | 0xFFFF0000);
        }
        float fMagneticZ = (float) iMagneticZ / 10000;

        //long finTiempo = System.nanoTime();
        long finTiempo = SystemClock.elapsedRealtimeNanos();
        CrutchApplication.dataProcessingDurations.add(finTiempo -
inicioTiempo);

        // TODO unlock here or at the end???
        myLock.unlock();

        intent.putExtra(EXTRA_ITERATION1, iIteration1);
        intent.putExtra(EXTRA_ITERATION2, iIteration2);
        intent.putExtra(EXTRA_FORCE, fForce);
        intent.putExtra(EXTRA_ALTITUDE, fAltitude);
        intent.putExtra(EXTRA_ROLL, fRoll);
        intent.putExtra(EXTRA_PITCH, fPitch);
        intent.putExtra(EXTRA_YAW, fYaw);
        intent.putExtra(EXTRA_ACCX, fAccX);
        intent.putExtra(EXTRA_ACCY, fAccY);
        intent.putExtra(EXTRA_ACCZ, fAccZ);
        intent.putExtra(EXTRA_GYROX, fGyroX);
        intent.putExtra(EXTRA_GYROY, fGyroY);
        intent.putExtra(EXTRA_GYROZ, fGyroZ);
        intent.putExtra(EXTRA_MAGNX, fMagneticX);
        intent.putExtra(EXTRA_MAGNY, fMagneticY);
        intent.putExtra(EXTRA_MAGNZ, fMagneticZ);
        sendBroadcast(intent);
    }

    /*********************************  initializeBluetoothObjects
*********************************
     * Initializes a reference to the local Bluetooth adapter */
    public boolean initializeBluetoothObjects() {
        if (mBluetoothManager == null) {
            mBluetoothManager = (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);
            if (mBluetoothManager == null) {
                Log.e(TAG, "Unable to initialize BluetoothManager.");
                return false;
            }
        }
        if (mBluetoothAdapter == null) {
            mBluetoothAdapter = mBluetoothManager.getAdapter();
            if (mBluetoothAdapter == null) {
                Log.e(TAG, "Unable to obtain a BluetoothAdapter.");
                return false;
            }
        }
        return true;
    }

    /*********************************  checkBluetoothEnabled
*********************************
     * Checks if Bluetooth enabled. If not, ask user to enable it
     */
    public boolean checkBluetoothEnabled(Activity activity) {
        // NOTE: first check mBluetoothAdapter == null prevents null
exceptions hen checking
```

```
        //  !mBluetoothAdapter.isEnabled() in case of mBluetoothAdapter =
null by using the "||"
        //  operator instead of "|"
        if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
            Log.d("ble-debug", "bluetooth disabled");
            Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            activity.startActivityForResult(enableBtIntent,
BT_REQUEST_CODE, null);
            return false;
        } else {
            Log.d("ble-debug", "bluetooth enabled");
            mBleScanner = mBluetoothAdapter.getBluetoothLeScanner();
            return true;
        }
    }

    /*********************************  checkBluetoothEnabled
*********************************
     * Checks if Bluetooth enabled. If not, ask user to enable it
     */
    public boolean checkLocationEnabled(Activity activity) {
        LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
        assert locationManager != null;
        if
(locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            return true;
        } else {
            // TODO [future] maybe location enabled only if Android > ... ?
            Intent enableLocationIntent = new
Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            activity.startActivityForResult(enableLocationIntent,
LOCATION_REQUEST_CODE, null);
            return false;
        }
    }

    /*************************************  SCAN DEVICE
*************************************
     * For apps targeting Build.VERSION_CODES#R or lower, this requires the
     *  Manifest.permission#BLUETOOTH_ADMIN permission which can be gained
with a simple
     *  <uses-permission> manifest tag.
     * For apps targeting Build.VERSION_CODES#S or or higher, this requires
the
     *  Manifest.permission#BLUETOOTH_SCAN permission which can be gained
with
     *  Activity.requestPermissions(String[], int)
     * An app must have ACCESS_COARSE_LOCATION permission in order to get
results
     * AND An App targeting Android Q or later must have
ACCESS_FINE_LOCATION permission in order to
     *  get results
     * OR a strong assertion that you will never derive the physical
location of the device. You can
     *  make this assertion by declaring
usesPermissionFlags="neverForLocation" on the relevant
     *  <uses-permission> manifest tag, but it may restrict the types of
Bluetooth devices you can
     *  interact with.*/
```

