Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO DE FIN DE MÁSTER

MONITORIZACIÓN DE VARIABLES FISIOLÓGICAS PARA PERSONAS CON MOVILIDAD REDUCIDA

Estudiante	Uriarte Rodríguez, Ane
Director/a	Cabanes Axpe, Itziar
Departamento	Ingeniería de Sistemas y Automática
Curso académico	2021/2022

RESUMEN

En la sociedad actual se está incrementando el número de personas con movilidad reducida, bien agravada por el envejecimiento, o bien por el sedentarismo y las enfermedades neurodegenerativas. Esta situación revela un empeoramiento de la calidad de vida de las personas, lo que está suponiendo una saturación en el Sistema Sanitario Público y un incremento del gasto económico en la sanidad.

En este contexto, este Trabajo Fin de Máster recoge la descripción de una solución software para la monitorización de personas en sedestación, que permitirá monitorizar a aquellas personas que sufren movilidad reducida y facilitará en un futuro la detección de estados anómalos en la salud de las personas.

A lo largo del trabajo se ha hecho la selección de los dispositivos más adecuados donde implantar la solución. Una vez elegidos los dispositivos, se ha pasado a programar tanto la interfaz como el algoritmo que se incorpora en la solución.

Finalmente, tras desarrollar la solución en un dispositivo móvil Android, se ha comprobado su correcto funcionamiento a través de diferentes ensayos.

Palabras clave: Envejecimiento, Movilidad Reducida y Variables Fisiológicas.

LABURPENA

Gaur egungo gizartean, mugikortasun murriztua duten pertsonen kopurua handitzen ari da, zahartzearen eta bizimodu sedentaroaren ondorioz, baita gaixotasun neurodegeneratiboak direla eta. Egoera hau, pertsonen bizi kalitatea txarrera joatea eragin du, osasun publikoa leporaino utziz eta gastu ekonomiko ikaragarria suposatuz.

Egoera honetan ezarriz, Master Amaierako Lan honek eserita dauden pertsonentzako aldagai fisiologikoen monitorizaziorako soluzio software bat proposatzen du. Lan honek, etorkizun batean persona hauen osasunean edozein anomalia detektatzea ahalbidetu eta erraztuko du.

Lan honen garapenean hobeto egokitzen diren gailuen bilaketa bat egin da, gerora hoberen egokitzen diren gailuen aukeraketa egiteko eta bertan inplementzatzeko soluzio berria. Gailuak behin aukeratuta, soluzioaren algoritmoa programatzera pasatu da eta bere interfasearen diseinua egitera.

Bukatzeko, behin aukeraketa eginda eta inplementazioa eginda, entsegu batzuk egin dira ziurtatzeko diseinatutako lana ondo dabilela.

Hitz gakoak: Zaharkitzea, Mugimendu Murriztua eta Aldagai Fisiologikoak.

ABSTRACT

In today's society, the number of people with reduced mobility is increasing, either aggravated by aging, sedentary lifestyles or neurodegenerative diseases. This is provoking deterioration in the quality of life of the people, which lead into a saturation of the Public Health System and the increase of the economic cost in the health area.

Situated in this context, this Master's thesis gathers the description of a software solution for monitoring physiological variables in seated people, which will allow monitoring people with reduced mobility and in a near future, to detect anomalies on the health status.

During the execution of this thesis, a search has been made about the multiple devices exists in the market and to choose the best one who adapts to the purpose of the thesis. Once the devices have been selected, the solution has been programmed and designed the interface.

Finally, after having the solution done, its correct functioning has been checked carrying out different tests.

Key words: Ageing, Reduced Mobility and Physiological Variables.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Los casos de población con movilidad reducida no han hecho más que aumentar en los últimos años. Esto se debe a que la esperanza de vida ha ido creciendo, aumentando así el número de personas con una edad elevada. Estas personas son las que más sufren de afecciones relacionadas con el movimiento, que junto con el sedentarismo que llevan en sus vidas, muchos de ellos requieren de silla de ruedas en su día a día. Asimismo, la inactividad física acaba perjudicando la calidad de vida de estas personas, aumentando el riesgo de padecer enfermedades crónicas. Esto supone que se tengan que hacer más revisiones periódicas y por consiguiente, más visitas y rehabilitaciones a los centros de salud, causando un gran gasto económico al sector sanitario.

En vista de esta situación, la tendencia a la telemedicina está aumentando, con la clara misión de que se pueda monitorizar remotamente a los pacientes y no tengan la necesidad de acudir a las consultas, que se desarrollen detectores automáticos de anomalías que procesen los datos capturados de manera que pueda llegar el aviso al sanitario para así solventar la situación lo antes posible.

En este Trabajo de Fin de Máster, comúnmente llamado TFM, se ha desarrollado una solución software para monitorizar las variables fisiológicas remotamente para personas en sedestación y de la manera menos invasiva posible. Esta memoria recoge todos los apartados que se han desarrollado a lo largo de la ejecución del trabajo.

En primer lugar, se explica el contexto del trabajo donde se cuenta el origen de este trabajo, el grupo de investigación del cual forma parte y dentro de ese grupo los objetivos especificados para este trabajo. Una vez definidos los objetivos, se define el alcance del TFM y las tareas a ejecutar.

A continuación, se ha realizado una búsqueda de las aportaciones hechas hasta el momento de la monitorización de las constantes vitales. Se han estudiado los diferentes dispositivos que existen para hacer el seguimiento de la salud de las personas. Posteriormente, se presenta un análisis de alternativas de los diferentes dispositivos encontrados para solventar el problema de la monitorización en sedestación, con el fin de seleccionar el mejor dispositivo que dé respuesta a tal problema y llevar a cabo la solución propuesta.

Tras la descripción de la solución software, se presenta la implementación llevada a cabo mediante el software Android Studio y su validación mediante diferentes ensayos experimentales, ratificando así la validez de la solución.

Finalmente, se presentan las conclusiones extraídas del TFM.

Todo el código desarrollado, para la implementación de la solución en el software Android Studio, se puede encontrar en la sección ANEXO II.

2. CONTEXTO

Este trabajo fin de máster forma parte de un proyecto nacional que se ha desarrollado dentro del **Grupo de Investigación Sensorización Virtual para Bioingeniería** (Virtual Sensorization Research Group for Bioengineering, ViSens) de la Universidad del País Vasco. Este grupo de investigación está formado por un grupo de investigadores del área de la Ingeniería y otro del ámbito de la Salud y las Ciencias de la Actividad Física y el Deporte.

El grupo Visens centra su esfuerzo en incrementar el grado de tecnologización en ámbitos sanitarios como son el Envejecimiento, la Fisioterapia y la Fisiología, buscando dar respuesta a problemáticas reales y generar desarrollos que tengan un impacto directo en la sociedad. La temática de investigación del grupo se enfoca por tanto en la Sensorización Virtual en el área de la Bioingeniería, siendo su objetivo principal el diseño y desarrollo de sensores basados en el procesamiento software con el fin de estimar variables de interés que permiten analizar el estado de un paciente, clasificar y/o diagnosticar, a través de la medida de variables básicas.

Actualmente el grupo está trabajando en un enfoque orientado a la mejora de la calidad de vida de personas que sufren enfermedades neurodegenerativas como pueden ser la esclerosis múltiple o la atrofia muscular, entre otros. En aquella fase en la que los pacientes todavía tienen la capacidad de moverse por su cuenta, a pesar de la dificultad, el grupo ha desarrollado un prototipo de muleta sensorizada capaz de monitorizar los patrones de marcha, cuantificar el uso que está realizando el usuario con la muleta (si la utiliza como apoyo o equilibrio) o clasificar las actividad de la marcha de la vida diaria (si anda lento, rápido, sube o baja escaleras, está quieto, se ha caído el usuario y/o la muleta). Concretamente en este proyecto también se presenta como objetivo detectar anomalías durante el desplazamiento en personas que requieren dispositivos de asistencia para la marcha, como son bastones y muletas.

Para gente que tiene más limitado el movimiento motriz, se está llevando a cabo un proyecto de Sensorización Virtual, cuyo principal objetivo es monitorizar la sedestación, tanto en silla fija como en silla de ruedas y detectar las anomalías en posturas de sedestación.

Para ello, se plantean tres retos (ver figura 1): el primero de ellos está enfocado en la monitorización postural del usuario, el segundo en la monitorización de los movimientos de la

silla de ruedas y las vibraciones que esta transmite a los usuarios debido a la irregularidad de la superficie y el tercer reto se centra en la monitorización de las variables fisiológicas del usuario de la silla de ruedas.

Figura 1. Retos en desarrollo del grupo Visens.

La monitorización postural tiene como objetivo identificar posturas nocivas para el usuario. Para ello, se ha desarrollado un cojín (denominado i-KuXin) que cuenta con sensores de presión distribuidos de forma estratégica en el asiento y respaldo, formando un mallado de presión, que es capaz de monitorizar la distribución de la fuerza que ejerce el cuerpo humano en estado de sedestación.

La monitorización de vibraciones y movimientos de la silla de ruedas tiene como objetivo diseñar e implementar un sistema multi-sensorial de adquisición de datos relativos a posiciones, velocidades y aceleraciones en ejes triaxiales (3D) e inferir en cómo afectan a los usuarios de la silla en cuanto a fatiga y otros indicadores sanitarios.

La monitorización de variables fisiológicas tiene como finalidad diseñar un sistema de medición y recogida de datos de los signos vitales para la detección de anomalías en el estado de salud.

Este trabajo fin de máster (TFM) se sitúa dentro del proyecto de Sensorización Virtual de personas con movilidad reducida, usuarias de una silla de ruedas de forma continuada. Concretamente, este TFM pretende dar respuesta al reto de la monitorización de variables

fisiológicas para este tipo de personas que, una vez finalizado, complementará a los otros dos retos.

A modo de conclusión, el trabajo de fin de máster consistirá en **diseñar e implementar un sistema de monitorización de variables fisiológicas remoto y no invasivo para ser utilizado en usuarios de sillas de ruedas**. Este sistema será capaz de medir en todo momento las variables fisiológicas y recoger esa información para su posterior procesamiento, con el fin de detectar posibles anomalías en personas con limitaciones severas de movimiento.

Este trabajo se encuentra directamente alineado con el tercer Objetivo de Desarrollo Sostenible “Salud y Bienestar”, establecido en la agenda 2030 a nivel mundial por Naciones Unidas; así como con la “EHUagenda 2030 por el desarrollo sostenible” de la UPV/EHU.

3. OBJETIVOS Y ALCANCE DEL TRABAJO

El principal objetivo de este TFM es el desarrollo de una solución software no invasiva para realizar una monitorización de variables fisiológicas en personas con movilidad reducida.

Los objetivos específicos que permiten el correcto desarrollo de este trabajo son los siguientes:

1. Búsqueda de la tecnología que existe actualmente en el mercado que permite la monitorización de variables fisiológicas y los sensores más utilizados para ello (como son los relojes inteligentes y/o móviles). Se hará uso de la suscripción que tiene la UPV/EHU a bases de datos bibliográficas, como Science Direct, IEEExplore, otras páginas de libre acceso de donde obtener información como Google Scholar o Scopus y todo documento relacionado.
2. Creación de una aplicación Android Wear OS, para un dispositivo sensorial, capaz de realizar la lectura de parámetros como la frecuencia cardiaca y su posterior envío al dispositivo de captura de datos.
3. Establecer una comunicación Bluetooth entre el dispositivo sensorial y dispositivo para la captura de datos (como móvil, Tablet…), generando y extrayendo en el formato adecuado la información.
4. Creación de una aplicación móvil que permita recibir las variables recogidas por el reloj, visualizarlas en tiempo real en la interfaz y guardarlas en una base de datos para su posterior análisis.
5. Comprobar y validar el correcto funcionamiento de las aplicaciones a desarrollar mediante diferentes ensayos experimentales y obtener conclusiones del conjunto.

En resumen, el alcance de este trabajo es el de proporcionar al grupo de investigación Sensorización Virtual para Bioingeniería un sistema de monitorización inalámbrico que esté completamente validado. De esta forma, dicho desarrollo se incorporará al sistema multi-sensorial completo en la silla de ruedas motorizada, siendo su finalidad detectar posibles anomalías en personas con limitaciones severas de movimiento y transmitir dicha información cuantificada a los sanitarios.

En la Figura 2 se muestra la visión general del objetivo de este TFM del ámbito de ingeniería con una clara aplicación al ámbito sanitario.

Figura 2. Planteamiento del TFM: monitorización y procesamiento de datos.

4. BENEFICIOS QUE APORTA EL TRABAJO

En este apartado se exponen los beneficios que aporta la realización de este trabajo fin de máster. Dentro de los beneficios, se pueden categorizar en tres grupos: beneficios tecnológicos, económicos y sociales.

4.1 Beneficios científico-técnicos

Este TFM, al formar parte de un proyecto de investigación del Grupo de Sensorización Virtual para Bioingeniería, permite avanzar en el desarrollo del proyecto nacional que se centra en detectar anomalías en personas en sedestación, que por su deterioro (físico, cognitivo o traumatológico) utilizan silla de ruedas. Siendo el objetivo principal de este trabajo la monitorización de variables fisiológicas, se va a crear una base de datos de variables fisiológicas para su posterior análisis y desarrollo de indicadores mediante técnicas de Inteligencia Artificial. Esa base de datos podrá ser utilizada tanto para este proyecto como en proyectos que se desarrollen a futuro y que requieran de estos datos recogidos.

Por otro lado, el desarrollo tecnológico de este trabajo permitirá recoger información cuantificada del estado de los pacientes, procesarla y enviarla al fisioterapeuta o personal sanitario para conseguir adecuar la terapia a su estado actual, logrando mejores resultados.

Finalmente mencionar que se encuentra alineado totalmente con el tercer Objetivo de Desarrollo Sostenible, Salud y Bienestar, establecido en la agenda 2030 por Naciones Unidas y también por el Gobierno Vasco y la UPV/EHU bajo el reto Salud Personalizada.

4.2. Beneficios económicos

Este sistema de monitorización inalámbrico y remoto, va a permitir que los pacientes no tengan la necesidad de acudir a las consultas, descongestionando los centros de salud y hospitalarios, reduciendo así el coste económico que supone atender a todos los pacientes y la realización de menos pruebas rutinarias.

Además, al tratarse de una solución software Low Cost, la aplicación será accesible para todas las personas que encuentren dificultades para presentarse en las instalaciones sanitarias y así podrán beneficiarse de las comodidades que esta aplicación aporta.

No solo se reducirá el coste que suponen las consultas, sino también el transporte que las personas con movilidad reducida requieren para acceder a los centros de salud y con ello lo que supone ese coste. En España, el gasto sanitario público en el año 2021 ascendió a 80 millones de euros, lo que supuso un gasto anual por habitante de 1.700€ [1].

4.3. Beneficios sanitarios y sociales

Hasta ahora todas las monitorizaciones y pruebas que valoran el estado fisiológico de las personas se realizan de manera puntual en las consultas. Este sistema de monitorización continuo y remoto, va a permitir hacer un seguimiento en todo momento del estado del paciente, ya sea de día y de noche. Permitirá ver cómo evoluciona el estado del paciente y poder asociar periodos del año, conductas, lugares o acciones que afectan al estado de salud del paciente. En los casos en que se identifique la causa que provoca el empeoramiento de la salud de las personas, se podrán tomar medidas más adecuadas para mejorar su día a día, evitando las situaciones que provocan el deterioro de la salud. Un claro ejemplo es el subir las escaleras, donde personas de avanzada edad encuentran grandes dificultades debido a su edad, su condición física y a patologías cardíacas.

De la misma manera, al realizar monitorizaciones de manera continua se detectarán en tiempo real anomalías que pueden ser causadas por alguna enfermedad. Pudiendo así, aumentar las posibilidades de éxito de los tratamientos y reducir las complicaciones de la enfermedad como las secuelas que los tratamientos pueden causar.

Por lo tanto, el beneficio social que ofrece el diseño de este sistema es el de mejorar la vida de las personas en cuanto a calidad de vida y aumentar la esperanza de vida.

5. ESTADO DEL ARTE

En este apartado se va a explicar la situación actual que supone el problema de la movilidad reducida (MR) en la sociedad, las causas que lo agravan y las repercusiones que esto provoca.

En segundo lugar, se presentará el estudio de la técnica o los antecedentes bibliográficos relativos a la monitorización de las variables fisiológicas. Posteriormente, se presentará el trabajo de búsqueda de los diferentes sensores que existen para la monitorización de los signos vitales y los trabajos relacionados existentes, finalizando con una breve conclusión.

5.1. LA MOVILIDAD REDUCIDA

La movilidad se entiende como el conjunto de desplazamientos que realiza una persona en un entorno físico para poder desplazarse. Este concepto está relacionado con la movilidad física, que incluye movimientos motores del cuerpo humano y la coordinación que se asocia a esos movimientos. Existen múltiples razones por las que esta movilidad puede verse afectada, deteriorada o incluso llegar a la inmovilidad.

A pesar de que cualquier persona puede sufrir una alteración en la movilidad motriz, estas alteraciones son más comunes en personas con enfermedades crónicas o agudas, con lesiones traumáticas o dolor crónico. Las principales enfermedades que afectan directamente a la movilidad son las enfermedades del sistema nervioso o las musculares. Entre ellas se puede destacar la parálisis cerebral, la esclerosis múltiple y la enfermedad de Parkinson [2].

Otras de las causas habituales de la perdida de movilidad son las enfermedades que contribuyen al agotamiento físico, como son la insuficiencia cardíaca y la enfermedad pulmonar obstructiva crónica [1].

La afección de estas enfermedades impide a las personas la posibilidad de caminar, erguirse o desplazarse. Es por ello que para solventar estas circunstancias los pacientes necesiten de dispositivos asistentes activos o pasivos, como la silla de ruedas, bastones y muletas, respectivamente.

Debido a la mejora de la sanidad pública, la mejora de las condiciones de vida y los avances de la tecnología, son muchas las personas que viven con alguna afección la cual les impide tener una movilidad física total. Según el Observatorio Estatal de la Discapacidad (OED), el número

total de declarantes con discapacidad en el ejercicio en 2019 fue de más de dos millones de personas [3].

Adicionalmente, el crecimiento de la población adulta está causando un aumento en el número de personas con movilidad reducida (MR), ya que la edad es un factor determinante en la aparición de este fenómeno. En la Figura 3 se aprecia esta tendencia, siendo la franja de edad más crítica o la que mayor población con discapacidad presenta de 75 a 84 años. Cabe resaltar cómo, a edades tempranas (a partir de 6 años), en la pirámide se visualizan ya los primeros casos de aparición.

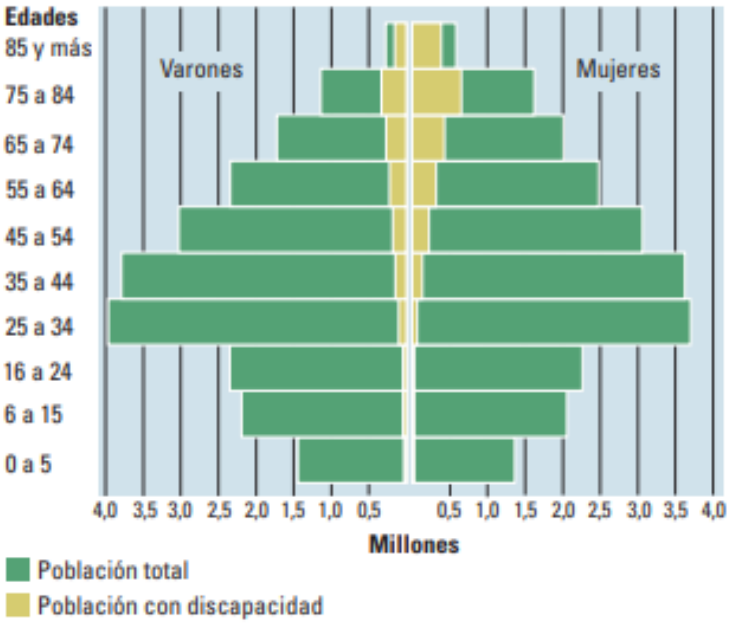

Figura 3. Pirámide de población [3].

Varios estudios han demostrado que las personas mayores de 65 años presentan dificultad de movimiento en un 18% y mayores de 75 años en un 50% [4]. Con el envejecimiento, el deterioro de la marcha no hace más que aumentar de manera progresiva y definitiva.

Las alteraciones de la marcha y, por tanto de la movilidad, hacen que las personas sientan miedo a las caídas, por lo que les lleva a optar por una vida más sedentaria. A medida que la edad aumenta, se puede apreciar más este fenómeno en la Figura 4, donde las personas entre 75 y 84 años tienen un porcentaje de presentar sedentarismo por encima del 41%, y aquellos

cuya edad es superior a 85 años, alcanzan el 60% y esto no hace más que agravar la situación existente.

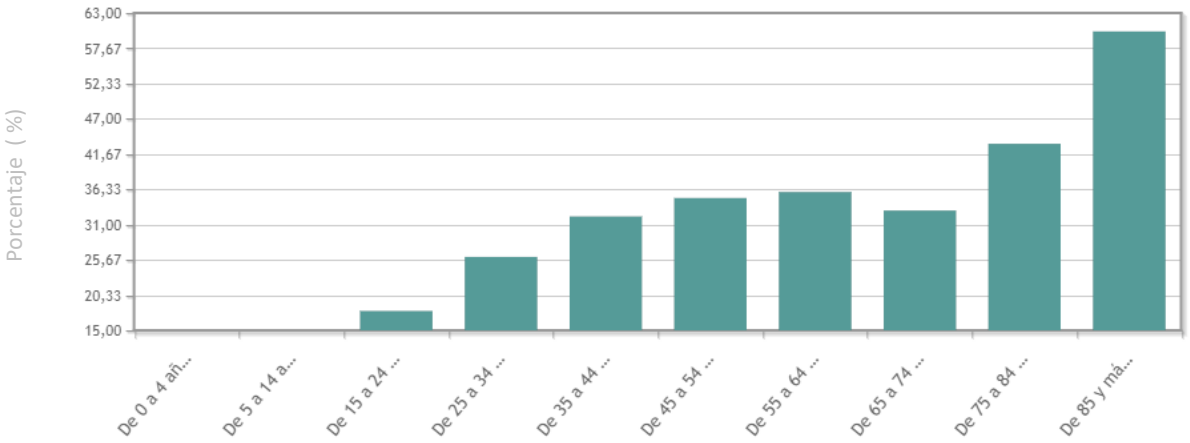

Figura 4. Sedentarismo según edades, año 2020 [5].

El imparable ascenso de los casos de MR, junto con el envejecimiento y la inactividad física está teniendo graves consecuencias en la calidad de vida de las personas y en el Sistema Sanitario Público (SSP).

En cuanto a la calidad de vida de las personas, estudios recientes han demostrado que la inactividad física y los comportamientos sedentarios aumentan el riesgo de diabetes, obesidad, enfermedades cardiovasculares y, en general, la mortalidad [6], [7], [8].

En lo que a lo sanitario respecta, las personas con MR requieren de revisiones más periódicas, debido al riesgo que sufren a padecer enfermedades. Además, requieren de medios de transporte adicionales y adaptados para aproximarse a las instalaciones sanitarias, como son las ambulancias o autobuses especializados, además de sesiones de ejercicio para fortalecer su estado físico [9]. Todo ello supone un gasto extra en la sanidad pública. De hecho, considerando únicamente la inactividad física, esta supuso un coste a los sistemas sanitarios de 53.800 millones de dólares según un informe publicado en 2016 con datos de 142 países [7].

De igual manera, la sanidad cada vez está más saturada, incrementada también por las necesidades del colectivo MR, la existencia de un mayor riesgo de desarrollar una discapacidad

y por el aumento en general en la asistencia de la población [10]. Debida a la carga existente en la sanidad y, las dificultades que las personas con MR tienen para acudir a las consultas, los profesionales sanitarios apuestan por monitorizar los posibles cambios que puedan suceder en el estado de salud de sus pacientes, sin que estos tengan que acudir al centro de salud [11]. Así, la obtención de datos en tiempo real es vital para la detección de cambios anormales en las constantes vitales que pueden ser producidas por una enfermedad [12].

Como consecuencia de la necesidad de monitorización, se produce un aumento en la investigación y en el diseño de nuevos dispositivos capaces de realizar una monitorización remota, precisa y en tiempo real. Gracias a los avances que se han realizado en materia de hardware y en la mejora de redes de sensores inalámbricos, que junto con nuevas tecnologías inalámbricas de corto alcance, como es el caso de Bluetooth, permiten la posibilidad de integrar sensores inteligentes para diagnósticos remotos o tratamientos médicos [9]. Estos sensores juegan un papel muy importante a la hora de captar, valorar y poder procesar la información [11]. Una gran ventaja de esta aproximación es que no se necesita una intervención humana para su funcionamiento y permitiría aprovechar al máximo el rendimiento y la eficiencia del Sistema Sanitario. Sin embargo, todavía siguen existiendo barreras e incomodidades provocadas por la necesidad de la utilización de cables en ciertos dispositivos [11]. Por ello, hay una firme tendencia a diseñar dispositivos totalmente inalámbricos y no invasivos, de modo que no afecten en la vida diaria de los pacientes.

Con el concepto de salud inteligente, los usuarios son capaces de ver su estado de salud en cualquier lugar utilizando la tecnología de comunicación móvil y el dispositivo inteligente [13]. La tendencia se focaliza en hacer un seguimiento completo, es decir, no únicamente recoger el estado de una variable fisiológica, sino de un conjunto de todas a la vez para tener una mayor y mejor información, como puede ser la temperatura, la oxigenación en sangre, la frecuencia cardíaca y la postura corporal.

Un claro ejemplo de estos dispositivos son los sistemas de asistencia sanitaria móvil para personas con sillas de ruedas, los cuales permiten monitorizar en todo momento al individuo y, en caso de caída o detección de anomalía, el sistema manda una alerta [14].

Toda esta problemática de la movilidad reducida junto con las nuevas tecnologías está impulsando al sector científico-tecnológico a buscar soluciones para paliar esta situación. Por

todo ello, es de esperar un avance en paralelo de las nuevas técnicas de monitorización y nuevos sensores menos intrusivos, cada vez más imperceptibles para el usuario.

5.2. MONITORIZACIÓN DE VARIABLES FISIOLÓGICAS

Los signos vitales son un componente importante en el seguimiento de la evolución de las personas que sufren alguna afección, como puede ser el caso de la movilidad reducida, puesto que permiten detectar acontecimientos adversos en la salud e indican el estado funcional del individuo. Las principales variables fisiológicas son la frecuencia cardíaca, la presión arterial, la temperatura corporal y la saturación de oxígeno [15]–[17]. Es por ello que es necesario medir con precisión estos parámetros, ya que un cuerpo sano los mantiene dentro de unos rangos de correcto funcionamiento. Debido a que un gran número de enfermedades vienen acompañadas de un cambio significativo de estas variables, se recomienda monitorizar estos parámetros.

La frecuencia cardíaca mide el número de veces que el corazón late por minuto. Por regla general, se ha determinado que la frecuencia cardiaca de una persona debe estar entre 60 y 100 latidos por minuto mientras está en reposo. La medición del pulso proporciona una información importante acerca de la salud cardiovascular de un individuo o por el contrario determina si sufre alguna anomalía.

Hay algunos aspectos que pueden alterar la salud cardiovascular como puede ser el ejercicio físico. Al realizar un esfuerzo físico, el corazón produce una respuesta llamada taquicardia en la que aumenta la frecuencia por encima de 100 latidos por minuto. Existen también otros casos en los que la frecuencia cardíaca aumenta, como son el estrés mental, el ejercicio isométrico y la postura. Es importante conocer en todo momento el valor de la frecuencia cardíaca ya que los adultos en sedestación tienen 3 veces más probabilidades de sufrir alguna enfermedad cardíaca que los adultos sin disfunciones motoras [18].