```
    public boolean scanDevice(final String address) {
        if (mBleScanner == null) {
            mBleScanner = mBluetoothAdapter.getBluetoothLeScanner();
        }
        mBleScanner.stopScan(mLeScanCallback);
        List<ScanFilter> filters = new ArrayList<>();
        ScanFilter filter = new ScanFilter.Builder()
                .setDeviceAddress(address)
                .build();
        filters.add(filter);
        ScanSettings scanSettings = new ScanSettings.Builder()
                .setScanMode(ScanSettings.SCAN_MODE_BALANCED)
                .setCallbackType(ScanSettings.CALLBACK_TYPE_FIRST_MATCH)
                .setMatchMode(ScanSettings.MATCH_MODE_AGGRESSIVE)
                .setNumOfMatches(ScanSettings.MATCH_NUM_ONE_ADVERTISEMENT)
                .setReportDelay(0L)
                .build();
        Log.d("ble-debug", "Scan started");
        mBleScanner.startScan(filters, scanSettings, mLeScanCallback);
        return true;
    }

    /************************************  STOP SCANNING
*************************************/
    public void stopScanning() {
        if (mBleScanner == null) {
            Log.d("ble-debug", "Scan object not initialized");
        } else {
            Log.d("ble-debug", "Scan stopped from
BluetoothLeService:stopScan");
            mBleScanner.stopScan(mLeScanCallback);
        }
    }


    /************************************  connectDevice
************************************
     * Connects to the GATT server hosted on the Bluetooth LE device.
     * For apps targeting Build.VERSION_CODES#S or or higher, this requires
the
     * Manifest.permission#BLUETOOTH_CONNECT permission
     *
     * @param address The device address of the destination device.
     * @return Return true if the connection is initiated successfully. The
connection result
     * is reported asynchronously through the
     * {@code
BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGa
tt, int, int)}
     * callback.
     */
    public boolean connectDevice(final String address) {
        deviceScanned = false;
        if (mBluetoothAdapter == null || address == null) {
            Log.w(TAG, "BluetoothAdapter not initialized or unspecified
address.");
            return false;
        }
        // Previously connected device, try to reconnect
        if (mBluetoothDeviceAddress != null &&
address.equals(mBluetoothDeviceAddress)
```

```
                && mBluetoothGatt != null) {
            deviceScanned = true;
            Log.d(TAG, "Trying to use an existing mBluetoothGatt for
connection.");
            if (mBluetoothGatt.connect()) {
                mConnectionState = STATE_CONNECTING;
                return true;
            } else {
                return false;
            }
        }
        // New device, scan for it in order to connect
        return scanDevice(address);
    }

    /************************************  disconnectGatt
************************************
     * Disconnects an existing connection or cancel a pending connection.
The disconnection result
     * is reported asynchronously through the
     * {@code
BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGa
tt, int, int)}
     * callback.
     */
    public void disconnectGatt() {
        // This variable is set here because disconnectGatt is only called
when the service is
        // stopped
        stoppingService = true;
        Toast.makeText(getApplicationContext(), "Disconnecting...",
Toast.LENGTH_SHORT).show();
        if (mBluetoothAdapter == null || mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized. No need to
disconnect");
            return;
        }
        mBluetoothGatt.disconnect();
    }

    /*************************************  closeGatt
************************************
     * After using a given BLE device, the app must call this method to
ensure resources are
     * released properly.
     */
    public void closeGatt() {
        if (mBluetoothGatt == null) {
            Log.d(TAG, "BluetoothGatt already closed (== null)");
            return;
        }
        mBluetoothGatt.close();
        mBluetoothGatt = null;
        Log.d(TAG, "BluetoothGatt successfully closed (== null)");
    }

    /*********************************  GET CHARACTERISTIC
*********************************/
    public BluetoothGattCharacteristic getCharacteristic(UUID uuid) {
        if (mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothGatt not initialized (not connected)");
```

```
            // TODO maybe better a try-catch?
            return null;
        }
        BluetoothGattService service =
mBluetoothGatt.getService(serviceUUID);
        return service.getCharacteristic(uuid);
    }