La saturación en sangre indica el nivel de oxígeno en la sangre. Generalmente se suele realizar esta medición para conocer el estado de disnea o dificultad para respirar del paciente y para determinar si padece alguna enfermedad pulmonar. El nivel normal de oxígeno en sangre oscila entre el 95 y 100%, considerándose bajos los valores inferiores al 90% [19]. Este porcentaje indica cuanto oxígeno transporta la sangre en relación al máximo que es capaz de

transportar [20]. Los niveles de saturación pueden ser un poco más bajos y considerarse aceptables en personas que sufren una enfermedad pulmonar obstructiva crónica (EPOC) [21].

La temperatura corporal es una medida de la capacidad que tiene el organismo de generar y eliminar calor. El propio cuerpo es capaz de regular su temperatura de manera muy eficiente y mantenerla dentro de unos límites seguros, incluso cuando la temperatura exterior cambia mucho. Esta puede variar en función de diferentes factores tales como la edad, la hora del día, el clima, la ansiedad, las enfermedades y la actividad física.

La monitorización de la temperatura corporal es de gran alcance y puede tener implicaciones importantes para atletas y personas que padecen alguna enfermedad. La evaluación continua de la temperatura proporciona unos datos fundamentales acerca del funcionamiento del cuerpo y la condición o el estado de salud [22].

En la tabla 1 se resumen las principales variables fisiológicas en edad adulta (hasta 64 años), en edad anciana (a partir de 65 años), con enfermedades respiratorias, insuficiencias cardíacas o similares.

EDAD O AFECCIÓN	TEMPERATURA	FRECUENCIA CARDÍACA	OXÍGENO EN SANGRE
ADULTOS (< 65 años)	36.5°C	60-100 rpm	95-100 %
ENVEJECIMIENTO (> 65 años)	33.5-35.5°C	60-100 rpm	95-100%
OBESIDAD	36 °C o inferior	80-120 rpm	95-100%
ENFERMEDADES RESPIRATORIAS	36 °C o inferior	90-130 rpm	<90%
INSUFICIENCIA CARDÍACA	36 °C o inferior	Menor que 60 rpm	<90%
ESTRÉS	36°C o inferior	>100 rpm	95-100%
TABAQUISMO	36°C o inferior	Superior o igual a 85 rpm	90-95%

Tabla 1. Cifras normales de los signos vitales según la edad [15] [23].

A continuación, se analizarán los diferentes sensores que existen en el mercado para la medición de las variables fisiológicas mencionadas.

5.2.1. SENSORES PARA LA MEDICIÓN DE LA FRECUENCIA CARDÍACA

Para obtener la frecuencia cardíaca hay tres métodos tradicionales: 1) extraer la señal del electrocardiograma (ECG), 2) calcular la frecuencia del pulso a partir de la fluctuación medida por el sensor de presión al medir la presión arterial y 3) el método de volumen fotoeléctrico. Los dos primeros métodos limitan la actividad del paciente si se utiliza durante mucho tiempo, aumentando el malestar físico del paciente [13].

En trabajos de otros autores, [24], se ha utilizado un sistema de monitorización SEEQ, el cual consiste en un dispositivo externo adhesivo, sin cables, que se adhiere al pecho del paciente (ver Figura 5). Este dispositivo utiliza un modem inalámbrico portátil para la transmisión vía satélite [24] .

Figura 5. Dispositivo de monitorización inalámbrico satelital SEEQ [24].

Otro sistema para monitorizar la frecuencia cardíaca es mediante pulseras inteligentes. Estas pulseras inciden un haz de luz en la piel. Cuando el corazón bombea sangre, los vasos sanguíneos están llenos de sangre. La sangre tiende a absorber la luz verde y a reflejar la luz roja por lo que el corazón producirá diferentes colores cuando se contraiga y se relaje [13]. En estos casos, la pulsera registra la frecuencia cardíaca en función de la luz detectada.

Aunque está más dirigido a corredores, también podemos encontrar otro aparato para medir el ritmo cardíaco: el monitor cardíaco de pecho. Se trata de un dispositivo resistente al agua que se coloca en el pecho con la ayuda de una banda elástica. Entre sus características más relevantes se encuentra su peso ligero y su comodidad [25]. En el ámbito sanitario, es más utilizado el monitor cardíaco implantable. Este dispositivo se coloca debajo de la piel del pecho

y se utilizan para detectar latidos cardíacos irregulares (ver Figura 6). Este último es un sistema invasivo, ya que para implantarlo requiere de una breve intervención quirúrgica [26].

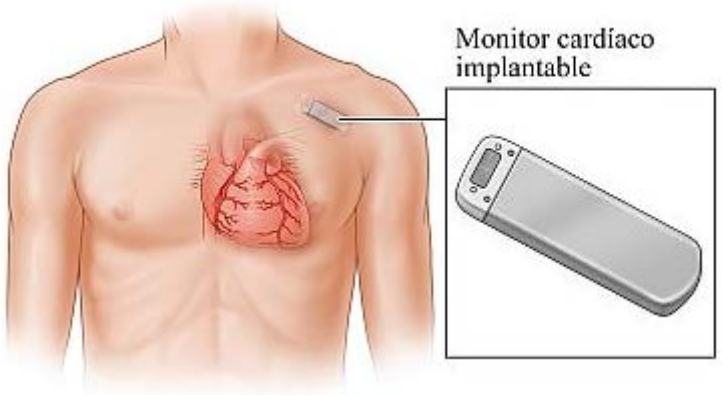

Figura 6. Monitor cardiaco implantable [26].

5.2.2. SENSORES PARA LA MEDICIÓN DEL OXÍGENO EN SANGRE

Para determinar la saturación de oxígeno (SpO2) se utiliza el procedimiento llamado oximetría de pulso. Este determina el porcentaje de oxígeno que hay en sangre a través de dispositivos que incorporan métodos fotoeléctricos.

El método fotoeléctrico consiste en la emisión de dos longitudes de onda diferentes: luz roja e infrarroja. Las ondas se transmiten por un emisor hasta un fotodetector a través de la zona donde se haya colocado el dispositivo. Como se puede apreciar en la Figura 7, de forma general, esta variable se suele medir en la yema del dedo de la mano ya que es una de las partes del cuerpo donde existe más microcirculación. Con el fin de obtener el porcentaje de saturación de oxígeno en sangre, se mide la absorbancia de cada longitud de onda que llega al fotodetector [27].

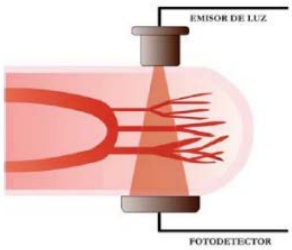

Figura 7. Efecto fotoeléctrico a través de un dedo [27].

Para una correcta detección es necesario que el dispositivo, llamado pulsioxímetro, se coloque en zonas del cuerpo relativamente translucidas donde el flujo sanguíneo sea abundante, como son los dedos de la mano o el pie, los lóbulos de las orejas o el puente de la nariz [27].

Figura 8. Pulsioxímetro para la medición del Sp02.

Los valores adecuados de saturación en sangre son los que están por encima del 95%. Cuando la saturación alcanza valores inferiores al 90% se considera de la oxigenación en sangre es deficiente, lo que puede indicar alguna insuficiencia cardíaca, enfermedades neurológicas, asma o neumonía.

5.2.3. SENSORES PARA LA MEDICIÓN DE LA TEMPERATURA CORPORAL

El control de la temperatura en personas con movilidad reducida es fundamental, ya que al no realizar la termo-regulación de manera efectiva, como podría ser caminando, tienen más riesgo a sufrir cambios bruscos de temperatura cuando las situaciones climatológicas son desfavorables [28].

Los lugares anatómicos más idóneos para la medición de la temperatura corporal son el esófago, la arteria pulmonar, nasofaringe o la vejiga [29]. Sin embargo, estos métodos de medición son invasivos por lo que solo se suelen realizar en unidades de cuidados intensivos.

En la práctica, los métodos no invasivos realizan las mediciones desde distintos lugares del cuerpo, recogiendo la temperatura axilar, rectal u oral.

En cuanto a los métodos no invasivos se pueden encontrar diferentes tipos de termómetros: de mercurio, de galistán, infrarrojos, de oídos y termómetros digitales (ver figura 9). Además de esos aparatos, se pueden encontrar equipos de cámaras termográficas que detectan cambios en la temperatura corporal.

Figura 9. Sistemas de medición corporal portátil.

Se han realizado estudios para la medición de la temperatura corporal de neonatos dentro de incubadoras mediante la técnica de la radiometría. Estas mediciones se basan en el principio de los cuerpos que emiten radiaciones electromagnéticas espontáneas de origen térmico [30]. Para coger estas medidas se ha usado un sensor de antena conectado a un radiómetro de microondas, que recoge la potencia de ruido térmico que es emitida por el cuerpo humano.

5.3. SITUACIÓN ACTUAL

Un 1% de la población mundial sufre algún problema de movilidad [31]. Este porcentaje sigue al alza debido al aumento del envejecimiento de la sociedad y a la vida sedentaria. Todo ello, está afectando muy seriamente en la calidad de vida de las personas, disminuyendo la capacidad de valerse por uno mismo, la independencia que uno puede tener para vivir y, por supuesto, la esperanza de vida.

El Sistema Sanitario Público también se está viendo afectado debido a estas casusas. Los recursos sanitarios están dejando de ser suficientes debida a la cantidad de personas con problemas de salud que requieren de asistencia sanitaria. La saturación en las consultas se está convirtiendo en algo común, a lo que ya a nadie le extraña, y el gasto sanitario no cesa de aumentar.

Con el fin de terminar con esta situación, empresas relacionadas con la tecnología y grupos de investigación de Universidades y Centros Tecnológicos, están enfocando todos sus conocimientos a desarrollar dispositivos de monitorización de signos vitales. La tendencia exige que estos sensores sean lo menos invasivos posibles. De esta forma, no afectarán a la vida de las personas y serán totalmente remotos, sin necesidad de ningún tipo de cableado. No obstante, en el caso de monitorizar signos vitales para personas en silla de ruedas, los dispositivos menos invasivos son los relojes inteligentes y las bandas de pecho. A la hora de monitorizar su estado fisiológico, como primera aproximación sólo se medirá una variable de las tres previamente mencionadas, la frecuencia cardíaca. La ventaja que tiene este signo vital es que se puede medir desde cualquier extremidad del cuerpo, sin apenas alterar el día a día de las personas, tanto en reposo como en movimiento. Además, es muy significativo a la hora de valorar el estado de salud de las personas.

En cuanto a la monitorización de la oxigenación, cabe mencionar que en este TFM no se ha promovido por las siguientes razones. Por un lado, se trata de una variable que se mantiene constante (ver Tabla 1) y no se verían cambios a no ser que se padezca alguna enfermedad pulmonar. Además los relojes inteligentes que miden la oxigenación están diseñados para deportistas y en el caso de personas en sedestación no sería muy útil ya que su movilidad es

reducida. Por otro lado, el otro dispositivo que existe para medir la oxigenación es el pulsioxímetro que, pese a ser un dispositivo no invasivo, crea incomodidad a la persona y limita el uso de las manos. Por último, hasta el momento no hay estudios que demuestren cambios o alteraciones en la medición de la saturación de oxígeno en personas en sedestación. Por tanto, en el desarrollo de este TFM, no se tendrá en cuenta pero sí se realizará una validación con la solución desarrollada, con objeto de demostrar esta hipótesis extraída de la revisión bibliográfica realizada.

Se descarta también la idea de medir la temperatura corporal en sedestación ya que las medidas pueden ser muy poco precisas. Para medir la temperatura corporal es necesario que se mida en el tronco del cuerpo, ya que esta temperatura se mantiene constante y, ante cualquier cambio, es esta temperatura la que determina el estado del individuo. Pero para ello, sería necesario que la persona llevase en todo momento una banda colocada en el pecho, lo cual supone una incomodidad [32]. Si se quisiese medir la temperatura vía un reloj o pulsera inteligente, está iría colocada en una extremidad y la temperatura de las extremidades varían con respecto a la temperatura del tronco. Por no mencionar la poca fiabilidad que tendrían esas mediciones debido a que la pulsera se encuentra en un lugar donde la sudoración, la exposición al sol y al agua son continúas y pueden alterar la medición de la temperatura.

De igual manera, la tecnología comercial existente no permite realizar mediciones de la presión arterial y por consiguiente no será posible realizar mediciones con un dispositivo comercial.

En definitiva, todos los esfuerzos de este TFM se están dirigidos a monitorizar la frecuencia cardíaca, debida a su facilidad de medición y a su efectividad en la detección de anomalías en el estado de salud de las personas en sedestación.

En la Figura 10 se muestra un resumen de los sensores más adecuados para medir las variables fisiológicas más relevantes de la manera menos invasiva posible, tras el estudio de la tecnología actual realizado en esta temática.

Figura 10. Resumen de los sensores de variables fisiológicas más adecuados para realizar una monitorización.

6. ANÁLISIS DE ALTERNATIVAS

En este apartado se ha hecho un análisis de los diferentes métodos y dispositivos, tanto Hardware como Software, que existen en la actualidad para realizar la conexión inalámbrica entre dos dispositivos digitales con el fin de poder realizar el envío de datos de parámetros obtenidos a través de la monitorización.

6.1. HARDWARE PARA LA MONITORIZACIÓN DE DATOS

Con el objetivo de dar la mejor respuesta a la monitorización de datos, se ha hecho una búsqueda de los dispositivos que se pueden encontrar en el mercado. Como se ha explicado en el estado del arte, es necesario que el dispositivo posea de ciertos sensores específicos para poder monitorizar las variables fisiológicas y que sea totalmente inalámbrico. Como condición a la hora de la búsqueda del dispositivo, se ha establecido que el dispositivo vaya ajustado en la muñeca de manera que no incomode a la hora de la monitorización. Tras el análisis de las alternativas, se elegirá la opción más adecuada que permita conseguir la funcionalidad definida en los objetivos.

Se han barajado dos opciones en cuanto al hardware de adquisición: por una parte, existe la posibilidad de desarrollar un dispositivo desde cero (incluyendo toda la sensórica y electrónica adecuada y de pequeño tamaño). La segunda opción, es escoger un dispositivo "wearable" que exista actualmente en el mercado y que cumpla con las especificaciones requeridas, como sería el caso de un reloj inteligente, más conocido como *smart watch*.

De las dos posibles propuestas se ha descartado la de crear un dispositivo debido a la dificultad, precio y plazos que supone. Por lo tanto, se va a proceder a hacer una búsqueda y un posterior análisis de los relojes inteligentes que más se pueden adecuar a las condiciones arriba mencionadas.

6.1.1. RELOJES INTELIGENTES

Existen muchas opciones en el mercado a la hora de escoger un reloj inteligente, por ello, se ha establecido un criterio de selección para elegir el dispositivo que mejor se adapte a las necesidades del proyecto.

Como se ha explicado en el estado del arte, es necesario que el dispositivo tenga los sensores adecuados para monitorizar los signos vitales, en concreto el sensor de frecuencia cardíaca y el sensor de oxigenación en sangre.

En cuanto al sistema operativo del dispositivo, se han descartado los dispositivos que utilizan el sistema operativo iOS debido a los problemas que pueden existir de compatibilidad. Solo se han considerado los gadgets que trabajen con Android.

Como medio de transmisión inalámbrica, es necesario que tenga Bluetooth (BT) pero es recomendable que tenga Bluetooth Low Energy (BLE), protocolo que está basado en el Bluetooth normal pero que requiere de menos energía que este.

En la Tabla 2 se muestran las características de los relojes inteligentes que cumplen los requisitos previamente mencionados.

MODELO	SENSOR DE SpO2	SENSOR DE FRECUENCIA CARDÍACA	BLUETOOTH	SOFTWARE	ALMACENAMIENTO INTERNO
TicWatch Pro 3	SI	SI	SI Bluetooth 4.2 y BLE	Wear OS	8GB
Garmin Vivoactive 4	SI	SI	SI Bluetooth y BLE	Garmin OS Compatible con Android	4GB
Fitbit Versa 2	SI	SI	SI Bluetooth 4.0	Fitbit OS Compatible con Android	4GB

Tabla 2. Características de los relojes inteligentes.

Entre los seleccionados se ha hecho un análisis de las ventajas y desventajas de cada uno de ellos. Se ha puntuado cada característica con unos valores numéricos entre 1 y 3 (siendo 1 malo, 2 normal y 3 bueno).

MODELO	PRECIO	BATERÍA	SISTEMA OPERATIVO	ALMACENAMIENTO INTERNO	BLUETOOTH	TOTAL
TicWatch Pro 3	1	2	3	3	3	12
Garmin Vivoactive 4	1	3	2	2	3	11
Fitbit Versa 2	1	2	2	2	2	9

Tabla 3. Comparativa de elección de los relojes inteligentes.

Tras haber realizado el análisis de los tres dispositivos, en cuanto a requisitos, el dispositivo más adecuado es el TicWatch Pro, siendo el dispositivo que tiene el sistema operativo más adecuado y la unidad de almacenamiento más grande respecto a las otras dos alternativas.

En cuanto a la posibilidad de captura los datos, los tres sistemas operativos permiten acceder a los datos de los sensores. No obstante, los sistemas operativos Garmin OS y Fitbit OS guardan esos datos en sus propios servidores y para acceder a ellos es necesario pedir permiso a ese servidor a través de sus plataformas. El inconveniente que tiene este método de obtención de datos, es que no se pueden obtener los datos en tiempo real. Con el sistema operativo Wear OS, al contrario, sí que se pueden leer y recoger los datos en tiempo real.

Por lo tanto, el reloj inteligente seleccionado va a ser el **TicWatch Pro 3** [33], ya que es el único reloj de los tres que opera con Wear OS y va a permitir recoger los datos en tiempo real.

6.2. HARDWARE DE OBTENCIÓN DE DATOS

A la hora de recibir, almacenar y procesar los datos, se necesita de un medio físico que sea capaz de recoger la información de manera inalámbrica. Para ello se utilizará un teléfono móvil que mediante una conexión Bluetooth se conecte al reloj y se realice el envío de datos.

Actualmente existe una amplia variedad de teléfonos móviles con todo tipo de características y precios. Para hacer un primer descarte se han impuesto los siguientes criterios de selección. El primero será que el dispositivo venga de fábrica con una versión de Android 11 o superior

instalada. El segundo criterio está relacionado con el precio, limitándolo a un máximo de 300€, ya que se quiere probar que la aplicación que se vaya a instalar funcione de manera adecuada en móviles con procesadores más lentos, de modo que se asegure que en móviles con mejores procesadores no haya ningún problema. Como último criterio, se ha priorizado la elección de móviles con gran capacidad de almacenamiento.

En la Tabla 4 se hace una comparativa de los ciertos móviles, previamente seleccionados, dependiendo de las ventajas y desventajas que cada uno tiene, puntuando del 1 al 3 cada característica (siendo 1 malo, 2 normal y 3 bueno).

MÓVIL	PRECIO	PESO	PROCESADOR	ALMACENAMIENTO	TOTAL
Realme GT Master Edition	1	2	2	2	10
Xiaomi 11 Lite 5G NE	1	3	3	3	12
POCO F3	2	1	2	3	11
Xiaomi Redmi Note 10 5G	3	2	1	2	11

Tabla 4. Comparativa de teléfonos móviles.

A la vista de los resultados obtenidos de la comparativa realizada, el móvil con mejor puntuación es el **Xiaomi 11 Lite 5G NE** [34] y, por tanto, el teléfono móvil que se usará para este trabajo fin de máster.

6.3. SOFTWARE DE ADQUISICIÓN DE DATOS

Para la obtención y el procesamiento de datos, se va a trabajar en el entorno de Android Studio. Android Studio es el entorno de desarrollo integrado oficial para el desarrollo de apps para Android. Este entorno permite crear una aplicación en el reloj capaz de acceder al sensor

y monitorizar la frecuencia cardíaca. De la misma manera, se puede crear una aplicación para el dispositivo móvil para que este sea capaz de recibir la información y procesarla.

Como se puede apreciar en la Figura 11, el entorno de desarrollo Android Studio ofrece tres lenguajes de programación en los que se puede programar: Java, Kotlin y C++. Partiendo de la base de que no se tienen nociones básicas de ninguno de los tres idiomas, se va a hacer una comparativa de los tres lenguajes, realizando posteriormente la elección de uno de ellos.

Figura 11. Lenguajes de programación de Android Studio.

C++ es un lenguaje cuyo aprendizaje es más complejo y además es el idioma de programación más antiguo de los tres, por lo que tiene muchas reglas y limitaciones. Cuando se comete algún error al codificar, no presenta mensajes de errores o excepciones, lo que dificulta aún más su aprendizaje.

Java es un lenguaje más adecuado para aquellos que quieren comenzar a programar. A diferencia de C++, este lenguaje imprime mensaje de error y ayuda a encontrar qué está mal en el código, proporcionando un nivel de aprendizaje más sencillo. Sin embargo, sigue siendo un lenguaje complicado.

Kotlin es el lenguaje más nuevo desarrollado por Google y de entre los tres es el más accesible. Es un lenguaje relativamente simple, con menos formalidades y reglas que los anteriores. Sin embargo, al ser el lenguaje más nuevo, no está estandarizado, hay menos soporte y menos soluciones de problemas disponibles. Lo que puede suponer una gran dificultad a la hora de aprender a programar en este idioma.

En la Tabla 5 se muestra una comparativa de los tres lenguajes de programación que existen para desarrollar la solución software. Cada característica del lenguaje ha sido puntuada del 1 al 3 (siendo 1 malo, 2 normal y 3 bueno).

LENGUAJE	FACILIDAD APRENDIZAJE	RECURSOS	ORIENTADO A OBJETOS	VÁLIDO PARA OTROS SOFTWARES	MÁS UTILIZADO	TOTAL
JAVA	2	3	3	3	3	19
KOTLIN	3	1	3	1	1	14
C++	1	3	3	3	2	12

Tabla 5. Comparativa de lenguajes de programación.

En vista de los resultados de las puntuaciones, el lenguaje con mayor puntuación es Java y por consiguiente, el seleccionado. A pesar de no ser el idioma más fácil de aprender, es el idioma de programación más utilizado por el momento. Además, cuenta con una gran comunidad de personas que programan en este lenguaje y esto permite poder realizar más consultas y obtener más variedad de soluciones a la hora de programar.

6.4. TIPOS DE ALMACENAMIENTOS

Cuando se trabaja con Android Studio, existe la posibilidad de almacenar la información obtenida de las monitorizaciones en una base de datos. Existen dos opciones para esta implementación.

La primera opción es Firebase. Esta es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Esta plataforma cuenta con bases de datos en tiempo real, que se almacenan en la nube en formato JSON. Tiene la capacidad de almacenar y disponer de los datos de la aplicación en tiempo real, manteniéndolos actualizados sin que el usuario realice ninguna acción. La parte negativa de esta opción es que no es de código abierto, depende del proveedor y solo está disponible en las bases de datos que no tienen formato tabla.

La otra opción es SQLite, una herramienta de software libre que permite almacenar la información en dispositivos empotrados de una manera sencilla, eficaz, potente y rápida. Como inconveniente hay que mencionar que no soporta cantidades de datos superiores a terabytes.

De entre las dos opciones posibles se ha escogido SQLite por ser una herramienta de software que permite almacenar información en dispositivos con poca capacidad de hardware, como es el caso de los teléfonos móviles y por su facilidad de uso y configuración.

7. DESARROLLO DE LA SOLUCIÓN SOFTWARE PARA LA MONITORIZACIÓN DE VARIABLES FISIOLÓGICAS

Con el objetivo de lograr una solución software para monitorizar la frecuencia cardíaca de manera no invasiva para personas con movilidad reducida, se va a proceder a explicar el procedimiento llevado a cabo. Conociendo la situación actual que existe de saturación en los centros de salud debido a aumento de personas de elevada edad y personas que sufren de limitaciones físicas, se quiere desarrollar una solución capaz de monitorizar los estados de salud de estas personas sin que tengan la necesidad de acudir a las consultas. Para ello, se diseñará una solución software que permitirá ser implantada en dispositivos no invasivos, de manera que no altere las vidas de estas personas.

La solución software, que consta de dos aplicaciones, irán instaladas en un reloj inteligente y en un teléfono móvil, de manera que se pueda realizar la monitorización de forma no invasiva. Los datos recogidos vía Bluetooth se almacenarán, para luego poder procesarlos. Esto permitirá al personal sanitario realizar una detección temprana y disminuir riesgos de la evolución de las enfermedades.

A continuación, se expondrá una visión general de la solución a desarrollar y sus fases. Posteriormente, se explicará el proceso realizado para el desarrollo de ambas aplicaciones que tendrán la finalidad de capturar y recoger los datos. Finalmente, para validar la solución, se explicarán los ensayos experimentales realizados. En la siguiente imagen (Figura 12) se esquematiza como se implementará la solución en la vida real, una vez finalizada su desarrollo.

Figura 12. Proceso completo de la solución propuesta.

7.1. VISIÓN GENERAL

El trabajo consta de dos fases principales, un primer desarrollo software para la obtención de datos desde el reloj inteligente; y la segunda fase, un desarrollo software para la recogida, almacenaje y procesamiento de los datos en el móvil.

En la Figura 13 se muestra un esquema de las tres principales funcionalidades que va a suponer la monitorización y a qué dispositivo pertenece cada función. En el caso de la captura de los datos de las frecuencias, estas se recogerán accediendo al sensor del reloj inteligente. Los datos se enviarán al teléfono móvil vía BT, para visualizarse en la pantalla en tiempo real y guardar esas frecuencias en la base de datos. Finalmente, los datos se enviarán por email para poder procesarlos desde un ordenador.

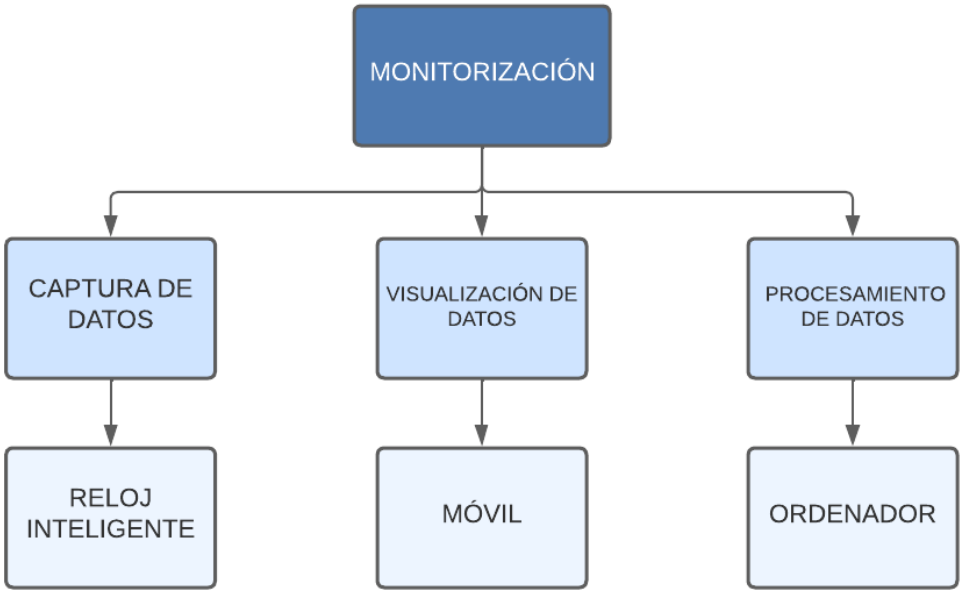

Figura 13. Visión general del proyecto.

El diseño de las aplicaciones será sencillo e intuitivo, para facilitar el uso de ellas. Por ello, se ha priorizado la funcionalidad frente al diseño de las aplicaciones.

7.2. DESARROLLO DE LA APLICACIÓN SOFTWARE PARA LA CAPTURA DE DATOS EN EL RELOJ INTELIGENTE

La aplicación a diseñar bajo el sistema operativo Android para el smartwatch, consiste en acceder a los datos que recoge el sensor de frecuencia cardíaca para luego enviarlos vía BT al móvil.