    /*********************************  SET CONNECTION PRIORITY
*********************************/
    public void setConnectionPriority(int priority) {
        if (mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothGatt not initialized (not connected).
Couldn't set connection priority");
        } else {
            mBluetoothGatt.requestConnectionPriority(priority);
        }
    }

    /*****************************  setCharacteristicsNotification
*****************************
     * Enables or disables notification on given characteristics.
     *
     * @param arrayUuids    Characteristics to act on.
     * @param enabled       If true, enable notification. False otherwise.
     */
    public void setCharacteristicsNotification(ArrayList<UUID> arrayUuids,
boolean enabled) {
        if (mBluetoothAdapter == null || mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized. Can't set
characteristics notifications");
            return;
        }
        // Initialize global variables
        if (numNotifications == 0) {
            numNotifications = arrayUuids.size();
            numUuids = numNotifications;
            arrayCharacteristicsUuids = arrayUuids;
            notificationEnabled = enabled;
        }
        // Extract current characteristic
        BluetoothGattCharacteristic characteristic =
                getCharacteristic(arrayUuids.get(numUuids -
numNotifications));
        // Change notification value
        mBluetoothGatt.setCharacteristicNotification(characteristic,
enabled);
        BluetoothGattDescriptor descriptor =
characteristic.getDescriptor(CCCD_UUID);
        if (enabled) {

descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
        } else {

descriptor.setValue(BluetoothGattDescriptor.DISABLE_NOTIFICATION_VALUE);
        }
        mBluetoothGatt.writeDescriptor(descriptor);
        Log.w(TAG, "characteristic notification changed to: " + enabled);
    }
```

```
}
```

CrutchApplication.java

```
package com.visens.crutch_v2;

/* ////////////////////////////////////   IMPORT LIBRARIES
////////////////////////////////////// */

import android.app.Application;
import android.content.Intent;
import android.util.Log;

import java.util.ArrayList;


/************************************  CrutchApplication
************************************
 * Extension of Application class to store global variables and constants
that need to exist
 * through the whole lifetime of the app */
public class CrutchApplication extends Application {

    /* ////////////////////////////////////   VARIABLES
////////////////////////////////////// */
    // Other variables
    public static boolean appInitialized = false;    // for initializing
bluetooth objects just once basically
    public static String from = null;                // for previous
activity identification
    // "MAC" table related variables
    // NOTE: although MAC address variable is only used in MainActivity, it
should be initialized
    //  only once per app initialization because once a device is connected
its value must be the
    //  same; and the device only disconnects when the app is closed
    // NOTE 2: that's exactly what mBluetoothDeviceAddress from
BluetoothLeService does, so that one
    //  is used
    // "user" table related variables
    public static int rowIDUserTable = -1;
    public static String userId = null;
    public static String iteration = null;
    public static String time = null;
    public static String date = null;
    public static int numSamples = -1;
    public static int numErrors = -1;
    public static int numMissing = -1;
    // "data" table related variables
    // NOTE: due to RAM limitations data values are no longer read all at
once so the ArrayLists of
    // data values are made local to VisualizeDataActivity

    // TODO for counting execution times
    public static ArrayList<Long> dataProcessingDurations;
    public static ArrayList<Long> dataSavingDurations;
    public static ArrayList<Long> screenUpdateDurations;
    public static ArrayList<Long> dataAcquisitionAbsoluteStartTimes; // =
new ArrayList<>();  // used in gattCallback
```

```
    public static ArrayList<Long> characteristicArrivalTimes;
    public static long inicioRecepcionDatos = 0L;


    /* ////////////////////////////////////   METHODS
//////////////////////////////////////// */

    /************************************  ON CREATE
*************************************
     * Starts Bluetooth LE Service so it runs through the whole app
lifetime and thus prevents its
     * variables from resetting when an Activity unbinds from the Service
*/
    @Override
    public void onCreate() {
        super.onCreate();
        Log.d("ble-debug", "***************  App started
***************");
        // For highest versions of Android, an app can't start a Service if
it's in the background.
        // To prevent crashes a try-catch statement is implemented
        try {
            startService(new Intent(this, BluetoothLeService.class));
            Log.d("ble-debug", "BLE Service started from
CrutchApplication");
        } catch (Exception exception) {
            Log.d("ble-debug", "Runtime Exception: couldn't start
Service");
            Log.e(getPackageName(), "", exception);
        }
    }

    // IMPORTANT: onTerminate method is for use in emulated process
environments. It will never be
    // called on a production Android device, where processes are removed
by simply killing them;
    // no user code (including this callback) is executed when doing so.
    //
https://developer.android.com/reference/android/app/Application#onTerminate
()
    // !!! This implies that the Bluetooth LE Service must stop itself
calling stopSelf() on
    // onTaskRemoved method !!!