Los objetivos que deberá cumplir la aplicación a desarrollar son los siguientes:

- Acceder al sensor de la frecuencia cardiaca del reloj.
- Leer los datos en tiempo real.
- Comprobar el estado del Bluetooth (BT).
- Establecer una conexión Bluetooth con el dispositivo receptor.
- Enviar los datos en tiempo real.

La interfaz del programa desarrollado consta de una única pantalla (Figura 14), la cual está compuesta por tres botones y un widget, que muestra en pantalla el valor de la frecuencia cardíaca que se envía por BT.

Figura 14. Interfaz gráfica del reloj.

Como proceso general, la aplicación internamente, ejecuta el algoritmo que se puede ver esquematizado en la siguiente Figura 15. El código correspondiente a este programa se puede encontrar detallado en el ANEXO II.

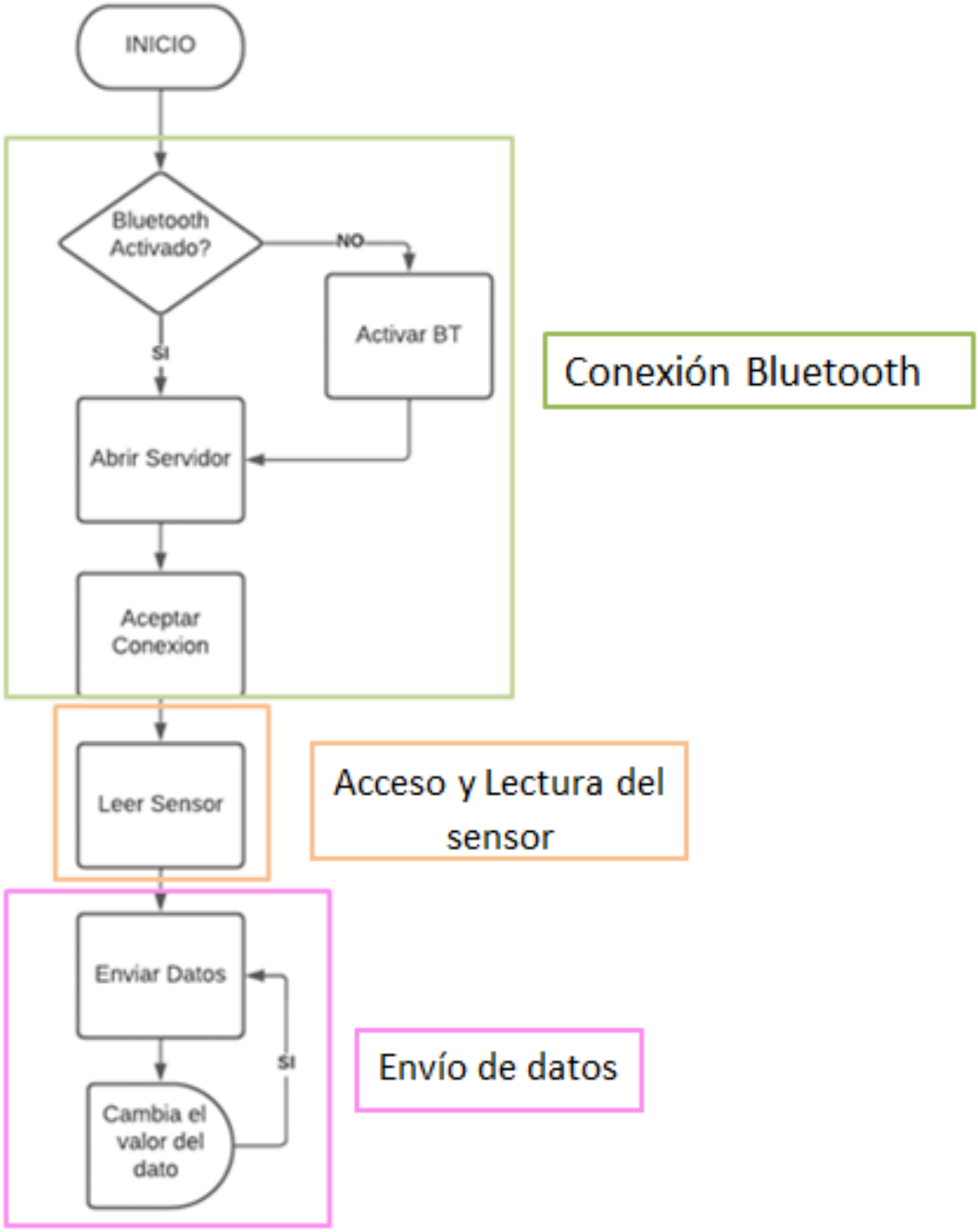

Figura 15. Diagrama del funcionamiento general de la aplicación del reloj.

El programa se puede dividir en tres subapartados principales:

1) Conexión Bluetooh.

2) Acceso al sensor y lectura de datos.

3) Envio de datos.

A su vez, estos procesos están programados en diferentes clases:

- En la clase MainActivity se gestionará toda la interfaz y el acceso al sensor y la lectura de datos.
- En la clase BluetoothConnectionLeService, estará todo el código relacionado con la conexion BT y con el envío de datos.

A continuación, se procederá a explicar en detalle los procedimientos realizados en la aplicación. Antes de comenzar con la explicación de cada método, es necesario dar permisos a la aplicación tanto para que pueda acceder al Bluetooth del reloj como para que pueda acceder a los sensores. Todo esto se debe de implantar en el manifiesto, tal y como se muestra en el código de la Figura 16.

Archivo: AndroidManifest.xml

El archivo AndroidManifest describe la información esencial de la aplicación Android para las herramientas de creación en Android. Para poder acceder a las funciones del sistema ha sido necesario incluir en el archivo AndroidManifest.xml los permisos de Bluetooth y de acceso a los sensores. Esto se ha realizado añadiendo estas líneas de código.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
<uses-permission android:name="android.permission.BLUETOOTH_PRIVILEGED" />
<uses-permission android:name="android.permission.BODY_SENSORS" />
```

Figura 16. Permisos del reloj inteligente.

Para dispositivos que ejecutan una interfaz de programación de aplicaciones (API) superior a la 23 no es suficiente con añadir en el manifiesto los permisos si no que se deben comprobar los

permisos del BT en el propio código de la aplicación. Para ello se ha implementado el código mostrado en la Figura 17:

```
private void checkBTPermissions() {
    if(Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP){
        int permissionCheck = this.checkSelfPermission("Manifest.permission.ACCESS_FINE_LOCATION");
        permissionCheck += this.checkSelfPermission("Manifest.permission.ACCESS_COARSE_LOCATION");
        if (permissionCheck != 0) {

            this.requestPermissions(new String[]{Manifest.permission.ACCESS_FINE_LOCATION,
                    Manifest.permission.ACCESS_COARSE_LOCATION}, requestCode: 1001); //Any number
        }
    }else{
        Log.d(TAG, msg: "checkBTPermissions: No need to check permissions. SDK version < LOLLIPOP.");
    }
}
```

Figura 17. Método para aceptar permisos manualmente.

7.2.1. CONEXIÓN BLUETOOTH

El primer paso ha sido crear el método que active y desactive el BT. Para ello se ha colocado el botón de ON/OFF en la interfaz, que en cuanto se pulse comprobará si el BT está activado y de no serlo así, lo activará. En la Figura 18 se muestra el método correspondiente a la comprobación del estado del BT y en función de su estado a su posterior activación o desactivación.

```
// Conectar y desconectar Bluetooth
public void enableDisableBT(){
    if(mBluetoothAdapter == null){
        Log.d(TAG, msg: "enableDisableBT: Does not have BT capabilities.");
    }
    if(!mBluetoothAdapter.isEnabled()){
        Log.d(TAG, msg: "enableDisableBT: enabling BT.");
        Intent enableBTIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivity(enableBTIntent);

        IntentFilter BTIntent = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
        registerReceiver(mBroadcastReceiver1, BTIntent);
    }
    if(mBluetoothAdapter.isEnabled()){
        Log.d(TAG, msg: "enableDisableBT: disabling BT.");
        mBluetoothAdapter.disable();

        IntentFilter BTIntent = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
        registerReceiver(mBroadcastReceiver1, BTIntent);
    }

}
```

Figura 18. Método de conexión y desconexión del Bluetooth.

Una vez conectado el BT, es necesario configurar el canal de transmisión. En este caso la conexión BT utiliza un protocolo de cliente-servidor. El reloj, que en este caso será el encargado de transferir los datos, funcionará como servidor. Esto quiere decir que el reloj es el encargado de abrir un canal de comunicación BluetoothServerSocket a través de un identificador UUID y se quedará a la espera de que otro dispositivo solicite establecer una conexión con él. Para ello se ha implementado el siguiente código (Figuras 19 y 20):

```
public AcceptThread() {
    BluetoothServerSocket tmp = null;
    // Create a new listening server socket
    try {
        tmp = mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(appName, MY_UUID_INSECURE);

        Log.d(TAG, msg: "AcceptThread: Setting up Server using: " + MY_UUID_INSECURE);
    } catch (IOException e) {
        Log.e(TAG, msg: "AcceptThread: IOException: " + e.getMessage());
    }
    mmServerSocket = tmp;
}
```

Figura 19. Método para crear servidor.

```
public void run() {
    Log.d(TAG, msg: "run: AcceptThread Running.");
    BluetoothSocket socket = null;
    try {
        // This is a blocking call and will only return on a
        // successful connection or an exception
        Log.d(TAG, msg: "run: RFCOM server socket start.....");
        //The socket is accepted when other device ask to establish a connection
        socket = mmServerSocket.accept();
        Log.d(TAG, msg: "run: RFCOM server socket accepted connection.");

    } catch (IOException e) {
        Log.e(TAG, msg: "AcceptThread: IOException: " + e.getMessage());
    }
    if (socket != null) {
        connected(socket, mmDevice);
    }
    Log.i(TAG, msg: "END mAcceptThread ");
}
```

Figura 20. Método de espera y aceptación de la conexión del servidor.

Una vez solicitada la petición de establecer la conexión se ha programado un método encargado de mantener la conexión BT y de gestionar la transferencia de datos. Para ello, se creará un socket llamado BluetoothSocket para poder realizar la transferencia de datos. Mientras la conexión se mantenga establecida, el BluetoothSocket irá tomando los valores que

recoga el sensor hasta que se cree una excepción, es decir, hasta que se cierre la conexión. En la próxima imagen (Figura 21) se puede ver el algoritmo que sigue el proceso completo de la conexión Bluetooth por parte del reloj.

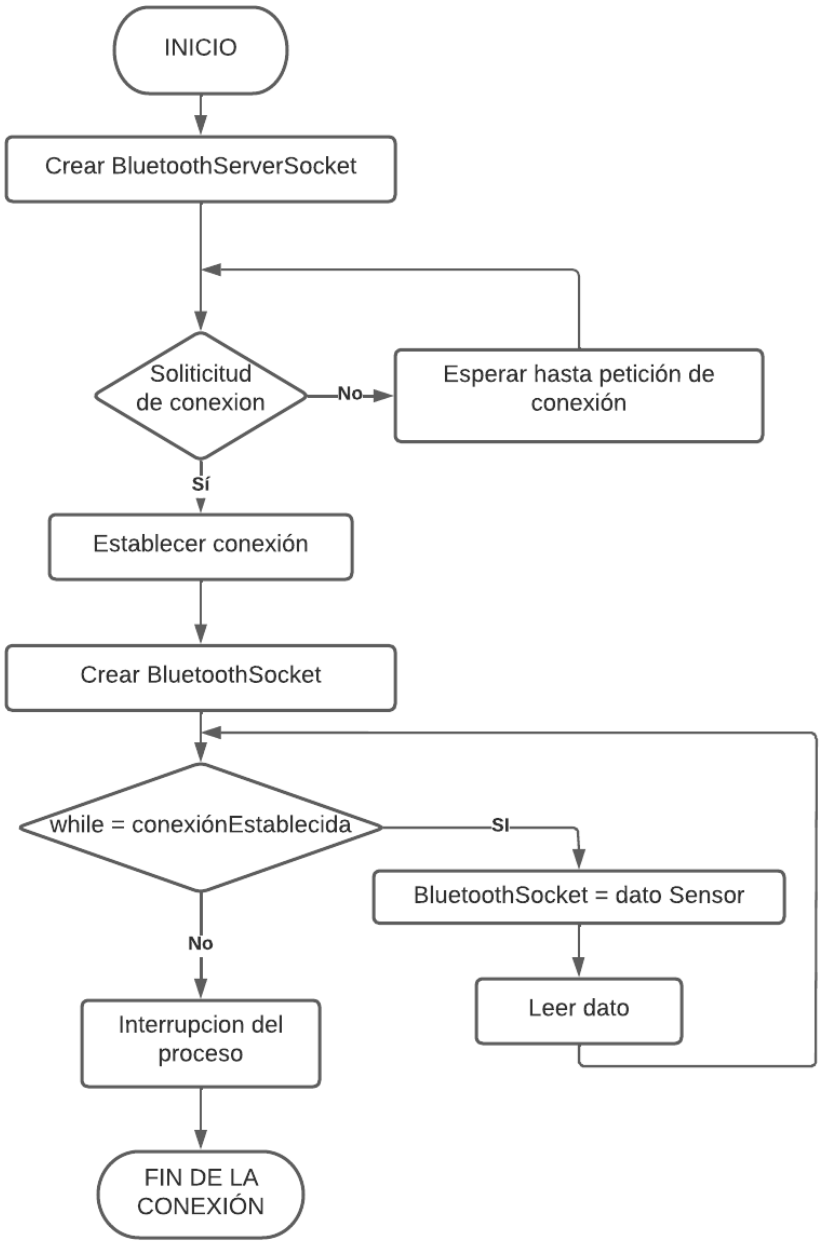

Figura 21. Diagrama de la conexión Bluetooth.

7.2.2. ACCESO Y LECTURA DEL SENSOR

Para leer los valores que detecta el sensor primero de todo se debe comprobar que existe el sensor de la frecuencia cardíaca. Una vez hecha la comprobación, se debe de acceder al sensor y leer los los datos que el sensor captura. Se ha programado el método para que la lectura se haga cada 1 segundo y se muestre en la pantalla del reloj el valor numérico de la frecuencia cardíaca.

En la Figura 22 se muestra el método correspondiente a la comprobación de la existencia del sensor, el método encargado de finalizar la lectura del sensor y el método que lee los valores de las frecuencias que el sensor va capturando cada segundo. La Figura 22 representa, a través de un diagrama de flujo, el algoritmo que siguen las líneas de código programadas en la Figura 23.

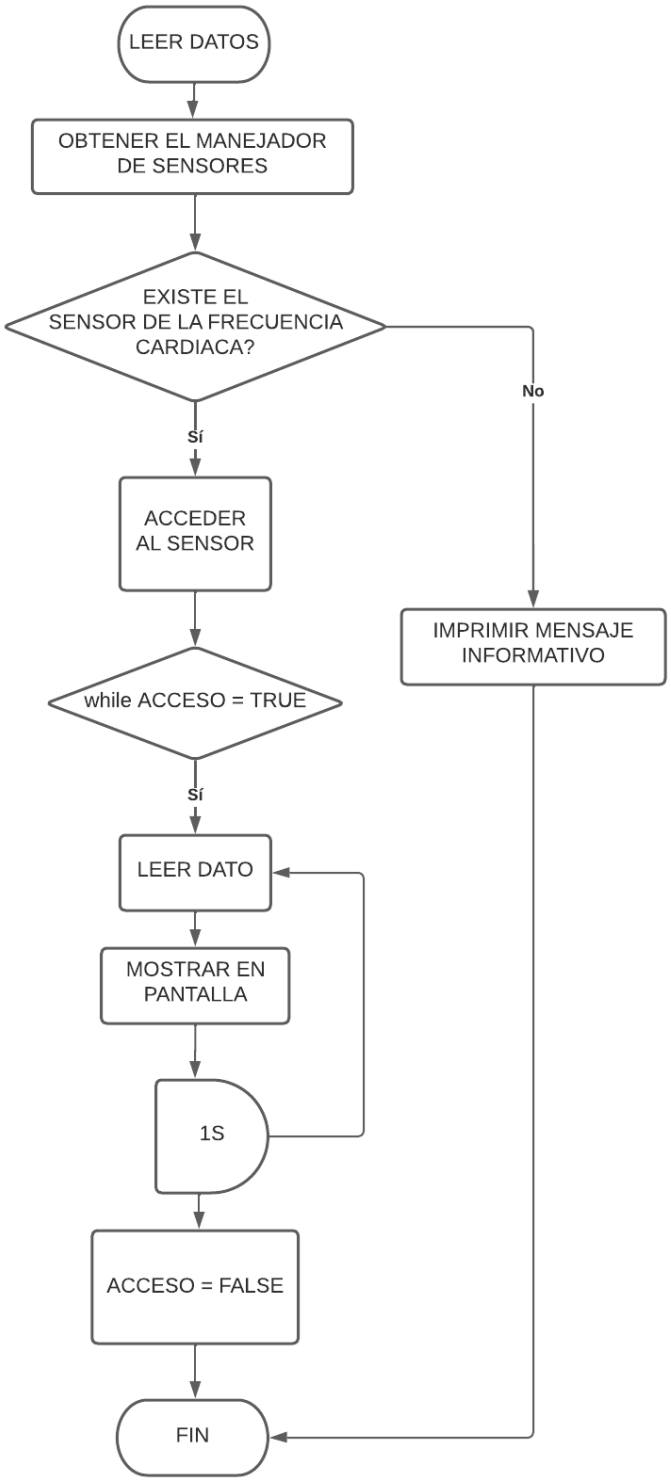

Figura 22. Diagrama de la lectura de datos.

```
private void getHeartRate() {
    mSensorManager = ((SensorManager)getSystemService(SENSOR_SERVICE));
    if (mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE) != null){
        mHeartRateSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
        sensorregistered = mSensorManager.registerListener( listener: this, mHeartRateSensor,
                SensorManager.SENSOR_DELAY_NORMAL);}
    else {
        Log.d(TAG, msg: "Sensor no disponible");
    }
}
private void stopmeasure() { mSensorManager.unregisterListener(this); }
@Override
public void onAccuracyChanged (Sensor sensor, int accuracy) {
    Log.d(TAG, msg: "onAccuracyChanged - accuracy: " + accuracy);
}
@Override
public void onSensorChanged (SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_HEART_RATE) {
        heartrate = (int) event.values[0];
        msg = "Frecuencia Cardiaca: " + (int) event.values[0];
        etSend.setText(msg);
       //Lamada al metodo para que envie los datos al dispositivo remoto
        mBluetoothConnection.write(heartrate);
        Log.d(TAG, msg: "Dato enviado:" + heartrate);
    } else
        Log.d(TAG, msg: "Sensor de frencuencia cardiaca no reconocido");
}
```

Figura 23. Métodos para acceder y leer datos del sensor.

7.2.3. ENVÍO DE DATOS

Para realizar el envío de datos, se ha programado un método que se encarga de extraer el valor del BluetoothSocket y enviarlo al otro dispositivo, que se activará al pulsar el botón ENVIAR de la interfaz. Este método se ha creado mediante la implementación de estas líneas de código y será llamado desde la clase principal para realizar el envío de datos. En la Figura 24 se muestra el código correspondiente al método encargado de enviar los valores de las frecuencias cardíacas al dispositivo receptor.

```
//Call this from the main activity to send data to the remote device
public void write(int bytes) {
    int text = bytes;
    Log.d(TAG,  msg: "write: Writing to outputstream: " + text);
    try {
        mmOutStream.write(bytes);
    } catch (IOException e) {
        Log.e(TAG,  msg: "write: Error writing to output stream. " + e.getMessage() );
    }
}
```

Figura 22. Código de envío de datos al receptor.

7.3. DISEÑO DE APLICACIÓN SOFTWARE PARA MÓVIL

La aplicación software diseñada para móvil funcionará como sistema de monitorización de la frecuencia cardíaca. Su código completo está disponible en el apartado ANEXO II.

Los objetivos principales que debe cumplir esta aplicación son los siguientes:

- Establecer una conexión Bluetooth con el reloj.
- Recoger y visualizar los datos en tiempo real.
- Presentar los datos en un gráfico.
- Almacenar los datos en una base de datos.
- Realizar búsquedas a través del acceso de la base de datos.
- Exportar los datos vía email.

La aplicación esta formada por una interfaz que cuenta con cuatro pantallas.

1) Registra un nuevo usuario.
2) Activa el Bluetooth, establece la conexión y visualiza los datos.
3) Muestra el gráfico en pantalla, guarda los datos en la base de datos, los exporta a un archivo CSV y manda por email el archivo.
4) Realiza la búsqueda del usuario introducido y muestra en pantalla sus datos recogidos.

En la Figura 25 se muestran las 4 pantallas de las que está formada la aplicación de móvil con sistema operativo Android.

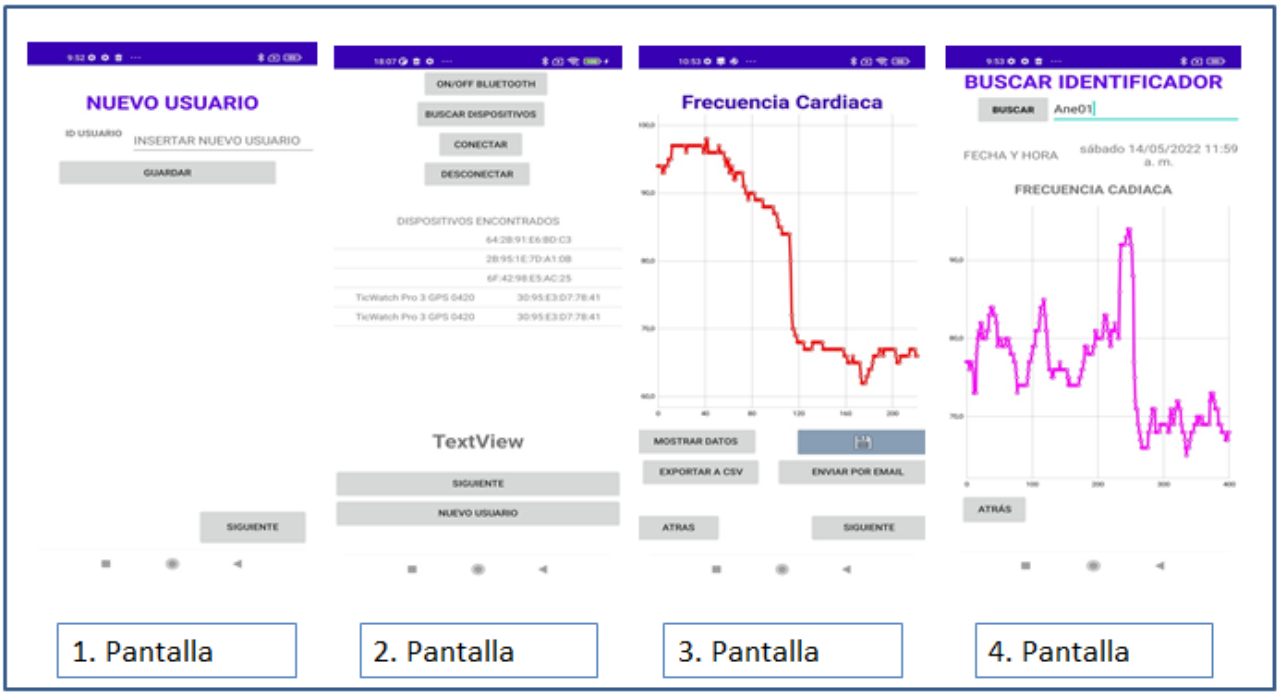

Figura 23. Pantallas que componen la aplicación.

Archivo: AndroidManifests.xml

De la misma forma que se ha hecho para la aplicación del reloj, en el caso de la aplicación móvil, además de incluir los permisos del Bluetooth y sensores, se han incluido los permisos de localización, los permisos de lectura externa y los permisos de escritura externa. Los permisos incluidos se pueden ver en las siguientes líneas de código que se muestra en la Figura 26:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_PRIVILEGED"
    tools:ignore="ProtectedPermissions" />
<uses-permission android:name="android.permission.BODY_SENSORS" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Figura 24. Permisos del móvil.

Clase: NuevoUsuarioActivity

Para registrar los datos del usuario, se ha diseñado la interfaz mostrada en la Figura 28. Consta de un botón de guardado, un botón que permite pasar a la siguiente pantalla y una celda donde escribir la información a registrar. Internamente el algoritmo que sigue la interfaz se representa en la Figura 27:

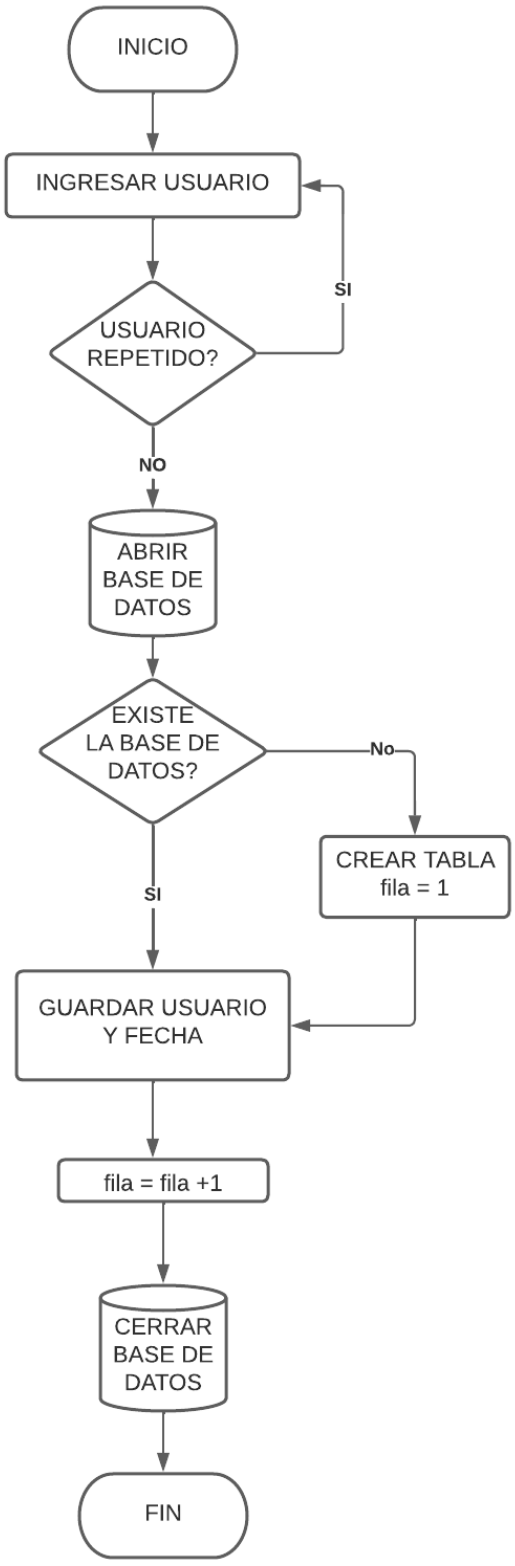

Figura 27. Diagrama de la clase NuevoUsuario.

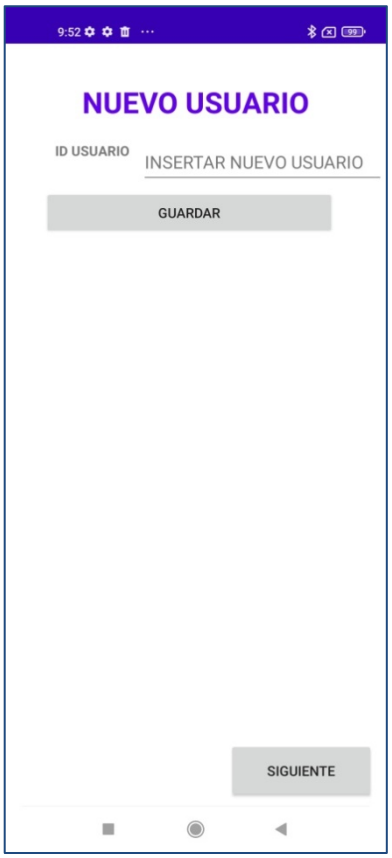

Figura 28. Interfaz de registro de nuevo usuario

En la Figura 27 se muestra el diagrama del flujo del algoritmo encargado de crear la tabla correspondiente a los usuarios, añadir a la tabla los nuevos registros de usuarios junto con la fecha y hora de registro y comprobar que no haya usuarios repetidos.

Esta clase se encarga de comprobar si existe un usuario guardado en la base de datos que tiene el mismo nombre cuando se introduce un nuevo usuario. Si es así, se informará en pantalla de que ya existe un registro con ese nombre y se pedirá que se introduzca un nuevo usuario. Al pulsar el botón de guardar, si es el primer usuario que se registra se creará una base de datos que tendrá una tabla (ver Figura 29) donde se guardará el usuario, la fecha y la hora en la que se ha realizado el registro. Si la base de datos ya ha sido creada anteriormente, se añadirán los datos en una nueva fila.

Identificador	Fecha_y_Hora
Ana	martes 21/06/2022 11:46 a. m.
Juan	martes 21/06/2022 11:51 a. m.
Mikel	martes 21/06/2022 11:53 a. m.
Patrick	martes 21/06/2022 11:55 a. m.

Figura 25. Tabla de Usuario de la Base de Datos.

Clase: ConnectionService

Esta clase se encarga de gestionar la conexión Bluetooth. En el caso del móvil, él es quien recibe la información por lo que funciona como cliente. Para que esta clase se pueda ejecutar primero se comprueba que el Bluetooth está activado, de lo contrario muestra un mensaje informando de que es necesario conectar el BT. La clase ConnectionLeService se encarga de pedir permiso al servidor para poder conectase a él y una vez establecida la conexión abre un canal para la transferencia de datos. Para el caso del móvil se ha implementado el siguiente código (ver Figuras 30) que permite establecer un canal de comunicación con otro dispositivo el cual posibilita la trasferencia de información:

```
private class ConnectThread extends Thread {
    private BluetoothSocket mmSocket;
    public ConnectThread(BluetoothDevice device, UUID uuid) {
        Log.d(TAG,  msg: "ConnectThread: started.");
        mmDevice = device;
        deviceUUID = uuid;
    }

public void run() {
    BluetoothSocket tmp = null;
    Log.i(TAG,  msg: "RUN mConnectThread ");
    // Get a BluetoothSocket for a connection with the given BluetoothDevice
    try {
        Log.d(TAG,  msg: "ConnectThread: Trying to create InsecureRfcommSocket using UUID: "
                + MY_UUID_INSECURE);
        tmp = mmDevice.createRfcommSocketToServiceRecord(deviceUUID);
    } catch (IOException e) {
        Log.e(TAG,  msg: "ConnectThread: Could not create InsecureRfcommSocket " + e.getMessage());
    }
    mmSocket = tmp;
    // Always cancel discovery because it will slow down a connection
    mBluetoothAdapter.cancelDiscovery();
    // Make a connection to the BluetoothSocket
    try {
        // This is a blocking call and will only return on a successful connection or an exception
        mmSocket.connect();
        Log.d(TAG,  msg: "run: ConnectThread connected.");
    } catch (IOException e) {
        // Close the socket
        try {
            mmSocket.close();
            Log.d(TAG,  msg: "run: Closed Socket.");
        } catch (IOException e1) {
            Log.e(TAG,  msg: "mConnectThread: run: Unable to close connection in socket " + e1.getMessage());
        }
        Log.d(TAG,  msg: "run: ConnectThread: Could not connect to UUID: " + MY_UUID_INSECURE);


    }
    connected(mmSocket, mmDevice);
}
```

Figura 26. Código implementado para la conexión BT.

Clase: MainActivity

Esta clase es la encargada de activar el BT, buscar los dispositivos BT disponibles, establecer la conexión BT y visualizar los datos de la frecuencia cardiaca en tiempo real. Para ello se ha diseñado la interfaz mostrada en la Figura 31:

Figura 27. Interfaz de la clase MainActivity.

Hay un total de seis botones, dos de ellos para gestionar el paso a otras activities y los otros cuatro botones para gestionar todas las funciones previamente mencionadas. El botón ON/OFF activa y desactiva el BT. El botón BUSCAR DISPOSITIVOS inicia la búsqueda de los dispositivos que se encuentran disponibles e imprime en pantalla el listado de dichos dispositivos. La lista permite seleccionar el dispositivo deseado (en este caso el TicWatch, pudiendo ser otros teléfonos móviles, relojes inteligentes…) y recoge sus datos (nombre del dispositivo y dirección MAC). Al mismo tiempo que sucede eso, se para la búsqueda de dispositivos.

En cuanto a establecer la conexión BT, el botón CONECTAR hace la llamada a la clase ConnectionService para que esta envíe una petición de conexión proporcionando sus datos al dispositivo seleccionado y se queda a la espera de que el otro dispositivo acepte su petición.

En el TextView que aparece en pantalla se imprimirán los valores numéricos de la frecuencia en cuanto estos se empiecen a recibir. Para finalizar la recepción de datos se ha colocado el botón DESCONECTAR. En la Figura 32 se muestra el diagrama de flujo de la clase MainActivity que gestiona la conexión BT, la búsqueda de dispositivos disponibles para establecer la conexión y una vez seleccionado el dispositivo gestiona la conexión y permite visualizar los datos que le llegan desde el reloj inteligente.

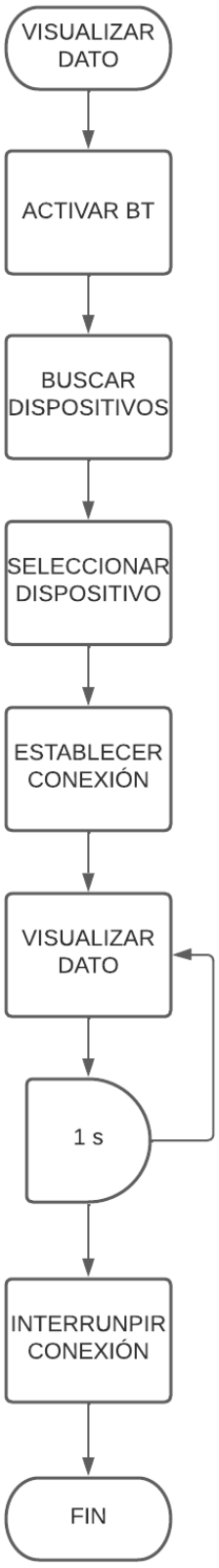

Figura 28. Diagrama de la clase MainActivity.

Clase: MostrarSeleccionActivity

Esta clase realiza una búsqueda en la base de datos en función del usuario que se introduce e imprime los datos que tiene guardados. Esos datos son la fecha en la que se ha hecho la monitorización y la gráfica de los valores que se han obtenido en ese proceso. En la siguiente pantalla, Figura 33, se representa lo que hace internamente esta activity.

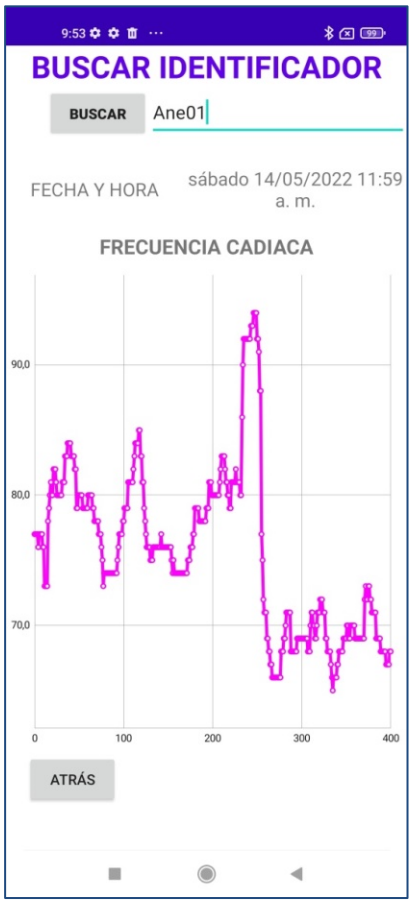

Figura 29. Interfaz del MostrarSeleccionActivity.

En la Figura 34 se presenta el flujo de datos de la algoritmia encargada de realizar la búsqueda del usuario que se introduce de manera manual y mostrar a través de una grafica los datos pertenecientes a ese usuario.

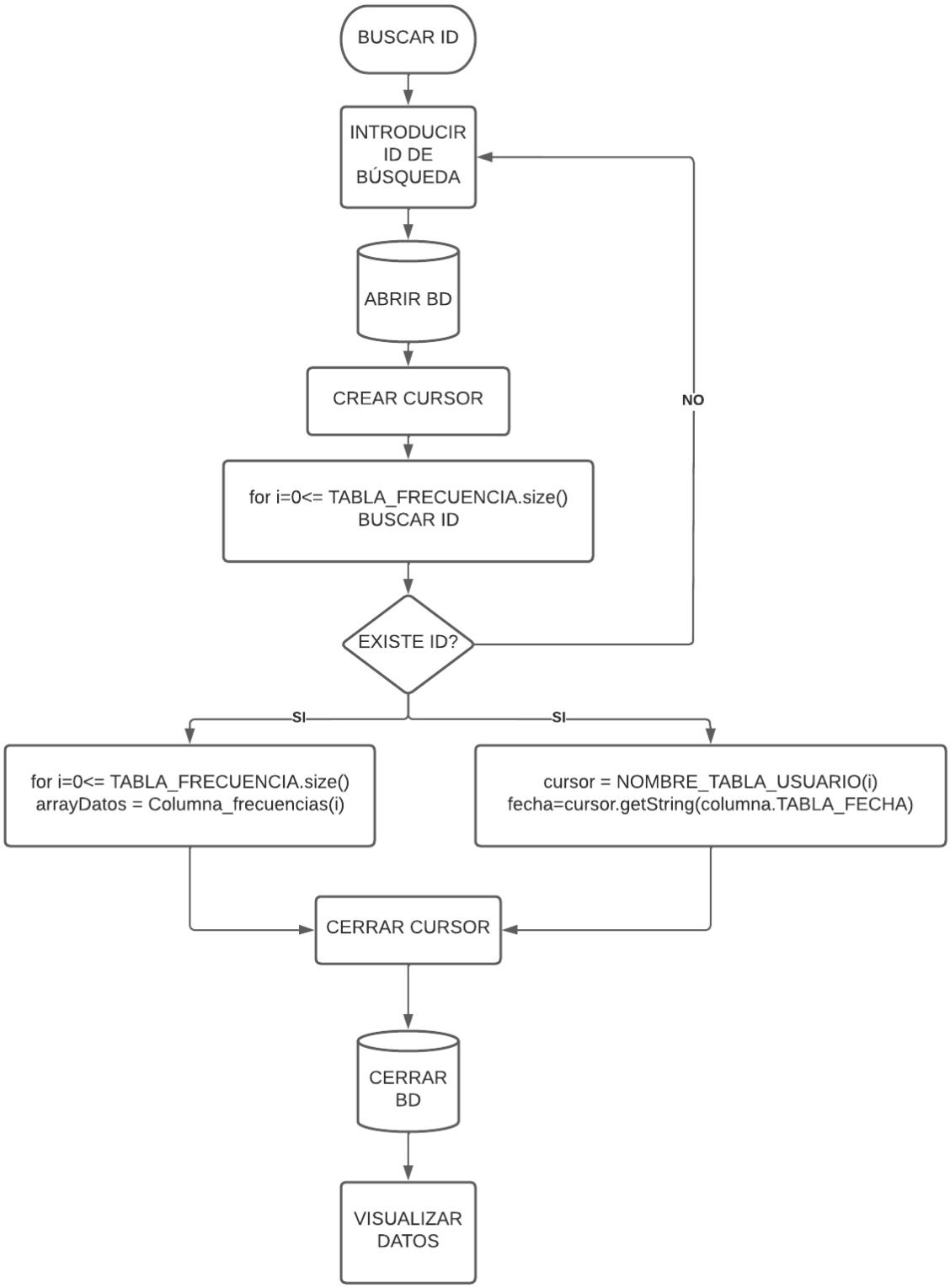

Figura 30. Diagrama de flujo de la clase MostrarSelección.

Clase: SQLiteHelper

Esta clase se encarga de crear la base de datos. Se han creado dos tablas, una tabla de usuario que consta de dos columnas: el usuario y la fecha. La otra que almacena la información relacionada con la frecuencia cardiaca que consta de tres columnas: el número de registro de la frecuencia, el usuario al que pertenece ese dato y la frecuencia cardiaca.

En la Figura 35 se muestra el código implementado del diseño de las tablas de la Base de Datos en SQLite.

```
/************************ On Create ***********************************/
@Override
public void onCreate(SQLiteDatabase database) {

    /** CREAR TABLA DE USUARIO **/
    /*---------------------- FORMATO TABLA ---------------------*/
    //- DATOS DEL USUARIO
    // | ID DEL USUARIO | HORA Y FECHA |

    database.execSQL(" CREATE TABLE " + NOMBRE_TABLA_USUARIO + " ( "
            + TABLA_ID +" INTEGER , "
            + TABLA_FECHA + " DATE DEFAULT CURRENT_DATE)");

    /*---------------------- FORMATO TABLA ---------------------*/
    //- DATOS DE LOS PARAMETROS
    // | Numero de Frecuencia | ID del Usuario | Fecha | Frecuencia Cardiaca |

    /**CREAR TABLA DE FRECUENCIAS CARDIACAS **/

    database.execSQL(" CREATE TABLE " + TABLA_FRECUENCIA + " ( "
            + COLUMNA_ENUMERADOR + " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "
            + COLUMNA_IDFRECUENCIAS + " INTEGER ,"
            + COLUMNA_TIMESTAMP  + "  DATE DEFAULT CURRENT_DATE,  "
            + COLUMNA_FRECUENCIA + " INTEGER )");

}// </ Oncreate >
```

Figura 31. Método de diseño de las tablas de la base de datos.

Clase: MostrarHeartRateActivity

Esta activity grafica en pantalla los valores que se han ido obteniendo de la frecuencia cardiaca. Como se puede ver en la Figura 36, la interfaz cuenta con 6 botones, 2 de ellos para gestionar el cambio a otras pantallas, y un gráfico donde mostrar los datos en pantalla.

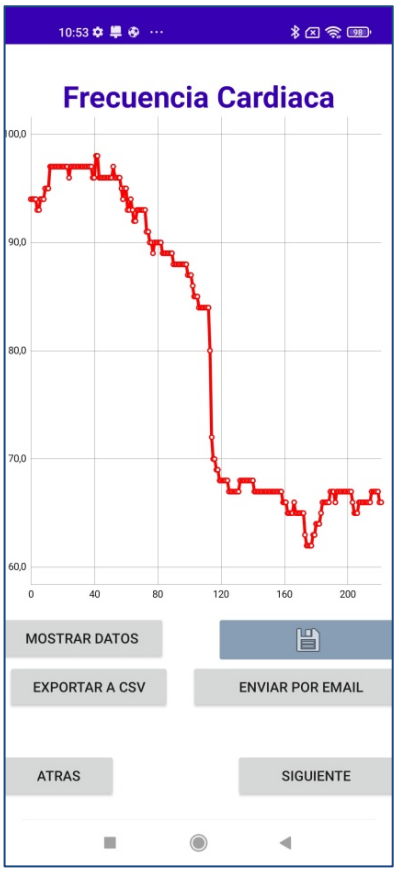

Figura 32. Interfaz de la clase MostrarHeartRateActivity.

Mientras la transmisión de datos se esté efectuando los datos se guardarán en un array para luego poderlos mostrar en el gráfico. Los datos se guardarán de manera automática al mismo tiempo que se visualizan en pantalla. Cuando los datos recogidos se quieran guardar en la base de datos, solo habrá que pulsar el botón de guardado. Estos datos se introducirán en las tablas previamente diseñadas de la base de datos (ver Figura 37). Mediante el siguiente diagrama, Figura 38, se explica lo que ocurre internamente en el código cuando se guardan los datos.

Numero_frecuencia	Identificador_Usuario	Fecha	Frecuencia
84	Ana	2022-06-21	65
85	Ana	2022-06-21	65
86	Ana	2022-06-21	65
87	Ana	2022-06-21	65
88	Ana	2022-06-21	65
89	Ana	2022-06-21	66
90	Ana	2022-06-21	66
91	Ana	2022-06-21	66
92	Ana	2022-06-21	66
93	Ana	2022-06-21	66
94	Ana	2022-06-21	66
95	Juan	2022-06-21	71
96	Juan	2022-06-21	71
97	Juan	2022-06-21	71
98	Juan	2022-06-21	71
99	Juan	2022-06-21	72
100	Juan	2022-06-21	71

Figura 33. Tabla de Frecuencias de la Base de Datos.

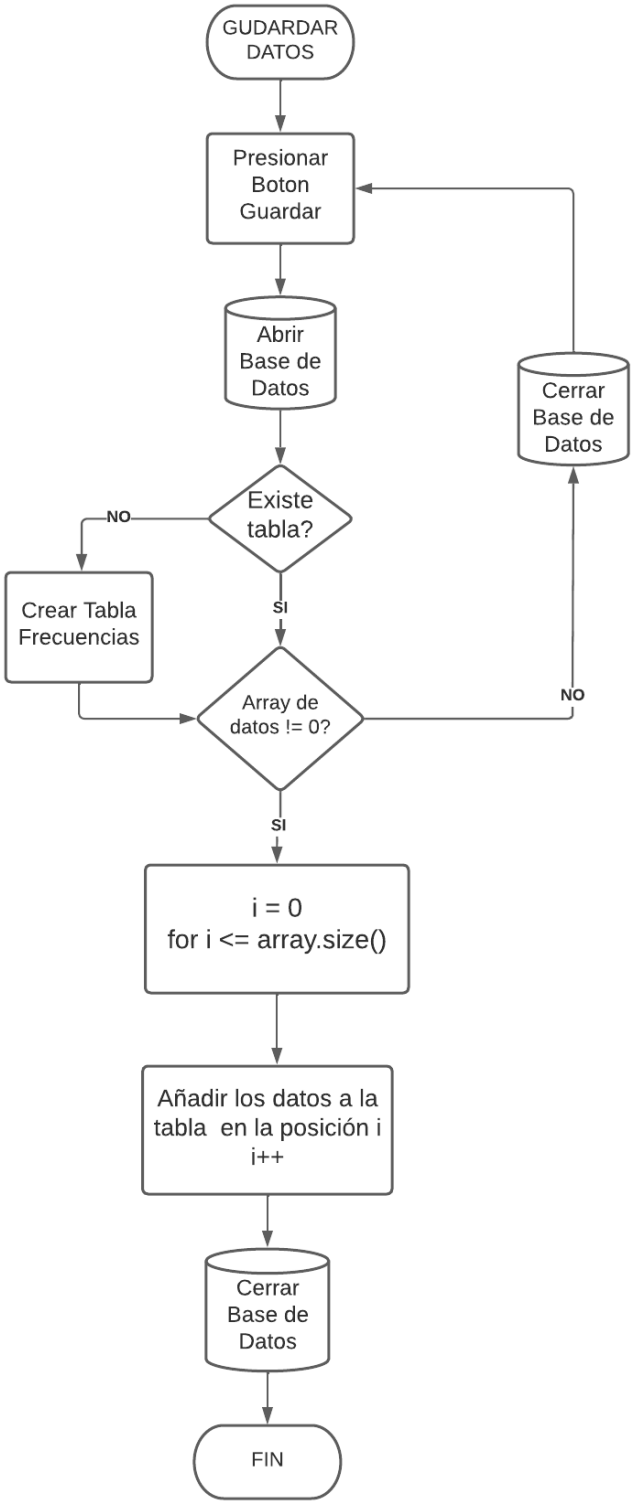

Figura 34. Diagrama de almacenamiento de datos.

En esa misma pantalla, se pueden exportar los datos a formato csv pulsando el botón EXPORTAR A CSV. Para ello será necesario crear una carpeta y un directorio donde guardar esa carpeta. A continuación, se deberá acceder a la base de datos, seleccionar de las tablas de la base de datos la información que se quiere exportar y añadirla al fichero anteriormente creado. Para terminar, una vez insertada toda la información en el fichero, se cerrará la base de datos. En la Figura 39 se presenta el algoritmo que sigue el método encargado de exportar la Base de Datos a un archivo con formato cvs.

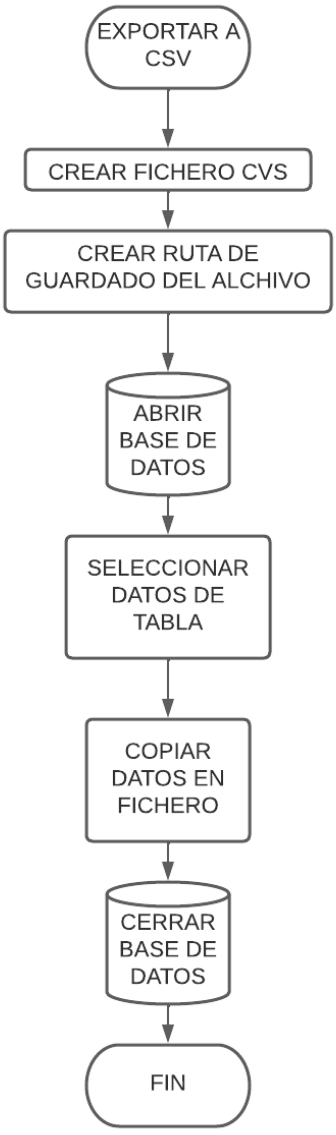

Figura 35. Diagrama de flujo de exportación de datos a formato CSV.

Clase: DeviceListAdapter

Esta clase define la estructura que tiene la ListView que muestra los diferentes dispositivos encontrados mediante Bluetooth. En la parte izquierda de la lista se imprimirá el nombre del dispositivo y en la derecha la dirección MAC como se puede ver en la Figura 40.

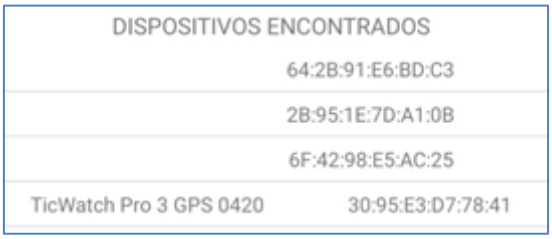

Figura 36. Diseño de la ListView.

Para el diseño de esta ListView se ha implementado el siguiente código (Figura 41):