}  // </ public class CrutchApplication extends Application >
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.visens.crutch_v2">
    <!-- ******* Required hardware for app to work properly: Bluetooth Low
Energy (BLE) ******** -->
    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="true" />
    <!-- ***************** Required permission concerning Bluetooth
activities ***************** -->
```

```
    <!-- BLUETOOTH: (Android 11 or lower) Allows applications to connect to
paired bluetooth devices -->
    <!-- BLUETOOTH_ADMIN: (Android 11 or lower) Allows applications to
discover and pair bluetooth devices -->
    <!-- BLUETOOTH_SCAN: Required to be able to discover and pair nearby
Bluetooth devices -->
    <!-- BLUETOOTH_CONNECT: Required to be able to connect to paired
Bluetooth devices -->
    <!-- ACCESS_COARSE_LOCATION: (Android 9 or lower) -->
    <!-- ACCESS_FINE_LOCATION: (Android 9 to 11; 12 or higher too or strong
assertion that app won't
     derive physical location) -->
    <!-- TODO [future] permissions only for needed API versions not sure if
correct -->
    <uses-permission
        android:name="android.permission.BLUETOOTH"
        android:maxSdkVersion="30" />
    <uses-permission
        android:name="android.permission.BLUETOOTH_ADMIN"
        android:maxSdkVersion="30" />
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION"
        android:maxSdkVersion="28" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
    <!-- ************** Required permission concerning File management
activities ************** -->
    <!-- WRITE_EXTERNAL_STORAGE: Allows an application to write to external
storage -->
    <!-- READ_EXTERNAL_STORAGE: Allows an application to read from external
storage -->
    <!-- MANAGE_EXTERNAL_STORAGE: Allows an application a broad access to
external storage in scoped storage -->
    <!-- TODO [future] in Android 11 and higher there is a better,
compulsory way of saving
          documents with the Storage Access Framework
          https://developer.android.com/training/data-
storage/shared/documents-files
          (more info) https://developer.android.com/training/data-storage
          but this way is messier as it requires intents and the result of
the intent (activity) in
          the new way implemented in MainActivity -->
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission
        android:name="android.permission.MANAGE_EXTERNAL_STORAGE"
        tools:ignore="ScopedStorage" />

    <!-- NOTE: Added name attribute in application to instantiate extended
application class CrutchApplication -->
    <application
        android:name=".CrutchApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/textAppNameInPhoneMenu"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

```
        android:theme="@style/AppTheme">
        <service android:name=".BluetoothLeService" />
        <activity android:name=".SaveDataActivity" />
        <activity android:name=".VisualizeDataActivity" />
        <activity android:name=".SelectSavedDataActivity" />
        <!-- Screen orientation in DataAcquisitionActivity locked to
portrait to prevent Bluetooth
        data looses because activities restart when screen orientation
changes -->
        <activity
            android:name=".DataAcquisitionActivity"
            android:screenOrientation="portrait"
            tools:ignore="LockedOrientationActivity" />
        <activity android:name=".ConfigureNewUserActivity" />
        <activity android:name=".ConfigurationActivity" />
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:alwaysRetainTaskState="true">
            <!-- TODO [future] check comment below -->
            <!-- NOTE: alwaysRetainTaskState seems important so the user
can have the app in the
            background acquiring data for a long period of time and be
confident that they would be
            able to save it.
            Documentation:
            Normally, the system clears a task (removes all activities from
the stack above the
            root activity) in certain situations when the user re-selects
that task from the home
            screen. Typically, this is done if the user hasn't visited the
task for a certain amount
            of time, such as 30 minutes.
            https://developer.android.com/guide/topics/manifest/activity-
element#always -->
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```