```
public class DeviceListAdapter extends ArrayAdapter<BluetoothDevice> {
    private LayoutInflater mLayoutInflater;
    private ArrayList<BluetoothDevice> mDevices;
    private int mViewResourceId;
    public DeviceListAdapter(Context context, int tvResourceId, ArrayList<BluetoothDevice> devices) {
        super(context, tvResourceId, devices);
        this.mDevices = devices;
        mLayoutInflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        mViewResourceId = tvResourceId;
    }
    public View getView(int position, View convertView, ViewGroup parent) {
        convertView = mLayoutInflater.inflate(mViewResourceId,  root: null);
        //Si se encuentra un dispositivo imprimir en pantalla el nombre y su dirección MAC
        BluetoothDevice device = mDevices.get(position);
        if (device != null) {
            TextView deviceName = (TextView) convertView.findViewById(R.id.idDeviceName); //Nombre
            TextView deviceAdress = (TextView) convertView.findViewById(R.id.idDeviceAddress); //Dirección MAC
            deviceName.setText(device.getName());
            deviceAdress.setText(device.getAddress());
        }
        return convertView;
    }
} //<DeviceListAdapter>
```

Figura 37. Código de diseño de la ListView.

Tras haber desarrollado todo el código explicado previamente, se instalarán ambas aplicaciones en el reloj inteligente y en el teléfono móvil para poder realizar monitorizaciones. En la Figura 42 se muestra la implementanción real de la solución software desarrollada. El reloj inteligente capturará las frecuencias del individuo y estos serán enviados vía BT al teléfono móvil, graficando los datos y guardándolos en la base de datos.

Figura 38. Implementación de la aplicación desarrollada.

7.4. VALIDACIÓN DE LA SOLUCIÓN

Con la finalidad de validar la solución desarrollada, incluyendo la comunicación, captura de las variables, monitorización y guardado de los datos mediante la aplicación software desarrollada, se ha diseñado los siguientes ensayos experimentales.

Para dicha validación, se va a utilizar el teléfono móvil Xiaomi 11 Lite, donde se instalará la aplicación correspondiente al teléfono móvil y la aplicación wearable se instalará en el TicWatch. Para verificar la eficacia de la solución, se compararán los valores obtenidos de esta solución software con los valores obtenidos de otras monitorizaciones realizadas con diferentes dispositivos.

Con el propósito de ratificar la eficacia y robustez de las mediciones que se obtienen a través del reloj inteligente, se han realizado diferentes pruebas de monitorización. Se han comparado las medidas obtenidas del TicWatch con las obtenidas a través de un AppleWatch de séptima generación y de un pulsioxímetro, puesto que el pulsioxímetro solo recoge muestras válidas en estático. Los datos monitorizados por el pulsioxímetro se recogerán a través de una placa

Arduino que irá conectada al pulsioxímetro (Figura 43). De esta manera podrá ser posible hacer la comparativa entre los valores obtenidos por los diferentes dispositivos.

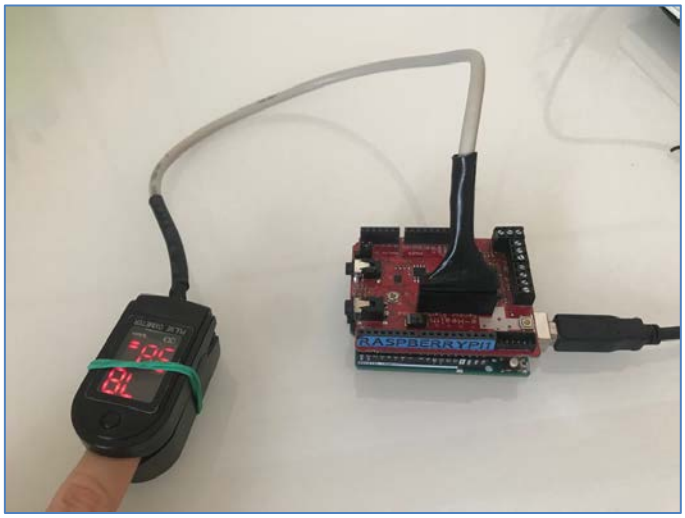

Figura 39. Pulsioxímetro utilizado para validación conectado a placa Arduino.

Para validar que el reloj inteligente TicWatch recoge valores veraces de la frecuencia cardíaca, se ha realizado una monitorización con los tres dispositivos al mismo tiempo (pulsioxímetro, TicWatch y Applewatch). Así, la monitorización de los tres dispositivos al mismo tiempo va a aportar los datos de las frecuencias de un mismo periodo de tiempo y tras ellos, se podrá comprobar si las mediciones realizadas por los tres dispositivos son iguales, comprobando así la eficacia de la solución diseñada. La utilización de los 2 dispositivos (Applewatch y pulsioximetro) ha sido para cerciorase de que los valores obtenidos son los correctos, ya que siempre existe la posibilidad de que un dispositivo esté defectuoso o esté descalibrado.

La prueba de monitorización ha consistido en un primer período en el cual el individuo ha permanecido en sedestación, seguido de una prueba de esfuerzo y finalmente vuelta a la posición de sedestación para su recuperación.

Como se puede ver en el siguiente gráfico (Figura 44), al inicio y al final de la prueba, los tres dispositivos miden unos valores muy parecidos. Sin embargo, a la hora de realizar un esfuerzo (franja limitada por las rayas discontinuas verdes, desde los segundos 260 a 340), los valores de ambos relojes inteligentes son parecidos, mientras que los valores del pulsioxímetro se mantienen constantes. Como ya se ha mencionado antes, el pulsioxímetro únicamente es

válido para monitorizar valores en estático. Se puede ver claramente en el gráfico cómo los valores se mantienen constantes durante la realización del esfuerzo.

Como conclusión, con este ensayo queda comprobado que tanto el TicWatch como la aplicación diseñada son válidos para realizar la monitorización de la frecuencia cardíaca. Adicionalmente, el TicWatch ha tenido una gran aceptación por el individuo al que se le han realizado las pruebas.

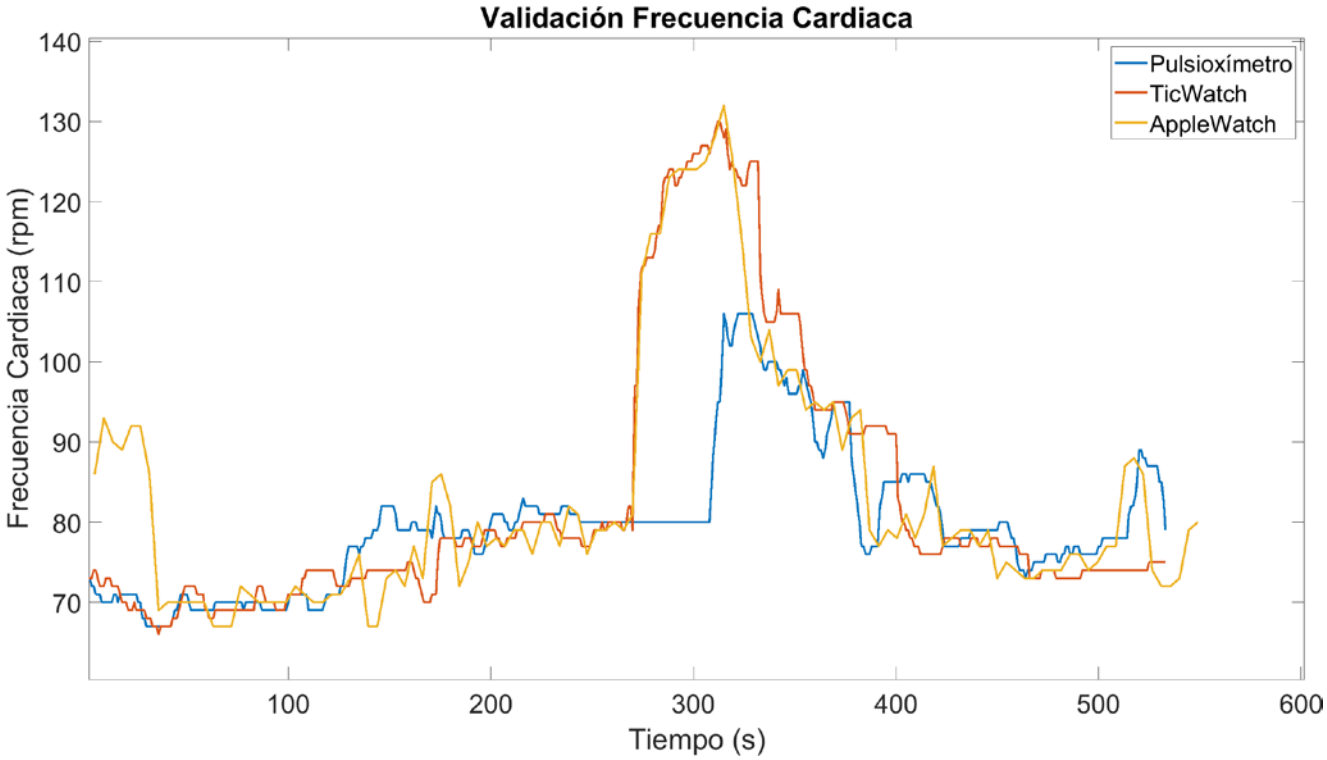

Figura 40. Valores obtenidos de la validación.

Como se ha comentado en el apartado 5.3, se ha descartado la idea de monitorizar la oxigenación en sangre debido a que no hay ningún estudio que afirme que en sedestación los valores vayan a cambiar de manera significativa. Para validar dicha hipótesis de partida, se ha realizado una prueba en la cual se monitoriza durante un período de tiempo la oxigenación de una persona en silla de ruedas. Para que esta prueba sea aun más real y se compruebe mejor si existen cambios, la monitorización se ha realizado en la calle (con el embaldosado de Bilbao

típico que se caracteriza por sus relieves), pasando por diferentes tramos, cuestas, zonas con baches y zonas de obras.

En la Figura 45 se diferencian las diferentes zonas por donde se ha realizado la monitorización. La primera de ellas se trata de una zona de cuesta con pendiente ascendente; la segunda, una zona de cuesta descendente; la tercera, una zona del embaldosado propio de Bilbao y la cuarta y última, una zona de obras. La monitorización se ha realizado durante aproximadamente 18 minutos y no se ha perdido la comunicación Bluetooth en ningún momento.

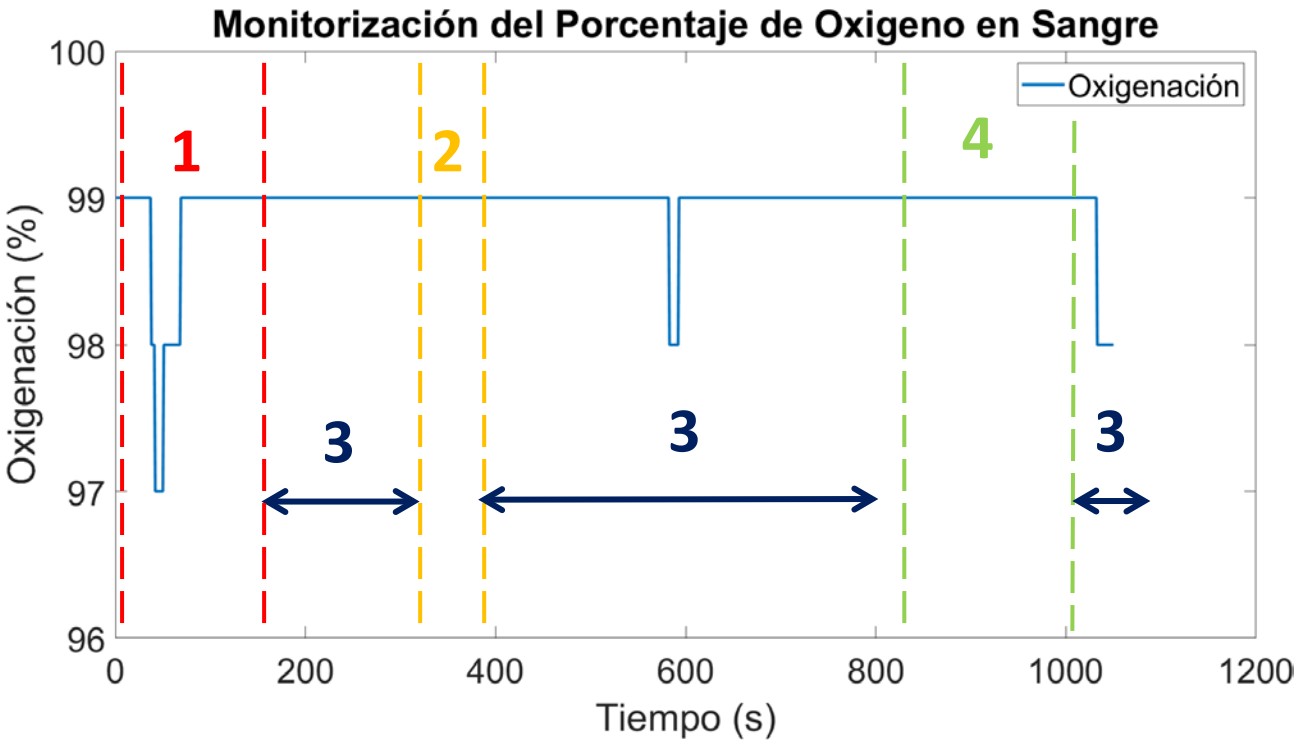

Figura 41. Monitorización de la oxigenación.

Los resultados obtenidos que se pueden ver en la Figura 45 muestran unos cambios muy poco significativos en los niveles de oxígeno en sangre, manteniéndose en todo momento unos niveles normales de saturación entre 95 y 100%. Por lo tanto, como ya se ha mencionado anteriormente, la saturación de oxígeno no se ve alterada en personas en sedestación y efectivamente es lo que se ve en la Figura 45. Por lo tanto, queda comprobado que no es necesario monitorizar la saturación de oxígeno en sangre, ya que no va a aportar datos relevantes para diagnosticar anomalías.

8. METODOLOGÍA

En este apartado se describe la metodología seguida durante la ejecución del TFM. Además de ello, se mostrará el diagrama de Gantt correspondiente a la planificación temporal del proyecto junto con los hitos a realizar.

8.1. DESCRIPCIÓN DE TAREAS

- **T1. Búsqueda de información.**

  - Duración: 1 semana.

  - Descripción: Realización de una búsqueda bibliográfica de los diferentes métodos que existen para la monitorización de variables fisiológicas. En función de la información obtenida, se ha determinado el alcance de este.

  - Hito: Informe del estado del arte o del estudio de la técnica de monitorización de variables fisiológicas.

  - Recursos utilizados: Ordenador con acceso a internet y suscripciones a bases de datos de investigaciones académicas.

- **T2. Análisis de alternativas.**

  - T2.1. Variables fisiológicas y sensores.

    - Duración: 1 semana.

    - Descripción: Seleccionar las variables fisiológicas más importantes que determinan el estado de una persona y realizar una búsqueda del estado de la tecnología en cuanto a los diferentes sensores y dispositivos que existen para hacer esas mediciones.

    - Hito: Informe del estado de la tecnología en cuanto a sensores y dispositivos de captura de señales fisiológicas.

    - Recursos utilizados: Ordenador con acceso a internet y suscripciones a bases de datos de investigaciones académicas.

  - T2.2. Búsqueda de alternativas Hardware y Software.

    - Duración: 1 semana.

- Descripción: Búsqueda de los posibles dispositivos a utilizar, junto con los programas software y sistemas de almacenamiento de la información.
- Hito: Informe de la selección de componentes y herramientas software para desarrollar el sistema de monitorización mediante móvil.
- Recursos utilizados: Ordenador con acceso a internet.

- **T3. Aprendizaje de la herramienta software.**
  - Duración: 1 semana.
  - Descripción: Instalación del software. Aprendizaje del uso del software mediante tutoriales y ejemplos de la página web de Android.
  - Recursos utilizados: Ordenador con acceso a internet. Android Studio.

- **T4. Programación del código.**
  - T4.1. Acceder al sensor y leer los datos del reloj.
    - Duración: 1 semana.
    - Descripción: Crear la aplicación para Wear Os, programar el código para acceder al sensor y obtener los datos de lectura del sensor.
    - Recursos utilizados: Ordenador con acceso a internet, Android Studio, reloj inteligente.
  - T4.2. Establecer la conexión Bluetooth.
    - Duración: 2 semanas.
    - Descripción: Crear aplicación para el móvil. Codificar tanto en la aplicación móvil como en la aplicación propietaria del reloj el código relativo a la conexión Bluetooth y lograr establecer una conexión entre los dispositivos.
    - Recursos utilizados: Ordenador con acceso a internet, Android Studio, reloj inteligente, móvil.
  - T4.3. Diseño de la interfaz de la aplicación móvil.
    - Duración: 2 semanas.
    - Descripción: Identificar los recursos necesarios que necesita tener la interfaz y diseñar la pantalla principal de la aplicación de móvil.
    - Recursos utilizados: Móvil, Ordenador, Android Studio.

- T4.4. Recoger y visualizar los datos en formato numérico.

  - Duración: 1 semana.

  - Descripción: Programar la interfaz de la aplicación desarrollada para que internamente el móvil reciba vía Bluetooth los datos leídos por el reloj y los monitorice en pantalla en tiempo real.

  -Recursos utilizados: Móvil, Ordenador, Android Studio, reloj inteligente.

- T4.5. Graficar los datos y guardarlos en una base de datos.

  - Duración: 1 semana.

  - Descripción: Diseñar y programar la estructura de los gráficos en los que se quiere visualizar los resultados. Generar la base de datos con las distintas tablas requeridas para guardar los datos que se reciben del reloj: 1) Tabla con nombre de usuario y fecha de registro, 2) Tabla con nombre de usuario y frecuencias obtenidas. Tras ello, guardar esos datos de manera ordenada.

  -Recursos utilizados: Móvil, Ordenador, Android Studio, Base de datos SQLite.

- T4.6. Pantalla de registro de usuario y fecha.

  - Duración: 1 semana.

  - Descripción: Diseñar la interfaz de la pantalla de nuevo usuario. Ser capaz de añadir un usuario y la fecha de registro en la base de datos anteriormente creada.

  -Recursos utilizados: Móvil, Ordenador, Android Studio, Base de datos SQLite.

- T4.7. Pantalla para búsqueda de los datos.

  - Duración: 1 semana.

  - Descripción: Diseñar la interfaz de la pantalla de búsqueda de datos y programar el código correspondiente a la búsqueda de los datos de un usuario accediendo a la información almacenada en la base de datos para luego presentarla en pantalla.

  -Recursos utilizados: Móvil, Ordenador, Android Studio, Base de datos SQLite.

- T4.8. Exportar datos vía email.

  - Duración: 1 semana.

  - Descripción: Programar el código que permita exportar los datos almacenados en la base de datos a formato CSV para luego enviarlos vía email.

- Hito: Entregable del código completo así como de aplicaciones completas desarrolladas.

-Recursos utilizados: Móvil, Ordenador, Android Studio, reloj inteligente, móvil.

- **T5. Pruebas de validación.**
  - T5.1. Puesta a punto de dispositivo de validación.

    - Duración: 1 semana.

    - Descripción: Descargar el software Arduino y descargar los códigos de la página oficial del lector de la frecuencia cardiaca. Verificar que los programas se han cargado adecuadamente y que el pulsioxímetro lee los datos de la frecuencia cardiaca y de la oxigenación.

    -Recursos utilizados: Arduino, placa Arduino y pulsioxímetro.
  - T5.2. Realización de pruebas experimentales.

    - Duración: 1 semana.

    - Descripción: Captura de datos continuada simulando una situación normal de este tipo de personas con movilidad reducida. Se realizaron ensayos durante X minutos en situación real de ruta por la calle en silla de ruedas para monitorizar las variables con el reloj inteligente al mismo tiempo que se monitoriza el pulsioxímetro.

    - Hito: Entregable del código en correcto funcionamiento (código depurado) tras la validación de las pruebas experimentales.

    - Recursos utilizados: Ordenador portátil, placa Arduino, pulsioxímetro, reloj inteligente, móvil, silla de ruedas motorizada.

- **T6. Fase final.**

  - Duración: 4 semanas.

  - Descripción: Se procede a redactar el informe de la Memoria Final de todo lo llevado a cabo en el Trabajo de Fin de Master. Adicionalmente, se documenta el código y se realiza un manual de usuario con el fin de explicar los pasos a seguir para realizar una monitorización recogido en el Anexo I.

-Hito: Entregable del documento Memoria Final junto con los Anexos correspondientes y el código documentado.

-Recursos utilizados: Microsoft Word, Microsoft Project, Lucidchart.

8.2. DIAGRAMA DE GANTT

En este apartado se lleva a cabo la planificación temporal mediante la tabla 5 donde se han indicado las tareas anteriormente descritas y los hitos establecidos.

En la Figura 45 se muestra el diagrama de Gantt en el que se incorpora la distribución temporal de las tareas y los hitos descritos anteriormente.

Fecha de comienzo: 1/02/2022

Fecha de finalización: 28/06/2022

Trascurso total del trabajo: 5 meses

Nombre de tarea	Comienzo	Fin	Predecesoras
Gantt Ane	**mar 01/02/22**	**mar 28/06/22**	
T1.Búsqueda de información	**mar 01/02/22**	**lun 07/02/22**	
Busqueda de información, definición de objetivos y alcance del trabajo	mar 01/02/22	lun 07/02/22	
Entrega: Alcance y objetivos	lun 07/02/22	lun 07/02/22	2
T2.Análisis de alternativas.	**mar 08/02/22**	**lun 14/02/22**	
Variables fisiologicas y sensores	mar 08/02/22	lun 14/02/22	2
Búsqueda de las alternativas actuales	mar 08/02/22	lun 14/02/22	2
Entrega: Estado del arte	lun 14/02/22	lun 14/02/22	6
T3.Familiarización con el software.	**mar 15/02/22**	**lun 21/02/22**	
Formación de Android Studio	mar 15/02/22	lun 21/02/22	6
T4.Programación del código	**mar 22/02/22**	**lun 04/04/22**	
Acceder al sensor y leer los datos del reloj	mar 22/02/22	lun 28/02/22	9
Establecer la conexión Bluetooth	mar 01/03/22	lun 14/03/22	11
Diseño de la interfaz de la aplicación móvil	mar 22/02/22	lun 07/03/22	9
Recoger y visualizar los datos en formato numérico	mar 01/03/22	lun 07/03/22	11
Graficar los datos y guardalos en una base de datos	mar 08/03/22	lun 14/03/22	14
Pantalla de registro de usuario y fecha	mar 15/03/22	lun 21/03/22	15
Pantalla para búsqueda de los datos	mar 22/03/22	lun 28/03/22	16
Exportar datos via email	mar 29/03/22	lun 04/04/22	17
Funcionamiento correcto	lun 04/04/22	lun 04/04/22	18
T5.Pruebas de validación	**lun 04/04/22**	**lun 18/04/22**	
Puesta a punto de dispositivo de validación	mar 05/04/22	lun 11/04/22	18
Realización de pruebas	mar 12/04/22	lun 18/04/22	21
Validación de las pruebas	lun 04/04/22	lun 04/04/22	18
T6. Fase final	**mar 08/02/22**	**mar 28/06/22**	**2**
Desarrollo de la memoria	mar 08/02/22	mar 28/06/22	1
Entrega final: Memoria completa	mar 28/06/22	mar 28/06/22	25

Tabla 6. Tabla de tareas e hitos.

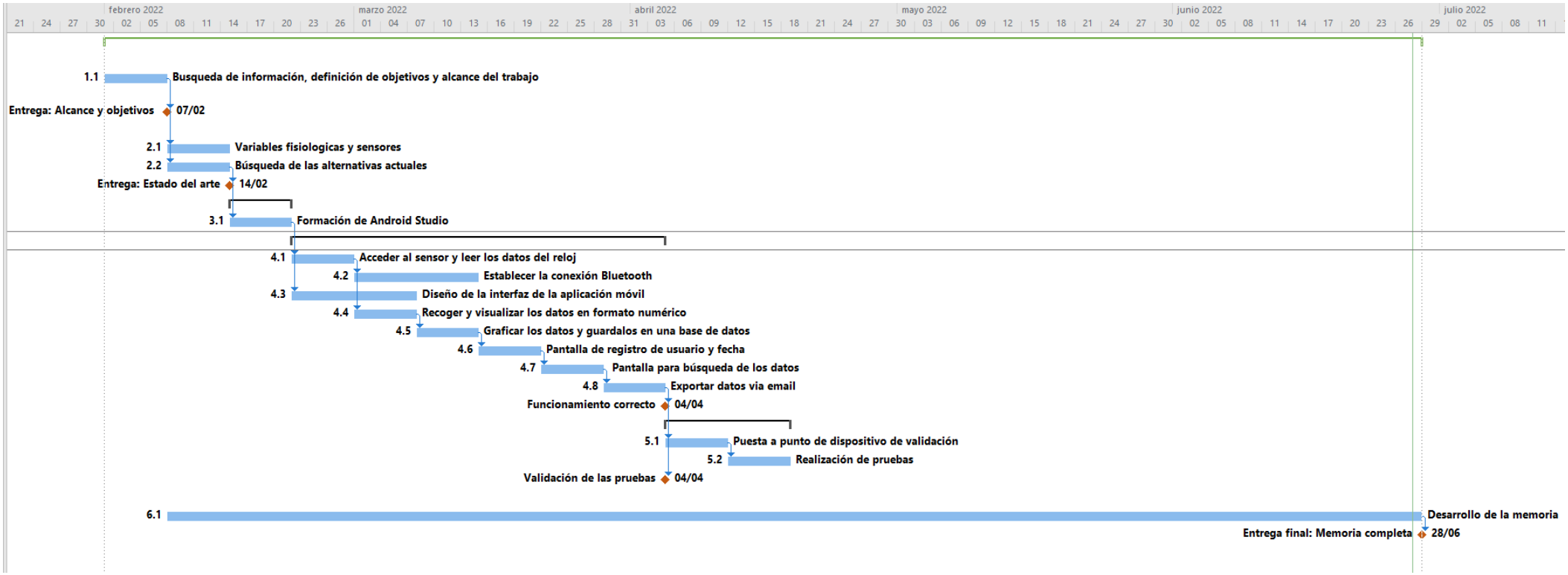

Figura 42. Diagrama de Gantt del TFM.

9. ASPECTOS ECONÓMICOS

En este apartado se hará una valoración de los gastos económicos que ha supuesto este TFM. Los gastos se presentarán en tres conceptos: en horas, amortizaciones y materiales.

- **Horas**

El coste por hora se ha establecido en función de las tareas y de la responsabilidad de cada persona.

HORAS INTERNAS			
Concepto	Horas Empleadas	Coste Unitario	Coste
Ingeniero Junior	600	30€/h	18.000€
Director de Proyecto	40	60€/h	2.400€
		TOTAL	**20.400€**

Tabla 7. Horas.

- **Inversiones: Amortización de los activos utilizados**

En este apartado se calculará el coste asociado a las herramientas no esenciales que se han utilizado para este TFM. Dentro de estas herramientas están la placa Arduino y el pulsioxímetro que se han utilizado para validar las pruebas, los programas software y el ordenador personal. Cabe mencionar que el software de Android es totalmente gratuito.

AMORTIZACIONES				
Inversión	**Precio Inicial**	**Vida útil**	**Utilización en el proyecto**	**Amortización**
Ordenador personal	1300€	4 años	4 meses	108,33€
Matlab	1980€	3 años	2 semanas	27,5€
Arduino	35€	3 años	3 semanas	0,73€

Pulsioxímetro	30€	3 años	2 semanas	0,42€
Placa E-Health	40€	3 años	2 semanas	0,56€
Excel	149€	1 año	1 mes	12,42€
			TOTAL	**149,96€**

Tabla 8. Amortizaciones.

- **Materiales**

En esta sección se calculará el coste relacionado con los materiales utilizados para la realización de la solución software.

Materiales	Unidades	Precio/unidad	Coste
Móvil	1	263€	263€
Reloj Inteligente	1	299,99€	299,99€
		TOTAL	**562,99€**

Tabla 9. Materiales.

- **Coste total del proyecto**

En esta sección de mostrará el extracto completo de los gastos. Se tendrán en cuenta todos los gastos anteriores calculados y se reservará un 10% del gasto total para posibles improvistos.

COSTE TOTAL	
Concepto	Coste
Mano de obra	20.400€
Amortizaciones	149,96€
Materiales	562,99€
TOTAL	21.112,95€

Imprevistos (10%)	2111,295€
TOTAL	**23.224,245€**

Tabla 10. Coste total del proyecto.

10. CONCLUSIONES

En este apartado se exponen las conclusiones extraídas del trabajo desarrollado en este TFM. Además, dado su ámbito investigador y la temática del mismo, este trabajo supone una base sólida para construir sistemas de monitorización remota que den respuesta a una población con movilidad reducida. De hecho, todo lo aprendido durante la realización de este trabajo será útil para incluir futuras ampliaciones, como se mencionarán en este apartado.

10.1. CONCLUSIONES

A la vista del incremento de la población con movilidad reducida existente hoy en día, agravada por la vida sedentaria y el envejecimiento, se están impulsando los avances en la telemedicina, donde se apuestan por sistemas de monitorización remota. En la ejecución de este TFM, se ha desarrollado, diseñado y validado un sistema de monitorización de variables fisiológicas, totalmente remoto, para las personas en sedestación.

Así, se ha diseñado y programado una solución software que permite capturar, monitorizar y guardar los signos vitales de las personas en tiempo real. Este desarrollo posibilita procesar posteriormente esos datos y generar un sistema inteligente de detección de anomalías en personas usuarias de sillas de ruedas.

Se ha verificado el correcto funcionamiento de las aplicaciones desarrolladas a través de pruebas de validación donde se han utilizado como gold-standard otros dos dispositivos para la captura de datos. Tras realizar la comparativa correspondiente, se ha confirmado la robustez y eficiencia del sistema desarrollado.

Para el desarrollo de la solución software se ha hecho uso del entorno de desarrollo de aplicaciones Android Studio. Este entorno ha permitido crear una solución software capaz de mantener en todo momento una conexión Bluetooth sólida, en la que no se ha registrado ningún tipo de pérdida de información, tanto recogida de desde los sensores como transferida a otro dispositivo.

En general, el dispositivo ha tenido una buena aceptación por parte del individuo al que se le ha monitorizado, sin causarle ningún tipo de molestia o incomodidad.

Adicionalmente, se han identificado posibles nuevos módulos que podrían incorporarse para aumentar la funcionalidad y el potencial de dicho sistema. Estos son:

- Añadir a la aplicación la monitorización de más variables fisiológicas (como podría ser la presión arterial).
- Diseñar un sistema de alerta si el estado de la persona está fuera del rango establecido por el personal sanitario como valores normales.
- Crear una plataforma web dirigida al colectivo sanitario donde puedan visualizar los estados de salud de las personas en función de los datos extraídos de las monitorizaciones.

10.2. TRABAJO FUTURO

Para terminar, se explicarán las líneas futuras de investigación a las que da lugar la realización de este trabajo.

Como se ha mencionado previamente, el objetivo principal de este TFM ha sido crear una solución software que permite monitorizar variables fisiológicas para personas con movilidad reducida. Hasta el momento, solo se han monitorizado una serie de constantes vitales, pero a futuro se pretende ampliar la aplicación con otro conjunto de sensores (como podrían ser sensores de temperatura corporal y de presión arterial), para complementar los datos recogidos por el sistema actual desarrollado.

Dado que este TFM forma parte de un proyecto de investigación más ambicioso, cuyo objetivo principal es monitorizar la sedestación en silla de ruedas y detectar anomalías en posturas de sedestación, el siguiente paso será la integración de los tres retos mencionados: 1) Monitorización postural del usuario, 2) Monitorización de los movimientos de la silla de ruedas, y 3) Monitorización de las variables fisiológicas del usuario.

Por tanto, el siguiente paso será la integración en el prototipo de la silla de ruedas del grupo ViSens del sistema de captura del movimiento de la silla (vibraciones y aceleraciones laterales), el dispositivo i-KuXin que permite la monitorización postural y el módulo de monitorización de variables fisiológicas aquí desarrollado. Tras la incorporación de los tres módulos, se realizarán

estudios con pacientes de la Asociación Feekor, que requieren la silla de ruedas en su día a día, para detectar situaciones en las que se puedan apreciar anomalías en la salud. Con los datos recogidos se creará una Base de Datos que a posteriori será útil para aplicar Técnicas de Machine Learning que servirán para desarrollar un sistema inteligente de detecciones de anomalías en la salud de las personas que avisará de manera oportuna al médico e inferirá previsiblemente en el Sistema Sanitario.

Este resultado está dirigido a dar respuesta al tercer objetivo de desarrollo sostenible ODS 3-Salud y Bienestar [35], que pretende garantizar una vida sana y promover el bienestar en todas las edades para el desarrollo sostenible. Para ello, se creará una plataforma sanitaria eHealth donde se recogerán los datos de las monitorizaciones. El concepto eHealth es la implementación de las Tecnologías de la Información y Comunicación (TIC) en el ámbito de la salud, en forma de herramientas que facilitan el diagnóstico, el tratamiento y la prevención entre otras [36]. Por ello, esta plataforma eHealth será una solución que facilitará el trabajo de los profesionales sanitarios y la comunicación con sus pacientes.

BIBLIOGRAFÍA

[1] «Ministerio de Sanidad - Portal Estadístico del SNS - Gasto sanitario público: millones de euros, porcentaje sobre el PIB y euros por habitante según los países de Unión Europea (UE-28)». https://www.sanidad.gob.es/estadEstudios/sanidadDatos/tablas/tabla30_1.htm (accedido 3 de junio de 2022).

[2] A. Crawford y H. Harris, «Cuidar a adultos con deterioro de la movilidad física», *Nursing (Ed. española)*, vol. 34, n.º 4, pp. 32-37, jul. 2017, doi: 10.1016/j.nursi.2017.07.010.

[3] M. de, «Gráfico 1. Personas con discapacidad incluidas en las declaraciones IRPF en el Territorio de Régimen Fiscal Común por sexo y grupo de edad. Ejercicio 2019.», p. 7.

[4] L. N. G. Madrigal, «Síndrome de inmovilización en el Adulto Mayor», p. 5.

[5] «Sedentarismo según grupos de edad y periodo.», *INE*. https://www.ine.es/jaxi/Datos.htm?path=/t00/mujeres_hombres/tablas_1/l0/&file=d06006.px#!tabs-grafico (accedido 25 de mayo de 2022).

[6] Q. He y E. O. Agu, «Towards sedentary lifestyle prevention: An autoregressive model for predicting sedentary behaviors», en *2016 10th International Symposium on Medical Information and Communication Technology (ISMICT)*, Worcester, MA, USA, mar. 2016, pp. 1-5. doi: 10.1109/ISMICT.2016.7498879.

[7] J. I. Arocha Rodulfo, «Sedentarism, a disease from xxi century», *Clínica e Investigación en Arteriosclerosis (English Edition)*, vol. 31, n.º 5, pp. 233-240, sep. 2019, doi: 10.1016/j.artere.2019.04.001.

[8] L. Yang *et al.*, «Trends in Sedentary Behavior Among the US Population, 2001-2016», *JAMA*, vol. 321, n.º 16, pp. 1587-1597, abr. 2019, doi: 10.1001/jama.2019.3636.

[9] E. Chiauzzi, C. Rodarte, y P. DasMahapatra, «Patient-centered activity monitoring in the self-management of chronic health conditions», *BMC Medicine*, vol. 13, n.º 1, p. 77, abr. 2015, doi: 10.1186/s12916-015-0319-2.

[10] D. Cardona Arango y E. Peláez, «Envejecimiento poblacional en el siglo XXI: Oportunidades, retos y preocupaciones», *Revista Salud Uninorte*, vol. 28, n.º 2, pp. 335-348, dic. 2012.

[11] H. Ramos-Morillo, F. Maciá Pérez, y D. Marcos-Jorquera, «Redes Inalámbricas de Sensores Inteligentes. Aplicación a la Monitorización de Variables Fisiológicas», abr. 2022.

[12] S. Honda, H. Hara, T. Arie, S. Akita, y K. Takei, «A wearable, flexible sensor for real-time, home monitoring of sleep apnea», *iScience*, vol. 25, n.º 4, p. 104163, abr. 2022, doi: 10.1016/j.isci.2022.104163.

[13] N. Xiao, W. Yu, y X. Han, «Wearable heart rate monitoring intelligent sports bracelet based on Internet of things», *Measurement*, vol. 164, p. 108102, nov. 2020, doi: 10.1016/j.measurement.2020.108102.

[14] L. Yang, Y. Ge, W. Li, W. Rao, y W. Shen, «A home mobile healthcare system for wheelchair users», en *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Hsinchu, Taiwan, may 2014, pp. 609-614. doi: 10.1109/CSCWD.2014.6846914.

[15] S. P. Penagos, L. D. Salazar, y F. E. Vera, «Control de signos vitales», p. 9.

[16] D. Evans, B. Hodgkinson, y J. Berry, «Vital signs in hospital patients: a systematic review», *International Journal of Nursing Studies*, vol. 38, n.º 6, pp. 643-650, dic. 2001, doi: 10.1016/S0020-7489(00)00119-X.

[17] J. V. González, O. A. V. Arenas, y V. V. González, «Semiología de los signos vitales: Una mirada novedosa a un problema vigente», *Archivos de Medicina*, p. 21, 2012.

[18] «2014-05-vitalsigns.pdf».

[19] C. Rotariu, R. G. Bozomitu, A. Pasarica, D. Arotaritei, y H. Costin, «Medical system based on wireleless sensors for real time remote monitoring of people with disabilities», en *2017 E-Health and Bioengineering Conference (EHB)*, Sinaia, Romania, jun. 2017, pp. 753-756. doi: 10.1109/EHB.2017.7995533.

[20] B. F. Rn, S. L. Rn, y M. Sockrider, «Oximetría de pulso», p. 2.

[21] «Pulsioximetría: Prueba de laboratorio de MedlinePlus». https://medlineplus.gov/spanish/pruebas-de-laboratorio/pulsioximetria/ (accedido 31 de mayo de 2022).

[22] Y. Sekiguchi, L. N. Belval, R. L. Stearns, D. J. Casa, y Y. Hosokawa, «MONITOREO DE LA TEMPERATURA CORPORAL INTERNA», vol. 29, n.º 192, p. 5, 2019.

[23] «Saturación de oxígeno: qué es y valores normales», *Tua Saúde*. https://www.tuasaude.com/es/saturacion-de-oxigeno/ (accedido 17 de junio de 2022).

[24] D. Vanegas-Cadavid, Z. Valderrama-Barbosa, y L. Ibatá-Bernal, «Experiencia clínica en monitorización cardiaca extendida con el sistema inalámbrico satelital tipo SEEQ», *Revista Colombiana de Cardiología*, vol. 25, n.º 3, pp. 176-184, may 2018, doi: 10.1016/j.rccar.2017.09.003.

[25] «Mejores Monitores de Frecuencia Cardíaca | Top 5», 9 de marzo de 2021. https://elultimotriatleta.com/mejores-monitores-frecuencia-cardiaca/ (accedido 9 de mayo de 2022).

[26] «Aprenda acerca de los monitores cardíacos implantables | Kaiser Permanente». https://espanol.kaiserpermanente.org/es/health-wellness/health-encyclopedia/he.aprenda-acerca-de-los-monitores-card%C3%ADacos-implantables.abs1276 (accedido 9 de mayo de 2022).

[27] D. L. Serrano, C. P. Fuster, y F. M. Meoro, «DISEÑO E IMPLEMENTACIÓN DE UN PULSIOXÍMETRO», p. 59.

[28] «Invierno y temperatura corporal en silla de ruedas», *Sunrise Medical*. https://www.sunrisemedical.es/blog/mantener-temperatura-corporal-invierno (accedido 26 de mayo de 2022).

[29] M. Q. Gorgas, «SISTEMAS DE MEDICIÓN DE TEMPERATURAS», p. 7.

[30] P.-Y. Cresson, C. Ricard, L. Dubois, S. Vaucher, T. Lasri, y J. Pribetich, «Temperature Measurement by Microwave Radiometry», en *2008 IEEE Instrumentation and Measurement Technology Conference*, Victoria, BC, Canada, may 2008, pp. 1344-1349. doi: 10.1109/IMTC.2008.4547251.

[31] «Discapacidad y salud». https://www.who.int/es/news-room/fact-sheets/detail/disability-and-health (accedido 3 de junio de 2022).

[32] «Investigación Biomédica - Temperatura Corporal - Dalcame: Grupo de». https://www.dalcame.com/tc.html#.Yo-iBShBy01 (accedido 26 de mayo de 2022).

[33] «TicWatch Pro 3 GPS smartwatch - Go Beyond Limits.» http://www.mobvoi.com/us/pages/ticwatchpro3gps (accedido 22 de junio de 2022).

[34] «Xiaomi 11 Lite 5G NE | Xiaomi España | Mi.com», *Xiaomi España*. https://www.mi.com/es/xiaomi-11-lite-5g-ne/ (accedido 22 de junio de 2022).

[35] «Salud», *Desarrollo Sostenible*. https://www.un.org/sustainabledevelopment/es/health/ (accedido 3 de junio de 2022).

[36] «eHealth, un mundo nuevo para una salud más personalizada», *Asepeyo*, 23 de enero de 2020. https://www.asepeyo.es/blog/gestion-y-rrhh/ehealth-un-mundo-nuevo-para-una-salud-mas-personalizada/ (accedido 3 de junio de 2022).

ANEXO I: MANUAL DE USUARIO

En este ANEXO se explican todos los pasos que hay que seguir para establecer la conexión Bluetooth, la visualización de datos y el guardado de datos en la base de datos.

1. Seleccionar el icono de la aplicación del móvil.

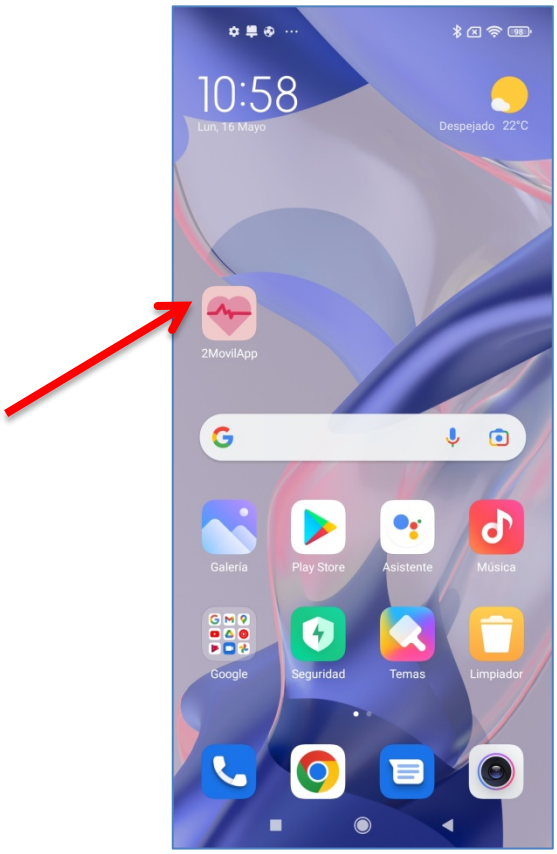

Figura 43. Menú principal del móvil.

2. Una vez abierta la aplicación se muestra una pantalla para añadir un nuevo Identificador del usuario. Añadir el usuario y presionar el botón guardar. A continuación pasar a la siguiente pantalla presionando el botón siguiente.

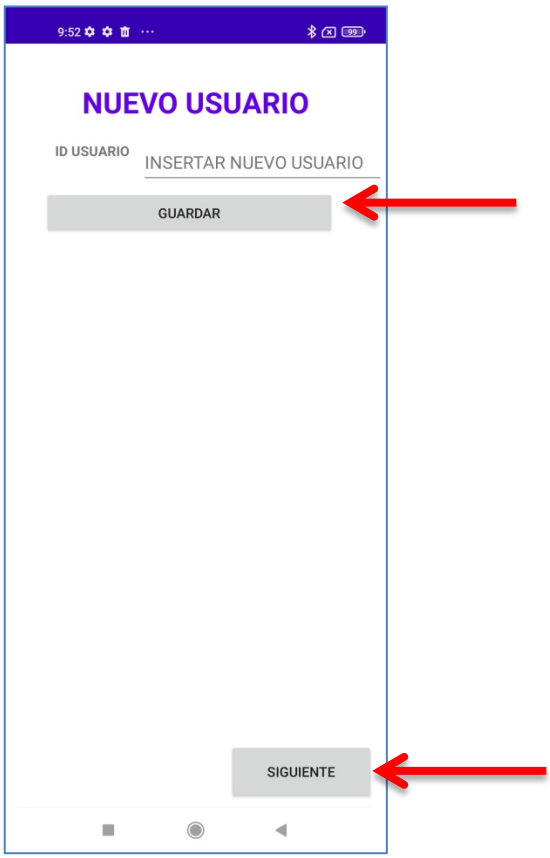

Figura 44. Pantalla de Nuevo Usuario.

3. Encender el Bluetooth presionando el botón ON/OFF.

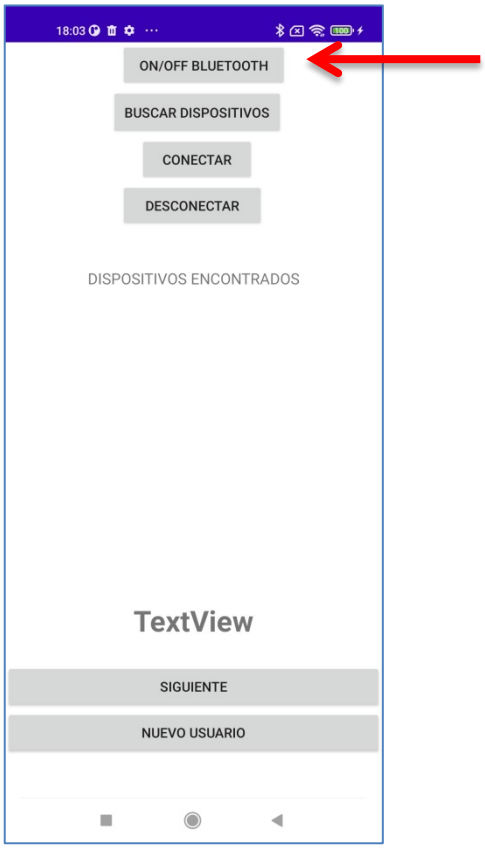

Figura 45. Pantalla de visualización de la frecuencia cardiaca.

4. Abrir la aplicación del reloj, se abrirá automáticamente una pantalla pidiendo permiso para que el reloj sea visible para otros dispositivos. Aceptar las condiciones.

Figura 46. Menú de aplicaciones del reloj.

Figura 47. Pantalla de aceptación de las condiciones.

5. Encender el Bluetooth presionando el botón ON/OFF.

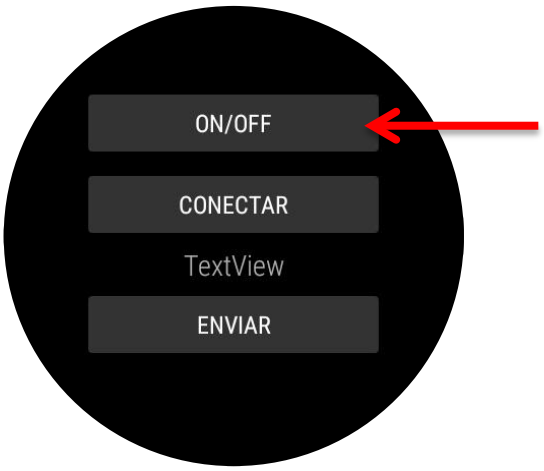

Figura 48. Pantalla principal de la aplicación del reloj.

6. En la aplicación móvil presionar el botón BUSCAR DISPOSITIVOS. Acto seguido se imprimirá una lista de los dispositivos Bluetooth encontrados. Seleccionar de la lista el TicWatch Pro 3 GPS 0420.

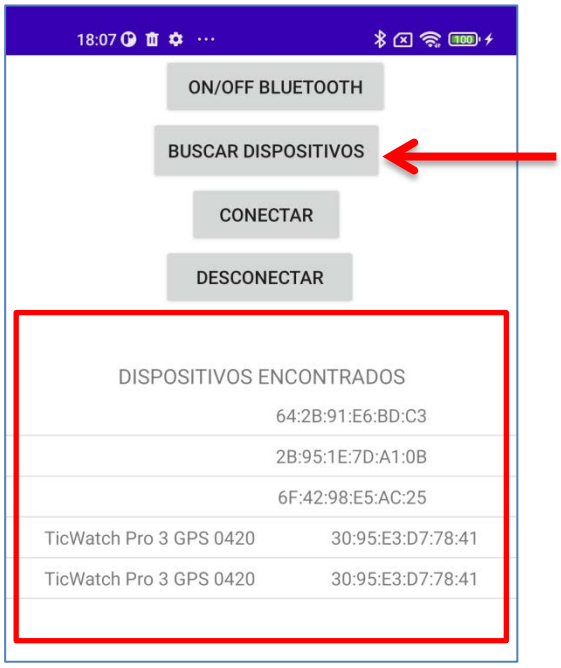

Figura 49. ListView de los dispositivos encontrados.

7. Presionar en el reloj el botón de CONECTAR y después presionar en el botón CONECTAR de la aplicación móvil. **IMPORTANTE: Presionar los botones en ese orden, primero en el reloj y después en el móvil, si no la conexión no se realizará con éxito.**

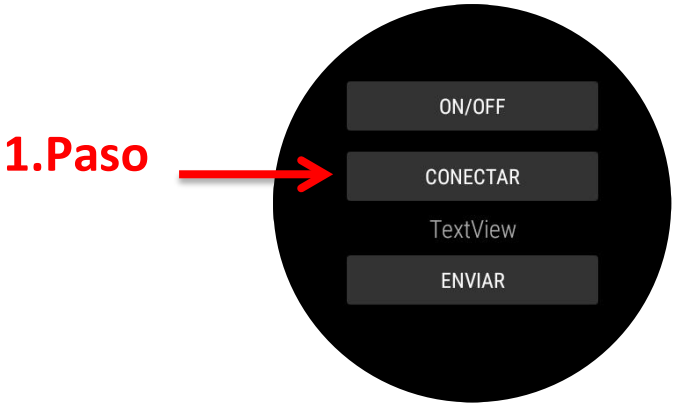

Figura 50. Pantalla del reloj.

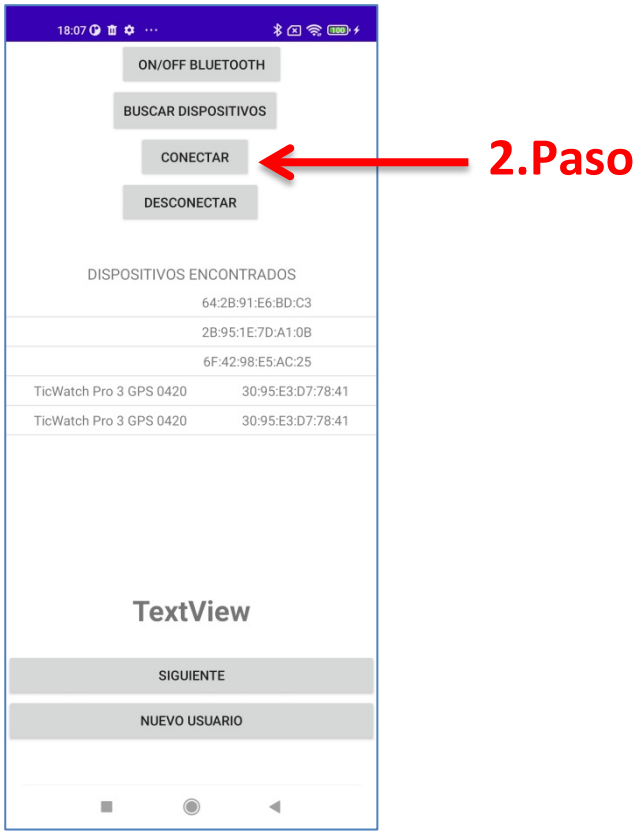

Figura 51. Pantalla de visualización de datos.

8. Presionar el botón del reloj ENVIAR para comenzar a transferir y visualizar los datos. Automáticamente se visualizará tanto en el reloj como en la pantalla principal del móvil la frecuencia cardiaca.

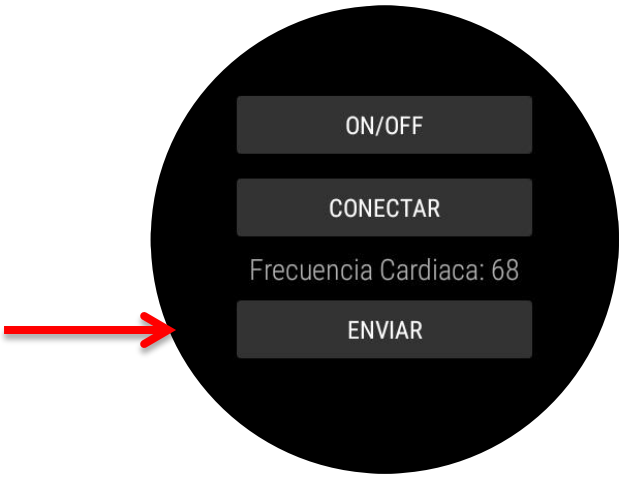

Figura 52. Visualización de datos en pantalla.

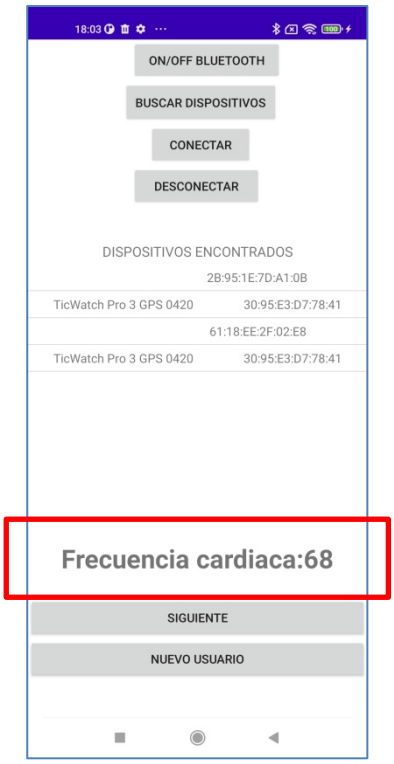

Figura 53. Visualización de los datos en el móvil.

9. Para parar el envío de datos presionar el botón DESCONECTAR.

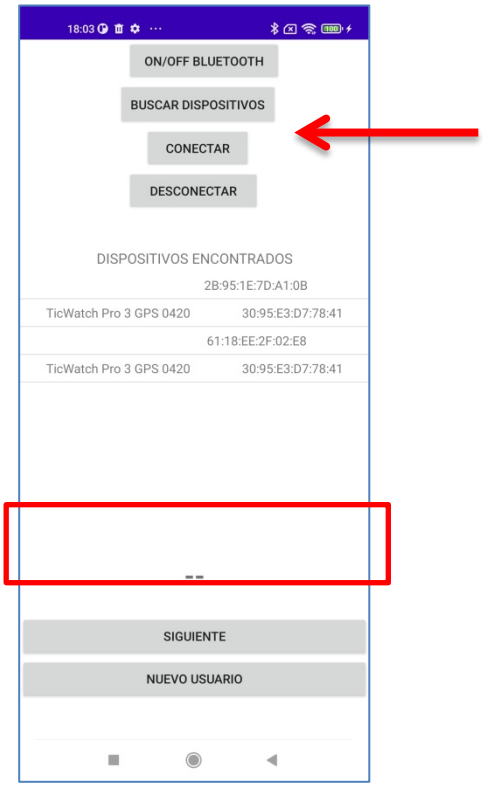

Figura 54. Paso para desconectar la conexión.

10. Para visualizar los datos en una gráfica, seleccionar el botón siguiente de la pantalla principal. En la nueva pantalla, presionar el botón MOSTRAR DATOS.

Figura 55. Paso para pasar a la siguiente pantalla.

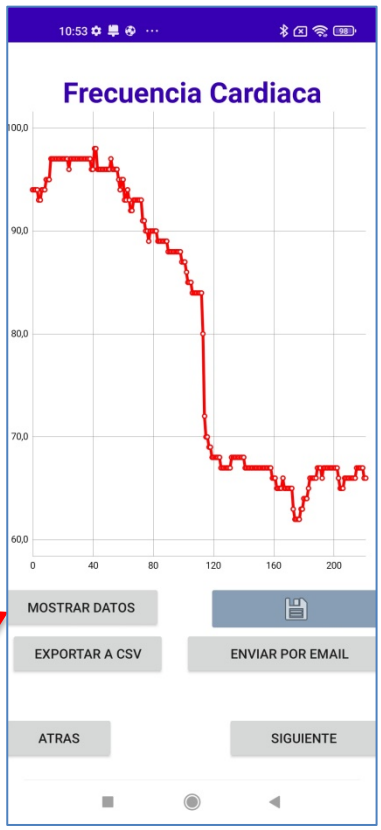

Figura 56. Pantalla de visualización de los datos en formato gráfico.

11. En esa misma pantalla, seleccionando el botón con la imagen de disco duro se guardan los datos recogidos en la base de datos. Si se presiona el botón EXPORTAR A CSV se crea un archivo de tipo con los datos recogidos. Para enviar ese archivo por email presionar el botón ENVIAR POR EMAIL.

Figura 57. Paso para guardar los datos.

12. Para realizar una búsqueda de la base de datos seleccionar el botón SIGUIENTE de la pantalla donde se grafican los datos. En la nueva pantalla introducir el nombre de usuario que se quiera buscar y presionar el botón BUSCAR. Para volver a la pantalla principal presionar el botón ATRÁS.

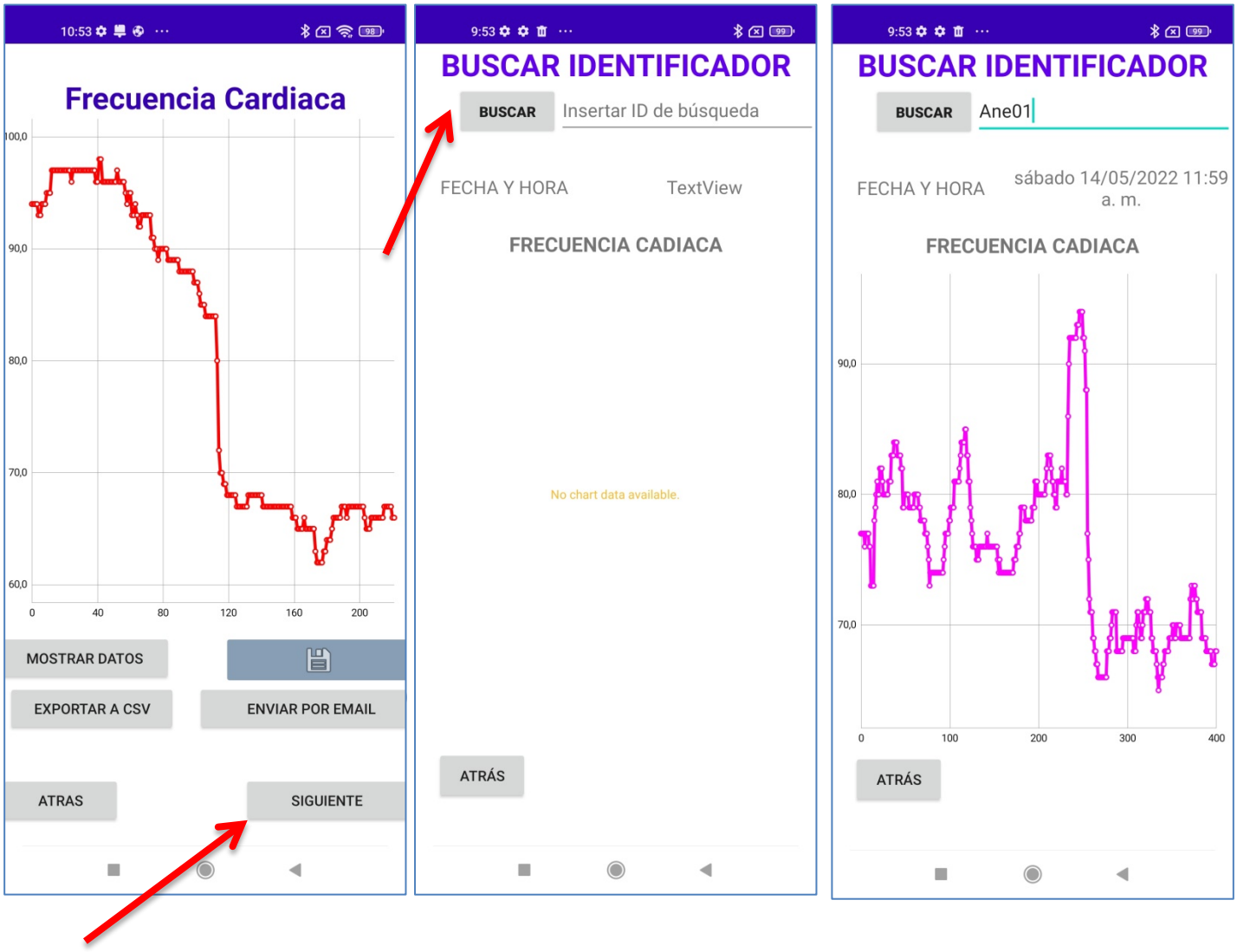

Figura 58. Pasos a seguir para realizar una búsqueda.

CÓDIGO DE LA APLICACIÓN DEL RELOJ

MainActivity

```
public class MainActivity extends Activity implements
SensorEventListener {
    private static final String TAG = "MainActivity";
    private static final UUID MY_UUID_INSECURE =
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    public static String msg = null;

    public ArrayList<BluetoothDevice> mBTDevices = new ArrayList<>();
    public ArrayAdapter<String> mBTArrayAdapter;
    public static ArrayList marrayList = new ArrayList(10000);

    BluetoothAdapter mBluetoothAdapter;
    BluetoothDevice mBTDevice ;
    SensorManager mSensorManager;
    Sensor mHeartRateSensor;

    public static int heartrate;
    Boolean sensorregistered = false;

    Button btnStartConnection;
    Button btnEnviar;
    TextView etSend;
    boolean bestado= false;
    BluetoothConnectionService mBluetoothConnection;

/************************************ BROADCASTS
****************************************************/
    // Create a BroadcastReceiver for ACTION_FOUND
    private final BroadcastReceiver mBroadcastReceiver1 = new
BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            // When discovery finds a device
            if (action.equals(mBluetoothAdapter.ACTION_STATE_CHANGED))
{
                final int state =
intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
mBluetoothAdapter.ERROR);

                switch(state){
                    case BluetoothAdapter.STATE_OFF:
                        Log.d(TAG, "onReceive: STATE OFF");
                        break;
                    case BluetoothAdapter.STATE_TURNING_OFF:
                        Log.d(TAG, "mBroadcastReceiver1: STATE TURNING
OFF");
                        break;
                    case BluetoothAdapter.STATE_ON:
                        Log.d(TAG, "mBroadcastReceiver1: STATE ON");
                        break;
                    case BluetoothAdapter.STATE_TURNING_ON:
```

```
                        Log.d(TAG, "mBroadcastReceiver1: STATE TURNING
ON");
                        break;
                }
            }
        }
    };

    /**
     * Broadcast Receiver for changes made to bluetooth states such
as:
     * 1) Discoverability mode on/off or expire.
     */
    private final BroadcastReceiver mBroadcastReceiver2 = new
BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();

            if
(action.equals(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)) {

                int mode =
intent.getIntExtra(BluetoothAdapter.EXTRA_SCAN_MODE,
BluetoothAdapter.ERROR);

                switch (mode) {
                    //Device is in Discoverable Mode
                    case
BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
                        Log.d(TAG, "mBroadcastReceiver2:
Discoverability Enabled.");
                        break;
                    //Device not in discoverable mode
                    case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
                        Log.d(TAG, "mBroadcastReceiver2:
Discoverability Disabled. Able to receive connections.");
                        break;
                    case BluetoothAdapter.SCAN_MODE_NONE:
                        Log.d(TAG, "mBroadcastReceiver2:
Discoverability Disabled. Not able to receive connections.");
                        break;
                    case BluetoothAdapter.STATE_CONNECTING:
                        Log.d(TAG, "mBroadcastReceiver2:
Connecting....");
                        break;
                    case BluetoothAdapter.STATE_CONNECTED:
                        Log.d(TAG, "mBroadcastReceiver2: Connected.");
                        break;
                }

            }
        }
    };
```

```
    private BroadcastReceiver mBroadcastReceiver3 = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            Log.d(TAG, "onReceive: ACTION FOUND.");

            if (action.equals(BluetoothDevice.ACTION_FOUND)) {
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                mBTDevices.add(device);

                Log.d(TAG, "onReceive: " + device.getName() + ": " +
device.getAddress());


            }
        }
    };

    /**
     * Broadcast Receiver that detects bond state changes (Pairing
status changes)
     */
    private final BroadcastReceiver mBroadcastReceiver4 = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();


if(action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)){
                BluetoothDevice mDevice =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                //3 cases:
                //case1: bonded already
                if (mDevice.getBondState() ==
BluetoothDevice.BOND_BONDED){
                    Log.d(TAG, "BroadcastReceiver: BOND_BONDED.");
                    //inside BroadcastReceiver4
                    mBTDevice = mDevice;
                }
                //case2: creating a bone
                if (mDevice.getBondState() ==
BluetoothDevice.BOND_BONDING) {
                    Log.d(TAG, "BroadcastReceiver: BOND_BONDING.");
                }
                //case3: breaking a bond
                if (mDevice.getBondState() ==
BluetoothDevice.BOND_NONE) {
                    Log.d(TAG, "BroadcastReceiver: BOND_NONE.");
                }
            }
        }
    };
```

```
    @Override
    protected void onDestroy() {
        Log.d(TAG, "onDestroy: called.");
        super.onDestroy();
        unregisterReceiver(mBroadcastReceiver1);
        unregisterReceiver(mBroadcastReceiver2);
        unregisterReceiver(mBroadcastReceiver3);
        unregisterReceiver(mBroadcastReceiver4);
        //mBluetoothAdapter.cancelDiscovery();
    }
/********************************** ON CREATE
*********************************************/
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnONOFF = (Button) findViewById(R.id.btnONOFF);
        mBTDevices = new ArrayList<>();
        btnStartConnection = (Button)
findViewById(R.id.btnStartConnection);
        btnEnviar = findViewById(R.id.btnSend);
        etSend = findViewById(R.id.idHR);

        IntentFilter filter = new
IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
        registerReceiver(mBroadcastReceiver4, filter);

        Intent discoverableIntent =
                new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATI
ON, 300);
        startActivity(discoverableIntent);

        // Keep the Wear screen always on (for testing only!)

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        Log.d(TAG, "BluetoothAdapter: "+ mBluetoothAdapter);
        //checkBTPermissions();

        btnONOFF.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                boolean action ;
                Log.d(TAG, "onClick: enabling/disabling bluetooth.");
                enableDisableBT();
                checkBTPermissions();

                //Direccion MAC del XIAOMI 0C:C6:FD:C1:FF:71
```

```
            }
        });

        btnStartConnection.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // startConnection();

                mBTDevice =
mBluetoothAdapter.getRemoteDevice("0C:C6:FD:C1:FF:71");
                mBluetoothConnection = new
BluetoothConnectionService(MainActivity.this);
                mBluetoothConnection.start();
                // Toast.makeText(this, "Start
server",Toast.LENGTH_SHORT).show();
            }
        });

        btnEnviar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
               getHeartRate();
            }

        });


    }
/****************************************** METHODS
*****************************************/
    //create method for starting connection
//***remember the conncction will fail and app will crash if you
haven't paired first
    public void startConnection(){

        startBTConnection(mBTDevice,MY_UUID_INSECURE);
    }

    /**
     * starting chat service method
     */
    public void startBTConnection(BluetoothDevice device, UUID uuid){
        Log.d(TAG, "startBTConnection: Initializing RFCOM Bluetooth
Connection.");

        mBluetoothConnection.startClient(device,uuid);
    }

    // Conectar y desconectar Bluetooth
    public void enableDisableBT(){
        if(mBluetoothAdapter == null){
            Log.d(TAG, "enableDisableBT: Does not have BT
```

```
capabilities.");
        }
        if(!mBluetoothAdapter.isEnabled()){
            Log.d(TAG, "enableDisableBT: enabling BT.");
            Intent enableBTIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enableBTIntent);

            IntentFilter BTIntent = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(mBroadcastReceiver1, BTIntent);
        }
        if(mBluetoothAdapter.isEnabled()){
            Log.d(TAG, "enableDisableBT: disabling BT.");
            mBluetoothAdapter.disable();

            IntentFilter BTIntent = new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(mBroadcastReceiver1, BTIntent);
        }

    }

   //Hacer el dispositivo visible para los demas dispositivos
    public void btnEnableDisable_Discoverable(View view) {
        Log.d(TAG, "btnEnableDisable_Discoverable: Making device
discoverable for 300 seconds.");

        Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATI
ON, 300);
        startActivity(discoverableIntent);

        IntentFilter intentFilter = new
IntentFilter(mBluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
        registerReceiver(mBroadcastReceiver2,intentFilter);

    }

    public void btnDiscover(View view) {
        Log.d(TAG, "btnDiscover: Looking for unpaired devices.");

        if(mBluetoothAdapter.isDiscovering()){
            mBluetoothAdapter.cancelDiscovery();
            Log.d(TAG, "btnDiscover: Canceling discovery.");

            //check BT permissions in manifest
            checkBTPermissions();

            mBluetoothAdapter.startDiscovery();
            IntentFilter discoverDevicesIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(mBroadcastReceiver3,
```

```
discoverDevicesIntent);
        }
        if(!mBluetoothAdapter.isDiscovering()){

            //check BT permissions in manifest
            checkBTPermissions();

            mBluetoothAdapter.startDiscovery();
            IntentFilter discoverDevicesIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(mBroadcastReceiver3,
discoverDevicesIntent);
        }
    }

    /**
     * This method is required for all devices running API23+
     * Android must programmatically check the permissions for
bluetooth. Putting the proper permissions
     * in the manifest is not enough.
     *
     * NOTE: This will only execute on versions > LOLLIPOP because it
is not needed otherwise.
     */
    private void checkBTPermissions() {
        if(Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP){
            int permissionCheck =
this.checkSelfPermission("Manifest.permission.ACCESS_FINE_LOCATION");
            permissionCheck +=
this.checkSelfPermission("Manifest.permission.ACCESS_COARSE_LOCATION")
;
            if (permissionCheck != 0) {

                this.requestPermissions(new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,

Manifest.permission.ACCESS_COARSE_LOCATION}, 1001); //Any number
            }
        }else{
            Log.d(TAG, "checkBTPermissions: No need to check
permissions. SDK version < LOLLIPOP.");
        }
    }

    /******************************* HEART RATE METHODS
*************************************/
    private void getHeartRate() {
        mSensorManager =
((SensorManager)getSystemService(SENSOR_SERVICE));
        if (mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE) !=
null){
            mHeartRateSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
            sensorregistered = mSensorManager.registerListener(this,
mHeartRateSensor,
```

```
                    SensorManager.SENSOR_DELAY_NORMAL);}
        else {
            Log.d(TAG,"Sensor no disponible");
        }
    }
    private void stopmeasure() {
        mSensorManager.unregisterListener(this);
    }
    @Override
    public void onAccuracyChanged (Sensor sensor, int accuracy) {
        Log.d(TAG, "onAccuracyChanged - accuracy: " + accuracy);
    }
    @Override
    public void onSensorChanged (SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_HEART_RATE) {
            heartrate = (int) event.values[0];
            msg = "Frecuencia Cardiaca: " + (int) event.values[0];
            etSend.setText(msg);
           //Lamada al metodo para que envie los datos al dispositivo
remoto
            mBluetoothConnection.write(heartrate);
            Log.d(TAG, "Dato enviado:" + heartrate);
        } else
            Log.d(TAG, "Sensor de frencuencia cardiaca no
reconocido");
    }

    private final BroadcastReceiver mBroadcastReceiver5 = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();


if(action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)){
                BluetoothDevice mDevice =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                //3 cases:
                //case1: bonded already
                if (mDevice.getBondState() ==
BluetoothDevice.BOND_BONDED){
                    Log.d(TAG, "BroadcastReceiver: BOND_BONDED.");
                    //inside BroadcastReceiver4
                    mBTDevice = mDevice;
                }
                //case2: creating a bone
                if (mDevice.getBondState() ==
BluetoothDevice.BOND_BONDING) {
                    Log.d(TAG, "BroadcastReceiver: BOND_BONDING.");
                }
                //case3: breaking a bond
                if (mDevice.getBondState() ==
BluetoothDevice.BOND_NONE) {
                    Log.d(TAG, "BroadcastReceiver: BOND_NONE.");
                }
```

```
            }
        }
    };
}
```

BluetoothConnectionService

```
public class BluetoothConnectionService {
    private static final String TAG = "BluetoothConnectionServ";

    private static final String appName = "MYAPP";

    private static final UUID MY_UUID_INSECURE =
            UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    private final BluetoothAdapter mBluetoothAdapter;
    Context mContext;
    public static Boolean bestado = false;

    private AcceptThread mInsecureAcceptThread;

    private ConnectThread mConnectThread;
    private BluetoothDevice mmDevice;
    private UUID deviceUUID;
    ProgressDialog mProgressDialog;

    private ConnectedThread mConnectedThread;


    public BluetoothConnectionService(Context context) {
        mContext = context;
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        start();
    }


    /**
     * This thread runs while listening for incoming connections. It
behaves
     * like a server-side client. It runs until a connection is
accepted
     * (or until cancelled).
     */
    private class AcceptThread extends Thread {

        // The local server socket
        private final BluetoothServerSocket mmServerSocket;

        public AcceptThread(){
            BluetoothServerSocket tmp = null;

            // Create a new listening server socket
            try{
                tmp =
mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(appName,
```

```
MY_UUID_INSECURE);

                Log.d(TAG, "AcceptThread: Setting up Server using: " +
MY_UUID_INSECURE);
            }catch (IOException e){
                Log.e(TAG, "AcceptThread: IOException: " +
e.getMessage() );
            }

            mmServerSocket = tmp;
        }

        public void run(){
            Log.d(TAG, "run: AcceptThread Running.");

            BluetoothSocket socket = null;

            try{
                // This is a blocking call and will only return on a
                // successful connection or an exception
                Log.d(TAG, "run: RFCOM server socket start.....");

                socket = mmServerSocket.accept();

                Log.d(TAG, "run: RFCOM server socket accepted
connection.");

            }catch (IOException e){
                Log.e(TAG, "AcceptThread: IOException: " +
e.getMessage() );
            }

            //talk about this is in the 3rd
            if(socket != null){
                connected(socket,mmDevice);
            }

            Log.i(TAG, "END mAcceptThread ");
        }

        public void cancel() {
            Log.d(TAG, "cancel: Canceling AcceptThread.");
            try {
                mmServerSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "cancel: Close of AcceptThread ServerSocket
failed. " + e.getMessage() );
            }
        }

    }

    /**
     * This thread runs while attempting to make an outgoing
connection
```

```
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private BluetoothSocket mmSocket;

        public ConnectThread(BluetoothDevice device, UUID uuid) {
            Log.d(TAG, "ConnectThread: started.");
            mmDevice = device;
            deviceUUID = uuid;
        }

        public void run(){
            BluetoothSocket tmp = null;
            Log.i(TAG, "RUN mConnectThread ");

            // Get a BluetoothSocket for a connection with the
            // given BluetoothDevice
            try {
                Log.d(TAG, "ConnectThread: Trying to create
InsecureRfcommSocket using UUID: "
                        +MY_UUID_INSECURE );
                tmp =
mmDevice.createRfcommSocketToServiceRecord(deviceUUID);
            } catch (IOException e) {
                Log.e(TAG, "ConnectThread: Could not create
InsecureRfcommSocket " + e.getMessage());
            }

            mmSocket = tmp;

            // Always cancel discovery because it will slow down a
connection
            mBluetoothAdapter.cancelDiscovery();

            // Make a connection to the BluetoothSocket

            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                mmSocket.connect();

                Log.d(TAG, "run: ConnectThread connected.");
            } catch (IOException e) {
                // Close the socket
                try {
                    mmSocket.close();
                    Log.d(TAG, "run: Closed Socket.");
                } catch (IOException e1) {
                    Log.e(TAG, "mConnectThread: run: Unable to close
connection in socket " + e1.getMessage());
                }
                Log.d(TAG, "run: ConnectThread: Could not connect to
UUID: " + MY_UUID_INSECURE );
            }
```

```
            //will talk about this in the 3rd video
            connected(mmSocket,mmDevice);
        }
        public void cancel() {
            try {
                Log.d(TAG, "cancel: Closing Client Socket.");
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "cancel: close() of mmSocket in
Connectthread failed. " + e.getMessage());
            }
        }
    }



    /**
     * Start the chat service. Specifically start AcceptThread to
begin a
     * session in listening (server) mode. Called by the Activity
onResume()
     */
    public synchronized void start() {
        Log.d(TAG, "start");

        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mInsecureAcceptThread == null) {
            mInsecureAcceptThread = new AcceptThread();
            mInsecureAcceptThread.start();
        }
    }

    /**
     AcceptThread starts and sits waiting for a connection.
     Then ConnectThread starts and attempts to make a connection with
the other devices AcceptThread.
     **/

    public void startClient(BluetoothDevice device,UUID uuid){
        Log.d(TAG, "startClient: Started.");

        //initprogress dialog
        mProgressDialog = ProgressDialog.show(mContext,"Connecting
Bluetooth"
                ,"Please Wait...",true);

        mConnectThread = new ConnectThread(device, uuid);
        mConnectThread.start();
    }
    //Innecesario, ya esta el metodo start() que inicia el servidor.
```

```
    public void startServer(){
        //initprogress dialog
        mProgressDialog = ProgressDialog.show(mContext,"Connecting
Bluetooth Server"
                ,"Please Wait...",true);
        Log.d(TAG, "startServer: Comenzado");
        AcceptThread mAcceptThread = new AcceptThread();
        mAcceptThread.start();
    }

    /**
     Finally the ConnectedThread which is responsible for maintaining
the BTConnection, Sending the data, and
     receiving incoming data through input/output streams
respectively.
     **/
    public class ConnectedThread extends Thread {
        public final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "ConnectedThread: Starting.");
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;
            //dismiss the progressdialog when connection is
established
            try{
                mProgressDialog.dismiss();
            }catch (NullPointerException e){
                e.printStackTrace();
            }
            try {
                tmpIn = mmSocket.getInputStream();
                tmpOut = mmSocket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run(){
            byte[] buffer = new byte[1024];  // buffer store for the
stream

            int bytes; // bytes returned from read()

            // Keep listening to the InputStream until an exception
occurs
            while (true) {
                // Read from the InputStream
```

```
                try {
                    bytes = mmInStream.read(buffer);
                    String incomingMessage = new String(buffer, 0,
bytes);
                    Log.d(TAG, "InputStream que se lee en el reloj: "
+ incomingMessage);
                    Thread.sleep(1000);
                } catch (IOException | InterruptedException e) {
                    Log.e(TAG, "write: Error reading Input Stream. " +
e.getMessage() );
                    break;
                }
            }
        }

        //Call this from the main activity to send data to the remote
device
        public void write(int bytes) {
            int text = bytes;
            Log.d(TAG, "write: Writing to outputstream: " + text);
            try {
                mmOutStream.write(bytes);
            } catch (IOException e) {
                Log.e(TAG, "write: Error writing to output stream. " +
e.getMessage() );
            }
        }

        /* Call this from the main activity to shutdown the connection
*/
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) { }
        }
    }

    private void connected(BluetoothSocket mmSocket, BluetoothDevice
mmDevice) {
        Log.d(TAG, "connected: Starting.");

        // Start the thread to manage the connection and perform
transmissions
        mConnectedThread = new ConnectedThread(mmSocket);
        mConnectedThread.start();
    }

    /**
     * Write to the ConnectedThread in an unsynchronized manner
     *
     * @param heartrate The bytes to write
     * //@see ConnectedThread#write(byte[])
     */
    public void write(int heartrate) {
        int i;
```

```
        // Create temporary object
            ConnectedThread r;
            r = mConnectedThread;
            // Synchronize a copy of the ConnectedThread
            Log.d(TAG, "write: Write Called.");
            //perform the write
            mConnectedThread.write(heartrate);

    }

}
```

CÓDIGO DE LA APLICACIÓN MÓVIL

NuevoUsuarioActivity

```
public class NuevoUsuarioActivity extends Activity {
    private static boolean inicializado = false;
    private Button btnSiguiente, btnguardar;
    public static EditText userID;
    private TextView usuario,titulousuario;
    LocalDateTime fecha;
    SQLiteHelper sqLiteHelper;
    Boolean bexiste;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_usuario);
        /**Inicializar todas las vistas del layout**/
        setContentView(R.layout.activity_usuario);
        btnSiguiente = findViewById(R.id.btnnext);
        btnguardar = findViewById(R.id.idbtnguardar);
        usuario = findViewById(R.id.idtxtidusuario);
        userID = findViewById(R.id.iedtxtidusuario);
        titulousuario = findViewById(R.id.idtitulousuario);
        sqLiteHelper = new SQLiteHelper(getApplicationContext(),
"variablesfisiologicas.db", null, 1);
        btnSiguiente.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                /**n Pasar al siguiente activity, la de la conexion
del Bluetooth **/
                startActivity(new
Intent(NuevoUsuarioActivity.this,MainActivity.class));
                /**Terminar el ciclo de vida del activity **/
                NuevoUsuarioActivity.this.finish();
            }
        });

        /**Proceso de guardado de la informacion del usuario en la
base de datos **/
        btnguardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                bexiste = existeUsuario(userID);
```

```
                if (bexiste == false) {
                    SQLiteHelper sqLiteHelper = new
SQLiteHelper(getApplicationContext(), SQLiteHelper.DB_NAME, null, 1);
                    SQLiteDatabase database =
sqLiteHelper.getWritableDatabase();
                    ContentValues values = new ContentValues();
                    values.put(SQLiteHelper.TABLA_FECHA, getDate());
                    values.put(SQLiteHelper.TABLA_ID,
userID.getText().toString());
                    //values.put(SQLiteHelper.TABLA_HORA,
System.currentTimeMillis());
                    Long idResultante =
database.insert(SQLiteHelper.NOMBRE_TABLA_USUARIO, null, values);
                    Toast.makeText(getApplicationContext(), "Registro
numero:" + idResultante, Toast.LENGTH_SHORT).show();
                    database.close();
                }else {
                    Toast.makeText(getApplicationContext(), "Usuario
Existente, registre usuario nuevo",Toast.LENGTH_SHORT).show();
                }


            }
        });


    }
    private String getDate() {            // se vería así: miercoles
26/09/2018 05:30 p.m.
        SimpleDateFormat dateFormat = new SimpleDateFormat("EEEE
dd/MM/yyyy hh:mm  a", Locale.getDefault());
        Date date = new Date();
        return dateFormat.format(date);
    }
    public boolean existeUsuario(EditText edtbuscarID){
        SQLiteDatabase database = sqLiteHelper.getReadableDatabase();
        String[] buscarID = {edtbuscarID.getText().toString()};
        String[]  campos = {SQLiteHelper.TABLA_ID};
        Cursor cursor =
database.query(SQLiteHelper.NOMBRE_TABLA_USUARIO,campos,SQLiteHelper.T
ABLA_ID+ "=?",buscarID,null,null,null);
        cursor.moveToFirst();
        boolean existe = cursor.moveToFirst();

        return existe;
    }


}// <NuevoUsuarioActivity
```

ConnectionService

```
public class ConnectionService   {
  private static final String TAG = "BluetoothConnectionServ";

   private static final String appName = "MYAPP";
   public static BluetoothSocket mysocket;

   private static final UUID MY_UUID_INSECURE =
         UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66");

   private BluetoothAdapter mBluetoothAdapter;
   Context mContext;

   private AcceptThread mInsecureAcceptThread;

   private ConnectThread mConnectThread;
   private BluetoothDevice mmDevice;
   private UUID deviceUUID;
   ProgressDialog mProgressDialog;

   private static ConnectedThread mConnectedThread;
   public static Integer sdato,idato;

   public static String incomingMessage;
   public static ArrayList<Integer> arrayList;

   public static Boolean bestado = false;
   public static final String EXTRA_HEARTRATE = "EXTRA_HEARTRATE";
   private static final String ACTION_DATA_CHANGE = "DATO ACTUALIZADO";

   public ConnectionService(Context context) {
      mContext = context;
      mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
      start();
   }
```

```
  /**
   * This thread runs while listening for incoming connections. It behaves
   * like a server-side client. It runs until a connection is accepted
   * (or until cancelled).
   */
  private class AcceptThread extends Thread {

    // The local server socket
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
      BluetoothServerSocket tmp = null;

      // Create a new listening server socket
      try {
        tmp = mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(appName,
MY_UUID_INSECURE);

        Log.d(TAG, "AcceptThread: Setting up Server using: " + MY_UUID_INSECURE);
      } catch (IOException e) {
        Log.e(TAG, "AcceptThread: IOException: " + e.getMessage());
      }

      mmServerSocket = tmp;
    }

    public void run() {
      Log.d(TAG, "run: AcceptThread Running.");

      BluetoothSocket socket = null;

      try {
        // This is a blocking call and will only return on a
        // successful connection or an exception
        Log.d(TAG, "run: RFCOM server socket start.....");

        socket = mmServerSocket.accept();

        Log.d(TAG, "run: RFCOM server socket accepted connection.");

      } catch (IOException e) {
```

```
            Log.e(TAG, "AcceptThread: IOException: " + e.getMessage());
        }

        //talk about this is in the 3rd
        if (socket != null) {
            connected(socket, mmDevice);
        }

        Log.i(TAG, "END mAcceptThread ");
    }

    public void cancel() {
        Log.d(TAG, "cancel: Canceling AcceptThread.");
        try {
            mmServerSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "cancel: Close of AcceptThread ServerSocket failed. " + e.getMessage());
        }
    }

}

/**
 * This thread runs while attempting to make an outgoing connection
 * with a device. It runs straight through; the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {
    private BluetoothSocket mmSocket;


    public ConnectThread(BluetoothDevice device, UUID uuid) {
        Log.d(TAG, "ConnectThread: started.");
        mmDevice = device;
        deviceUUID = uuid;
    }

    public void run() {
        BluetoothSocket tmp = null;
        Log.i(TAG, "RUN mConnectThread ");
```

```
        // Get a BluetoothSocket for a connection with the
        // given BluetoothDevice
        try {
            Log.d(TAG, "ConnectThread: Trying to create InsecureRfcommSocket using UUID: "
                    + MY_UUID_INSECURE);
            tmp = mmDevice.createRfcommSocketToServiceRecord(deviceUUID);
        } catch (IOException e) {
            Log.e(TAG, "ConnectThread: Could not create InsecureRfcommSocket " +
e.getMessage());
        }

        mmSocket = tmp;

        // Always cancel discovery because it will slow down a connection
        mBluetoothAdapter.cancelDiscovery();


        // Make a connection to the BluetoothSocket

        try {
            // This is a blocking call and will only return on a
            // successful connection or an exception
            mmSocket.connect();
            Log.d(TAG, "run: ConnectThread connected.");
        } catch (IOException e) {
            // Close the socket
            try {
                mmSocket.close();
                Log.d(TAG, "run: Closed Socket.");
            } catch (IOException e1) {
                Log.e(TAG, "mConnectThread: run: Unable to close connection in socket " +
e1.getMessage());
            }
            Log.d(TAG, "run: ConnectThread: Could not connect to UUID: " +
MY_UUID_INSECURE);
        }

        //will talk about this in the 3rd video
        connected(mmSocket, mmDevice);
    }
```

```
    public void cancel() {
        try {
            Log.d(TAG, "cancel: Closing Client Socket.");
            mmSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "cancel: close() of mmSocket in Connectthread failed. " + e.getMessage());
        }
    }
}


/**
 * Start the chat service. Specifically start AcceptThread to begin a
 * session in listening (server) mode. Called by the Activity onResume()
 */
public synchronized void start() {
    Log.d(TAG, "start");

    // Cancel any thread attempting to make a connection
    if (mConnectThread != null) {
        mConnectThread.cancel();
        mConnectThread = null;
    }
    if (mInsecureAcceptThread == null) {
        mInsecureAcceptThread = new AcceptThread();
        mInsecureAcceptThread.start();
    }
}

/**
 * AcceptThread starts and sits waiting for a connection.
 * Then ConnectThread starts and attempts to make a connection with the other devices
AcceptThread.
 **/

public void startClient(BluetoothDevice device, UUID uuid) {
    Log.d(TAG, "startClient: Started.");

    //initprogress dialog
    mProgressDialog = ProgressDialog.show(mContext, "Connecting Bluetooth"
            , "Please Wait...", true);
```

```
        mConnectThread = new ConnectThread(device, uuid);
        mConnectThread.start();
    }

    /**
     * Finally the ConnectedThread which is responsible for maintaining the BTConnection,
Sending the data, and
     * receiving incoming data through input/output streams respectively.
     **/
    private class ConnectedThread extends Thread {
        private final  BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket)  {
            Log.d(TAG, "ConnectedThread: Starting.");

            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            //dismiss the progressdialog when connection is established
            try {
                mProgressDialog.dismiss();
            } catch (NullPointerException e) {
                e.printStackTrace();
            }


            try {
                tmpIn = mmSocket.getInputStream();
                tmpOut = mmSocket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
```

```
    }

    public void run() {

        byte[] buffer = new byte[1024];  // buffer store for the stream
        int bytes; // bytes returned from read()
        // Keep listening to the InputStream until an exception occurs
        while (true) {
            // Read from the InputStream
            try {
                String action;
                action = ACTION_DATA_CHANGE;
                bytes = mmInStream.read(buffer);
                incomingMessage = new String(buffer, 0, bytes);
                Log.d(TAG, "InputStream: " + (int) incomingMessage.charAt(0));
                UsoDeDatos.frecuencia = (int) incomingMessage.charAt(0);
                BroadcastUpdate(ACTION_DATA_CHANGE);


                MainActivity.datofinal.setText("Frecuencia cardiaca:" + (int)
ConnectionService.incomingMessage.charAt(0));

                /** Guardar dato de frecuencia en un array **/
                UsoDeDatos.datosfrecuencias.add((int) incomingMessage.charAt(0));
                Log.d(TAG, "Dato guardado en Array");

            } catch (IOException e) {
                Log.e(TAG, "write: Error writing  InputStream. " + e.getMessage());
                break;
            }
        }
    }

    //Call this from the main activity to send data to the remote device
    public void write(byte[] bytes) {
        String text = new String(bytes, Charset.defaultCharset());
        Log.d(TAG, "write: Writing to outputstream: " + text);
        try {
            mmOutStream.write(bytes);
        } catch (IOException e) {
            Log.e(TAG, "write: Error writing to output stream. " + e.getMessage());
```

```
        }
    }

    /* Call this from the main activity to shutdown the connection */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) {
        }
    }
}

private void connected(BluetoothSocket mmSocket, BluetoothDevice mmDevice) {
    Log.d(TAG, "connected: Starting.");

    // Start the thread to manage the connection and perform transmissions
    mConnectedThread = new ConnectedThread(mmSocket);
    mConnectedThread.start();

}
public static void  cancelConexion (){
    mysocket = mConnectedThread.mmSocket;
    try {
        mysocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
/**
 * Write to the ConnectedThread in an unsynchronized manner
 *
 * @param *out The bytes to write
 //* @see ConnectedThrea#dwrite(byte[])
 */

private void BroadcastUpdate (final String action){
    final Intent intent = new Intent(action);
    mContext.sendBroadcast(intent);
}

private void broadcastUpdateWithProcessing (final String action){
```

```
        final Intent intent = new Intent(action);

        intent.putExtra(EXTRA_HEARTRATE, (int) incomingMessage.charAt(0));
        mContext.sendBroadcast(intent);
    }

    private void broadcastUpdateHR (final String action2){
        final Intent intent2 = new Intent(action2);
        intent2.putExtra(SQLiteHelper.COLUMNA_FRECUENCIA, (int)
incomingMessage.charAt(0));
        Log.d(TAG, "Dato guardado en BASE DE DATOS");
        mContext.sendBroadcast(intent2);
    }
}//<ConnectionService>
```

MostrarSeleccionActivity

```
public class MostrarSeleccionActivity extends Activity {
    LineChart lineChart;
    Button buscar,atras,añadir;
    TextView horayfecha,txthora;
    EditText edtnombre,edtensayo,edtidusuario,edtbuscarID;
    SQLiteHelper conn;
    Spinner idspinner1;
    public static ArrayList idlist;
    public static ArrayList<Integer> listfrecuencias;
    public static ArrayList<Integer> listabuscado = new ArrayList<>();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mostrarseleccion);
        buscar = findViewById(R.id.idbtnidusuario);
        atras = findViewById(R.id.idatras2);
        lineChart = findViewById(R.id.idgrafico2);
        horayfecha = findViewById(R.id.idhoraencontrada);
        txthora = findViewById(R.id.idtxthora);
        edtbuscarID = findViewById(R.id.idebuscarID);


        //Para conectar con la base de datos anteriormente creada
        conn = new SQLiteHelper(getApplicationContext(),
"variablesfisiologicas.db", null, 1);

        buscar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                boolean existe2 = existeID(edtbuscarID);
```

```
                if(existe2 == true) {
                    //Imprimirá la fecha en la que se realizó la
prueba del identificador
                    consultarID(edtbuscarID);
                    listabuscado = consultarFrecuencias(edtbuscarID);
                    crearGrafico(lineChart,listabuscado,"");
                }else if (existe2==false){
                    Toast.makeText(getApplicationContext(),"ID no
encontrado", Toast.LENGTH_SHORT).show();
                }

            }
        });

        atras.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new
Intent(MostrarSeleccionActivity.this, MainActivity.class));
            }
        });
    }

    public void crearGrafico (LineChart graphicChart,
ArrayList<Integer> valores, String axisName) {

        graphicChart.setVisibility(View.INVISIBLE);
        graphicChart.setVisibility(View.VISIBLE);
        graphicChart.setDragEnabled(true);// Permite desplazarse a lo
largo del eje x
        graphicChart.setScaleEnabled(true);  //Permite aumentar la
imagen

        ArrayList<Entry> yValues = new ArrayList<>();
        for (int i = 0; i < valores.size(); i++) {
            yValues.add(new Entry(i, valores.get(i)));
        }

        // Line configuration
        LineDataSet lineDatasetGraphic = new LineDataSet(yValues,
"GRÁFICO" + " " + axisName);
        lineDatasetGraphic.setFillAlpha(110);
        lineDatasetGraphic.setColor(Color.MAGENTA);
        lineDatasetGraphic.setLineWidth(3f);

lineDatasetGraphic.setValueTextColor(android.R.color.transparent);
        lineDatasetGraphic.setCircleColor(Color.MAGENTA);


        // Axis configuration
        //  - x axis
        XAxis xAxis = graphicChart.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
        xAxis.setTextSize(10f);
        xAxis.setTextColor(Color.BLACK);
```

```
        xAxis.setDrawAxisLine(true);

        //  - y axis
        YAxis yAxis = graphicChart.getAxisLeft();
        yAxis.setTextSize(10f);
        yAxis.setValueFormatter(new DefaultValueFormatter(1));
        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??)
        graphicChart.getAxisRight().setEnabled(false);

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

        // Legend configuration
        Legend myLegend = graphicChart.getLegend();
        myLegend.setEnabled(false);


        ArrayList<ILineDataSet> dataSets = new ArrayList<>();
        dataSets.add(lineDatasetGraphic);
        LineData data = new LineData(dataSets);
        graphicChart.setData(data);

    }

    private void consultarID (EditText edtbuscarID){
        SQLiteDatabase database = conn.getReadableDatabase();
        String[] buscarID = {edtbuscarID.getText().toString()};
        String[]  campos = {SQLiteHelper.TABLA_FECHA};
        Cursor cursor =
database.query(SQLiteHelper.NOMBRE_TABLA_USUARIO,campos,SQLiteHelper.T
ABLA_ID+ "=?",buscarID,null,null,null);
        cursor.moveToFirst();
        horayfecha.setText(cursor.getString(0));
        cursor.moveToFirst();

            cursor.close();
            database.close();

    }
    /** Imprime un booleano en funcion de si existe el dato que se
quiere buscar o no
     * Existe --> true
     * No Existe --> false **/
```

```
    public boolean existeID(EditText edtbuscarID){
            SQLiteDatabase database = conn.getReadableDatabase();
            String[] buscarID = {edtbuscarID.getText().toString()};
            String[]  campos = {SQLiteHelper.TABLA_FECHA};
            Cursor cursor =
database.query(SQLiteHelper.NOMBRE_TABLA_USUARIO,campos,SQLiteHelper.T
ABLA_ID+ "=?",buscarID,null,null,null);
            cursor.moveToFirst();
            //horayfecha.setText(cursor.getString(0));
            boolean existe = cursor.moveToFirst();
                //cursor.close();
               // database.close();

        return existe;
        }

    /** Busca las frecuencias que coinciden con el identificador
buscado y las guarda en un array
     * para lugo graficarlas **/
    public ArrayList<Integer>  consultarFrecuencias (EditText
edtbuscarID){
        SQLiteDatabase database = conn.getReadableDatabase();
        String[] buscarID = {edtbuscarID.getText().toString()};
        String [] campos = {SQLiteHelper.COLUMNA_FRECUENCIA};
        Cursor cursor =
database.query(SQLiteHelper.TABLA_FRECUENCIA,campos,SQLiteHelper.COLUM
NA_IDFRECUENCIAS+ "=?",buscarID,null,null,null);
        cursor.moveToFirst();
        int columnidex =
cursor.getColumnIndex(SQLiteHelper.COLUMNA_FRECUENCIA);

        listfrecuencias = new ArrayList<Integer>();
        while( !cursor.isAfterLast()){
            listfrecuencias.add(cursor.getInt(columnidex));
            cursor.moveToNext();
        }
        cursor.close();
        database.close();
        return listfrecuencias;
    }

} //<MostrarSeleccionActivity>
```

DeviceListAdapter

```
public class DeviceListAdapter extends ArrayAdapter<BluetoothDevice> {

    private LayoutInflater mLayoutInflater;
    private ArrayList<BluetoothDevice> mDevices;
    private int mViewResourceId;

    public DeviceListAdapter(Context context, int tvResourceId,
ArrayList<BluetoothDevice> devices) {
        super(context, tvResourceId, devices);
```

```
        this.mDevices = devices;
        mLayoutInflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        mViewResourceId = tvResourceId;
    }

    public View getView(int position, View convertView, ViewGroup
parent) {
        convertView = mLayoutInflater.inflate(mViewResourceId, null);

        BluetoothDevice device = mDevices.get(position);

        if (device != null) {
            TextView deviceName = (TextView)
convertView.findViewById(R.id.idDeviceName);
            TextView deviceAdress = (TextView)
convertView.findViewById(R.id.idDeviceAddress);
            deviceName.setText(device.getName());
            deviceAdress.setText(device.getAddress());
            /*if (deviceName != null) {
                deviceName.setText(device.getName());
            }
            if (deviceAdress != null) {
                deviceAdress.setText(device.getAddress());
            }*/
        }

        return convertView;
    }
} //<DeviceListAdapter>
```

MostratHeartRateActivity

```
public class MostrarHeartRateActivity extends Activity {

    public TextView datofinalHR;
    ConnectionService mBluetoothConnection;
    Button btnatras, btnmostrar, btnexportar, btnEmail, btnsiguiente;
    ImageButton guardar;
    LineChart lineChart;
    SQLiteHelper sqLiteHelper;
    public static File csvFilePath;
    File csvFile;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_heartrate);
        datofinalHR = findViewById(R.id.iddatofinal);
        btnatras = findViewById(R.id.idatras);
        btnmostrar = findViewById(R.id.idimprimir);
        lineChart = findViewById(R.id.idgrafico2);
        btnsiguiente = findViewById(R.id.btsiguiente);
        guardar = findViewById(R.id.idguardar1);
        btnexportar = findViewById(R.id.idexportarcsv);
        btnEmail = findViewById(R.id.idenvio);

        sqLiteHelper = new SQLiteHelper(getApplicationContext(),
SQLiteHelper.DB_NAME, null, 1);
        //Botón para volver a la pantalla principal
        btnatras.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new
Intent(MostrarHeartRateActivity.this, MainActivity.class));
                //Para terminar la vida del activity
                // MostrarHeartRateActivity.this.finish();
            }
        });
        //Botón para printear el gráfico
        btnmostrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                crearGrafico(lineChart, UsoDeDatos.datosfrecuencias,
"");
            }

        });
        //Guardar los datos en la base de datos
        guardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                guardarDatos(UsoDeDatos.datosfrecuencias,
getApplicationContext());
                Toast.makeText(getApplicationContext(), "Datos
guardados", Toast.LENGTH_SHORT).show();
            }
        });
        //Botón para pasar a la siguiente pantalla
        btnsiguiente.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new
Intent(MostrarHeartRateActivity.this,
MostrarSeleccionActivity.class));
            }
        });
        //Exportar los datos a formato csv
        btnexportar.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View view) {
                isStoragePermissionGranted();
                pedirPermisos();
                exportarACSV();
            }
        });
        //Enviar los datos por email
        btnEmail.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                pedirPermisos();
                SendDataByEmailButton();
            }
        });


    }

    /********************************** CREAR GRAFICO
*****************************************/

    public void crearGrafico(LineChart graphicChart,
ArrayList<Integer> valores, String axisName) {

        graphicChart.setVisibility(View.INVISIBLE);
        graphicChart.setVisibility(View.VISIBLE);
        graphicChart.setDragEnabled(true);   // Permite desplazarse a
lo largo del eje x
        graphicChart.setScaleEnabled(true);  //Permite aumentar la
imagen

        ArrayList<Entry> yValues = new ArrayList<>();
        for (int i = 0; i < valores.size(); i++) {
            yValues.add(new Entry(i, valores.get(i)));
        }

        // Line configuration
        LineDataSet lineDatasetGraphic = new LineDataSet(yValues,
"GRÁFICO" + " " + axisName);
        lineDatasetGraphic.setFillAlpha(110);
        lineDatasetGraphic.setColor(Color.RED);
        lineDatasetGraphic.setLineWidth(3f);

lineDatasetGraphic.setValueTextColor(android.R.color.transparent);
        lineDatasetGraphic.setCircleColor(Color.RED);


        // Axis configuration
        //  - x axis
        XAxis xAxis = graphicChart.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
        xAxis.setTextSize(10f);
        xAxis.setTextColor(Color.BLACK);
        xAxis.setDrawAxisLine(true);
```

```
        //  - y axis
        YAxis yAxis = graphicChart.getAxisLeft();
        yAxis.setTextSize(10f);
        yAxis.setValueFormatter(new DefaultValueFormatter(1));
        yAxis.setTextColor(Color.BLACK);

        graphicChart.getAxisRight().setEnabled(false);

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

        // Legend configuration
        Legend myLegend = graphicChart.getLegend();
        myLegend.setEnabled(false);


        ArrayList<ILineDataSet> dataSets = new ArrayList<>();
        dataSets.add(lineDatasetGraphic);
        LineData data = new LineData(dataSets);
        graphicChart.setData(data);


    }

    /*************************** GUARDAR FRECUENCIAS EN BASE DE DATOS
*****************************/

    public void guardarDatos(ArrayList<Integer> datos, Context
context) {

        SQLiteDatabase database = sqLiteHelper.getWritableDatabase();
        ContentValues values = new ContentValues();

        int arraylong = datos.size();

        for (int i = 0; i < arraylong; i++) {
            values.put(SQLiteHelper.COLUMNA_FRECUENCIA, datos.get(i));
            values.put(SQLiteHelper.COLUMNA_IDFRECUENCIAS,
NuevoUsuarioActivity.userID.getText().toString());
            long insert =
database.insert(SQLiteHelper.TABLA_FRECUENCIA, null, values);
        }

        database.close();

    }
    //Hay que asegurarse de que se han dado los permisos
```

```
correspondientes para exportar los datos
    //El usuario tiene que aceptar manualmente los permisos

    /********************************** PEDIR PERMISOS
*********************************/
    public void pedirPermisos() {
        if (ContextCompat.checkSelfPermission(getApplicationContext(),
                Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
PackageManager.PERMISSION_GRANTED) {

ActivityCompat.requestPermissions(MostrarHeartRateActivity.this,
                    new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.WRITE_EXTERNAL_STORAGE}, 0);
        }
    }

    /******************************** OBTENER DATOS
*************************************/
    public void obtenerDatos(){
        UsoDeDatos.datosfrecuencias.clear();
        SQLiteDatabase database = sqLiteHelper.getWritableDatabase();
        //Cursor cursor = database.rawQuery("select * from
TABLA_FRECUENCIAS",null);
        Cursor cursor = database.rawQuery("select * from
UsoDeDatos.datosfrecuencias",null);
        if (cursor != null & cursor.getCount() != 0){
            cursor.moveToFirst();
            int columnidex1 =
cursor.getColumnIndex(SQLiteHelper.TABLA_ID);
            int columnidex2 =
cursor.getColumnIndex(SQLiteHelper.COLUMNA_FRECUENCIA);
            int columnidex3 =
cursor.getColumnIndex(SQLiteHelper.COLUMNA_ENUMERADOR);
            int columnidex4 =
cursor.getColumnIndex(SQLiteHelper.TABLA_FECHA);
            while (!cursor.isAfterLast()) {
                UsoDeDatos.idfrecuencias.add(columnidex3);
                UsoDeDatos.frecuencias.add(columnidex2);
                UsoDeDatos.nombres.add(String.valueOf(columnidex1));
                UsoDeDatos.fechas.add(String.valueOf(columnidex4));
                cursor.moveToNext();
            }

        } else {
            Toast.makeText(this, "No hay
registros",Toast.LENGTH_SHORT).show();


             }

        } //<ObtenerDatos>

    /************************ PEDIR PERMISOS MANUALMENTE AL USUARIO
************************/
```

```
    public boolean isStoragePermissionGranted() {
        String TAG = "Storage Permission";
        if (Build.VERSION.SDK_INT >= 23) {
            if
(this.checkSelfPermission(android.Manifest.permission.WRITE_EXTERNAL_S
TORAGE)
                    == PackageManager.PERMISSION_GRANTED) {
                Log.v(TAG, "Permission is granted");
                return true;
            } else {
                Log.v(TAG, "Permission is revoked");
                ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);
                return false;
            }
        }
        else { //permission is automatically granted on sdk<23 upon
installation
            Log.v(TAG,"Permission is granted");
            return true;
        }
    }



    /*************** EXPORTAR BASE DE DATOS A CSV
**************************/

    public void exportarACSV() {
        final  File externalStorage = getExternalFilesDir(null);
        final String externalStoragePath =
externalStorage.getAbsolutePath();
        csvFilePath = new File( externalStoragePath+ "/" +
"ExportarSQLiteCSV", "");
        csvFile = new File(csvFilePath, "Frecuencias.csv");
        csvFile.mkdir();
        final String externalStorageState =
Environment.getExternalStorageState();
        final String csvFilesPathName = externalStoragePath + "/" +
"ExportarSQLiteCSV";

        // 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;
        }

       boolean folderCreated = false;
        // Create folder
```

```
        if (!csvFilePath.exists()) {
            folderCreated = csvFilePath.mkdir();
            Toast.makeText(this, "Carpeta creada",
Toast.LENGTH_SHORT).show();
        }
        // Create file
        boolean fileCreated = false;
       if(!csvFile.exists()) {
        try {
                fileCreated = csvFile.createNewFile();
                Toast.makeText(getApplicationContext(), "Archivo
creado: " + fileCreated, Toast.LENGTH_SHORT).show();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
        // Write to file
        FileWriter filewriter = null;
        try {
            filewriter = new FileWriter(csvFile);
        SQLiteHelper admin = new SQLiteHelper(getApplicationContext(),
SQLiteHelper.DB_NAME, null, 1);
            SQLiteDatabase database = admin.getReadableDatabase();
            Toast.makeText(this, "Base de datos abierta",
Toast.LENGTH_SHORT).show();
            Cursor cursor = database.rawQuery("SELECT * FROM " +
sqLiteHelper.TABLA_FRECUENCIA, null);
           if (cursor != null & cursor.getCount() != 0) {
               cursor.moveToFirst();
               do {
                   filewriter.write(cursor.getString(0));
                   filewriter.write(";");
                   filewriter.write(cursor.getString(1));
                   filewriter.write(";");
                   filewriter.write( cursor.getString(2));
                   filewriter.write("\n");
               } while (cursor.moveToNext());
           }else {
               Toast.makeText(this, "NO HAY REGISTROS",
Toast.LENGTH_SHORT).show();
            }
            Toast.makeText(this, "Archivo CSV creado",
Toast.LENGTH_SHORT).show();
            database.close();
            filewriter.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /********************************** ENVIAR POR EMAIL
*****************************************/
    private void SendDataByEmailButton() {
```

```
        btnEmail.setOnClickListener(v -> {
            // Check for external storage writing permission (needed
to save .xls file)
            if
(ContextCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.WRITE_EXTERNAL_STORAGE)
                    == PackageManager.PERMISSION_GRANTED) {

                String email = ""; //Email al que quieres mandar el
archivo
                // Send email
                Uri myUri = Uri.fromFile(csvFile);
                Intent emailIntent = new Intent(Intent.ACTION_SENDTO);
                emailIntent.setData(Uri.parse("mailto:"));

                //emailIntent.setType("*/*");
                emailIntent.putExtra(Intent.EXTRA_EMAIL, new
String[]{email});
                emailIntent.putExtra(Intent.EXTRA_SUBJECT, "File " +
"Archivo Frecuencias");
                emailIntent.putExtra(Intent.EXTRA_STREAM, myUri);
//The attachment document
                emailIntent.putExtra(Intent.EXTRA_TEXT, "Hola, Aquí te
adjunto el archivo en formato csv. ");

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

                startActivity(Intent.createChooser(emailIntent,
"Enviar email..."));
                Toast.makeText(getApplicationContext(), "Email sent",
Toast.LENGTH_SHORT).show();

            } else {
                ActivityCompat.requestPermissions((Activity)
getApplicationContext(),
                        new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                        1);
            }
        });  // </ buttonEmail.setOnClickListener() >
    }
}
```

SQLiteHelper

```
public class SQLiteHelper extends SQLiteOpenHelper {
    //Variables del archivo de datos
    private File excelFile;
    private String TAG = "Debugear";
    Context context;
    private static final String ACTION_DATA_CHANGE = "DATO
ACTUALIZADO";
```

```
    //Variables de la base de datos(DATABASE = DB)
    public static final int DB_VERSION = 1;
    public static final String DB_NAME = "variablesfisiologicas.db";

    // Constantes tabla usuario
    public static final String NOMBRE_TABLA_USUARIO =
"TABLA_DE_USUARIO";
    public static final String TABLA_FECHA = "Fecha_y_Hora";  //
userName
    public static final String TABLA_HORA = "Hora";    // Numero de
ensayo
    public static final String TABLA_ID = "Identificador";  //
IDentificador del usuario

    //TABLA DE LA FRECUENCIA CARDIACA
    public static final String TABLA_FRECUENCIA = "TABLA_FRECUENCIAS";
    public static final String COLUMNA_ENUMERADOR =
"Numero_frecuencia";
    public static final String COLUMNA_FRECUENCIA = "Frecuencia";
    public static final String COLUMNA_IDFRECUENCIAS =
"Identificadorusuario";  // IDentificador del usuario para la tabla de
frecuencias
    public static final String COLUMNA_TIMESTAMP = "Fecha";


    public SQLiteHelper( Context context, String name,
SQLiteDatabase.CursorFactory factory, int version) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    /************************ On Create
*********************************/
    @Override
    public void onCreate(SQLiteDatabase database) {

        /** CREAR TABLA DE USUARIO **/
        /*---------------------- FORMATO TABLA ---------------------*/
        //- DATOS DEL USUARIO
        // | ID DEL USUARIO | HORA Y FECHA |

        database.execSQL(" CREATE TABLE " + NOMBRE_TABLA_USUARIO + " (
"

                + TABLA_ID +" INTEGER , "
                + TABLA_FECHA + " DATE DEFAULT CURRENT_DATE )");

        /*---------------------- FORMATO TABLA ---------------------*/
        //- DATOS DE LOS PARAMETROS
// | Numero de Frecuencia | ID del Usuario | Fecha | Frecuencia
Cardiaca |


        /**CREAR TABLA DE FRECUENCIAS CARDIACAS **/
```

```
        database.execSQL(" CREATE TABLE " + TABLA_FRECUENCIA + " ( "
                + COLUMNA_ENUMERADOR + " INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, "
                + COLUMNA_IDFRECUENCIAS + " INTEGER , "
                 + COLUMNA_TIMESTAMP  + "  DATE DEFAULT CURRENT_DATE ,
"

                + COLUMNA_FRECUENCIA + " INTEGER )");

    }// </ Oncreate >

    /******************************** OnUpGrade
*************************/
    //Si el valor de la base de datos cambia se llamará al metodo
OnUpgrade
    public void onUpgrade(SQLiteDatabase database, int oldVersion, int
newVersion) {
        //Elimina la version de la tabla anterior para crear una nueva
        database.execSQL("DROP TABLE IF EXISTS " +
NOMBRE_TABLA_USUARIO);
        database.execSQL("DROP TABLE IF EXISTS " + TABLA_FRECUENCIA);
        onCreate(database);


    } //<OnUpGrade>

    /******************************** SaveData
*************************/
    /** Guardadr los datos de la frecuencia cardiaca **/
    public void saveHRData (int iHR){
        SQLiteDatabase database = getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(SQLiteHelper.COLUMNA_FRECUENCIA,iHR);
        database.insert(SQLiteHelper.TABLA_FRECUENCIA,null, values);
        database.close();


    }

    /** ESTO FUNCIONA COMO ELIMINACION DE UN VALOR DE LA BASE DE DATOS
**/
    public void deleteData (int duplicateData){
        SQLiteDatabase database = getWritableDatabase();
        String[] selecteddata = {String.valueOf(duplicateData)};

database.delete(TABLA_FRECUENCIA,COLUMNA_FRECUENCIA,selecteddata);
        database.close();

    }

    /** Guardar en la base de datos informacion del usuario
     * @param usuario
     * @param ensayo
     * @param Identificador **/
```

```
    public void saveNewUser (EditText usuario, EditText ensayo,
EditText Identificador){
        SQLiteDatabase database = getWritableDatabase();

        if(database!= null){
            database.execSQL("INSERT INTO NOMBRE_TABLA_USUARIO
TABLE('"+usuario+"','"+ensayo+"','"+ Identificador +"' )");
            database.close();

        }
    }
    /**Actualizar la tabla del usuario **/
    public void updateUserRow (String usuario, int ensayo, String
identificador){
        SQLiteDatabase database = getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(SQLiteHelper.TABLA_FECHA, usuario);
        values.put(SQLiteHelper.TABLA_HORA, ensayo);
        values.put(SQLiteHelper.TABLA_ID, identificador);

database.insert(SQLiteHelper.NOMBRE_TABLA_USUARIO,null,values);
        database.close();
    }

    private final BroadcastReceiver mHeartRateBroadcastReceiver = new
BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            // When arrives a new data
            if (action.equals(ACTION_DATA_CHANGE)) {
                SQLiteDatabase database = getWritableDatabase();
                ContentValues values = new ContentValues();
                values.put(SQLiteHelper.COLUMNA_FRECUENCIA, (int)
ConnectionService.incomingMessage.charAt(0));
                database.insert(SQLiteHelper.TABLA_FRECUENCIA,null,
values);
                database.close();
            }
        }
    };
} //< SQLitehelper >
```

UsoDeDatos

```
public class UsoDeDatos {

    public static ArrayList<Integer> datosfrecuencias = new
ArrayList<>();
    public static Integer frecuencia = null;
    public static ArrayList<Integer> idfrecuencias = new
ArrayList<>();
    public static ArrayList<Integer> frecuencias = new ArrayList<>();
```

```
    public static ArrayList<String> nombres = new ArrayList<>();
    public static ArrayList<String> fechas =new ArrayList<>();

}
```